COLLEGE CODE : 8203

COLLEGE NAME : A.V.C COLLEGE OF ENGINEERING

DEPARTMENT : CSE -B

STUDENT NM-ID : 234A1BB7E11F6C1984743CCF0B39DDA9

ROLL NO :23cs118

DATE :15-09-2025

Completed the project named as phase II

TECHNOLOGY PROJECT NAME : Job Application Tracker

SUBMITTED BY,

NAME: M.Yazhini

MOBILE NO:7339440384

Phase 2 — Solution Design & Architecture (Deadline – Week 7)

1. Tech Stack Selection

The Job Application Tracker is designed to provide a seamless way to track job applications and manage updates efficiently. The selected technology stack is as follows:

- Backend: Node.js with Express (RESTful API development)
- Database: MongoDB (NoSQL database for flexible schema)
- Frontend (optional extension): React.js (for UI, if required)
- Tools: Postman (API testing), Git/GitHub (version control), Jest/Mocha (testing)

2. UI Structure / API Scheme Design

UI Structure (if frontend included):

- Dashboard: Overview of all job application
  1. Add Job Form: Input company, role, status, note
  2. Application List: Table or cards displaying applications with actions (edit/delete)
  3. Search & Filter: By company, status, or date

- API Endpoints (Express + MongoDB):

  - POST /api/jobs → Add a new job application
  - GET /api/jobs → Retrieve all job applications
  - GET /api/jobs/:id → Retrieve specific job application
  - PUT /api/jobs/:id → Update job application details
  - DELETE /api/jobs/:id → Remove job application

3. Data Handling Approach

The application uses MongoDB with Mongoose ORM to define schemas and handle data. Local state may be used for prototyping. Data is validated and sanitized before saving. Example schema:

```
Job Schema:
  company: { type: String, required: true }
  position: { type: String, required: true }
  status: { type: String, enum: ['applied', 'interview', 'offer', 'rejected'], default: 'applied' }
  appliedDate: { type: Date, default: Date.now }
  notes: { type: String }
```

4. Component / Module Diagram

- Modules:
  - Server.js: Entry point, Express app setup
  - Routes: Define API endpoints (/api/jobs)
  - Controllers: Handle business logic (CRUD operations)
  - Models: Define Mongoose schemas
  - Config: Database connection settings
  - Middleware: Error handling, authentication (if extended)
- Diagram (Textual Representation):

```
Client (UI) → API Routes → Controllers → Models → MongoDB
                  ↑
              Middleware
```

5. Basic Flow Diagram

The following describes the flow of operations:

1. User submits job application details via UI or API request
2. Request hits Express route (/api/jobs)
3. Controller validates input and interacts with Mongoose model
4. Data is saved/retrieved from MongoDB
5. Response is sent back to user (success or error)
6. UI updates to reflect changes

Text Flow Representation:

User → [Express Route] → [Controller] → [Model] → [MongoDB] → Response → User