**Лабораторна робота №5**

Використання Docker, для розгортання додатку Node.JS

**Мета роботи:**

Ознайомитись з використанням Docker, удосконалити знання в Node.js та модифікувати створений додаток з використанням docker compose.

# Результат виконання:

Модифікування створеного додатку продемонстровані на лістингах 1-6.

Лістинг 1 - app.js

const express = require("express");

const app = express();

require("dotenv").config();

const jsonParser = express.json();

const {

connectMongoDB,

closeMongoDB,

getUsers,

getUser,

insertUser,

removeUserByName,

updateUser,

removeAllUsers,

} = require("./mongoDB.js");

const host = process.env.HOST;

const port = process.env.PORT;

app.use(express.static("public"));

app.get("/api/users", async (req, res) => {

await connectMongoDB();

const users = await getUsers();

await closeMongoDB();

res.json(users);

});

app.get("/api/users/:id", async (req, res) => {

await connectMongoDB();

const user = await getUser(req.params.id);

await closeMongoDB();

res.json(user);

});

app.post("/api/users", jsonParser, async (req, res) => {

await connectMongoDB();

const response = await insertUser(req.body);

await closeMongoDB();

res.json(response);

});

app.delete("/api/users/", async (req, res) => {

await connectMongoDB();

await removeAllUsers();

await closeMongoDB();

res.json({ message: "All users removed" });

});

app.delete("/api/users/:id", async (req, res) => {

await connectMongoDB();

await removeUserByName(req.params.id);

await closeMongoDB();

res.json({ message: "User removed", \_id: req.params.id });

});

app.put("/api/users/", jsonParser, async (req, res) => {

await connectMongoDB();

const response = await updateUser(req.body);

await closeMongoDB();

res.json(response);

});

app.listen(port, () => {

console.log(`Server is running at ${host}:${port}`);

});

Лістинг 2 – index.html

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<meta name="viewport" content="width=device-width" />

<title>Список пользователей</title>

<link

href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"

rel="stylesheet"

/>

</head>

<body>

<h2>Список пользователей</h2>

<form name="userForm">

<input type="hidden" name="id" value="0" />

<div class="form-group">

<label for="name">Имя:</label>

<input class="form-control" name="name" />

</div>

<div class="form-group">

<label for="age">Возраст:</label>

<input class="form-control" name="age" />

</div>

<div class="panel-body">

<button type="submit" class="btn btn-sm btn-primary">Сохранить</button>

<button type="reset" class="btn btn-sm btn-primary">Сбросить</button>

<button id="remove-all" class="btn btn-sm btn-primary">

Удалить все

</button>

</div>

</form>

<table class="table table-condensed table-striped table-bordered">

<thead>

<tr>

<th>Id</th>

<th>Имя</th>

<th>возраст</th>

<th></th>

</tr>

</thead>

<tbody></tbody>

</table>

<style>

body {

margin: 20px;

}

</style>

<script>

// Получение всех пользователей

async function GetUsers() {

const response = await fetch("/api/users", {

method: "GET",

headers: { Accept: "application/json" },

});

if (response.ok === true) {

const users = await response.json();

let rows = document.querySelector("tbody");

console.log(users);

users.forEach((user) => {

rows.append(row(user));

});

}

}

// Получение одного пользователя

async function GetUser(id) {

const response = await fetch("/api/users/" + id, {

method: "GET",

headers: { Accept: "application/json" },

});

if (response.ok === true) {

const user = await response.json();

const form = document.forms["userForm"];

form.elements["id"].value = user.\_id;

form.elements["name"].value = user.name;

form.elements["age"].value = user.age;

}

}

// Добавление пользователя

async function CreateUser(userName, userAge) {

const response = await fetch("api/users", {

method: "POST",

headers: {

Accept: "application/json",

"Content-Type": "application/json",

},

body: JSON.stringify({

name: userName,

age: parseInt(userAge, 10),

}),

});

if (response.ok === true) {

const user = await response.json();

reset();

document.querySelector("tbody").append(row(user));

}

}

// Изменение пользователя

async function EditUser(userId, userName, userAge) {

const response = await fetch("api/users", {

method: "PUT",

headers: {

Accept: "application/json",

"Content-Type": "application/json",

},

body: JSON.stringify({

\_id: userId,

name: userName,

age: parseInt(userAge, 10),

}),

});

if (response.ok === true) {

const user = await response.json();

reset();

document

.querySelector("tr[data-rowid='" + user.\_id + "']")

.replaceWith(row(user));

}

}

// Удаление пользователя

async function DeleteUser(id) {

const response = await fetch("/api/users/" + id, {

method: "DELETE",

headers: { Accept: "application/json" },

});

if (response.ok === true) {

const user = await response.json();

document.querySelector("tr[data-rowid='" + user.\_id + "']").remove();

}

}

// сброс формы

function reset() {

const form = document.forms["userForm"];

form.reset();

form.elements["id"].value = 0;

}

// создание строки для таблицы

function row(user) {

const tr = document.createElement("tr");

tr.setAttribute("data-rowid", user.\_id);

const idTd = document.createElement("td");

idTd.append(user.\_id);

tr.append(idTd);

const nameTd = document.createElement("td");

nameTd.append(user.name);

tr.append(nameTd);

const ageTd = document.createElement("td");

ageTd.append(user.age);

tr.append(ageTd);

const linksTd = document.createElement("td");

const editLink = document.createElement("a");

editLink.setAttribute("data-id", user.\_id);

editLink.setAttribute("style", "cursor:pointer;padding:15px;");

editLink.append("Изменить");

editLink.addEventListener("click", (e) => {

e.preventDefault();

console.log(user.\_id);

GetUser(user.\_id);

});

linksTd.append(editLink);

const removeLink = document.createElement("a");

removeLink.setAttribute("data-id", user.\_id);

removeLink.setAttribute("style", "cursor:pointer;padding:15px;");

removeLink.append("Удалить");

removeLink.addEventListener("click", (e) => {

e.preventDefault();

DeleteUser(user.\_id);

});

linksTd.append(removeLink);

tr.appendChild(linksTd);

return tr;

}

document.querySelector("#remove-all").addEventListener("click", (e) => {

e.preventDefault();

fetch("/api/users", {

method: "DELETE",

headers: { Accept: "application/json" },

}).then((response) => {

if (response.ok === true) {

document.querySelector("tbody").innerHTML = "";

}

});

});

// отправка формы

document.forms["userForm"].addEventListener("submit", (e) => {

e.preventDefault();

const form = document.forms["userForm"];

const id = form.elements["id"].value;

const name = form.elements["name"].value;

const age = form.elements["age"].value;

if (!name || !age) return;

if (id == 0) CreateUser(name, age);

else EditUser(id, name, age);

});

GetUsers();

</script>

</body>

</html>

Лістинг 3 – MongoDB.js

const mongoose = require("mongoose");

const { Schema } = mongoose;

const url = `mongodb://my\_mongodb:27017/usersdb`;

const userSchema = new Schema({

name: String,

age: Number,

});

const User = mongoose.model("User", userSchema);

const connectMongoDB = async () => {

try {

await mongoose.connect(url);

} catch (e) {

console.log(e);

}

};

const closeMongoDB = async () => {

try {

await mongoose.connection.close();

} catch (e) {

console.log(e);

}

};

const insertUser = async (user) => {

try {

const newUser = new User(user);

const savedUser = await newUser.save();

return savedUser;

} catch (e) {

console.log(e);

}

};

const getUser = async (id) => {

try {

return await User.findById(id);

} catch (e) {

console.log(e);

}

};

const getUsers = async () => {

try {

return await User.find({});

} catch (e) {

console.log(e);

}

};

const removeUserByName = async (id) => {

try {

await User.findByIdAndDelete(id);

} catch (e) {

console.log(e);

}

};

const removeAllUsers = async () => {

try {

await User.deleteMany({});

} catch (e) {

console.log(e);

}

};

const updateUser = async (user) => {

try {

const updatedUser = await User.findByIdAndUpdate(user.\_id, user, {

new: true,

});

return updatedUser;

} catch (e) {

console.log(e);

}

};

module.exports = {

connectMongoDB,

closeMongoDB,

insertUser,

getUser,

getUsers,

removeUserByName,

updateUser,

removeAllUsers,

};

Лістинг 4 – Dockerfile

FROM node:latest

RUN mkdir -p /usr/src/app/

WORKDIR /usr/src/app/

COPY . /usr/src/app/

RUN npm install

EXPOSE 3000

CMD ["node", "--env-file=.env", "api/app.js"]

Лістинг 5 – docker-compose.yaml

version: "3"

volumes:

mongo\_db\_data:

services:

my\_mongodb:

image: mongo:latest

volumes:

- mongo\_db\_data:/data/db

app:

build: .

ports:

- "3000:3000"

environment:

- MONGO\_DB\_HOSTNAME=my\_mongodb

- MONGO\_DB\_PORT=27017

- MONGO\_DB=usersdb

Лістинг 6 – .dockerignore

node\_modules

docker-compose.yaml

Dockerfile

package-lock.json

[Посилання на репозиторій](https://github.com/chstu-labs/6-sem-labs-software-architecture-bogdan)

В результаті отримали додаток, який працює з БД MongoDB на Docker compose на рисунках 1-2.

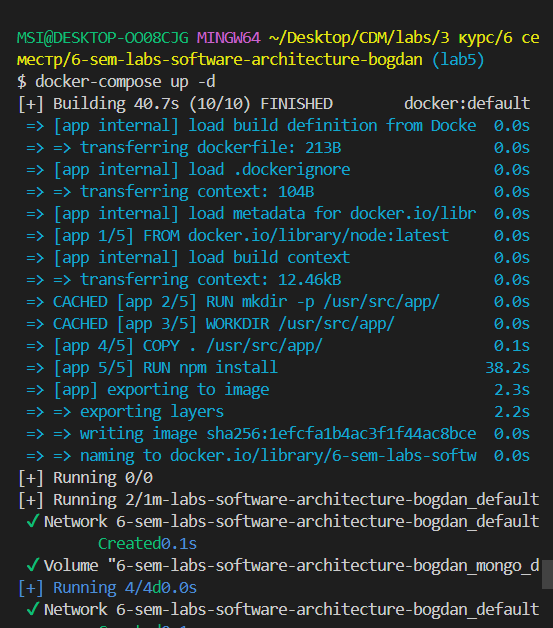


Рисунок 1 – Запуск docker compose

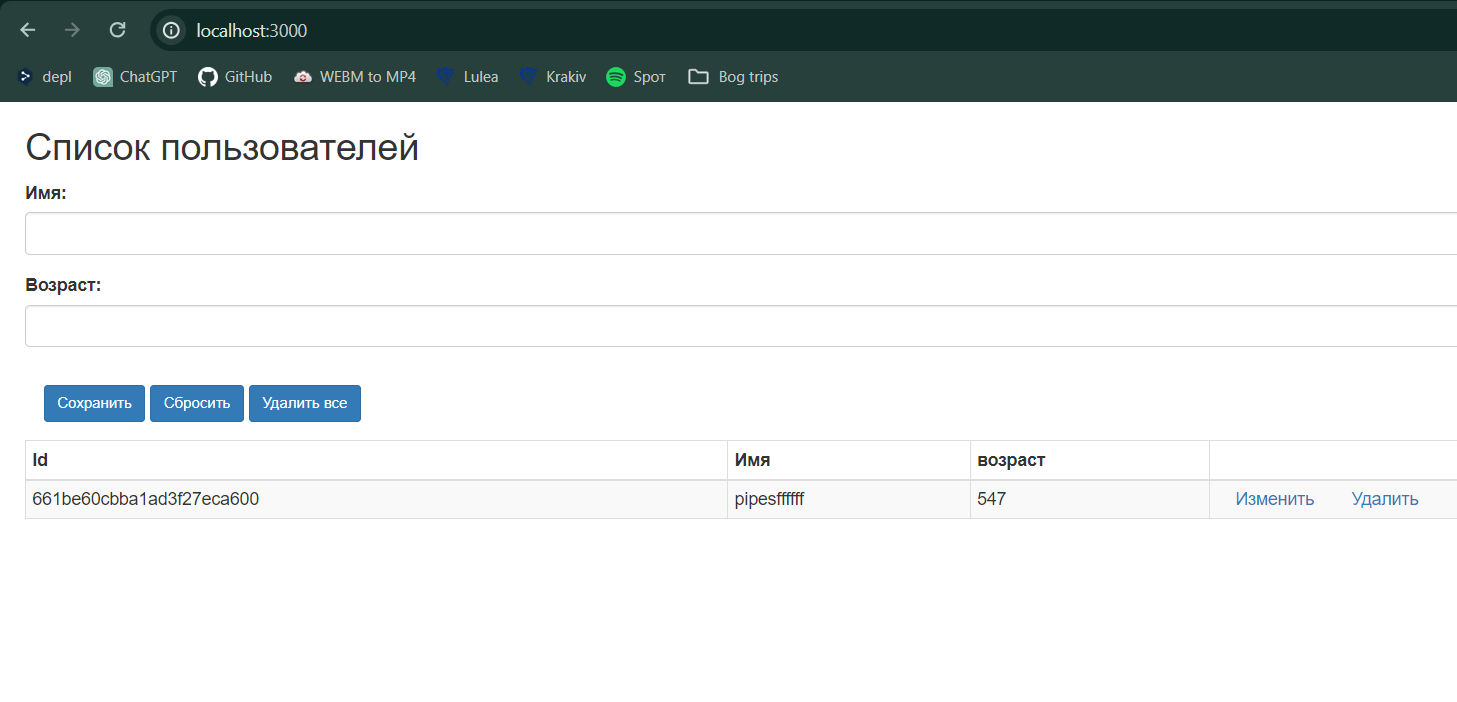


Рисунок 2 – Створений новий додаток

# Висновок

В ході лабораторної роботи я ознайомився з Docker, удосконалив свої знання в Node.js та модифікував створений додаток з використанням docker compose та власної бази MongoDB.