PLACEMENT MANAGEMENT SYSTEM

ATISHAY JAIN - 2210110217

SHREEYA ARORA- 2210110569

TANISHA MADAAN - 2210110613

Table of Contents

CONTENTS

INTRODUCTION

The **Placement Management System(PMS)** is designed as a comprehensive tool to facilitate the interaction between students, universities, and potential employers. It aims to manage and optimize the placement process through a web-based interface built using modern technologies.

KEY FEATURES

1. Data Manipulation:

- Add/Delete Data :

2.Comprehensive Company Database:

- Admin-managed repository for adding and updating company details.
- Student-accessible profiles with information on visiting companies and recruitment specifics**.**

3.Student Record Management:

- Admin control over student data addition and maintenance.
- Student profiles containing academic, skill, and interest details.

.

TECHNOLOGIES USED

1.) REACT.JS

Description: React.js serves as the foundation of our frontend development, providing a declarative and component-based approach to building user interfaces.

Key Features: With React, we create dynamic and responsive UI components, facilitating smooth navigation and interaction for administrators and students alike.

Benefits: React's virtual DOM ensures optimal rendering performance, while its component reusability simplifies development and maintenance tasks.

```
import React, { useState, useEffect } from "react";
import axios from "axios";
import { isStudent, isAuthenticated } from "./../auth/isAuthorized";
import { Link } from "react-router-dom";

function Company() {
  const [data, setData] = useState([]);
  const user = isAuthenticated();

  const loadCompanies = async () => {
    try {
      const response = await axios.get(
        "http://localhost:5000/api/get/companies"
```

```
PS C:\Users\91941\Desktop\placement-final> cd server
PS C:\Users\91941\Desktop\placement-final\server> nodemon index.js
[nodemon] 3.1.0
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node index.js`
body-parser deprecated undefined extended: provide extended option index.js:5:27
connected to databse successfully!
```

2.) NODE.JS

- **Description:** Node.js forms the backend infrastructure of our system, offering a scalable and event-driven runtime environment for server-side applications.
- **Key Features:** Leveraging Node.js and its lightweight Express.js framework, we handle HTTP requests, routing, and middleware integration seamlessly.
- **Benefits:** Node.js enables efficient handling of concurrent connections, ensuring optimal performance and responsiveness for our placement management system.

```
const mysql=require("mysql");
const express= require ("express");
const app = express();
const bodyParser = require("body-parser");
const encoder= bodyParser.urlencoded();
const cors = require('cors');
```

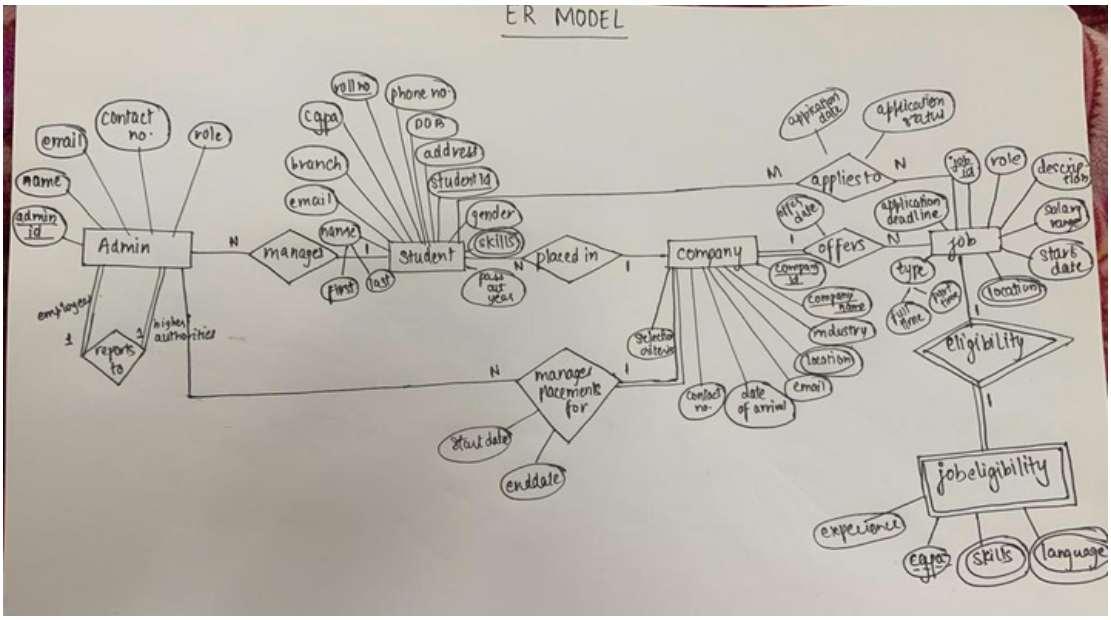
```
PS C:\Users\91941\Desktop\placement-final> cd server
PS C:\Users\91941\Desktop\placement-final\server> nodemon index.js
[nodemon] 3.1.0
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node index.js`
body-parser deprecated undefined extended: provide extended option index.js:5:27
connected to databse successfully!
```

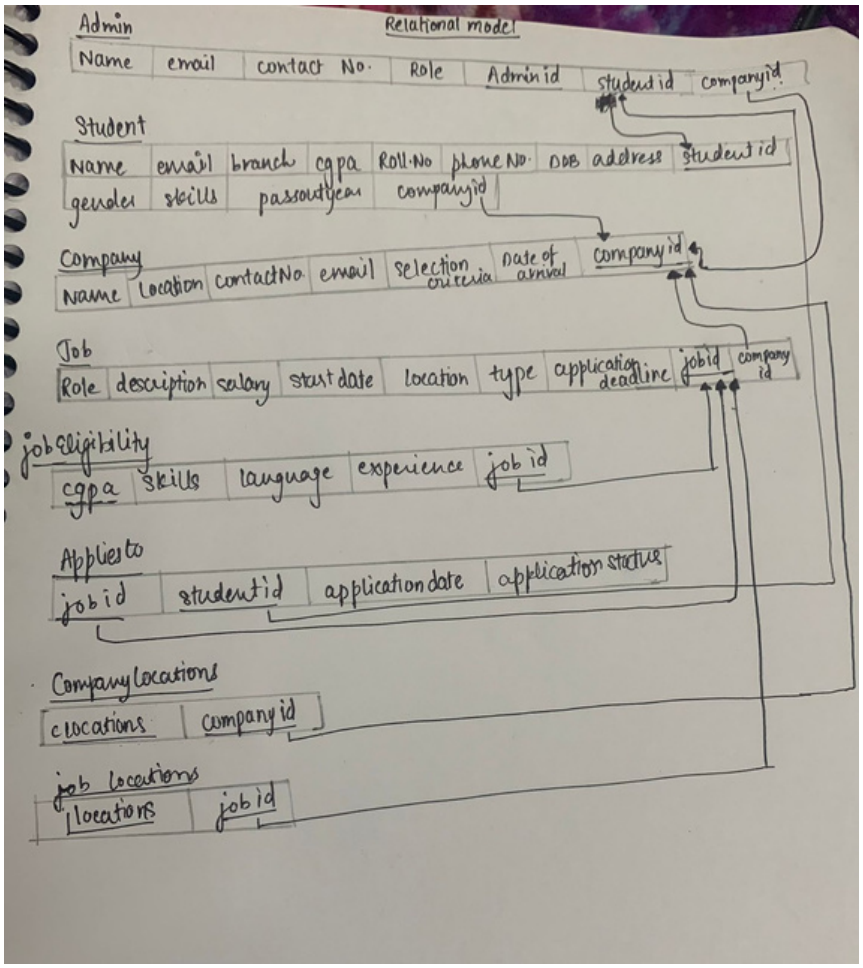
3.) MySQL

- **Description:** MySQL serves as the relational database management system (RDBMS) for storing and managing structured data related to students, companies, events, and other entities within our system.
- **Key Features:** Utilizing SQL queries and transactions, we ensure efficient data retrieval, manipulation, and integrity within our placement management system.
- **Benefits:** MySQL offers robust security features, ACID compliance, and scalability options, making it a reliable choice for managing critical data in our application.

```
const mysql=require("mysql");
const express= require ("express");
const app = express();
const bodyParser = require("body-parser");
const encoder= bodyParser.urlencoded();
const cors = require('cors');
```

ER MODEL


ER MODEL
Admin
email
contact no.
role
name
admin id
manages
Student
roll no
phone no.
cgpa
DOB
address
branch
student id
gender
skills
name
email
first
last
passout year
placed in
company
company id
company name
industry
location
email
contact no.
date of arrival
selection criteria
offers
applies to
application date
application status
offer date
application deadline
job
job id
role
description
salary range
start date
type
full time
part time
location
eligibility
jobeligibility
experience
cgpa
skills
language
reports to
employees
higher authorities
manages placements for
start date
enddate


Relational model
Admin
Name email contact No. Role Admin id studentid companyid
Student
Name email branch cgpa ROLL No phone No. DOB address studentid
gender skills passoutyear companyid
Company
Name Location contactNo email selection criteria Date of arrival companyid
Job
Role description salary start date location type application deadline jobid company id
jobeligibility
cgpa skills language experience job id
Applies to
jobid studentid application date application status
Company locations
clocations companyid
job locations
locations jobid

NORMALIZED RELATIONS

Normalization is the process of arranging data in a database to eliminate redundancy and ensure data integrity. In this regard, the two tables available can be normalized into a set of tables where each table serves a specific purpose and minimal redundancy exists.
Below are the normalized relations based on the given columns:

1. COMPANIES TABLE

id	name	gpa	Type	Arrival_date	offered_ctc	year	role
23	Amazon	8.00	Product	2024-04-16	200000	2024	Software Dev
24	Google	7.50	Service	2024-04-23	2500000	2024	Software Dev
25	Apple	8.50	Product	2024-04-12	2300000	2024	Software Dev
27	Goldmann Sachs	8.00	Service	2024-04-30	2200000	2024	Software Dev
28	Delloite	6.00	Service	2024-04-09	100000	2024	NULL
29	HCL	6.50	Product	2024-04-11	1200000	2024	Software Dev
30	Bain	7.00	Product	2024-04-11	200000	2024	Software Dev
35	Microsoft	8.50	Service	2024-04-12	1200000	2024	Software Dev
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Companies Table: Contains company-specific in formation.

CompanyID (Primary Key)
Name
Type
gpa
Arrival_Date
Role
Offered_CTC
Year

2. **STUDENTS TABLE**

id	name	branch	year	batch	cgpa	backlogs
4	Tanisha Madaan	CSE	2	2026	8.00	0
5	Tushar Juneja	ECE	2	2026	8.50	0
6	Jay	ECE	3	2025	8.00	0
7	Akash	CSE	2	2027	7.00	0
8	Saniya	Mech	3	2025	7.50	0
9	Rakshit	ECE	3	2025	7.00	2
10	Asmita	CSE	2	2026	8.50	0
12	Aradhya	Mech	3	2024	7.00	1
13	Atishay	CSE	2	2026	7.00	0
14	Rishi	ECE	2	2026	8.00	0

Students Table: Contains student-specific information

StudentID (Primary Key)
Name
Branch
Year
Batch
backlogs
StudentID (Foreign Key)
CGPA

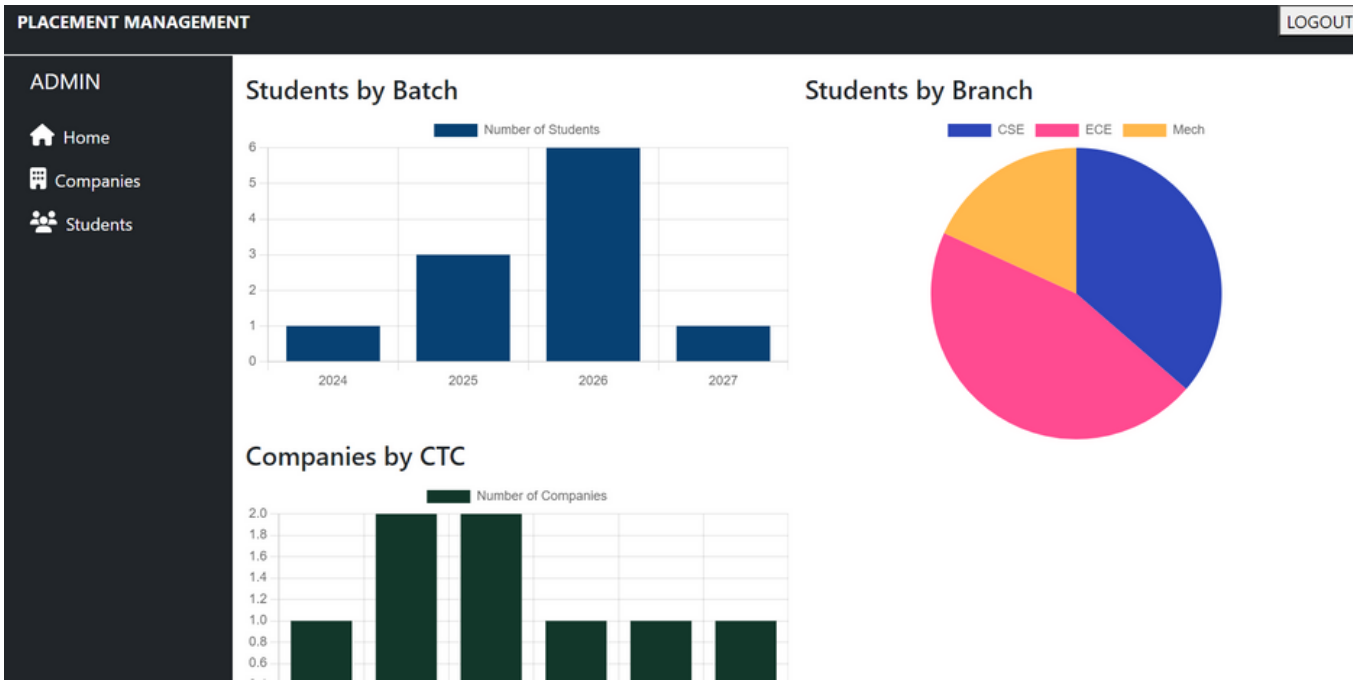
Below are the normalized relations based on the given columns:

1. **Company Details:**
   - company_id (Primary Key)
   - name
   - type
   - Arrival_date
   - offered_ctc
2. **Company Requirements:**
   - requirement_id (Primary Key)
   - company_id (Foreign Key referencing Company Details)
   - gpa
   - year
   - rol
3. **Student Details:**
   - student_id (Primary Key)
   - name
   - branch
   - year
   - batch
   - cgpa
   - backlogs

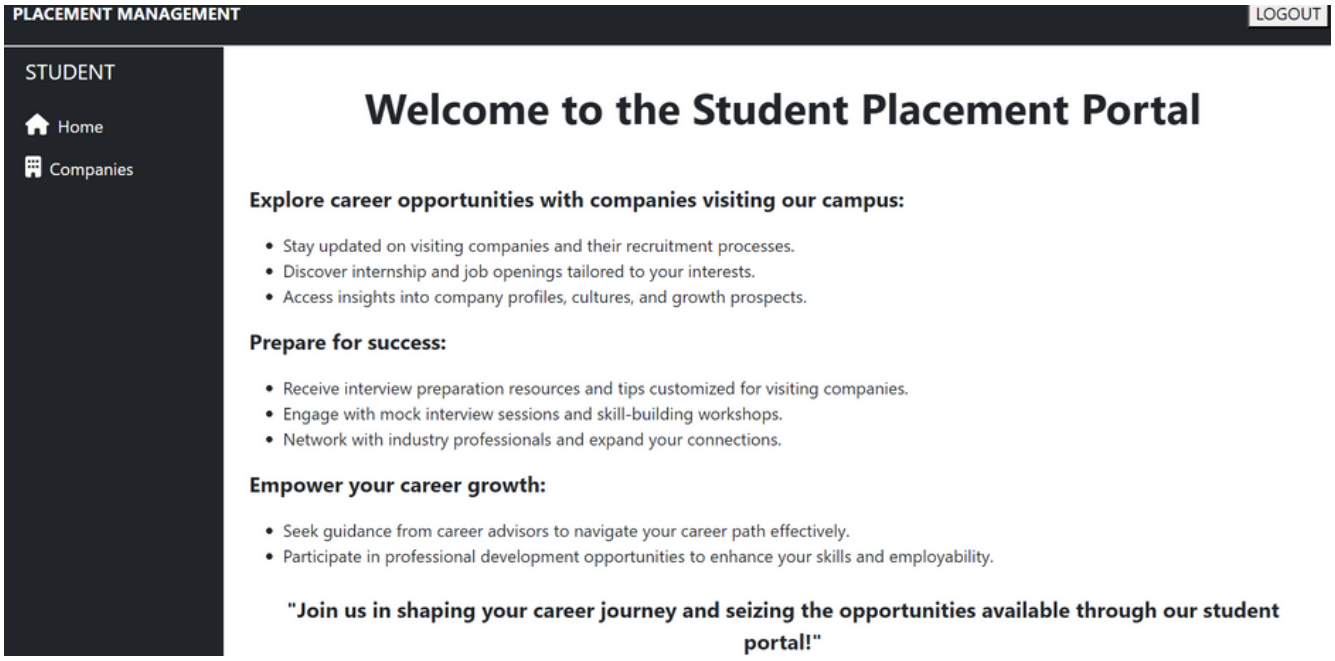
COMPONENTS

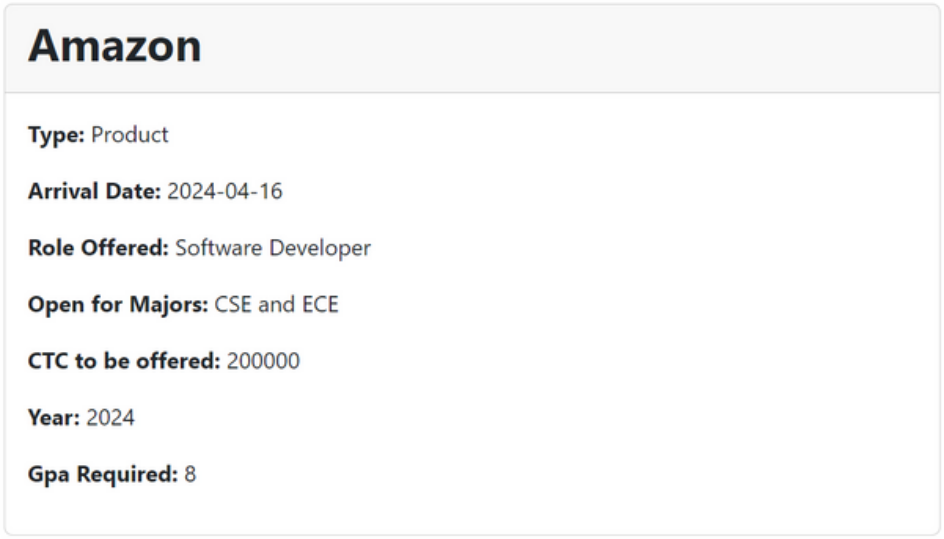
HEADER:

HOME PAGE ADMIN DASHBOARD:

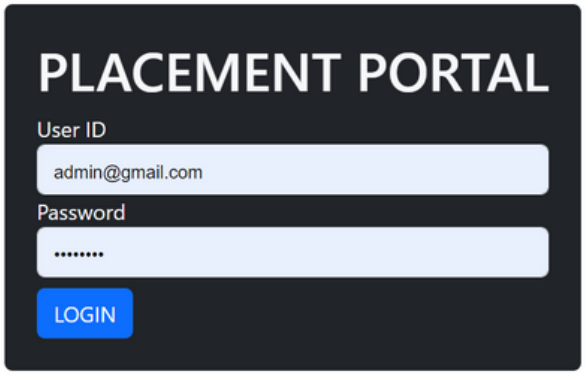


HOME PAGE STUDENT DASHBOARD:





LOGIN PORTAL:



DATA RENDERING:

Add

S.No	Name	Type	Date of Arrival	CTC	Year	Actions
1	Amazon	Product	2024-04-16	200000	2024	Edit Delete
2	Google	Service	2024-04-23	2500000	2024	Edit Delete
3	Apple	Product	2024-04-12	2300000	2024	Edit Delete
4	Goldmann Sachs	Service	2024-04-30	2200000	2024	Edit Delete
5	Delloite	Service	2024-04-09	100000	2024	Edit Delete
6	HCL	Product	2024-04-11	1200000	2024	Edit Delete
7	Bain	Product	2024-04-11	200000	2024	Edit Delete
8	Microsoft	Service	2024-04-12	1200000	2024	Edit Delete

Add

S.No	Name	Branch	Year	Batch	CGPA	Backlogs	Actions
1	Tanisha Madaan	CSE	2	2026	8	0	Edit Delete
2	Tushar Juneja	ECE	2	2026	8.5	0	Edit Delete
3	Jay	ECE	3	2025	8	0	Edit Delete
4	Akash	CSE	2	2027	7	0	Edit Delete
5	Saniya	Mech	3	2025	7.5	0	Edit Delete
6	Rakshit	ECE	3	2025	7	2	Edit Delete
7	Asmita	CSE	2	2026	8.5	0	Edit Delete
8	Aradhya	Mech	3	2024	7	1	Edit Delete

ADDING DATA:

Add Company

Name
Enter Name

Type
Enter Type

Date of Arrival
dd-mm-yyyy

CTC
Enter CTC

Year
Enter Year

Gpa required
Enter gpa required

Open For
Enter open for

Role
Enter Role Offered

Submit

Add Student

Name
Enter Name

Branch
Enter Branch

Year
Enter year

Batch
Enter Batch

CGPA
Enter cgpa

Backlogs
Enter backlogs

Submit

PROJECT STRUCTURE:

1.) Entry Point (index.js):

This is the main file where your Express application is initialized.
Commonly includes package imports, server setup, and middleware configuration.
Routes (routes folder):
Routes are organized in separate files.
Each route file handles specific endpoints or groups of related endpoints.

Code:

```
const mysql=require("mysql");
const express= require ("express");
const app = express();
const bodyParser = require("body-parser");
const encoder= bodyParser.urlencoded();
const cors = require('cors');

app.use("/assets",express.static("assets"));
app.use(cors());
app.use(express.json());
const connection= mysql.createConnection({
    host: "localhost",
    user: 'root',
    password:"2114",
    database: "nodejs"
});

//connect to database

connection.connect(function(error){
    if(error) throw error
        else console.log("connected to databse successfully!")
})

app.get("/",function(req,res){
    res.sendFile(__dirname + "/login.html");
})
```

2)Routing:

Routes are defined for different paths. Routing logic is modularized, separating concerns and improving code organization.

Code:

```jsx
import { BrowserRouter, Routes, Route } from "react-router-dom";
import ProtectedRoute from "./auth/ProtectedRoute";
// import Login from "./pages/auth/Login";
import { isStudent } from "./auth/isAuthorized";
import Student from "./pages/Student";
import CreateStudent from "./pages/CreateStudent";
import { Bar } from "react-chartjs-2";

function App() {
  return (
    <BrowserRouter>
      <Routes>
        {/* <Route path="/login" element={<Login />} /> */}
        {/* <Route path="/login" element={<Login />} /> */}

        <Route path="/" element={<Company />}></Route>
        {/* <Route path="/update/:id" element={<UpdateCompany />} />
        <Route
          path="/create"
          element={
            <ProtectedRoute allow={() => !isStudent()}>
              <CreateCompany />
            </ProtectedRoute>
          }
        />
```

3)Middleware:

Middleware functions are used to handle tasks such as parsing request bodies, setting response headers, and handling errors.

. **Code:**

```jsx
import React, { useEffect, useState } from "react";
import { Route, useNavigate } from "react-router-dom";
import { isAuthenticated, isStudent } from "./isAuthorized";
// import { isAuthenticated } from "./authService";

const ProtectedRoute = (props) => {
  const navigate = useNavigate();
  const [isLoggedIn, setLoggedIn] = useState(false);
  //   const auth_status = true;
  const user = isAuthenticated();
  const checkAuth = () => {
    if (!user || !props.allow()) {
      setLoggedIn(false);
      return navigate("/login");
    }
    setLoggedIn(true);
  };
  useEffect(() => {
    checkAuth();
  }, []);
  return isLoggedIn ? props.children : "";
};

export default ProtectedRoute;
```

4)JSX Requests:

JSX handles various requests to get the company and student details.

Code:

```
app.get('/api/get-users', function(req, res) {
    const db_query = 'select * from loginuser';
    connection.query(db_query, (err, result) => {
        res.send(result);
    });
});

app.get('/api/get/companies', function(req, res) {
    const db_query = 'select * from companies';
    connection.query(db_query, (err, result) => {
        res.send(result);
    });
});
```

5)Database Interaction:

Express routes interact with the MySQL database to fetch and send data.
Routes like /getConstituencies, /getYears, and /display execute SQL queries to retrieve relevant data from the database.

code:

The follwing triggers have been used to update and delete data in all tables to maintain data integrity and consistency

Triggers

```
DELIMITER //

CREATE TRIGGER update_details
AFTER UPDATE ON companies
FOR EACH ROW
BEGIN
    UPDATE students
    SET company_name = NEW.company_name
    WHERE company_id = NEW.company_id;
END;
//

DELIMITER ;
```

```
DELIMITER //

CREATE TRIGGER delete_details
AFTER DELETE ON companies
FOR EACH ROW
BEGIN
    DELETE FROM students
    WHERE company_id = OLD.company_id;
END;
//

DELIMITER ;
```

Following is the control to interact with database to retreive required data for the dropdown menus

code:

```
app.get('/api/get-users', function(req, res) {
    const db_query = 'select * from loginuser';
    connection.query(db_query, (err, result) => {
        res.send(result);
    });
});

app.get('/api/get/companies', function(req, res) {
    const db_query = 'select * from companies';
    connection.query(db_query, (err, result) => {
        res.send(result);
    });
});

app.get('/api/get/students', function(req, res) {
    const db_query = 'select * from students';
    connection.query(db_query, (err, result) => {
        res.send(result);
    });
});

app.get('/api/get/company/:id', function(req, res) {
    const db_query = 'select * from companies where id = ' + req.params.id;
    console.log(db_query);
    connection.query(db_query, (err, result) => {
        res.send(result);
```

```
app.get('/api/get-users', function(req, res) {
    const db_query = 'select * from loginuser';
    connection.query(db_query, (err, result) => {
        res.send(result);
    });
});

app.get('/api/get/companies', function(req, res) {
    const db_query = 'select * from companies';
    connection.query(db_query, (err, result) => {
        res.send(result);
    });
});

app.get('/api/get/students', function(req, res) {
    const db_query = 'select * from students';
    connection.query(db_query, (err, result) => {
        res.send(result);
    });
});

app.get('/api/get/company/:id', function(req, res) {
    const db_query = 'select * from companies where id = ' + req.params.id;
    console.log(db_query);
    connection.query(db_query, (err, result) => {
        res.send(result);
```

Following is the controller to handle update without writing queries everytime to update the companies and student table

code:

```
app.put('/update/:id', (req, res) => {
    console.log('receiving data');
    const sql = "UPDATE companies SET Name = ?, Gpa = ?, role = ?, open_for = ?, Type
    = ?, Arrival_Date = ?, offered_ctc = ?, Year = ? WHERE id = ?";
    const values = [req.body.name, req.body.gpa, req.body.role, req.body.openfor, req.
    body.type, req.body.date, req.body.ctc, req.body.year, req.params.id];
    connection.query(sql, values, (err, data) => {
        if (err) return res.status(500).json({ error: err.message });
        return res.json({ message: "Company updated successfully" });
    });
});

app.put('/students/update/:id', (req, res) => {
    console.log('receiving data');
    const sql = "UPDATE students SET Name = ?, branch = ?, year = ?, batch = ?, cgpa
    = ?, backlogs = ? WHERE id = ?";
    const values = [req.body.name, req.body.branch, req.body.year, req.body.batch, req.
    body.cgpa, req.body.backlogs, req.params.id];
    connection.query(sql, values, (err, data) => {
        if (err) return res.status(500).json({ error: err.message });
        return res.json({ message: "Company updated successfully" });
    });
});
```

Following is the controller to handle insert without writing queries everytime to insert data. Iinto the companies and student table

code:

```
app.post('/api/create/company', (req, res) => {
    const sql = "INSERT INTO companies (name, gpa, role, open_for, type, Arrival_Date,
    offered_ctc, Year) VALUES (?, ?, ?, ?, ?, ?, ?, ?);";

    // Adjusted column name
    const values = [
        req.body.name,
        req.body.gpa,
        req.body.role,
        req.body.openfor,

        req.body.type,
        req.body.date, // Use req.body.date for Arrival_Date
        req.body.ctc,
        req.body.year,
    ];
    connection.query(sql, values, (err, data) => {
        if (err) return res.json(err);
        return res.json(data);
    });
});
app.post('/api/create/student', (req, res) => {
    const sql = "INSERT INTO students (name, branch, year, batch, cgpa, backlogs)
    VALUES (?, ?, ?, ?, ?, ?);"; // Adjusted column name
    const values = [
        req.body.name,
```

Following is the controller to handle delete without writing queries everytime to delete data from the companies and student table

code:

```
app.delete('/api/delete/:id', (req, res) => {
    const studentid = req.params.id;

    const sql = "DELETE FROM students WHERE id = ?";

    connection.query(sql, [studentid], (err, data) => {
        if (err) {
            console.error("Error deleting student:", err);
            return res.status(500).json({ error: "An error occurred while deleting the
            student" });
        }
```

Connecting Backend to Database

using mysql library provide by node package manager, we have established connection to the mysql database as follows:

code:

```
const bodyParser = require("body-parser");
const encoder= bodyParser.urlencoded();
const cors = require('cors');

app.use("/assets",express.static("assets"));
app.use(cors());
app.use(express.json());
const connection= mysql.createConnection({
    host: "localhost",
    user: 'root',
    password:"2114",
    database: "nodejs"
});

//connect to database

connection.connect(function(error){
    if(error) throw error
        else console.log("connected to databse successfully!")
})
```

LIMITATIONS

1. Complex Queries: Normalization can lead to more complex queries, particularly when retrieving data from multiple normalized tables, resulting in increased query complexity and potentially impacting performance.
2. Data Duplication: While normalization reduces redundancy, it can also lead to increased data duplication due to the need for joining tables, consuming additional storage space and complicating data maintenance.
3. Update Anomalies: Normalization may introduce update anomalies, where modifying a single piece of information requires updates across multiple tables, increasing the risk of inconsistencies and errors in data management.

REFERENCES USED

- **MySQL in Node.js: Getting Started with DB & CRUD Operations**
- **Placement Management System**