Лабораторна 1

Посилання на курс https://learn.ztu.edu.ua/course/view.php?id=2950

Копія Лабораторної з Льорну

ВИБИРАЄМО ІНДИВІДУАЛЬНУ ПІДПИСКУ, СТАВИМО НА ПЕРСОНАЛЬНИЙ, НЕ КОРПОРАТИВНИЙ КОМПЮТЕР.

Лабораторна робота №1. Встановлення Docker та ознайомлення з базовими командами

- Встановіть Docker на вашу операційну систему. Запустіть Docker.

Знайдіть в документації Docker та виконайте такі команди:

- Перевірте статус встановленого Docker

Завдання 1. Завантажте та запустіть образ hello-world (однією командою)

- Отримайте список образів Docker на вашій машині
- Отримайте список діючих контейнерів на вашій машині
- Отримайте список всіх контейнерів

Завдання 2. Завантажте і запустіть образ веб-сервера nginx на порту 1234. Протестуйте роботу контейнера в браузері (127.0.0.1:1234)

- Здійсніть зупинку контейнера nginx
- Під час захисту роботи потрібно продемонструвати дані команди, а також видалення контейнерів та образів.

Джерела

https://docs.docker.com/

Підготовка до роботи

Перед початком переконайтеся, що ваш комп'ютер відповідає **системним вимогам Docker** (64-розрядна операційна система, підтримка віртуалізації).

Хід роботи

1. Встановлення Docker

Першим кроком є встановлення Docker на вашу операційну систему. Залежно від вашої ОС, оберіть відповідну інструкцію:

- **Для Windows:** Завантажте **Docker Desktop** з офіційного сайту. Після завантаження запустіть інсталятор і дотримуйтесь вказівок. Можливо, доведеться увімкнути Hyper-V (для Windows 10 Pro) або встановити WSL 2 (для Windows 10 Home). Перезавантажте комп'ютер після встановлення.

- **Для macOS:** Завантажте **Docker Desktop for Mac** з офіційного сайту. Перетягніть іконку Docker до папки Applications і запустіть програму
- **Для Linux:** Встановлення залежить від дистрибутиву. Наприклад, для **Ubuntu** виконайте команди:

Опис для лінукс

Bash
Оновіть індекс пакетів
sudo apt update

Встановіть необхідні пакети
sudo apt install apt-transport-https ca-certificates curl gnupg lsb-release

Додайте офіційний GPG ключ Docker
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg

Налаштуйте репозиторій Docker
**echo **
**"deb [arch=amd64 signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu **
$(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

Встановіть Docker Engine, CLI, та Containerd
sudo apt update
sudo apt install docker-ce docker-ce-cli containerd.io

Додайте поточного користувача до групи docker (щоб не використовувати sudo)
sudo usermod -aG docker $USER

Перезавантажте оболонку або вийдіть/увійдіть

Після встановлення **запустіть Docker Desktop** (для Windows/macOS) або переконайтеся, що **служба Docker запущена** (для Linux) за допомогою команди:

Bash
sudo systemctl start docker

---

2. Базові команди Docker

Відкрийте **термінал** або **командний рядок** (PowerShell, Command Prompt, Git Bash) і виконайте наступні команди:

Перевірка версії Docker: Ця команда підтвердить, що Docker встановлено і доступно.
Bash
docker --version

- **Вивід:** `Docker version 24.0.5, build 24.0.5-0ubuntu1~22.04.1`

Перевірка статусу Docker: Переконайтеся, що служба Docker запущена.
Bash
docker info

- **Вивід:** Детальна інформація про Docker: кількість контейнерів, образів, версія тощо.

---

3. Завдання 1: Робота з образом `hello-world`

Завантаження та запуск образу `hello-world`: Ця команда автоматично завантажить (якщо його немає локально) і запустить контейнер.

Bash
docker run hello-world

- **Пояснення:** Команда `run` поєднує в собі `pull` (завантаження образу) та `create/start` (створення та запуск контейнера).
- **Вивід:** З'явиться повідомлення "Hello from Docker!". Це означає, що контейнер успішно запущено і відпрацьовано.

Отримання списку образів: Перевірте, чи образ `hello-world` з'явився у вашому локальному сховищі.
Bash
docker images

- **Вивід:** Таблиця з назвами образів, тегами, ID, розміром та датою створення. Ви повинні побачити `hello-world`.

Отримання списку діючих контейнерів: На момент перевірки `hello-world` вже зупинився, тому ця команда нічого не покаже.
Bash
docker ps
Отримання списку всіх контейнерів: Ця команда покаже всі контейнери, включно із зупиненими. Ви повинні побачити контейнер `hello-world`.
Bash
docker ps -a

- **Вивід:** Таблиця з ID, назвою образу, командою, статусом та іншими деталями.

---

4. Завдання 2: Запуск веб-сервера NGINX

Завантаження та запуск образу NGINX: Запустіть контейнер з веб-сервером `nginx`, зіставивши порт 1234 на вашому хості з портом 80 контейнера (стандартний порт NGINX). Опція `-d` (detached) запускає контейнер у фоновому режимі.
Bash
docker run -d -p 1234:80 nginx

- **Пояснення:** * `-d`: Запускає контейнер у фоновому режимі.

  `-p 1234:80`: **Мапінг портів**. Перенаправляє трафік з порту `1234` вашого комп'ютера на порт `80` всередині контейнера.

- **Тестування роботи:** Відкрийте браузер і перейдіть за адресою **`http://127.0.0.1:1234`**. Ви повинні побачити сторінку "Welcome to nginx!".

Перевірка діючих контейнерів:
Bash
docker ps -a

- **Вивід:** Ви побачите запущений контейнер `nginx` з його ID, назвою, станом та мапінгом портів.

Зупинка контейнера: Використовуйте ID контейнера, щоб його зупинити.

Потім виконайте команду, замінивши <container_id> на реальний ID
docker stop <container_id1> <container_id2>

-

---

5. Видалення контейнерів та образів

На захисті лабораторної роботи важливо продемонструвати вміння "прибирати" за собою.

Видалення зупинених контейнерів:
Bash
docker rm <container_id1> <container_id2>
Наприклад: docker rm 1a2b3c4d5e6f 1a2b3c4d5e6f

-
    - **Порада:** Щоб видалити всі зупинені контейнери, скористайтеся командою `docker container prune`.

Видалення образів: Перед видаленням образу переконайтеся, що немає запущених контейнерів, які його використовують.
Bash
docker rmi <image_name>
Наприклад: docker rmi nginx

-
    - **Порада:** Щоб видалити всі невикористані образи, скористайтеся командою `docker image prune`.

Захист роботи

Під час захисту продемонструйте викладачу:

1. Успішний запуск Docker.
2. Виконання команди `docker run hello-world` та показ виводу.
3. Список образів (`docker images`) та контейнерів (`docker ps -a`).
4. Запуск контейнера NGINX (`docker run -d -p 1234:80 nginx`) і його роботу в браузері.

5. Зупинку та видалення контейнера.
6. Видалення образу.

Лабораторна робота №2. Створення і запуск образа Docker

План

- Створення додатку на NodeJS (API сервіс)
- Підготовка і побудова образу для API сервісу
- Запуск API сервісу

Хід роботи

1. Створіть папку проекту ***/home/{user}/HelloDocker***
2. В папці проекту створіть папку для API-додатку***/api***
3. В папці API створіть проект на NodeJS з використанням express:
    a. Виконавчий файл src/index.js

```
const express = require("express")
const app = express()

app.get("/test", (req, res) => {
    res.send("Server is working correctly")
})

app.listen(8080, () => {
    console.log("API-service is working")
})
```

    b. В package.json задайте запуск в режимі production:

```
"scripts": {
  "start": "node src/index.js"
```

    c. Протестуйте в браузері, запустивши команду `npm run start`
    d. Зупиніть додаток

4. Підготовте Dockerfile для створення образу API-сервісу. Для цього в папці ***api*** створіть файл Dockerfile і задайте йому 7 директив:
    a. Базовий образ Node (alpine-версія)
    b. Робоча папка для файлів проекту /usr/src/app
    c. Копіюємо файли package.json та package-lock.json в образ
    d. Встановлюємо залежності
    e. Копіюємо вихідний код
    f. Повідомте Docker про порт 8080, на якому прослуховується ваш сервіс
    g. Проінформуйте про команду запуску сервісу (npm run start)
5. Створіть файл *.dockerignore* в тій же папці, що і Dockerfile, та запишіть в нього назви файлів і папок, що не будуть копіюватись в образ, а саме *node_modules*, *npm-debug.log*
6. Побудуйте образ з Dockerfile та здійсніть запуск додатку в контейнері

https://gitlab.com/web-systems-docker/docker-nodejs-app-lab1/

Лабораторна робота №2: Створення і запуск образа Docker

Мета роботи

Навчитися створювати власний образ Docker для веб-сервісу на NodeJS, використовуючи файл **`Dockerfile`**, а також запускати його як контейнер.

Хід роботи

1. Створення додатку на NodeJS за допомогою клонування репозиторію
сторити папку ***HelloDocker*** і в ній склонувати репозиторій

Bash

git clone https://gitlab.com/web-systems-docker/docker-nodejs-app-lab1/

Перейменувати сколений репозиторій на api та заходимо всередину

Bash
cd docker-nodejs-app-lab1
npm i
npm run start

Протестуйте додаток: Запустіть додаток, щоб переконатися, що він працює на 3001 порті, після чого замініть порт на 8080
Bash

> Відкрийте браузер і перейдіть за адресою `http://localhost:3001`. Ви повинні побачити повідомлення `Hello from Dockerized Node.js App!`. Після цього зупиніть додаток, натиснувши `Ctrl+C` у терміналі.

---

2. Підготовка образу Docker

Тепер створимо інструкції для Docker, які дозволять побудувати образ для нашого сервісу.

Створіть файл `.dockerignore`: У папці `~/HelloDocker/api` створіть файл `.dockerignore` і додайте до нього наступний вміст. Це запобігатиме копіюванню непотрібних файлів і папок, що прискорить процес побудови та зменшить розмір образу.

Bash
touch .dockerignore

Вміст файлу `.dockerignore`:

node_modules

npm-debug.log

Створіть `Dockerfile`: У тій самій папці `~/HelloDocker/api` створіть файл `Dockerfile` без розширення.
Bash
touch Dockerfile

Вміст файлу `Dockerfile`:
Dockerfile
Використовуємо офіційний образ Node.js як базовий

FROM node:18-alpine

Встановлюємо робочу директорію в контейнері

WORKDIR /usr/src/app

Копіюємо package.json та package-lock.json для встановлення залежностей

COPY package*.json ./

Встановлюємо залежності, використовуючи npm

RUN npm install

Копіюємо всі файли вихідного коду проекту в робочу директорію

COPY . .

Повідомляємо, що контейнер прослуховує порт 8080

EXPOSE 8080

Визначаємо команду для запуску додатку

CMD ["npm", "run", "start"]

---

3. Побудова та запуск образу

Побудуйте образ: Переконайтеся, що ви знаходитесь у папці `~/HelloDocker/api` і запустіть команду. Не забудьте про крапку `.` в кінці.
Bash
docker build -t hello-node-app-<NAME> .

- `-t` **hello-node-app-<NAME>** : Присвоює ім'я **hello-node-app-<NAME>** нашому образу, де **-<NAME> ваше імя та фамілія на англ мові**

  наприклад hello-node-app-ivan-borysenko

- : Вказує Docker, що він повинен шукати `Dockerfile` та інші файли для збірки в поточній директорії.

Перевірте, що образ створено:
Bash
docker images

Ви побачите новий образ **hello-node-app-<NAME>** у списку.

Запустіть контейнер: Запустіть контейнер у фоновому режимі, зіставивши порт 8080 вашого комп'ютера з портом 8080 всередині контейнера.
Bash
docker run -p 8080:8080 -d hello-node-app-<NAME>
(без крапки вкінці)

- `-p 8080:8080`: Мапінг портів.
- `-d`: Запускає контейнер у фоновому режимі (detached mode).

Перевірте, що контейнер запущено:
Bash
docker ps

Ви побачите запущений контейнер з іменем **hello-node-app-<NAME>** .

Протестуйте сервіс: Знову відкрийте браузер і перейдіть за адресою `http://localhost:8080`. Ви побачите те саме вітальне повідомлення, але цього разу воно буде згенероване з контейнера Docker.

Зупинка та видалення: Після тестування зупиніть та видаліть контейнер, а потім і образ. Це важливий крок для очищення вашої системи.

Зупиніть контейнер:
Bash
Спочатку знайдіть ID контейнера за допомогою 'docker ps'

docker stop <container_id>

Видаліть контейнер:
Bash
docker rm <container_id>

Видаліть образ:
Bash
docker rmi **hello-node-app-<NAME>**

Лабораторна робота 3

Практична робота (необов'язково). Використання Docker Compose

Мета роботи: створити маніфест docker-compose.yml та односервісний застосунок. Налаштувати і запустити API-сервіс. Створити і використати змінну оточення.

Передумови: встановлений Docker Compose. Для перевірки можна виконати команду `docker compose --version`

Порядок роботи

1. В кореневій папці проекту попередньої лабораторної роботи (*HelloDocker)* створіть файл *docker-compose.yml* з таким вмістом:

```
version: "3"
services:
 api:
   build: ./api
   command: npm start
   ports:
     - "3001:3001"
   environment:
     - PORT=3001
     - HOST=localhost
```

Даний файл *docker-compose.yml* визначає лише один веб-сервіс *api*. Сервіс використовує образ, створений із *Dockerfile* у каталозі *api*. Далі він реєструє команду *npm start* для запуску застосунку в контейнері і прив'язує контейнер і хост-машину до відкритого порту *3001* (зліва) з переадресацією на внутрішній порт *3001* (порт самого застосунку, записаний справа). Також в сервісі визначено дві змінні середовища: *PORT* і *HOST*.

2. Виконайте побудову котейнеризованого застосунку і його запуск:

`docker compose up -d`

3. Якщо потрібно перебудувати застосунок (після внесення змін в коді наприклад) і запустити, виконуємо команду:

`docker-compose up --build -d`

4. Перегляньте список працюючих контейнерів:

`docker ps`

5. Для зупинки контейнерів виконуємо команду:

```
docker compose stop
```

6. Для зупинки і видалення контейнерів виконуємо команду:

```
docker compose down
```

7. Використаємо змінні середовища *PORT і HOST*. Змінні середовища містять інформацію, що можуть використовувати виконавчі файли програм. В подальшому ми створимо змінні середовища для портів, налаштування БД, конфігурації поштового серверу тощо.

Використання змінних оточення в коді застосунку:

```
const PORT = process.env.PORT;
const HOST = process.env.HOST;
```

Отже, ми розглянули створення простої конфігурації проекту з використанням інструменту Docker Compose та використали базові команди Docker Compose для побудови образів, запуску і зупинки контейнерів. В наступній лабораторній роботі підготуємо Docker Compose для роботи з двома сервісами: серверним застосунком та базою даних.

Лабораторна робота 3. Використання Docker Compose для роботи з базою даних.

Мета роботи: створити конфігурацію Docker Compose з двома сервісами: серверний застосунок api та сервіс для баз даних api_db.

Адреса репозиторію для клонування:

https://gitlab.com/web-systems-docker/docker-nodejs-app-lab2

Хід роботи:

1. Створіть новий проект в середовищі розробки *HelloDockerRealWorld*
2. Перейдіть в директорію *HelloDockerRealWorld* та здійсніть клонування в поточну папку репозиторію з NodeJS-застосунком, що працює з БД:

```
cd HelloDockerRealWorld
git clone <repository_url> .
```

3. Огляньте архітектуру застосунку. З'ясуйте, як працюють модулі, модель, маршрутизатор, як отримується конфігурація із змінних середовища.
4. В кореневій папці проекту створимо файл-маніфест *docker-compose.yml*:

```
version: "3"
services:
 api:
   build: ./api
   command: npm start
   ports:
     - "3001:3001"
   environment:
     - PORT=3001
     - HOST=localhost
     - MONGO_URL=mongodb://api_db:27017/api
   depends_on:
     - api_db
 api_db:
   image: mongo:latest
```

Даний файл *docker-compose.yml* визначає 2 сервіси: веб-сервіс *api* та сервіс БД *api_db*. Сервіс *api_db* не використовує власний Dockerfile, натомість його образ будується

одразу з образу *mongo:latest*. Для сервісу api визначили змінну середовища *MONGO_URL* з рядком з'єднання з БД.

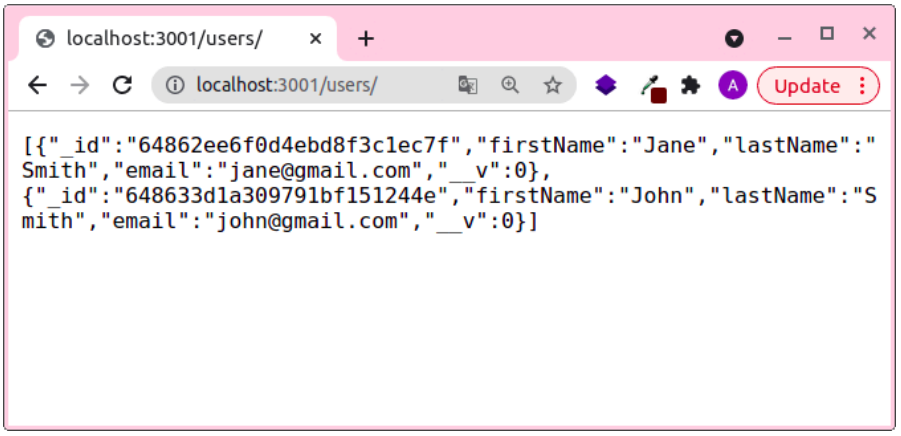
5. Виконайте побудову котейнеризованого застосунку і його запуск:

```
docker-compose up --build -d
```

6. Для виконання POST-запиту додавання нового користувача в БД можна скористатись *Postman* або утилітою *curl*:

```
curl -X POST http://localhost:3001/users -H "Content-Type:
application/json" -d '{"firstName": "Jane", "lastName": "Smith", "email":
"jane@gmail.com"}'
```

7. Виконайте декілька разів запит на додавання різних користувачів і протестуйте в браузері: *http://localhost:3001/users*



Лабораторна робота 4. Використання Docker томів.

(детальні інструкція в репозиторію https://github.com/victorchei/ztu-docker-lr-4)

Мета роботи: навчитись створювати і використовувати Docker-томи

Завдання:

- створити іменований том в багатоконтейнерній Docker-системі для постійного зберігання даних
- розділити конфігурацію запуску і роботи додатку для робочого середовища (Production Mode) і середовища розробки (Development Mode)

Передумови: дана робота є продовженням лабораторної роботи 3, тому завдання виконуємо в тому ж проекті.

Хід роботи:

1. Зупиніть контейнери, видаліть контейнер *api_db* та перезапустіть застосунок повторно. Переконайтесь, що дані в БД не збереглися.
2. Створіть *іменований том* з іменем *mongodb_api*, для постійного зберігання даних. Перебудуйте застосунок. Переконайтесь, що видалення контейнеру *api_db* не впливає на зберігання даних.
3. Створіть *конфігурацію для середовища розробки* (файл docker-compose.dev.yml):
    - Для сервісу *api* задайте команду запуску *npm run dev*

- Для сервісу *api* створіть *рядковий неіменований том* для переадресації з робочої папки контейнера на папку з вихідним кодом

4. В Dockerfile додайте директиву для встановлення *nodemon* та в *package.json* додайте скрипт для команди *npm run dev*
5. Здійснимо запуск контейнерів в середовищі розробки, враховуючи що потрібно використати два файла *docker-compose.yml* та поверх нього *docker-compose-dev.yml.*
6. Переконайтесь, що при роботі додатку в середовищі розробки запускатись будуть скрипти з директорії, на яку веде симлінк.

Хід перевірки виконаної роботи:

Без томів:

- запустити застосунок, додати дані в БД, зупинити застосунок;
- видалити контейнер;
- знову запустити застосунок, переконатись що БД порожня.

З томом для БД:

- запустити зазстосунок, додати дані в БД, зупинити застосунок;
- видалити контейнер;
- знову запустити застосунок, переконатись що БД не порожня.

З томом для середовища розробки:

- запустити застосунок в середовищі розробки;
- зробити незначні зміни в коді;
- переконатись, що Docker реагує на зміни в коді і перезапускає контейнер автоматично.

Лабораторна робота №5. Розробка клієнтського сервісу (Frontend)

(детальні інструкція в репозиторію https://github.com/victorchei/ztu-docker-lr-4)

Робота виконується в тому ж проєкті, що і попередня робота.

План

- Створити клієнтський сервіс з Create React App (сервіс Frontend)
- Розробити Dockerfile для клієнтського сервісу
- Здійснити конфігурацію клієнтського додатку для режиму Production
- Налаштувати Docker Compose для клієнтського сервісу в режимах Dev і Production для побудови образів та їх роботи в контейнерах

Хід роботи

1. В проекті створіть папку *frontend* поряд з *api*
2. В папку *frontend* встановіть середовище *Create React App*
3. В папці *frontend* створіть Dockerfile для побудови образу клієнтського сервісу
4. В docker-compose.yml створіть сервіс frontend і задайти для нього первинні налаштування:

```yaml
frontend:
  build: ./frontend
  container_name: realworld-docker-frontend
  command: npm run start
  ports:
    "3000:3000"
  restart: unless-stopped
  stdin_open: true
  tty: true
```

npm run start, stdin_open, tty - пізніше перенесемо в *docker-compose-dev.yml*

5. Побудуйте проект і перевірте роботу контейнерів. Виконайте запит localhost:3000 і в режимі перегляду html переконайтесь, що код не мініфікований.

В завданнях 6-10 здійснимо конфігурацію frontend-сервісу для **production режиму**

6. Здійсніть компіляцію коду *frontend*-частини. При цьому створиться папка *build* зі статичними файлами.
7. Встановіть *serve*-пакет для можливості обробки запитів з папки *build* в режимі production.
8. В docker-compose.yml змініть команду запуску сервісу frontend: *serve -s build -l 3000*
9. В Dockerfile дописати команди для компіляції статичних файлів та установки serve-пакету
10. Перейменувати Dockerfile на Dockerfile.prod. Внести зміни в docker-compose.yml про посилання на Dockerfile.prod

В завданнях 11-12 здійснимо конфігурацію frontend-сервісу для **режиму розробки**

11. Зробити копію Dockerfile.prod і перейменувати її на Dockerfile.dev. Директиви для компіляції файлів і установки serve-пакету прибрати.
12. В *docker-compose-dev.yml* створити сервіс frontend з налаштуваннями:
    a. Посилання на файл *Dockerfile.dev*
    b. command: npm start (робота контейнера для режиму розробки)
    c. Перенести команди для можливості роботи інтерактивної консолі
    d. Створити змінну оточення WATCHPACK_POLLING=true
    e. Створити рядкові томи:

```yaml
volumes:
  - './frontend:/usr/src/app'
  - '/app/node_modules'
```

Перевірка проекту в режимі production:

- Запуск `docker-compose up --build`
- HTML повинен бути мініфікованим

Перевірка проекту в режимі development:

- Запуск `docker-compose -f docker-compose.yml -f docker-compose-dev.yml up --build`
- HTML повинен бути ***не*** мініфікованим

- При внесенні змін в вихідний код клієнтської частини проекту контейнер автоматично перезапускається і зміни повинні примінятись без перевантаження сторінки

Джерела

Docker для ProductionMode:

```
FROM node:16-alpine3.15
WORKDIR /usr/src/app
COPY package*.json ./
RUN npm install
COPY . .
RUN npm run build
RUN npm install -g serve
```

Dockerfile для DevMode:

```
FROM node:16-alpine3.15
WORKDIR /usr/src/app
COPY package*.json ./
RUN npm install
RUN npm install nodemon -g
COPY . .
RUN chmod 777 /usr/src/app/node_modules
```

docker-compose.yaml:

```
frontend:
  build:
    context: frontend
    dockerfile: Dockerfile.prod
  command: serve -s build -l 3000
  ports:
    - "3000:3000"
  depends_on:
    - api
```

docker-compose.dev.yml:

```yaml
frontend:
  build:
    context: frontend
    dockerfile: Dockerfile.dev
  command: npm start
  volumes:
    - ./frontend:/usr/src/app
    - /usr/src/app/node_modules
  stdin_open: true
  tty: true
```

Репозиторій з детальним поясненням за посиланням
https://github.com/victorchei/ztu-docker-lr-4/tree/main

Заочна форма

щоб захистити лабораторні 4-5 потрібно запустити їх в режимі продакшену, і вони мають працювати (оцінка 4)

на оцінку 5 потрібно продемонструвати роботу в режимі девелопменту

- змінити щось в апі, наприклад імя помилки
- змінити залоговок в форонтенді і отримати оновлення без перезапуску докера

Лабораторна робота №6. Створення сервісу nginx. Проксірування запитів (Оноленe завдання)

Хід роботи

Інструкції до виконання дивимось в [лекції №5]

- Створіть в проекті папку *nginx*, в якій створіть файли *nginx.prod.conf* та *nginx.dev.conf*
- Створіть віртуальний хост, наприклад *websystem-docker.com*
- Створити сервіс для *nginx*, здійснити первинні налаштування докера для даного сервісу (*docker-compose.yml*) та конфігурації самого серверу (*nginx.prod.conf*)
- Використовучи *axios* та проксірування запитів, здійснити запит між клієнтською частиною та серверним додатком
- Видалити відкриті порти в yml-файлі (всі запити йдуть виключно через *nginx*)

Nginx для dev-режиму

- Створіть хост для режиму розробки, наприклад *websystem-docker.local*
- В файлі *nginx.conf.dev* пропишіть *server_name* на даний хост. Всі інші налаштування поки такі ж як і в *nginx.conf.prod*
- В yml-файлі для dev в сервісі *nginx* пропишіть *volume* на *nginx.conf.dev*
- Дозволити використання протоколу WS (web-sockets) в dev-конфігурації nginx:

```
location / {
    proxy_pass http://frontend:3000;
    # needed for sockets
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection "upgrade";
}
```

Як перевіряти

Для production-версії:

- Запити повинні здійснюватись за адресою http://websystem-<user-name>.com
- Заборонено здійснювати запити за адресами http://localhost:3000, http://localhost:3001
- Повинен працювати запит між frontend та api сервісами

Для dev-версії

- Запити повинні здійснюватись за адресою http://websystem--<user-name>.dev
- Зміни в коді повинні автоматично оновлювати відповідні компоненти сторінки в браузері

Зрозок кодової бази в попередньому репозиторіїї у вітці nginx

-