CareerCrafter, The Job Portal

Problem Statement:

CareerCrafter is a web-based application that aims to address these challenges by providing a robust platform for employers to post job listings and for job seekers to search, apply, and manage their applications. The application will also include a resume database feature, allowing job seekers to store and share their qualifications seamlessly.

Scope:

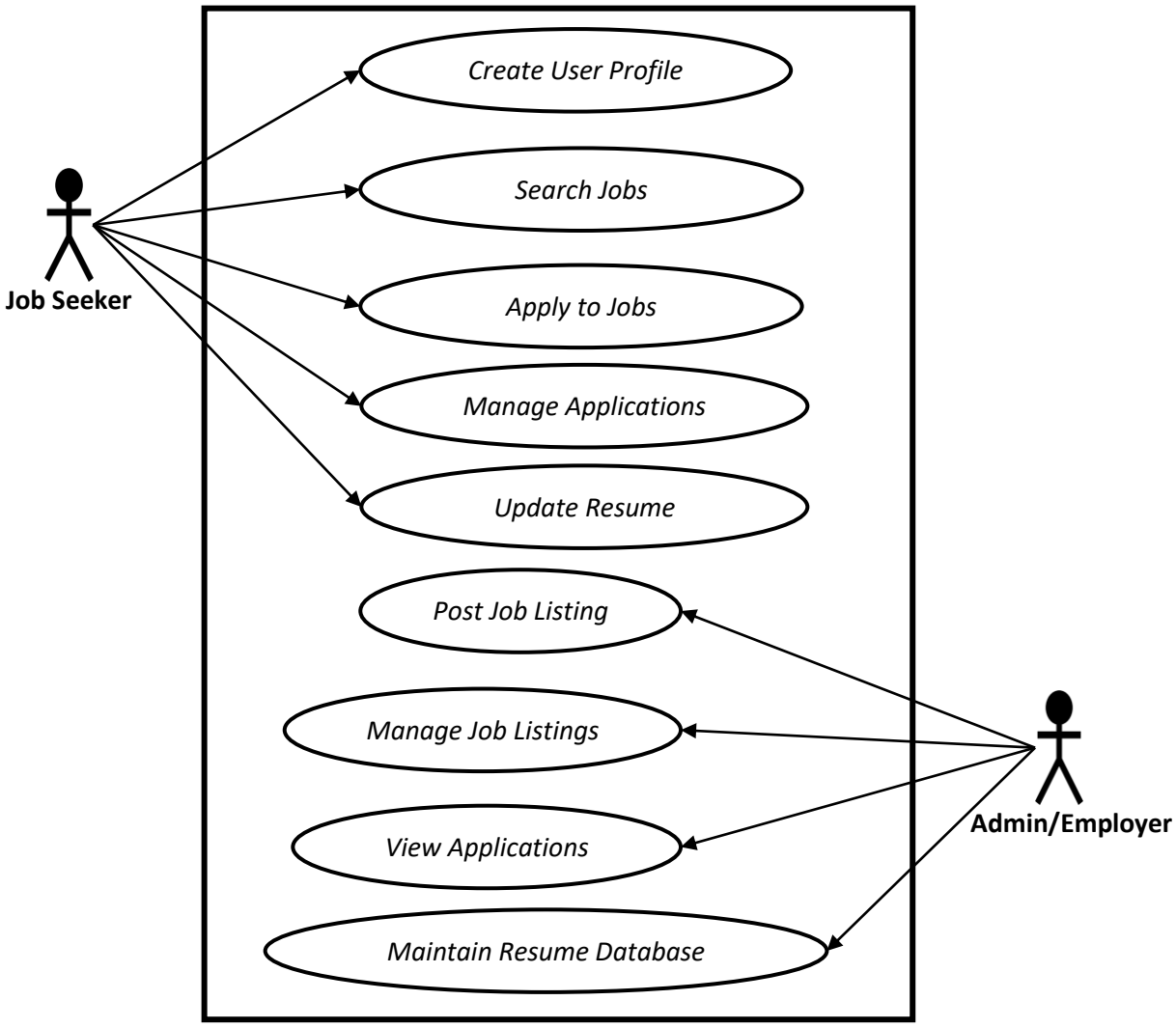
1. **User Registration and Authentication:**
    - Allow users to register and create accounts with a secure authentication process.
    - Implement role-based access control for employers and job seekers.
2. **Employer Module:**
    - Enable employers to create, edit, and manage job listings with detailed information.
    - Provide a dashboard for employers to track applications, view candidate profiles, and manage job postings.
3. **Job Seeker Module:**
    - Allow job seekers to create and manage user profiles with personal, educational, and professional details.
    - Implement a robust search functionality for job seekers to find relevant job listings based on various criteria.
    - Enable job seekers to apply to jobs directly through the platform, submitting their resumes and other required documents.
4. **Application Management:**
    - Develop a feature for job seekers to track and manage their job applications.
    - Provide real-time notifications for application status updates and communication with employers.
5. **Resume Database:**
    - Create a secure and scalable database to store job seekers' resumes and related documents.
    - Implement functionalities for job seekers to update, delete, and share their resumes with employers.

6. **Search and Recommendation Engine:**
    - Implement advanced search capabilities for both employers and job seekers.
    - Develop a recommendation engine to suggest relevant jobs to job seekers based on their profiles and preferences.

Technologies:

- **Frontend**: React.js / Angular.
- **Backend**: Java, Spring Boot/C#, .Net / Python Django for API development.
- **Database**: MySQL / SQL Server.
- **Authentication**: JSON Web Tokens (JWT) for secure user authentication.

Use Case Diagram:



Use Cases:

Actor: Employer

- Use Case: Post Job Listing
- Use Case: Manage Job Listings
- Use Case: View Applications

Actor: Job Seeker

- Use Case: Create User Profile
- Use Case: Search Jobs
- Use Case: Apply to Jobs
- Use Case: Manage Applications
- Use Case: Update Resume

Actor: System

- Use Case: Authenticate Users
- Use Case: Maintain Resume Database
- Use Case: Ensure Security

Development Process:

1. **Employer:**
   - Post Job Listing: The employer can create and publish job listings, providing details such as job title, description, qualifications, and application instructions.
   - Manage Job Listings: Edit, update, or remove existing job listings as needed.
2. **Job Seeker:**
   - Create User Profile: Job seekers can create a user profile by entering personal information, educational background, work experience, and skills.
   - Search Jobs: Job seekers can search for job listings based on various criteria such as job title, location, and industry.
   - Apply to Jobs: Users can apply to jobs directly through the platform, submitting their resume and any additional required documents.
   - Manage Applications: Job seekers can track and manage their job applications, view application status, and receive notifications.

3. **Security and Compliance:**

- User authentication and authorization are enforced to ensure data privacy.

1. JWT Authentication:

JWT authentication involves generating a token upon successful user login and sending it to the client. The client includes this token in subsequent requests to authenticate the user.

- User Login: Upon successful login (using valid credentials), generate a JWT token on the server.
- Token Payload: The token typically contains user-related information (e.g., user ID, roles, expiration time).
- Token Signing: Sign the token using a secret key known only to the server. This ensures that the token hasn't been tampered with.
- Token Transmission: Send the signed token back to the client as a response to the login request.
- Client Storage: Store the token securely on the client side (e.g., in browser storage or cookies).

2. JWT Authorization:

JWT authorization involves checking the token on protected routes to ensure that the user has the required permissions.

- Protected Routes: Define routes that require authentication and authorization.
- Token Verification:

1. Extract the token from the request header.
2. Verify the token's signature using the server's secret key.

- Payload Verification:

1. Decode the token and extract user information.
2. Check user roles or permissions to determine access rights.

- Access Control: Grant or deny access based on the user's roles and permissions.

3.Logout:

- Logging out involves invalidating the JWT token on both the client and the server to prevent further unauthorized requests.

Project Development Guidelines

The project to be developed based on the below design considerations.

1	**Backend Development**	• Use Rest APIs (Springboot/ASP.Net Core WebAPI to develop the services • Use Java/C# latest features. • Use ORM with database. • perform backend data validation. • Use Swagger to invoke APIs. • Implement API Versioning • Implement security to allow/disallow CRUD operations. • Message input/output format should be in JSON (Read the values from the property/input files, wherever applicable). Input/output format can be designed as per the discretion of the participant. • Any error message or exception should be logged and should be user-readable (not technical) • Database connections and web service URLs should be configurable. • Implement Unit Test Project for testing the API. • Implement JWT for Security • Implement Logging • Follow Coding Standards with proper project structure.

Frontend Constraints

1.	**Layout and Structure**	Create a clean and organized layout for your registration and login pages. You can use a responsive grid system (e.g., Bootstrap or Flexbox) to ensure your design looks good on various screen sizes.
2	**Visual Elements**	**Logo:** Place your application's logo at the top of the page to establish brand identity.
		Form Fields: Include input fields for email/username and password for both registration and login. For registration, include additional fields like name and possibly a password confirmation field.
		Buttons: Design attractive and easily distinguishable buttons for "Register," "Login," and "Forgot Password" (if applicable).
		Error Messages: Provide clear error messages for incorrect login attempts or registration errors.
		Background Image: Consider using a relevant background image to add visual appeal.
		Hover Effects: Change the appearance of buttons and links when users hover over them.
		Focus Styles: Apply focus styles to form fields when they are selected

3.	**Color Scheme and Typography**	Choose a color scheme that reflects your brand and creates a visually pleasing experience. Ensure good contrast between text and background colors for readability. Select a legible and consistent typography for headings and body text.
4.	**Registration Page, Doctor Consultation Page, Patient Appointment Booking Page, Add New Doctor Admin update details page**	**Form Fields:** Include fields for users to enter their name, email, password, and any other relevant information. Use placeholders and labels to guide users.
		Validation: Implement real-time validation for fields (e.g., check email format) and provide immediate feedback for any errors. **Form Validation:** Implement client-side form validation to ensure required fields are filled out correctly before submission.
	Registration Page	**Password Strength:** Provide real-time feedback on password strength using indicators or text. **Password Requirements**: Clearly indicate password requirements (e.g., minimum length, special characters) to help users create strong passwords.
		Registration Success: Upon successful registration, redirect users to the login page.
5.	**Login Page**	**Form Fields:** Provide fields for users to enter their email and password.
		Password Recovery: Include a "Forgot Password?" link that allows users to reset their password.
6.	**Common to React/Angular**	• Use Angular/React to develop the UI. • Implement Forms, data binding, validations, error message in required pages. • Implement Routing and navigations. • Use JavaScript to enhance functionalities. • Implement External and Custom JavaScript files. • Implement Typescript for Functions Operators. • Any error message or exception should be logged and should be user-readable (and not technical). • Follow coding standards. • Follow Standard project structure. • Design your pages to be responsive so they adapt well to different screen sizes, including mobile devices and tablets.

Good to have implementation features.

- Generate a SonarQube report and fix the required vulnerability.
- Use the Moq framework as applicable.
- Create a Docker image for the frontend and backend of the application.
- Implement OAuth Security.
- Implement design patterns.
- Deploy the docker image in AWS EC2 or Azure VM.
- Build the application using the AWS/Azure CI/CD pipeline. Trigger a CI/CD pipeline when code is checked-in to GIT. The check-in process should trigger unit tests with mocked dependencies.
- Use AWS RDS or Azure SQL DB to store the data.