CHANDIGARH UNIVERSITY

Discover. Learn. Empower.

UNIVERSITY INSTITUTE OF ENGINEERING

PROJECT REPORT

SKILLSYNC - AI RECRUITMENT PLATFORM

Subject Name – Project Based Learning in Java

Subject Code – 23CSH-304

Submitted To:
Er. Deep Prakash Gupta
(E18557)

Submitted By:
Name:Mandeep Kaur
UID: 23bcs10854
Section: KRG_2B

SkillSync: System Design Document

---

1. Introduction & System Goals

This document provides a comprehensive technical overview of the SkillSync platform, detailing its system architecture, core components, and underlying data flows. It is intended for developers, architects, and key stakeholders to understand the platform's design and functionality.

The primary goal of the SkillSync project is to create an AI-powered recruitment platform that intelligently connects job seekers with recruiters. This is achieved through a set of integrated features designed to streamline the hiring process. Key capabilities include AI-driven matching of resume content against job skill requirements, real-time database integration using MongoDB, and the provision of distinct, role-based dashboards for Recruiters, Job Seekers, and Administrators.

This document will first detail the specific technologies chosen to build the platform, illustrating how they support its core objectives.

2. Technology Stack

The SkillSync technology stack is strategically selected to ensure a scalable, responsive, and high-performance application. The chosen technologies across the frontend, backend, and database layers work in concert to deliver a seamless user experience while providing a robust foundation for future development.

The core components of the stack are outlined below.

Core Technology Stack

Layer	Technology	Purpose
Frontend	React (Vite), Axios, TailwindCSS / Custom CSS	Building a dynamic, responsive, and modern single-page application (SPA) for the user interface. Axios is used for efficient API communication with the backend.
Backend	FastAPI (Python)	Creating high-performance, asynchronous RESTful API endpoints that serve as the logical core of the platform.
Database	MongoDB	Providing a flexible, scalable, document-based NoSQL database ideal for storing semi-structured data such as job postings and applications.
AI Logic	NLP keyword-based Resume–Skill Matching (PyPDF2 + regex)	Implements NLP keyword-based skill matching by parsing PDF resumes with PyPDF2 and using regular expressions or substring searches for keyword analysis.
Storage	Local /uploads directory	Offering a direct and simple file persistence solution for

		storing uploaded PDF resumes.
Middleware	CORS Middleware	Enabling secure cross-origin communication, allowing the React frontend to interact with the FastAPI backend server running on a different port.

With an understanding of the technologies used, the next section will detail how data is structured and stored within the MongoDB database.

3. Data Architecture

MongoDB serves as the central data repository for the SkillSync platform. Its flexible, document-based model is exceptionally well-suited for storing the varied and semi-structured information associated with job postings and candidate applications. The data is organized into logical collections, each serving a distinct purpose.

MongoDB Collections

jobs

This collection is the primary repository for all job postings. Each document within this collection represents a unique job opportunity uploaded by a recruiter, containing details such as job title, description, and required skills.

applications

This collection stores all applications submitted by job seekers. Each document links a job seeker to a specific job from the jobs collection and includes information about the applicant, including a reference to their uploaded resume.

Collections for Future Implementation

The following collections are planned to support future enhancements and are not yet implemented:

- **recruiters**: Intended to store detailed profile information for recruiter accounts.
- **seekers**: Intended to store detailed profile information for job seeker accounts.

Database Access Layer (mongodb.py)

All interactions between the FastAPI backend and the MongoDB database are managed through a dedicated data access layer. This layer contains functions that encapsulate database operations, promoting clean and maintainable code.

- insert_job(job): Inserts a new job document into the jobs collection.
- fetch_all_jobs(): Retrieves all job documents from the jobs collection.
- fetch_jobs_by_title(title): Fetches job documents from the jobs collection that match a specific title keyword.
- apply_for_job(application): Inserts a new application document into the applications collection.

- fetch_applicants_by_job(job_title): Retrieves all application documents for a specific job title from the applications collection.

These data access functions are consumed by specific API endpoints to execute business logic. The insert_job(job) function is called exclusively by the POST /api/recruiter/upload-job endpoint. Similarly, apply_for_job(application) is triggered by POST /api/apply-job, while the various fetch functions serve their corresponding GET endpoints for recruiters and job seekers.

This structured data layer provides the foundation for the backend logic, which is exposed through a series of RESTful API endpoints.

4. Backend Architecture & API Endpoints

The backend of the SkillSync platform is built using FastAPI, a modern, high-performance Python web framework. It serves as the stateless logic engine, exposing a RESTful API to orchestrate data flow between clients and the database. The API is logically organized by user role and functionality.

System & Utility

These endpoints are used for system health checks, debugging, and serving static files.

Method	Endpoint	Description
GET	/	Root route used as a basic backend health check.
GET	/api/debug/jobs	Shows all jobs in the database for debugging purposes.
Static	/uploads/{filename}	Serves uploaded PDF resumes, making them accessible in the browser.

Recruiter APIs

These endpoints provide the functionality required for the Recruiter Dashboard.

Method	Endpoint	Description
POST	/api/recruiter/upload-job	Allows a recruiter to upload a new job posting.
GET	/api/recruiter/jobs	Fetches all jobs that have been uploaded by recruiters.
GET	/api/recruiter/applicants?job_title=XYZ	Retrieves a list of all applicants for a specific job title.

Job Seeker APIs

These endpoints support the features available on the Job Seeker Dashboard.

Method	Endpoint	Description
GET	/api/jobs?title=developer	Search for jobs by title or fetch all jobs.
POST	/api/upload-resume	A mock endpoint for uploading a resume.
POST	/api/match-resume?job_skills=python,sql	Triggers the AI-based skill matching against an uploaded resume.
POST	/api/apply-job	Allows a job seeker to apply for a job, including a resume

		file upload.

Admin APIs (Future Scope)

These endpoints are designed for the Admin Dashboard but are not yet implemented on the backend. Their development is part of the future project roadmap.

Method	Endpoint	Description
GET	/api/admin/stats	Planned to provide a summary of platform statistics.
DELETE	/api/admin/job/{id}	Planned to allow an administrator to delete any job posting.
DELETE	/api/admin/application/{id}	Planned to allow an administrator to remove any application.
GET	/api/admin/users	Planned to fetch a list of all recruiters and job seekers.

Among these endpoints, the AI-powered skill-matching service is the most complex and serves as a core differentiator for the platform. Its workflow is examined in the next section.

5. Core AI Component: Resume-Skill Matching

The AI-powered resume-skill matching module is SkillSync's key value proposition. It provides job seekers with immediate, data-driven feedback on their qualifications for a role *before* they formally apply, empowering them to make more informed career decisions. This functionality is exposed through a dedicated API endpoint.

API Endpoint: POST /api/match-resume

The workflow for the AI matching logic proceeds as follows:

1. The system receives a request containing an uploaded resume file (PDF) and a comma-separated string of required job_skills.
2. The PyPDF2 library is used to parse the PDF resume and extract its raw text content page by page.
3. Both the extracted resume text and the input job_skills are normalized by converting them to lowercase, ensuring a case-insensitive comparison.
4. The system iterates through the list of required skills and performs a case-insensitive substring search for each required skill within the extracted resume text.
5. A match_percentage is calculated as the ratio of skills found in the resume to the total number of required skills.

The endpoint returns a structured JSON object to the frontend, providing a detailed breakdown of the analysis.

```
{
  "match_percentage": 68.75,
  "found_skills": ["python", "sql"],
  "missing_skills": ["machine learning"],
  "status": "eligible"
}
```

This module is strategically integrated into the Job Seeker Dashboard, where the results are displayed to the user just before they are presented with the "Apply Now" option. This

immediate feedback loop is central to the platform's user experience, guiding applicants toward their best-fit opportunities.

The next section will detail the frontend dashboards where this data is consumed and presented to the user.

6. Frontend Architecture & User Dashboards

The SkillSync frontend is a modern single-page application (SPA) built with React. It is designed to deliver a clean, responsive, and intuitive user experience tailored to the specific needs of each user role. This is achieved through three distinct, role-based dashboards.

Recruiter Dashboard

- **Primary Component:** RecruiterDashboard.jsx
- **Core Capabilities:**
    - Upload new job postings via a web form, which calls the POST /api/recruiter/upload-job endpoint to create a new record in the jobs collection.
    - View a comprehensive list of all jobs previously uploaded.
    - View a detailed list of applicants for each job by calling GET /api/recruiter/applicants, which fetches data from the applications collection.
    - Access applicant resumes directly through clickable URLs that point to the static file server path (/uploads/{filename}).

Job Seeker Dashboard

- **Primary Component:** UserDashboard.jsx
- **Core Capabilities:**
    - Search and filter job listings by title using the GET /api/jobs endpoint.
    - Upload a resume to perform an AI-powered skill match against a job's requirements via the POST /api/match-resume endpoint.
    - Review the skill match percentage, including lists of found and missing skills, before deciding to apply.
    - Apply for jobs via the POST /api/apply-job endpoint, which triggers the backend to store the resume in the /uploads/ directory and create a new record in the applications database collection.

Admin Dashboard

- **Primary Component:** AdminDashboard.jsx
- **Current Status:** The user interface for this dashboard is fully built, featuring an analytics-style layout with tables and action buttons. However, it currently relies on mock data for display. Full backend integration with the /api/admin/* endpoints is planned for a future development phase.

The interaction between these frontend components and the backend APIs creates a seamless data flow, which is illustrated in the following section.

7. System Interaction & Data Flow

To provide a clear picture of how SkillSync operates, this section illustrates the end-to-end data flow for a typical user journey. It demonstrates how actions initiated on the frontend trigger specific backend API calls, which in turn lead to interactions with the database.

The complete lifecycle from a recruiter posting a job to viewing applications is detailed below.

Step	User Action	Triggered API Endpoint	Database Effect
1	A recruiter submits a new job form.	POST /api/recruiter/upload-job	A new document is inserted into the jobs collection.
2	A job seeker searches for a job.	GET /api/jobs	Data is fetched from the jobs collection to display listings.
3	The job seeker checks their resume fit.	POST /api/match-resume	No database effect; AI logic performs an in-memory skill check.
4	The job seeker applies for the job.	POST /api/apply-job	A new document is inserted into the applications collection.
5	The recruiter checks for new applicants.	GET /api/recruiter/applicants	Data is fetched from the applications collection for the specific job.
6	(Future) An admin reviews platform activity.	GET /api/admin/*	Data is aggregated from various collections.

This flow is enabled by specific system configurations that allow the different parts of the platform to communicate securely and manage files effectively.

8. System Configuration & Deployment

The successful operation of the SkillSync platform depends on several key configurations that govern communication protocols and file management. These settings ensure that the frontend and backend can interact securely and that user-uploaded files are stored and served correctly.

CORS Configuration

A Cross-Origin Resource Sharing (CORS) middleware is implemented in the FastAPI backend to manage requests from the frontend. This is essential because the React development server (http://localhost:5173) and the FastAPI server (http://127.0.0.1:8000) run on different origins.

The configuration specifies the following rules:

- allow_origins: Restricts access to requests originating from http://localhost:5173.
- allow_credentials: Set to True to permit cookies or other credentials in requests.
- allow_methods: Allows all standard HTTP methods ("*").
- allow_headers: Allows all request headers ("*").

Static File Serving for Uploads

The system is configured to handle the storage and retrieval of uploaded resumes. All resumes are saved directly to the /uploads/ directory in the project's root. To make uploaded files web-accessible, FastAPI statically mounts the local /uploads/ directory to the /uploads URL path.

With the current system defined, the final section outlines the roadmap for future enhancements.

9. Future Scope

While the core system is production-ready, the following roadmap outlines planned enhancements to evolve the platform into a more comprehensive and mature recruitment ecosystem.

Planned Enhancements

- **Full Backend Implementation for Admin APIs:** The existing admin UI will be connected to a fully functional backend. This includes implementing the planned GET and DELETE endpoints to allow administrators to manage jobs, applications, and user accounts directly from their dashboard.
- **New Database Collections:** To support more detailed user management, the optional recruiters and seekers collections will be implemented. These collections will store comprehensive profile information, enabling more personalized experiences and advanced platform features.
- **Optional Microservice Integration:** The architecture is designed with the potential for future microservice integration. A service, such as one built with Spring Boot, could be introduced to handle specific concerns like user authentication or notifications, further decoupling the system and enhancing scalability.

These future developments are focused on building upon the strong foundation of the current system to create an even more powerful and feature-rich platform for recruiters and job seekers alike.