Лабораторна робота №3

Node.js та MongoDB.

# Виконання роботи

## Створити CRUD методи для бд.

### Лістинг створених CRUD команд

const { MongoClient, ObjectId } = require("mongodb");

const uri = "mongodb://localhost:27017";

const client = new MongoClient(uri);

async function connectToMongo() {

  try {

    await client.connect();

    console.log("Connected to MongoDB");

  } catch (error) {

    console.error("Error connecting to MongoDB:", error);

    throw new Error(error?.message);

  }

}

async function closeMongo() {

  try {

    await client.close();

    console.log("MongoDB connection closed.");

  } catch (error) {

    console.error("Error closing MongoDB connection:", error);

    throw new Error(error?.message);

  }

}

function errorHandler(method) {

  return async function (...params) {

    try {

      return await method(...params);

    } catch (error) {

      console.error("Error in method:", error);

      throw new Error(error?.message);

    }

  };

}

async function getUser(query) {

  const database = client.db("swat-lab3");

  const collection = database.collection("users");

  const user = await collection.findOne(query);

  return user;

}

async function insertUser(user) {

  const database = client.db("swat-lab3");

  const collection = database.collection("users");

  const result = await collection.insertOne(user);

  return result;

}

async function updateUser(query, update) {

  if (update?.\_id) {

    delete update.\_id;

  } else if (query?.\_id) {

    query.\_id = ObjectId.createFromHexString(query.\_id);

  }

  const database = client.db("swat-lab3");

  const collection = database.collection("users");

  const result = await collection.updateOne(query, {

    $set: update,

  });

  return result;

}

async function deleteUser(query) {

  if (query?.\_id) {

    query.\_id = ObjectId.createFromHexString(query.\_id);

  }

  const database = client.db("swat-lab3");

  const collection = database.collection("users");

  const result = await collection.deleteOne(query);

  if (result.deletedCount === 0) {

    throw new Error("User not found");

  }

  return result;

}

async function getUsers(query) {

  if (query?.\_id) {

    query.\_id = ObjectId.createFromHexString(query.\_id);

  }

  const database = client.db("swat-lab3");

  const collection = database.collection("users");

  const users = await collection.find(query).toArray();

  return users;

}

module.exports = {

  connectToMongo,

  closeMongo,

  getUser: errorHandler(getUser),

  insertUser: errorHandler(insertUser),

  updateUser: errorHandler(updateUser),

  deleteUser: errorHandler(deleteUser),

  getUsers: errorHandler(getUsers),

};

## Створити сервер для реалізації взаємодії з бд

### Лістинг створення і запуску серверу

const express = require("express");

const bodyParser = require("body-parser");

const cors = require("cors");

const {

  getUsers,

  insertUser,

  updateUser,

  deleteUser,

  closeMongo,

  connectToMongo,

} = require("./mongodb");

const localhost = "http://localhost";

const host = process.env.HOST || localhost;

const port = 3000;

const app = express();

if (host === localhost) {

  app.use(cors());

  console.log("CORS enabled");

}

app.use(bodyParser.urlencoded({ extended: false }));

app.use(bodyParser.json());

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

// API Routes

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

  try {

    const users = await getUsers();

    res.json(users);

  } catch (error) {

    res.status(500).json({ error: "Internal server error" });

  }

});

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

  try {

    const newUser = req.body;

    const result = await insertUser(newUser);

    res.json(result);

  } catch (error) {

    res.status(500).json({ error: "Internal server error" });

  }

});

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

  try {

    const { id } = req.params;

    const result = await deleteUser({ \_id: id });

    res.json(result);

  } catch (error) {

    res.status(500).json({ error: "Internal server error" });

  }

});

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

  try {

    const { id } = req.params;

    const newUserData = req.body;

    const result = await updateUser({ \_id: id }, newUserData);

    res.json(result);

  } catch (error) {

    res.status(500).json({ error: "Internal server error" });

  }

});

// Start the server

app.listen(port, async () => {

  console.log(`Server is running on port ${port}`);

  await connectToMongo();

});

// Graceful shutdown

process.on("SIGINT", async () => {

  console.log("Server is shutting down...");

  await closeMongo();

  process.exit(0);

});

## Створити користувацький інтерфейс для взаємодії з сервером

### Лістинг головного html файлу

<!DOCTYPE html>

<html lang="en">

  <head>

    <meta charset="UTF-8" />

    <meta name="viewport" content="width=device-width, initial-scale=1.0" />

    <title>MongoDB Methods Test</title>

    <link rel="preconnect" href="https://fonts.googleapis.com" />

    <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin />

    <link

      href="https://fonts.googleapis.com/css2?family=Roboto&display=swap"

      rel="stylesheet"

    />

    <script src="https://cdn.jsdelivr.net/npm/axios/dist/axios.min.js"></script>

    <style>

      body {

        font-family: "Roboto", sans-serif;

        font-weight: 400;

        font-style: normal;

      }

    </style>

  </head>

  <body>

    <h1>MongoDB Methods Test</h1>

    <!-- Form for adding a new user -->

    <h2>Add New User</h2>

    <form id="add-user-form" onsubmit="addUser(event)">

      <label for="name">Name:</label>

      <input type="text" id="name" name="name" required />

      <label for="age">Age:</label>

      <input type="number" id="age" name="age" required />

      <button type="submit">Add User</button>

    </form>

    <!-- Table for displaying all users -->

    <h3>Users</h3>

    <table>

      <thead>

        <tr>

          <th>ID</th>

          <th>Name</th>

          <th>Age</th>

          <th>Actions</th>

        </tr>

      </thead>

      <tbody id="users">

        <!-- Users will be added here -->

      </tbody>

    </table>

    <script>

      const form = document.getElementById("add-user-form");

      const usersTable = document.getElementById("users");

      // Function to get all users

      const getUsers = async () => {

        try {

          const response = await axios.get("/api/users");

          const users = response.data;

          console.log(users);

          usersTable.innerHTML = "";

          users.forEach((user) => {

            usersTable.innerHTML += `

                <tr data-id="${user.\_id}">

                  <td>${user.\_id}</td>

                  <td>${user.name}</td>

                  <td>${user.age}</td>

                  <td>

                    <button onclick="deleteUser('${user.\_id}')">Delete</button>

                    <button onclick="makeRowEditable('${user.\_id}')">Update</button>

                  </td>

                </tr>

              `;

          });

        } catch (error) {

          console.error(error);

        }

      };

      // Function to add a new user

      const addUser = async (event) => {

        event.preventDefault();

        const name = event.target.name.value;

        const age = event.target.age.value;

        try {

          const response = await axios.post("/api/users", { name, age });

          const user = response.data;

          await getUsers();

          event.target.reset();

        } catch (error) {

          console.error(error);

        }

      };

      // Function to delete a user

      const deleteUser = async (id) => {

        try {

          await axios.delete(`/api/users/${id}`);

          await getUsers();

        } catch (error) {

          console.error(error);

        }

      };

      // Function to make a row editable

      const makeRowEditable = (id) => {

        const row = document.querySelector(`tr[data-id="${id}"]`);

        const name = row.children[0].textContent;

        const age = row.children[1].textContent;

        row.innerHTML = `

          <td><input type="text" value="${name}" /></td>

          <td><input type="number" value="${age}" /></td>

          <td>

            <button onclick="updateUser('${id}')">Save</button>

            <button onclick="getUsers()">Cancel</button>

          </td>

        `;

      };

      // Function to update a user

      const updateUser = async (id) => {

        const row = document.querySelector(`tr[data-id="${id}"]`);

        const name = row.children[0].children[0].value;

        const age = row.children[1].children[0].value;

        try {

          await axios.put(`/api/users/${id}`, { name, age });

          await getUsers();

        } catch (error) {

          console.error(error);

        }

      };

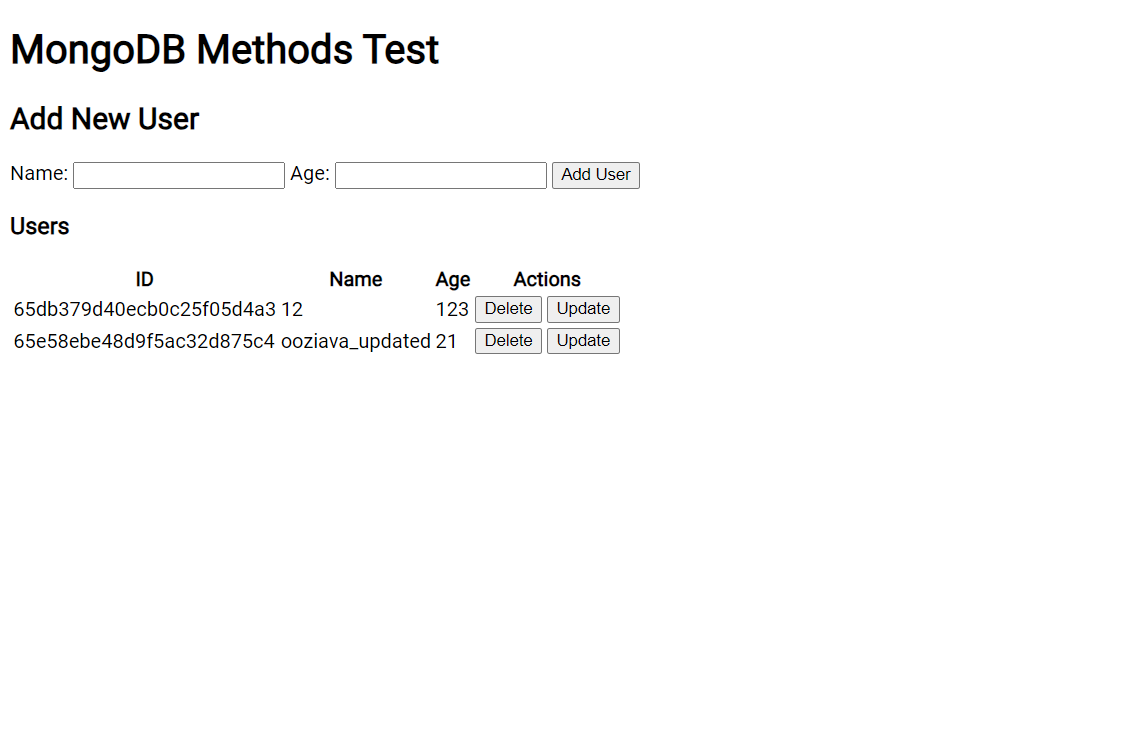
      getUsers();

    </script>

  </body>

</html>

### Вигляд інтерфейсу



## Модифікувати створений додаток

В ролі вдосконалення я обрав додавання масового видалення користувачів

### Лістинг методу для масового видалення з бд, та створення потрібного роуту на сервері

async function bulkDeleteUsers(ids) {

  if (!ids?.length) throw new Error("No IDs provided");

  const database = client.db("swat-lab3");

  const collection = database.collection("users");

  const result = await collection.deleteMany({

    \_id: {

      $in: ids.map((id) => ObjectId.createFromHexString(id)),

    },

  });

  return result;

}

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

  try {

    const { id } = req.params;

    const result = await deleteUser({ \_id: id });

    res.json(result);

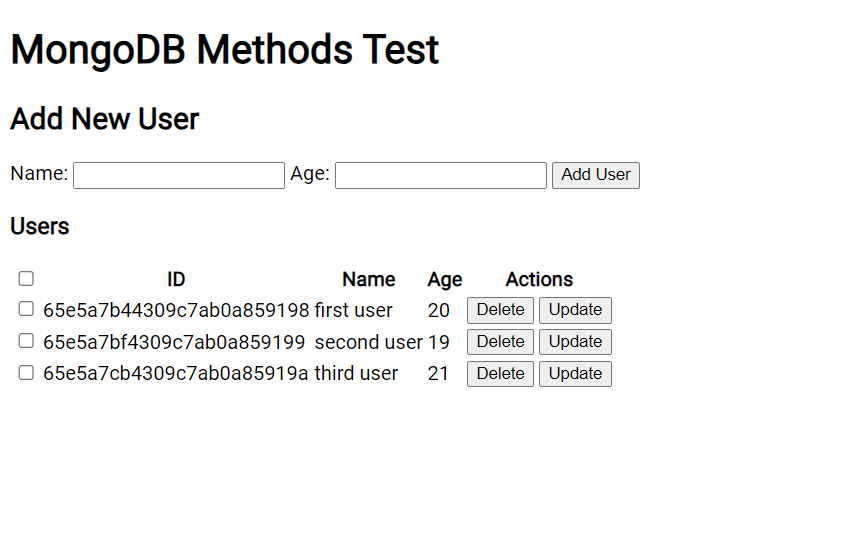
  } catch (error) {

    defaultErrorHandler(error, req, res);

  }

});

### Вигляд оновленого інтерфейсу



### Лістинг методів використаних для реалізації масового видалення користувачів

      // Function to delete selected users

      const deleteSelectedUsers = async () => {

        const selectUser = document.getElementsByName("select-user");

        const ids = [];

        selectUser.forEach((user) => {

          if (user.checked) {

            ids.push(user.parentElement.parentElement.dataset.id);

          }

        });

        try {

          await axios.delete(`/api/users`, { data: { ids } });

          await getUsers();

        } catch (error) {

          console.error(error);

        }

      };

      // Function to select all users

      const selectAllUsers = () => {

        const selectAll = document.getElementById("select-all");

        const selectUser = document.getElementsByName("select-user");

        const deleteButton = document.getElementById("delete-users");

        selectUser.forEach((user) => {

          user.checked = selectAll.checked;

        });

        deleteButton.style.display = selectAll.checked ? "block" : "none";

      };

      // Fuction to show button to delete selected users

      const showDeleteButton = () => {

        const selectUser = document.getElementsByName("select-user");

        const deleteButton = document.getElementById("delete-users");

        const selectAll = document.getElementById("select-all");

        let checked = false;

        let allChecked = true;

        selectUser.forEach((user) =>

          user.checked ? (checked = true) : (allChecked = false)

        );

        deleteButton.style.display = checked ? "block" : "none";

        selectAll.checked = allChecked;

      };

### Лістинг методів які зазнали модифікацій після реалізації масового видалення користувачів

      // Function to make a row editable

      const makeRowEditable = (id) => {

        const row = document.querySelector(`tr[data-id="${id}"]`);

        const name = row.children[0].textContent;

        const age = row.children[1].textContent;

        row.innerHTML = `

          <td></td>

          <td><input type="text" value="${name}" /></td>

          <td><input type="number" value="${age}" /></td>

          <td>

            <button onclick="updateUser('${id}')">Save</button>

            <button onclick="getUsers()">Cancel</button>

          </td>

        `;

      };

      // Function to get all users

      const getUsers = async () => {

        try {

          const response = await axios.get("/api/users");

          const users = response.data;

          console.log(users);

          usersTable.innerHTML = "";

          users.forEach((user) => {

            usersTable.innerHTML += `

                <tr data-id="${user.\_id}">

                  <td><input type="checkbox" name="select-user" onchange="showDeleteButton()" /></td>

                  <td>${user.\_id}</td>

                  <td>${user.name}</td>

                  <td>${user.age}</td>

                  <td>

                    <button onclick="deleteUser('${user.\_id}')">Delete</button>

                    <button onclick="makeRowEditable('${user.\_id}')">Update</button>

                  </td>

                </tr>

              `;

          });

          const deleteButton = document.getElementById("delete-users");

          const selectAll = document.getElementById("select-all");

          deleteButton.style.display = "none";

          selectAll.checked = false;

        } catch (error) {

          console.error(error);

        }

      };

# Висновок

В ході лабораторної роботи я стартував сервер, підключився до бд MongoDB, реалізував CRUD команди для взаємодії з бд, створив простий користувацький інтерфейс у вигляді html, дізнався про принципи роботи сервера з бд, при виконанні самостійного завдання дізнався про методи роботи з масивами даних.