System Design Document (SDD)

Project Name: TechBoard Prepared by: Arun kundu Date: May 17, 2025

Date	version	Prepared by	Checked by
20/03/2025	1.0.0	Arun kundu	

1. Introduction

1.1 Purpose

This document outlines the software design for **TechBoard**, a global job board platform designed for the tech industry. It aims to provide job seekers and hiring companies with an AI-driven system that optimizes hiring, improves resumes, and matches skills with opportunities.

1.2 Scope

The platform will support the following stakeholders:

- **Job Seekers:** Search and apply for jobs, receive recommendations, build resumes.
- **Employers/Recruiters:** Post job listings, review applicants, manage freelance contracts.
- **Admins:** Manage platform content, monitor activity, and moderate listings.

2. System Overview

2.1 Core Features

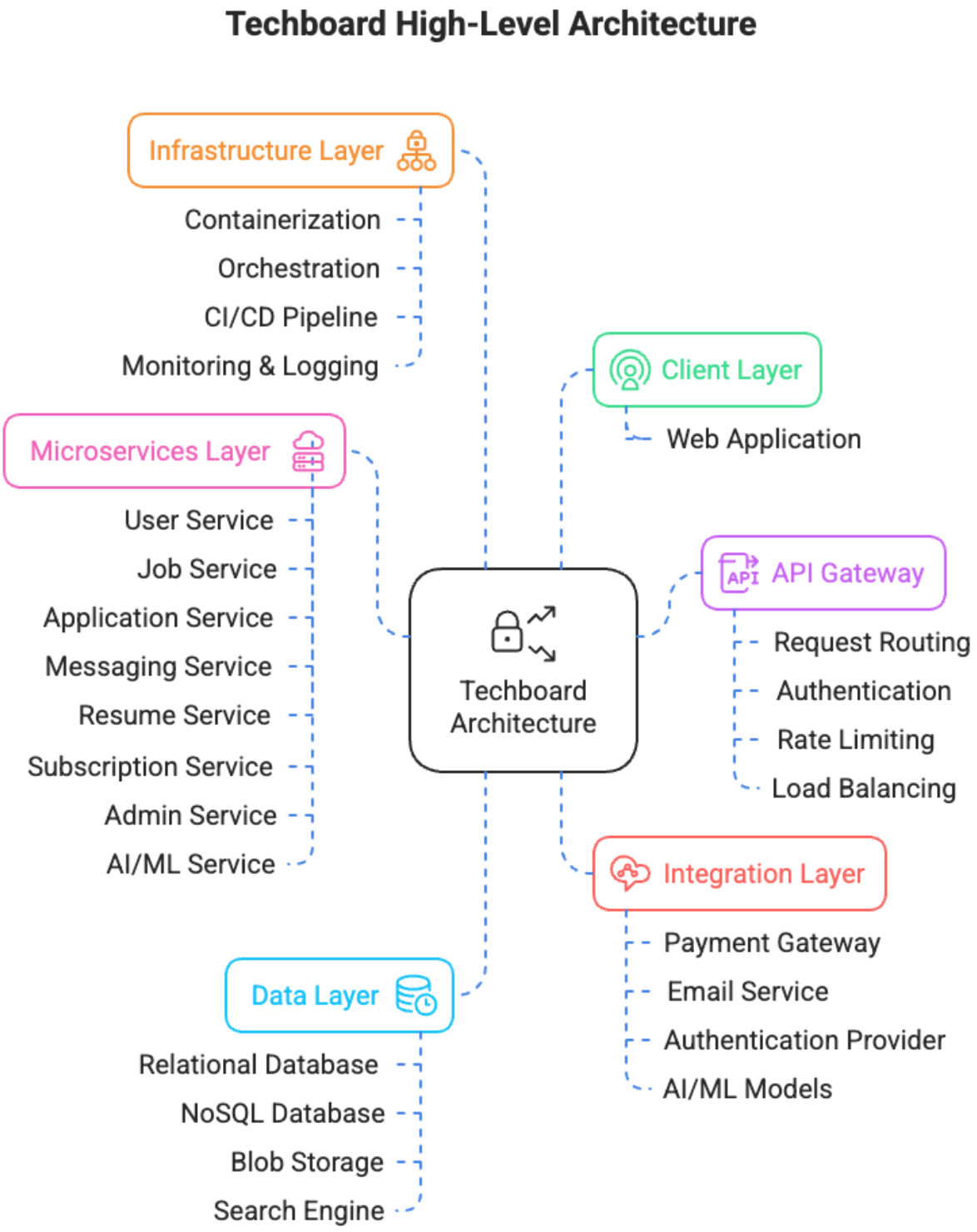
- **Job Listings:** Local, remote, freelance positions with filters for tech stack, location, salary, and experience.
- **Recruiter Platform:** Tools for posting, managing, and analyzing job listings.
- **Job Seeker Platform:** Profiles, resume building, job recommendations, and skill suggestions.
- **AI-Powered Features:** Resume grading, job recommendations, career path predictions.

- **Contract Hiring System:** Freelance project-based system with milestone tracking, chat, and file sharing.
- **Subscription System:** Access to features based on free, pro, or enterprise plans for both job seekers and recruiters.
- **Admin Panel:** Platform moderation, user management, and billing controls.

2.2 Stakeholders

- Job Seekers
- Employers/Recruiters
- Platform Administrators

3. Architecture diagram



4. Component Breakdown

4.1 Frontend (Nuxt 3, TypeScript, Tailwind CSS)

- **Pages:** Job Listings, Resume Builder, User Profile, Application Tracker, Admin Dashboard.
- **Key Features:** SSR/CSR hybrid rendering, OAuth2 authentication, responsive UI, accessibility-compliant design.

4.2 Backend API (NestJS or Go)

- **RESTful API Design:** Secure, stateless architecture with JWT/OAuth2 for user authentication and Refresh token flow, Integrate API versioning etc
- **Key Services:** User, Job, Application, Resume, Messaging, Admin.
- **Real-Time Support:** WebSocket/SSE for live notifications and updates.

4.3 Database (PostgreSQL)

- **Normalized Schema:** Efficient relational model for job seekers, employers, applications, and job listings.
- **Indexes:** Fast querying for job searches and recommendations.
- Use read replicas in future to scale read-heavy operations (job search, AI reads).

4.4 Caching (Redis)

- **Job and Skill Data:** Cache frequently accessed information for quicker responses.
- **Queues:** Redis queues for AI processing and notifications.
- Set TTLs (time-to-live) on cache to keep data fresh.

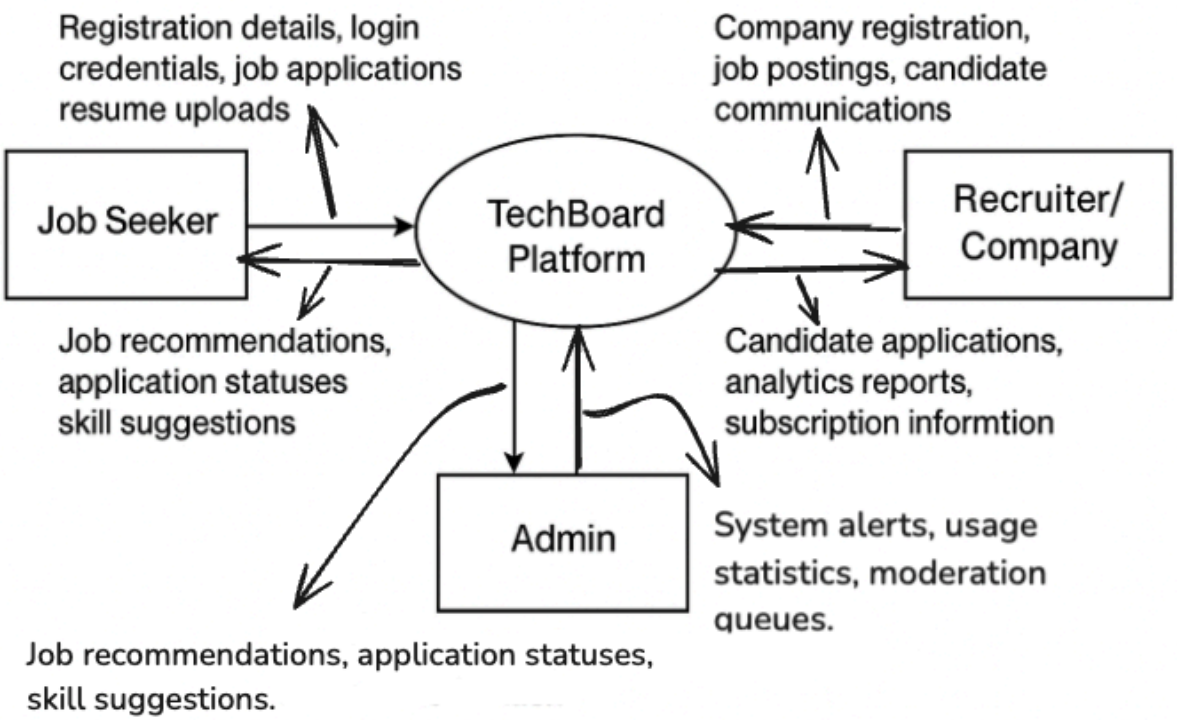
4.5 AI Microservices (Python)

- **Resume Parsing & Grading:** Powered by AI to analyze resumes and give feedback.
- **Job-Skill Matching:** Semantic algorithms to match job seekers with appropriate job listings.

4.6 File Storage (AWS S3 or Cloudinary)

- **Resume & Portfolio Storage:** Safe and scalable storage for media files.
- **Content Delivery Network (CDN):** Optimized global delivery for assets.

5. Data Flow Diagram



6. Database Design

6. Security

- Authentication: OAuth2 with JWT tokens for secure user sessions.
- Access Control: Role-based access control (RBAC) for job seekers, recruiters, and admins.
- Input Validation: Prevent SQL injection, XSS, and other security threats.

- Protection: Rate limiting, CSRF protection, and optional two-factor authentication.

7. Performance & Scalability

- Microservices Architecture: Independent services for scalability.
- Caching: Redis caching for frequently accessed job data and recommendations.
- Search Optimization: Elasticsearch for fast, semantic search capabilities.
- AI Services: Asynchronous processing to handle large-scale resume evaluations and job matching.
- Auto-Scaling: Kubernetes handles scaling based on demand.
- Database Optimization: PostgreSQL indexing and query optimization for fast job searches.

8. Deployment Strategy

- Containerization: Docker images for all services, managed by Kubernetes for orchestration.
- CI/CD Pipeline: GitHub Actions for continuous integration and deployment.
- Deployment Strategies: Blue/green or rolling deployments to minimize downtime.
- Environment Setup: Separate dev, staging, and production environments.

9. Monitoring & Logging

- Monitoring: Prometheus and Grafana for real-time monitoring and metrics.
- Logging: ELK stack or Loki for logging.
- Alerting: Alertmanager to notify admins of any issues.
- Error Reporting: Sentry for capturing application errors.