Київський національний торговельно-економічний університет
Кафедра інженерії програмного забезпечення та кібербезпеки

ВИПУСКНИЙ КВАЛІФІКАЦІЙНИЙ ПРОЕКТ

на тему:

«Розробка програмного забезпечення управління відділом технічного обслуговування КНТЕУ»

Студента 2м курсу, 6 групи, спеціальності 121 «Інженерія програмного забезпечення» спеціалізації «Інженерія програмного забезпечення» _____ підпис студента — Хруща Олексія Геннадійовича

Науковий керівник кандидат економічних наук, доцент кафедри інженерії програмного забезпечення та кібербезпеки _____ підпис керівника — Палагута Катерина Олексіївна

Гарант освітньої програми доктор технічних наук, професор кафедри інженерії програмного забезпечення та кібербезпеки _____ підпис гаранта — Криворучко Олена Володимирівна

КИЇВ - 2020

Київський національний торговельно-економічний університет

Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення та кібербезпеки
Освітній ступінь магістр
Спеціальність 121 «Інженерія програмного забезпечення»

Затверджую
Зав. кафедри інженерії програмного забезпечення та кібербезпеки
Криворучко О. В.
8 листопада 2019 р.

Завдання
на випускний кваліфікаційний проект студента

Хруща Олексія Геннадійовича
(прізвище, ім'я, по батькові)

1. Тема випускного кваліфікаційного проекту «Розробка програмного забезпечення управління відділом технічного обслуговування КНТЕУ»
Затверджена наказом ректора від «13» грудня 2019 р. № 4304

2. Строк здачі студентом закінченої проекту 01 грудня 2020 р.

3. Цільова установка та вихідні дані до проекту
Мета проекту розробка та впровадження інформаційної системи для управління відділом технічного обслуговування КНТЕУ
Об'єкт дослідження організація послуг для технічного обслуговування
Предмет дослідження методи та алгоритми створення веб-сайту для технічного обслуговування КНТЕУ

4. Консультанти проекту із зазначенням розділів, які консультують:

Розділ	Консультант (прізвище, ініціали)	Підпис, дата	
		Завдання видав	Завдання прийняв

5. Зміст випускного кваліфікаційного проекту (перелік питань за кожним розділом)

ВСТУП

РОЗДІЛ 1 ЗАГАЛЬНІ ВІДОМОСТІ ТА ОПИС ТЕХНОЛОГІЙ

1.1. Опис проекту

1.2. Використання REST API

1.3. Docker-контейнеризація веб-сайту

1.4. Висновки до розділу 1

РОЗДІЛ 2 ВИБІР ТЕХНОЛОГІЙ, ПРОЕКТУВАННЯ ПРОЕКТУ

2.1. Вибір технологій

2.2. Проектування бази даних

2.3. Проектування взаємодії між різними частинами системи

2.4. Проектування системи email-повідомлень

2.5. Вибір для розташування і збереження проекту

2.6. Висновки до розділу 2

РОЗДІЛ 3 РОЗРОБКА ПРОЕКТУ, DOCKER-КОТЕЙНЕРИЗАЦІЯ ТА НАЛАШТУВАННЯ CI/CD

3.1. Налаштування Docker-контейнерів

3.2. Розробка з використанням Traefik Proxy

3.3. Розробка клієнтської частини

3.4. Розробка серверної частини

3.5. Налаштування CI/CD

3.6. Висновки до розділу 3

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

ТЕХНІЧНЕ ЗАВДАННЯ

ТЕСТУВАННЯ ВЕБ-САЙТА

ДОДАТКИ

6. Календарний план виконання проекту

№ пор.	Назва етапів випускного кваліфікаційного проекту	Строк виконання етапів проекту	
		за планом	фактично
1	2	3	4
1.	*Вибір теми випускного кваліфікаційного проект*	*20.09.2019*	*20.09.2019*
2.	*Розробка та затвердження завдання на проект магістра*	*08.11.2019*	*08.11.2019*
3.	*Вступ та перелік літературних джерел*	*24.01.2020*	*24.01.2020*
4.	*Наукова стаття*	*01.09.2020*	*01.09.2020*
5.	*Технічне завдання*	*28.02.2020*	*28.02.2020*
6.	*Розділ 1. Загальні відомості та опис технологій*	*25.06.2020*	*25.06.2020*
7.	*Розділ 2. Вибір технологій, проектування проекту*	*07.09.2020*	*07.09.2020*
8.	*Розділ 3. Розробка проекту, Docker контейнеризація та налаштування CI/CD*	*19.10.2020*	*19.10.2020*
9.	*Програма та методика тестування*	*21.10.2020*	*21.10.2020*
10.	*Керівництво користувача*	*23.10.2020*	*23.10.2020*
11.	*Висновки та пропозиції*	*30.10.2020*	*30.10.2020*
12.	*Здача випускного кваліфікаційного проекту на кафедру (перша перевірка)*	*05.11.2020*	*05.11.2020*
13.	*Підготовка автореферату та презентації доповіді*	*05.11.2020*	*05.11.2020*
14.	*Попередній захист випускного кваліфікаційного проекту*	*25.11.2020-27.11.2020*	*25.11.2020-27.11.2020*
15.	*Зовнішнє рецензування випускного кваліфікаційного проекту*	*01.12.2020*	*01.12.2020*
16.	*Підготовка до публічного захисту випускного кваліфікаційного проекту*	*08.12.2020-09.12.2020*	

7. Дата видачі завдання «8» листопада 2019 р.

8. Науковий керівник випускного кваліфікаційного проекту Палагута К. О.
(прізвище, ініціали, підпис)

9. Гарант освітньої програми Криворучко О. В.
(прізвище, ініціали, підпис)

10. Завдання прийняв до виконання студент Хрущ О. Г.
(прізвище, ініціали, підпис)

11. Відгук керівника випускного кваліфікаційного проекту

Науковий керівник випускного кваліфікаційного проекту

(підпис, дата)

Відмітка про попередній захист Палагута К. О.

(ПІБ, підпис, дата)

12. Висновок про випускний кваліфікаційний проект

Випускний кваліфікаційний проект студента Хруща О. Г.

(прізвище, ініціали)

може бути допущена до захисту екзаменаційній комісії.

Гарант освітньої програми Криворучко О. В.

(прізвище, ініціали, підпис)

Завідувач кафедри Криворучко О. В.

(підпис, прізвище, ініціали)

«_____»_____20 ___ р.

АНОТАЦІЯ

Відповідно до мети дослідження, робота присвячена розробці веб-сайту для управління відділом технічного обслуговування КНТЕУ, що дозволить працівникам створювати замовлення через інтернет та відстежувати її онлайн.

Розробка була поділена на 3 основні частини:

- Серверна частина, де використовувалась мова програмування PHP з Laravel фреймворком.
- Клієнтська частина, де використовувалась мова програмування Javascript з Vue.js фреймворком.
- Система для оновлень в реальному часі, де використовувалась мова програмування Typescript з Node.js фреймворком.

Для взаємодії цих частин, використовувались додаткові сервіси та інструменти. Дані зберігаються в реляційній базі даних MySQL.

Готовий веб-сайт було успішно протестовано відповідно до функціональних вимог.

Ключові слова: Веб-сайт, Laravel, SPA, websocket, інформаційна система.

ABSTRACT

In accordance with the purpose of the study, the work is devoted to the development of a website for the management of the maintenance department of KNTEU, which will allow employees to create orders via the Internet and track it online.

- The development was divided into 3 main parts:
- The server part where the PHP programming language with the Laravel framework was used.

- The client part where the Javascript programming language with the Vue.js framework was used.
- Real-time websocket system using the Typescript programming language with the Node.js framework.

For the interaction of these parts, additional services and tools were used. The data is stored in a relational MySQL database.

The finished website has been successfully tested in accordance with the functional requirements.

Keywords: Website, Laravel, SPA, websocket, information system.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ПК – персональний комп’ютер.

БД – база даних.

JSON (англ. JavaScript Object Notation) –текстовий формат обміну даними між комп'ютерами.

MVC (англ. Model–View–Controller) – шаблон дизайну програмного забезпечення.

SPA (англ. Single-Page Application) – веб-сайт, якому не потрібно перезавантажувати сторінку під час використання та працює в браузері.

CI/CD (англ. Continuous Integration and Continuous Delivery) – набір принципів та набір практик, що дозволяють розробникам частіше та надійніше пояснювати зміни коду.

CORS (англ. Cross-Origin Resource Sharing) – механізм безпеки сучасних браузерів, який дозволяє веб-сторінкам використовувати дані, що знаходяться на інших доменах.

VCS (англ. Version Control System) – це програмне забезпечення, яке забезпечує функціональність управління версіями.

API (англ. Application Program Interface) – структура для створення служб HTTP.

ЗМІСТ

ВСТУП

Актуальність. У наш час складно уявити робоче місце без використання якої-небудь техніки, особливо, якщо ця техніка працюватиме справно довгий час. Для цього є працівники відділу технічного обслуговування, які займаються таким питанням, та надають послуги користувачам, якщо та або інша техніка вийшла з ладу.

Для зниження витрат за рахунок своєчасного обслуговування та можливість відстежувати, контролювати та створювати замовлення дистанційно – створюється веб-сайт.

За допомогою веб-сайту, користувачі мають змогу створювати замовлення, не відходячи від свого робочого місця. В свою чергу, відповідальні люди за технічне обслуговування КНТЕУ, обробляють нові замовлення, збирають первинну інформацію і за необхідності мають можливість відправити декілька коментарів для уточнення, або підкріпити будь-який файл до створеного замовлення. Сайт добре відображається як на ПК версії, так і на мобільних пристроях.

Мета дослідження: розробка та впровадження інформаційної системи для управління відділом технічного обслуговування КНТЕУ.

Об’єкт дослідження: організація послуг для технічного обслуговування.

Предмет дослідження: методи та алгоритми створення веб-сайту для технічного обслуговування КНТЕУ.

У відповідності з метою дослідження поставлені наступні завдання:

					КНТЕУ 121 06-20.МР		
Зм.	*Аркуш*	*№ докум.*	*Підпис*	*Дата*			
Зав. каф.		Криворучко О.В.		*24.01.20*	Розробка програмного забезпечення управління відділом технічного обслуговування КНТЕУ	*Стадія*	*Арку*	*Аркушів*
Керівник		Палагута К.О.		*24.01.20*		*В*	*4*	*44*
Гарант		Криворучко О.В.		*24.01.20*		Факультет інформаційних технологій 2м курс, 6 група		
Розробив		Хрущ О.Г.		*24.01.20*	*Вступ*			

- Дослідити інформаційну систему відділу технічного обслуговування КНТЕУ.
- Визначити процес, який необхідно автоматизувати для підвищення ефективності взаємодії з веб-сайтом.
- Написати веб-сайт згідно створеного технічного завдання.
- Протестувати наявний функціонал за допомогою unit-тестування.

Методи дослідження: процес взаємодії працівників відділу технічного забезпечення з працівниками КНТЕУ.

Наукова новизна дослідження полягає в створенні стабільно працюючого сервера за рахунок використання багатьох Docker-контейнерів, які взаємодіють між собою.

РОЗДІЛ 1.
ЗАГАЛЬНІ ВІДОМОСТІ ТА ОПИС ТЕХНОЛОГІЙ

1.1. Опис проекту

Основна ідея цього проекту – є створення інформаційної системи, за допомогою якої працівники вищого навчального закладу КНТЕУ, мають змогу використовувати створений веб-сайт з будь-якого пристрою та місця розташування, створювати електронне замовлення на обслуговування технічного засобу, дізнаватися статус виконання замовлення та отримувати додаткову інформацію по замовленню. Працівники центру інформаційних технологій – отримувати замовлення, обробляти та виконувати їх.

За допомогою цього сайту, вищий навчальний заклад отримає:

- Система керування замовленнями.
- Можливість підкріплювати один або декілька файлів, скріншотів.
- Коментувати замовлення.
- Отримувати інформацію на електронну адресу, після будь-яких змін на сайті.
- Система керування користувачами.
- Система керування ролями, доступами до певних дій на сайті.
- Гнучка система налаштування прав.
- Система керування списком обладнань, підв’язаних до серійного або інвентарного номеру.
- Глобальні налаштування на сайті за допомогою інтерфейса.

- Можливість редагувати маніфест файлу, який дозволяє виносити «ярлик» веб-сайту до робочого столу на ПК або телефоні.
- Система керування фоновими завданнями, відновлення завдань.
- Отримання інформації в реальному часі за допомогою технології Websocket.
- Додаткові дані в базі даних для керування пріоритетами замовлення, моделями обладнань та ін.

1.2. Використання REST API

REST (Representational State Transfer) – це стиль програмної архітектури, що забезпечує зручний та послідовний підхід до запиту та модифікації.

У системі RESTful ресурси зберігаються в сховищі даних; клієнт надсилає запит про те, що сервер виконує певну дію (наприклад, створює, отримує, оновлює або видаляє ресурс), а сервер виконує дію та надсилає відповідь, часто у вигляді подання зазначеного ресурсу.

Клієнт визначає дію, використовуючи HTTP тип, такий як POST, GET, PUT або DELETE.

Розуміння дизайну REST API:

- Клієнт-Сервер. Обмеження клієнт-сервер працює на концепції, що клієнт і сервер повинні бути відокремлені один від одного і дозволяти розвиватися індивідуально та незалежно.
- Не має стан. API REST не мають стану, це означає, що виклики можуть здійснюватися незалежно один від одного, і кожен виклик містить усі дані, необхідні для успішного завершення.
- Кеш. Оскільки API без статусу може збільшити накладні витрати на обробку великих навантажень вхідних та вихідних викликів,

REST API повинен бути розроблений для заохочення зберігання кешованих даних.

- Уніфікований інтерфейс. Ключем до роз'єднання клієнта з сервером є наявність єдиного інтерфейсу.
- Багаторівнева система. Як випливає з назви багаторівнева система – це система, що складається з шарів, причому кожен шар має певну функціональність та відповідальність

1.3. Docker-контейнеризація веб-сайту

У сучасній розробці, веб-сайти складаються з величезного числа технологій, крім розроблюваних розробником частин веб-сайту: frontend, backend, безліч сервісів, які мають певну логіку, є ще й зовнішні сервіси і додатки – база даних, система кешування, Load Balancer, Message Brokers, різні системи моніторингу. Існують інструменти і мови програмування: git, npm, php, node, golang і інші, до цього всього є ще й операційні системи. Все це активно оновлюється, в кожного є своя версія і розробникам потрібно постійно слідкувати за оновленнями зовнішніх сервісів або інструментів та підтримувати свій код для коректної роботи з новими версіями.

Для цього необхідно змінювати версію обраного сервіса на локальній машині, тестувати, можливо щось коригувати в коді, якщо нова версія не підтримується розробленим веб-сайтом. Всі ці зміни відправляти на CI/CD (continuous integration and continuous delivery), і вже після цього зробити теж саме на production сервері. Це складно, довго, необхідно постійно перемикатися між версіями, що не завжди є таким простим завданням. Якщо у нас кілька проектів, на які ми зазвичай перемикається - це також складно, тому що зовнішні версії сервісів та інструментів ми зазвичай не змінюємо з проекту в проект.

Цю проблему вирішує Docker контейнеризація. Для прикладу: розроблений веб-сайт відмінно працює з MySQL 5.6.x версії, тому що MySQL встановлювали одразу після встановлення операційної системи Linux і на той момент ця версія була останньою, і звісно, вся розробка відбувалась з використанням саме цієї версії. Настав час, коли розроблений веб-сайт потрібно залити на продакшн сервер. Встановлюємо всі інструменти і бібліотеки за прикладом того, як робили це на локальному ПК, але під час запуску веб-сайт щось пішло не так, виявляється, що при встановленні MySQL, система за замовчуванням підтягує останню стабільну версію, яка на той момент вже становила 8.0.x, в якій, для прикладу, розробники змінили спосіб з'єднання з самою базою даних, а розроблений продукт не підтримує новий спосіб з'єднання.

Якщо під час розробки цього проекту або на його завершення, використовувався Docker, нам не потрібно було б витрачати зайвий час на налаштування серверу, особливо, якщо збірка Linux відрізняється, як Ubuntu, Arch, CentOS та ін., в кожному який є свій пакетний менеджер, різні конфігурації та ін.

Docker — це програмне забезпечення з відкритим кодом, розроблене для полегшення та спрощення розробки веб-сайтів. Це набір «платформа як послуга» продуктів, які створюють ізольовані віртуалізовані середовища для побудови, розгортання та тестування додатків. Docker відокремлює веб-сайт від базової інфраструктури, з'єднавши свій код і всі його залежності в автономну сутність, яка буде працювати однаково в будь-якій підтримуваній системі.

Насправді, не початкова розробка є складною, а все те, що приходить після розробка. Розгортання коду є проблемою. Управління залежностями є проблемою. Відкатити останні зміни є проблемою. Усі ці речі стають складнішими, оскільки середовища розробки рідко ідентичні продакшн

серверу, і саме тут Docker-контейнери можуть допомогти. Вони дозволяють взяти будь-який розроблений веб-сайт, який необхідно запустити, згрупувати його в узгоджений формат і запустити його в будь-яку інфраструктуру, яка вміє обробляти контейнер. Кінцевий результат дуже схожий як на єдиний exe файл для всього розробленого коду.

За допомогою Docker, незалежно від мови програмування, яка використовується, або дистрибутиву Linux, на якому розроблявся веб-сайт, ми маємо змогу згорнути все це в один блок, який називається контейнером і він знає, як запускати веб-сайт.

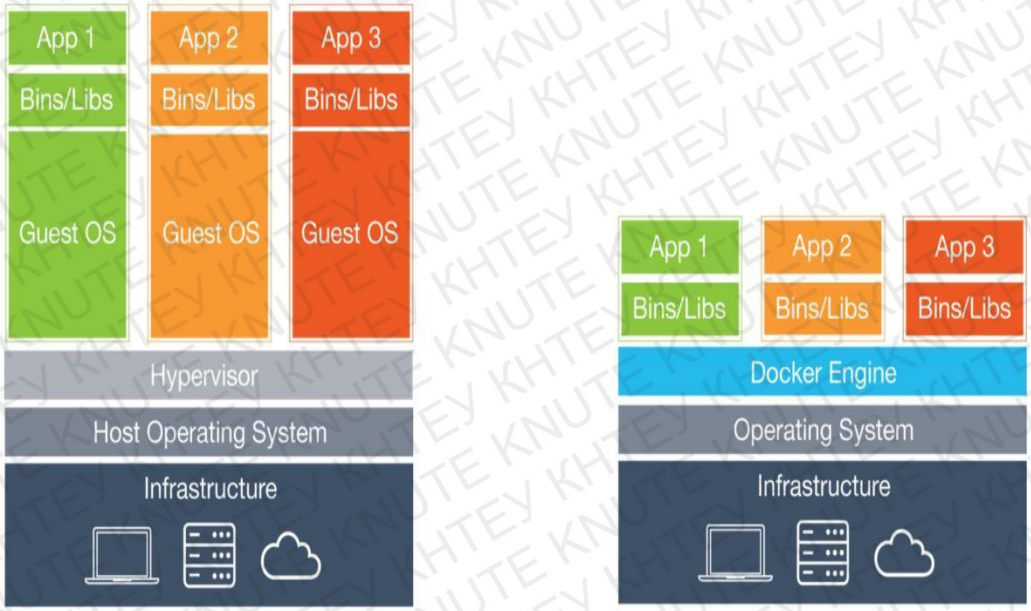

Рис. 1.1. Різниця між стеком віртуалізації та контейнерної інфраструктури

Контейнер Docker — це віртуалізоване середовище виконання, де користувачі можуть ізолювати веб-сайти від базової системи. Ці контейнери – це компактні, портативні пристрої, в яких можна швидко та легко запустити додаток.

Цінною особливістю є стандартизація обчислювального середовища, що працює всередині контейнера. Це не лише забезпечує те, що веб-сайт працює в однакових обставинах, але й спрощує обмін з іншими товаришами по команді.

Оскільки контейнери є автономними, вони забезпечують сильну ізоляцію, гарантуючи, що вони не перебивають інші запущені контейнери, а також сервер, який їх підтримує. На відміну від віртуальних машин, де віртуалізація відбувається на апаратному рівні, контейнери віртуалізуються на рівні програми. Вони можуть використовувати одну машину, спільно використовувати її ядро та віртуалізувати операційну систему для запуска ізольованих процесів - це робить контейнери надзвичайно легкими.

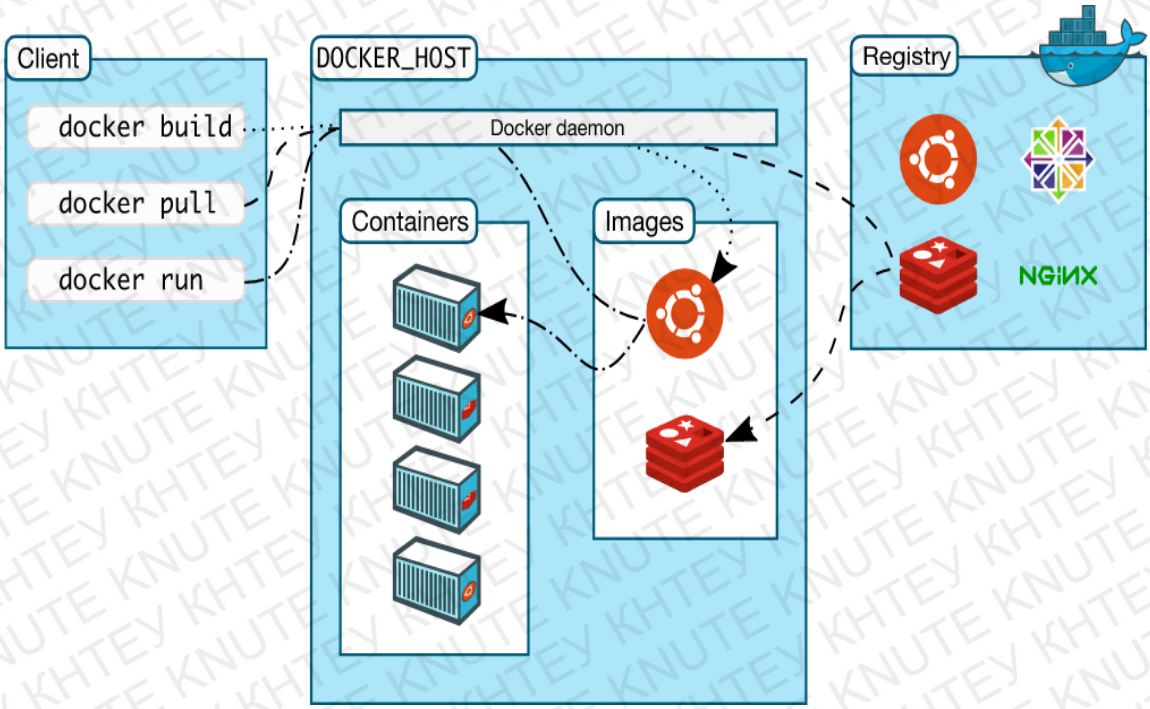

Рис. 1.2. Архітектура Docker

- Docker Daemon — прослуховує запити Docker API і керує об'єктами Docker, такими як зображення, контейнери, мережі та томи (volumes).
- Docker Client — є основним способом взаємодії багатьох Docker користувачів з Docker. Під час використання таких команд, як docker run, клієнт відправляє ці команди dockerd, який виконує їх. Команда docker використовує API Docker.
- Docker Registries — зберігає зображення Docker. Docker Hub - це публічний реєстр, який може використовувати кожен, і Docker

налаштований шукати зображення на Docker Hub за замовчуванням.

- Docker Images — це шаблон, доступний лише для читання, з інструкціями щодо створення контейнера Docker. Часто зображення базується на іншому зображенні, з деякими додатковими налаштуваннями.
- Docker Containers — це запущений екземпляр зображення.
- Docker Services — дозволяють масштабувати контейнери по декількох Docker Daemons, які працюють разом, як swarm з кількома менеджерами та працівниками.

Докер досягає ізоляції різних контейнерів за допомогою 4 основних понять:

- Cgroups
- Простори імен (Namespaces)
- Stackable Image-Layers і копіювання під час запису
- Мости віртуальної мережі (Virtual Network Bridges)

Операційна система Linux керує наявними апаратними ресурсами (memory, CPU, disk I/O, network I/O, та ін.) та забезпечує зручний спосіб для доступу та використання процесів до них. Наприклад, планувальник процесорних процесорів Linux, наприклад, піклується про те, щоб з часом кожен потік отримав деякий час на ядрі процесора, щоб жодна програма не застрягла в очікуванні часу на процесор.

Контрольні групи (cgroups) — це спосіб призначити підмножину ресурсів певній групі процесів. Це можна використати, наприклад, для того, щоб навіть якщо ваш процесор дуже зайнятий сценаріями Python, ваша база даних PostgreSQL все ще отримує виділені процесор і оперативну пам'ять.

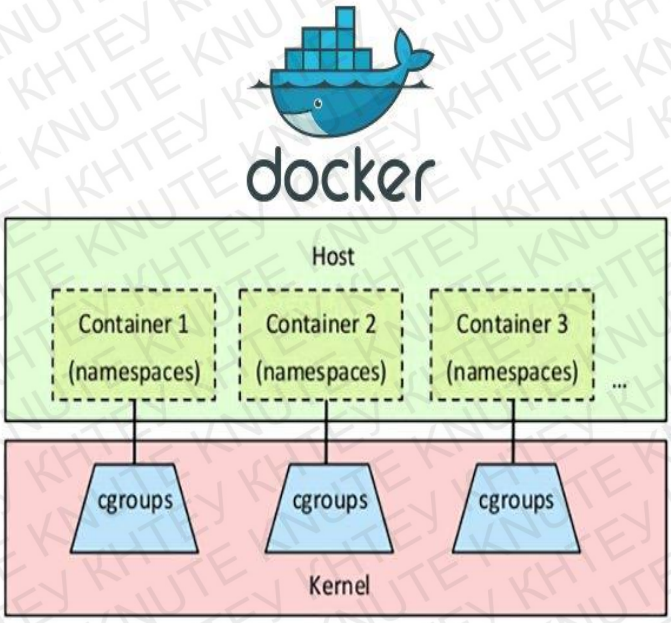

Рис. 1.3. Управління ресурсами Docker за допомогою cgroups

Namespaces. У той час як групи ізолюють апаратні ресурси, простори імен виділяють і віртуалізують системні ресурси. Приклади системних ресурсів, які можна віртуалізувати, включають ідентифікатори процесу, імена хостів, ідентифікатори користувачів, доступ до мережі, міжпроцесовий зв'язок та файлові системи.

PID Namespaces. Операційна система Linux організовує процеси в так званому дереві процесів. Корінь дерева — це перший процес, який починається після завантаження операційної системи та має PID 1. Оскільки може існувати лише одне дерево процесу, а всі інші процеси (наприклад, Firefox, емулятори терміналів, SSH-сервери) повинні бути (безпосередньо або опосередковано) розпочатий цим процесом. Через те, що цей процес ініціалізує всі інші процеси, його часто називають процесом init.

Filesystem Namespaces. Іншим випадком використання просторів імен є файлова система Linux. Подібно до просторів імен PID, простори імен файлової системи віртуалізують та ізолюють частини дерева — у цьому випадку дерево файлової системи. Файлова система Linux організована у вигляді дерева і має корінь, як правило, її називають «/». Для досягнення

ізоляції на рівні файлової системи, простір імен буде зіставити вузол у дереві файлової системи на віртуальний корінь всередині цього простору імен. Переглядаючи файлову систему всередині цього простору імен, Linux не дозволяє вам вийти за рамки віртуалізованого кореня.

Інші Namespaces. Окрім PID та Filesystem Namespaces, існують також інші види просторів імен.

Docker зберігає images в stackable шарах. Шар містить зміни до попереднього шару. Наприклад, якщо встановити першу версію PHP, а потім скопіювати скрипт PHP, то image матиме 2 додаткових шара: один містить виконувальний файл PHP, а інший – скрипт. Щоб тричі не зберігати Ubuntu, шари незмінні та загальні. Docker використовує функцію копіювання при записі, щоб зробити копію файлу лише у випадку змін. Запускаючи контейнер на основі зображення, Docker Daemon надасть усі шари, що містяться в цьому зображенні, і помістить його в ізольований простір Filesystem Namespaces для цього контейнера. Поєднання stackable шарів, копіювання при записі, і Filesystem Namespaces дозволяють запустити контейнер, повністю незалежний від речей, встановлених на хості Докера, не витрачаючи багато місця. Це одна з причин, чому контейнери більш легкі порівняно з віртуальними машинами.

Мережевий міст (network bridge) — це комп'ютерний мережевий пристрій, який створює єдину сукупну мережу з декількох мереж зв'язку або мережевих сегментів.

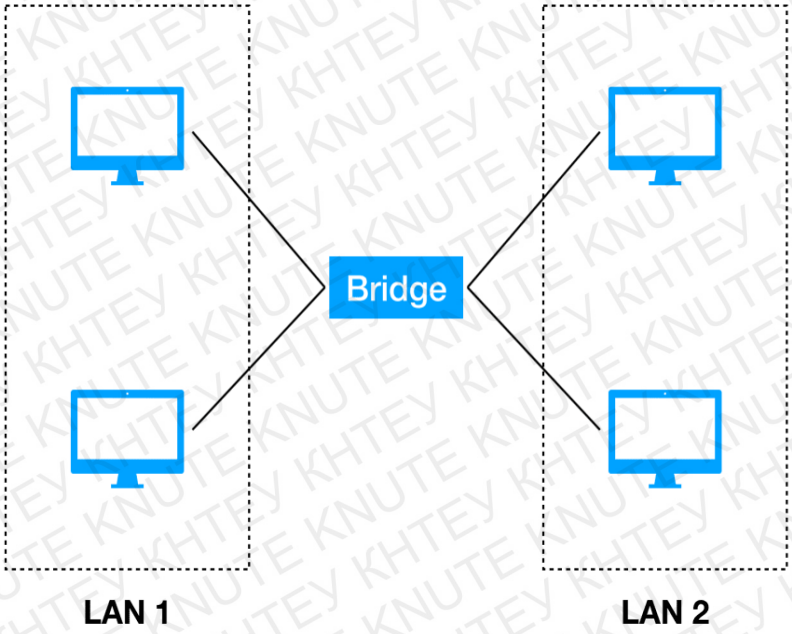

Рис. 1.4. З’єднання двох мережевих сегмента (LAN 1 та LAN 2)

Зазвичай у нас є лише обмежена кількість мережевих інтерфейсів (наприклад, фізичні мережеві карти) на хості Docker, і всі процеси так чи інакше потребують спільного доступу до нього. Для того, щоб ізолювати мережу контейнерів, Docker дозволяє створити віртуальний мережевий інтерфейс для кожного контейнера. Потім він підключає всі інтерфейси віртуальної мережі до адаптера хост-мережі.

1.4. Висновки до розділу 1

Для успішного створення веб-сайту, необхідно мати чітко визначені вимоги, добре спроектувати систему після технічного завдання, та дотримуватимуся їх під час розробки.

Створення односторінкового веб-сайту (SPA) – вимагає сучасних тенхологій, як на стороні серверної частини, так і при клієнтської частини. За допомогою сучасних фреймворків, таких як Vue.js, React або Angular – це стає можливістю, що змінює звичний процес розробки з серверного рендерингу на клієнтський.

За допомогою використання Docker технології, процес розробки, підняття сервера, налаштування оточення і розгортання веб-сайту на продакшн сервері – стає в декілька разів простіше, за рахунок використання певної абстракції, незалежність від ОС, відмінностях у версіях бібліотек та іншого.

РОЗДІЛ 2.
ВИБІР ТЕХНОЛОГІЙ, ПРОЕКТУВАННЯ ПРОЕКТУ

2.1. Вибір технологій

Перш за все, будь-який веб-сайт повинен розроблятись враховуючи швидкодію, захищеність та відмовостійкість.

- Швидкодія: проект буде поділений на кілька відокремлених модулів, кожен з яких може працювати незалежно від інших модулів, працювати на окремих серверах або на різних ядрах однієї системи.
- Захищеність: проект матиме unit-тести, систему авторизації та можливість встановлення на сервер і мати доступ тільки по внутрішній мережі.
- Відмовостійкість: проект працюватиме на операційній системі Linux та кожна система працюватиме в Docker контейнері, який буде слідкувати за доступністю і перезавантажувати при необхідності.

Система буде розподілена на 4 основні модулі:

- Backend – це серверна частина, яка є основною, працюватиме с запитами від користувачів, взаємодіяти з базою даних та іншими системами.
- Frontend – це клієнтська частина, яка відповідає виключно за відображення контенту користувача.
- Websocket – це також серверна частина, але тільки відповідає за відображення змінених даних в реальному часі.

- Інші сервіси, які необхідні для повноцінної роботи проекту

Для Backend частини були обрані такі технології та інструменти:

- PHP – це інтуїтивна мова сценаріїв на стороні сервера. Як і будь-яка інша мова сценаріїв, вона дозволяє розробникам створювати логіку у створенні вмісту веб-сторінок і обробляти дані, повернуті з веб-браузера. PHP також містить ряд розширень, які дозволяють легко взаємодіяти з базами даних, витягуючи дані для відображення на веб-сторінці і зберігаючи інформацію, введену відвідувачем веб-сайту, назад до бази даних.
- Laravel – це фреймворк PHP з відкритим вихідним кодом, який є надійним і легким для розуміння. Вона слідує шаблону model-view-controller (MVC).
- JSON Web Tokens (JWT) – це система авторизація, яка визначає компактний і автономний спосіб безпечної передачі інформації між сторонами як об'єкт JSON
- PHPStorm – це інноваційне інтегроване середовище розробки (IDE), розроблене JetBrains для PHP і веб-розробників. Програмне забезпечення використовує IntelliJ IDEA, щоб дозволити розробникам писати код на декількох мовах, включаючи CSS, HTML5, JavaScript.

Для Frontend частини були обрані такі технології та інструменти:

- Javascript – це мова сценаріїв або програмування, яка дозволяє реалізовувати складні речі на веб-сторінках - кожен раз, коли веб-сторінка більше, ніж просто статично відобразити інформацію для перегляду – відображення своєчасного оновлення вмісту, інтерактивних карт, тощо.

- Vue – це фреймворк для створення користувацьких інтерфейсів. На відміну від фреймворків-монолітів, Vue створений придатним для поступового впровадження. Його ядро в першу чергу вирішує завдання рівня уявлення (view), що спрощує інтеграцію з іншими бібліотеками та існуючими проектами.
- Webpack – це статичний модуль bundler для сучасних програм JavaScript. Коли webpack обробляє програму, вона внутрішньо створює графік залежностей, який відображає кожен модуль, необхідний вашому проекту, і генерує один або більше пакетів.
- Babel – це інструмент, який використовується в основному для перетворення коду ECMAScript 2015+ у зворотну сумісну версію JavaScript у поточних і старих браузерах або середовищах.
- Eslint – це тип статичного аналізу, який часто використовується для пошуку проблемних шаблонів або кодів, які не відповідають певним стилям. Для більшості мов програмування існують кодові лінти, і компілятори інколи включають linting в процес компіляції.
- Scss – це таблиця стилів. Вона дозволяє використовувати змінні, вкладені правила, міксини, функції та багато іншого, все з повністю CSS-сумісним синтаксисом. Scss допомагає зберігати великі таблиці стилів добре організованими.

Для Websocket частини були обрані такі технології та інструменти:

- Typescript – це мова програмування з відкритим кодом, розроблена та підтримувана корпорацією Майкрософт. Це сувора синтаксична надмножина JavaScript і додає до мови додаткову статичну типізацію.
- Nodejs – це серверна платформа, побудована на JavaScript Chrome від Google Chrome (V8 Engine).

- Jest – це тестування JavaScript, з акцентом на простоту. Працює з проектами, що використовують: Babel, TypeScript, Node.js, React, Angular і Vue.js
- Socket.io – дозволяє здійснювати зв'язок у режимі реального часу, двонаправлений і на основі подій. Він працює на кожній платформі, браузері або пристрої, орієнтуючись однаково на надійність і швидкість.

Інші сервіси:

- MySQL – це швидка, проста у використанні СУБД, яка використовується для багатьох малих і великих підприємств. MySQL розробляється, продається і підтримується компанією MySQL.
- Docker – це набір «платформа як послуга» продуктів, які створюють ізольовані віртуалізовані середовища для побудови, розгортання та тестування додатків. Docker відокремлює веб-сайт від базової інфраструктури, з'єднавши свій код і всі його залежності в автономну сутність, яка буде працювати однаково в будь-якій підтримуваній системі.
- Traefik – це сучасний зворотний проксі-сервер HTTP і балансувач навантаження, що полегшує розгортання мікропослуг. Інтегрується з існуючими компонентами інфраструктури (Docker, режим Swarm, Kubernetes, Marathon та ін.) та налаштовує себе автоматично та динамічно.
- Nginx – це програмне забезпечення з відкритим кодом для веб-обслуговування, зворотного проксі-сервера, кешування, балансування навантаження, потокового передавання медіа тощо.

- RabbitMQ – це програма для черги повідомлень, також відома як брокер повідомлень або менеджер черг.
- Redis – це сховище структур даних з відкритим кодом, яке використовується як посередник баз даних, кешу та повідомлень.
- MailDev – це простий спосіб перевірити згенеровані електронні листи проекту під час розробки.

2.2. Проектування бази даних

MySQL – це реляційна база даних, структурована або організована у вигляді таблиць, стовпців та рядків, де таблиці представляють об'єкти, стовпці – поля, а рядки – записи.

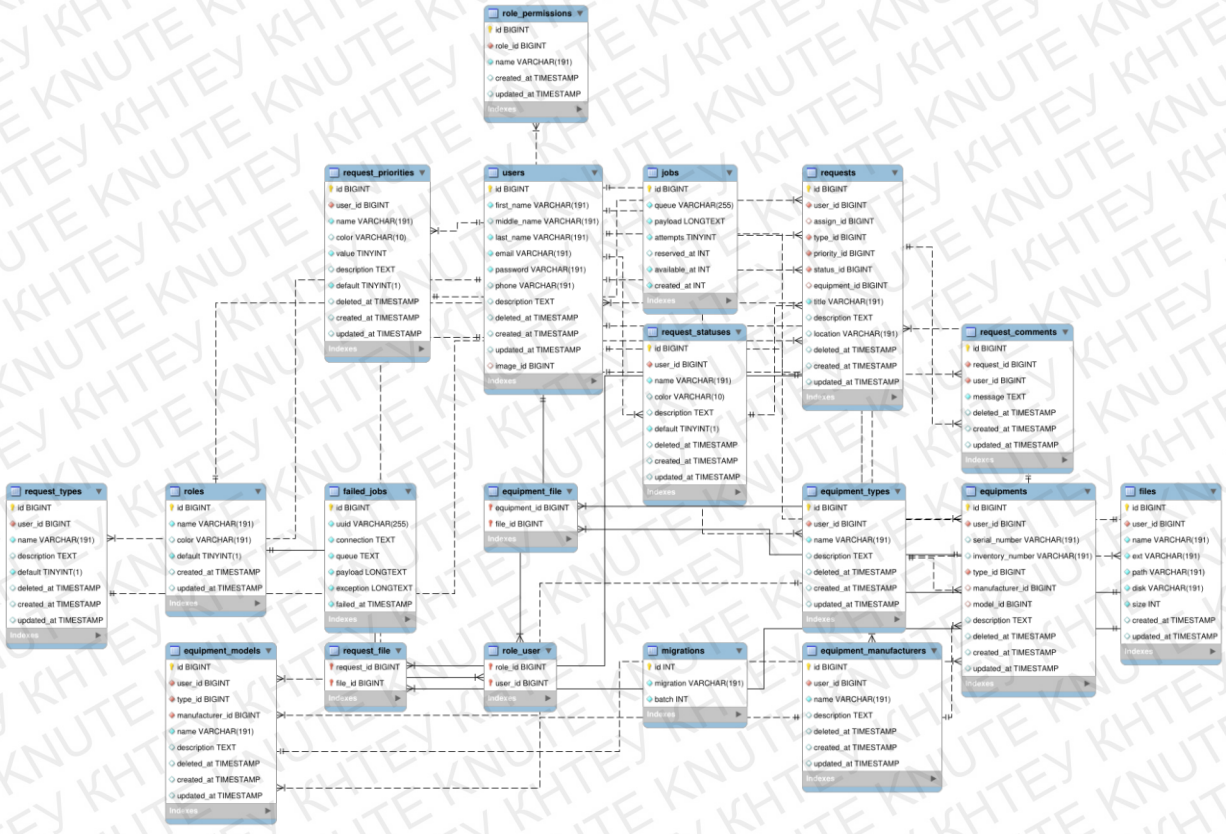

Рис. 2.1. Реляційна модель бази даних

MySQL – це реляційна база даних, структурована або організована у вигляді таблиць, стовпців та рядків, де таблиці представляють об'єкти, стовпці – поля, а рядки – записи.

Таблиці база даних:

- equipments – список обладнань.
- equipment_file – зв’язок many-to-many між файлами та обладнанням.
- equipment_manufacturers – список виробників обладнань.
- equipment_models – список моделей обладнань.
- equipment_types – список типів обладнань.
- failed_jobs – список невдалих задач при виконання фонових задач.
- files – список завантажених файлів на сайт.
- jobs – список фонових задач.
- migrations – створюється під час міграції від фреймворка laravel, містить в собі всі створені таблиці.
- requests – список замовлень.
- request_comments – список коментарів в замовленні.
- request_file – зв’язок many-to-many між файлами та замовленнями.
- request_priorities – список пріоритетів в замовленнях.
- request_statuses – список статусів в замовленнях.
- request_types – список типів в замовленнях.
- roles – список ролей.
- role_permissions – список активованих доступів для ролі.
- role_user – зв’язок many-to-many між ролями та користувачами.
- users – список користувачів.

Спроектована база даних дозволяє зберігати всі необхідні дані для цього проекту, які будуть використовуватись користувачами системи. Для подальшого оновлення, редагування чи видалення даних, потрібно спроектувати систему міграцій.

Міграції – це файли, які зберігаються в проекті, містять в собі набір запитів та виконуються на сервері по запиту адміністратора. Це дозволить безболісно керувати таблицями за допомогою коду, а також, в разі необхідності – скасувати дії останньої міграції.

2.3. Проектування взаємодії між різними частинами системи

Проект має багато технологій та систем, які необхідно спроектувати їх взаємодію до початку розробки. Як працюватиме та, чи інша система, з чим вона буде взаємодіяти і які дані зберігати або обробляти.

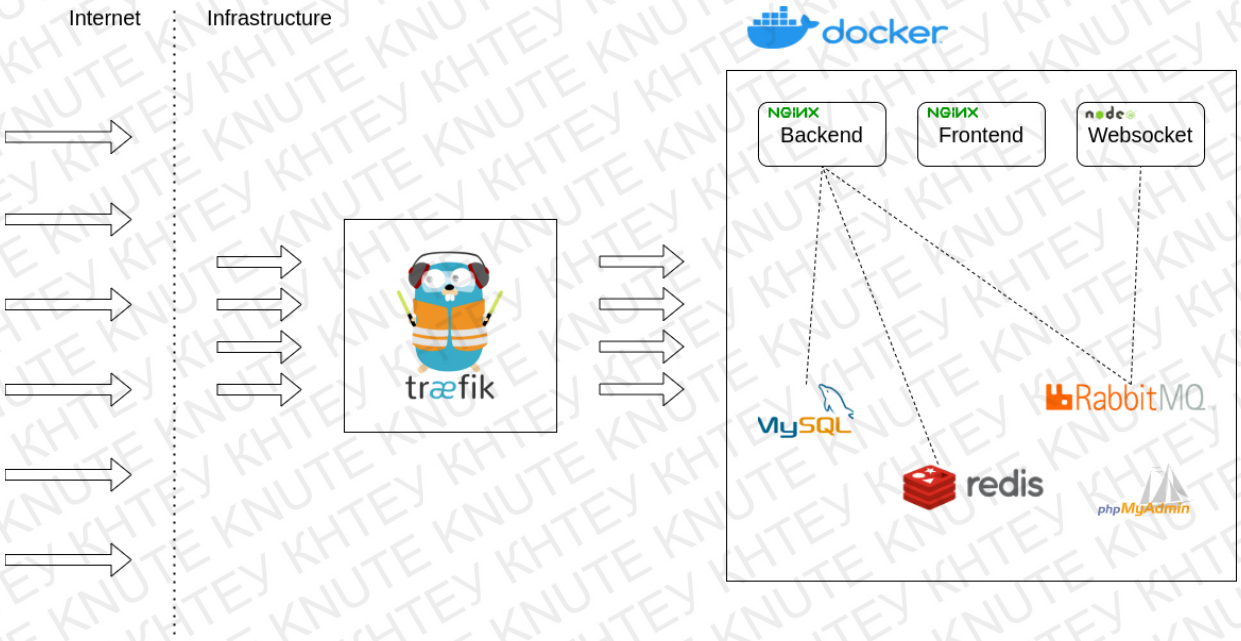

Рис. 2.2. Проектування взаємодії різних частин системи

Перш за все, всі запити з інтернету отримує Traefik load-balancer, який відкритий до мережі та знає все, що знаходиться всередині Docker контейнерів. В залежності від отриманих даних та налаштувань, Traefik перенаправляє запити до одного з контейнерів, який починає обробляти запит. Traefik система буде налаштована таким чином, що всі частини системи будуть доступні тільки за однією адресою, кожна з яких займає певний порт.

Кожен сервіс буде «обгорнутий» в Docker контейнер – це певний слой абстракції, який дозволяє налаштувати єдине середовище для того, щоб налаштування на продакш сервері були ідентичні, як при локальній розробці. Docker слідкує за всіма контейнерами, та автоматично перезавантажує, якщо сервіс не відповідає.

Redis буде використовуватись в якості кеша, це дозволяє значно пришвидшити сайт, якщо порівнювати з файловим кешом, а також дозволяє використовувати більше функцій, як наприклад – теги, за допомогою яких легко слідкувати та анулювати кеш.

RabbitMQ знадобиться для взаємодії з іншим серверами/процессами в системі, в цьому проекті – це взаємодія PHP та Node.js, що забезпечує оновлення інформації в реальному часі.

2.4. Проектування системи email-повідомлень

У цьому проекті буде відправлятися велике число email-повідомлень і потрібно реалізувати механізм, який не буде навантажувати систему, а також забезпечити швидкий зворотній зв'язок з користувачем (відгук від сервера), адже відправка одного повідомлення займає велику кількість часу.

Найоптимальніший спосіб вирішення цієї проблеми — це відкладена відправка. Наприклад, коли користувач додає нового користувача в систему, йому необхідно відправити email-повідомлення на почту з тимчасовим паролем, або ж коли створюється нове повідомлення – теж потрібно відправляти email-повідомлення користувачам.

Також необхідно пам’ятати, що сервер для відправки email-повідомлень теж може бути тимчасово недоступним, тим самим блокуючи роботу сайта.

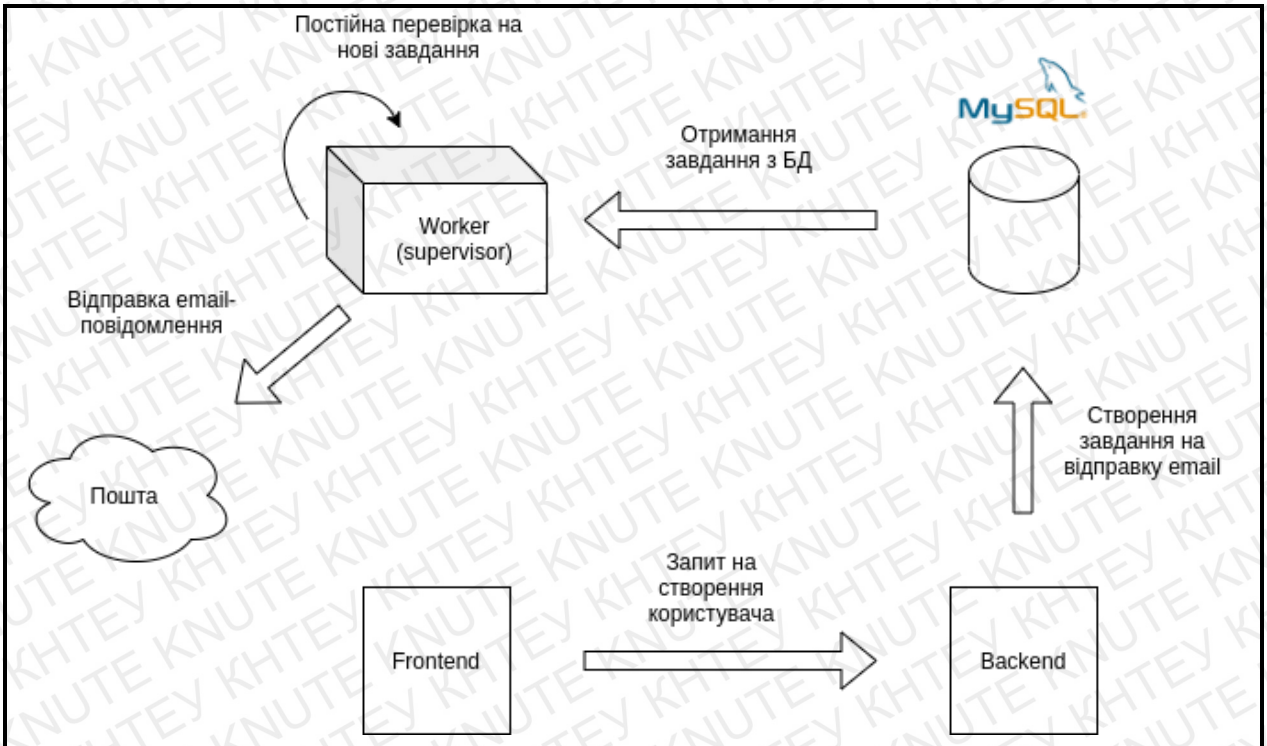

Рис. 2.3. Проектування взаємодії системи email-повідомлень

В наведеному прикладі (Рис 2.3) бачимо, що після того, як користувач створить запит на створення іншого користувача, відповідь від сервера прийде відразу ж, не чекаючи дійсної відправки email-повідомлення. Це дозволяє в декілька разів пришвидшити запит на створення користувача, відправляти email-повідомлення в порядку черги, а також дозволяє гнучко керувати завданнями на відправку email-повідомлень або відстежувати завдання, які не завершились успіхом. Для відстежування завдань – спроектовано 2 MySQL таблиці, які будуть зберігати ці дані.

2.5. Вибір для розташування і збереження проекту

Кожен проект необхідно десь зберігати та керувати. Для цього необхідно обрати систему контролю версій. VCS (Version Control System) — це програма для роботи з постійно мінливою інформацією. Таке ПО може зберігати безліч версій одного і того ж файлу (документа) і повертатися до попереднього стану.

У наш час існують такі основні VCS:

- Git;
- SVN;
- Mercurial;
- CVS (Concurrent Versions System);
- Team Foundation Server.

Перш ніж обрати VCS для проекту, необхідно розглянути переваги та недоліки існуючих ПО. Для порівняння виберемо Git і SVN, так як вони є найбільш відомими і популярними серед розробників

- Git – це розподілений VCS. SVN — це нерозподілений VCS.
- Git має централізований сервер і сховище. SVN не має централізованого сервера або сховища.
- Вміст у Git зберігається як метадані. SVN зберігає файли вмісту.
- З гілками Git працювати простіше, ніж з гілками SVN.
- Git не має функції глобального номера редакції, як у SVN.
- Git має кращий захист вмісту, ніж SVN.
- Git був розроблений для ядра Linux Лінусом Торвальдом. SVN був розроблений CollabNet, Inc.
- Git поширюється під GNU, а його підтримку контролює Хуніо Хамано. Apache Subversion (або SVN) — поширюється за ліцензією з відкритим кодом.

Але основним факторів вибору VCS є те, де проект буде зберігається. Для майданчика зберігання обрана платформа Github, яка працює поверх технології Git. Тому для цього проекту був обрана Git в якості VCS. Ще одним важливим фактором цього вибору став Github Actions, який допоможе побудувати зручний CI/CD.

2.6. Висновки до розділу 2

В цьому розділі був описаний функціонал проекту, який необхідно реалізувати, розглянули та обрали технології, за допомогою яких буде будуватися проект, як різні частини системи будуть взаємодіяти між собою. Основна система буде поділена на 3 основні частини, кожна з частин має свій набір технологій та інструментів, а саме:

- Backend розробляється на мові програмування PHP, з використання Laravel фреймворку.
- Frontend розробляється на мові програмування Javascript, з використанням Vue.js фреймворку.
- Webscoket розробляється на мові програмування Typescript, з використанням Node.js фреймворку.

В якості збереження даних була обрана реляційна база даних MySQL, під цю базу даних спроектували таблиці та зв'язки між даними. Обрані допоміжні сервіси, які необхідні для стабільної роботи веб-сайту, а також описана і спроектована їх взаємодія між основними частинами проекту. Обрано систему роботи з файлами (VCS) — Git. В якості платформи для зберігання, був обраний Github.

РОЗДІЛ 3.
РОЗРОБКА ПРОЕКТУ, DOCKER-КОНТЕЙНЕРИЗАЦІЯ ТА НАЛАШТУВАННЯ CI/CD

3.1. Налаштування Docker-контейнерів

Docker спростив створення середовища для розробки. Однак, якщо потрібно створити більше одного контейнера для веб-сайту, доведеться створити кілька файлів Docker. Це додає навантаження на їх утримання, а також досить трудомістке.

Docker Compose (docker-compose.yml) — вирішує цю проблему, дозволяючи використовувати файл YAML для одночасної роботи з декількома контейнерними сервісами. Можна встановити бажану кількість контейнерів, їх збірки та конструкції сховищ, а потім за допомогою одного набору команд можливо створювати, запускати та налаштовувати всі контейнери.

Docker Compose чудово підходить для розробки, тестування та stage середовищ, а також для постійних робочих процесів інтеграції.

Для цього проекту, необхідно створити докер-контейнери для таких частин/сервісів:

- Клієнтська частина (web)
- Серверна частина + nginx (server та nginx)
- Сервіс для підтримання оновлень в реальному часі (websocket)
- Load balancer (traefik)
- Система, яка обробляє певні процеси у фоновому режимі (php-worker)

- Реляційна база даних (mysql)
- NoSQL база даних (redis)
- Брокер повідомлень (rabbitmq)
- Система управління базою даних (phpmyadmin)
- Збір логів, відображення метриків (prometheus та grafana)

Цей набір сервісів повинен по-різному запускатися, в залежності від розробки, деплоя на продакшен або запуска тестів. Також, всі дані в docker контейнерах повинні зберігатися на диску, та не бути видалені при перезапуску контейнера, і щоб перезавантажувалися при будь-яких падіннях сервісу.

Розглянемо детальніше створення одного з контейнерів, які приведено в Додатку А — це php-worker.

```
php-worker:
  build: .docker/php-worker
  container_name: u_php_worker
  restart: always
  depends_on:
    - server
  volumes:
    - ./:${DOCKER_CONTAINER_PATH}
    - ./.docker/php-worker/supervisord.d:/etc/supervisord.d
```

Рис. 3.1. Параметри для php-worker в docker-compose.yml

Параметри складаються з таких основних частин:

- build: може бути вказаний як рядок, що містить шлях до контексту побудови або як об’єкт із шляхом, зазначеним у контексті, а також необов’язково Dockerfile та args.
- container_name: вказане власне ім'я контейнера, а не сформоване ім'я за замовчуванням.

- restart: «no» — це політика перезапуску за замовчуванням, і вона не перезапускає контейнер ні за яких обставин. Коли вказано «always», контейнер завжди перезавантажується.
- depends_on: вирішує залежність та порядок створення контейнера.
- volumes: щоб усі ці дані зберігалися, ми використовуватимемо Docker Volumes, які є лише частинами файлової системи Docker Host (каталогу або блочного пристрою, відформатованого за допомогою файлової системи), які можна встановити всередині контейнера в будь-якому бажаному місці файлової системи контейнера.

З зазначених налаштувань в файлі — це означає, що контейнер буде запущено тільки після запуску сервера, це важливо, тому що фоновий процес працює поверх серверної частини. Всі створені файли, та будь-які зміни в контейнері — будуть одразу ж відображені у файловій системі, тому що за допомогою Docker Volume були додані «сімлінки».

Всередині цього контейнера працює Supervisor — це система клієнт/сервер, яка дозволяє своїм користувачам контролювати ряд процесів в операційних системах, подібних до UNIX.

Особливості Supervisor:

- Простий. Налаштовується за допомогою простого конфігураційного файлу у стилі INI, який легко засвоїти. Він надає безліч варіантів для кожного процесу, які полегшують ваше життя, наприклад, перезапуск невдалих процесів та автоматичне обертання журналу.
- Централізований. Пропонує одне місце для запуску, зупинки та моніторингу процесів. Процесами можна керувати як окремо, так і групами. Можна налаштувати Supervisor для надання локального або віддаленого командного рядка та веб-інтерфейсу.

- Ефективний. Запускає свої підпроцеси за допомогою fork/exec, і підпроцеси не демонізуються. Операційна система повідомляє Supervisor одразу, коли процес завершується, на відміну від деяких рішень, які покладаються на неприємні файли PID та періодичне опитування для перезапуска невдалих процесів.
- Розширюваний. Має простий протокол сповіщення про події, який програми, написані будь-якою мовою, можуть використовувати для моніторингу, а також інтерфейс XML-RPC для управління.
- Сумісний. Працює майже на всьому, крім Windows. Він протестований та підтримується на Linux, Mac OS X, Solaris та FreeBSD. Він повністю написаний на Python, тому для встановлення не потрібен компілятор C.
- Перевірений. Хоча Supervisor сьогодні дуже активно розробляється, це не нове програмне забезпечення. Supervisor існує роками і вже використовується на багатьох серверах.

Ще один файл, який разом з проектом відправляється в Docker контейнер, це supervisord.conf. Цей файл є основним для Supervisor, який містить налаштування, та використовується в supervisord і supervisorctl.

```
[supervisord]
nodaemon=true
[supervisorctl]
[inet_http_server]
port=127.0.0.1:9001
[rpcinterface:supervisor]
supervisor.rpcinterface_factory=supervisor.rpcinterface:make_main_rpcinterface

[include]
files = supervisord.d/*.conf
```

Рис. 3.2. Налаштування supervisord.conf

Цей файл підгружає всі конфігураційні файли проекту, які вже будуть запущені у фоному процесі, в якому вказується команда, яка буде запущена та кількість процесів, які будуть оброблювати дані.

```
[program:laravel-email]
process_name=%(program_name)s_%(process_num)02d
command=php /var/www/artisan queue:work database --queue=email --tries=30
autostart=true
autorestart=true
numprocs=1
user=www-data
redirect_stderr=true
```

Рис. 3.3. Налаштування laravel-email.conf

На рисунку (Рис. 3.3) відображена команда, яка виконує відправлення email повідомлення користувачам системи.

```
FROM php:7.4-fpm-alpine

RUN apk add shadow && usermod -u 1000 www-data && groupmod -g 1000 www-data

RUN apk add supervisor autoconf build-base git libzip-dev zip curl-dev

RUN pecl install -o -f redis

RUN docker-php-ext-install mysqli && \
    docker-php-ext-install pdo_mysql && \
    docker-php-ext-install curl && \
    docker-php-ext-install zip && \
    docker-php-ext-install sockets && \
    docker-php-ext-enable redis

COPY supervisord.conf /etc/supervisord.conf

ENTRYPOINT ["/usr/bin/supervisord", "-n", "-c",  "/etc/supervisord.conf"]

WORKDIR /etc/supervisor/conf.d/
```

Рис. 3.4. Dockerfile для php-worker контейнера

З цього файлу Docker Compose створює образ контейнера (вказаний в build параметрі). В цьому файлі вказаний образ PHP мови програмування, з версією 7.4. Alpine — легкий, простий та безпечний дистрибутив Linux, орієнтований на безпеку, заснований на musl libc та busybox. Alpine приблизно в 30 разів менше, ніж Debian. Докеризована версія Alpine 3.6 важить 3,96 мб. — це є основною перевагою цього дистрибутива Linux.

Після налаштувань всіх інших контейнерів, які вказані в Додатку А, ми маємо змогу побачити в Traefik Load Balancer всі запущені docker образи, за

якими параметрами вхідного запиту від користувача, на який контейнер його відправить система.

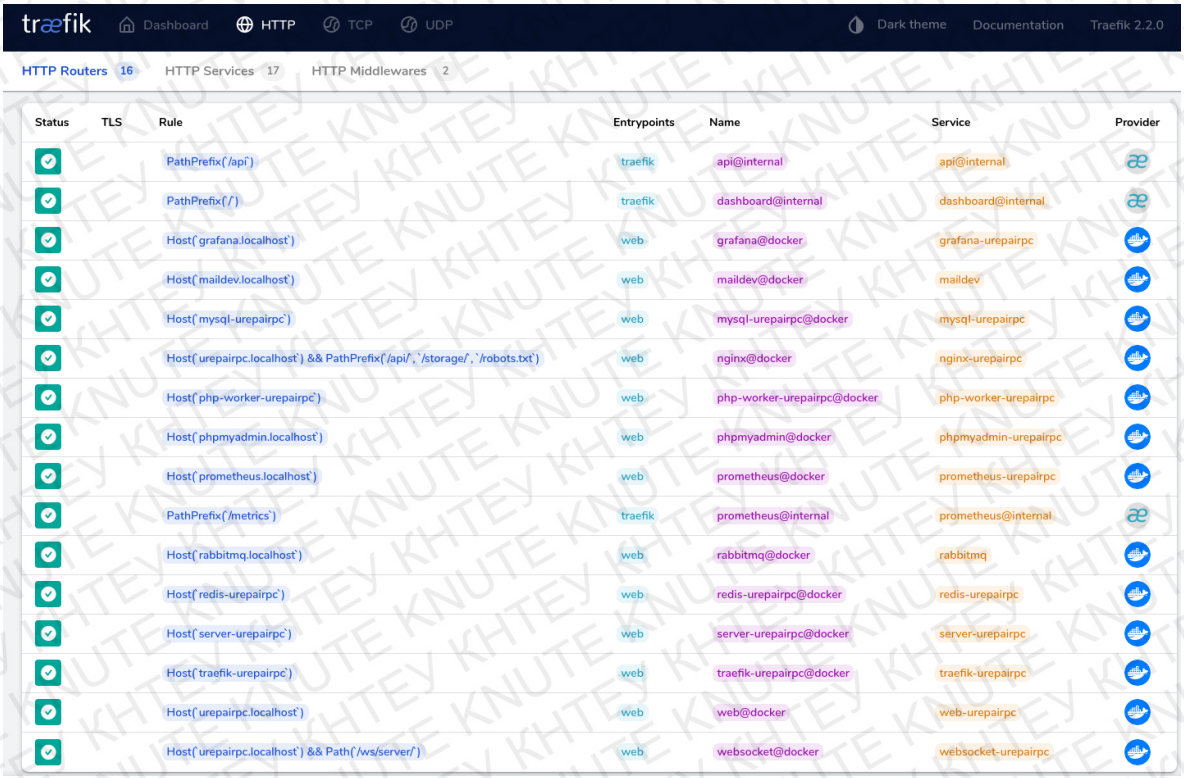

Рис. 3.5. Traefik HTTP з відображенням налаштованих docker-контейнерів

Після налаштувань, необхідно перевірити запущені контейнери – це можливо зробити за допомогою команди: *docker-compose ps*.

```
→ uRepairPC git:(dev) ✗ sudo docker-compose ps
    Name                  Command               State                                        Ports
-------------------------------------------------------------------------------------------------------------------------
u_grafana      /run.sh                          Up      3000/tcp
u_maildev      bin/maildev --web 80 --smtp 25   Up      0.0.0.0:25->25/tcp, 80/tcp
u_mysql        docker-entrypoint.sh mysqld      Up      0.0.0.0:3306->3306/tcp, 33060/tcp
u_nginx        /docker-entrypoint.sh nginx      Up      443/tcp, 80/tcp
u_php_worker   /usr/bin/supervisord -n -c ...   Up      9000/tcp
u_phpmyadmin   /docker-entrypoint.sh apac ...   Up      80/tcp
u_prometheus   /bin/prometheus --config.f ...   Up      9090/tcp
u_rabbitmq     docker-entrypoint.sh rabbi ...   Up      15672/tcp, 25672/tcp, 4369/tcp, 5671/tcp, 0.0.0.0:5672->5672/tcp
u_redis        docker-entrypoint.sh redis ...   Up      0.0.0.0:6379->6379/tcp
u_server       docker-php-entrypoint /bin ...   Up      9000/tcp
u_traefik      /entrypoint.sh --log.level ...   Up      0.0.0.0:80->80/tcp, 0.0.0.0:8080->8080/tcp
u_web          docker-entrypoint.sh /bin/ ...   Up      80/tcp
u_websocket    docker-entrypoint.sh /bin/ ...   Up      3000/tcp
```

Рис. 3.6. Відображення запущених процесів через docker-compose

3.2. Розробка з використанням Traefik Proxy

З використанням Traefik, підхід до розробки з використанням різноманітних сервісів – трохи змінилась. Перш за все, всі запущені сервіси займали певний порт і було складно налаштувати проксі для того, щоб запити

до різних серверів, відображались в браузері як запити до одного сервера. Це необхідно для того, щоб обійти механізм безпеки в сучасних браузерів – CORS (Cross-Origin Resource Sharing) і щоб запити не блокувались браузером в спробі відправити запит з localhost до localhost:8080 та ін.

Тому, при використанні Wepback Dev Server, розробники використовували proxy, яка підтримується бібліотекою. Це виглядало наступним чином:

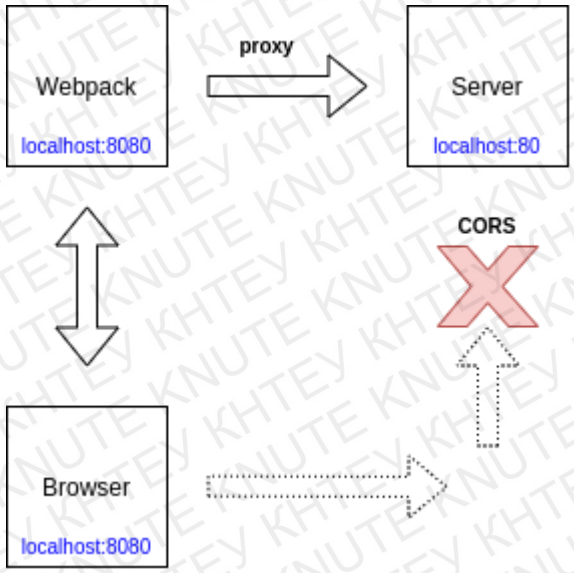

Рис. 3.7. Приклад роботи Proxy в Webpack Dev Server

Тепер розглянемо схему роботи без використання Proxy в Webpack Dev Server, а з налаштування Traefik Proxy. Перш за все, треба обрати будь-який домен, на який будуть надходити запити, та який Traefik буде направляти на Docker контейнери, в залежності від вхідних параметрів. Для локальної розробки були налаштовані такі параметри:

- Обраний домен: urepairpc.localhost. Оскільки, *.localhost домен має спеціальні прівілегії, то не має необхідності налаштовувати hosts, який на запитах до цього домену буде дивитися на 127.0.0.1 адресу.
- Запити на urepairpc.localhost домен, та з шляхом: /api/, /storage/ або /robots.txt – направляються до серверної частини.

- Запити на urepairpc.localhost домен, та з шляхом: /ws/server/ – направляються до системи оновлень в реальному часі (websocket).
- Всі інші запити надходять до клієнтської частини

Таким чином, всі запити надходять тільки на один домен, Traefik обробляє запит та перенаправляє до необхідного Docker контейнера. Схема роботи відображена на Рис. 3.7.

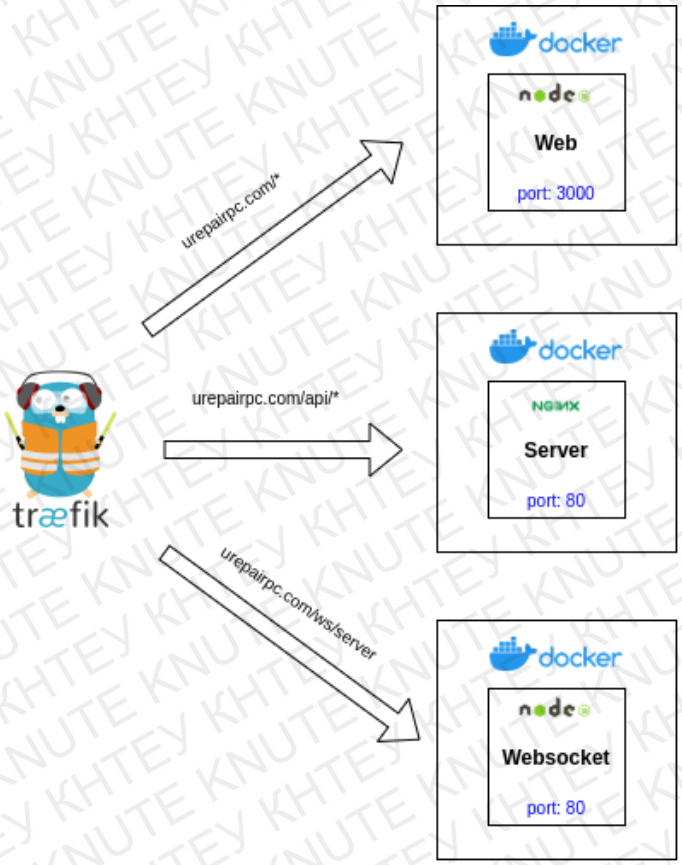

Рис. 3.8. Приклад роботи через Traefik Proxy

При цьому підході не має необхідності налаштовувати Proxy або дописувати сервер, щоб обробляти запити через CORS безпеку.

3.3. Розробка клієнтської частини

Вся клієнтська частина розробляється за допомогою фреймворка Vue, яка дозволяє досягнути SPA (Single Page Application) – це обслуговування

чудового користувацького досвіду без перезавантаження сторінок та без зайвого часу очікування. Односторінкова програма працює всередині браузера, і це лише одна веб-сторінка, яку ви відвідуєте, а потім завантажує весь інший вміст за допомогою JavaScript. Найпоширенішими SPA-зонами є: Gmail, Google Maps, Facebook, GitHub та багато інших.

SPA запитує дані та розмітку окремо та робить сторінки безпосередньо у браузері. Односторінкові сайти допомагають утримувати користувача в одному, зручному веб-просторі, де вміст представляється користувачеві просто, легко та ефективно.

При першому заході на сайт, якщо користувач не авторизований, то він автоматично переходить до сторінки «Авторизація». Тільки з цієї сторінки користувач має змогу потрапити до системи та отримувати які-небудь дані з сервера.

Рис. 3.9. Сторінка «Авторизація»

Після введення поштової скриньки та пароля і натиснувши на «Увійти» – відправляється запит на сервер, який в свою чергу перевіряє дані з сервера, генерує JWT токен, який діє 1 час та віддає користувачу. Цей токен зберігається в локальному сховищі (localStorage), та при відправленні

наступних запитів до серверу, цей токен підкріплюється. Це ж працює і при спробі завантажити файл з сервера.

Після успішного входу в систему, користувач потрапляє до головної сторінки веб-сайту, через який він має змогу швидко перейти до потрібної сторінки. Всі блоки та посилання на панелі веб-сайту – фільтруються, тобто, якщо у користувача не вистачає прав на перегляд списку ролей, то всі посилання з сайту видаляються.

В верхній частині веб-сайту розташоване посилання на створення замовлення, яке є основним функціоналом даного проекту, тому користувач швидко знайде потрібний йому функціонал для створення нового замовлення.

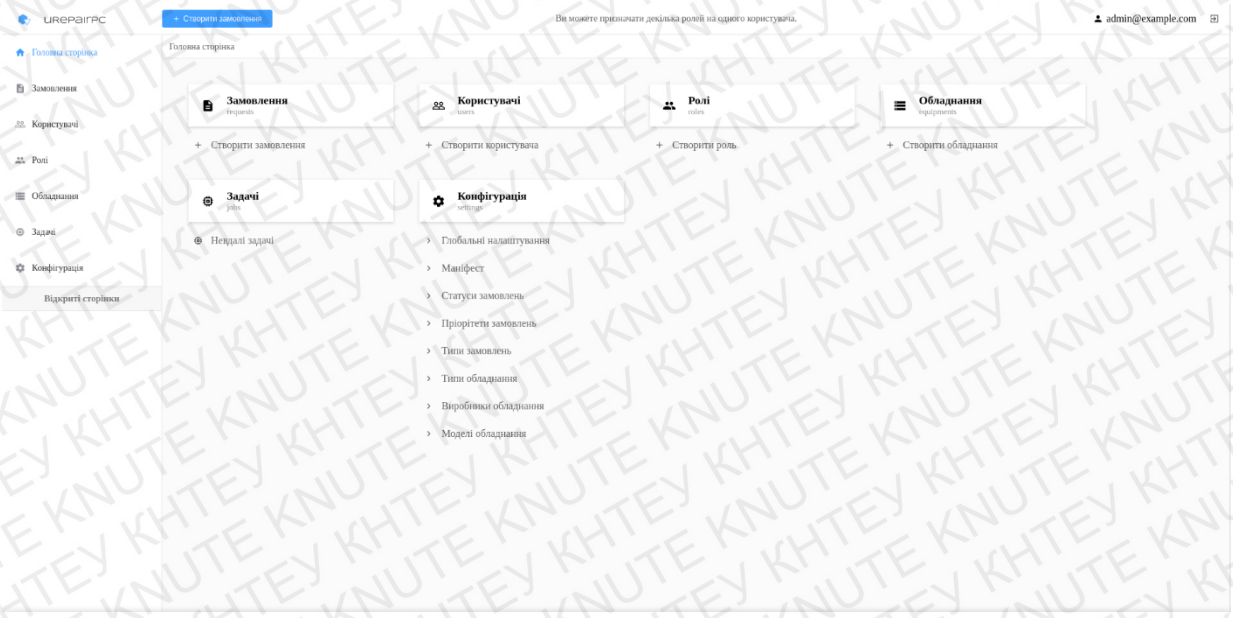

Рис. 3.10. Головна сторінка

При створенні нового замовлення, користувач заповнює інформацію, який технічний пристрій перестав працювати, де він знаходиться та інший додатковий опис.

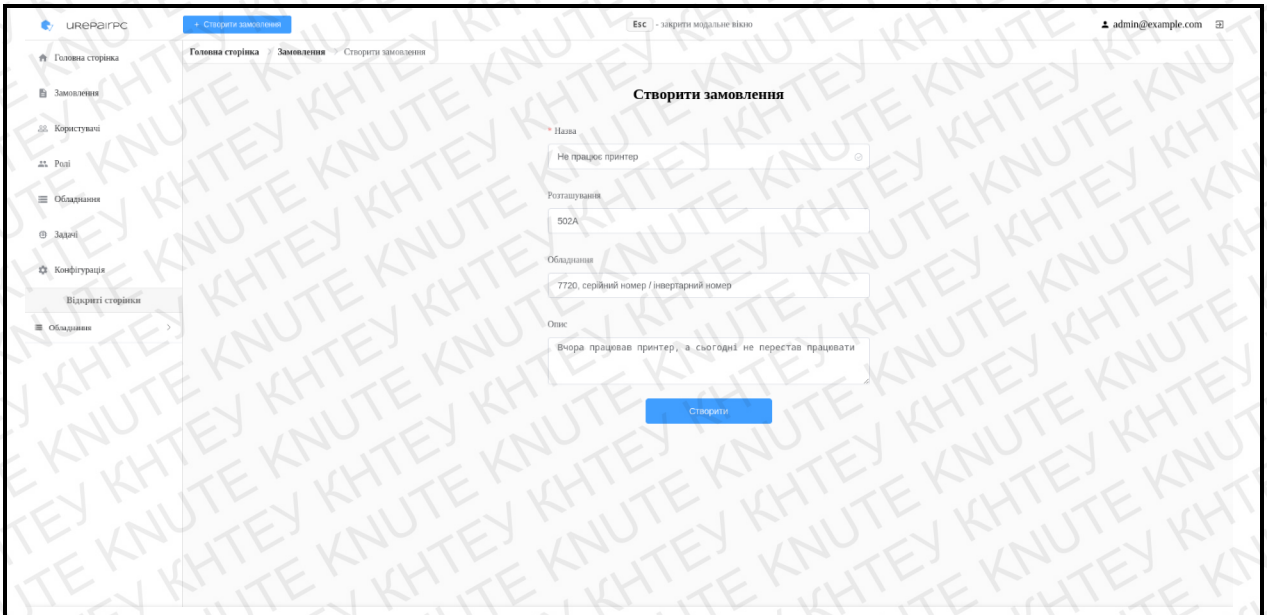

Рис. 3.11. Сторінка створення нового замовлення

Після того, як користувач створив нове замовлення, він переходить на сторінку цього замовлення, де відображається вся інформація по цьому замовленню.

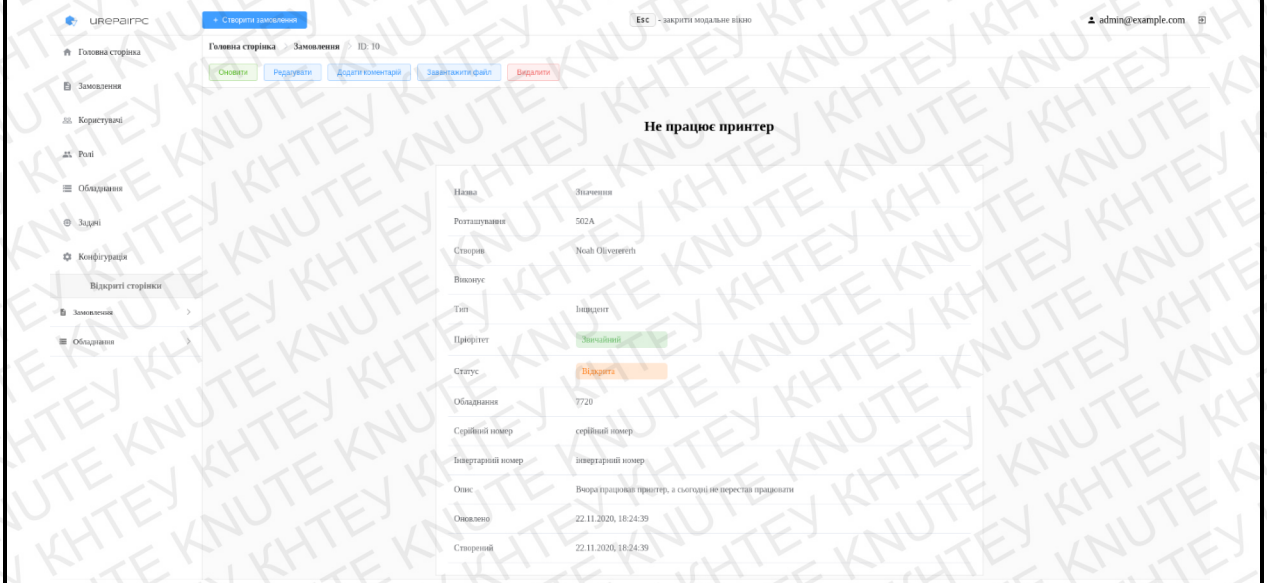

Рис. 3.12. Сторінка одного замовлення

3.4. Розробка серверної частини

За допомогою мови програмування PHP та Laravel фреймворку, вже налаштований MVC шаблон дизайну. Кожен запит до сервера виконує

певний набір функцій, розглянемо виконання запиту на створення нового замовлення в Рис. 3.13.

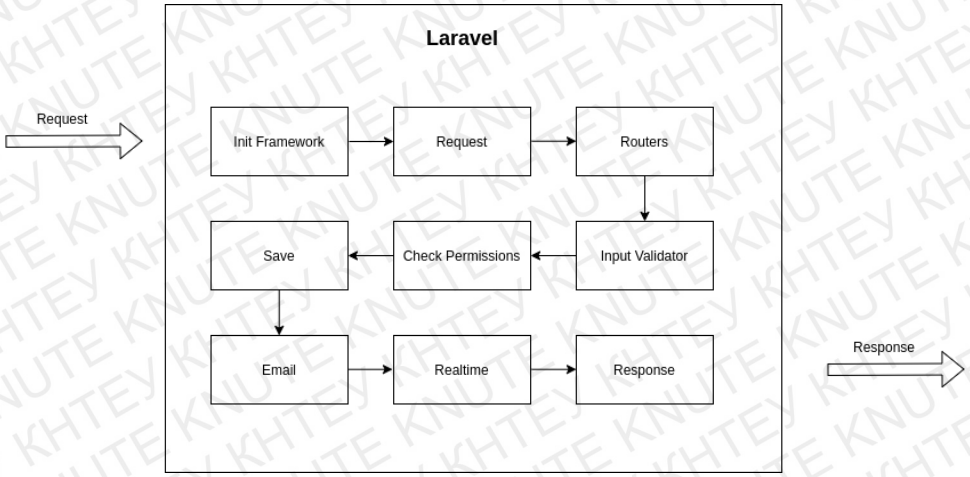

Рис. 3.13. Життєвий цикл створення нового замовлення

- Користувач відправляє запит на створення нового замовлення.
- Ініціалізується Laravel Framework.
- Зчитується запит.
- На основі запиту підбираємо потрібний роут, який буде обробляє цей запит.
- Валідуються дані для створення нового замовлення.
- Система перевіряє права користувача, чи може він створювати.
- Зберігається новий запис в MySQL.
- Додається завдання в MySQL, який відправить повідомлення на Email (фоновий процес).
- Додається повідомлення в RabbitMQ, який оновить дані всім користувачам в реальному часі (websocket).
- Відправляється відповідь користувачу.

Лістинг коду для файла, де виконується створення нового замовлення приведено в Додатку Б.

3.5. Налаштування CI/CD

Кодова база проекта постійно зростає, тому необхідна автоматизувати процес перевірки на успішний білд docker-контейнерів, а також запуск тестів. Для цього використовуємо Github Actions, який дозволяє при редагування файлів — виконувати певні дії.

Для цього необхідно створити Workflows для всіх 3-х систем, які будуть виконуватися окремо, це дозволить швидше отримувати результат тестів по окремій системі, а також швидко знаходити помилки в коді та виправляти їх.

```yaml
name: Server

on: [push]

jobs:
  build:
    name: Build
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - name: Copy .env.testing to .env
        run: cp .env.testing .env
      - name: Init core containers
        run: docker-compose -f docker-compose.yml -f docker-compose.test.yml up -d traefik mysql redis rabbitmq
      - name: Run container
        run: docker-compose -f docker-compose.yml -f docker-compose.test.yml up --abort-on-container-exit --exit-code-from server --build server
```

Рис. 3.14. Налаштування Github Workflow для серверної частини

Необхідно створити новий docker-compose.yml, який буде трохи відрізнятись від розробки, так як він лише підніме docker-контейнер, виконає певні команди для виконання тестів, а потім завершить виконання операції. Код налаштувань приведено в Додатку С.

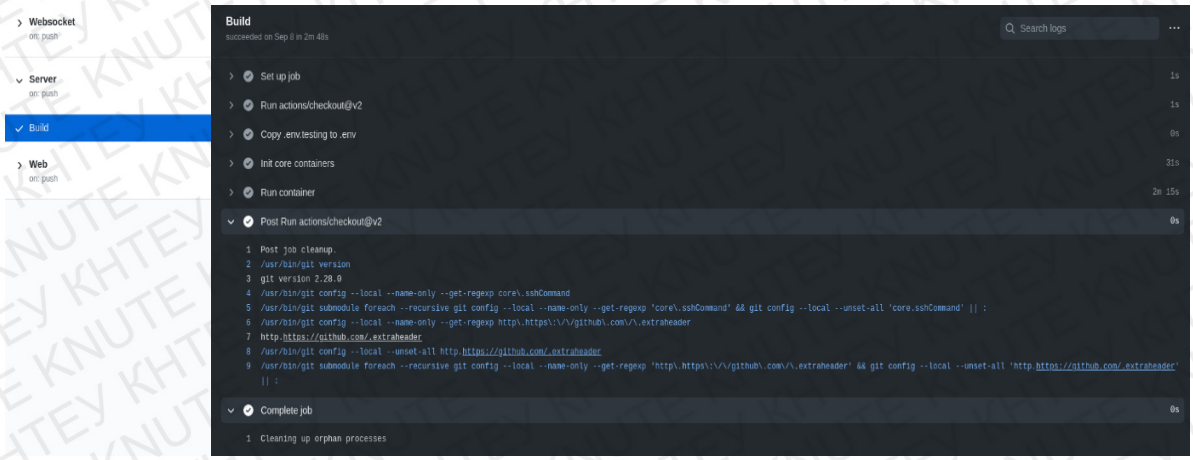

Рис. 3.15. Результат виконання Github Actions

3.6. Висновок до розділу 3

В даному розділі, були описані всі етапи розробки, починаючи від налаштування Docker-контейнерів, розробкою всіх частин проекту, закінчуючи CI/CD. Це дало змогу без особих ускладнень розгортати проект, зі всіма бібліотеками та іншими системами.

Проаналізували, як з використанням Traefik Proxy змінився підхід до розробки, що необхідно для того, щоб налаштувати цю систему, та як це працює.

Розробили клієнтську і серверну частину проекта, які взаємодіють між собою за допомогою запитів.

Налаштували CI/CD за допомогою Github Actions, а також з використанням docker-compose інструменту, щоб з кожним оновленням файлів, розробник був впевнений, що все працює правильно.

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

З проведених досліджень можна зробити наступні висновки:

- Проведено аналіз існуючих технологій та обрані технології, за допомогою яких можна вирішити технічне завдання найбільш оптимальним способом.
- Весь процес розробки був поділений на 3 частини, кожна з частин доповнює одна-одну:
  - Серверна частина — працює с запитами від користувача та надає дані з база даних.
  - Клієнтська частина — відображає інтерфейс користувача.
  - Система оновлень — надає оновлення користувачам в реальному часі.
- В процесі реалізації проекту використовувався Traefik Proxy, який змінив підхід до розробки, де використовується декілька різних систем, з різними мовами програмування, займаючи різні порти.
- Описаний принцип роботи Docker-контейнеризації та використаний цей інструмент для вирішення багатьох проблем з використанням різних ОС та версіями бібліотек.
- Обрана реляційна база даних MySQL та спроектовані таблиці згідно технічного завдання, для покриття всіх вимог.
- Серверна частина розроблена з використанням багатьох архітектурних концепцій Laravel фреймворку, що забезпечує стабільну та безпечну роботу з запитами і даними.

- Клієнтська частина використовує найновітніші технології фронтенд розробки, що дозволяє швидко розробляти інтерфейс і гнучко маніпулювати даними з серверної частини.
- Розроблений веб-сайт повністю задовольняє поставлені функціональні вимоги. Тестування продемонструвало повну відповідність функціональним вимогам та надійну роботу веб-сайта.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Laravel - The PHP Framework For Web Artisans [Електронний ресурс]. – Режим доступу: https:/laravel.com/
2. MySQL – [Електронний ресурс]. – Режим доступу: https://www.mysql.com/
3. JSON Web Tokens – jwt.io [Електронний ресурс]. – Режим доступу: https://jwt.io/
4. Docker Documentation [Електронний ресурс]. – Режим доступу: https://docs.docker.com
5. TypeScript: The starting point for learning TypeScript [Електронний ресурс]. – Режим доступу: https://www.typescriptlang.org/docs
6. Index | Node.js v14.15.1 Documentation [Електронний ресурс]. – Режим доступу: https://nodejs.org/dist/latest-v14.x/docs/api/
7. Vue.js [Електронний ресурс]. – Режим доступу: https://vuejs.org/
8. Traefik [Електронний ресурс]. – Режим доступу: https://doc.traefik.io/traefik/

ТЕХНІЧНЕ ЗАВДАННЯ

1. Загальні відомості

Назва проекту: uRepairPC.

Планові строки розробки: липень 2020 – вересень 2020.

Потенційні користувачі: користувачі стаціонарних ПК або смартфона.

Призначення програмного рішення: надати можливість користувачу створювати замовлення на технічне обслуговування пристрою за допомогою веб-сайту.

Мета створення програмного рішення: полегшити процес створення замовлення на технічне обслуговування пристрою для працівників КНТЕУ.

2. Структура проекту

Проект складається з декількох основних частин:

- Серверна частина (backend): виступає в якості API, яка обробляє запити від користувачів, взаємодіє з реляційною базою даних MySQL та відправляє повідомлення до системи оновлень в реальному часі (websocket).
- Клієнтська частина (frontend): відповідає виключно за відображення контенту користувача.
- Система для оновлень в реальному часі (websocket): відправляє користувачам оновлення, які відбуваються з даними.
- Інші сервіси, які необхідні для повноцінної роботи проекту.

3. Функціональні вимоги до додатку

Сайт повинен бути доступним тільки для зареєстрованих користувачам. Автентифікацію необхідно реалізувати по логіну та паролю, взаємодіяти за

допомогою JSON Web Token (JWT) технології або сесій браузера, та зберігати в куках, якщо це сесії, або ж в локальному сховищі — для JWT.

Спроектувати сервер виключно для роботи з AJAX запитами, які будуть надходити від клієнтської частини, та з можливістю розширювати можливості в майбутньому, для взаємодії з кожним ресурсом на сайті, необхідно реалізувати різні доступи, які можливо редагувати в налаштуваннях.

Спроектувати реляційну базу даних MySQL або PostgreSQL, яка буде вмістити в собі всю необхідну інформацію по замовленням, списком обладнання та іншими даними, які необхідні для повноцінного функціонування сайту. Необхідна можливість кешування інформацію для швидкого доступу до певної інформацію, кеш може зберігатися у вигляді файлів або у Redis.

На сайті повинен бути реалізован пошук, фільтрація, пагінація та інший додатковий функціонал, з яким зручно взаємодіяти та оброблювати інформацію, яка виводиться на сайті.

Розробити функціонал в серверній та клієнтській частині, за яким можливо завантажувати та зберігати файли, які в подальшому можливо підкріплювати до замовлення або обладнання. Файли повинні містити інформацію в БД та доступ до завантаження тільки після проходження авторизації.

Клієнтська частина повинна бути реалізована за допомогою фреймворків — Vue.js, React або Angular, з роутінгом і глобальним сховищем. Для збору клієнтської частини, необхідно обрати одну з наступних технологій — Webpack або esbuild.

Для підтримання оновлень в реальному часі, потрібно розробити систему на основі websocket технології, яка буде працювати на Node.js з мовою програмування Typescript.

Кожен сервіс та частина проекту повинен бути запущений на основі Docker технології, і використаний як при локальній розробці, та й при роботі в продакшн режимі.

ТЕСТУВАННЯ ВЕБ-САЙТА

З метою забезпечення якості розробленого веб-сайту, необхідно провести тестування всіх частин системи. Для unit-тестування використовуються такі інструменти:

- Jest: для Javascript/Typescript мови програмування
- PHPUnit: для PHP мови програмування

На основі вимог до веб-сайту (технічного завдання), було протестована обов'язковий функціонал.

Це дуже важливі і складні функції, які можуть приймати будь-які дані, такі як: string, number, boolean, function та ін. І цей функціонал перевіряє, чи є у користувача той чи інший доступ, в результаті якого відображається або ховається певний функціонал на сайті.

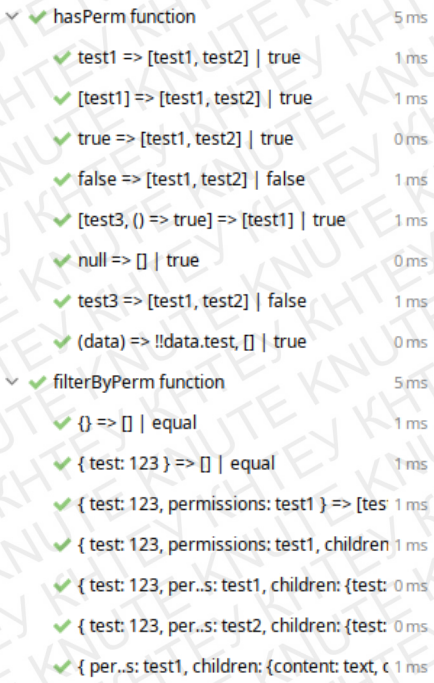

Рис. Т.1. Результат виконання unit-тестів

					КНТЕУ 121 06-20.МР		
Зм.	*Аркуш*	*№ докум.*	*Підпис*	*Дата*			
Зав. каф.		Криворучко О.В.		24.01.20	Розробка програмного забезпечення управління відділом технічного обслуговування КНТЕУ	*Стадія*	*Аркуш*	*Аркушів*
Керівник		Палагута К.О.		24.01.20		*ПМТ*	*48*	*44*
Гарант		Криворучко О.В.		24.01.20		Факультет інформаційних технологій 2м курс, 6 група		
Розробив		Хрущ О.Г.		24.01.20	*Програма та методика тестування*			

ДОДАТКИ

Додаток А

Конфігурація для docker-compose.yml файла

```
version: '3'

services:
  traefik:
    image: traefik:v2.2
    container_name: u_traefik
    restart: always
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock:ro
    command:
      - --log.level=INFO
      - --metrics.prometheus=true
      - --api.insecure=true
      - --providers.docker=true
      - --entrypoints.web.address=:80
    ports:
      - 80:80
      - 8080:8080

  server:
    build: .
    container_name: u_server
    restart: always
    volumes:
      - ./:${DOCKER_CONTAINER_PATH}

  web:
    container_name: u_web
```

```
  restart: always
  labels:
   - traefik.http.routers.web.rule=Host(`${DOCKER_DOMAIN}`)
   - traefik.http.routers.web.entrypoints=web

 websocket:
  container_name: u_websocket
  restart: always
  depends_on:
   - rabbitmq
  labels:
   - traefik.http.routers.websocket.rule=Host(`${DOCKER_DOMAIN}`) && Path(`/ws/server/`)
   - traefik.http.routers.websocket.entrypoints=web

 php-worker:
  build: .docker/php-worker
  container_name: u_php_worker
  restart: always
  volumes:
   - ./:${DOCKER_CONTAINER_PATH}
   - ./.docker/php-worker/supervisord.d:/etc/supervisord.d

 nginx:
  build: ./.docker/nginx
  container_name: u_nginx
  restart: always
  volumes:
   - ./:${DOCKER_CONTAINER_PATH}
   - ./storage/app/public:${DOCKER_CONTAINER_PATH}/public/storage/
   - ./.docker/nginx/nginx.conf:/etc/nginx/nginx.conf
```

```
    - ./.docker/nginx/sites/:/etc/nginx/sites-available
    - ./.docker/nginx/conf.d/:/etc/nginx/conf.d
    depends_on:
    - server
    labels:
    - traefik.http.routers.nginx.rule=Host(`${DOCKER_DOMAIN}`) && PathPrefix(`/api/`, `/storage/`,
`/robots.txt`)
    - traefik.http.routers.nginx.entrypoints=web

  mysql:
    build: ./.docker/mysql
    container_name: u_mysql
    restart: always
    environment:
    - MYSQL_DATABASE=${DB_DATABASE}
    - MYSQL_USER=${DB_USERNAME}
    - MYSQL_PASSWORD=${DB_PASSWORD}
    - MYSQL_ROOT_PASSWORD=${DB_ROOT_PASSWORD}
    volumes:
    - ${DOCKER_DATA_PATH}/mysql:/var/lib/mysql
    ports:
    - ${DB_PORT}:3306

  redis:
    build: ./.docker/redis
    container_name: u_redis
    restart: always
    volumes:
    - ${DOCKER_DATA_PATH}/redis:/data
    ports:
    - ${REDIS_PORT}:6379
```

```
rabbitmq:
  build: ./.docker/rabbitmq
  container_name: u_rabbitmq
  restart: always
  volumes:
   - ${DOCKER_DATA_PATH}/rabbitmq:/var/lib/rabbitmq
  labels:
   - traefik.http.routers.rabbitmq.rule=Host(`rabbitmq.localhost`)
   - traefik.http.services.rabbitmq.loadbalancer.server.port=15672
  ports:
   - ${RABBITMQ_PORT}:5672

prometheus:
  image: prom/prometheus
  container_name: u_prometheus
  restart: always
  user: "0"
  labels:
   - traefik.http.routers.prometheus.rule=Host(`prometheus.localhost`)

grafana:
  image: grafana/grafana:7.0.5
  container_name: u_grafana
  restart: always
  depends_on:
   - prometheus
  user: "0"
  volumes:
   - ${DOCKER_DATA_PATH}/grafana:/var/lib/grafana
  labels:
```

```
 - traefik.http.routers.grafana.rule=Host(`grafana.localhost`)

phpmyadmin:
 image: phpmyadmin/phpmyadmin:5.0
 container_name: u_phpmyadmin
 restart: always
 environment:
  - PMA_HOST=mysql
  - MYSQL_USER=${DB_USERNAME}
  - MYSQL_PASSWORD=${DB_PASSWORD}
  - MYSQL_ROOT_PASSWORD=${DB_ROOT_PASSWORD}
 labels:
  - traefik.http.routers.phpmyadmin.rule=Host(`phpmyadmin.localhost`)

netdata:
 image: netdata/netdata:v1.23.0
 container_name: u_netdata
 restart: always
 cap_add:
  - SYS_PTRACE
 labels:
  - traefik.http.routers.netdata.rule=Host(`netdata.localhost`)
```

Класс RequestController.php

```
<?php

namespace App\Http\Controllers;

use App\Enums\Perm;
use App\Models\Equipment;
use App\Models\RequestType;
use App\Models\RequestStatus;
use App\Models\RequestPriority;
use App\Realtime\Requests\EJoin;
use App\Http\Helpers\FilesHelper;
use Illuminate\Http\JsonResponse;
use App\Realtime\Requests\ECreate;
use App\Realtime\Requests\EUpdate;
use Illuminate\Support\Facades\Gate;
use App\Http\Requests\RequestRequest;
use App\Models\Request as RequestModel;
use Illuminate\Auth\Access\AuthorizationException;

class RequestController extends Controller
{
   public function permissions(): array
   {
      return [
         'index' => [Perm::REQUESTS_VIEW_OWN, Perm::REQUESTS_VIEW_ALL,
Perm::REQUESTS_VIEW_ASSIGN],
         'show' => [Perm::REQUESTS_VIEW_OWN, Perm::REQUESTS_VIEW_ALL,
Perm::REQUESTS_VIEW_ASSIGN],
         'store' => Perm::REQUESTS_CREATE,
         'update' => [Perm::REQUESTS_EDIT_OWN, Perm::REQUESTS_EDIT_ALL,
Perm::REQUESTS_EDIT_ASSIGN],
         'destroy' => [Perm::REQUESTS_DELETE_OWN, Perm::REQUESTS_DELETE_ALL,
Perm::REQUESTS_DELETE_ASSIGN],
      ];
   }

   /**
    * Display a listing of the resource.
    * @param  RequestRequest  $request
```

```
     * @return JsonResponse
     */
    public function index(RequestRequest $request): JsonResponse
    {
        $query = RequestModel::querySelectJoins();

        if ($request->has('search') && $request->exists('columns')) {
            foreach ($request->columns as $column) {
                $query->orWhere(RequestModel::SEARCH_RELATIONSHIP[$column] ?? $column, 'LIKE', $request->search.'%');
            }
        }

        if ($request->has('sortColumn')) {
            $query->orderBy(
                RequestModel::SORT_RELATIONSHIP[$request->sortColumn] ?? $request->sortColumn,
                $request->sortOrder === 'descending' ? 'desc' : 'asc'
            );
        }

        if ($request->priority_id) {
            $query->where('requests.priority_id', $request->priority_id);
        }
        if ($request->status_id) {
            $query->where('requests.status_id', $request->status_id);
        }
        if ($request->type_id) {
            $query->where('requests.type_id', $request->type_id);
        }

        $list = $query->filterByUser(auth()->user())->paginate();
        EJoin::dispatchAfterResponse(...$list->items());

        return response()->json($list);
    }

    /**
     * Store a newly created resource in storage.
     * @param  RequestRequest  $request
     * @return JsonResponse
```

```
*/
public function store(RequestRequest $request): JsonResponse
{
    $model = new RequestModel($request->validated());
    $model->user_id = auth()->id();
    $model->type_id = RequestType::getDefaultValue()->id;
    $model->priority_id = RequestPriority::getDefaultValue()->id;
    $model->status_id = RequestStatus::getDefaultValue()->id;

    if ($request->has('equipment_id')) {
        if (perm(Perm::EQUIPMENTS_VIEW_ALL)) {
            $model->equipment_id = $request->equipment_id;
        } else {
            $equipment = Equipment::findOrFail($request->equipment_id);
            if (Gate::allows('owner', $equipment)) {
                $model->equipment_id = $request->equipment_id;
            }
        }
    }

    $model->save();

    $model = RequestModel::querySelectJoins()->findOrFail($model->id);
    ECreate::dispatchAfterResponse($model);

    return response()->json([
        'message' => __('app.requests.store'),
        'request' => $model,
    ]);
}

/**
 * Display the specified resource.
 * @param  RequestModel  $requestModel
 * @return JsonResponse
 * @throws AuthorizationException
 */
public function show(RequestModel $requestModel): JsonResponse
{
```

```
        $this->authorize('show', $requestModel);

        EJoin::dispatchAfterResponse($requestModel);

        return response()->json([
            'message' => __('app.requests.show'),
            'request' => $requestModel,
        ]);
    }

    /**
     * @param  RequestRequest  $request
     * @param  RequestModel  $model
     * @return JsonResponse
     * @throws AuthorizationException
     */
    public function update(RequestRequest $request, RequestModel $model): JsonResponse
    {
        $this->authorize('update', $model);

        $model->fill($request->validated());

        if ($request->has('equipment_id')) {
            if (perm(Perm::EQUIPMENTS_VIEW_ALL)) {
                $model->equipment_id = $request->equipment_id;
            } else {
                $equipment = Equipment::findOrFail($request->equipment_id);
                if (Gate::allows('owner', $equipment)) {
                    $model->equipment_id = $request->equipment_id;
                }
            }
        }

        if ($request->has('assign_id') && perm(Perm::REQUESTS_EDIT_ALL)) {
            $model->assign_id = $request->assign_id;
        }

        if (perm(Perm::REQUESTS_CONFIG_VIEW_ALL)) {
            if ($request->has('type_id')) {
                $model->type_id = $request->type_id;
```

```
        }
        if ($request->has('priority_id')) {
            $model->priority_id = $request->priority_id;
        }
        if ($request->has('status_id')) {
            $model->status_id = $request->status_id;
        }
        }

        $model->save();

        $model = RequestModel::querySelectJoins()->findOrFail($model->id);
        EUpdate::dispatchAfterResponse($model->id, $model);

        return response()->json([
            'message' => __('app.requests.update'),
            'request' => $model,
        ]);
    }

    /**
     * @param  RequestRequest  $request
     * @param  RequestModel  $requestModel
     * @return JsonResponse
     */
    public function destroy(RequestRequest $request, RequestModel $requestModel): JsonResponse
    {
        $this->authorize('delete', $requestModel);

        if ($request->files_delete && ! FilesHelper::delete($requestModel->files)) {
            return response()->json(['message' => __('app.files.files_not_deleted')], 422);
        }

        $requestModel->delete();

        return response()->json([
            'message' => __('app.requests.destroy'),
        ]);
    }
}
```

Додаток С

Конфігурація для docker-compose.test.yml файла

```
version: '3'

services:
 server:
  build:
   context: .
   dockerfile: test.Dockerfile

 web:
  build:
   context: ./web
   dockerfile: test.Dockerfile

 websocket:
  build:
   context: ./websocket
   dockerfile: test.Dockerfile
```