Crisp - Full Stack Take Home Test

Introduction

In this project, you will implement a financial accounting application that allows users to track, maintain, and review financial accounts and their entries. You will build your application in two parts: a Spring Boot API application and a single-page application (SPA) using a JavaScript SPA framework.

For the backend, you may use any JVM language for your application, but we recommend using Java or Kotlin. You will be expected to connect your Spring Boot application to a PostgreSQL or MySQL database, and you will need to provide instructions on how to configure the database connection properties. You may use any library you want for database access from Spring Boot. We recommend using Spring Initializr to create your project.

For the frontend, you may use React, Angular, Vue.js, or Svelte for your application. We recommend using React if you have used it professionally in the past. Using a component library and other standard libraries is also acceptable. The application may be run directly — e.g. using `yarn dev`. You are also free to use a bundler of your choice — e.g. Webpack, Rollup, Bun, etc. — so long as you provide instructions on how to build and run the application.

This problem documentation is not intended to be sufficient to deliver a "production grade product". It is deliberately nonspecific. In a real development situation, you would have the opportunity to ask clarifying questions about requirements. In this case, you will not be able to ask such questions. Instead, when you find a gap in the requirements, make an assumption, document it, and deliver documented assumptions as part of the solution. These will serve as examples of where you would have asked for clarification.

Product MVP Requirements

- Users can create, update, and delete financial accounts
    - A financial account has a name, an account ID (numerical with dashes), and a type
    - Types include e.g. "Accounts Payable", "Fixed Assets", etc.
- Users can create, update, and delete entries for a financial account
    - An entry has a value, a name, and a date
    - Value can be positive or negative USD
- Users can view the list of entries for a financial account

User Interview Highlights

- From Sam, AP/AR specialist: "I need to be able to categorize entries to track different types of expenses. It's important for me to see stuff like spending on software services versus construction contractors."
- From Brenda, FP&A: "My job requires me to slice up the history of our overall monthly spend for the last year. This lets me forecast our future expenses per-account for things like building maintenance and wages."
- From Aaron, CFO: "I give a presentation to the board every quarter where I show them how much our overall categorized spend for the last quarter compares to what we planned on spending."
- From Christine, accountant: "We go through an audit every year, and auditors like to pick the most random entries out of thousands to ask for supporting documentation. They'll usually give me an entry number, account number, and a date, but not always all three."
- From Tony, IT manager: "Our machines are pretty cheap. We don't have the budget for new machines. Anyway, we get a lot of tickets that complain about slow computers, and it's usually because they've burnt through most of their 4 GB of RAM. Normally, we make them open Chrome and close their tabs."

Database Seeds

As part of your solution, you must use a PostgreSQL or MySQL database. Below, you will find a database seed that you are required to integrate into your solution. You do not need to integrate this seed *directly* into your solution, but you must at least use the structures and data as a starting point for your database and provide instructions on how to set up the database.

Database Seed

Out of Scope

These are deliberately left out of scope and do not need to be included in your solution:

- Security solutions — login, user-scoped data, etc.
- API versioning support
- Deployment/environment considerations beyond running on a developer laptop
- Scalability, reliability, most other -ilities

Deliverables

- Running code and a passing test suite
- Instructions on how to build and run the code with a fully seeded database
- Documentation of your code for anything that isn't self-documenting or clear to a fellow developer
- Documentation of any assumptions that you made while interpreting the requirements
- Documentation of the architecture and design of your solution
- Explanation of how your solution could be extended, optimized, or updated to account for other non-functionals that were not required, e.g. security, API versioning, scalability, reliability, etc.

What Are We Looking For?

Try balancing some common sense foresight with the simplest thing that could work. A large chunk of this assignment is very straightforward, but there are also some aspects of API design, UX design, and implementation details that could be solved in a wide range of ways. Pick a reasonable approach and be prepared to speak to alternatives in the review. This aside, these are the other areas we're scoring during the review:

- Application runs and works
- Consistent UI and UX design
- Well-structured, readable, and maintainable code
- Unit tests and (optionally) snapshot tests
- Fulfillment of MVP requirements
- Fulfillment of at least some requirements related to the user interviews
- Quality of the documentation

Submission Format

Please zip (or tar) up the code and documentation and share it with the recruiter via Dropbox, Google Drive or any other file sharing service you prefer.

How much time should I spend?

Short answer:
Spend six hours and turn it in. If your mindset won't let you do that, spend an extra six hours, and then turn it in.

Longer answer:
Here at Crisp, we're trying to do what makes sense, instead of what's always been done before. The traditional interview process for a developer usually includes a series of phone screens, an on-line coding test, and culminates with an all-day, on-site interview. The process is costly, intrusive, and fails to replicate what developers actually do on a day-to-day basis.

We're trying to streamline the process, and get at what actually matters: how likely are you to be successful, and thrive, if we welcome you onto our team? Towards that end we've replaced the on-line coding tests and on-site interviews with a realistic project that represents something you might actually build at work.

As with any project, you will be presented with the traditional tradeoff between functionality, quality, and cost. We're not looking for perfection, or coverage of every imaginable use case. And we're not looking for you to invest more time in our interview process than you would in a more traditional, on-site interview. Senior developers, using their preferred tools, should be able to deliver a quality implementation of our take home project, with significant levels of functionality, in about a day.

If that seems insurmountable to you, it does not automatically mean that you are not qualified to work with us! Please feel free to narrow the implementation scope, or spend a bit more time iterating on your solution — whichever you are more comfortable with. We will ask you to set a deadline that *you* can meet while working at your own pace. It's okay to set that deadline several weeks out if that's what you will need due to work, life, or family circumstances.

Please do *not* set an extended deadline, intending to invest extraordinary time in this project. We've had candidates be successful having spent as little as four hours, and we've had candidates spend over thirty hours and not advance to the final part of the interview process. We want your time commitment to reflect the investment you would put into any company's interview process, and not be an undue burden.

Good luck, and have fun!