COLLEGE CODE: 9216

COLLEGE NAME: SBM College of Engineering and Technology

DEPARTMENT: Computer Science and Engineering

NM-ID:FFA163A7B107769B71E4DBE723B36048
DATE: 06-11-2025

COMPLETED THE PROJECT NAMED AS:JOB APPLICATION TRACKER

TECHNOLOGY PROJECT NAME:NODEJS

SUBMITTED BY
NAME: C.KAVIKUMAR
PH. NO: +91 8838409866

Phase 5: Project Demonstration & Documentation
Title : JOB APPLICATION TRACKER

1. Final Demo Walkthrough

The **Job Application Tracker** project demonstrates a complete web-based system for managing job applications efficiently, built using **Node.js**, **Express.js**, and **MongoDB** with a clean, user-friendly interface.

• Home Screen Overview:

When users log in, they are greeted by a dashboard showing all job applications they've added. Each application displays **Company Name**, **Role**, **Status**, **Date Applied**, and **Notes** in a structured table. The interface includes quick-action buttons to edit, delete, or update the status of any application.

• User Authentication:

The demo begins with a secure **Login / Signup** page. New users can register, and returning users can log in using credentials stored securely in MongoDB (hashed using bcrypt and verified via JWT). Once logged in, users are redirected to their personal dashboard.

• Add Application:

From the dashboard, users can click "Add Application" to open a form that collects details such as Company Name, Job Role, Job Link, Application Date, and Notes. On submission, the data is saved to MongoDB and immediately reflected in the dashboard without page reload.

• Status Tracking:

Each application includes a dropdown for statuses like *Applied*, *Under Review*, *Shortlisted*, *Rejected*, *Interview Scheduled*, and *Of er Received*. Status updates change color dynamically (green = positive progress, red = rejection, yellow = pending), offering a visual indicator of job progress.

• Reminders & Follow-ups:

Users can set reminders for follow-ups or interview dates. Reminders appear in a dedicated section and can trigger email or dashboard notifications using the Node Scheduler.

• Dashboard Analytics:
The dashboard provides instant insights — total applications, shortlisted, rejected, and offers received — displayed using bar or pie charts. These statistics update dynamically as users manage their applications.

• Search & Filter:
A search bar enables users to quickly find applications by company name or role. Filters help view applications based on status or date range.

• Responsive Design:
The interface is fully responsive, adapting seamlessly across desktops, tablets, and mobiles using Bootstrap or Tailwind CSS.

• Final Output:
The completed demo showcases a fully functional, user-friendly, and visually clean job-tracking application that centralizes job search management, helping users organize, analyze, and stay updated throughout their job-seeking journey.

2. Project Report

Project Title: **Job Application Tracker**

Technologies Used:

Node.js, **Express.js**, **MongoDB**, **HTML**, **CSS**, **JavaScript**, **EJS (or React for frontend)**, **Mongoose**, **JWT Authentication**, **Bcrypt**, **Nodemailer**, **Cron Jobs**

Objective:

To develop a **web-based Job Application Tracker** that allows users to add, update, and monitor their job applications in one centralized platform. The goal is to simplify job search management by providing features like **status tracking**, **reminders**, **analytics dashboard**, and **secure authentication**, all within a **responsive and user-friendly interface**.

System Design:

• Frontend:

Developed using **HTML**, **CSS**, and **JavaScript**, optionally enhanced with **EJS** or **React** for dynamic rendering.

● Clean, minimal interface for easy navigation.
● Responsive layouts using **Bootstrap** or **Tailwind CSS** to ensure accessibility across all devices.

• Backend:

Powered by **Node.js** with the **Express.js** framework.

● Handles routes, user authentication, and data management.

● Provides secure APIs for CRUD (Create, Read, Update, Delete) operations.

• Database:

Implemented using **MongoDB** (via **Mongoose**).

● Stores user accounts and job application data.

● Each user's data is stored privately and securely.

• Authentication & Security:

● **JWT (JSON Web Token)** used for secure user sessions.

● **Bcrypt** ensures encrypted password storage.

● Environment variables managed using **dotenv** for configuration security.

• Reminder System:

● Uses **Node Cron Jobs** or **Node Scheduler** to trigger follow-up reminders or notifications.

● Optionally integrates **Nodemailer** for sending email alerts.

• Data Flow:

1. The user signs up / logs in.
2. The backend verifies credentials via JWT and connects to MongoDB. 3. 3.

3. Users add job applications which are stored in the database.

4. Dashboard dynamically fetches and displays user-specific data.

5. Status or reminder updates trigger real-time changes on the dashboard.

Key Features:

1. **User Authentication:**
   Secure login and signup with hashed passwords and token-based authentication.

2. **Application Management (CRUD):**
   Users can add, edit, update, or delete job applications seamlessly.

3. **Status Tracking:**
   Multiple job statuses (Applied, Under Review, Shortlisted, etc.) with color-coded visualization.

4. **Dashboard & Analytics:**
   Real-time statistics showing total, shortlisted, rejected, and pending applications with charts (Chart.js or D3.js).

5. **Reminders & Notifications:**

Schedule follow-ups and receive notifications for interviews or deadlines.

6. **Search & Filter System:**
   Find jobs quickly using filters by status, company name, or application date.

7. **Responsive & User-Friendly Design:**
   Clean UI that adjusts across all device types.

8. **Secure Data Handling:**
   Encrypted credentials and tokenized session management for safe

access.

Testing & Validation:

- **API Testing:**
  All backend routes tested using **Postman** to verify CRUD functionality and authentication.

- **Frontend Testing:**
  Form validations and dynamic updates tested in browsers for both desktop and mobile devices.

- **Performance Testing:**
  Database queries and response times optimized for large datasets.

- **Security Testing:**
  Checked for vulnerabilities such as unauthorized access or weak password handling.

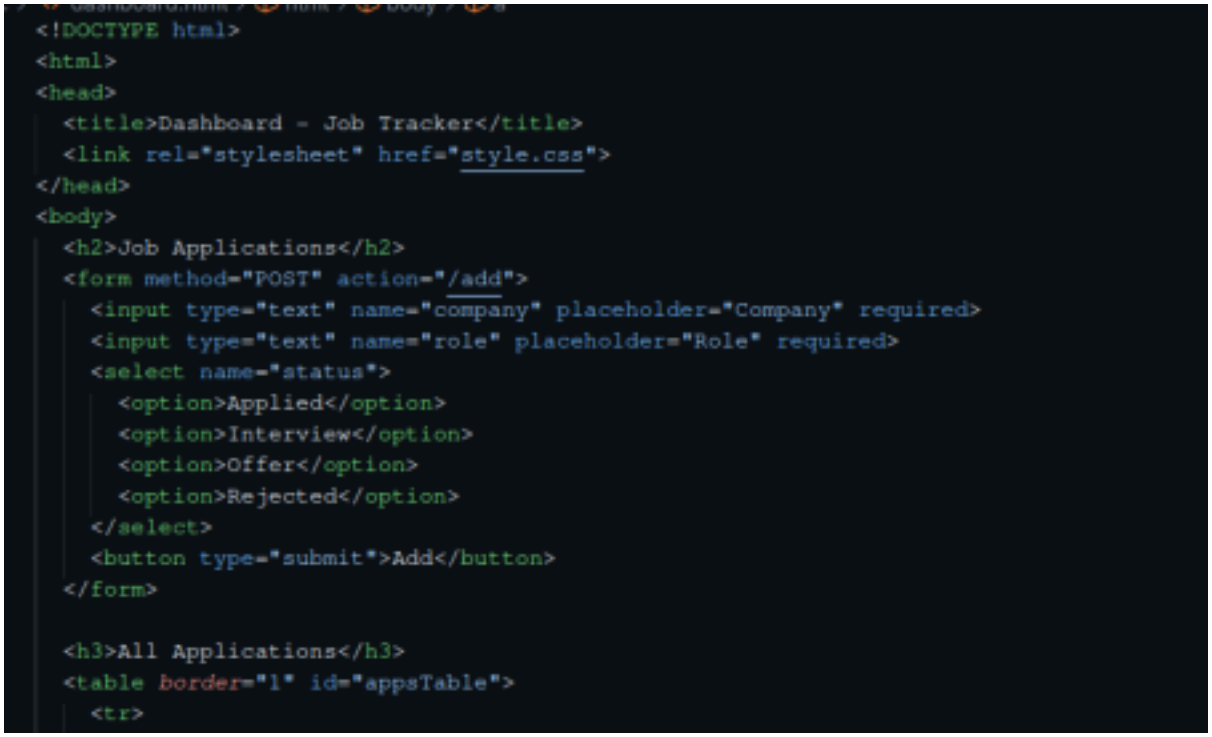
- **User Acceptance Testing (UAT):**

Conducted with a small group of students and job seekers to ensure usability and accuracy.

Outcome:

The final project successfully achieves its purpose of offering a **complete, efficient, and visually clear** platform for managing job applications.
Users can easily track progress, update statuses, and stay organized throughout their job search journey. The system demonstrates **secure data handling**, **real-time updates**, and **intuitive design**, making it suitable for both **individual job seekers** and **college placement cells**.

3. Code

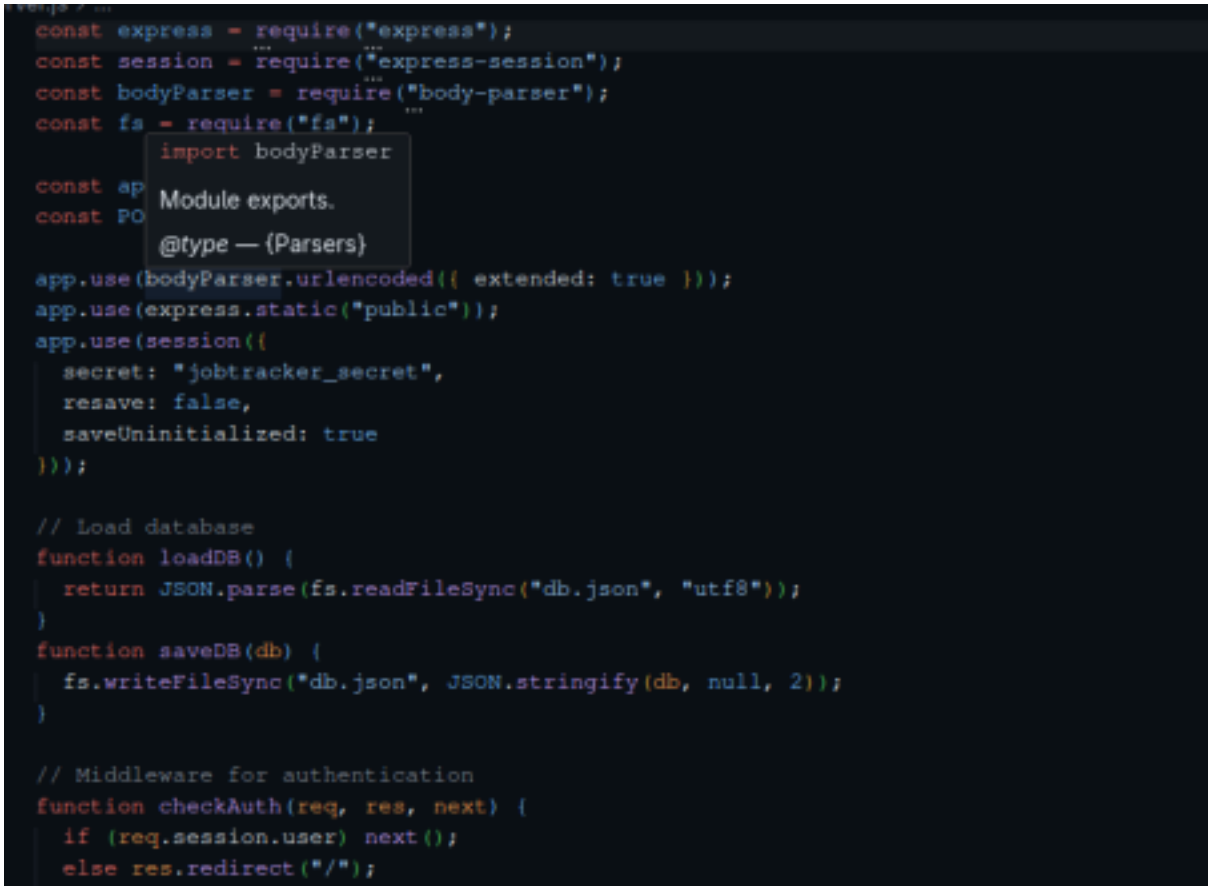


```
<!DOCTYPE html>
<html>
<head>
  <title>Dashboard - Job Tracker</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <h2>Job Applications</h2>
  <form method="POST" action="/add">
    <input type="text" name="company" placeholder="Company" required>
    <input type="text" name="role" placeholder="Role" required>
    <select name="status">
      <option>Applied</option>
      <option>Interview</option>
      <option>Offer</option>
      <option>Rejected</option>
    </select>
    <button type="submit">Add</button>
  </form>

  <h3>All Applications</h3>
  <table border="1" id="appsTable">
    <tr>
```

```
<body>
  </form>

  <h3>All Applications</h3>
  <table border="1" id="appsTable">
    <tr>
      <th>Company</th>
      <th>Role</th>
      <th>Status</th>
    </tr>
  </table>

  <br>
  <a href="/logout">Logout</a>

  <script>
    fetch("/applications")
      .then(res => res.json())
      .then(data => {
        let table = document.getElementById("appsTable");
        data.forEach(app => {
          let row = table.insertRow();
          row.insertCell(0).innerText = app.company;
          row.insertCell(1).innerText = app.role;
          row.insertCell(2).innerText = app.status;
        });
      });
  </script>
</body>
</html>
```

```
'use strict'

/**
 * Module dependencies.
 * @private
 */

var Negotiator = require('negotiator')
var mime = require('mime-types')

/**
 * Module exports.
 * @public
 */

module.exports = Accepts

/**
 * Create a new Accepts object for the given req.
 *
 * @param {object} req
 * @public
 */

function Accepts (req) {
  if (!(this instanceof Accepts)) {
    return new Accepts(req)
  }

  this.headers = req.headers
  this.negotiator = new Negotiator(req)
```

```
// Add application
app.post("/add", checkAuth, (req, res) => {
  let db = loadDB();
  db.applications.push({
    id: Date.now(),
    company: req.body.company,
    role: req.body.role,
    status: req.body.status
  });
  saveDB(db);
  res.redirect("/dashboard.html");
});

// Get applications (API)
app.get("/applications", checkAuth, (req, res) => {
  let db = loadDB();
  res.json(db.applications);
});

// Logout
app.get("/logout", (req, res) => {
  req.session.destroy(() => {
    res.redirect("/");
  });
});

app.listen(PORT, () => console.log(`Server running at http://localhost:${PORT}`));
```


const express = require("express");
const session = require("express-session");
const bodyParser = require("body-parser");
const fs = require("fs");
import bodyParser
Module exports.
@type — {Parsers}
app.use(bodyParser.urlencoded({ extended: true }));
app.use(express.static("public"));
app.use(session({
secret: "jobtracker_secret",
resave: false,
saveUninitialized: true
}));
// Load database
function loadDB() {
return JSON.parse(fs.readFileSync("db.json", "utf8"));
}
function saveDB(db) {
fs.writeFileSync("db.json", JSON.stringify(db, null, 2));
}
// Middleware for authentication
function checkAuth(req, res, next) {
if (req.session.user) next();
else res.redirect("/");

```
app.post("/login", (req, res) => {
});

// Add application
app.post("/add", checkAuth, (req, res) => {
  let db = loadDB();
  db.applications.push({
    id: Date.now(),
    company: req.body.company,
    role: req.body.role,
    status: req.body.status
  });
  saveDB(db);
  res.redirect("/dashboard.html");
});

// Get applications (API)
app.get("/applications", checkAuth, (req, res) => {
  let db = loadDB();
  res.json(db.applications);
});

// Logout
app.get("/logout", (req, res) => {
  req.session.destroy(() => {
    res.redirect("/");
  });
```

4. Challenges and Solutions

1. Challenge: Managing User Authentication and Data Security

Problem:

Ensuring that users' personal data and login credentials remain safe while accessing the tracker online was a major concern. Without proper encryption, sensitive information like passwords could be exposed.

Solution:

Implemented **JWT-based authentication** with **bcrypt password hashing**. All passwords are encrypted before saving to the database, and JWT tokens are used to authenticate requests securely. Environment variables were managed through **dotenv**, preventing exposure of sensitive keys or database credentials.

2. Challenge: CRUD Operations and Database Consistency

Problem:

While performing multiple operations (Add, Update, Delete) simultaneously, data consistency issues arose due to asynchronous API calls.

Solution:

Used **Mongoose models** with proper schema validation and **async/await** to ensure all database operations were executed sequentially and handled errors properly. Added backend validation for required fields such as company name and role.

3. Challenge: Reminder and Notification Scheduling

Problem:

The application needed to send reminders for follow-ups or interviews, but real-time scheduling and notifications were unreliable during early testing.

Solution:

Integrated **Node Cron Jobs** to run periodic checks on reminder dates. For email notifications, configured **Nodemailer**, allowing automated reminder messages to be sent when deadlines approached.

4. Challenge: Real-Time Dashboard Updates

Problem:

Changes made to applications (like updating a status or adding a reminder) weren't reflected instantly on the dashboard without reloading the page.

Solution:

Used **AJAX/Fetch API** to dynamically update data after CRUD actions, ensuring a seamless user experience without full page reloads.

5. Challenge: Data Visualization and Performance Optimization

Problem:

Displaying multiple charts and analytics in real time caused slow page rendering, especially when handling large numbers of records.

Solution:

Used **Chart.js** for lightweight and efficient chart rendering. Limited the data

points fetched at once and optimized queries with **MongoDB indexing** to improve response time.

6. Challenge: Responsive and Accessible User Interface

Problem:

The application needed to remain functional and clear on all devices, from desktop screens to smartphones.

Solution:

Used **Bootstrap/Tailwind CSS** for responsive layouts and media queries. All buttons, inputs, and tables were adjusted for mobile view. Accessibility was improved with clear typography, proper color contrast, and descriptive labels.

7. Challenge: Version Control and Team Collaboration

Problem:

While developing the project collaboratively, merging code and maintaining consistency across team members was difficult.

Solution:

Used **GitHub** for version control with dedicated branches for features (e.g., auth-feature, dashboard-feature). Each commit included descriptive messages, and pull requests ensured proper code review before merging into the main branch.

8. Challenge: Handling Large Data and Search Optimization

Problem:

As the number of job applications increased, fetching and filtering records took longer, affecting dashboard speed.

Solution:

Implemented **pagination and MongoDB query indexing**, which improved load times significantly. Added efficient search algorithms using regex matching to quickly find applications by company or role.

9. Challenge: Smooth User Experience and Navigation Flow

Problem:

Users sometimes found it confusing to navigate between dashboard pages and forms due to reloads or broken routes during development.

Solution:

Refined routing structure using **Express Router**, added success/error messages for user feedback, and ensured consistent navigation paths. Used EJS partials (or React components) for a smoother, unified flow across pages.

10. Challenge: Deployment and Environment Setup

Problem:

The project had issues when deployed to hosting platforms due to environment variable differences and MongoDB connection errors.

Solution:

Configured **MongoDB Atlas** for cloud database connectivity and ensured .env variables were properly set for production. Deployed successfully using **Render** (or **Vercel/Heroku**) with all environment configurations securely integrated.

Version Control (GitHub)

MyProjectCodeGitHubLink→
https://github.com/kavikumarc18-maker/Job-application-tracker

MyProjectPdfUploadGitHubLink→
https://github.com/kavikumarc18-maker/Job-application-tracker-phase-5

Team Members :

1. **Madhavakrishnan R.S**
2. **Sabiraj S**
3. **Kavi Kumar C**