**Lab 4: Stock Picker**

**SED 500 Introduction to Software Engineering**

**Nov 10, 2024**

**Jasleen Kaur**

**Ma Toan Bach**

**Kevin Liu**

Table of Contents

[User Interface 4](#_Toc182474429)

[User interface requirement list 5](#_Toc182474430)

[Operational Details 5](#_Toc182474431)

[Area-Level Details 5](#_Toc182474432)

[Individual Element Details 6](#_Toc182474433)

[User Interface Decision Guided by Triple Constraints 8](#_Toc182474434)

[Cost 8](#_Toc182474435)

[Scope 8](#_Toc182474436)

[Time 9](#_Toc182474437)

[Cost, Time, Scope and Quality 10](#_Toc182474438)

[Cost Reduction 10](#_Toc182474439)

[Reduce Implementation Time 10](#_Toc182474440)

[Control the Scope 11](#_Toc182474441)

[Ensure High Quality 11](#_Toc182474442)

[Detailed Low-level Design of the User Interface 12](#_Toc182474443)

[HTML Structure (StockPicker.html) 12](#_Toc182474444)

[JavaScript Functionality (StockPicker.js) 12](#_Toc182474445)

[The Search Engine 15](#_Toc182474446)

[Strategies to Improve the Search Engine 16](#_Toc182474447)

[Implement Advanced Search Filters: 16](#_Toc182474448)

[Integrate Real-Time Data Updates 16](#_Toc182474449)

[Enhance Data Visualization Tools: 16](#_Toc182474450)

[Alternative Stock Data APIs: 17](#_Toc182474451)

[Alpha Vantage: 17](#_Toc182474452)

[Polygon.io: 17](#_Toc182474453)

[MarketStack: 17](#_Toc182474454)

[Aletheia API: 17](#_Toc182474455)

[Testing and Rework 18](#_Toc182474456)

[Requirements Potentially Not Implemented 19](#_Toc182474457)

[Sustaining Engineering 20](#_Toc182474458)

[Release Strategy for StockPicker Using FMP cloud 21](#_Toc182474459)

[Release 1.0: Core Stock Search and Basic Financial Data 21](#_Toc182474460)

[Release 1.1: Financial Statements and Key Metrics 21](#_Toc182474461)

[Release 1.2: Stock Screener with Filters 22](#_Toc182474462)

[Release 2.0: Historical Data and Data Visualization 22](#_Toc182474463)

[Release 2.1: Advanced Features (DCF, Analyst Ratings, and Real-Time Data) 23](#_Toc182474464)

[Exploring ChatGPT API 24](#_Toc182474465)

[The Risks 25](#_Toc182474466)

[Protecting the Webpage from Viruses 26](#_Toc182474467)

[Handling False Data and Liability Concerns 26](#_Toc182474468)

[Dealing with FMP Cloud Going Out of Business 27](#_Toc182474469)

[User Identity Protection for Account-Based Access 27](#_Toc182474470)

# **User Interface**

## User interface requirement list

### Operational Details

* **Availability**: The Stock Search interface should be accessible at any time. However, peak usage may be expected during market hours.
* **User Access**: Users should be able to access the interface via desktop or mobile web browsers. Accessibility features are required to accommodate different user needs.
* **Expected Volume**: Anticipate moderate user activity, with potential increases during market openings and closings.
* **Platform Compatibility**: Should work across popular browsers like Chrome, Firefox, Safari, and Edge, with responsive design to support mobile devices.

### Area-Level Details

* **Main Screen**:
  + Purpose: Display the search fields and action buttons for stock information retrieval.
  + Layout: Clearly organized sections for entering the "Company Symbol," selecting a "Category," and submitting or resetting the query.
  + Default Category: "Income Statement" should be pre-selected as a default to simplify initial user interaction.
* **Form Section**:
  + Contains input fields and buttons.
  + Provides intuitive labels and adequate spacing for user comfort.

### Individual Element Details

* **Company Symbol Field**:
  + **Label**: “Company Symbol”
  + **Input Type**: Text input with validation to ensure only alphanumeric symbols are allowed.
  + **Tooltip**: “Enter the company’s stock symbol (e.g., AAPL for Apple Inc.).”
* **Category Dropdown**:
  + **Label**: “Category”
  + **Options**: Income Statement, Balance Sheet, Cash Flow, etc.
  + **Tooltip**: “Select the financial report category for the selected company.”
* **Submit Button**:
* **Label**: “Category”
* **Options**:
  + Income Statement
  + Balance Sheet Statement
  + Cash Flow Statement
  + Financial Statements as Reported on SEC
  + Financial Ratios
  + Key Metrics
  + Enterprise Value
  + Financial Statements Growth
  + Discounted Cash Flow Value
  + Rating
  + Market Capitalization
  + Stock Screener
  + Real-time Quote
  + Ticker Search
  + Company Profile
  + Daily Stock Dividend
* **Tooltip**: “Select the financial report category for the selected company.”
* **Default Selection**: “Income Statement” (to simplify the initial interaction)
* **Validation**: Ensures a valid selection is made before submitting the query.
* **Reset Button**:
  + **Label**: “Reset”
  + **Action**: Clears input fields.
  + **Tooltip**: “Clear all fields and start over.”
* **Data Visualization**: Incorporate charts and graphs for data like stock trends, dividends, and financial metrics.

## User Interface Decision Guided by Triple Constraints

### Cost

* **Simple and Cost-Effective Design**:
  + The UI should use standard HTML/CSS with minimal custom styling to avoid high development costs.
  + Avoid using complex animations or high-cost libraries; prioritize open-source tools.
* **Low Maintenance Cost**:
  + Ensure that the UI is easy to update without extensive rework or reconfiguration, reducing future maintenance costs.
  + Use common, well-supported web technologies that minimize the need for specialized expertise.
  + Design UI components (e.g., buttons, input fields) to be modular and reusable
* **Minimal Custom Development:**
  + Implement a clean, minimalist design that reduces the need for complex UI elements or animations, which can increase development time and cost.
* **Use Open-Source Tools and Libraries**:
  + Leverage open-source or free libraries (e.g., Bootstrap for basic styling and layout) to avoid licensing fees and reduce design effort.
  + Use open-source libraries like [Chart.js](https://www.chartjs.org/) for visualization and Bootstrap for styling.
  + Leverage existing templates to speed up development.
* **Avoid Over-Engineering**:
  + Focus on essential functionality only, avoiding features or options that are not required for the initial release. This keeps development costs low while fulfilling user needs.

### Time

* **Rapid Development and Deployment**:
  + Implement the UI with a focus on quick turnaround, using familiar, straightforward web technologies (HTML, CSS, JavaScript) that allow for fast development and testing.
  + Prioritize simple and clear interactions to reduce time spent on complex user flow designs.
* **Efficient Testing**:
  + Limit testing requirements to essential interactions (e.g., validation, button clicks) to streamline QA and deployment.
  + Use basic cross-browser testing to ensure functionality in major browsers without extensive customizations for less common platforms.
* **Rapid Prototyping with Simple Technologies**:
  + Use HTML, CSS, and JavaScript for fast prototyping and development, reducing the learning curve for developers and allowing quicker iteration.
* **Lean Testing Approach**:
  + Limit testing to the essential functionalities (e.g., input validation, dropdown selection, submit action) to streamline the QA process without sacrificing reliability.
* **Adopt Agile Practices**:
  + Implement the UI iteratively, allowing quick feedback and adjustments during development. This helps ensure that requirements are met early and issues are identified sooner.
* **Pre-Built Frameworks**:
  + Utilize pre-built CSS frameworks like Bootstrap or Materialize, which enable fast layout design and styling without building from scratch.

### Control the Scope

* **Define Clear, Limited Requirements**: Focus on the core requirements of the UI—input field for company symbol, category dropdown, and Submit/Reset buttons. This keeps the project on track and avoids scope creep.
* **Prioritize Essential Features**: Restrict the initial UI to essential elements, leaving room for future enhancements without overloading the initial development phase.
* **Flexible Architecture for Future Extensions**: Design the UI with flexibility in mind so that additional fields or categories can be easily added later without restructuring the entire layout.
* **Stakeholder Sign-Off on Design**: Present a prototype or wireframe for stakeholder approval early on. This ensures alignment on scope and prevents unnecessary changes or additions later in development.
* **Core Functionality**:
  + The interface should enable users to input a company symbol, select a financial category, and submit the query with only essential features.
  + Focus on basic validation for required fields (Company Symbol and Category) without overloading the interface with additional options or features not directly requested by the client.
* **Flexible Design for Future Enhancements**:
  + Design the UI to allow for easy addition of new features or fields if needed (e.g., more financial categories, filtering options), without major redesign.
* **Error Handling**:
  + Include basic error messages (e.g., “Please enter a valid company symbol”) to guide users if input validation fails, within the scope of essential functionality.

### Ensure High Quality

* **Consistency in Design**: Use a consistent color scheme, typography, and spacing for a clean, professional appearance. This not only improves the user experience but also maintains a high standard of visual quality.
* **Basic Accessibility Standards**: Ensure fields and buttons are accessible, with proper labels and keyboard navigation support, to cater to users with different needs.
* **Cross-Browser Testing**: Perform tests on major browsers to ensure functionality and appearance consistency, avoiding potential issues for end users.
* **User-Centered Design**: Focus on usability by making sure labels are clear, the layout is intuitive, and interactions (e.g., Submit and Reset buttons) are obvious and straightforward.
* **Error Handling and Validation**: Include basic validation for the input fields (e.g., ensuring the Company Symbol field is not empty) and provide user-friendly error messages to guide users effectively.

## Detailed Low-level Design of the User Interface

### HTML Structure (StockPicker.html)

The HTML file provides the UI layout and forms for user interaction. The key sections include:

* **Title and Form Header**: Title is "Stock Picker" with a form labeled "Stock Search".
* **Input Fields**:
  + **Company Symbol Input** (<input type="text" name="symbol" id="symbol" required="">): A text box for entering the company's stock symbol.
  + **Category Selector** (<select id="category" name="category">): Allows the user to select a specific category for stock search.
  + **Additional Filters**: Based on the selected category, more specific filters like "Stock Screener" and "Stock Value" are conditionally displayed.
* **Event Handling**:
  + **SetVisibility()** function for conditional input visibility based on category choice.
  + **Submit Handling**: Form submission triggers the JavaScript functionality, processing user inputs and initiating API requests.

### JavaScript Functionality (StockPicker.js)

The JavaScript file manages UI interaction and backend data retrieval via HTTP requests. Here’s a breakdown of the modules:

* **HTTP Request Module**:
  + Uses XMLHttpRequest to fetch data related to stock prices and dividends from an API.
  + Modules for initiating requests (xhr1, xhr2) and handling responses.
* **Form Handling Module**:
  + **Symbol and Category Extraction**: Extracts the stock symbol and selected category from the form on submission.
  + **Category-Specific Actions**: If the selected category is “stock-screener,” further parameters are captured to refine the search (like Screener and Value).
  + **Submit Action**: Validates inputs and sends requests based on user selection.
* **Data Parsing and Display Module**:
  + **DOMParser** for parsing XML data from API responses.
  + **Data Variables** (historicalPrice, historicalDividend): To store historical stock data.
  + **Chart Handling** (chart): Variable prepared for charting, used with Chart.js.

**Interaction Design**

* **User Actions:**
* **Select Category**: Triggers visibility changes in UI fields.
* **Submit**: Sends the symbol and filter data for processing.
* **Display Modules:**
* **Dynamic Result Display**: Upon data retrieval, the UI updates with stock data and historical data visualization.

**Language Requirements and Libraries**

* **Programming Languages**:
  + **HTML and JavaScript** for UI and dynamic interaction.
  + **CSS** (referenced as styles.css) for styling.
* **Potential Graphics Library**:
  + The chart variable suggests a graphics library like **Chart.js** or **D3.js** could be used for data visualization.

**Interaction Flow**

1. **User Input**: Enters a stock symbol and selects a category.
2. **Visibility Adjustment**: If certain categories require additional fields, JavaScript reveals these options.
3. **Submit and Fetch**: On form submission, JavaScript processes the inputs and sends an HTTP request to retrieve stock data.
4. **Parse and Display**: The system parses response data, updates relevant fields, and potentially visualizes data in the chart area.

# The Search Engine

## Strategies to Improve the Search Engine

### Implement Advanced Search Filters:

* + **Sector and Industry Filters**: Allow users to narrow down searches by specific sectors (e.g., Technology, Healthcare) or industries, facilitating targeted stock analysis.
  + **Financial Metrics**: Enable filtering based on key financial indicators such as price-to-earnings (P/E) ratio, dividend yield, and market capitalization, aiding in the identification of stocks that meet specific investment criteria.
  + **Performance Indicators**: Incorporate filters for historical performance metrics, including year-to-date returns or volatility measures, to assist users in assessing stock stability and growth potential.

Integrate Real-Time Data Updates:

* + **Live Market Data**: Provide users with up-to-the-minute stock prices, trading volumes, and market news to support informed decision-making.
  + **Automated Refresh**: Implement features that automatically update displayed data at regular intervals, ensuring users have access to the latest information without manual intervention.

### Enhance Data Visualization Tools:

* + **Interactive Charts**: Offer customizable and interactive charts that allow users to analyze stock trends over various timeframes, with options to apply technical indicators.
  + **Comparison Features**: Enable side-by-side comparisons of multiple stocks, facilitating a comprehensive analysis of potential investments.

## Alternative Stock Data APIs:

To improve data quality and functionality, consider integrating the following reputable stock data APIs:

### [Alpha Vantage](https://www.alphavantage.co/):

* + **Features**: Provides free APIs in JSON and CSV formats for real-time and historical stock market data, covering options, forex, commodities, and cryptocurrencies. It also offers over 50 technical indicators and global market news powered by AI and machine learning.

### [Polygon.io](https://polygon.io/):

* + **Features**: Offers real-time and historical tick data for stocks, with unlimited usage via REST or WebSockets. Data is available in standardized JSON and CSV formats, and an instant access free tier is provided.

### [MarketStack](https://marketstack.com/):

* + **Features**: Delivers a free, easy-to-use REST API interface providing worldwide stock market data in JSON format. It supports real-time, intraday, and historical market data, covering over 170,000 stock tickers from more than 70 global exchanges.

### [Aletheia API](https://aletheiaapi.com/):

* + **Features**: A free, real-time financial data API offering stock data, financials, SEC filings, fundamental analysis, and insider trading information. It provides unlimited access to millions of transactions and data points, both real-time and historical.

# Testing and Rework

## Requirements Potentially Not Implemented

|  |  |  |
| --- | --- | --- |
| Requirement  Identifier | Description | Reasons |
| RE-01.5 | The stock search UI must allow a user to select a category. The categories are: income statement, balance sheet statement, cash flow statement, financial statements as reported on SEC, financial ratios, key metrics, enterprise value, financial statements growth, discounted cash flow value, rating, market capitalization, stock screener, real-time quotes, ticker search, company profile, and daily stock dividend. | “Real-time quotes” is a typo from our manager. Testers only follow the requirements |
| RE-05 | The stock screener UI should display all data in columns of four after a submit query. | I’m using a small screen so it is display in columns of one after a submit query |
| RE-07 | The reset should remove all data from a submit query. | Not seen |
| RE-08 | The reset must return to the default stock search UI. | Not seen |

Please refer to the code for the test scripts.

# Sustaining Engineering

## Release Strategy for StockPicker Using FMP cloud

### Release 1.0: Core Stock Search and Basic Financial Data

**Features**:

* **Basic Stock Search**: Allows users to search for stock by symbol.
* **Basic Financial Data**: Fetch key data points such as market capitalization, P/E ratio, and daily price.
* **Error Handling**: Display error messages for invalid symbols or if data retrieval fails.
* **UI Design for Core Elements**: Simple interface for search input and data display.

**Goals**:

* **Reduce Costs**: By keeping the initial version simple with limited API calls, we can start with Fmp Cloud’s free tier and minimize development costs.
* **Reduce Implementation Time**: Focus on implementing only the essential features.
* **Quality Assurance**: Conduct unit testing and integration testing to ensure stable data retrieval and UI display.

### Release 1.1: Financial Statements and Key Metrics

**Features**:

* **Add Financial Statements**: Integrate endpoints to display income statements, balance sheets, and cash flow statements.
* **Key Financial Ratios**: Include key metrics like ROE, ROA, and gross margin.
* **UI Enhancement**: Update UI to organize financial data into tabs or collapsible sections for better readability.

**Goals**:

* **Reduce Costs**: Still stay within the free or low-cost API usage by limiting requests per session.
* **Scope Control**: Add only the most requested financial statements to avoid API overload.
* **Quality Control**: Perform usability testing to ensure data readability and that navigation is intuitive.

### Release 1.2: Stock Screener with Filters

**Features**:

* **Stock Screener**: Enable filtering stocks by metrics such as market capitalization, P/E ratio, dividend yield, and beta.
* **Data Caching**: Implement caching for frequently accessed data to reduce API call frequency and costs.
* **UI Enhancements for Screener**: Add a dedicated section for filtering options, allowing users to set multiple criteria.

**Goals**:

* **Cost Reduction**: Use caching to limit redundant API requests and reduce API costs.
* **Implementation Time Reduction**: Limit filters to the most popular ones (market cap, P/E, dividend yield).
* **High Quality**: Run performance tests to ensure screener operates smoothly even with multiple filters.

### Release 2.0: Historical Data and Data Visualization

**Features**:

* **Historical Data**: Allow users to view historical stock data, including daily price, for trend analysis.
* **Data Visualization**: Integrate with a library like Chart.js or D3.js to display stock trends graphically.
* **Enhanced Error Handling**: Include more detailed error handling for scenarios like API limits or connectivity issues.

**Goals**:

* **Cost Control**: Use a moderate-tier API plan if demand justifies it, focusing on limiting the frequency of historical data requests.
* **Scope Control**: Introduce only basic chart types (e.g., line charts for price trends).
* **Ensure High Quality**: Conduct load testing on charts to ensure performance remains smooth with increased data.

### Release 2.1: Advanced Features (DCF, Analyst Ratings, and Real-Time Data)

**Features**:

* **Discounted Cash Flow (DCF) Valuation**: Provide a DCF calculation for companies, allowing users to assess stock valuation.
* **Analyst Ratings**: Display ratings, target prices, and recommendations from analysts.
* **Real-Time Data**: Add an option for users to enable real-time updates for stock prices, if API costs permit.

**Goals**:

* **Cost Control**: Make real-time data optional or on-demand, as it requires a higher-tier API plan.
* **High Quality**: Ensure real-time data is accurate and tested under various market conditions.
* **Implementation Time**: Focus on core valuation metrics like DCF to keep scope manageable.

## Exploring ChatGPT API

**from openai import OpenAI**

**client = OpenAI(api\_key="MY\_API\_KEY")**

**response = client.chat.completions.create(**

**model="gpt-4o-mini",**

**messages=[{"role": "user", "content": "Explain what a Price-to-Earnings (P/E) ratio is."}],**

**)**

**print(response.choices[0].message.content)**

# The Risks

## Protecting the Webpage from Viruses

* **Security Measures**: Implement Content Security Policies (CSP) to restrict the types of content that can be loaded. This helps prevent malicious scripts from being executed.
* **Sanitizing User Input**: Use libraries like DOMPurify for sanitizing user input to prevent cross-site scripting (XSS) attacks.
* **Regular Security Audits**: Conduct regular audits and vulnerability assessments to identify and mitigate potential threats.
* **Server Security**: Keep server software up-to-date, apply security patches, and use firewalls and intrusion detection systems (IDS) to protect against unauthorized access.

## Handling False Data and Liability Concerns

* **Disclaimer and Terms of Service**: Clearly state in the terms of service and at the data display points that the information is for informational purposes only, and the provider is not liable for decisions based on the data.
* **Source Verification**: Indicate that data is sourced from a reputable API like FMP Cloud but note that the accuracy of data is dependent on the data provider.
* **Real-Time Data Verification**: If possible, implement checks against multiple sources to verify data accuracy.
* **Fallback Mechanism**: In cases where data accuracy is critical, use cached or previously verified data to avoid showing completely incorrect information if there are temporary issues with the live data.

## Dealing with FMP Cloud Going Out of Business

* **Modular Code Design**: Structure the application code with an abstraction layer, allowing the data provider (API) to be swapped with minimal changes.
* **Alternative API Compatibility**: Implement the system with modular functions so that each data retrieval call is made through a standardized interface. This enables easy integration of other APIs (e.g., Alpha Vantage, Polygon.io) if needed.
* **Data Caching**: Cache historical data where feasible. If FMP Cloud becomes unavailable temporarily, cached data can still provide users with recent information.
* **Explore Alternative Providers**: Plan by researching and testing alternative APIs that could be seamlessly integrated, ensuring the system remains functional without significant redevelopment.

## User Identity Protection for Account-Based Access

* **Secure Authentication**: Use secure protocols like OAuth 2.0 for user authentication. Avoid storing sensitive data directly and use encrypted tokens for session management.
* **Data Encryption**: Encrypt all personal data both in transit (using HTTPS) and at rest (using encryption standards such as AES).
* **Minimal Data Collection**: Collect only necessary information for user accounts to reduce the impact of a data breach.
* **Two-Factor Authentication (2FA)**: Offer 2FA to enhance account security.
* **Privacy Policy and Data Deletion Options**: Clearly outline how user data is stored and managed, and provide users with options to delete their data upon request.