**LIVE DASHBOARD WITH CLIENT-SERVER ARCHITECTURE**

**COMPUTER NETWORK**

Mini Project Report

Submitted by

**NAME: SIVA KAVINDRA T**

**ROLLNO: 22CDR096**

**NAME: SWADHINI MK**

**ROLLNO: 22CDR107**

**NAME: YESHWANTH M**

**ROLLNO: 22CDR119**

*In partial fulfilment of the requirements*

*for the award of the degree of*

**BACHELOR OF ENGINEERING**

**IN**

**COMPUTER SCIENCE AND DESIGN**

**DEPARTMENT OF COMPUTER SCIENCE AND DESIGN**



**KONGU ENGINEERING COLLEGE**

**(Autonomous)**

**PERUNDURAI, ERODE - 638060**

**LIVE DASHBOARD WITH CLIENT-SERVER ARCHITECTURE**

**AIM:**

The aim of this project is to develop a Live Dashboard utilising a client-server architecture using java script. The Live Dashboard offers real-time insights into the distribution of students and faculty members across various departments. It provides visualizations through Pie Charts and Bar Charts, making it easy to analyse and compare student and faculty data. The frontend is developed using ReactJS for displaying the content and chart.

**SOFTWARE REQUIRED:**

To develop the Live Dashboard, the following software and tools are required:

* **Frontend Technologies**:
  + **ReactJS**: To create the user interface for inputting the department details and viewing the pie chart and the bar chart.
  + **Tail Wind CSS**: For styling the webpage and improving visual appeal.
* **Backend Technologies**:
  + **Programming Language**: JavaScript
  + **Database:** MongoDB (to store the faculty and the student data).
* **Development Environment**:
  + **Web Browser**: To view and test the frontend.
  + **Text Editor or IDE**: To write the code (Visual Studio Code).

**PROGRAM**:

**1). CLIENT SIDE:**

* **App.js**

import React from "react";

import { Dashboard } from "./components/dashboard";

function App() {

  return (

    <div>

      <Dashboard />

    </div>

  );

}

export default App;

* **DashBoard.js**

import React, { useEffect, useState } from 'react';

import { PieChart, Pie, BarChart, Bar, Tooltip, Cell, XAxis, YAxis, CartesianGrid, Legend } from 'recharts';

import axios from 'axios';

import './Dashboard.css';

const COLORS = ['#0088FE', '#00C49F', '#FFBB28', '#FF8042', '#FF6699'];

export const Dashboard = () => {

  const [data, setData] = useState([]);

  const [facultyData, setFacultyData] = useState([]);

  const [newDepartment, setNewDepartment] = useState({

    name: '',

    numberOfStudents: '',

    numberOfFaculty: '',

  });

  const [errorMessage, setErrorMessage] = useState('');

  useEffect(() => {

    const fetchData = async () => {

      try {

        const response = await axios.get('http://localhost:5000/api/departments');

        console.log('Fetched data:', response.data);

        setData(response.data);

        const facultyChartData = response.data.map(dep => ({

          name: dep.name,

          numberOfFaculty: dep.numberOfFaculty || 0,

        }));

        setFacultyData(facultyChartData);

      } catch (error) {

        console.error('Error fetching department data:', error);

      }

    };

    fetchData();

  }, []);

  const handleAddDepartment = async e => {

    e.preventDefault();

if(!newDepartment.name||!newDepartment.numberOfStudents||!newDepartment.numberOfFaculty) {

      setErrorMessage('Please fill out all fields.');

      return;

    }

    try {

      const response = await axios.post('http://localhost:5000/api/departments', newDepartment);

      setData([...data, response.data]);

      const newFacultyData = [

        ...facultyData,

        { name: response.data.name, numberOfFaculty: response.data.numberOfFaculty },

      ];

      setFacultyData(newFacultyData);

      setNewDepartment({ name: '', numberOfStudents: '', numberOfFaculty: '' });

      setErrorMessage('');

    } catch (error) {

      console.error('Error adding department:', error);

      setErrorMessage('Failed to add department. Department might already exist.');

    } };

  const handleInputChange = e => {

    setNewDepartment({

      ...newDepartment,

      [e.target.name]: e.target.value,

    });

  };

  const generateLegendColors = () => {

    return data.map((dep, index) => ({

      name: dep.name,

      color: COLORS[index % COLORS.length],

    }));

  };

  const legendColors = generateLegendColors();

  return (

    <div className="dashboard-container">

      <div className="charts-container">

        <div className="chart-wrapper">

          <h3 className="chart-title">Students Distribution</h3>

          <PieChart width={600} height={400} className="pie-chart">

            <Pie

              data={data}

              cx={300}

              cy={200}

              labelLine={false}

              label={({ name, percent }) => `${name}: ${(percent \* 100).toFixed(0)}%`}

              outerRadius={120}

              fill="#8884d8"

              dataKey="numberOfStudents"

            >

              {data.map((entry, index) => (

                <Cell key={`cell-${index}`} fill={COLORS[index % COLORS.length]} />

              ))}

            </Pie>

            <Tooltip />

          </PieChart>

        </div>

        <div className="chart-wrapper">

          <h3 className="chart-title">Faculty Distribution</h3>

          <PieChart width={600} height={400} className="pie-chart">

            <Pie

              data={facultyData}

              cx={300}

              cy={200}

              labelLine={false}

              label={({ name, percent }) => `${name}: ${(percent \* 100).toFixed(0)}%`}

              outerRadius={120}

              fill="#82ca9d"

              dataKey="numberOfFaculty"

            >

              {facultyData.map((entry, index) => (

                <Cell key={`cell-${index}`} fill={COLORS[index % COLORS.length]} />

              ))}

            </Pie>

            <Tooltip />

          </PieChart>

        </div>

      </div>

      <div className="legend-container">

        {legendColors.map((legend, index) => (

          <div className="legend-item" key={`legend-${index}`}>

            <div className="legend-color" style={{ backgroundColor: legend.color }} />

            <div className="legend-label">{legend.name}</div>

          </div>

        ))}

      </div>

      <div className="charts">

      <div className="p">

      <form className="department-form" onSubmit={handleAddDepartment}>

        <div className="form-group">

          <label htmlFor="name">Department Name</label>

          <input

            type="text"

            id="name"

            name="name"

            value={newDepartment.name}

            onChange={handleInputChange}

            className="form-input"

            placeholder="Enter department name"

          />

        </div>

        <div className="form-group">

          <label htmlFor="numberOfStudents">Number of Students</label>

          <input

            type="number"

            id="numberOfStudents"

            name="numberOfStudents"

            value={newDepartment.numberOfStudents}

            onChange={handleInputChange}

            className="form-input"

            placeholder="Enter number of students"

          />

        </div>

        <div className="form-group">

          <label htmlFor="numberOfFaculty">Number of Faculty</label>

          <input

            type="number"

            id="numberOfFaculty"

            name="numberOfFaculty"

            value={newDepartment.numberOfFaculty}

            onChange={handleInputChange}

            className="form-input"

            placeholder="Enter number of faculty"

          />

        </div>

        {errorMessage && <div className="error-message">{errorMessage}</div>}

        <button type="submit" className="form-submit-btn">

          Add Department

        </button>

      </form>

      </div>

      <div className="bar-chart-container">

        <h3 className="chart-title">Department Data</h3>

        <BarChart width={600} height={400} data={data}>

          <CartesianGrid strokeDasharray="3 3" />

          <XAxis dataKey="name" />

          <YAxis />

          <Tooltip />

          <Legend />

          <Bar dataKey="numberOfStudents" fill="#8884d8" />

          <Bar dataKey="numberOfFaculty" fill="#82ca9d" />

        </BarChart>

      </div>

    </div>

    </div>

  );

};

export default Dashboard

**2). SERVER SIDE:**

const express = require('express');

const mongoose = require('mongoose');

const cors = require('cors');

const app = express();

const PORT = process.env.PORT || 5000;

app.use(express.json());

app.use(cors());

mongoose.connect('mongodb://localhost:27017/college', {

  useNewUrlParser: true,

  useUnifiedTopology: true,

})

.then(() => console.log('MongoDB connected successfully'))

.catch(err => console.error('MongoDB connection error:', err));

const departmentSchema = new mongoose.Schema({

  name: String,

  numberOfStudents: Number,

  numberOfFaculty: Number,

});

const Department = mongoose.model('Department', departmentSchema);

app.get('/api/departments', async (req, res) => {

  try {

    const departments = await Department.find();

    res.json(departments);

  } catch (error) {

    console.error('Error fetching departments:', error);

    res.status(500).json({ error: 'Failed to fetch departments' });

  }

});

app.post('/api/departments', async (req, res) => {

  const { name, numberOfStudents, numberOfFaculty } = req.body;

  try {

    let department = await Department.findOne({ name });

    if (department) {

      return res.status(400).json({ message: 'Department already exists' });

    }

    department = new Department({ name, numberOfStudents, numberOfFaculty });

    await department.save();

    res.status(201).json(department);

  } catch (error) {

    console.error('Error adding department:', error);

    res.status(500).json({ error: 'Failed to add department' });

  }

});

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

  const { numberOfStudents, numberOfFaculty } = req.body;

  try {

    const department = await Department.findById(req.params.id);

    if (!department) {

      return res.status(404).json({ message: 'Department not found' });

    }

    if (numberOfStudents !== undefined) department.numberOfStudents = numberOfStudents;

    if (numberOfFaculty !== undefined) department.numberOfFaculty = numberOfFaculty;

    await department.save();

    res.json(department);

  } catch (error) {

    console.error('Error updating department:', error);

    res.status(500).json({ error: 'Failed to update department' });

  }

});

app.listen(PORT, () => {

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

});

**CONCEPTUAL ANALYSIS:**

A Live Dashboard is an interactive platform that provides real-time visualization of data, offering instant insights through dynamic charts, graphs, and metrics. It is designed to present data updates without requiring manual refreshes, ensuring users always have the most recent information.

In a client-server architecture, the frontend (client) fetches data from the backend (server) through API calls, while the backend retrieves the latest data from a database. The dashboard can display various types of visualizations, such as Pie Charts for proportion comparisons and Bar Charts for category-wise breakdowns, facilitating quick decision-making.

**CLIENT-SERVER WORKING MODEL**

The client-server architecture ensures smooth interaction between the frontend (client) and backend (server) to display real-time student and faculty distribution data using visual elements like Pie Charts and Bar Charts.

**Flow (Client-side):**

1. User loads the dashboard.
2. Dashboard sends a request to the backend to fetch the latest data.
3. Pie Chart and Bar Chart components render based on the received data.
4. User interactions (like adding department) dynamically update the charts.

**Flow (Server-side):**

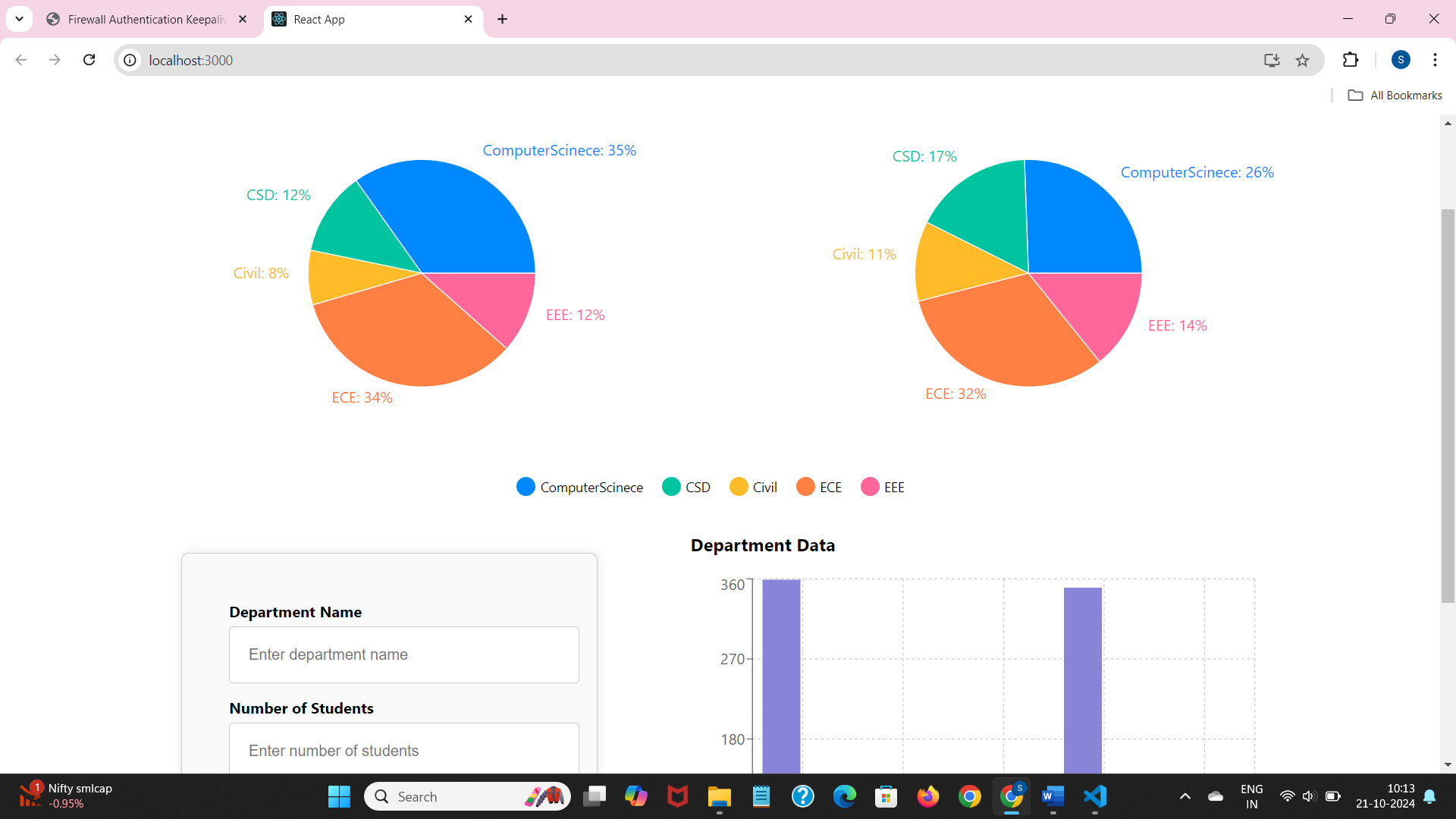
1. The server receives a request from the frontend to fetch distribution data.
2. Database queries fetch the latest counts of students and faculty by department.
3. Server sends a JSON response containing the required data to the frontend.

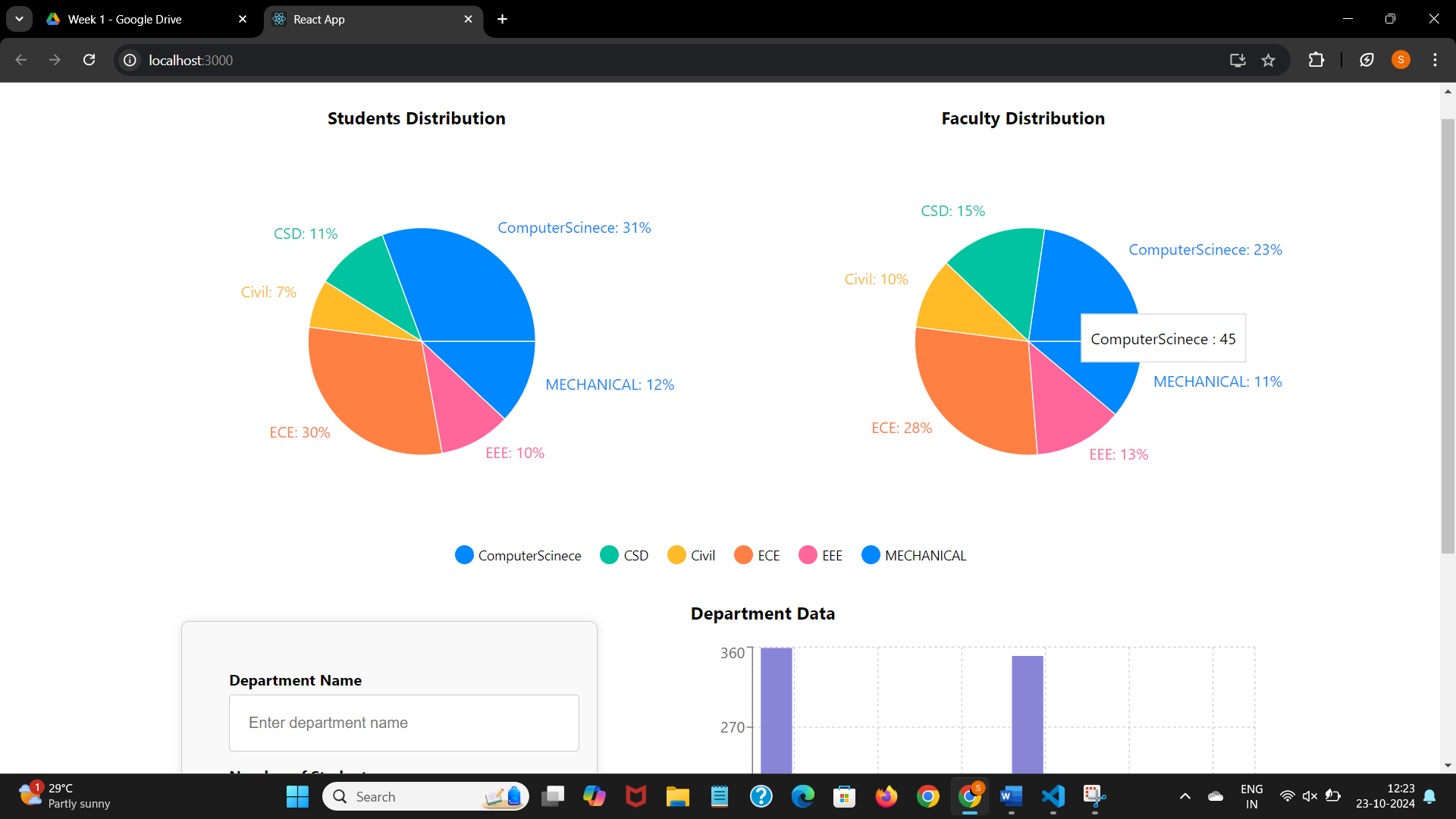
**Flow (Database):**

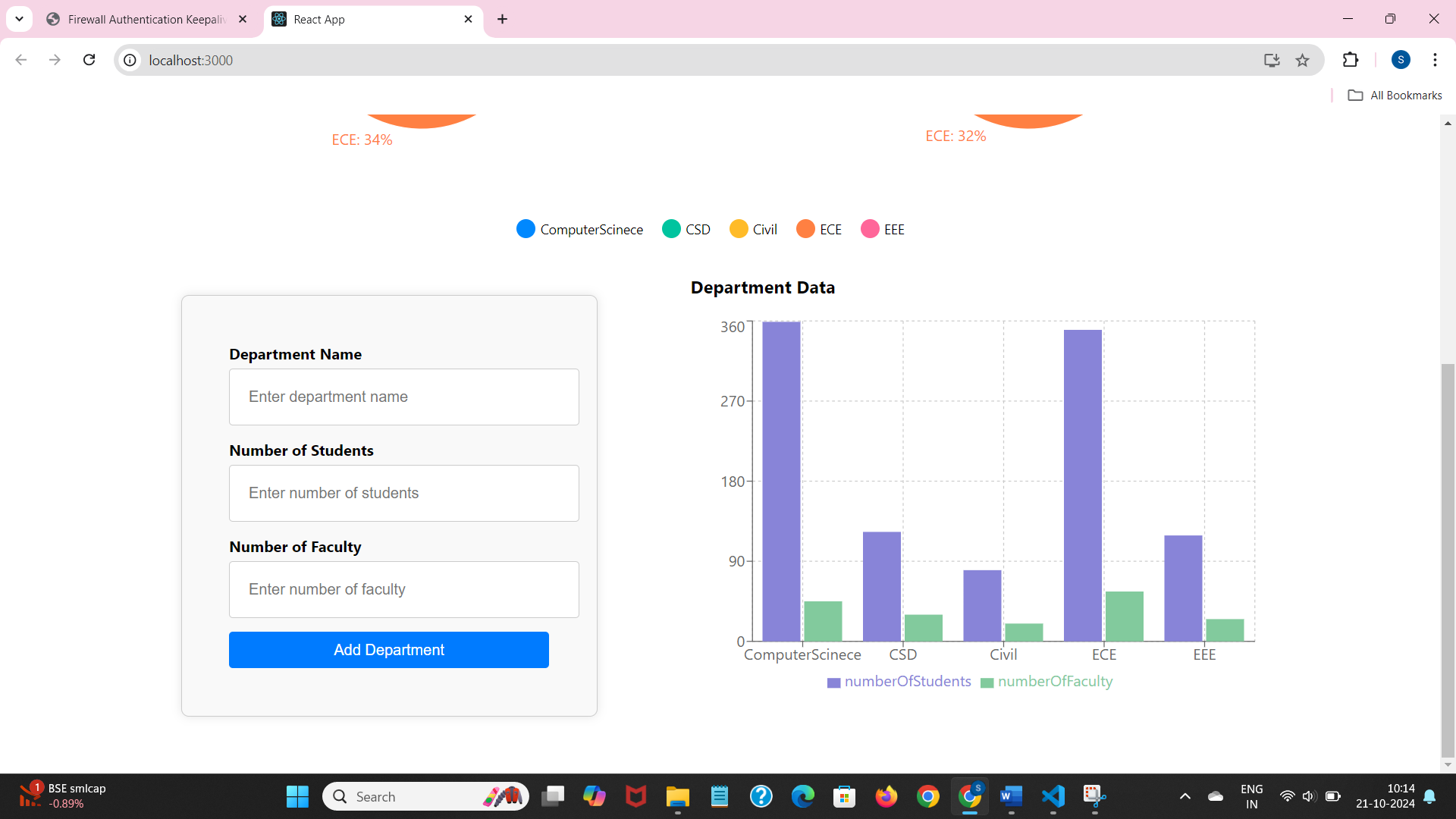
1. Data is inserted or updated in the database periodically at the beginning of each semester.
2. The backend queries the database to fetch the latest data upon request from the frontend.

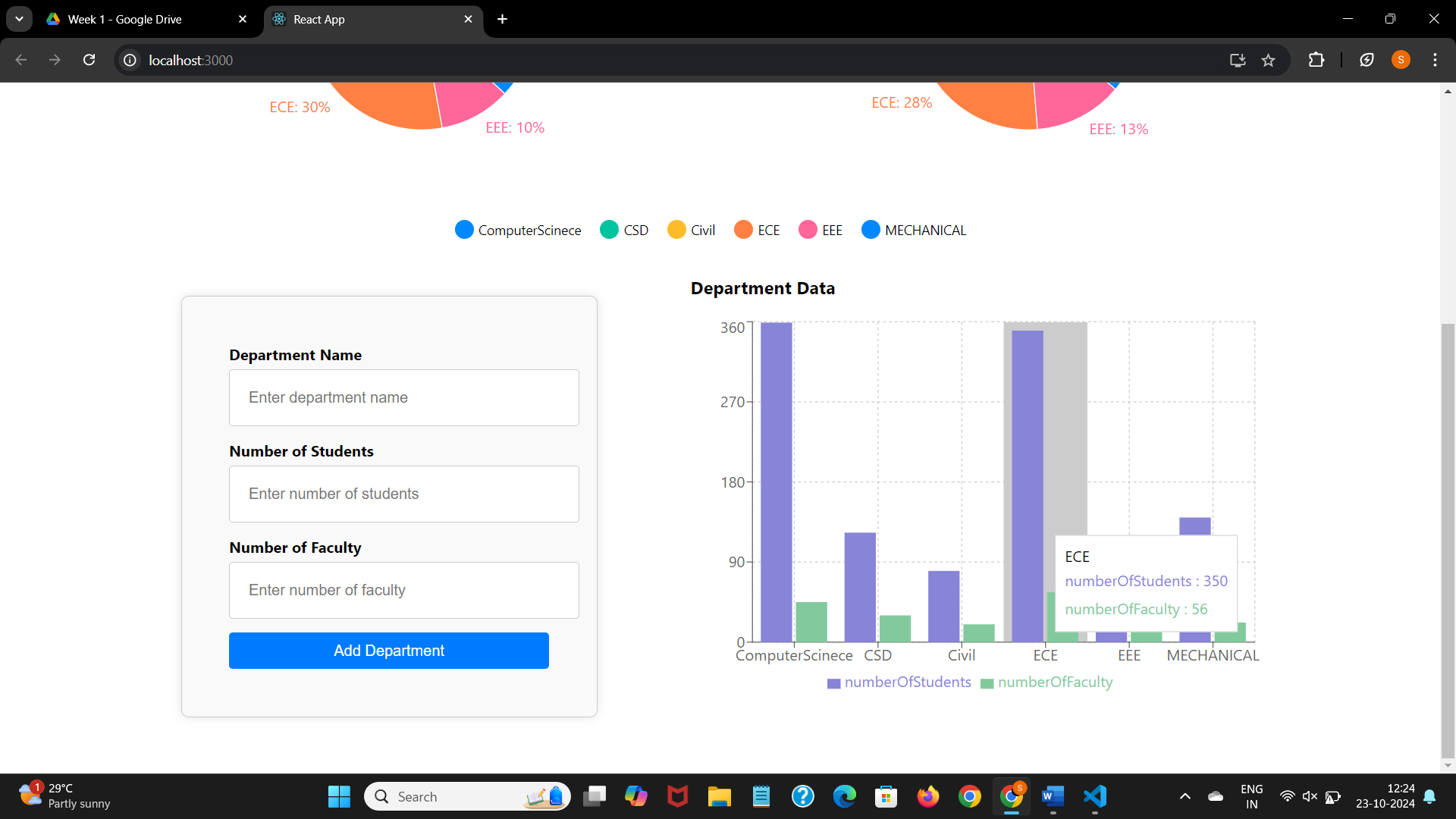
**OUTPUT**:

**FRONTEND WEB DESIGN:**

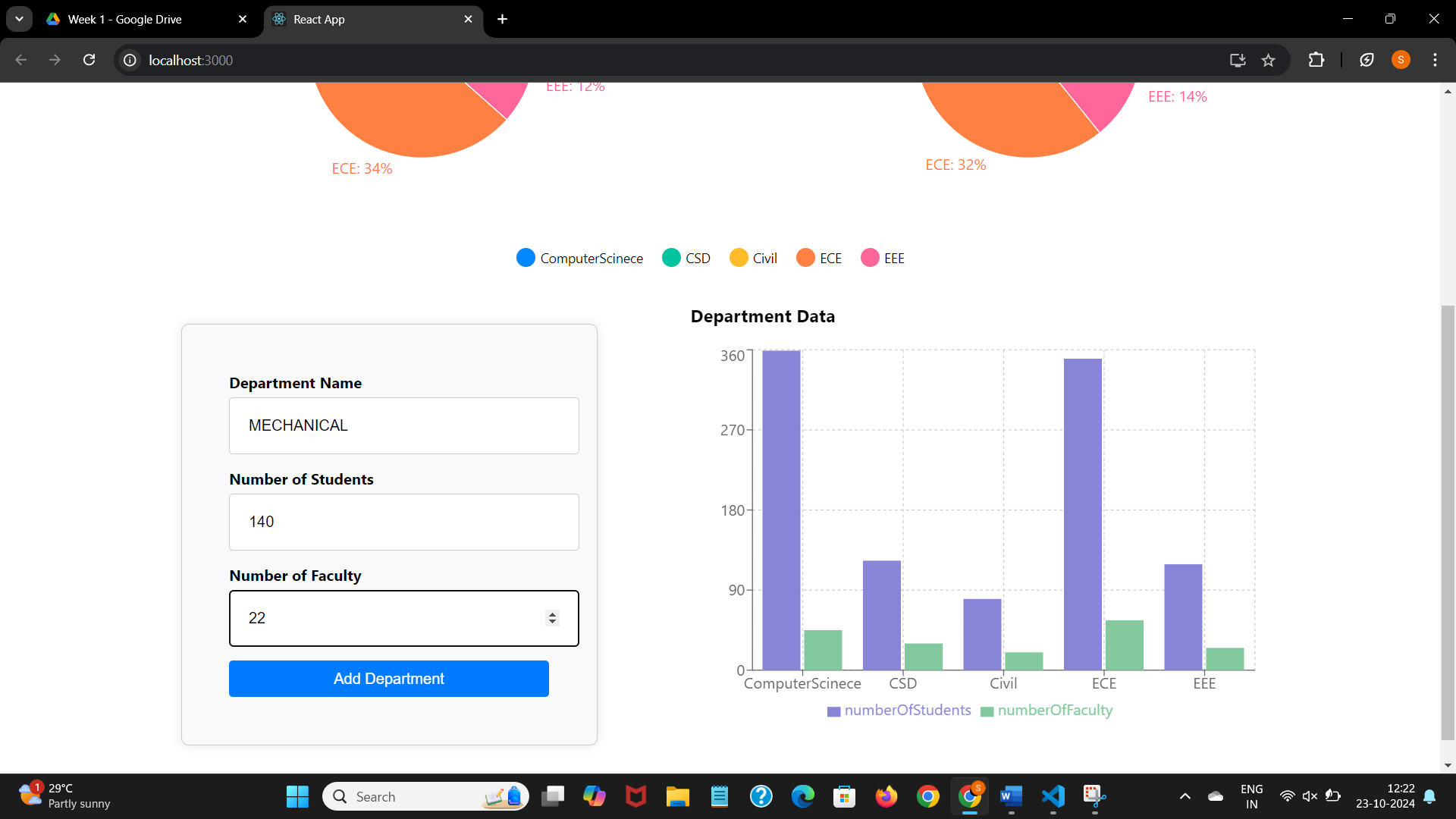




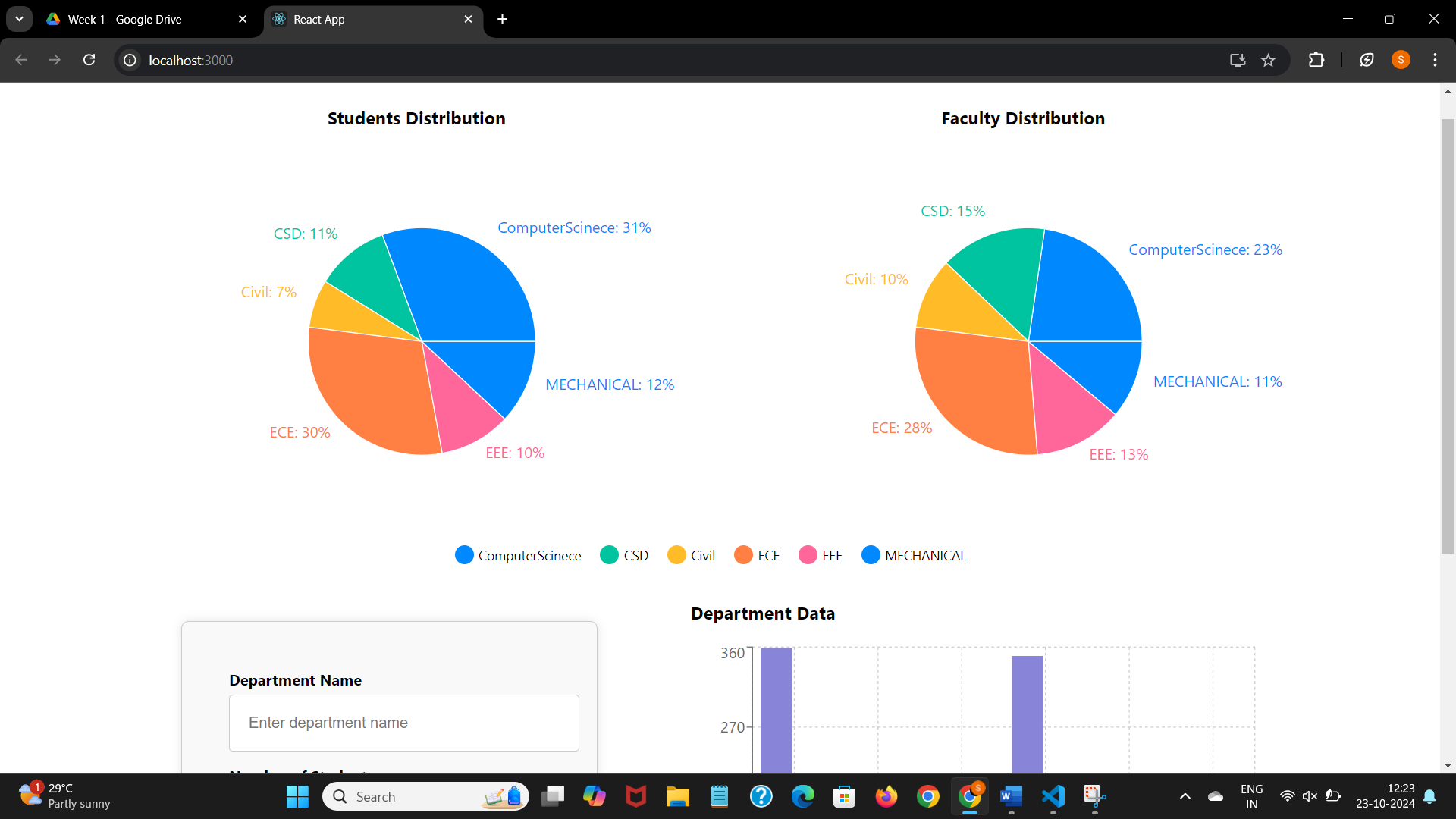




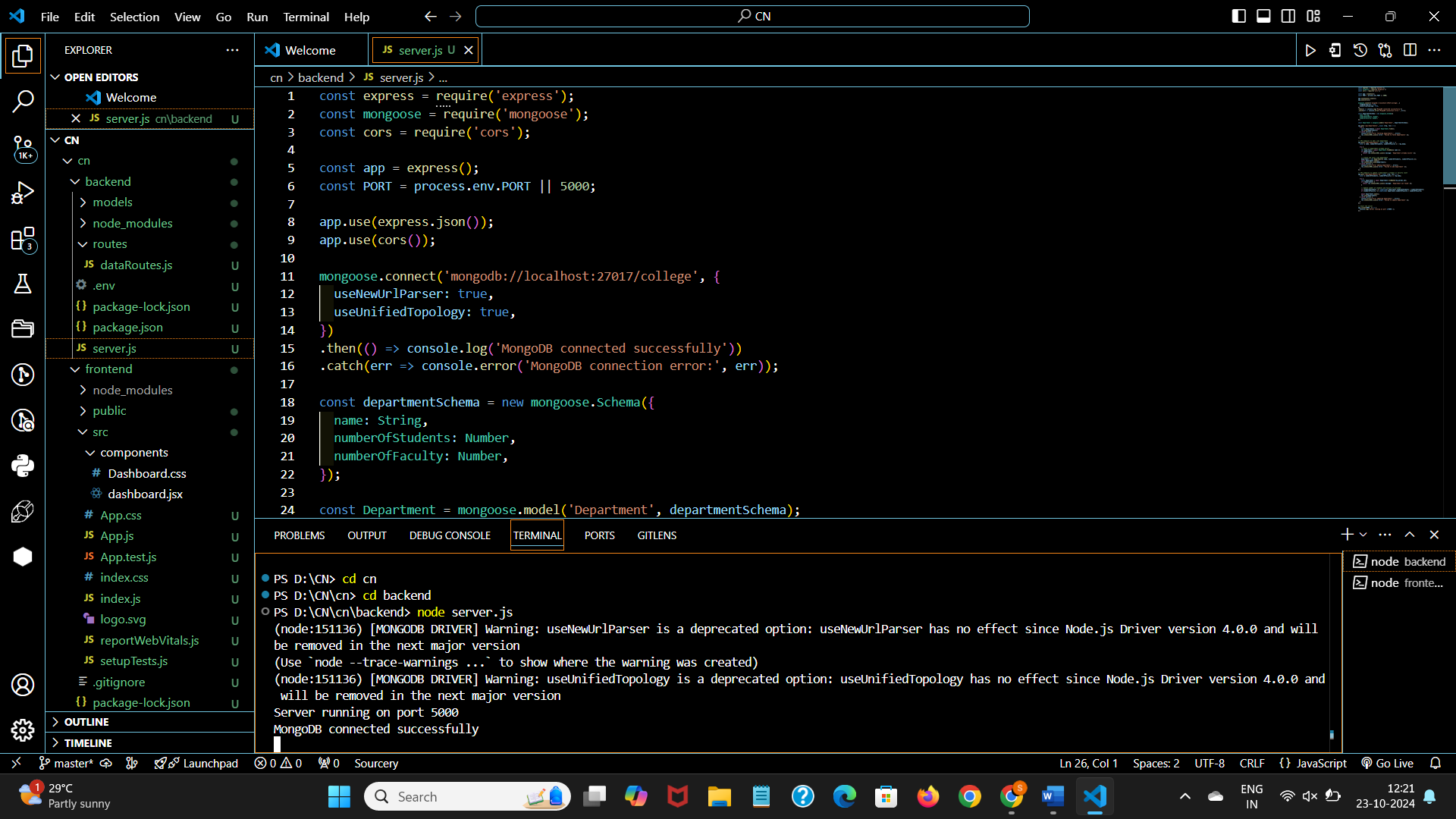
**ADDING MECHANICAL DEPARTMENT:**



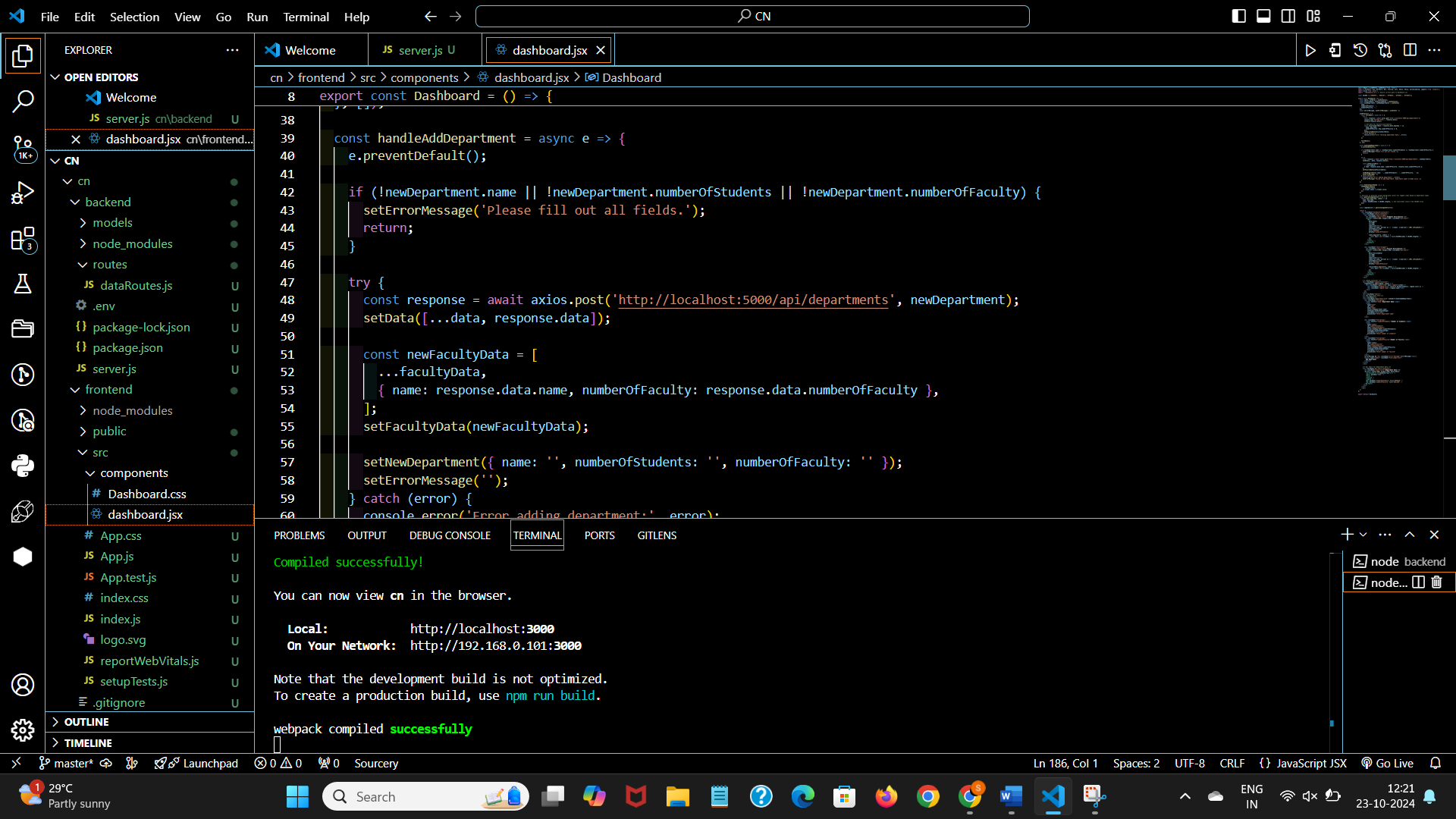
**CHANGES DONE AFTER ADDING MECHANICAL DEPARTMENT:**



**SERVER SIDE:**



**CLIENT SIDE:**



**DATABASE:**

