Bachelor SEIS	HORIZON SCHOOL OF DIGITAL TECHNOLOGIES	2024-2025

NodeJS—Backend CRUD

Goal: Implement Create, Read, Update, and Delete (CRUD) operations on users.

- Implement basic CRUD functionality
- Use protected routes and cookies for session management

📝 Step 1: Backend - Create User Routes

1. Open `routes/` and create a new file:

   `routes/userRoutes.js`

2. Add the following **CRUD routes**:

- **Definition:**

  `Express.Router()` is a mini Express application that allows grouping routes together.

- **Why it's used:**

- Keeps routes modular and organized
- Avoids cluttering `server.js` with all route logic

```
const express = require("express");
const User = require("../models/User");
const router = express.Router();

//  Get All Users (READ)
router.get("/", async (req, res) => {
  try {
    const users = await User.find();
    res.status(200).json(users);
  } catch (err) {
    res.status(500).json({ message: err.message });
  }
});

//  Create New User (CREATE)
router.post("/create", async (req, res) => {
  try {
    const { name, email, password } = req.body;
```

```
    // Check if user exists
    let user = await User.findOne({ email });
    if (user) return res.status(400).json({ message: "User already exists"
});


    // Create new user
    X = new User({ name, email, password });
    await X.save();


    res.status(201).json(X);
  } catch (err) {
    res.status(500).json({ message: err.message });
  }
});


//  Update User (UPDATE)
router.put("/:id", async (req, res) => {
  try {
    const { name, email } = req.body;
    const updatedUser = await User.findByIdAndUpdate(
      req.params.id,
      { name, email },
      { new: true }
```

```
    );

    res.status(200).json(updatedUser);

  } catch (err) {

    res.status(500).json({ message: err.message });

  }

});



//  Delete User (DELETE)

router.delete("/:id", async (req, res) => {

  try {

    await User.findByIdAndDelete(req.params.id);

    res.status(200).json({ message: "User deleted successfully" });

  } catch (err) {

    res.status(500).json({ message: err.message });

  }

});



module.exports = router;
```

Step 2: Link Routes in `server.js`

In `server.js`, add:

```
const userRoutes = require("./routes/userRoutes");
app.use("/api/users", userRoutes);
```

Step 3: Frontend - Create Dashboard Page

1. In the `src/` folder, create `Dashboard.js`.

📌 Explanation of Each Library Used in `Dashboard.js`

1️⃣ React (`import React from "react";`)

- **Purpose:** React is a JavaScript library for building user interfaces.
- **Why it's used:** It allows us to create reusable UI components like `Dashboard`.

2️⃣ useState (`import { useState } from "react";`)

- **Purpose:** `useState` is a React Hook that lets us store and update values in a component.
- **Why it's used:**
  - It manages **users data** (`users` state).
  - It keeps track of **form inputs** (`form` state).
  - It stores **editing user information** (`editingUser` state).

Example in `Dashboard.js`:

```
const [users, setUsers] = useState([]); // Stores user list

const [form, setForm] = useState({ name: "", email: "" }); // Stores form input values
```

```
const [editingUser, setEditingUser] = useState(null); // Tracks which user is being edited
```

3️⃣ useEffect (`import { useEffect } from "react";`)

- **Purpose:** `useEffect` is a React Hook that runs code when the component loads or updates.
- **Why it's used:**
  - It **fetches users** when the component loads.
  - It **checks if the user is logged in** when the page loads.

Example in `Dashboard.js`:

```
useEffect(() => {
  fetchUsers(); // Fetches users when Dashboard loads
}, []); // Runs only once
```

4️⃣ useNavigate (`import { useNavigate } from "react-router-dom";`)

- **Purpose:** `useNavigate` is a React Router Hook for **programmatic navigation** (redirecting users).
- **Why it's used:**
  - Redirect users to the **login page** (`navigate("/")`) if they are not logged in.
  - Redirect users after **logging out**.

Example in `Dashboard.js`:

```
const navigate = useNavigate();

if (!token) {

  navigate("/"); // Redirects to login if no token

}
```

5️⃣ fetch API (`fetch("http://localhost:5000/api/users")`)

- **Purpose:** The `fetch` function makes HTTP requests to the backend.
- **Why it's used:**
  - Fetch **user data** from the backend (`GET /api/users`).
  - Send **form data** to create (`POST /api/users/create`) or update (`PUT /api/users/:id`) users.
  - Delete users (`DELETE /api/users/:id`).

Example in `Dashboard.js`:

```
const fetchUsers = async () => {

  const res = await fetch("http://localhost:5000/api/users"); // GET request

  const data = await res.json();

  setUsers(data); // Store fetched users in state

};
```

📌 Summary of Libraries

Library	Purpose	Usage
React	UI Library	Builds the Dashboard component
useState	Manages state	Stores users, form input, and editing user
useEffect	Runs side effects	Fetches users and checks login status
useNavigate	Navigates between pages	Redirects users if not logged in
fetch API	Communicates with backend	Fetches, creates, updates, and deletes users

🚀 Now you have a **clear understanding** of why each library is used in your `Dashboard.js` file! Let me know if you need more details. 🎯

```
import React, { useState, useEffect } from "react";
import { useNavigate } from "react-router-dom";

const Dashboard = () => {
  const [users, setUsers] = useState([]);
  const [form, setForm] = useState({ name: "", email: "" });
  const [editingUser, setEditingUser] = useState(null);
  const navigate = useNavigate();

  // Check if user is logged in
  useEffect(() => {
    const token = localStorage.getItem("token");
    if (!token) {
      navigate("/"); // Redirect to login if no token
    }
  }, [navigate]);

  // Fetch Users
  const fetchUsers = async () => {
    const res = await fetch("http://localhost:5000/api/users");
    const data = await res.json();
    setUsers(data);
  };

  useEffect(() => {
    fetchUsers();
  }, []);

  // Handle Input Changes
  const handleChange = (e) => {
    setForm({ ...form, [e.target.name]: e.target.value });
  };

  // Create or Update User
  const handleSubmit = async (e) => {
    e.preventDefault();

    const method = editingUser ? "PUT" : "POST";
    const url = editingUser
      ? `http://localhost:5000/api/users/${editingUser._id}`
```

```
      : "http://localhost:5000/api/users/create";  //  Correct endpoint

    // If creating a new user, add a default password
    const userData = editingUser
      ? { name: form.name, email: form.email }
      : { name: form.name, email: form.email, password:
"defaultPassword123" };  //  Add password

    try {
      const response = await fetch(url, {
        method,
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify(userData),
      });

      if (!response.ok) {
        const errorData = await response.json();
        alert("Error: " + errorData.message);
        return;
      }

      fetchUsers(); //  Refresh user list after creation
      setForm({ name: "", email: "" });
      setEditingUser(null);
    } catch (error) {
      console.error("Error:", error);
      alert("Failed to create/update user. Check your backend.");
    }
  };


  // Delete User
  const deleteUser = async (id) => {
    await fetch(`http://localhost:5000/api/users/${id}`, { method:
"DELETE" });
    fetchUsers();
  };

  // Edit User
  const editUser = (user) => {
```

```
    setForm({ name: user.name, email: user.email });
    setEditingUser(user);
  };

  // Logout function
  const handleLogout = () => {
    localStorage.removeItem("token");
    navigate("/");
  };

  return (
    <div>
      <h2>User Management</h2>

      <button onClick={handleLogout}>Logout</button>

      {/* User Form */}
      <form onSubmit={handleSubmit}>
        <input
          type="text"
          name="name"
          placeholder="Name"
          value={form.name}
          onChange={handleChange}
          required
        />
        <input
          type="email"
          name="email"
          placeholder="Email"
          value={form.email}
          onChange={handleChange}
          required
        />
        <button type="submit">{editingUser ? "Update" : "Create"}
User</button>
      </form>

      {/* User List */}
      <table border="1">
```

```
        <thead>
          <tr>
            <th>Name</th>
            <th>Email</th>
            <th>Actions</th>
          </tr>
        </thead>
        <tbody>
          {users.map((user) => (
            <tr key={user._id}>
              <td>{user.name}</td>
              <td>{user.email}</td>
              <td>
                <button onClick={() => editUser(user)}>Edit</button>
                <button onClick={() =>
deleteUser(user._id)}>Delete</button>
              </td>
            </tr>
          ))}
        </tbody>
      </table>
    </div>
  );
};

export default Dashboard;
```

Step 4: Add Routing in `App.js`

In `src/App.js`, modify:

```
import React from "react";
import { BrowserRouter as Router, Route, Routes } from "react-router-dom";
import Dashboard from "./Dashboard";
```

```
function App() {
  return (
    <Router>
      <Routes>
        <Route path="/dashboard" element={<Dashboard />} />
      </Routes>
    </Router>
  );
}

export default App;
```

App.js:

```
import React from "react";
import { BrowserRouter as Router, Routes, Route, Navigate } from
"react-router-dom";
import Login from "./Login";
import Dashboard from "./Dashboard";

// Protected Route Component
const ProtectedRoute = ({ element }) => {
  const isAuthenticated = localStorage.getItem("token"); // Check if token
exists
  return isAuthenticated ? element : <Navigate to="/" />; // Redirect to
Login if not authenticated
};

function App() {
  return (
    <Router>
      <Routes>
        {/* Public Route - Login */}
        <Route path="/" element={<Login />} />

        {/* Protected Route - Dashboard */}
```

```
        <Route path="/dashboard" element={<ProtectedRoute
element={<Dashboard />} />} />
      </Routes>
    </Router>
  );
}

export default App;
```

Step 5: Test the CRUD

1. **Start the Backend:**

cd backend

npm run dev

2. Start the Frontend:

cd frontend

npm start

Go to `http://localhost:3000/dashboard`

- **Create a user**
- **View users**
- **Edit user**
- **Delete user**