## 

**COLLEGE CODE: 8203**

**COLLEGE: AVC COLLEGE OF ENGINEERING**

**DEPARTMENT: INFORMATION TECHNOLOGY**

**STUDENT NM-ID:** **8EFD03039C61BA1B601456D562D0067A**

**ROLL NO: 23IT106**

**DATE: 03/10/2025**

**Completed the project named as Phase 4**

**TECHNOLOGY PROJECT NAME:**  **Job Application Tracker**

**SUBMITTED BY,**

**NAME: SRINATH R**

**MOBILE NO: 9342680867**

## Phase 4 – Enhancements & Deployment

**Tools:** Node.js, Express, MongoDB

**Goal:** To create a secure, feature-rich, and user-friendly web application for tracking job applications.

**Core Functionality:**

* User submits job details via API.
* MongoDB stores key fields like **company**, **status**, **date applied**, and **notes**.
* User can **update or delete** existing entries.
* Entries can be **filtered by status**: Applied, Interview, Offered, Rejected, etc.
* An **Authentication system** is required to separate and secure individual users' data.

### 1. Additional Features

To make the Job Application Tracker more robust and user-friendly, the following features should be added:

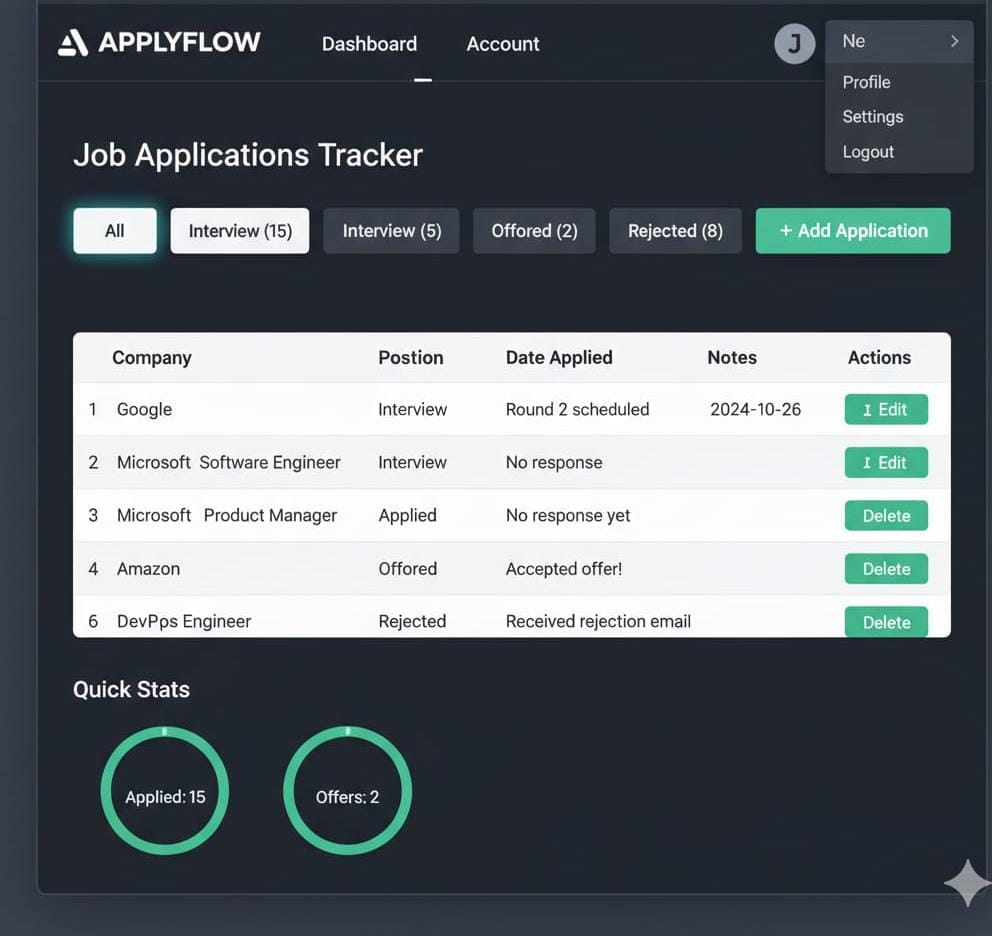
* **Follow-up Reminders:** Allow users to set a follow-up date for an application.

This requires adding a followUpDate field to the MongoDB schema and a simple cron job to notify the user (e.g., via email alert) on that date.

* **Job Link Storage:** Add a mandatory field for the **original job posting URL** to the MongoDB schema.
* **Archiving:** Implement an option for users to "archive" or "hide" applications that have a final status (e.g., Offered or Rejected) to keep the main view clean.This requires a isArchive boolean field and a corresponding filter.
* **Filtering by Date Range:** Allow users to filter applications based on the dateApplied within a specific range.

### 2. UI/UX Improvements

Focus on a clear, efficient interface to manage the job search:

* **Dashboard View:** Create a main dashboard showing a **summary of applications by status** (e.g., a count of 'Applied,' 'Interview,' and 'Offered').
* **Kanban/Card View:** Implement a view where applications are displayed as cards and can be dragged and dropped between status columns (e.g., Applied → Interview) for easy status updates.
* **Clear Feedback:** Provide immediate visual feedback, such as a **"Application updated successfully!"** toast notification.
* **Intuitive Forms:** Ensure the form for adding a new application is streamlined and easy to use.
* **Responsive Design:** Guarantee the user interface is fully functional on both desktop and mobile devices for on-the-go updates.
* 

### 

### 3. API Enhancements

Refine the REST API for security and power:

#### **Core API Endpoints**

The API must be protected with a

**JSON Web Token (JWT)** to ensure users can only manage their own application data.

| Functionality | HTTP Method | Endpoint Path | Description |
| --- | --- | --- | --- |
| **Register User** | POST | /api/auth/register | Creates a new user account. |
| **Login User** | POST | /api/auth/login | Authenticates and returns a JWT. |
| **Create Application** | POST | /api/applications | Creates a new job application entry. |
| **Get All Applications** | GET | /api/applications | Retrieves a list of applications for the authenticated user (supports filtering). |
| **Update Application** | PUT | /api/applications/:id | Modifies an existing application's details (e.g., status notes) |  |
| **Delete Application** | DELETE | /api/applications/:id | Removes a job application entry by its ID. |  |

#### API Refinements

* **Input Validation:** Use a library like **Joi** or **express-validator** to ensure all incoming data is correct before processing.
* **Improved Error Handling:** Provide clear, specific JSON error messages, such as a 404 Not Found if the application ID does not exist.
* **Pagination:** Enhance the GET /api/applications endpoint to support pagination (e.g., ?page=1&limit=20) to handle large application lists efficiently.

### 4. Performance & Security Checks

Before deployment, the application must be fast and secure:

#### Performance

* **Database Indexing:** Add an index to the **userId** and **status** fields in the MongoDB collection to significantly speed up filtering and retrieval queries.
* **Load Testing:** Use a tool like **Artillery** or **JMeter** to simulate multiple users concurrently accessing their application lists and updating entries.

#### Security

* **Authentication (JWT):** Implement user authentication using **JSON Web Tokens (JWT)** to ensure **every request** to manage applications is authenticated and authorized.
* **Environment Variables:** Store all sensitive information in a **.env file** and **never** commit it to version control like Git.
* **Rate Limiting:** Use a package like express-rate-limit to prevent users from spamming the application creation or update endpoints.

### 5. Testing of Enhancements

Thorough testing is crucial to ensure reliability:

* **Unit Tests:** Write tests for individual functions, such as the logic for filtering applications by status or verifying the date format.
* **Integration Tests:** Test the interaction between different system parts. For example, test the flow from hitting the **"create application"** API endpoint to verifying a new document is saved in **MongoDB** with the correct **userId**.
* **End-to-End (E2E) Tests:** Simulate a real user scenario: log in, create a new application, update its status from 'Applied' to 'Interview', and then delete the entry.

### 6. Deployment

The application requires a continuously running server to manage the API and any potential follow-up reminder cron jobs.

* **Recommended Platforms:** **Heroku**, **Render**, **AWS Elastic Beanstalk**, or **Google Cloud App Engine** are suitable for a persistent Node.js server. (Netlify and Vercel are generally less suitable for this type of application) .
* **Externalize Database:** Use a cloud-based database service like **MongoDB Atlas** for production data storage.
* **Configuration:** Add all secret keys and the database URI to the chosen platform's **Environment Variable** settings.
* **Deployment Pipeline:** Connect the GitHub repository for automated deployment whenever new code is pushed.
* **Monitoring:** Set up logging and monitoring tools on the deployment platform to track application uptime and errors.