Voting App

Group Members :

1. Reno Bakti **(001202300218)**
2. Carel Edgar Napitupulu **(001202300172)**

1. Introduction

In the modern era of web development, deploying applications in a consistent and reproducible environment is crucial. This project was created as part of a Distributed and Parallel System course, aiming to demonstrate how a fullstack web application can be containerized using Docker.

The application developed is a **simple online voting system**, where users can vote for one of several candidates and see real-time results. The system is composed of:

- A **frontend** built with React and Vite for a responsive user interface
- A **backend** built with Express.js to handle API requests and voting logic
- A **MySQL database** to persist vote data
- A **phpMyAdmin** interface to manage the database visually

To ensure consistency across development and production environments, each component is packaged in its own **Docker container**, and the entire system is orchestrated using **Docker Compose**. This approach not only simplifies deployment but also teaches the fundamentals of container-based architecture.

The goal of this project is to provide hands-on experience in building a complete web application using modern tools and to apply best practices for containerized deployment.

2. Project Description

This project is a simple web-based voting system application that includes:

- **Frontend** built using React + Vite
- **Backend** built using Express.js (Node.js)
- **Database** using MySQL
- **phpMyAdmin** for database management

All components run inside **Docker containers**, orchestrated using docker-compose.

3. Project Structure

```
voting-app/
├── backend/
│   ├── Dockerfile
│   ├── index.js
│   └── ...
├── frontend/
│   ├── Dockerfile
│   ├── src/
│   └── ...
├── docker-compose.yml
└── README.md
```

4. How to Run the Application

Clone the Repository

git clone https://github.com/renobadhak/voting-app.git

cd voting-app

Start Docker Compose

docker-compose up –build

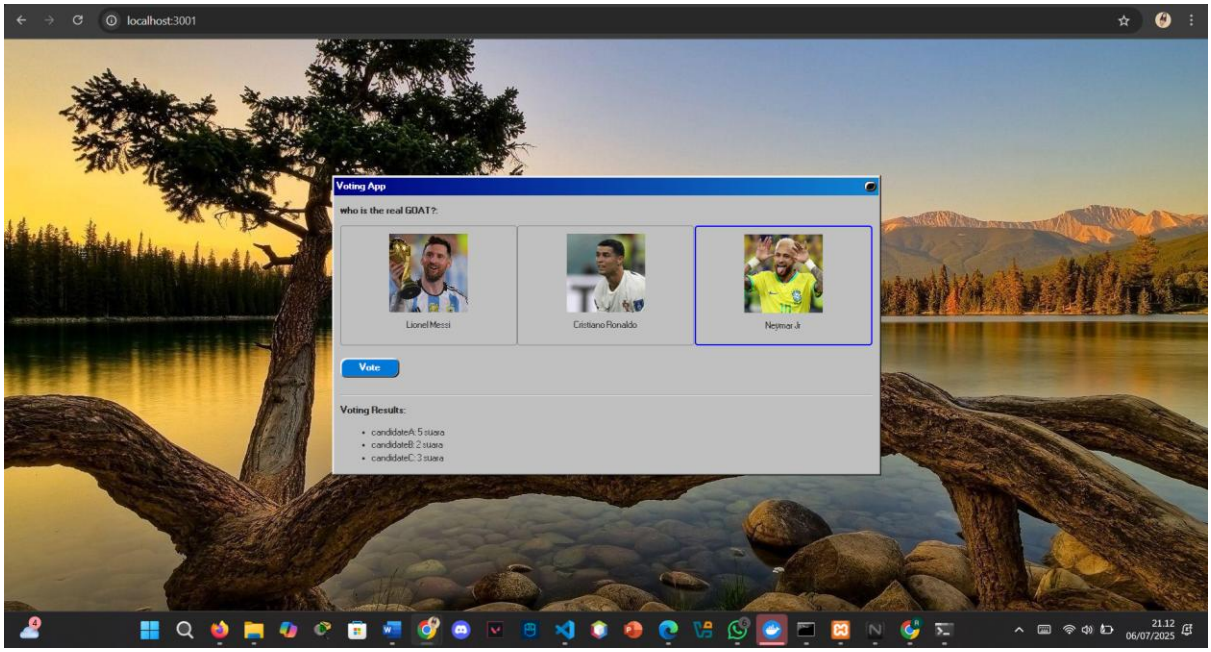
This command will build and run 4 containers:

- frontend: React App (on port 3001)
- backend: Node.js API (on port 5000)
- db: MySQL database (on port 3307 from host)
- phpmyadmin: GUI for database (on port 8080)

5. Access Points

Frontend	http://localhost:3001
Backend API	http://localhost:5000/votes
phpMyAdmin	http://localhost:8080
DB Credentials	Username: root, Password: root

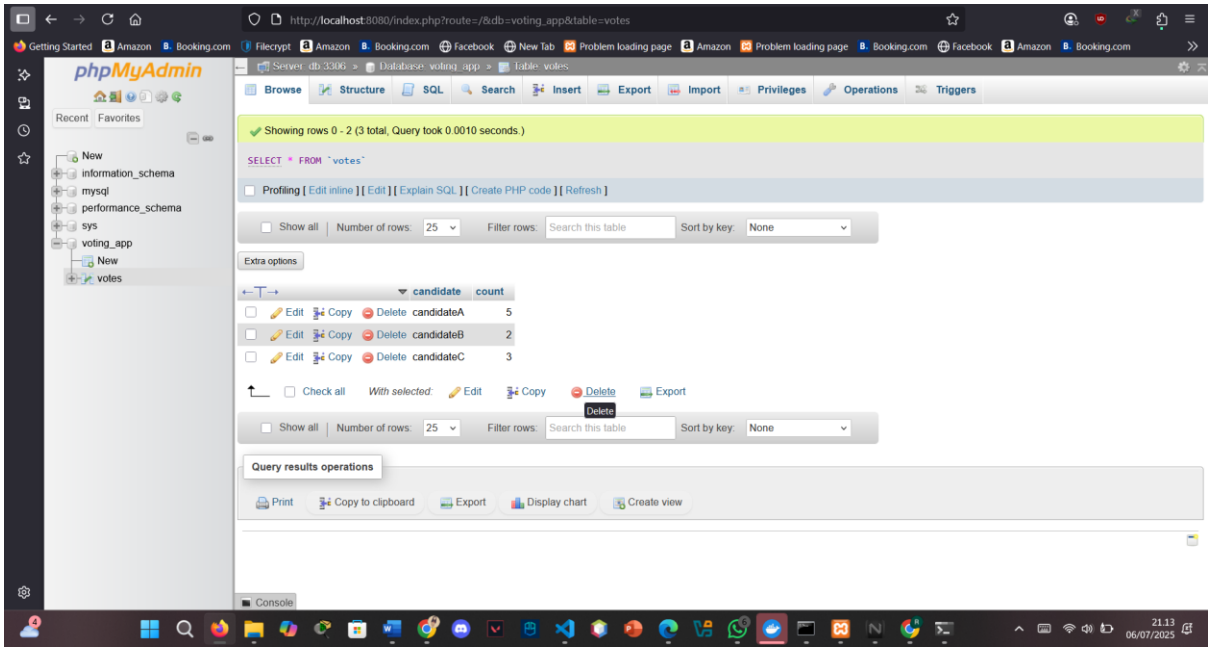
1. Frontend voting page



There are 3 options with pictures and the Below are the voting results user can choose the option first and then click “vote” button.

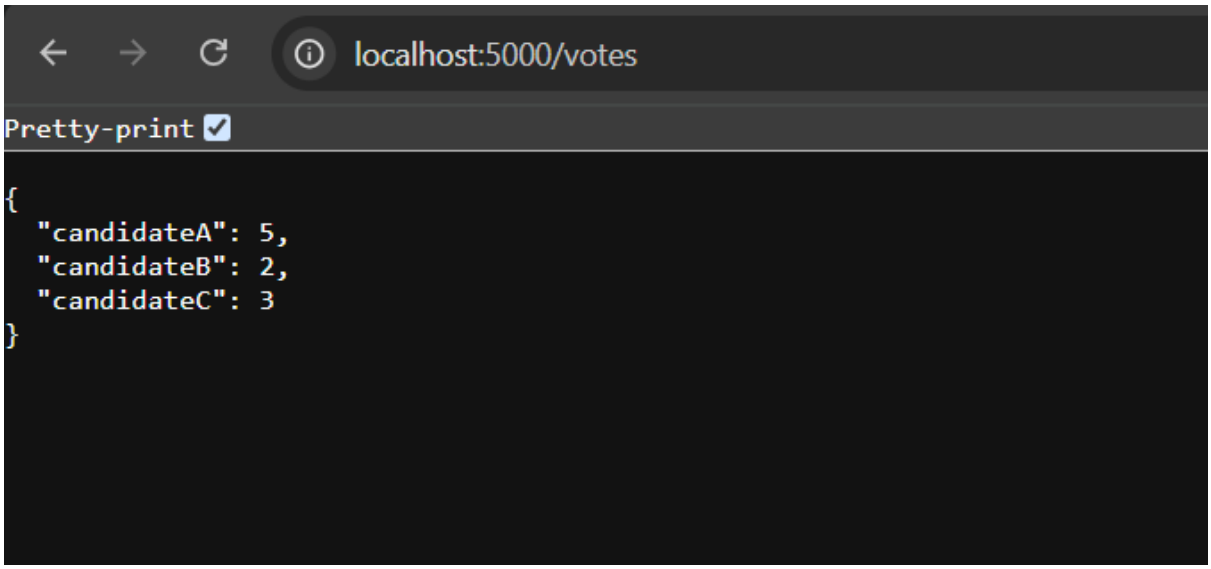
After that the data will be input and entered into the database

2. phpMyAdmin showing votes table



As you can see there are data that has been entered

3. API response from /votes



6. Program File

Backend/ Docker File

```
FROM node:18

WORKDIR /app

COPY package*.json ./
RUN npm install

COPY . .

# Salin script wait-for-it ke dalam image
ADD https://raw.githubusercontent.com/vishnubob/wait-for-it/master/wait-for-it.sh /app/wait-for-it.sh
RUN chmod +x /app/wait-for-it.sh

EXPOSE 5000

CMD ["sh", "-c", "./wait-for-it.sh db:3306 -- node index.js"]
```

- The Node.js container is built and dependencies are installed.
- All application code is copied into the container.
- The `wait-for-it.sh` script is added to wait for the database.
- When the container starts, the `wait-for-it.sh` script ensures that the `db` (MySQL) service is up and running.
- After that, the Express backend (`index.js`) is executed.
- The application is ready to receive requests on port 5000.

Backend/ index.js

```
const express = require('express');
const cors = require('cors');
const mysql = require('mysql2');

const app = express();
const port = 5000;

app.use(cors());
app.use(express.json());

// Koneksi ke MySQL
const db = mysql.createPool({
  host: 'db',
  user: 'root',
  password: 'root',
  database: 'voting_app',
  connectionLimit: 10
});

// Inisialisasi tabel jika belum ada
db.query(`
  CREATE TABLE IF NOT EXISTS votes (
    candidate VARCHAR(255) PRIMARY KEY,
    count INT DEFAULT 0
)`, (err) => {
  if (err) {
    console.error('Error creating table:', err);
  } else {
    // Inisialisasi data kandidat jika kosong
    const candidates = ['candidateA', 'candidateB', 'candidateC'];
    candidates.forEach(candidate => {
      db.query(
        'INSERT IGNORE INTO votes (candidate, count) VALUES (?, 0)',
        [candidate]
      );
    });
    console.log('Voting table initialized.');
  }
});

// Endpoint root
app.get('/', (req, res) => {
  res.send('Voting API is running.');
});

// Ambil hasil voting
app.get('/votes', (req, res) => {
```

```
  db.query('SELECT * FROM votes', (err, results) => {
    if (err) {
      return res.status(500).json({ error: 'Database error' });
    }
    const formatted = {};
    results.forEach(row => {
      formatted[row.candidate] = row.count;
    });
    res.json(formatted);
  });
});

// Kirim vote
app.post('/vote', (req, res) => {
  const { candidate } = req.body;
  db.query(
    'UPDATE votes SET count = count + 1 WHERE candidate = ?',
    [candidate],
    (err, result) => {
      if (err) {
        return res.status(500).json({ error: 'Database error' });
      }
      if (result.affectedRows === 0) {
        return res.status(400).json({ error: 'Invalid candidate' });
      }
      res.json({ message: 'Vote counted' });
    }
  );
});

app.listen(port, () => {
  console.log(`Backend running on port ${port}`);
});
```

- Initializes the voting table and candidate data automatically
- Allows fetching vote results and voting for candidates via API
- Works perfectly with the frontend React app and Docker setup

Frontend/ Docker File

```
# build
FROM node:18 as build
WORKDIR /app
COPY . .
RUN npm install && npm run build

# serve
FROM nginx:alpine
COPY --from=build /app/dist /usr/share/nginx/html
COPY nginx.conf /etc/nginx/conf.d/default.conf
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

The first container is used to **build the React application**:

- Installs all required dependencies.
- Compiles the code into static HTML, JS, and CSS files (located in the `dist/` folder).

The second container runs **Nginx**:

- Serves the static files from the build to the user's browser.
- The application can be accessed via http://localhost:3001.

I will explain about nginx.conf

```
server {
  listen 80;

  location / {
    root /usr/share/nginx/html;
    index index.html;
    try_files $uri /index.html;
  }

  location /votes {
    proxy_pass http://backend:5000/votes;
  }

  location /vote {
    proxy_pass http://backend:5000/vote;
  }
}
```

- **NGINX serves the static frontend files** from the `dist` folder.

- **All API requests** (`/votes` and `/vote`) are **proxied to the backend server** running in a separate container.

- **React can handle client-side routing properly**, ensuring that navigation within the app works even when refreshing or accessing deep links.

- **All communication happens between containers** through the defined Docker network (`app-network`), enabling seamless interaction without exposing internal services to the public.

docker-compose.yml

```yaml
version: '3.8'

services:
  db:
    image: mysql:8
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: root
      MYSQL_DATABASE: voting_app
    ports:
      - "3307:3306"
    networks:
      - app-network

  backend:
    build: ./backend
    ports:
      - "5000:5000"
    depends_on:
      - db
    environment:
      DB_HOST: db
      DB_USER: root
      DB_PASSWORD: root
      DB_NAME: voting_app
    networks:
      - app-network
    command: sh -c "./wait-for-it.sh db:3306 -- node index.js"

  frontend:
```

```
    build: ./frontend
    ports:
      - "3001:80"
    depends_on:
      - backend
    networks:
      - app-network

  phpmyadmin:
    image: phpmyadmin/phpmyadmin
    restart: always
    ports:
      - "8080:80"
    environment:
      PMA_HOST: db
      PMA_PORT: 3306
    depends_on:
      - db
    networks:
      - app-network

networks:
  app-network:
```

7. Conclussion:

This project demonstrates how to build and deploy a fullstack web application using Docker. The integration between frontend, backend, and MySQL ensures a consistent development environment and simplifies the deployment process.

GitHub Repository
https://github.com/renobadhak/voting-app