**Documentation for ATS Application**

**Project Overview**

The ATS (Applicant Tracking System) application is a React-based frontend with a backend built using Flask and Python, responsible for ranking candidates based on various criteria like skills and experience. The application uses machine learning models (Random Forest) to classify and rank candidates for different job roles.

**File Structure Breakdown**

**Root Directory**

* .gitignore: Lists files and directories that should be ignored by Git.
* README.md: Project documentation (this file can contain instructions, setup steps, and an overview of the project).
* ats-backend-deployment.yaml & ats-frontend-deployment.yaml: Deployment configurations for the backend and frontend, likely used for setting up CI/CD pipelines in platforms like Azure.
* docker-compose.yml: Defines services for Docker Compose to manage both the backend and frontend in containers.

**Backend Directory**

* **candidate\_ranking.py**: Contains the core logic for loading, processing, and ranking candidates based on their skills, experience, and job role requirements. The file also includes a Flask app that provides a REST API for the frontend to interact with.

**Important Functions:**

* + load\_and\_prepare\_data(file\_path, is\_job\_roles): Loads CSV data and processes it, handling NaN values and standardizing column names.
  + engineer\_features(df): Adds feature engineering logic like handling experience and calculating a skill match score.
  + train\_models(df1, df2): Trains Random Forest models using the input data for classification (job role suitability) and regression (salary prediction).
  + predict\_and\_rank(class\_model, reg\_model, new\_data, required\_role, extracted\_skills, skills\_weight, top\_n): Predicts and ranks candidates based on their skills, experience, and role requirements.

**Flask Endpoints:**

* + /rank-candidates: POST request for ranking candidates based on data passed in the request (job role, number of candidates, etc.).
* ats-backend-service.yaml: Configuration file for the backend service (likely Kubernetes or similar orchestration).
* **Dockerfiles**:
  + Dockerfile.backend: Contains instructions for creating a Docker image for the backend service.
  + Dockerfile.python: Backend-related Dockerfile, likely used for setting up a Python environment and dependencies.
* **CSV Files**:
  + FinalResult.csv: Contains the final results of the candidate ranking process.
  + job\_skills.csv: Contains job skills that are used for model training.
  + skills\_and\_salaries\_categories\_p.csv: Contains additional information about skills and salary categories for different roles.
* **Other files**:
  + requirements.txt: Lists Python dependencies for the backend, such as Flask, scikit-learn, and pandas.
  + server.js and server1.js: Node.js servers for handling requests, likely linked to Docker and CI/CD pipeline integration.

**Frontend Directory**

* **src**: Contains all the source code for the React frontend.
  + App.js: Main entry point of the React app.
  + components/: Contains reusable components, such as forms, buttons, and data tables, for interacting with the backend.
  + pages/: Contains different pages of the application (e.g., Home, Candidate Ranking).
  + api/: Axios or fetch API calls to interact with the backend Flask API.
* **build**, **public**: Standard directories in a React project for static files and the built frontend.
* **Deployment YAMLs**:
  + ats-frontend-deployment.yaml: Used for deploying the React frontend (potentially to GitHub Pages or a cloud platform).
* **Docker-related files**:
  + docker-compose.yml: Configures the frontend service in Docker Compose.
  + Dockerfile: Builds the React app in a Docker container.

**Backend**

**Flask API Setup**

The backend is a Python Flask API responsible for:

* Loading CSV data (candidate data, job roles).
* Training models using Random Forest to classify candidates for job roles and predict salary.
* Exposing an endpoint /rank-candidates that takes job role information, required number of candidates, and CSV files as input, and returns a ranked list of candidates.

**How to Run the Backend**

1. **Set Up Python Environment**:
   * Install dependencies from requirements.txt:

pip install -r requirements.txt

1. **Run the Flask App**:

python candidate\_ranking.py

The Flask app will start on port 5000.

1. **API Endpoint**:
   * POST /rank-candidates:
     + Input: JSON body with fields like requiredRole, topN, and file paths to the CSV files.
     + Output: Ranked candidates with skill match score and predicted salary.
2. **Dockerization**:
   * Build the Docker image for the backend:

docker build -f Dockerfile.python -t ats-backend .

1. **Run the Docker Container**:
   * Using Docker Compose:

docker-compose up

**Frontend**

**React Application Overview**

The frontend is a React app where users can input job roles, specify the number of candidates to rank, and trigger the backend ranking process.

* **User Flow**:
  + User selects or inputs job roles.
  + A button triggers an API call to the backend to fetch ranked candidates.
  + Results are displayed on the UI in a table format.

**Running the Frontend Locally**

1. **Install Dependencies**:

npm install

1. **Start the React App**:

npm start

The app will run on http://localhost:3000.

1. **Dockerization**:
   * Build the Docker image for the frontend:

docker build -t ats-frontend .

1. **Run the Docker Container**:
   * Using Docker Compose:

docker-compose up

**API Integration**

The frontend sends requests to the Flask backend via the /rank-candidates API. The user specifies the job role and number of candidates, and the frontend uses this information to make an API call. Results are fetched and displayed on the frontend.

**Conclusion**

This documentation provides a complete overview of the ATS project. The backend uses Flask and Random Forest models to rank candidates based on skills and experience, while the React frontend interacts with the backend API to display results to the user. The application is fully Dockerized for seamless deployment.