Лабораторна робота 1. Створення та розгортання застосунку

Мета

Створити VueJs застосунок та запустити його у вигляді Docker контейнера.

Теоретичні відомості

Vue — фреймворк для створення користувацьких інтерфейсів, що спирається на звичайні HTML, CSS і JavaScript та надає декларативну компонентну модель програмування. Декларативність означає, що розробник описує, який вигляд має мати інтерфейс за поточного стану даних, а не послідовність дій, якими цього вигляду досягають. Такий опис працює лише тоді, коли хтось приводить сторінку у відповідність до нього щоразу, як стан змінюється. Цю роботу бере на себе реактивність, тобто Vue відстежує зміни стану самостійно й оновлює саме ті частини сторінки, яких вони стосуються, без прямого звертання до дерева документа.

Компонент є одиницею побудови інтерфейсу. У проєктах зі збіркою його записують окремим файлом із розширенням `.vue`, який містить разом логіку, шаблон і стилі; такий файл називають однофайловим компонентом. Для розробника це зручно, але браузер такого файлу не розуміє, і саме звідси починається все інше.

Між кодом, який редагує розробник, і сторінкою, яку відкриває користувач, стоять два кроки:

- перетворення коду компонентів на файли, придатні для відображення браузером;
- розміщення цих файлів у середовищі, здатному віддавати їх за протоколом HTTP(S).

Виконати перший крок можна різними інструментами, один з яких — Node.js.

Node.js — вільне кросплатформне середовище виконання JavaScript з відкритим кодом, яке дає змогу запускати його поза браузером і створювати сервери, вебзастосунки, інструменти командного рядка та скрипти. Модель виконання в ньому подієва й асинхронна. Майже жодна функція не виконує введення-виведення безпосередньо, тому процес не простоює в очікуванні відповіді диска чи мережі, а тим часом обробляє інші події. Саме тому на Node.js будують і мережеві служби, і інструменти складання проєктів, зокрема ті, якими користується екосистема Vue.

Разом із Node.js постачається npm, менеджер пакетів, який встановлює потрібні проєктові бібліотеки і зберігає їхній перелік у файлі `package.json`. Там також описують скрипти проєкту — іменовані команди, які запускають через `npm run`, завдяки чому спосіб того, як зібрати чи запустити проєкт, перестає бути усним знанням.

Vite — інструмент збірки, який виконує перетворення файлів і працює у двох режимах:

- при розробці запускає сервер із швидким оновленням змінених модулів;
- при збірці перетворює проєкт на статичні файли, оптимізовані для промислового використання.

Після збірки застосунок Vue перестає бути проєктом і стає набором звичайних HTML, CSS і JavaScript.

Лишається другий крок — розміщення зібраних файлів, адже результатом роботи є не лише код, а й середовище, у якому він виконується. Контейнеризація вирішує це питання тим, що постачає застосунок разом із його середовищем. Docker описує середовище файлом `Dockerfile`, збирає з нього незмінний образ і запускає з образу контейнер — робочий екземпляр цього образу.

Відтворюваність тримається на тому, що всі складники образу названі точно. Тег `latest` цю умову порушує, бо позначає не конкретну версію, а поточну, і той самий `Dockerfile` через півроку збере інший застосунок. Тому в базових образах указують явний тег.

Зібраний образ можна запустити окремими командами, але кожна з них має параметри — які порти відкрити, як назвати контейнер, звідки брати збірку тощо. Docker Compose описує все це одним файлом у форматі YAML, після чого запуск зводиться до однієї команди, однакової на будь-якому комп'ютері.

Функціональні вимоги

- Застосунок відкривається за адресою http://localhost:8080 і віддається з контейнера, а не з dev-сервера Vite
- Компонент HelloWorld.vue показує дані студента
- npm run docker-run піднімає контейнер, npm run docker-stop його зупиняє

Технічні вимоги

- Образи Node і Nginx вказані з явним тегом версії, без latest
- У Dockerfile збірка й віддача статики розділені
- .dockerignore виключає node_modules і dist зі збіркового контексту

Необхідне обладнання та матеріали

- Node.js 20 або новіший, npm
- Docker Engine з плагіном Docker Compose
- Браузер
- Обліковий запис GitHub, GitLab або Bitbucket

Хід роботи

1. Підготовка середовища

Встановіть Node.js 20 або новіший, npm і Docker Engine з плагіном Docker Compose. Перевірте, що все стало:

```
node -v
```

```
npm -v
docker compose version
```

Остання команда має спрацювати саме в такому написанні. Compose існує у двох виглядах: як підкоманда `docker compose` і як окрема програма `docker-compose`. Документація Docker називає окрему програму нерекомендованою й такою, що підтримується лише задля зворотної сумісності, тому в роботі використовується підкоманда.

2. Створення проєкту

Створіть репозиторій `mjsf-mag` на GitHub, GitLab або Bitbucket і склонуйте його. Усі роботи курсу лежатимуть у ньому окремими каталогами, тому проєкт цієї роботи створюється каталогом `lab1` у корені репозиторію:

```
npm create vite@latest lab1 -- --template vue
cd lab1
npm install
npm run dev
```

Переконайтеся, що застосунок відкривається за адресою, яку вивів Vite.

3. Дані про себе

Знайдіть компонент `src/components/HelloWorld.vue` і замініть його вміст даними про себе: спеціальність, курс, група, прізвище, ім'я та по батькові.

Це робиться не для оформлення, а щоб під час перевірки було видно, чий саме контейнер запущено.

4. Образ Docker

Створіть у каталозі `lab1` файл `Dockerfile`:

```
# Стадія збірки: образ Node для складання застосунку
FROM node:22-alpine AS build-stage
WORKDIR /app
# Спершу тільки package.json і package-lock.json — щоб кеш npm
install
# не скидався від кожної правки коду
COPY package*.json ./
RUN npm ci
COPY . .
RUN npm run build

# Стадія віддачі: легкий Nginx для статичних файлів
FROM nginx:1.27-alpine AS production-stage
COPY --from=build-stage /app/dist /usr/share/nginx/html
EXPOSE 80
# Nginx має працювати на передньому плані, інакше контейнер одразу
завершиться
CMD ["nginx", "-g", "daemon off;"]
```

Обидва образи мають явний тег версії. `node:latest` тут не годиться: сенс контейнера в тому, що збірка відтворюється, а плаваючий тег цю властивість скасовує — через півроку той самий `Dockerfile` збере інший застосунок.

Поруч створіть `.dockerignore`:

```
node_modules
dist
.git
.env
*.log
```

5. Docker Compose і скрипти npm

Створіть `docker-compose.yml`:

```
services:
  vue-app:
    build: .
    ports:
      - "8080:80"
    container_name: vue-lab-1-instance
```

Додайте до секції `scripts` у `package.json` два записи:

```
{
  "scripts": {
    "docker-run": "docker compose up -d --build",
    "docker-stop": "docker compose down"
  }
}
```

Додавайте саме записи, а не всю секцію: у `package.json` уже є `dev`, `build` і `preview`. Після коми останнього запису коми бути не повинно — JSON із висячою комою не читається, і npm відмовиться працювати з проєктом.

6. Перевірка

```
npm run docker-run
```

Відкрийте `http://localhost:8080` і переконайтеся, що застосунок віддається з контейнера, а не з `npm run dev`. Зробіть знімок екрана.

Зупиніть контейнер:

```
npm run docker-stop
```

7. Публікація

Закомітьте роботу й надішліть у репозиторій:

```
git add lab1
git commit -m "Лабораторна робота 1: Vue у Docker"
git push
```

Що здається

Посилання на репозиторій `mjsf-mag` із каталогом `lab1`, у якому є `Dockerfile`, `.dockerignore`, `docker-compose.yml` і змінений `HelloWorld.vue`, а також знімок екрана з працюючим застосунком за адресою `http://localhost:8080`.

Вимоги до змісту та оформлення звіту

- Проєкт Vue, у якому компонент HelloWorld.vue містить дані студента (спеціальність, курс, група, ПІБ)
- Dockerfile і .dockerignore у корені проєкту
- docker-compose.yml і скрипти docker-run та docker-stop у package.json
- Знімок екрана з працюючим застосунком за адресою http://localhost:8080
- Посилання на репозиторій із роботою в каталозі lab1

Контрольні питання

1. Які кроки потрібно виконати перед початком роботи з Vue.js?
2. Що таке Vue CLI і як його встановити?
3. Як створити новий Vue.js проект?