**System Design Document: Automated Portfolio Insights Application**

**1. Overview**

The Automated Portfolio Insights application is designed to automate the tracking and management of my stock portfolios by integrating with Charles Schwab API to fetch portfolio data, updating Google Sheets with the fetched data, and generating insights on portfolio performance. This system will help me efficiently manage my investments across different account types such as Roth IRA, Primary, Options.

**2. System Architecture**

**2.1 High-Level Architecture**

The system is built on a serverless architecture using Azure Functions, integrating with Google Sheets API and Charles Schwab API. The application will use Python as the primary programming language to handle data fetching, processing, and updating Google Sheets.

* **Azure Function App:** A Python-based serverless compute service that triggers portfolio updates weekly.
* **Charles Schwab API:** Provides access to portfolio data.
* **Google Sheets API:** Facilitates updating Google Sheets with portfolio data.
* **Azure Key Vault:** Securely stores API keys and sensitive information.

**2.2 Components Overview**

* **Function App:** Hosts the logic for fetching portfolio data and updating Google Sheets.
* **Scheduler:** Schedules the function to run every Saturday at 9:30 AM.
* **Portfolio Data Fetcher:** Handles API requests to fetch data from Charles Schwab.
* **Google Sheets Updater:** Updates the relevant sheets with the fetched data.
* **Logging:** Provides logging information for function execution.

**3. System Requirements**

**3.1 Functional Requirements**

* **Data Fetching:** The system should fetch portfolio data from Charles Schwab API for Roth IRA, Primary, and Options accounts.
* **Data Processing:** Process the fetched data to extract relevant financial insights.
* **Google Sheets Update:** Update predefined Google Sheets with the latest portfolio data.
* **Scheduling:** Automatically trigger data fetching and update operations every Saturday at 9:30 AM.

**3.2 Non-Functional Requirements**

* **Scalability:** The system should handle large data sets from multiple portfolios.
* **Security:** API keys and sensitive information must be stored securely using Azure Key Vault.
* **Reliability:** Ensure the function executes successfully and logs any errors for troubleshooting.
* **Performance:** The system should complete data fetching and updates within a reasonable time frame (e.g., less than 1 minute).

**4. Detailed Design**

**4.1 Azure Function App**

* **Runtime:** Python 3.11
* **Trigger:** Timer-based trigger scheduled for every Saturday at 9:30 AM.
* **Code Structure:**
  + PortfolioDataFetcher: Handles API requests to Charles Schwab.
  + UpdateGoogleSheets: Handles updating the Google Sheets with fetched data.
  + app.schedule: Main function that orchestrates the data fetching and updating process.

**4.2 Google Sheets API Integration**

* **Authentication:** Google Sheets API authentication using a service account.
* **Sheets:** Two main sheets are updated:
  + **Net Worth:** Summary of the total value of assets.
  + **Stock Portfolio:** Detailed breakdown of individual investments.

**4.3 Charles Schwab API Integration**

* **Endpoints:** Specific endpoints for Roth IRA, Primary, and Options accounts.
* **API Key:** Stored securely in Azure Key Vault.

**5. Deployment**

**5.1 Bicep Template**

The deployment uses a Bicep template to provision the necessary Azure resources, including:

* **Function App:** Configured with the required settings and linked to the storage account.
* **Application Insights:** Monitors function execution and provides analytics.
* **Key Vault:** Securely stores the API keys.

**5.2 Deployment Steps**

1. Deploy the Bicep template to provision the Azure resources.
2. Configure the Azure Function App with the necessary environment variables.
3. Upload the Python code to the Function App.
4. Schedule the function to run weekly.

**6. Security Considerations**

* **API Keys:** Stored securely in Azure Key Vault and accessed via managed identities.
* **OAuth Authentication:** Used for secure access to Google Sheets API and Charles Schwab API.
* **HTTPS Only:** All communication is secured using HTTPS.

**7. Monitoring and Logging**

* **Application Insights:** Provides real-time monitoring, logging, and alerting for the Azure Function App.
* **Logs:** Detailed logs are captured for each function execution, including success, failure, and performance metrics.

**8. Testing and Validation**

* **Integration Testing:** Test the complete workflow from data fetching to sheet updates.
* **Performance Testing:** Ensure the function executes within the desired time frame.

**9. Future Enhancements**

* **User Interface:** Develop a dashboard for users to visualize portfolio performance.
* **Multi-User Support:** Extend the application to support multiple users with different portfolios.
* **Customizable Schedules:** Allow users to customize the frequency of data updates.

**10. Conclusion**

This Automated Portfolio Insights application is designed to streamline the process of managing my stock portfolios, providing me with real-time updates and insights through a serverless, scalable, and secure architecture. The design ensures ease of deployment, monitoring, and future extensibility.