# Job Recruitment Platform

- Description: Develop a job recruitment platform where employers can post job openings, and job seekers can apply for positions, manage their applications, and build professional profiles.

- Objective: Create a dynamic platform that connects job seekers with employers, streamlines the recruitment process, and provides tools for managing job applications and hiring.

- Notes:
  - Use personal github for source code management
  - Use free PostgreSQL hosting service

- Key Features:

- User Authentication and Authorization:

  Implement user registration and login using JWT and/or OAuth.

  Role-based access control (e.g., job seeker, employer, admin).

- Job Posting and Management:

  Employers can create, edit, and delete job postings.

  Job postings include details such as title, description, requirements, salary, and location.

- Profile Management:

  Job seekers can create and update professional profiles, including personal information, work experience, education, and skills.

Employers can create company profiles with information about their organization.

- Job Search and Application:

Implement search functionality for job seekers to find job postings based on keywords, categories, and filters.

Job seekers can apply for jobs directly through the platform, submitting resumes and cover letters.

- Application Tracking:

Employers can track and manage job applications, including viewing resumes.

Job seekers can track the status of their applications and receive notifications about updates. (Optional)

- Dashboard:

Provide job seekers with a dashboard to manage their profiles, applications, and job alerts.

Provide employers with a dashboard to manage job postings, applications, and candidate communications.

- Notifications (Optional):

Real-time notifications for job posting updates, application status changes, and new messages.

- Search and Filtering:

Search functionality to find job postings by keywords.

Filter job postings by categories, experience level, and location.

- Responsive Design:

Ensure the platform is mobile-friendly and responsive across different devices.

- Technical Requirements:

Backend (.NET Core 8):

Set up a RESTful API for managing users, job postings, applications, and notifications.

Implement WebSocket support for real-time updates.

Ensure secure authentication and authorization mechanisms.

Frontend (React):

Develop a user-friendly interface for creating, browsing, and applying for jobs.

Implement state management (e.g., Redux) for handling application state.

Use responsive design principles to ensure usability across devices.

Database (PostgreSQL):

Design the database schema to support users, job postings, applications, and notifications.

Ensure efficient querying and indexing for optimal performance.

- Suggested Timeline:

- ❑ Week 1-2: Project Setup and User Authentication/Authorization

- Week 1

- o Ticket 1.1: Project Setup

1. Set up the project repository and initialize .NET Core backend and React frontend projects.
2. Initialize .NET Core 8 project.
3. Initialize React project.
4. Set up PostgreSQL database.
5. Configure project structure in the repository.

- o Ticket 1.2: User Registration API

1. Implement the API endpoint for user registration.
2. Create user model and database schema.
3. Implement registration endpoint.
4. Validate user input.
5. Hash passwords before storing them in the database.

- o Ticket 1.3: User Login API

1. Implement the API endpoint for user login.
2. Implement login endpoint.
3. Validate user credentials.
4. Generate and return JWT token upon successful login.

- o Ticket 1.4: Frontend User Registration

1. Create a user registration form in React.
2. Design registration form UI.
3. Implement form validation.
4. Connect form to registration API.

- Ticket 1.5: Frontend User Login

1. Create a user login form in React.
2. Design login form UI.
3. Implement form validation.
4. Connect form to login API.

- Week 2

  - Ticket 2.1: User Authentication Middleware

1. Implement middleware for user authentication in .NET Core.
2. Create middleware to validate JWT tokens.
3. Protect API routes using the authentication middleware.

- Ticket 2.2: Role-Based Authorization

1. Implement role-based authorization in .NET Core.
2. Define user roles (e.g., job seeker, employer, admin).
3. Implement authorization middleware to check user roles.

- Ticket 2.3: Frontend Authentication Handling

1. Implement authentication handling in React.
2. Store JWT token securely in local storage.
3. Implement logic to redirect unauthenticated users to login page.
4. Protect routes in React based on user authentication status.

❑ Week 3-4: Job Posting and Profile Management

- Week 3

○ Ticket 3.1: Job Posting Model and Schema

1. Define the job posting model and database schema in .NET Core.
2. Create job posting model.
3. Define job posting schema in the database.

○ Ticket 3.2: Create Job Posting API

1. Implement the API endpoint for creating job postings.
2. Implement job posting creation endpoint.
3. Validate job posting data.
4. Save job posting details to the database.

○ Ticket 3.3: Frontend Job Posting Form

1. Create a form in React for employers to create job postings.
2. Design job posting form UI.
3. Implement form validation.
4. Connect form to create job posting API.

- Week 4

  - Ticket 4.1: List Job Postings API

1. Implement the API endpoint to list all job postings.
2. Implement endpoint to fetch job postings from the database.
3. Implement pagination for large data sets.

  - Ticket 4.2: Frontend Job Postings List

1. Create a React component to display a list of job postings.
2. Design job postings list UI.
3. Fetch job postings from the list job postings API.
4. Implement pagination in the frontend.

  - Ticket 4.3: Edit Job Posting API

1. Implement the API endpoint for editing job postings.
2. Implement job posting editing endpoint.
3. Validate edited data.
4. Update job posting details in the database.

  - Ticket 4.4: Frontend Job Posting Editing

1. Create a form in React for editing job postings.
2. Design job posting editing form UI.
3. Implement form validation.
4. Connect form to edit job posting API.
5. Week 5-6: Job Search, Application, and Tracking

- Week 5

- Ticket 5.1: Search API

1. Implement the API endpoint for searching job postings.
2. Implement search endpoint.
3. Enable searching by keywords.

- Ticket 5.2: Frontend Search Functionality

1. Implement search functionality in React.
2. Design search UI.
3. Connect search input to search API.
4. Display search results.

- Ticket 5.3: Application Model and Schema

1. Define the application model and database schema in .NET Core.
2. Create application model.
3. Define application schema in the database.

- Ticket 5.4: Apply for Job API

1. Implement the API endpoint for job seekers to apply for jobs.
2. Implement job application endpoint.
3. Validate application data.
4. Save application details to the database.

Week 6

- Ticket 6.1: Frontend Job Application Form

1. Create a form in React for job seekers to apply for jobs.

2. Design job application form UI.
3. Implement form validation.
4. Connect form to job application API.

- Ticket 6.2: Application Tracking API

1. Implement the API endpoint for tracking job applications.
2. Implement endpoint to fetch application status.
3. Notify job seekers of application updates.

- Ticket 6.3: Frontend Application Tracking

1. Create a React component for job seekers to track their applications.
2. Design application tracking UI.
3. Fetch application status from the application tracking API.
4. Display application updates and notifications.

❑ Week 7-8: Dashboard, Notifications(Optional), and Final Polish

- Week 7

- Ticket 7.1: Job Seeker Dashboard Backend

1. Implement the backend logic for the job seeker dashboard.
2. Create endpoints to fetch profile, applications, and notifications data.
3. Implement logic to track job search progress and application status.

- Ticket 7.2: Job Seeker Dashboard Frontend

1. Create a dashboard for job seekers in React.
2. Design dashboard UI.
3. Fetch and display profile, applications, and notifications data.
4. Implement real-time updates for application status and messages.

- Ticket 7.3: Employer Dashboard Backend

1. Implement the backend logic for the employer dashboard.
2. Create endpoints to fetch job postings, applications, and notifications data.
3. Implement logic to manage job postings and track applications.

- Week 8

○ Ticket 8.1: Employer Dashboard Frontend

1. Create a dashboard for employers in React.
2. Design dashboard UI.
3. Fetch and display job postings, applications, and notifications data.
4. Implement real-time updates for job applications and messages.

○ Ticket 8.2: Real-Time Notifications Backend (Optional)

1. Implement backend logic for real-time notifications.
2. Create notification model and schema.
3. Implement logic to send notifications via WebSockets.

○ Ticket 8.3: Real-Time Notifications Frontend (Optional)

1. Implement frontend logic to handle real-time notifications.

○ Establish WebSocket connection for notifications. (Optional)

1. Display notifications in the UI.
2. Week 9: Search, Filtering, Comments, and Updates

- Week 9

  - Ticket 9.1: Filter API

1. Implement API endpoints for filtering job postings.
2. Implement filtering logic based on categories, experience level, and location.

  - Ticket 9.2: Frontend Filtering Functionality

1. Implement filtering functionality in React.
2. Design filtering UI.
3. Connect filter options to filter API.
4. Display filtered results.

  - Ticket 9.3: Job Posting Updates API

1. Implement the API endpoint for job posting updates.
2. Create update model and schema.
3. Implement CRUD operations for updates.

  - Ticket 9.5: Frontend Job Posting Updates Section

1. Implement job posting updates section in React.
2. Design updates UI.
3. Fetch and display updates.
4. Implement adding new updates.

❑ Week 10: Final Testing and Bug Fixing

- Week 10

  - Ticket 10.1: Integration Testing

1. Perform integration testing for all major functionalities.
2. Write and execute integration tests for APIs.

3. Ensure all components work together seamlessly.

- Ticket 10.2: Unit Testing

1. Conduct Unit Testing  to ensure the application meets user requirements.
2. Define Unit Testing scenarios.

- Ticket 10.3: Bug Fixing

1. Fix any bugs identified during testing.
2. Prioritize and resolve bugs.
3. Re-test to ensure issues are resolved.

- Ticket 10.4: Documentation

1. Create comprehensive documentation for the project.
2. Document API endpoints.
3. Ensure code comments and documentation are up-to-date.