UNIVERSITY OF KARACHI

NAME: BISMAH NASIR

SEAT NUMBER: B21110006023

BSCS-4TH YEAR SEC "B" (MORNING PROGRAM)

COURSE: BSCS 633 INTERNET APPLICATION DEVELOPMENT

COURSE INSTRUCTOR: DR. HUMERA TARIQ

DEPARTMENT OF COMPUTER SCIENCE

ASSIGNMENT

CRUD OPERATIONS APP

CRUD OPERATION APP

(FULL STACK DEVELOPMENT)

INTRODUCTION:

In today's web development landscape, mastering **CRUD (Create, Read, Update, Delete) operations** is essential for building dynamic and interactive applications. This project focuses on implementing CRUD functionalities to strengthen my understanding of both **frontend and backend development**, as well as their seamless integration. The application allows users to manage a list of projects, enabling them to **add, view, edit, and delete projects** efficiently. To achieve this, the project leverages **React.js** for the frontend, **Node.js and Express.js** for the backend, and **JSON** for data storage. This hands-on approach provides valuable experience in full-stack development, API handling, and state management, making it a practical learning exercise in modern web development.

GITHUB REPOSITORY:

Below is the github repository link of the project.

https://github.com/bismah-nasir/crud-mini-project

FEATURES:

This project provides a comprehensive set of functionalities to efficiently manage a list of projects using **CRUD** operations. The following features ensure a seamless user experience:

- **Add New Projects**
  - Users can enter a project name and add it to the list.
  - Duplicate project names are not allowed.
  - Project names must be at least three characters long.
- **Display Projects in a Structured Table Format**
  - All projects are displayed in a visually structured table.
  - Each row shows the **Project ID** and **Project Name** for clarity.

    - Buttons for **editing** and **deleting** each project are included in the table.

- **Edit Existing Projects**
    - Users can update the name of an existing project.
    - When clicking the **Edit** button, the project name becomes an editable text field.
    - Changes can be saved or canceled, preventing accidental modifications.
- **Delete Projects (With Deleted Project Details Shown)**
    - Users can remove projects from the list by clicking the **Delete** button.
    - After deletion, the details of the removed project (ID and Name) are displayed.
    - This prevents accidental deletions from going unnoticed.
- **Refresh Projects List**
    - A **Refresh Projects** button allows users to manually re-fetch the project list.
    - This ensures that the latest data is always displayed without needing to reload the page.
- **Show the Total Number of Projects**
    - The total number of stored projects is displayed at the top of the list.
    - This number updates dynamically whenever a project is added or removed.

TECHNOLOGIES USED:

- **Frontend:** React.js (useState, useEffect, Fetch API)
- **Backend:** Node.js, Express.js
- **Database:** JSON file

API ENDPOINTS:

This section describes all the API routes used in this project, including their endpoints, request methods, request bodies (if applicable).

- **GET - Fetch All Projects**

  Endpoint:

  GET /api/projects

  Description:

  Retrieves a list of all stored projects.

  Response Example:

```
{
    "status": "success",
    "projects": [
        { "id": 1, "name": "AI-Powered Chatbot" },
        { "id": 2, "name": "Machine Learning Stock Predictor" }
    ]
}
```

  API Code:

```
// GET - Fetch all projects
app.get("/api/projects", (req, res) => {
    res.json({ status: "success" , projects});
});
```

- **GET - Fetch Total Number of Projects**

  Endpoint:

  GET /api/projects/count

  Description:

  Returns the total number of projects available.

Response Example:

```
{
    "status": "success",
    "count": 5
}
```

API Code:

```
// GET - Fetch count (number of projects)
app.get("/api/projects/count", (req, res) => {
    res.json({ status: "success" , count: projects.length});
});
```

- **POST - Add a New Project**

  Endpoint:

  POST /api/projects

  Description:

  Creates and adds a new project to the list.

  Request Body:

```
{
    "name": "Project Gamma"
}
```

  Response Example (Success):

```
{
    "status": "success",
    "project": { "id": 3, "name": "Project Gamma" }
}
```

  Response Example (Error – Name Too Short):

```
{
    "status": "error",
    "message": "Project name must be at least 3 characters"
}
```

Response Example (Error – Duplicate Name):

```
{
    "status": "error",
    "message": "Project name already exists"
}
```

API Code:

```
// POST - Add a new project
app.post("/api/projects", (req, res) => {
    const { name } = req.body;
    if (!name || name.length < 3) {
        return res.status(400).json({ status: "error", message: "Project
name must be at least 3 characters" });
    }

    // Prevent duplicate names (case-insensitive check)
    if (projects.some(proj => proj.name.toLowerCase() ===
name.toLowerCase())) {
        return res.status(400).json({ status: "error", message: "Project
name already exists" });
    }

    // Assign a unique ID
    const newProjectId = projects[projects.length-1].id + 1;
    const newProject = { id: newProjectId, name };
    projects.push(newProject);
    writeDataToFile();

    res.status(201).json({ status: "success", project: newProject });
});
```

- **PUT - Update a Project**

  Endpoint:

  PUT /api/projects/:id

  Description:

  Modifies an existing project's name.

  Request Body:

```
{
    "name": "Updated Project Name"
}
```

  Response Example (Success):

```
{
    "status": "success",
    "project": { "id": 2, "name": "Updated Project Name" }
}
```

  Response Example (Error – Page Not Found):

```
{
    "status": "error",
    "message": "Project not found"
}
```

  Response Example (Error – Name Too Short):

```
{
    "status": "error",
    "message": "Project name must be at least 3 characters"
}
```

  API Code:

```
// PUT - Update a project
app.put("/api/projects/:id", (req, res) => {
    const { id } = req.params;
    const{ name } = req.body;

    if (!name || name.length < 3) {
```

```
        return res.status(400).json({ status: "error", message: "Project
name must be atleast 3 characters" });
    }

    const project = projects.find(p => p.id == id);

    if (!project) {
        return res.status(404).json({ status: "error", message: "Project
not found"});
    }

    project.name = name;
    writeDataToFile();
    res.json({ status: "success", project });
});
```

- **DELETE - Remove a Project**

  Endpoint:

  DELETE /api/projects/:id

  Description:

  Deletes a project by its ID and returns the deleted project details.

  Response Example (Success):

```
{
    "status": "success",
    "deletedProject": { "id": 2, "name": "Project Beta" }
}
```

  Response Example (Error – Page Not Found):

```
{
    "status": "error",
    "message": "Project not found"
}
```

API Code:

```
// DELETE - Remove a project and return details
app.delete("/api/projects/:id", (req, res) => {
    const { id } = req.params;
    const projectIndex = projects.findIndex(p => p.id == id);

    if (projectIndex === -1){
        return res.status(404).json({ status: "error", message: "Project
not found" });
    }

    const deletedProject = projects.splice(projectIndex, 1)[0];
    writeDataToFile();
    res.json({ status: "success", deletedProject });
});
```

FRONTEND IMPLEMENTATION:

The frontend of the project is built using **React.js** and interacts with the backend API to perform CRUD operations. Below is an explanation of how different parts of the UI work:

Fetching Projects (useEffect Hook):

- When the component mounts, useEffect() fetches the project list from the backend.
- The fetchProjects() function sends a GET request to retrieve project data and update the UI.
- Similarly, fetchProjectCount() retrieves the total number of projects.

```
import { useState, useEffect } from "react";
import "./App.css";

function ProjectManager() {
  const [projects, setProjects] = useState([]);     // Stores project list
(intially empty)
  const [loading, setLoading] = useState(true);     // Tracks loading state
  const [projectCount, setProjectCount] = useState(0);     // Project Count
  const [newProjectName, setNewProjectName] = useState("");     // Stores new
project name
```

```
  const [editProjectId, setEditProjectId] = useState(null);     // Stores
updated project id
  const [editProjectName, setEditProjectName] = useState("");     // Stores
updated project name
  const [deletedProject, setDeletedProject] = useState(null);     // Stores
deleted

  const fetchProjects = () => {
    setLoading(true);
    fetch("http://localhost:5000/api/projects")
      .then((response) => response.json())
      .then((data) => {
        setProjects(data.projects);
        setLoading(false);
      })
      .catch((error) => {
        console.error("Error fetching projects:", error);
        setLoading(false);
      });
  };

  const fetchProjectCount = () => {
    fetch("http://localhost:5000/api/projects/count")
      .then((response) => response.json())
      .then((data) => {
        setProjectCount(data.count);
      })
      .catch((error) => console.error("Error fetching project count:", error));
  };

  useEffect(() => {
    fetchProjects();
    fetchProjectCount();
  }, []);

  const handleRefresh = () => {
    fetchProjects();
    fetchProjectCount();
  };
```

Displaying Projects:

- The table dynamically displays projects stored in projects.
- If no projects are available, a message is displayed instead.

```
return (
  <div className="main-container">
    <h1 className='main-heading'>Project List</h1>
    <h3 className='project-count'>Total Projects: {projectCount}</h3>  {/*
Display total project count */}

    <div className='btn-section'>
      {/* Refresh Button */}
      <button onClick={handleRefresh} className='btn'>Refresh Projects</button>

      {/* Add New Project */}
      <div className='add-project'>
        <h2>Add Project</h2>
        <input
          type="text"
          placeholder="Enter project name"
          value={newProjectName}
          onChange={(e) => setNewProjectName(e.target.value)}
          className='input-project'
        />
        <button onClick={addProject} className='btn'>Add Project</button>
      </div>
    </div>


    {/* Loading Message */}
    {loading ? (
      <p>Loading...</p>
    ) : (
      <table border="1">
        <thead className='table-head'>
          <tr>
            <th className='table-pro-id'>ID</th>
            <th className='table-pro-name'>Project Name</th>
            <th className='table-pro-actions'>Actions</th>
          </tr>
        </thead>
        <tbody>
          {projects.length > 0 ? (
            projects.map((project) => (
              <tr key={project.id}>
                <td className='table-pro-id'>{project.id}</td>
                <td className='table-pro-name'>
                  {editProjectId === project.id ? (
                    <input
                      type="text"
```

```
                    value={editProjectName}
                    onChange={(e) => setEditProjectName(e.target.value)}
                    className='input-project'
                  />
                ) : (
                  project.name
                )}
              </td>
              <td className='table-pro-actions'>
                {editProjectId === project.id ? (
                  <div className='btn-section-2'>
                    <button onClick={() => updateProject(project.id)}
className='btn'>Save</button>
                    <button onClick={() => setEditProjectId(null)}
className='btn'>Cancel</button>
                  </div>
                ) : (
                  <div className='btn-section-2'>
                    <button onClick={() => { setEditProjectId(project.id);
setEditProjectName(project.name); }} className='btn'>
                      Update
                    </button>
                    <button onClick={() => deleteProject(project.id)}
className='btn'>Delete</button>
                  </div>
                )}
              </td>
            </tr>
          ))
        ) : (
          <tr>
            <td colSpan="3" className='table-pro-actions'>No projects
available</td>
          </tr>
        )}
      </tbody>
    </table>
  )}

  {/* Display Deleted Project Message */}
  {deletedProject && (
      <div className="deleted-message">
        <p>
          ❌ Deleted Project: <strong>{deletedProject.name}</strong> (ID:
{deletedProject.id})
        </p>
      </div>
  )}

</div>
);
```

Handling CRUD Operations (Add, Edit, Delete):

Adding a New Project

- The addProject() function sends a **POST request** to the backend.
- If the request is successful, the project is added to the UI.

```
// Add a new project
  const addProject = () => {
    if (newProjectName.trim().length < 3) {
      alert("Project name must be atleast 3 characters!");
      return;
    }

    fetch("http://localhost:5000/api/projects", {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({ name: newProjectName }),
    })
      .then((response) => response.json())
      .then((data) => {
        if (data.status === "success") {
          setProjects([...projects, data.project]);
          setNewProjectName("");
          setProjectCount(projectCount+1);
        }

        else {
          alert(data.message);
        }
      })
      .catch((error) => console.error("Error adding project:", error));
  };
```

Updating an Existing Project

- The updateProject() function sends a **PUT request** with the new project name.
- If successful, the UI updates the project name.

```
// Update an existing project
  const updateProject = (id) => {
    if (editProjectName.trim().length < 3) {
      alert("Project name must be atleast 3 characters!");
      return;
    }

    fetch(`http://localhost:5000/api/projects/${id}`, {
      method: "PUT",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({ name: editProjectName }),
```

```
    })
      .then((response) => response.json())
      .then((data) => {
        if (data.status === "success") {
          setProjects(
            projects.map((project) =>
              project.id === id ? { ...project, name: editProjectName } :
project
            )
          );

          setEditProjectId(null);
          setEditProjectName("");
        }
        else {
          alert(data.message);
        }
      })
      .catch((error) => console.error("Error updating project:", error));
  };
```

Deleting a Project

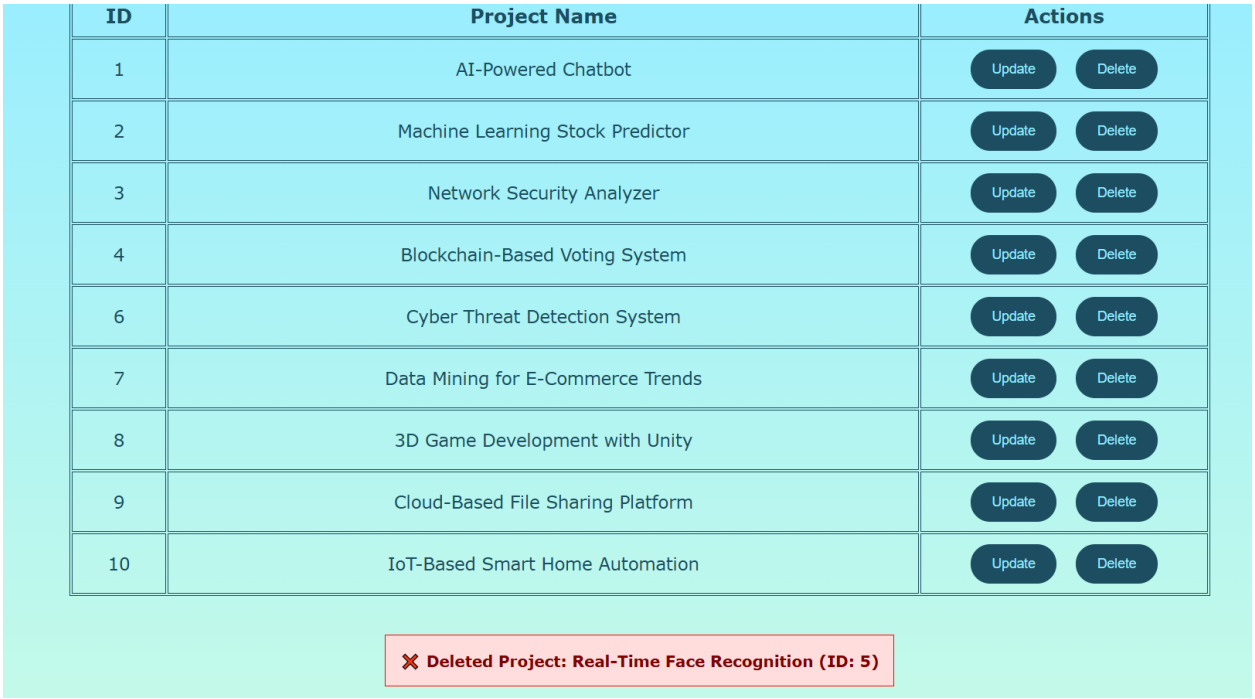
- The deleteProject() function sends a **DELETE request**.
- If successful, the project is removed from the UI, and a deleted message is shown.

```
// Delete a project
  const deleteProject = (id) => {
    fetch(`http://localhost:5000/api/projects/${id}`, {
      method: "DELETE",
    })
      .then((response) => response.json())
      .then((data) => {
        if (data.status === "success") {
          setDeletedProject(data.deletedProject); // Store deleted
project info
          setProjects(projects.filter((project) => project.id !== id));
          setProjectCount(projectCount - 1);
          // Clear the deleted project message after 5 seconds
          setTimeout(() => {
            setDeletedProject(null);
          }, 5000);
        }
        else {
          alert("Error deleting project");
        }
      })
      .catch((error) => console.error("Error deleting project:", error));
  };
```

SCREENSHOTS & UI OVERVIEW:

UI Overview

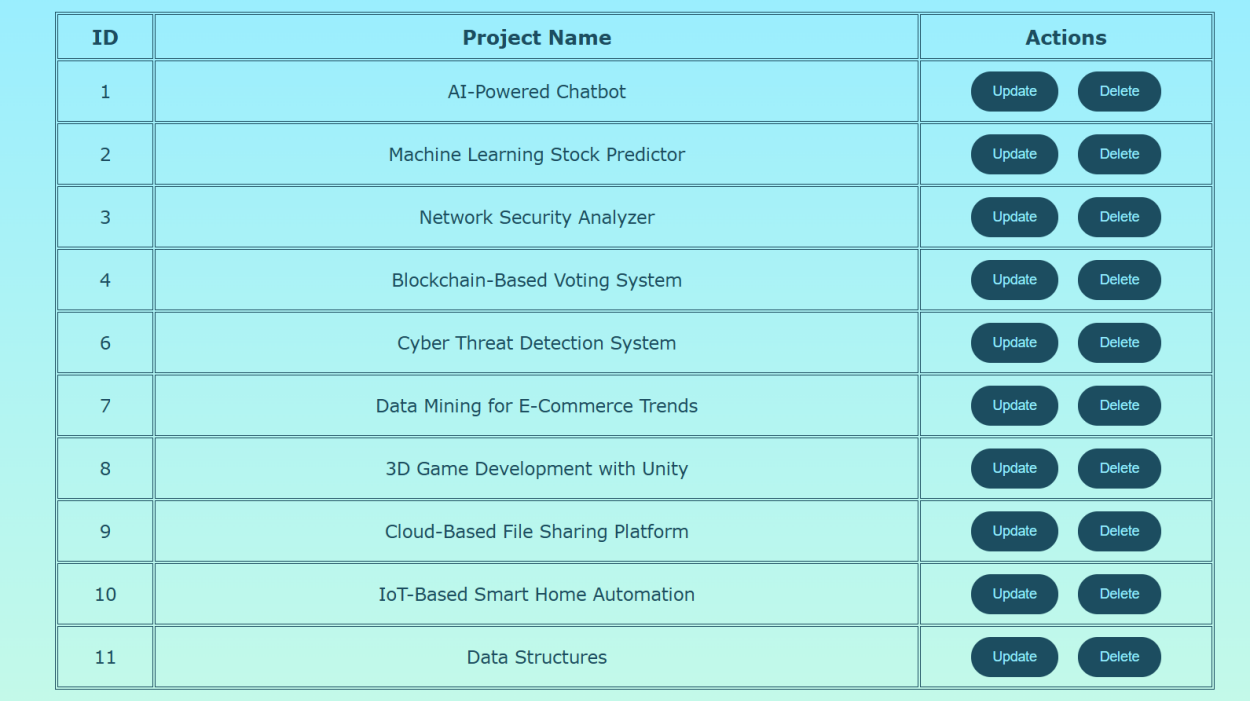
DELETING A PROJECT:



Deleted Project with Id: 5

ADDING A NEW PROJECT:

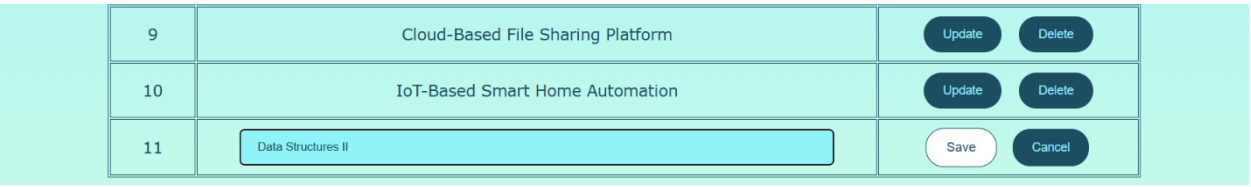
Adding New Project



ID	Project Name	Actions
1	AI-Powered Chatbot	Update Delete
2	Machine Learning Stock Predictor	Update Delete
3	Network Security Analyzer	Update Delete
4	Blockchain-Based Voting System	Update Delete
6	Cyber Threat Detection System	Update Delete
7	Data Mining for E-Commerce Trends	Update Delete
8	3D Game Development with Unity	Update Delete
9	Cloud-Based File Sharing Platform	Update Delete
10	IoT-Based Smart Home Automation	Update Delete
11	Data Structures	Update Delete

Project is added in the last

UPDATING A PROJECT:



Updating the last project

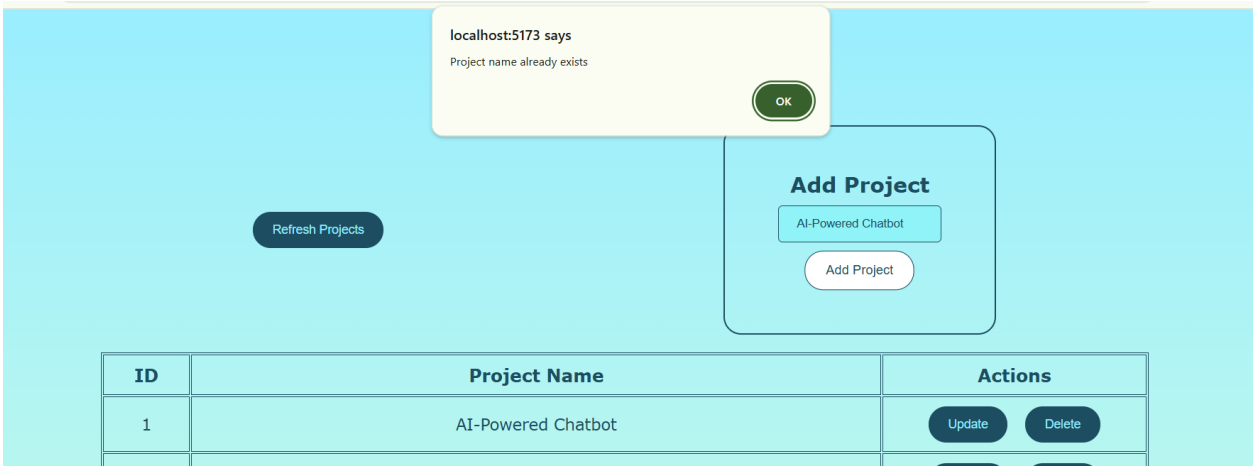
Changing name from Data Structures to Data Structures II

ID	Project Name	Actions
1	AI-Powered Chatbot	Update Delete
2	Machine Learning Stock Predictor	Update Delete
3	Network Security Analyzer	Update Delete
4	Blockchain-Based Voting System	Update Delete
6	Cyber Threat Detection System	Update Delete
7	Data Mining for E-Commerce Trends	Update Delete
8	3D Game Development with Unity	Update Delete
9	Cloud-Based File Sharing Platform	Update Delete
10	IoT-Based Smart Home Automation	Update Delete
11	Data Structures II	Update Delete

Project Name Updated

ERROR:

Adding a Project Name of Length 2



Adding a Project Name which already exists

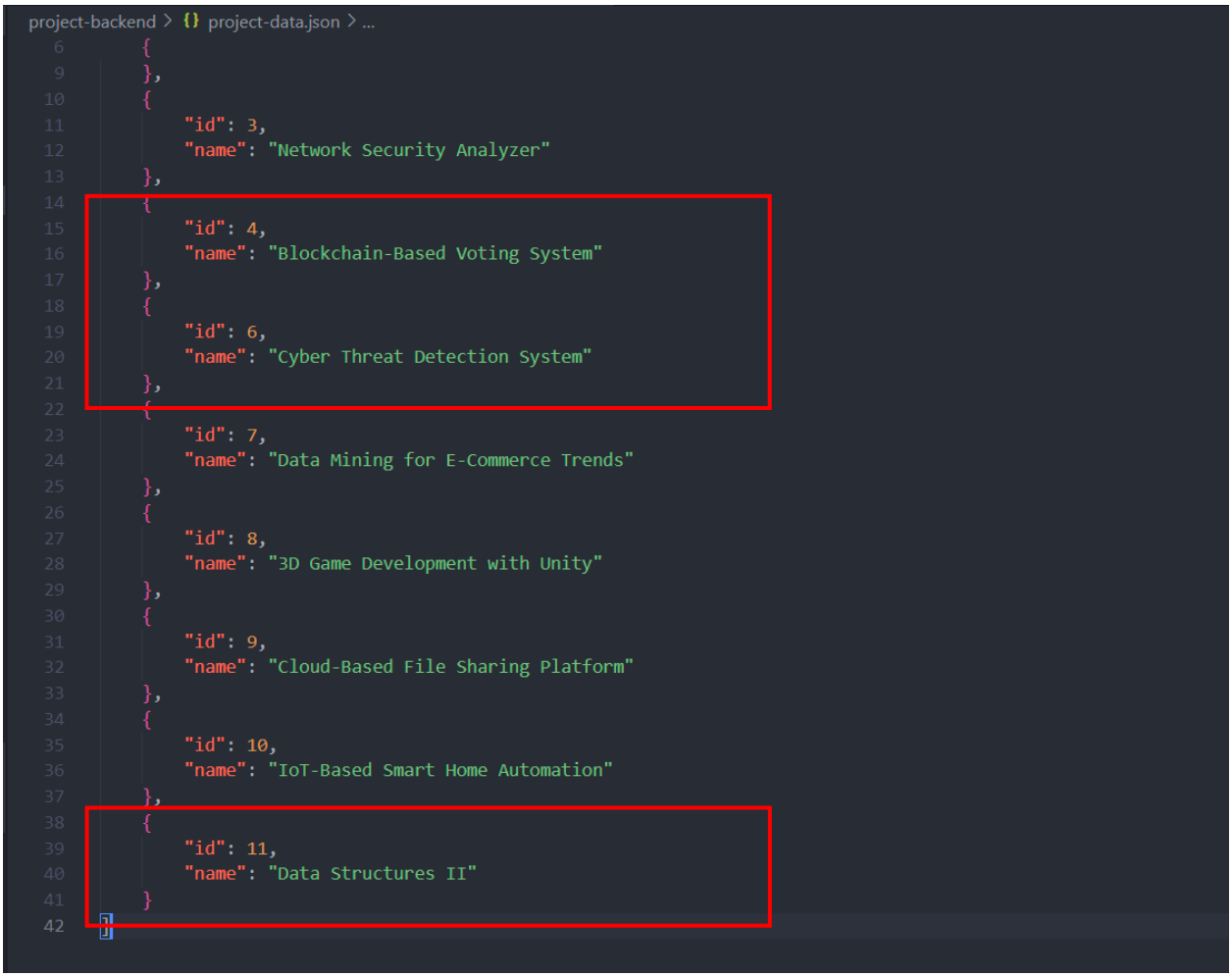
BACKEND OVERVIEW:

BEFORE MANIPULATION:

project-backend > {} project-data.json > ...

```
[
    {
      "id": 1,
      "name": "AI-Powered Chatbot"
    },
    {
      "id": 2,
      "name": "Machine Learning Stock Predictor"
    },
    {
      "id": 3,
      "name": "Network Security Analyzer"
    },
    {
      "id": 4,
      "name": "Blockchain-Based Voting System"
    },
    {
      "id": 5,
      "name": "Real-Time Face Recognition"
    },
    {
      "id": 6,
      "name": "Cyber Threat Detection System"
    },
    {
      "id": 7,
      "name": "Data Mining for E-Commerce Trends"
    },
    {
      "id": 8,
      "name": "3D Game Development with Unity"
    },
    {
      "id": 9,
      "name": "Cloud-Based File Sharing Platform"
    },
    {
      "id": 10,
      "name": "IoT-Based Smart Home Automation"
    }
]
```

AFTER MANIPULATION:



CHALLENGES AND SOLUTIONS:

1) Incorrect JSON Import Statement in Node.js:

ERROR:

```
node:internal/modules/esm/assert:89
        throw new ERR_IMPORT_ASSERTION_TYPE_MISSING(url, validType);
              ^

TypeError [ERR_IMPORT_ASSERTION_TYPE_MISSING]: Module
"file:///C:/Users/PMLS/Desktop/BSCS-633/FULL%20STACK%20PROJECT/project-
backend/project-data.json" needs an import attribute
of type "json"
    at validateAttributes (node:internal/modules/esm/assert:89:15)
    at defaultLoad (node:internal/modules/esm/load:153:3)
```

```
    at async ModuleLoader.load (node:internal/modules/esm/loader:396:7)
    at async ModuleLoader.moduleProvider (node:internal/modules/esm/loader:278:45) {
  code: 'ERR_IMPORT_ASSERTION_TYPE_MISSING'
}
```

SOLUTION:

I tried importing a .json file using ES module syntax without specifying "assert { type: 'json' }". Modify the import statement so the error get resolved:

```
import projects from "./project-data.json" assert { type: "json" };
```

2) Causing Duplicate IDs:

ISSUE:

```
const newProject = { id: projects.length + 1, name };
```

If a project was deleted, a new one could get the **same ID** as the deleted one.

SOLUTION:

```
// Assign a unique ID
const newProjectId = projects[projects.length-1].id + 1;
const newProject = { id: newProjectId, name };
```

Assigns a **unique ID** to a new project by taking the **last project's ID** from the projects array (`projects[projects.length - 1].id`) and incrementing it by **1**, ensuring that each new project gets a sequentially increasing ID.

3) Frontend Changes Not Reflecting in Backend:

ISSUE:

Initially, when manipulating data on the frontend, the changes were only applied to the UI. However, the backend JSON file remained unchanged, leading to inconsistency between the displayed data and the stored data.

SOLUTION:

A function was implemented to ensure that any updates to the data in the frontend were also written back to the backend JSON file. This function is called whenever modifications occur, such as adding, updating, or deleting a project.

IMPLEMENTED FUNCTION:

The following function is responsible for writing the updated data to the backend JSON file:

```
const writeDataToFile = () => {
    // Convert data to JSON string
    const jsonData = JSON.stringify(projects, null, 4); // Pretty format

    // Write JSON data to a file
    fs.writeFile("./project-data.json", jsonData, (err) => {
        if (err) {
            console.error("Error writing to file:", err);
        } else {
            console.log("Data successfully written to projects.json");
        }
    });
};
```

INTEGRATION IN API ROUTES:

To ensure that changes persist in the backend, the function is invoked in key API operations:

```
app.post("/api/projects", (req, res) => {
    // Add logic
    writeDataToFile();
});

app.put("/api/projects/:id", (req, res) => {
    // Update logic
    writeDataToFile();
});

app.delete("/api/projects/:id", (req, res) => {
    // Delete logic
    writeDataToFile();
});
```

This ensures that every change in the frontend is accurately reflected in the backend, maintaining data consistency and preventing potential discrepancies.

FUTURE ENHANCEMENTS:

To further improve this project, the following enhancements can be implemented:

Store projects in MongoDB instead of JSON – Currently, project data is stored in a JSON file, which is not ideal for scalability. Switching to MongoDB will allow us to efficiently manage and retrieve large datasets.

Add user authentication (Login/Register) – Implementing authentication will enable users to manage their projects securely. Features like user registration, login, and session handling will be added.

Implement pagination for large project lists – As the number of projects grows, displaying all of them at once may slow down performance. Pagination will help load and display projects in smaller, manageable sections.

CONCLUSION:

This project provided valuable hands-on experience in **full-stack development**, covering both frontend (React.js) and backend (Node.js with Express). I learned how to integrate a **RESTful API**, manage state in React, and perform **CRUD operations** efficiently.

Future improvements, such as **database integration, authentication, and pagination**, will make the project more scalable and user-friendly. This project has been a great learning experience in **web development and API integration**.