Software Design Document

1. Introduction

1.1 Project Title

Portfolio Analysis Tool (PAT)

1.2 Problem Statement

Retail investors lack free, accessible tools to preform deep, quantative analysis of their investments.

1.3 Solution Summary

A web application that allows user to input their holdings. The application will orchestrate the back-end to fetch real-time market data, perform calculations and display portfolio metric - risk, sector allocations, implied growth, sentiment analysis. The user is then given the tools to compare their current portfolios to potential ones. Users are able to securely (we'll try) create profiles, save, update their portfolio holdings.

1.4 Goals and Objectives

- **Primary Goal:** Deliver a Minimal Viable Product (MVP) that provides analytics on a manually entered portfolio
- **Technical Goal:** Build a clean, scalable, and well-documented microservice architecture using React, Python and PostgreSQL

2. Requirements

2.1 Functional Requirements

Describes the features available

1. **User Authentication**
    - User must be able to create an account with an email and password
    - Users must be able to log in / out of an account
    - User sessions are secured by JWT tokens*.
    - Allow guest users for demo purposes (no registration required), their portfolios are not saved
2. **Portfolio Management (CRUD)**
    - Logged-in user is able to create, read, update and delete a portfolio
    - Inside the portfolio, a user can create, read, update and delete holdings (Ticker, Quantatity, Date of Purchase, Date of Sale)
    - Users can copy, merge, XOR, intersect portfolios together.*
3. **Data & Calculations**
    - The system displays the latest end of day price of each holding
    - The system displays the total portfolio value
    - The system displays the percentage weighting of each holding
    - The system displays the sector allocation of the portfolio as a pie chart

        - User is able to decide whether to look through an ETF (let an ETF be its own type of allocation, or look through into the holdings of the ETF)
    - The system displays the portfolios historic Beta, historic Alpha, historic Volatility, historic Value-at-Risk.
4. **Analysis & Simulation**
    - The system provides a "What If" mode, where the user can create a copy of their portfolio
    - In What If mode user can CRUD their holdings, a side-by-side comparison is displayed of key metrics (market value, sector allocation, etc.) between the simulated portfolio and the existing one.
5. **Benchmarking**
    - User is able to pick from a list of passive strategies (ETFs) and benchmark their portfolios returns to it.
    - User can compare the metrics described in Data & Calculation of both their existing portfolio and the benchmarks
6. **RDCF**
    - User is shown the past cash flows of each holding
    - User is shown the "priced-in" cash flows and interest rates as predicted by the Reverse DCF model, more information is in the model document.
    - An aggregate measure of the priced-in cash-flows across the users portfolio is given.

2.2 Non-Functional Requirements

Describes how the system must perform

1. **Perfomance**
    - The dashboard page shall load with-in 3 seconds.
    - API responses shall be less than 0.5 seconds.
2. **Security**
    - All passwords are hashed *before* storage in database.
    - All API communication happens over HTTPS.
    - A user can only access their own portfolios and data.
3. **Usability**
    - The UI is intuitive and easy to navigate. No training should be required for a user familiar with financial topics.
    - Data is presented in primarily visual formats (Graphs, charts, figures).
4. **Availability**
    - The application has an above average uptime.

3. System Overview

3.1 High-Level Overview

This a description of how the user will interact with the application.

1. **User Registration & Authentication**: A new user signs-up and verifies through email. An existing user logs in securely.
2. **Portfolio Managment**: A logged-in user will be shown a dashboard. They can either click on an existing portfolio or create a new one.

3. **Data Input**: User edits their portfolio by inputing the ticker, amount, and time of trade.
4. **Communication with Backend**: Upon saving or editing the portfolio, the frontend sends an API request to the backend.
5. **Data Proccessing**: The backend fetches market data and preforms the necessary calculations
6. **Display**: The processed data is sent to the front-end and displayed on the users dashboard
7. **Advanced Features**: More advanced features are available. Users can benchmark their performance, use the What-If simulation tool and access models such as RDCF.

4. Architecture & Components

The project follows a client-server microservice architecture, for the seperation of concern and scalability.
Front-End Client: React.js web app
Back-End API: Python Django*
Data-Services: Edgar (SEC financial filings), Yfinance (real time data), historic data ???
Database: PostreSQL
Cache : ??

5. Implementation Details

6. Data Flow

6.1 Sequence : User Loads Portfolio

A detailed description of how the front end and backend interact with each other

6.2 Sequence : User Runs a Reverse DCF

7. Conclusion