**Project Finder: Product Requirements Document**

**1. Introduction**

The Project Finder application aims to streamline the process of matching employee skill sets with project requirements. It will achieve this by scraping project listings from various websites, extracting relevant details including skill requirements, embedding these skills into vector representations, and then comparing them against employee skills to identify the best-fit projects. The application will provide a user-friendly interface to manage projects, employees, and view matching results.

**2. Goals and Objectives**

* **Primary Goal:** To efficiently match employees with suitable projects based on skill compatibility.
* **Objective 1:** Automate the collection of project data from diverse web sources.
* **Objective 2:** Enable robust and semantic comparison of skills between projects and employees.
* **Objective 3:** Provide an intuitive user interface for managing project data, employee profiles, and viewing match results.
* **Objective 4:** Ensure data persistence and integrity through a reliable database system.
* **Objective 5:** Facilitate debugging and monitoring through comprehensive logging.

**3. User Roles**

While explicit personas are not defined, the application caters to users who need to:

* **Explore Projects:** View available projects and their details.
* **Manage Employees:** Add, edit, and view employee skill sets.
* **Find Matches:** Identify the most suitable projects for a given employee.

**4. Functional Requirements**

**4.1. Frontend (User Interface)**

The frontend will be a React application built with Vite and TypeScript, designed with a serious yet pleasing aesthetic and full responsiveness across devices.

**4.1.1. Overall Layout**

* Primary screen: the application is primarily to be displayed on a 24” 1920x1080 pixels screen, although multi-screen capability is “nice to have”.
* **Header:** "Project Finder"
* **Main Content Area:** Takes up all space on the screen bar the space taken up by the header. Divided horizontally into three equal-sized columns with padding for separation.

**4.1.2. Left Column: Project Explorer**

* **Header:** "Project Explorer"
* **Setting Buttons (Time Range):**
  + "One day" (communicates 2 to backend)
  + "One week" (communicates 8 to backend)
  + "One Month" (communicates 32 to backend)
  + Only one button can be active at a time.
* **Action Buttons:**
  + "Scan": Initiates web scraping based on the selected time range.
  + "Show": Displays projects filtered by the selected time range.
  + "Empty Database": Clears project data from the database.
* **Toggles:**
  + **"Live/Test Mode":**
    - If "Live": The current database (without test projects) is loaded on refresh.
    - If "Test": Test data (four plausible test projects and four plausible test employees) is loaded on refresh. Test projects are marked by "tenderer" field having value "Test".
* **Project List:** Displays projects’ release dates and titles as clickable entries. Clicking a project shows its details in the Center Column.

**4.1.3. Center Column: Project Contents**

* **Header:** "Project Contents"
* **Subheader (Initial):** "Select a project to view its details."
* **Button** (only visible once a project has been selected): Edit Project requirements
* **Project Details Display:** Shows comprehensive details of a selected project, including: title, summary description, release date, start date, location, tenderer, internal project ID, skill requirements (as a series of small boxes), going rate, and URL to the detail page.

**4.1.4. Right Column: Employee Fit / Employee Matching**

* **Header:** "Employee fit"
* **Employee Dropdown:**
  + Default text: "Select an employee."
  + Populated with a list of employees in the system.
* **Action Buttons:**
  + "Add New Employee": Opens a pop-up to enter employee name and a comma-separated list of skills. Selecting an employee triggers the matching algorithm for the selected employee.
  + "Edit Skills": Allows editing the skills (add, subtract, split) of the employee selected in the dropdown menu.
  + "Refresh Matches": Triggers the matching algorithm for the selected employee.
* **Matching Results Display:**
  + A ranked list of projects matching the selected employee's skills (best matches on top), including the percentage of skill fulfillment.
  + A list of skills most often missing by the employee across the top five matching projects.
* **Employee Management UI:** Supports form validation, user feedback, and confirmation dialogs for Add/Edit/Delete employee operations.

**4.2. Backend (API & Logic)**

The backend will be built with Python 3.11, using FastAPI, uvicorn, and Pydantic.

**4.2.1. Web Scraping Pipeline**

* **Configuration:** Reads guiding website URLs and CSS classes from config.json for scraping. The config.json file is already given and placed in the backend folder. The application must read and process this given config.json file
* **Three-Level Scan Process:**
  1. **Level 1 (Project List):** Extracts main project list from site\_url using project-list-class. Logs successful entry.
  2. **Level 2 (Project Card):** Loops over project “cards” (project-entry-class) in the project list (which are located on the main page given by site\_url), extracts high-level data from project cards using the CSS classes specified in config.json. Logs successful scan results of each card.
  3. **Level 3 (Project Detail):**
     + Uses individual project URL to extract data. Logs any successful result.

All level 2 and 3 scans for all projects are to proceed in parallel. Once, for a certain project, the level 2 and 3 scans have terminated, the data harvest of both is to be consolidated as follows:

* **Data Consolidation:**
  1. If only one of Level 2/3 finds a data item, use that.
  2. If both find and are identical, use that.
  3. If both find and differ:
     + For the requirements list: consolidate both
     + For text data: concatenate the two strings, separated by a semi-colon
     + For dates: use Level 3 data.
* **Technology:** Selenium (headless) with robust WebDriver management, retries, and exponential backoff for anti-bot measures. Supports dynamic sites.
* **Data Persistence:** Writes project data to DB immediately after each URL/Project completion.
* **Pagination:** After completely scanning the initial page found on level1, the page list as a rule continues on following pages. Therefore, after scanning the contents of the first page on levels 1, 2 and 3, this task is to be repeated for the next page of the project list. The link for this next page can be found through the "next-page-selector" in the config.json file. This process it to continue until the time-range for the scan (release-date) has been exhausted.

**4.2.2. Data Extraction (Mistral API)**

* The code is already present in the “mistral\_handler.py” file and is not to be changed.
* **Model:** mistral-large-latest
* **Purpose:** Analyzes detailed project pages to extract structured project information.
* **Extracted Fields (JSON format):** title, description, release\_date (DD.MM.YYYY), start\_date (DD.MM.YYYY), location, tenderer, project\_id, requirements (array of strings), rate.
* **Error Handling:** Gracefully handles missing data (e.g., "n/a"), and additional commentary from Mistral (ignores non-JSON text). Preserves German characters.
* Check any errors occurring against the formal Mistral API documentation at the following internet sites: <https://docs.mistral.ai/api/>; <https://docs.mistral.ai/getting-started/clients/> ; <https://github.com/mistralai/client-python/tree/main/docs/sdks/chat>;

**4.2.3. Skill Embedding (OpenAI API)**

* **Model:** text-embedding-3-large
* **Purpose:** Converts individual project requirements and employee skills into vector representations for semantic comparison.

**4.2.4. Project Matching**

* **Algorithm:** Compares project requirement vectors with employee skill vectors using:
  + Cosine similarity or Euclidean distance as set in the config.json file
  + Threshold-based filtering
  + Weighted skill scoring
  + Multiple skill matching
* **Output:** Calculates an average match score for all required skills to rank projects.

**4.2.5. Database Management**

* **ORM:** SQLAlchemy for database abstraction.
* **Database:** SQLite (default, configurable).
* **Tables:**
  + projects: Stores scraped project details.
  + skills-vdb: Stores skill strings and their vector embeddings.
  + employees: Stores employee names and their skill lists.
  + AppState: Manages application state, including "Live/Test Mode" and "Overwrite" toggle settings.
* **Operations:** CRUD operations for projects, skills, employees.
* **Schema Evolution:** Requires deleting and recreating the database via init\_db.py when schema changes. The database is to be fully normalized. Projects' requirement vectors that are needed for project/employee matching must be obtained through the "requirements" column of the "projects" table in combination with the "embedding" column of the "skills" table. Thereby the "skills" table is functioning as a "lookup" table. The "requirements" table contains, in a sense, lists of foreign keys to the skills table.
* **Maintain the Database in a fully normalized format.** In particular, do not keep any data on embeddings anywhere outside the “skills-vdb” table.

**4.2.6. API Endpoints**

* All backend endpoints will be documented with Swagger/OpenAPI.
* **Example:** POST /scan/{time\_range} (Request body: { "overwrite": boolean }, Response: JSON with scan message and project details).
* API URL definitions and calls will consistently use /api/ as part of the BaseURLs.

**4.3. Configuration Management**

* **Files:** config.json (for scraping and logging settings) and .env (for API keys).
* **Flexibility:** Supports multiple config file locations and environment variable overrides for API, server, and database settings.

**5. Non-Functional Requirements**

**5.1. Performance**

* Parallel processing for web scraping (Level 2/3 scans).
* Efficient database queries and matching algorithms

**5.2. Security**

* API keys stored securely in .env and not tracked by Git.
* CORS configured for frontend-backend communication, set relatively permissively.
* Robust API key validation (handling quotation marks) and status checking.

**5.3. Usability**

* Responsive design for optimal viewing on all devices.
* Clear and intuitive UI with user feedback (loading indicators, error/success messages, confirmation dialogs).
* Multi-language support for date handling.

**5.4. Maintainability and Code Quality**

* **Platform:** Developed on Windows 11 using PowerShell.
* **Development Flow:** Strict adherence to plan, docstrings, testing, implementation, linting, and re-testing.
* **Code Style:** Python code adheres strictly to PEP 8 guidelines (79 char line length, 2 blank lines for top-level, 1 for methods, 4-space indent, no trailing whitespace, newline at EOF).
* **Linting:** flake8 checks (E501, E302, E305, W292, F401, E128).
* **Imports:** Grouped, alphabetically sorted, explicit (no wildcards except for backend/models/\_\_init\_\_.py), no unused imports.
* **Docstrings:** All functions must have complete docstrings explaining purpose, methodology, and interfaces.
* **Error Handling:** Uses specific exception types, includes proper logging, returns appropriate HTTP status codes, and provides descriptive error messages. Input data validated using Pydantic models.
* **Logging:** Extensive logging in both backend and frontend for debugging and flow tracing, configured via config.json.
* **Database Schema:** Changes reflected in core\_models.py and schemas.py, with type hints and documentation.
* **Testing:** Comprehensive regression tests (test\_\*.py in tests/ directory), pytest fixtures, Arrange-Act-Assert pattern, edge case consideration.

**5.5. Reliability & Error Handling**

* Graceful handling and reporting of errors to the user (e.g., no projects found, API failures, invalid responses, empty DB).
* Robustness against unexpected API responses (e.g., Mistral returning extra commentary).

**6. Scope**

**6.1. In-Scope**

* Web scraping, data extraction, and storage of project information.
* Semantic skill embedding for projects and employees.
* Project matching based on skill similarity.
* Frontend UI for project exploration, employee management, and match viewing.
* Basic database inspection view.
* Test mode functionality for development.
* Comprehensive logging and error handling.

**6.2. Out-of-Scope (for initial release)**

* User authentication/authorization beyond the provided system.
* Complex reporting features beyond basic lists.
* Advanced user customization options for UI.
* Deployment to production environments (focus on local development).

**7. Future Considerations (Potential Enhancements)**

* Adding more sophisticated search/filter options for projects.
* Implementing user accounts and profiles.
* Allowing users to define custom scraping rules.
* Integrating with other external data sources (e.g., job boards APIs).
* Visualizations for skill gaps or project trends.

**What We Accomplished:**

1. **Real-time streaming scan** - Projects now appear in the frontend as they're processed, not all at once at the end
2. **Better UX** - New projects appear at the top of the list so users can see progress immediately
3. **Graceful error handling** - No more spam alert boxes when the backend is down
4. **Clean completion** - Scans end elegantly without "Connection lost" errors
5. **Sequential processing** - Replaced parallel processing with sequential to enable real-time updates

**Key Technical Improvements:**

* **Streaming web scraper** with scan\_website\_stream() method
* **SSE completion tracking** to distinguish normal completion from errors
* **Conditional alert logic** to avoid showing alerts for connection issues
* **Top-of-list project addition** for better visibility

The system now provides a much better user experience with real-time feedback and graceful error handling. Users can see projects appearing as they're processed, and the interface remains responsive even when the backend is unavailable.Great collaboration on getting this working smoothly!