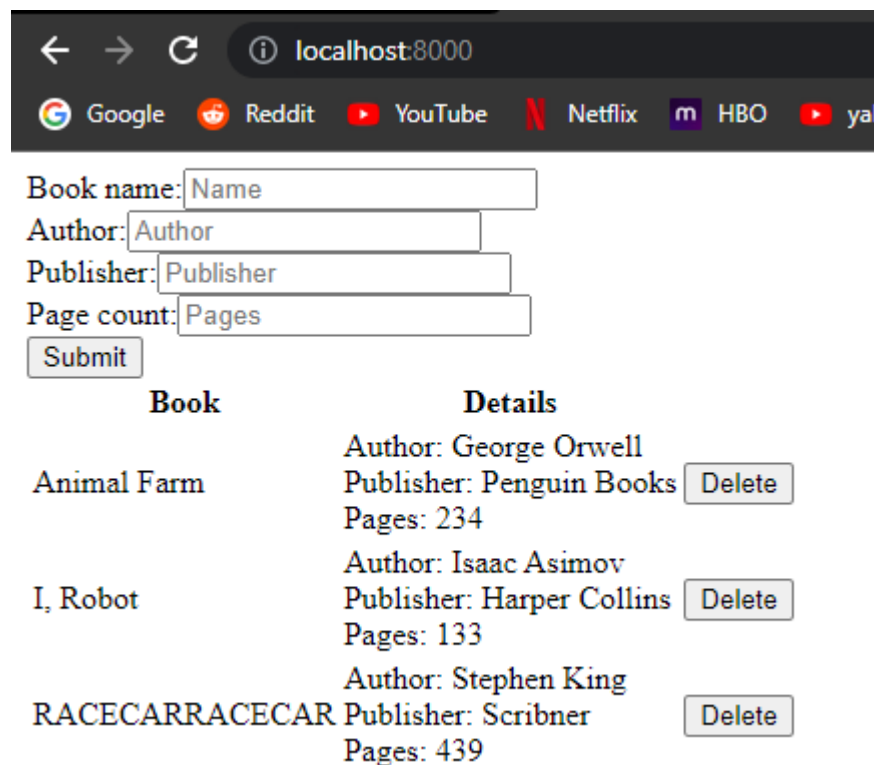


Demo Containere Docker

In acest demo voi prezenta containerizarea cu Docker a unei aplicatii simple ce permite folosirea unui frontend pentru a comunica cu un backend pentru a executa operatii pe o baza de date.

Limbaje/frameworkuri folosite:

- frontend: Vue.js
- backend: Flask
- db: MongoDB



Book	Details
Animal Farm	Author: George Orwell Publisher: Penguin Books Pages: 234 <button>Delete</button>
I, Robot	Author: Isaac Asimov Publisher: Harper Collins Pages: 133 <button>Delete</button>
RACECARRACECAR	Author: Stephen King Publisher: Scribner Pages: 439 <button>Delete</button>

1. Dockerfile Backend

```
FROM python:3.10-slim
WORKDIR /app
COPY requirements.txt .
RUN pip3 install -r requirements.txt
COPY . .
CMD ["flask", "--debug", "run", "--host=0.0.0.0"]
```

Deoarece server-ul nostru foloseste Flask are nevoie de python si toate librariile necesare (gasite in requirements.txt) ca sa ruleze, asa ca ne vom baza imaginea pe base image-ul de python, versiunea 3.10 si slim pentru a optimiza spatiul necesar.

Instalam requirements necesare aplicatiei si rulam server-ul.

Mentiune: backend-ul este facut in asa fel incat sa isi seteze portul pe care ruleaza si configuratiile pentru database din environment variables.

```
app.config["MONGO_URI"] = 'mongodb://' + os.environ['MONGODB_HOSTNAME'] + ':27017/' + os.environ['MONGODB_DATABASE']
```

```
if __name__ == "__main__":  
    ENVIRONMENT_DEBUG = os.environ.get("APP_DEBUG", True)  
    ENVIRONMENT_PORT = os.environ.get("APP_PORT", 5000)  
    app.run(port=ENVIRONMENT_PORT, debug=ENVIRONMENT_DEBUG)
```

2. Dockerfile frontend

```
FROM node:18-alpine3.16  
WORKDIR /app  
COPY package*.json ./  
RUN npm install  
COPY . .  
RUN npm run build  
CMD [ "npm", "run", "serve" ]
```

Aplicatia are nevoie de Vue si toate pachetele relevante gasite in fisierul package.json. Ne bazam imaginea pe cea de Node, versiunea 18, alpine deoarece este extrem de lightweight. Instalam librariile necesare folosind npm si rulam build si serve, aplicatia fiind servita pe portul default 8080.

3. Docker compose

```
version: "3.8"
services:
  mongodb-books-db:
    container_name: database
    image: mongo:6.0.2-focal
    environment:
      MONGO_INITDB_DATABASE: library
    ports:
      - 27017:27017
    volumes:
      - mongodbddata:/data/db
    networks:
      - backend
    restart: unless-stopped
```

In fisierul docker-compose.yml ne declaram serviciile incepand cu baza de date care foloseste imaginea de mongoDB. li setam enviroment-ul pentru a initializa un database pentru aplicatia noastra, mapam portul 27017 al containerului cu portul 27017 al masinii, si setam volumul pentru stocarea persistenta a datelor si network-ul pentru comunicarea cu backend-ul.

```
flask-books-backend:
  container_name: backend
  build: ./backend
  ports:
    - 5000:5000
  depends_on:
    - mongodb-books-db
  environment:
    APP_DEBUG: "True"
    APP_PORT: 5000
    MONGODB_DATABASE: library
```

```
MONGODB_HOSTNAME: mongodb-books-db
networks:
  - backend
  - frontend
restart: unless-stopped
```

Pentru backend definim ca dependenta serviciul bazei de date. Mapam portul 5000 al masinii cu portul 5000 al containerului unde ruleaza default backend-ul. Folosim environment variables pentru a specifica baza de date si numele acesteia.

```
vue-books-frontend:
  container_name: frontend
  build: ./frontend
  ports:
    - 8000:8080
  depends_on:
    - flask-books-backend
  environment:
    - PROXY_API=flask-books-backend
  networks:
    - frontend
  restart: unless-stopped
```

Frontend-ul mapam portul 8000 al masinii cu portul 8080 al container-ului. Definim ca dependenta serviciul de backend. Folosim variabila de environment PROXY_API pentru a trece peste problemele de CORS.

```
networks:
  frontend:
    driver: bridge
  backend:
    driver: bridge
volumes:
  mongodbbdata:
    driver: local
```

Definim network-urile si volumul necesar pentru servicii si db.

Ruland docker compose up --build imaginile noastre se vor construi si apoi vor rula in contextul containerelor separate.

<input type="checkbox"/>		NAME	IMAGE	STATUS	PORT(S)	STARTED	
<input type="checkbox"/>	▼	lab2 3 containers	-	Running (3/3)	-		Open
<input type="checkbox"/>		database 62885430365c	mongo	Running	27017	2 minutes ago	
<input type="checkbox"/>		backend e16fe27449df	lab2_flask-books-backend	Running	5000	2 minutes ago	
<input type="checkbox"/>		frontend 29ddd1026b8b	lab2_vue-books-frontend	Running	8000	2 minutes ago	

\$ docker ps						
CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
29ddd1026b8b	lab2_vue-books-frontend	"docker-entrypoint.s..."	2 minutes ago	Up 2 minutes	0.0.0.0:8000->8080/tcp	frontend
e16fe27449df	lab2_flask-books-backend	"flask --debug run -..."	8 minutes ago	Up 2 minutes	0.0.0.0:5000->5000/tcp	backend
62885430365c	mongo:6.0.2-focal	"docker-entrypoint.s..."	About an hour ago	Up 2 minutes	0.0.0.0:27017->27017/tcp	database

Acum aplicatia noastra este accesibila pe portul :8000 al masinii.