Zoe Financial Front-End Developer Test

In order to demonstrate that you are capable of writing functional and thoughtful code for a web-based solution, we would like you to complete the following exercise. The logic of the whole project should take less than an hour for a reasonably experienced front-end developer. However, we're not interested in how long this exercise takes you, rather we're interested in how your work on it demonstrates how you think and collaborate.

Successful implementation of the following product specification is obviously an important measure of your ability to properly work with our team. However, it's not the only important aspect. We want you to think critically about the specification below, and ask questions about how you should implement it. We want you to consider if there's any missing information in what's provided below, and ask about it before you begin your implementation.

Successful implementation also includes a number of bits beyond just a working prototype. Your complete work is a reflection of how you might work day-to-day with us, and we'd like to see how you do when you can't get as much time as you'd prefer to complete a task.

1. Goal

The goal of this exercise is to implement a web application that displays the match between 2 actors: the Visitor and a list of Agents. You should be able to interact with the Agents list based on their average income amounts. This intelligent match will use just 1 rule: "Income".

2. Our expectations

- The code you submit must work properly.
- You should be able to explain all parts of your app and walk us through the decisions you made.
- Also we could ask about the libraries used and why/how they were used.

3. Specifications

Base tech stack: **React**, **Typescript**, **CSS,** and **Jest**. Feel free to use any development environment tool and CSS pre-processor. If you require to use other third-party libraries you must justify the reason.

You have to pull the agents list dynamically via AJAX from the provided JSON file. The rule to filter agents' data is based on their income parameters. The use of React hooks is required (*useState*, *useEffect, etc*).

The Agents:

- Agent ID, Name, avatar, Income.
- List provided with the test (AGENTS_LIST.json).

4. Main logic

- You should show a main view with 1 input for the Visitors to type income value.
- 5 digits is the required length (input validation).
- A button labeled “MATCH” to trigger the algorithm.
- You must filter the pulled Agents data and find the ones with the closest income value, based on Visitor’s input.
- The algorithm must filter pulled Agents data by an average income that matches the threshold (Visitor typed income ± 10000).
- At the end of the match process you will show a second view, an interface with the results (Agent ID, name, avatar and income).

5. Final details

- List a maximum of 3 Agents, adding an option to “see more”. Each option click will show the next 3 advisors. Also add an option to “see less”.
- Include controls to order advisors (ascendent and descendent) name & income.
- Clicking on an Agent will hide their card from the UI
    - Store hidden Agents in memory
- Avatar images should match each Agent’s gender.
- In case of no matched Agents, an error message has to be shown to visitors: “No available Agents based on your income. Please try a different income value.”

6. Test coverage

Quality, best practices and continuous scalability are key pieces we are looking for in every piece of code developed by our team members. Unit testing is a main piece of our workflow. Write a short test that covers the “see more” & “see less” pagination logic.

7. Your final touch

Based on the information you already know about the experience and the pieces previously required, what do you think is missing in the experience? How can the interaction or the app be improved? Add a feature, UI element or interaction that fixes or improves how the app is structured or the user interacts.