FE Coding Task

September 2024

Instructions

Your task is to create a frontend application using React and TypeScript that presents a user's financial portfolio in a visually engaging way. Application contains two parts - public login page (simple local storage credentials is enough) and secure part with dashboard containing components below.

The main features you need to implement are:

1. **Portfolio Balance Donut Chart**: The chart should show the user's balance broken down by asset class or by specific asset. The user should be able to switch between these two views.

2. **Positions Table**: same data as previous chart but in the table format

3. **Historical Chart**: This chart should display the performance or total value of the portfolio over time.

You will be provided with the following RESTful API endpoints:

1. **GET /assets**: Fetches a list of all financial instruments the user may own, including cryptocurrency, stocks, and cash.
2. **GET /prices?assets=GBP,BTC,APPL&asOf=2023-01-01**: Fetches the latest price for one or multiple assets. You can provide the asOf parameter to get the price at a specific date, or omit it to get the current price. You can also provide a range by using the from and to parameters. All prices provided in USD.
3. **GET /portfolios?asOf=2023-01-01**: Fetches a list of user positions. You can provide the asOf parameter to get the portfolio at a specific date, or omit it to get the current portfolio.

You can create your own simple endpoint / API contact with mock data according to the descriptions provided – see the last pages as well! The provided contract provides an overall idea how it might be structured and might contain some missing parts. The application should update the charts whenever the user changes the asset or asset class in the Donut Chart, or when the time period for the Historical Chart changes.

Requirements

- Use React for the UI and TypeScript for static typing.
- Use a popular charting library such as Chart.js, D3.js, or Recharts to visualise the data.
- Tailwind theme to support white labelling
- Application should have error handling for API calls.
- The application should be responsive and have a user-friendly interface.
- Write clean, modular, and reusable code.

Deliverables

A GitHub repository containing your application, including all code, tests, and a README file containing instructions on how to install and run your application.
Evaluation Criteria:

Your task will be evaluated based on:

- **Functionality**: Does the application perform the tasks as required?
- **Code Quality**: Is the code clean, organised, and professional?
- **UI/UX**: Is the application easy to use, aesthetically pleasing, and responsive?
- **Error Handling**: Does the application handle potential errors gracefully?
- **Documentation**: Are the installation/run instructions in the README clear and concise?

Good luck! Let us know if you have any questions.

```yaml
openapi: 3.0.3
info:
  title: Vega API
  description: Vega FrontEnd coding task
  version: 1.0.0
paths:
  /assets:
    get:
      summary: get all assets
      description: fetch information about all available assets
      operationId: getAssets
      responses:
        '200':
          description: successful operation
          content:
            application/json:
              schema:
                type: array
                items:
                  $ref: '#/components/schemas/Asset'
  /prices:
    get:
      summary: get asset prices
      description: fetch list of asset prices
      operationId: getPrices
      parameters:
        - name: asset
          in: query
          description: Asset filter
          required: false
          explode: true
          schema:
            type: string
        - name: asOf
          in: query
          description: timestamp of the price
          required: false
          explode: true
          schema:
            type: string
            format: date
      responses:
        '200':
          description: successful operation
          content:
```

```
          application/json:
            schema:
              type: array
              items:
                $ref: '#/components/schemas/Price'
  /portfolios:
    get:
      summary: Get list of all investor positions
      description: fetch list of positions
      operationId: getPortfolio
      responses:
        '200':
          description: Successful operation
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Portfolio'
components:
  schemas:
    Portfolio:
      type: object
      properties:
        id:
          type: string
          format: uuid
        asOf:
          type: string
          format: date-time
        positions:
          type: array
          items:
            $ref: "#/components/schemas/Position"
    Position:
      type: object
      properties:
        id:
          type: integer
          format: int64
          example: 10
        asset:
          type: string
          format: uuid
        quantity:
          type: integer
          format: int32
```

```
      example: 7
    asOf:
      type: string
      format: date-time
    price:
      type: integer
      format: int32
      example: 7
Price:
  type: object
  properties:
    id:
      type: string
      format: uuid
    asset:
      type: string
      example: APPL
    price:
      type: integer
      format: int32
      example: 290.32
Asset:
  type: object
  properties:
    id:
      type: string
      format: uuid
    name:
      type: string
      example: APPL
    type:
      type: string
      example: stock|crypto|fiat
```