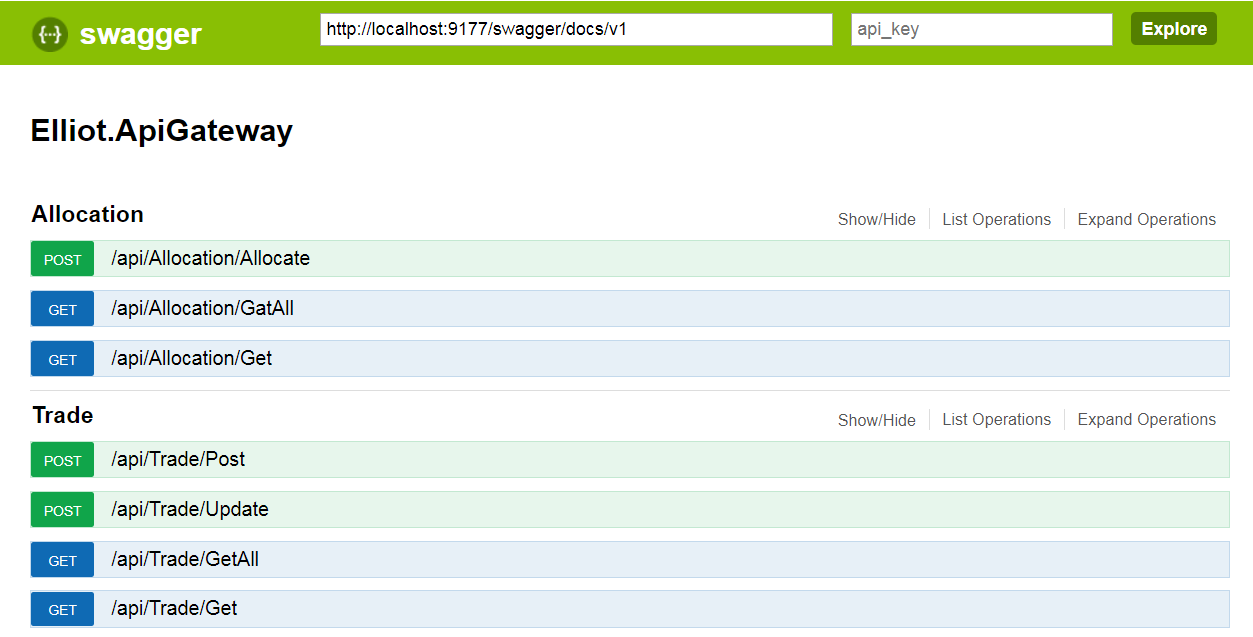
# Elliot Coding Exercise Implementation Summary

## Introduction

The required JSON API is implemented using Restful JSON Service hosted in a Windows service. Along with the restful service, a swagger is configured to make the API visible and more user friendly.

You can run the solution either by clicking the executable directly or running it in a Visual Studio. After the service gets started, you can view, browse or test the API in Chrome with this URI - <http://localhost:9177/swagger>. Below is a screenshot of the API implemented:



## 2. Functions Short Description

|  |  |  |
| --- | --- | --- |
| **End Point** | **Function Name** | **Short Description** |
| Allocation | Allocate | Allocate one or more trades into Fund A and Fund B |
| Allocation | GetAll | Return all allocations for securities traded so far in a day |
| Allocation | Get | Return the allocation for a security by symbol |
| Trade | Post | Post a new trade |
| Trade | Update | Amend an existing unallocated trade |
| Trade | GetAll | Return all trades so far in a day |
| Trade | Get | Return a trade by trade id ; can be used for allocation status check |

## 3. Assumptions

1. Securities only have two asset classes: equity and future.
2. Allocation supports only two funds: fund A and fund B.
3. All trades are denominated in U.S. dollars.
4. A rollover process will reset the system each day.

## 4. Scope of Work

The implementation is mainly to demonstrate a small software system design using OO methodology and the popular SOLID design pattern. C# is used for implementation. Key points of the work:

1. A workable solution can be executed in the console or installed as a windows service.
2. Restful API service can be tested and viewed through the browser (Chrome)
3. All business functional requirements have been implemented as descripted in section 2.
4. Thread safety has been considered. The slim read & write lock is used to guard shared data access.
5. Some unit tests have been implemented to test the main business functional requirements.
6. In memory cache is used to store trades and allocations.

Given the time constraints, the following are not included in the implementation but can be done:

1. Persist data to database or files.
2. In memory system crash/recovery.
3. Information logging.
4. Rest API permission control.
5. Task striping or other no-lock design
6. Allocation not limited to two funds, or two asset classes.
7. All exceptions are handled.
8. More detailed responses for API users.
9. Security, Trade, and Allocation can be designed with more attributes.

If you have any questions, please do not hesitate to reach out to me.