COLLEGE CODE: 8203

COLLEGE: AVC COLLEGE OF ENGINEERING

DEPARTMENT: INFORMATION TECHNOLOGY

STUDENT NM-ID: 8EFD03039C61BA1B601456D562D0067A

ROLL NO: 23IT106

DATE:22/09/2025

Completed the project named as Phase 3

TECHNOLOGY PROJECT NAME: Job Application Tracker

SUBMITTED BY,

NAME: SRINATH R

MOBILE NO: 9342680867

Phase 3 MVP Implementation: Job Application Tracker

Project Setup

This establishes the foundation and connects the required services.

Step	Description	Tools Involved
1. Project Initialization & Dependencies	Initialize the Node.js project and install necessary packages for the backend, database, and user authentication.	`npm init -y`, `npm install express mongoose **jsonwebtoken** dotenv`
2. Server and Environment Setup	Create `server.js` for the Express application. Set up a `.env` file to securely store the MongoDB URI and a **JWT_SECRET** key for authentication.	Node.js, Express, dotenv
3. Database Connection	Use **Mongoose** (the MongoDB ODM) in `db.js` to establish a connection to the MongoDB instance using the URI from the `.env` file.	MongoDB, Mongoose

Data Storage (Database)

Define the structure to store and manage user and application data in

MongoDB.

Mongoose Schemas:

1. **User Schema (for Auth System):** Define the model to store user login credentials and identity for data separation.
    - `username` (String, Required, Unique)
    - `password` (String, Required) - ***Should be stored as a hash.***
    - `CreatedAt` (Date)
2. **Application Schema:** Define the model to store job application details.
    - `User Id` (Reference to User Schema, Required) - **Crucial for separating users' data.**
    - `company` (String, Required)
    - `role` (String, Required)
    - `status` (String, Required) - Values like '**Applied**', '**Interview**', '**Offered**', '**Rejected**'.
    - `Date Applied` (Date, Required)
    - `notes` (String)
    - `url` (String)

Core Features Implementation

This involves implementing the logic for the three main system components:

Authentication, CRUD Operations, and Filtering.

1. **Authentication and Authorization:**

   Login Endpoint (`POST /api/users/login`):

   - Validates the user's username/password and, upon success, generates and sends a **JSON Web Token (JWT)**.
   - **Middleware:** Create a JWT verification middleware that runs on all application-related routes. This middleware extracts the `userId` from the token and attaches it to the request object, ensuring a user can only interact with their own data.

2. **REST API Endpoints (Express):** All endpoints must use the **JWT middleware** to ensure authorization and data separation.

   Create Entry (`POST /api/applications`):

   - Input Validation: Ensure all required job details (company, status, dateApplied) are present.
   - Data Persistence: Use Mongoose to save the new application document to MongoDB, automatically setting the
   - `userId` from the JWT middleware.

   Read Entries (`GET /api/applications`):

   - Retrieve **all** job entries for the authenticated `userId`.

   Update Entry (`PUT /api/applications/:id`):

   - Find and update a specific entry by its ID, but only if the entry's `userId` matches the authenticated user's `userId`.

   Delete Entry (`DELETE /api/applications/:id`):

   - Find and delete a specific entry by its ID, ensuring the `userId` matches for authorization.

3.**Filtering/Search Feature:**

Filtering by Status (`GET /api/applications?status=Applied`):

- 1.Modify the `GET /api/applications` route to accept a `status` query parameter.
- 2.The Mongoose query will filter the results by the authenticated user's `userId` **AND** the requested `status` (e.g., 'Applied', 'Interview', etc.).

Testing Core Features

Thorough testing is required to verify the integration between the authentication and data components.

Test Scenario	Description	Tool
User Authentication	Test the login endpoint with valid credentials to ensure a valid **JWT** is returned.	postman
Data Separation (Authorization)	Attempt to fetch data using a JWT, then attempt to fetch the same data using a JWT from a *different* user. Verify the second request fails or returns no data.	Postman
Create and Read	Submit a new job application and verify the new entry appears in the MongoDB collection, linked to the correct	`userId`.
Filtering	Submit multiple applications with different statuses. Query the endpoint using a filter (e.g.,	`?status=Interview`) and verify only the matching entries are returned.
Update and Delete	Successfully update a field (e.g., status from 'Applied' to 'Interview') and then successfully delete the entry, verifying the change/removal in the database.	Postman

Version Control (GitHub)

Maintain project integrity and allow for collaboration.

- **Repository:** https://github.com/SrinathR93/NM-PROJECT.git
- **Commits:** Regularly commit changes with descriptive messages after completing each major feature or bug fix.