Frontend Design Document

Job Portal with Smart Resume–Job Matching

1. Overview

This document describes the frontend design for the Job Portal application. The frontend is built with **React** and **TailwindCSS** and communicates with a FastAPI backend. Focus areas: pages & routes, component hierarchy, state management, API integration, responsive design, accessibility, validation, and sample component structure.

2. Tech Stack

- HTML5, CSS3, JavaScript (ES6+)
- React (Functional components + Hooks)
- TailwindCSS for styling
- Axios or Fetch for API calls
- React Router for client-side routing
- Formik + Yup (optional) for form handling & validation
- Context API or Redux (optional) for global state
- Vite or Create React App as the build tool

3. Pages & Routes

Routes (React Router):
- / : Landing / Home (public) — overview, hero, features
- /signup/candidate : Candidate signup form
- /login/candidate : Candidate login
- /signup/recruiter : Recruiter signup
- /login/recruiter : Recruiter login
- /dashboard/candidate : Candidate dashboard — recommended jobs, profile, applications
- /dashboard/recruiter : Recruiter dashboard — posted jobs, view candidates
- /jobs/:id : Job details page — apply button, job description
- /admin : Admin panel (protected) — manage users & jobs
- /profile : Edit profile, upload resume (candidate)

4. Component Hierarchy

Top-level layout components:
- App (Router)
- NavBar (responsive) — shows links based on auth and role
- Footer
- ProtectedRoute (HOC) — checks JWT and role

Feature components:
- Auth:
- SignupForm (Candidate/Recruiter)
- LoginForm
- Candidate Dashboard:
- RecommendedJobsList
- JobCard (shows title, company, short desc, similarity score)
- ProfileCard (with resume upload)
- Recruiter Dashboard:
- PostJobForm
- JobList (with top candidates link)
- CandidateShortlist (list of candidate cards with similarity score)
- Jobs:
- JobDetails

- ApplyModal / ApplyForm
- Common:
- Toast / Notification
- Modal
- LoadingSpinner
- ErrorBoundary

5. State Management

- Use React Context for auth state (user info + JWT token) and theme.
- Local component state for forms and lists.
- For complex needs (large app), use Redux or Zustand.
- Persist auth token in secure, HttpOnly cookies (preferred) or localStorage (if cookies not available).
- Example auth flow:
- On login, backend returns JWT and refresh token. Store refresh token in HttpOnly cookie, access token in memory or short-lived storage.
- Attach Authorization: Bearer header to API requests.

6. API Integration (examples)

Use Axios with an API client module to centralize calls and interceptors (for auth).

Example: Login (candidate)
Request (POST /api/candidates/login):
{ "email": "john@example.com", "password": "pass123" }
Response:
{ "token": "JWT_TOKEN_STRING", "user": { "id": 1, "name": "John Doe", "role": "candidate" } }

Example: Get recommended jobs (GET /api/candidates/recommended_jobs):
Headers: Authorization: Bearer <token>
Response: [{ "job_id": 10, "title": "Data Scientist", "similarity_score": 0.92 }, ...]

7. UX & UI Guidelines

- Mobile-first responsive design using Tailwind utility classes.
- Use consistent spacing, typography, and component sizes.
- Display similarity score prominently on JobCard (e.g., badge with percentage).
- Provide clear CTAs: Apply, Upload Resume, Post Job.
- Show loading states and skeletons while fetching data.
- Use accessible color contrast and semantic HTML (buttons, labels).

8. Security Considerations (Frontend)

- Do not store sensitive tokens in localStorage when possible; prefer HttpOnly cookies.
- Sanitize any HTML content before rendering (avoid dangerouslySetInnerHTML).
- Validate inputs on client-side, but always re-validate on server-side.
- Implement Content Security Policy (CSP) headers from backend.
- Use same-site cookies to mitigate CSRF.
- Rate-limit actions like login attempts via backend; show CAPTCHA after repeated failures.

9. Accessibility

- Use semantic HTML elements and ARIA attributes where necessary.
- Ensure keyboard navigability and focus outlines for interactive elements.
- Provide alt text for images and aria-labels for icons.
- Test with Lighthouse and axe for WCAG compliance.

10. Forms & Validation

- Use Formik + Yup for robust form management and validation, or React Hook Form.
- Validate: email format, password strength, required fields, resume file type/size.
- Show inline error messages and disable submit until valid.

11. Sample Component Structure (React)

```
// src/App.jsx import { BrowserRouter as Router, Routes, Route } from 'react-router-dom'; import { AuthProvider }
from './context/AuthContext'; import Home from './pages/Home'; import CandidateDashboard from
'./pages/CandidateDashboard'; import RecruiterDashboard from './pages/RecruiterDashboard'; import Login from
'./pages/Login'; import Signup from './pages/Signup'; import ProtectedRoute from './components/ProtectedRoute';
function App() { return ( } /> } /> } /> } /> } /> ); export default App;
```

12. Testing & QA

- Unit test components with React Testing Library and Jest.
- End-to-end tests with Cypress or Playwright (signup, login, job apply flow).
- Performance testing: Lighthouse score checks and bundle size optimization.

13. Deployment

- Build using Vite/CRA production build.
- Serve static frontend via CDN or platforms like Netlify, Vercel, or from backend (FastAPI with StaticFiles).
- Use environment variables for API base URL and keys.