🎨 The Skills Matrix Frontend: Technical Challenge

Disclaimer

> **Note:** This is a purely hypothetical exercise designed to assess technical skills and reasoning. It is not a "real-world" task, and no code submitted will be used in our production environment. We value your time and aim for this to take approximately **2 hours** of active coding.

Rules

- Keep notes and thoughts in SOLUTION.md as you go along - especially **why** you did something and **what** you would do if you had more time. Task 3 is really about cleaning out and expanding on this to demonstrate your long-term thinking.
- Please use AI/Searches/a little help from a friend - however you need to be able to articulate how and why you made certain decisions and will be quizzed on this during a full interview.

Scoring

Below are the scoring criteria and a short definition:

- **Core Execution & UX** Does it solve the prompt? Is the AntD implementation polished and functional?
- **Technical Excellence** Clean code, SOLID principles, proper React hooks usage, and performance (debouncing).
- **Documentation & Reasoning** Clarity of the SOLUTION.md, transparency about AI usage, and ability to explain "The Why."
- **Scalability & Architecture** How they structured components, handled state, and prepared the code for future growth.
- **Product & Critical Thinking** Use of intuition for the "Skill Gap" and "Expertise Levels." Did they build what the user needs or just what was asked?

Time Allocation Guide

We recommend spending roughly:

- **30 minutes:** Setup and understanding the React + Ant Design components.
- **45 minutes:** Task 1 - Fix the bugs (including the "Stale State" and linting issues).
- **30 minutes:** Task 2 - Implement the "Role Readiness" dashboard.
- **15 minutes:** Task 3 - Document your reasoning in `SOLUTION.md`.

Total: ~2 hours

The Scenario

Our startup is building a dashboard to help managers visualise the "Skills Matrix" of their teams using the ESCO skills framework. We have a React/Vite/Ant Design prototype built, but it’s currently a bit clunky. The UI is out of sync with the data, and it lacks the high-level insights our users are asking for.

Task 1: Fix Something (The 'Sync' Bug)

The HR team has reported that the skill search and employee list are behaving strangely.

- **Your Objective:** Identify and fix the faults in `src/components/SkillSearch.jsx` and `src/components/EmployeeList.jsx`.
- **The Constraints:** Fix the "Stale State" bug: Searching for a skill often shows the results of the *previous* search due to an asynchronous update issue.
- Ensure the search is performant (e.g. handle rapid typing gracefully via debouncing or similar).

Task 2: Make Something (The "Role Readiness" Dashboard)

Our product manager has received feedback from a customer: *"I want to see at a glance how an employee matches up against a 'Target Role' so I can plan their training."*

- **Your Objective:** Create a new "Readiness View" component that consumes the `/api/readiness` endpoint.
- **The Brief:** We are leaving the UI structure up to you, but you must use **Ant Design 4**. The customer has hinted they want to see:

1. **A Visual Match:** A clear graphical representation (e.g. `Progress`, `Statistic`, or `Gauge`) of their suitability.
2. **The "Skill Gap":** A list of missing skills categorised by importance or type.
3. **Product Intuition:** How would you display "Expertise Levels" (1-5) if the data provided them? Feel free to mock this UI-side if you want to demonstrate the concept.

- **The Challenge:** We are looking for clean component architecture and an intuitive user experience. Use the AntD library effectively to create a "professional" feel.

Task 3: Document Your Reasoning

In the provided `SOLUTION.md`, please provide a brief commentary on:

1. **AI Usage:** We encourage the use of AI tools (Copilot, ChatGPT, etc.). Which parts did it help with? Did you have to correct any of its suggestions?
2. **UX Decisions:** Why did you choose those specific Ant Design components for the "Readiness" view?
3. **The "If I Had More Time" List:** What would you add or refactor if you had more time?
4. **Framework Rules:** Did you encounter any limitations with Ant Design 4? How would you handle a custom-branded theme in the future?

Technical Setup

Prerequisites

- Docker Desktop installed and running
- A browser to view the application

Getting Started

The repository is dockerised with Vite + React (JS). The setup script will:

- ✅ Install all npm dependencies
- ✅ Run ESLint and Prettier checks
- ✅ Start a Mock API server (JSON Server)
- ✅ Start the Vite development server

1. Build and Start the Containers

```
# Build the Docker images and start services
docker-compose up --build

# This will automatically:
# - Install dependencies with npm ci (secure, reproducible builds)
# - Run linting checks (ESLint and Prettier)
# - Start the mock backend on port 8001
# - Start the React app on port 3000
```

2. Access the Application

- **Frontend URL:** `http://localhost:3000`
- **Mock API Data:** `http://localhost:8001/employees` (See `db.json` for data structure)

Code Quality Checks

The project includes **ESLint** and **Prettier** for code quality and formatting. The container runs both checks on startup.

Note: The codebase intentionally contains some linting errors (unused variables, console logs) and formatting issues that you should identify and fix as part of Task 1.

To run checks manually:

```
# Run ESLint
docker-compose exec frontend npm run lint

# Check Prettier formatting
docker-compose exec frontend npm run format:check

# Fix Prettier formatting automatically
docker-compose exec frontend npm run format
```

Project Structure

```
frontend-test/
├── src/
```

```
│   ├── components/
│   │   ├── SkillSearch.jsx   # ⚠ Contains state bugs (Task 1)
│   │   ├── EmployeeList.jsx  # ⚠ Needs performance optimisation
│   │   └── Readiness.jsx     # 📝 TODO: Implement this (Task 2)
│   ├── App.jsx               # Main layout and routing
│   ├── main.jsx
│   └── config.js             # Configuration file
├── mock-api/
│   └── db.json               # Mock data for skills and employees
├── docker-compose.yml
├── Dockerfile
├── .eslintrc.cjs             # ESLint configuration
├── .prettierrc.json          # Prettier configuration
├── .prettierignore           # Prettier ignore rules
├── package.json
├── README.md                 # This file
└── SOLUTION.md               # Template for your solution
```
