**Project Blueprint — *MatchFoundry***

**An AI-powered resume–job matching platform** built with **React + Spring Boot + GraphQL + FastAPI (NLP/ML)**.

## **High-Level Product Overview**

### **MVP (GraphQL-ready)**

* Upload resume → parsed to structured JSON.
* Choose job titles or paste job description links.
* System fetches sample JDs (seed dataset or official APIs).
* Compute match score (% alignment) using skills and experience.
* Dashboard (React) uses **GraphQL queries** to fetch combined data in one shot:

query {

user(id: 1) {

name

resumes {

id

skills

matches {

score

missingSkills

job {

title

company

}

}

}

}

}

* Show matched/missing skills and improvement suggestions.
* Optional: Integrate LeetCode / Codeforces stats.

### **Advanced (Stretch Phase)**

* Use **embeddings + semantic search** for similarity (e.g., SentenceTransformers).
* Automated job ingestion via APIs (LinkedIn/Indeed).
* Personalized learning plan from missing skills.
* AI-generated cover letter.
* Shareable recruiter profile (public URL via GraphQL query).
* Multi-tenant deployment + RBAC via GraphQL directives.

## **Architecture Overview (GraphQL-Centric)**

### **🧩 Core Concept**

* **Frontend ↔ GraphQL Gateway (Spring Boot)** — single entrypoint.
* **Microservices ↔ REST APIs** — internally modular and decoupled.

### **Architecture Layout**

frontend (React + Apollo Client)

↕

GraphQL Gateway (Spring Boot + GraphQL)

↕

├── auth-service (REST, Spring Boot)

├── resume-service (REST, Spring Boot)

├── nlp-service (REST, Python FastAPI)

├── job-service (REST, Spring Boot)

├── match-service (REST, Python/Spring Boot)

├── profile-service (REST, Spring Boot)

### 🧠 **Data Flow Example**

1. Frontend sends a GraphQL query requesting user → resumes → matches → job.
2. GraphQL **resolvers** call respective microservices’ REST endpoints.
3. Responses are aggregated and shaped as per the GraphQL schema.
4. Client gets only the data it requested — no overfetching.

## **GraphQL API Gateway Design**

### **Tech Stack:**

* **Spring Boot 3+**
* **spring-boot-starter-graphql** (GraphQL Java)
* **Spring Cloud Gateway** (optional for routing)
* **WebClient / FeignClient** for calling REST microservices
* **DataLoader** for batch fetching (avoid N+1 problem)

### **Example Schema**

**user.graphqls**

type User {

id: ID!

name: String!

email: String!

resumes: [Resume]

}

type Query {

user(id: ID!): User

}

**resume.graphqls**

type Resume {

id: ID!

skills: [String]

experience: [String]

matches: [MatchResult]

}

**match.graphqls**

type MatchResult {

id: ID!

score: Float!

matchedSkills: [String]

missingSkills: [String]

job: Job

}

**job.graphqls**

type Job {

id: ID!

title: String!

company: String!

location: String

}

### **Example Resolver (Spring Boot)**

@Component

public class UserResolver implements GraphQLQueryResolver {

private final ResumeServiceClient resumeClient;

private final MatchServiceClient matchClient;

public UserResolver(ResumeServiceClient resumeClient, MatchServiceClient matchClient) {

this.resumeClient = resumeClient;

this.matchClient = matchClient;

}

public User getUser(Long id) {

User user = // fetch from Auth service

user.setResumes(resumeClient.getResumesByUser(id));

return user;

}

}

## **Microservices Summary (Internal REST)**

Each service exposes REST endpoints used by the GraphQL gateway.

| **Service** | **Role** | **Tech Stack** |
| --- | --- | --- |
| **Auth Service** | Login, JWT, user profile | Spring Boot + Spring Security |
| **Resume Service** | Upload, parse resume, store structured JSON | Spring Boot + Apache Tika / PDFBox |
| **NLP Service** | Skill extraction, embeddings | FastAPI + spaCy + SentenceTransformers |
| **Job Service** | Job ingestion & caching | Spring Boot + Postgres |
| **Match Service** | Compute match score, generate suggestions | FastAPI / Spring Boot |
| **Profile Service** | Integrate coding profiles | Spring Boot + REST APIs |
| **Analytics Service (optional)** | Track match trends & feedback | Spring Boot + Postgres |

## **Datastores**

* **PostgreSQL** — structured data (users, resumes, jobs).
* **PGVector** — embeddings storage for semantic search.
* **Redis** — cache recent GraphQL results / sessions.
* **S3** — resume file storage.

## **Matching Algorithm (unchanged logic)**

Implemented inside **match-service** (Python):

def compute\_match(resume\_skills, jd\_skills):

resume\_vecs = embed(resume\_skills)

jd\_vecs = embed(jd\_skills)

matches = sum(1 for jd in jd\_vecs if max\_cosine(jd, resume\_vecs) > 0.78)

skill\_score = matches / len(jd\_vecs)

return 0.5 \* skill\_score + 0.2 \* seniority + 0.15 \* keywords + 0.1 \* title\_sim + 0.05 \* extras

GraphQL Gateway consumes this via REST:

mutation {

computeMatch(resumeId: 123, jobId: 456) {

score

matchedSkills

missingSkills

}

}

## **Frontend (React + Apollo Client)**

### **Stack**

* React + TypeScript + Tailwind
* Apollo Client (GraphQL queries & caching)
* React Router, Zustand/Redux for state
* React Query for hybrid fetching (if mixing REST)

### **Example** **Query**

const USER\_QUERY = gql`

query GetUser($id: ID!) {

user(id: $id) {

name

resumes {

skills

matches {

score

job { title company }

}

}

}

}

`;

const { data, loading } = useQuery(USER\_QUERY, { variables: { id: 1 } });

## **CI/CD & Deployment (GraphQL-Aware)**

* **GitHub Actions** → build/test/lint → Dockerize each service.
* **Docker Compose** → local setup.
* **Deployment**:
  + Frontend → Vercel or Netlify
  + GraphQL Gateway + microservices → AWS EC2 / Railway / Render
  + DB → AWS RDS (Postgres)
  + Storage → S3
* **Monitoring** → Grafana + Prometheus
* **Error Tracking** → Sentry
* **GraphQL Playground** → GraphiQL or Apollo Sandbox enabled.

## **Security & Privacy**

* TLS + JWT across GraphQL layer.
* Limit query depth to prevent overloading (maxQueryDepth).
* S3 pre-signed URLs for resume uploads.
* Data encryption for PII.
* Respect job board API terms & rate limits.

## **Testing Strategy**

| **Layer** | **Tool** | **Focus** |
| --- | --- | --- |
| **GraphQL Gateway** | JUnit + SpringGraphQLTest | Schema validation, resolver integration |
| **Microservices** | JUnit / Pytest | Business logic testing |
| **Frontend** | Jest + RTL | Component + GraphQL query mocks |
| **Integration** | Pact / Postman | Contract tests between GraphQL → REST services |
| **E2E** | Cypress | Upload → Match → Dashboard flow |

## **8-Week Roadmap (GraphQL Variant)**

| **Week** | **Goals** |
| --- | --- |
| **0** | Create mono-repo, GraphQL schema draft, CI pipeline skeleton. |
| **1** | Auth & Resume microservices (REST). GraphQL Gateway scaffolding. |
| **2** | Resume parsing → GraphQL integration. |
| **3** | Job service & match service (basic keyword match). |
| **4** | Frontend setup (React + Apollo Client). Query GraphQL for user → matches. |
| **5** | Add LeetCode/CF integrations + suggestions. |
| **6** | Implement embeddings (NLP service) + semantic matching. |
| **7** | Dockerize all services, integrate CI/CD. |
| **8** | Deploy to demo environment, publish documentation + walkthrough. |

## **Example Repository Structure**

matchfoundry/

│

├── frontend/ # React + Apollo Client

├── gateway-graphql/ # Spring Boot GraphQL Gateway

│ ├── src/main/resources/graphql/

│ │ ├── user.graphqls

│ │ ├── resume.graphqls

│ │ └── match.graphqls

│ ├── resolvers/

│ └── clients/ # REST service clients

│

├── services/

│ ├── auth-service/

│ ├── resume-service/

│ ├── job-service/

│ ├── match-service/

│ └── nlp-service/

│

├── docker-compose.yml

└── README.md

## **Resume & Interview Highlights**

**Add to Resume:**

“Built a distributed AI-driven resume–job matching platform with GraphQL API gateway aggregating multiple microservices (Spring Boot + FastAPI). Implemented semantic skill matching via embeddings and reactive GraphQL queries for optimal frontend efficiency.”

**You’ll Demonstrate:**

* System design & API federation skills
* ML + NLP integration in production
* GraphQL schema design mastery
* Microservice orchestration
* Full-stack deployment (React + Spring Boot + FastAPI + AWS)

## **TL;DR Summary**

✅ REST for microservices (Auth, Resume, NLP, Match, Job)  
✅ GraphQL Gateway (Spring Boot) to unify data access  
✅ React + Apollo Client frontend  
✅ FastAPI for ML/NLP intelligence  
✅ Docker + GitHub Actions + AWS deployment  
✅ Resume parsing, job matching, embeddings, and suggestions  
✅ Clean GraphQL schema + one unified API endpoint