Capstone Project Submission: Job Application Tracker

Project Title: Job Application Tracker
Submitted by: Anuj Kumar Pathak
Batch: WIPRO NGA - .NET React Phase 2 - C3
Instructor: Mrs. Jyoti S Patil
Date: 20/03/2025

TABLE OF CONTENTS

	Entity-Relationship Diagram (ERD)	
	Optimization techniques for efficient queries	

CHAPTER 1: PROBLEM DEFINITON AND OBJECTIVES

Brief overview of the problem statement

The core problem that this project seeks to address is the inefficiency in current system architectures for managing and optimizing large-scale applications. Many applications face challenges related to user experience, system integration, and scalability, especially in environments that demand seamless communication between the frontend and backend. The problem is particularly evident in systems that require real-time data processing, efficient state management, and robust API interactions. This project aims to build an efficient, scalable web application by using modern technologies that bridge the gap between frontend and backend components seamlessly.

Problem Statement:

Job seekers often struggle with tracking multiple job applications, their statuses, and interview schedules. Without a structured system, managing job applications manually through spreadsheets or notes can be inefficient and overwhelming.

Solution:

The **Job Application Tracker** is a **full-stack web application** designed to help users **store**, **track**, and **manage** their job applications efficiently. Users can:

- Add job applications with details like company name, position, status, and application date.
- Update application statuses (e.g., "Applied", "Interview Scheduled", "Rejected", "Hired").
- View all job applications in a structured format.
- Receive notifications for interview scheduling.

Project goals and objectives

The primary goal of this project is to develop a web-based application that offers an intuitive user interface, fast backend processing, and secure data handling. The specific objectives include:

1. To design and implement a frontend using modern JavaScript frameworks like React or Angular, ensuring a dynamic and responsive user interface.
2. To integrate the frontend with a secure and efficient backend using ASP.NET Core Web API.
3. To ensure a smooth database interaction with optimal data handling using an appropriate relational database management system (RDBMS).
4. To create an API that provides robust endpoints for data exchange, supports authentication mechanisms, and ensures data security.
5. To focus on optimizing database performance, ensuring fast query processing even with large volumes of data.

CHAPTER 2: FRONTEND & BACKEND ARCHITECTURE

Overview of chosen technology stack (React/Angular, ASP.NET Core Web API, Database)

The project utilizes a modern technology stack for both frontend and backend to ensure scalability, maintainability, and a seamless user experience:

Frontend: React was used to create a dynamic, component-based user interface. React allows for efficient rendering of UI components and smooth user interactions, while Angular offers a comprehensive framework with built-in tools like routing and state management.

Backend: The backend was powered by ASP.NET Core Web API, which provides a lightweight, high-performance API development framework. This choice allows for building RESTful APIs with easy integration into various frontend frameworks.

Database: An RDBMS like SQL Server was used to store and manage application data. These databases provide high performance, scalability, and security for data storage and retrieval. **2.2**

Technology Stack:

- **Frontend:** React.js, BootStrap.
- **Backend:** ASP.NET Core Web API, Entity Framework Core.
- **Database:** SQL Server
- **Authentication:** JWT (JSON Web Token)
- **Deployment:** Docker

Component Breakdown

Backend (ASP.NET Core)

Located in Job_Application_Tracker directory.

- **Controllers**
  - ApplicationController.cs
  - AuthController.cs
  - JobController.cs
  - JobsController.cs
  - NotificationController.cs
  - UsersController.cs

- **Models**
  - Application.cs
  - Job.cs
  - JobDetails.cs
  - Login.cs
  - Notification.cs
  - User.cs

- **Data Layer**
  - ApplicationDbContext.cs → Defines database context using Entity Framework Core.

Frontend (React.js)

Located in client directory.

Components

- Applicants.jsx
- Application.jsx
- Notification.jsx
- SendNotification.jsx
- UserProfile.jsx

- **Pages**
  - **Home**
  - Auth.jsx
  - Footer.jsx
  - Navbar.jsx
  - NotFound.jsx

- **Services**
  - Auth.jsx
  - Login.jsx
  - NotificationService.jsx
  - ProtectedRouter.jsx

    - Register.jsx

Frontend:

- **Framework:** React
- **State Management:** Redux
- **API Calls:** Handled in /services using Axios
- **Routing:** Managed with React Router
- **UI Library:** Bootstrap for styling

Backend:

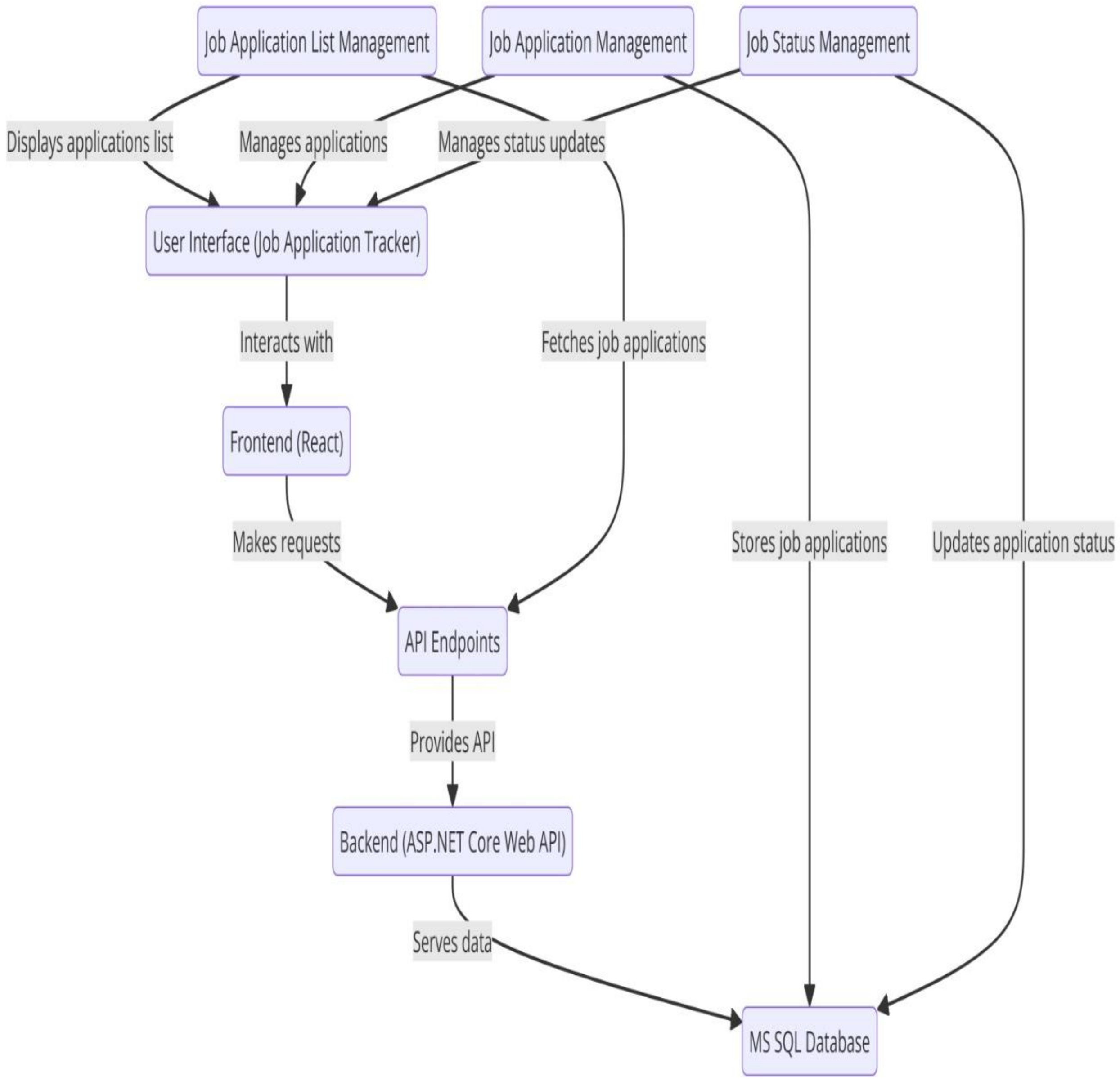
- **Framework:** ASP.NET Core Web API
- **Services:** Business logic (/Services)
- **Controllers:** Handle API requests (/Controllers)
- **Middleware:** Authentication, authorization, and error handling (/Middleware)

Database

- **Schema & Seed Data:**
- **Migrations:** Managed in /database/migrations
- **ORM:** Entity Framework Core for database interactions

Repository: Github [Repositry Link: https://github.com/Anuj02Pathak/Capstone.git]

System design diagram



CHAPTER 3: COMPONENT BREAKDOWN & API DESIGN

Breakdown of major components in frontend (state management, routing, UI components)

State Management: A state management solution like Redux (for React) was used to handle the global application state efficiently, enabling seamless communication between components and ensuring the state is consistent across the application.

Routing: React Router built-in routing was used to handle navigation between pages without refreshing the page. This enables smooth user experiences with URL-based navigation and parameter passing.

UI Components: Custom-built UI components was designed to ensure a consistent look and feel throughout the application. These components follow best practices for responsiveness and user accessibility.

API design, endpoints, authentication mechanism

- Company Dashboard: Create job postings, view employee applications.
- Employee Dashboard: View jobs, apply with name, email, and resume link.
- Modal Forms: Used for job application submissions.

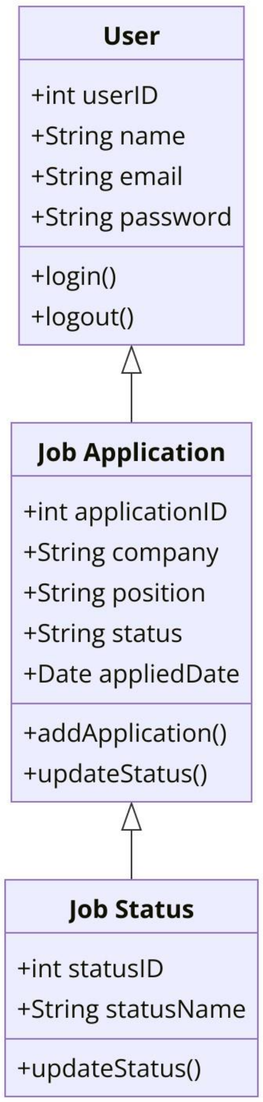
API Endpoints:

Endpoint	**Method**	**Description**
/api/jobpostings	GET	Fetch all job postings
/api/jobpostings	POST	Create a new job posting
/api/jobapplications	POST	Apply for a job

/api/jobapplications/ user?email={email}	GET	Fetch applications by user email
/api/jobapplications/ company	GET	Fetch applications for a company
/api/jobapplications/ {id}	PUT	Update application status

CHAPTER 4: DATABASE DESIGN & STORAGE OPTIMIZATION

Entity-Relationship Diagram (ERD)



Optimization techniques for efficient queries

1. **Indexes:** Indexes will be created on columns that are frequently used in queries, such as user ID, order date, and product ID, to speed up search operations.

2. **Query Optimization:** Complex queries will be optimized by breaking them down into smaller sub-queries, reducing redundancy, and leveraging joins effectively.

3. **Database Normalization:** The database schema will be normalized to reduce redundancy and improve data integrity. This will help maintain efficient storage and faster queries.

4. **Caching:** Frequently accessed data will be cached using a caching solution like Redis to reduce the load on the database and improve response times for users.