Canoe Tech Assessment for Remotely

Expected time to complete: 1-2 hours

Overview

Canoe's business is centered around various applications that automate workflows for the administration of investment funds. To do this efficiently, we need a master database that can be shared by these applications as a single source of truth for these investment funds. This dataset is constantly changing and requires some amount of manual curation by a data quality team using a web application that is built for the purpose of maintaining this data.

This exercise requires building a data model and back-end service to support a couple of the basic uses cases for this web application. Fund Records can be created / read / updated / deleted manually in this application. There also will be back-end processes that are creating new records automatically. Some of the automatically created records will be duplicates that will need to be manually reconciled

Background

There are several entities that make up the underlying dataset for this application:

Funds

- Name: name of the fund
- Start year: the year the fund was started
- Manager: The company that manages the fund, every fund has 1 manager
- Aliases: An alias is an alternative name for a fund, and every fund can have multiple aliases

Fund Managers

All funds are created and managed by an investment management company

- Name of the company that manages the fund
- Fund Managers can manage multiple funds

Companies

A fund is typically invested in multiple companies.

- Name
- Multiple funds can be invested in the same company

Tasks

1. Design and create a data model to store data for the entities described above. Please document your ER diagram.
2. Create a back-end service to support the following use cases:
    a. Display a list of funds optionally filtered by Name, Fund Manager, Year
    b. An Update method to update a Fund and all related attributes.
3. Create an event-driven back end process to support:
    a. If a new fund is created with a name and manager that matches the name or an alias of an existing fund with the same manager, throw a duplicate_fund_warning event.

b. Write a process to Consume the duplicate_fund_warning event

c. **Bonus if time permitting**: Add a method to the service created in #2 that will return a list of potentially duplicate funds

What We Are Looking For

- Clear instructions for running and testing your solution
- Cleanliness of code and tradeoffs you make, including considerations for these standard principles:
  - DRY/KISS/YAGNI/SOLID/etc…
- Separation of concerns
- Readability of code
- Testability of code
- Approach to error handling
- Data model/data structures you choose to use
- Scalability considerations:
  - How will your application work as the data set grows increasingly larger?
  - How will your application work as the # of concurrent users grows increasingly larger?