



COLLEGE NAME: MEENAKSHI COLLEGE OF ENGINEERING

COLLEGE CODE: 3114

YOUR NAME: ARIVANANTHAM.J

NAAN MUDHALVAN (NM) ID:

C243912FBA9215698764EA53EB29B62B

PHONE NO: 8807194452

EMAIL ID: nandhamarivu06@gmail.com

GIT HUB LINK: <https://github.com/Arivu611/crud.git>

COLLEGE COUNSELING MANAGEMENT SYSTEM

1. INTRODUCTION

The College Counseling System is a web-based application developed to help manage the college counseling process in an organized and efficient manner.

This system allows administrators to store, update, view, and delete student information digitally.

It reduces manual work and improves accuracy in handling student data during counseling.

2. PROBLEM STATEMENT

Traditional college counseling is mostly done manually, which leads to:

Data duplication

Human errors

Time consumption

Difficulty in managing large student records

Hence, there is a need for an automated system to manage the counseling process efficiently.

3. OBJECTIVES OF THE PROJECT

The main objectives of the College Counseling System are:

To automate the student counseling process

To store student data securely

To perform fast data retrieval

To reduce manual errors

To provide a structured counseling workflow

4. SCOPE OF THE PROJECT

The scope of this project includes:

Managing student details

Performing CRUD operations (Create, Read, Update, Delete)

Providing API-based backend support

Testing APIs using Postman

This system can be enhanced in the future by:

Adding a frontend user interface

Connecting to a database

Implementing real-time counseling logic

5. SYSTEM REQUIREMENTS

5.1 Hardware Requirements

Computer or Laptop

Minimum 4 GB RAM

Internet connection (optional)

5.2 Software Requirements

Operating System: Windows / Linux

Node.js

Visual Studio Code

Postman

Web Browser

6. TECHNOLOGIES USED

Programming Language: JavaScript

Backend Runtime: Node.js

Framework: Express.js

API Testing Tool: Postman

Development Tool: Visual Studio Code

7. SYSTEM ARCHITECTURE

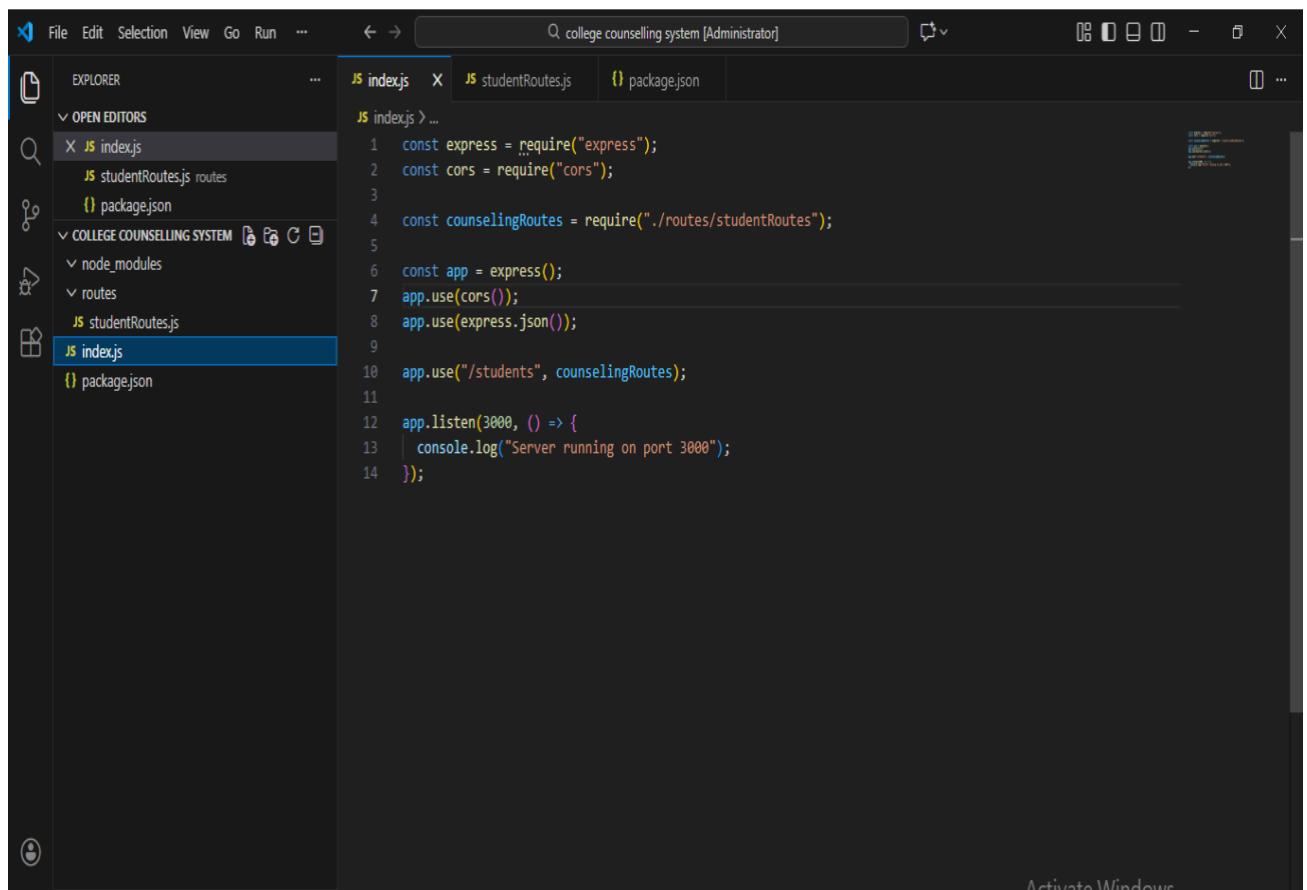
The College Counseling System follows a client-server architecture.

The client sends HTTP requests.

The server processes the requests.

The server returns responses in JSON format.

The system uses REST APIs to communicate between client and server.



A screenshot of the Visual Studio Code interface. The left sidebar shows a tree view of files and folders. In the center, there are three tabs: 'index.js' (selected), 'studentRoutes.js', and 'package.json'. The code editor displays the content of 'index.js':

```
1 const express = require("express");
2 const cors = require("cors");
3
4 const counselingRoutes = require("./routes/studentRoutes");
5
6 const app = express();
7 app.use(cors());
8 app.use(express.json());
9
10 app.use("/students", counselingRoutes);
11
12 app.listen(3000, () => {
13   console.log("Server running on port 3000");
14});
```

A screenshot of the Visual Studio Code interface. The title bar reads "college counselling system [Administrator]". The left sidebar shows a file tree with "OPEN EDITORS" containing "index.js", "studentRoutes.js routes", and "package.json". Under "COLLEGE COUNSELLING SYSTEM", there are "node_modules" and "routes" folders, both containing "studentRoutes.js", "index.js", and "package.json". The main editor tab is "studentRoutes.js", which contains the following code:

```
routes > JS studentRoutes.js > router.post("/") callback
1 const express = require("express");
2 const router = express.Router();
3
4 // Temporary data (acts like database)
5 let students = [
6   { id: 1, name: "Arun", dept: "CSE", age: 23 },
7   { id: 2, name: "Aravind", dept: "ECE", age: 21 }
8 ];
9
10 // GET - Get all students
11 router.get("/", (req, res) => {
12   res.json(students);
13 });
14
15 // POST - Add new student
16 router.post("/", (req, res) => {
17   const newStudent = req.body;
18   students.push(newStudent);
19
20   res.json({
21     message: "Student added successfully",
22     students
23   });
24 });
25
26 // PUT - Update student
27 router.put("/", (req, res) => {
28   const updatedStudent = req.body;
29
30   students = students.map(student =>
```

A screenshot of the Visual Studio Code interface. The title bar reads "college counselling system [Administrator]". The left sidebar shows a file tree with "OPEN EDITORS" containing "index.js", "studentRoutes.js", and "package.json". Under "COLLEGE COUNSELLING SYSTEM", there are "node_modules" and "routes" folders, both containing "studentRoutes.js", "index.js", and "package.json". The main editor tab is "package.json", which contains the following code:

```
{ "name": "college-counselling-system", "version": "1.0.0", "main": "node index.js", "scripts": { "start": "node index.js" }}
```

8. MODULE DESCRIPTION

Student Management Module

This module handles all student-related operations:

Add student details

View student list

Update student information

Delete student records

9. WORKING PRINCIPLE

The server is started using Node.js.

APIs are created using Express.js.

Postman is used to send requests to the server.

The server processes the request.

The response is sent back in JSON format.

10. API OPERATIONS

GET Operation

Used to fetch student details

The screenshot shows the Postman application interface. On the left, there's a sidebar with 'History' and a list of recent requests. In the main area, a request card for a 'GET' request to 'http://localhost:3000/students/' is displayed. The 'Body' tab is selected, showing a JSON response with two student records:

```

1  [
2   {
3     "id": 1,
4     "name": "Arun",
5     "dept": "CSE",
6     "age": 23
7   },
8   {
9     "id": 2,
10    "name": "Aravind",
11    "dept": "ECE",
12    "age": 21
13  }
14 ]

```

Below the response, the status bar shows 'Status: 200 OK' and other details like time and size. A 'Send' button is visible at the top right of the request card.

POST Operation

Used to add new student information

The screenshot shows the Postman application interface. In the main area, a request card for a 'POST' request to 'http://localhost:3000/students/' is displayed. The 'Body' tab is selected, showing a JSON payload being sent:

```

1  {
2   "id": 3,
3   "name": "Balu",
4   "dept": "EEE",
5   "age": 25
}

```

Below the request, the status bar shows 'Status: 200 OK' and other details. A 'Send' button is visible at the top right of the request card.

PUT Operation

Used to update existing student data

The screenshot shows the Postman application interface. In the left sidebar, there is a history of requests, including a recent PUT request to http://localhost:3000/students/. The main panel displays a PUT request to the same URL. The 'Body' tab is selected, showing a JSON payload:

```
1 {  
2   "id": 1, "name": "UpdatedName", "dept": "EEE", "age": 25  
3 }  
4 {  
5   "id": 2,  
6   "name": "Aravind",  
7   "dept": "ECE",  
8 }  
9 {  
10  "id": 3,  
11  "name": "Balu",  
12  "dept": "EEE",  
13 }
```

The response status is 200 OK, and the JSON response body is:

```
1 {  
2   "message": "Student updated successfully",  
3   "students": [  
4     {  
5       "id": 1,  
6       "name": "UpdatedName",  
7       "dept": "EEE",  
8       "age": 25  
9     },  
10    {  
11      "id": 2,  
12      "name": "Aravind",  
13      "dept": "ECE",  
14    },  
15    {  
16      "id": 3,  
17      "name": "Balu",  
18      "dept": "EEE",  
19    }  
20  ]  
21 }
```

DELETE Operation

Used to delete student records

The screenshot shows the Postman application interface. In the left sidebar, there is a history of requests, including a recent DELETE request to http://localhost:3000/students/. The main panel displays a DELETE request to the same URL. The 'Body' tab is selected, showing a JSON payload:

```
1 [2]
```

The response status is 200 OK, and the JSON response body is:

```
1 {  
2   "message": "Students deleted successfully",  
3   "students": [  
4     {  
5       "id": 1,  
6       "name": "UpdatedName",  
7       "dept": "EEE",  
8       "age": 25  
9     },  
10    {  
11      "id": 3,  
12      "name": "Balu",  
13      "dept": "EEE",  
14    }  
15  ]  
16 }
```

11. ADVANTAGES OF THE SYSTEM

Easy to use

Saves time

Reduces manual errors

Organized data handling

Scalable for future enhancement

12. LIMITATIONS

No frontend UI

No permanent database

Limited to backend operations

13. FUTURE ENHANCEMENTS

Database integration (MySQL / Mongo DB)

User authentication

Frontend interface

College cutoff logic

Role-based access (Admin / Student)

14. CONCLUSION

The College Counseling System provides an efficient solution for managing student counseling data.

It simplifies the counseling process and reduces manual effort.

This project demonstrates the practical implementation of REST APIs using Node.js and Express.

15. REFERENCES

Node.js Official Documentation

Express.js Documentation

Postman Learning Center

The screenshot shows a GitHub repository page for the user 'Arivu611' with the repository name 'crud'. The page includes the following details:

- Branch:** arivanantham (selected)
- Commits:** 2 Branches, 0 Tags
- Activity:** This branch is up to date with main .
- Contributors:** Arivu611 (1 Commit)
- Files:** index.js, package.json, studentRoutes.js (all added via upload)
- README:** A section with a 'Add a README' button and a note: "Help people interested in this repository understand your project."
- Statistics:** 0 stars, 0 watching, 0 forks
- Sections:** About, Releases, Packages, Languages