International Islamic University Chittagong

Department of Computer Science and Engineering

LAB REPORT

Course title	:	Software Engineering Sessional & Software Development 2
Course Code	:	**CSE-3638 & 3640**
Report No	:	**08**
Report Title	:	***Building a Full-Stack Application***

Submitted By

Name	:	Ariful Hasan Adil
ID No	:	**C223112**
Section	:	**6CM**
Semester	:	6th

Submitted To

Mohammad Arfizurrahman
Adjunct Faculty
Department of CSE, IIUC

Submission Date : / /2025

Overview

Developing a full-stack application means building a comprehensive software system that includes both the **frontend**—the part users see and interact with—and the **backend**, which manages data, server logic, and communication with databases and APIs.

🎯 Objective

The goal of this lab is to design and implement a fully functional full-stack application. This involves:

- Crafting an intuitive and responsive **user interface** using frontend technologies
- Building robust **server-side logic** to handle requests, process data, and manage interactions with a database
- Ensuring seamless **API communication** between the frontend and backend components

🛠 Tools & Environment Setup

To successfully build and test the application, you'll use the following tools:

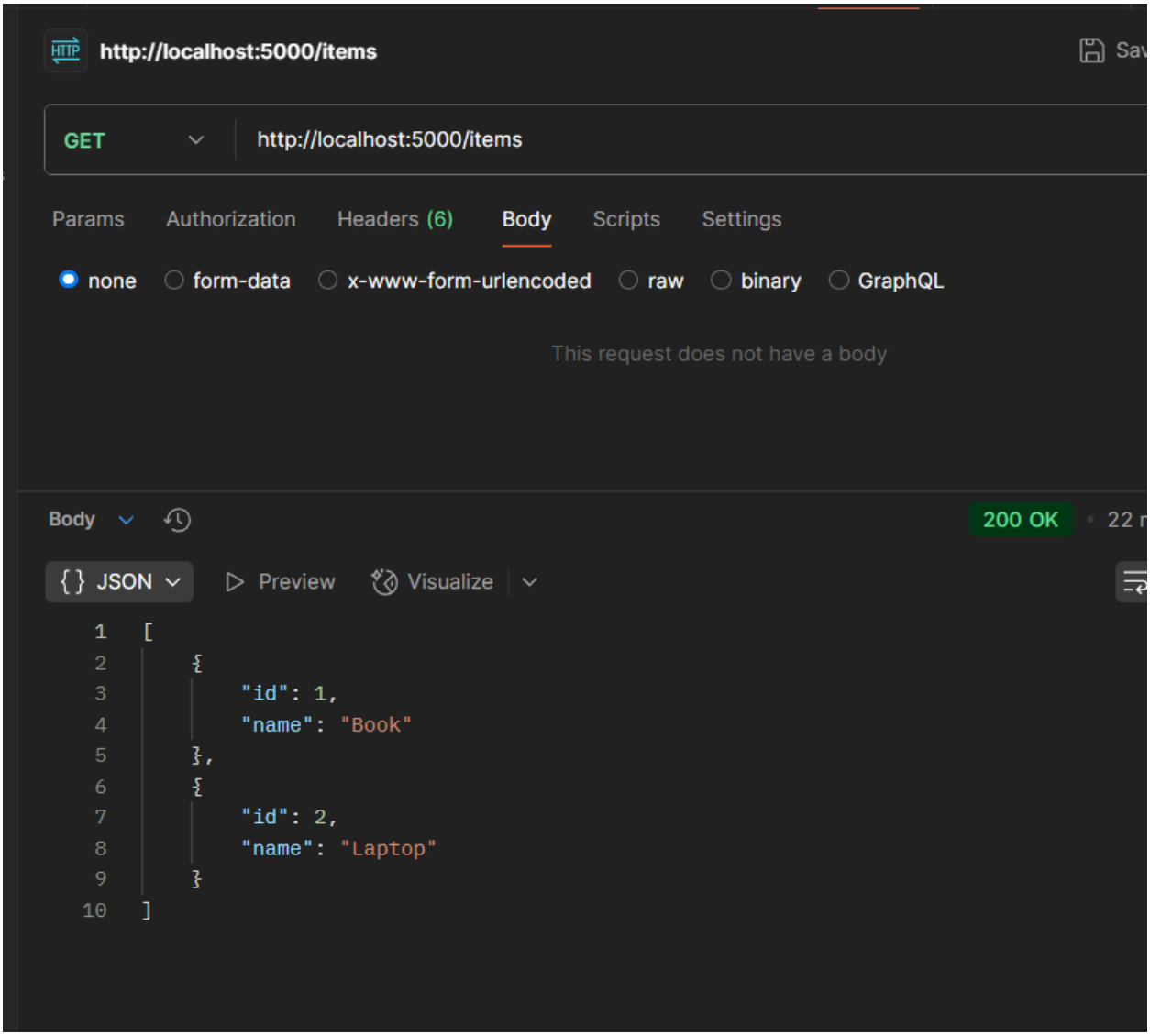
- **Backend Framework**: Express.js for creating the server and handling API routes
- **Frontend Framework**: React (initialized with Vite for fast development and optimized builds)
- • **API Testing**: Tools like Postman or browser console to test endpoints and debug requests

- **Code Editor**: Visual Studio Code (VS Code) for writing and managing your codebase efficiently

Lab Tasks:

1. Backend API (Reuse Previous Work or Use a Sample)

- Ensure your backend is running and exposes basic CRUD routes (e.g., `/items`, `/items/:id`)
- Test the API endpoints using Postman or browser



PUT http://localhost:5000/items/1

Params Authorization Headers (8) Body Scripts Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {
2   "id": 1,
3   "name": "Updated Item 1"
4 }
5
6
```

Body Cookies Headers (8) Test Results 200 OK 15 ms

{} JSON Preview Visualize

```
1 {
2     "id": 1,
3     "name": "Updated Item 1"
4 }
```

Overview Getting st POST http://lo GET http://loc PUT http://loc DEL http://loc No environme

http://localhost:5000/items/1 Save Share

DELETE http://localhost:5000/items/1 Send

Params Authorization Headers (6) Body Scripts Settings

Query Params

Key	Value	Description
Key	Value	Description

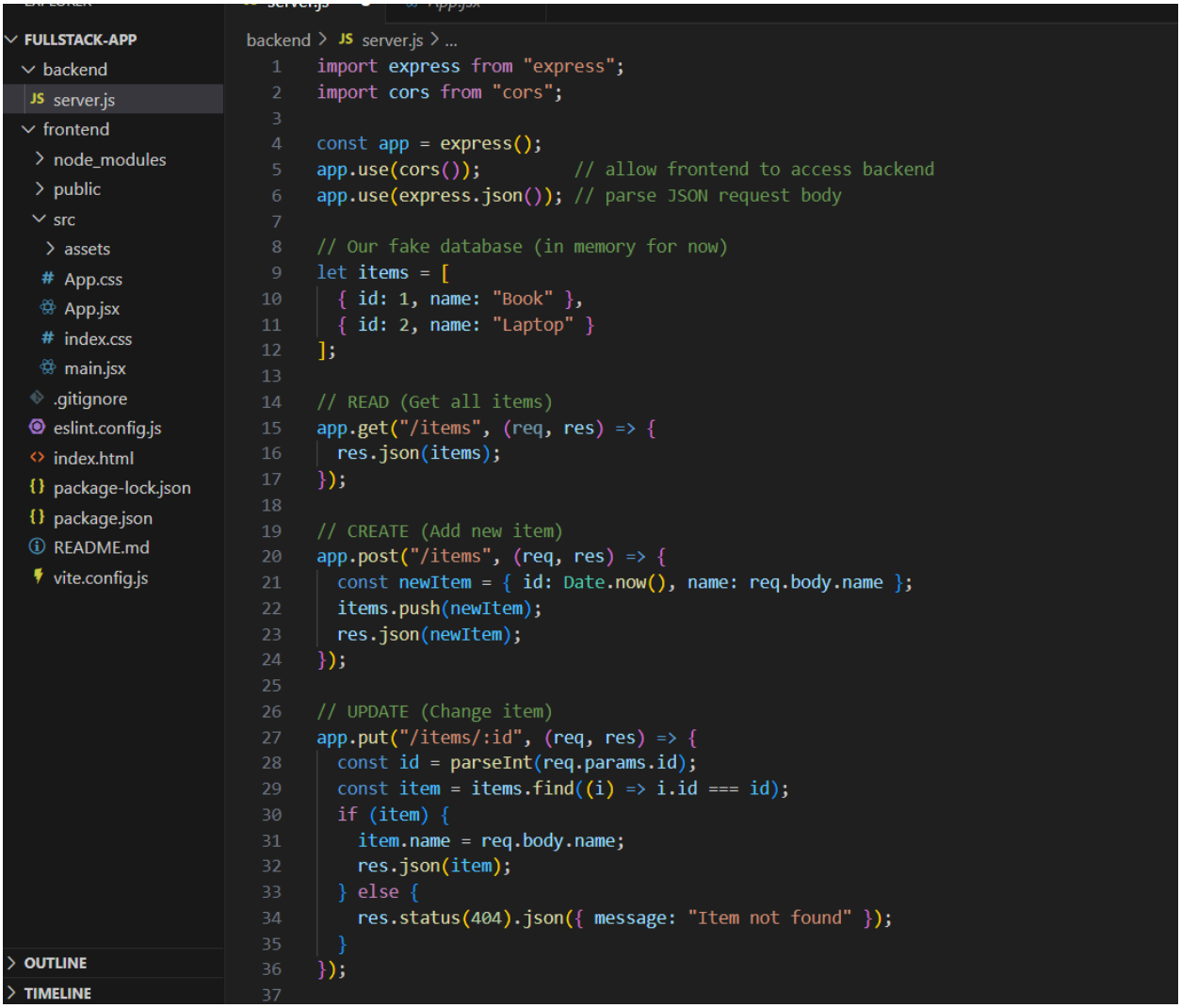
Body Cookies Headers (8) Test Results 200 OK 16 ms 301 B

{} JSON Preview Visualize

```
1 {
2     "message": "Deleted successfully"
3 }
```

2. Connect React Frontend to API

- In your React project:
  - Use `fetch` or `axios` to call backend API endpoints
  - Use `useEffect` to fetch data when the component loads
  - Store and manage data using `useState`



EXPLORER

FULLSTACK-APP
- backend
  - node_modules
  - package-lock.json
  - package.json
  - server.js
- frontend
  - node_modules
  - public
    - vite.svg
  - src
    - assets
    - App.css
    - App.jsx
    - index.css
    - ItemTable.jsx
    - main.jsx
  - .gitignore
  - eslint.config.js
  - index.html
  - package-lock.json
  - package.json
  - README.md
  - vite.config.js

server.js | package.json | App.jsx | ItemTable.jsx | main.jsx

frontend > src > App.jsx > App > addItem

```jsx
import React, { useEffect, useState } from "react";
import axios from 'axios';
import ItemTable from './ItemTable';

const API_BASE = "http://localhost:5000";
function App() {
  // States
  const [items, setItems] = useState([]);    // All items
  const [newItem, setNewItem] = useState(""); // Input value
  const [editItem, setEditItem] = useState(null); // Item being edited

  // READ: fetch items from backend
  useEffect(() => {
    fetch("http://localhost:5000/items")
      .then((res) => res.json())
      .then((data) => setItems(data));
  }, []);

  // CREATE: add new item
  function addItem(e) {
    e.preventDefault();
    fetch("http://localhost:5000/items", {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({ name: newItem })
    })
      .then((res) => res.json())
      .then((data) => {
        setItems([...items, data]);
```

PROBLEMS | OUTPUT | DEBUG CONSOLE | TERMINAL | PORTS

```
➜ Local:   http://localhost:5173/
```

EXPLORER

FULLSTACK-APP
- backend
  - node_modules
  - package-lock.json
  - package.json
  - server.js
- frontend
  - node_modules
  - public
    - vite.svg
  - src
    - assets
    - App.css
    - App.jsx
    - index.css
    - ItemTable.jsx
    - main.jsx
  - .gitignore
  - eslint.config.js
  - index.html
  - package-lock.json
  - package.json
  - README.md
  - vite.config.js

server.js | package.json | App.jsx | ItemTable.jsx | main.jsx

frontend > src > ItemTable.jsx > ItemTable > items.map() callback

```jsx
// src/ItemTable.jsx
import React from "react";

function ItemTable({ items, onEdit, onDelete }) {
  return (
    <table border="1" cellPadding="10" style={{ marginTop: "20px", width: "100%" }}>
      <thead>
        <tr>
          <th>ID</th>
          <th>Item Name</th>
          <th>Actions</th>
        </tr>
      </thead>
      <tbody>
        {items.length === 0 ? (
          <tr>
            <td colSpan="3" align="center">No items found</td>
          </tr>
        ) : (
          items.map((item, index) => (
            <tr key={item.id}>
              <td>{index + 1}</td> {/* Serial number starts from 1 */}
              <td>{item.name}</td>
              <td>
                <button onClick={() => onEdit(item)}>✏️ Edit</button>
                <button onClick={() => onDelete(item.id)}>❌ Delete</button>
              </td>
            </tr>
          ))
```

PROBLEMS | OUTPUT | DEBUG CONSOLE | TERMINAL | PORTS

```
➜ Local:   http://localhost:5173/
```

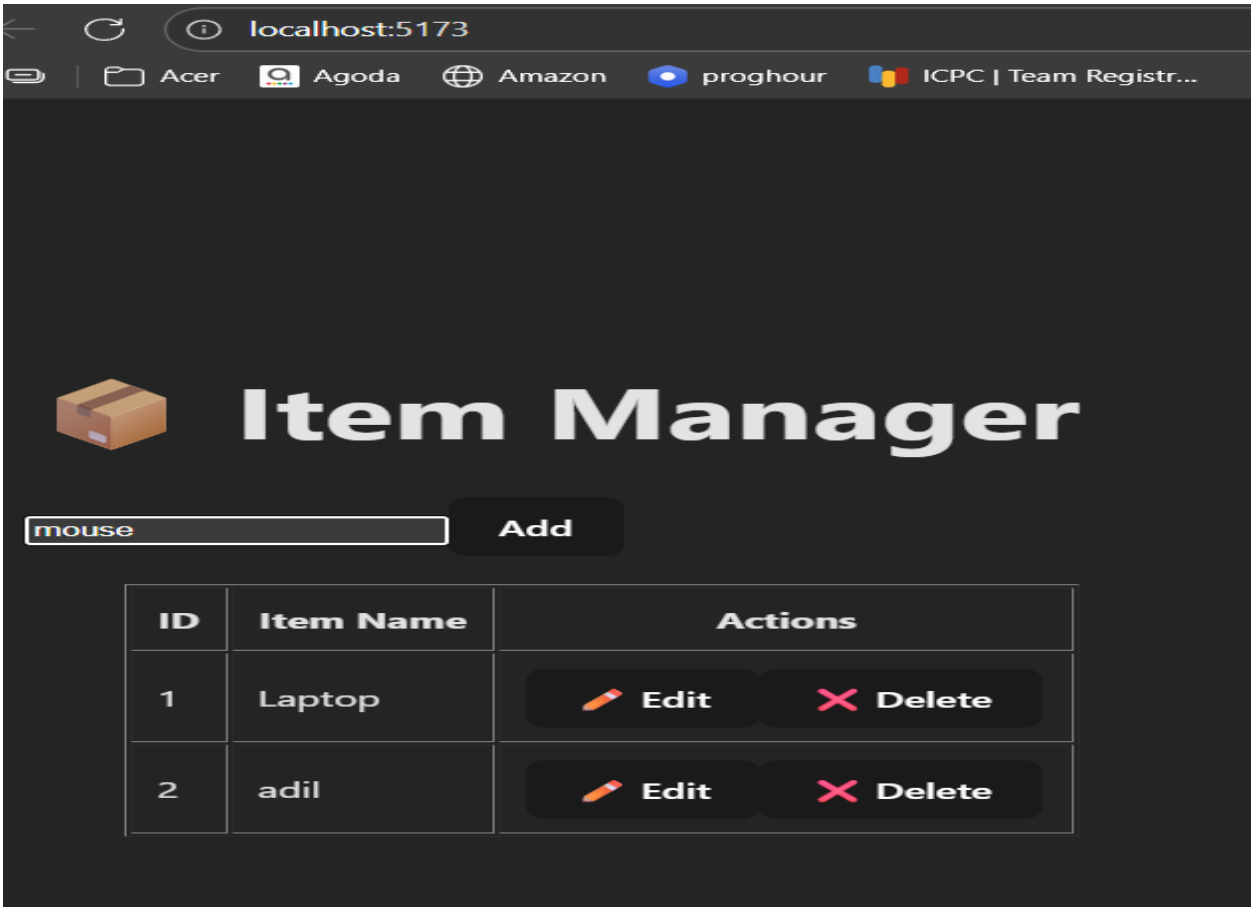
3. Display Data in React

- Create a UI component to list the data items (e.g., a table or cards)
- Display at least one field from each item

4. Implement CRUD Operations

- **Create**: Add a form that submits new data to the backend
- **Read**: Automatically or manually refresh the data after create/update/delete
- **Update**: Add a button to edit an item, update it via an API call
- **Delete**: Add a delete button for each item that triggers an API call

Ensure the UI updates correctly after each operation



Item Manager

Enter item

Add

ID	Item Name	Actions
1	Laptop	Edit Delete
2	adil	Edit Delete
3	mouse	Edit Delete