🗂️ Project Title: Automated Screener Project

🔍 Goal:

Automatically classify resumes by field and rank them against job descriptions using NLP + ML + Full Stack Deployment.

🧭 Project Timeline

📅 Phase 1: Project Planning & Requirements Gathering

Timeframe: Week 1
Activities:

- Defined project goals (CV classification + job matching).
- Decided tech stack:
    - ML: Random Forest, TF-IDF, Sentence-BERT.
    - Backend: FastAPI.
    - Frontend: React + TypeScript.
    - Deployment: Docker + AWS.

Challenges:

- Planning how to represent resumes and jobs in a comparable format.
- Choosing models that balance speed and accuracy for live predictions.

📅 Phase 2: Dataset Collection & Cleaning

Timeframe: Week 2–3
Activities:

- Collected 1000+ resumes from open datasets and scraped sources.
- Manually labeled data for supervised learning.
- Cleaned and normalized resume data using regular expressions and NLP preprocessing.

Challenges:

- Resume formats were inconsistent.
- Lack of labeled job-related fields required manual annotation.

📅 Phase 3: Model Training (Resume Field Classifier)

Timeframe: Week 4
Activities:

- Used `TfidfVectorizer` and trained a Random Forest Classifier to classify resumes into 24 fields.
- Saved model using Pickle for backend integration.

Challenges:

- Feature extraction (skills, education, experience) was noisy.
- Resume formats had lots of non-standard layouts (e.g., tables, images).
- Model overfitting on certain fields with more samples.

📅 Phase 4: Job-Resume Matching System

Timeframe: Week 5
Activities:

- Used Sentence-BERT for semantic similarity.
- Developed rule-based match scoring system:
    - Skill match %
    - Education match %
    - Experience match %
- Combined everything to calculate a final `resume_rank`.

Challenges:

- Designing a fair scoring formula.
- Matching extracted skills across synonyms/variants.

📅 Phase 5: Backend Development (FastAPI)

Timeframe: Week 6–7
Activities:

- Built upload and batch processing API endpoints.
- Integrated ML model into FastAPI.
- Used PyPDF2 and Flair NER to extract text and user name.
- Saved results to PostgreSQL using SQLAlchemy.

Challenges:

- PDF text extraction breaking on scanned resumes.

- Managing async file uploads with large PDFs.
- NER model performance sometimes inconsistent.

📅 Phase 6: Frontend Development (React + TypeScript)

Timeframe: Week 8
Activities:

- Built UI for resume + job upload.
- Displayed ranked results in frontend with progress bar and match breakdown.
- Connected to FastAPI using Axios.

Challenges:

- Handling large file uploads in React.
- Parsing API response structure into readable UI.
- Debugging CORS issues between frontend and backend.

📅 Phase 7: Dockerization & Deployment (AWS)

Timeframe: Week 9
Activities:

- Dockerized both FastAPI and React apps.
- Used Docker Compose for local orchestration.
- Deployed on AWS EC2 with Nginx reverse proxy.

Challenges:

- Managing Docker volumes and environment variables.
- Handling file permissions inside containers.
- Deployment pipeline errors (build context, CORS).

📅 Phase 8: Email Integration

Timeframe: Week 10
Activities:

- Extracted email using regex from resume text.
- Sent auto-email if resume rank ≥ 50%.
- Used threading to prevent API blocking during email sending.

Challenges:

- Extracting correct email from noisy text.
- Handling edge cases when emails not found.

📅 Phase 9: Batch Processing Module

Timeframe: Week 11
Activities:

- Allowed multiple resumes to be uploaded and matched against one job.
- Stored results in DB and returned ranked list.

Challenges:

- Processing multiple files in parallel.
- Filtering failed extractions gracefully.

📅 Phase 10: Final Testing & Optimization

Timeframe: Week 12
Activities:

- Performance testing for job-resume matching.
- Cleaned up unused code, optimized queries.
- Added error handling, logs, and user messages.

Challenges:

- Handling corrupted or image-based PDFs.
- Ensuring frontend stability with large response JSON.

🧩 Summary of Challenges You Faced

Category	Key Difficulties
🧠 ML	Lack of labeled data, noisy features
📄 NLP	Text extraction from varied PDF formats
🔄 Matching Logic	Designing fair ranking metrics
🔧 Backend	Async handling, email threading, CORS
🧱 Frontend	File handling, real-time ranking display
📦 Deployment	Docker networking, AWS service limits

✅ Final Features Achieved

- Resume classification using ML.

- Resume-job matching with semantic and rule-based logic.
- Upload + batch processing support.
- Ranked UI in React.
- Emails to shortlisted candidates.
- Fully deployed system on AWS using Docker.