Portfolio Data Aggregator Test Suite

Software Engineers:

* Cody Strange – Software Engineer

# Abstract

# Organization

## Coding Standards

### Commenting

##### Functions

* Brief description of function
* Parameters
* Return Values

*Classes*

* Brief description of class

*ChatGPT*

* I highly recommend letting ChatGPT do most of the commenting
* Double check any comments by ChatGPT

### Naming Conventions

* Classes: CapWords
* Functions: snake\_case
* Variables: snake\_case
* Constants: ALLCAPS
* Files/Folders: snake\_case

### Type Safety

* Every function should have the parameters types listed and the return type of the function listed
* 

## Software Requirements

Programming Languages and Frameworks

* Programming Languages
  + Python
* Test Frameworks & Libraries
  + Pytest
  + Response or Pytest-httpserver
  + Pydantic or jsonschema
  + Pytest-cov
  + Flake8 or ruff
  + Requests or httpx
* Data Handling
  + CSV File Export

Development Tools

* VS Code
* Venv
* Node.js + npm
* Browser

Version Control

* Git
* Github

Virtual Environment

* Venv

Additional Tools

* Lucidchart
* ChatGPT

## Backlog

Description: *This contains the tasks that we have completed each iteration as well as what we are currently working on, what known bugs exist, and what remains to be done. We add to the backlog as we discover new tasks but ignore them until we finish what we are working on.*

# Requirements Gathering

## Functional Requirements

Description: *These requirements are the specific goals that our application should be able to meet.*

* Test Portfolio Retrieval Endpoint
  + The test suite must verify that the /api/portfolio endpoint returns
    - A valid JSON response
    - Correct HTTP status codes (200 OK, 404 for missing portfolios)
    - Accurate portfolio metadata (name, ID, owner, total value)
    - A list of position IDs that are part of the portfolio
* Test Asset Pricing Endpoint
  + The /api/assests enpoint must:
    - Return current pricing for a list of asset symbols
    - Include data fields such as asset symbol, last price, data, and exchange
    - Handle unknown symbols gracefully (return 404 or empty result)
* Validate JSON Schema for All Endpoints
  + All responses must match defined Pydantic or JSON schema models
  + Schema mismatches must be detected and cause test failure

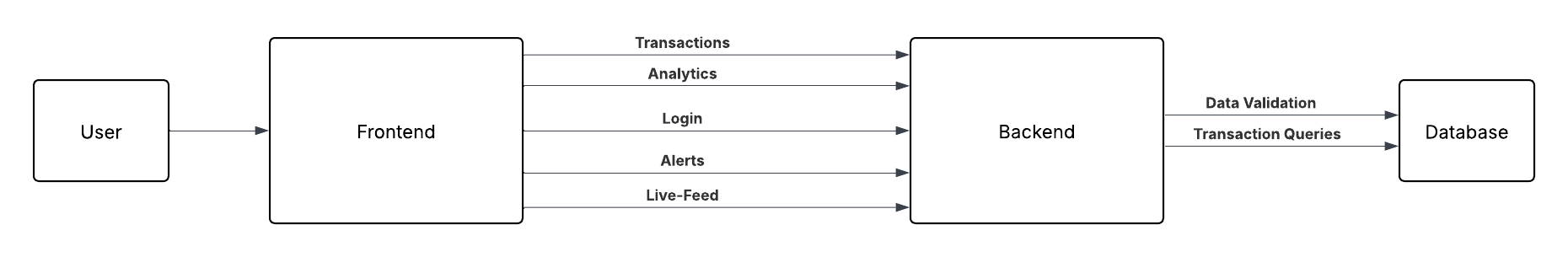
## Non-Functional Requirements

Description: *These requirements are the specific goals that our application should be able to meet.*

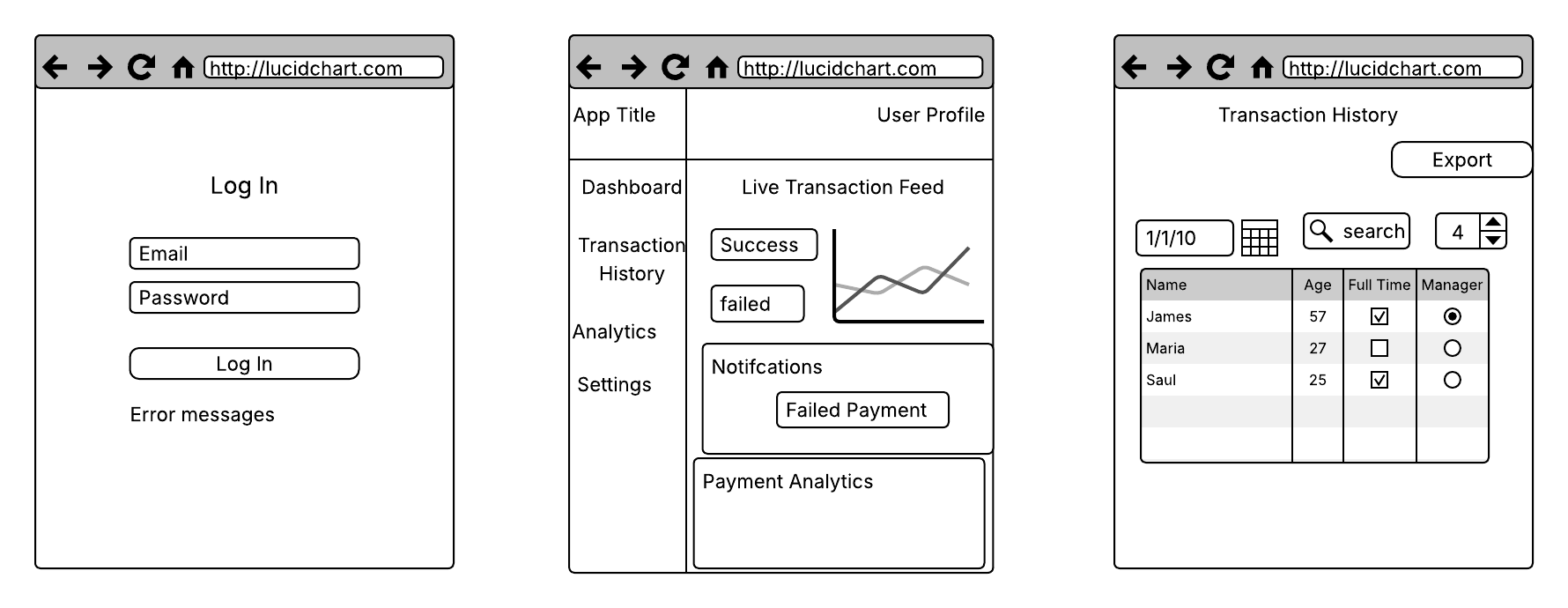
* Performance and Latency Testing
  + The suite must simulate high request volume for /api/portfolio using locust or k6
  + Measure response times and ensure they remain under a defined threshold (<500ms for 95% of request)
* Error and Exception Coverage
  + The system must return appropriate status codes for:
    - Malformed requests (400)
    - Unauthorized access (401)
    - Resource not found (404)
    - Server errors (500)
* Automated CI integration
  + The test suite must be runnable in a CI pipeline and return:
    - Exit codes indicating pass/fail
    - Coverage reports
    - Logs for failed tests

# Design

## API Endpoints



## Wireframes



# Development

## Code Snippets

### app.py

A screenshot of a computer program

AI-generated content may be incorrect.

### LiveTransactionFeed.js

A screenshot of a computer program

AI-generated content may be incorrect.

### LiveTransactionFeed.css

A screen shot of a computer program

AI-generated content may be incorrect.

### Live Transaction Feed

A screenshot of a computer

AI-generated content may be incorrect.

### Payment Analytics

A screenshot of a computer

AI-generated content may be incorrect.

### Notifications

A red line on a white background

AI-generated content may be incorrect.

### CSV Export A screenshot of a computer AI-generated content may be incorrect.

# Review

### Overview

This project was a deep dive into real-time full-stack development. It involved building a React dashboard that listens to a Flask-SocketIO backend simulating live payment data. The goal was to create a functional interface for monitoring transactions in real time, analyzing the data, detecting failures or anomalies, and providing immediate alert and reporting tools.

### What I Learned

### How to build a modular, component-driven React application using best practices.

### How to established live communication using Socket.IO to stream data from a Flask backend.

### How to implemented real-time alerting logic to highlight failed transactions

### How to design and style a responsive UI using vanilla CSS with structured layout and dynamic components.

### How to Create CSV export functionality for transaction history using JavaScript.

### Struggles I Faced

### Socket.IO version mismatches caused silent data delivery failures between React and Flask.

### WebSocket upgrade failures in development led to temporary fallbacks to polling transport.

### 'broadcast=True' emit error after Flask-SocketIO version updates required adjusting emit logic.

### React connection issues due to eventlet monkey-patching errors when imports weren't ordered correctly.

### Attempted login implementation was abandoned mid-project after shifting focus to public, real-time demonstration.

### Successes

### Built a complete, polished real-time monitoring dashboard.

### Achieved clean WebSocket integration using Flask-SocketIO + React (Socket.IO Client).

### Styled a professional UI with dynamic feedback and real-time alert banners.

### Built an analytics module to track success rates, averages, and total values in real time.

### Created a persistent notifications panel and data export functionality to support user interaction.

### Wrote a fully structured and well-documented codebase with a comprehensive README.

### I’m now more confident in:

### Building event-driven front-end apps that sync with a real-time backend.

### Structuring a React codebase with clean separation of components and state.

### Using Flask-SocketIO and understanding the underlying behaviors of WebSocket servers.

### Troubleshooting cross-version compatibility issues between JavaScript libraries and Python servers.

### Designing interactive UIs that respond fluidly to live data updates.