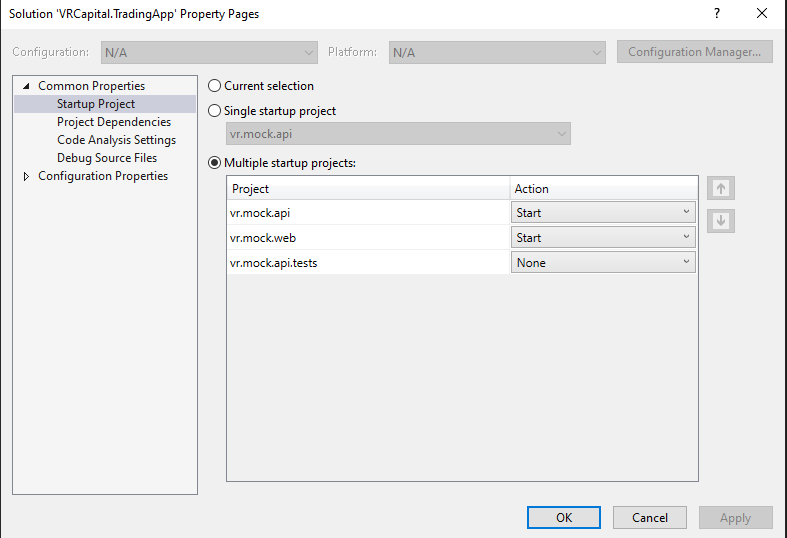
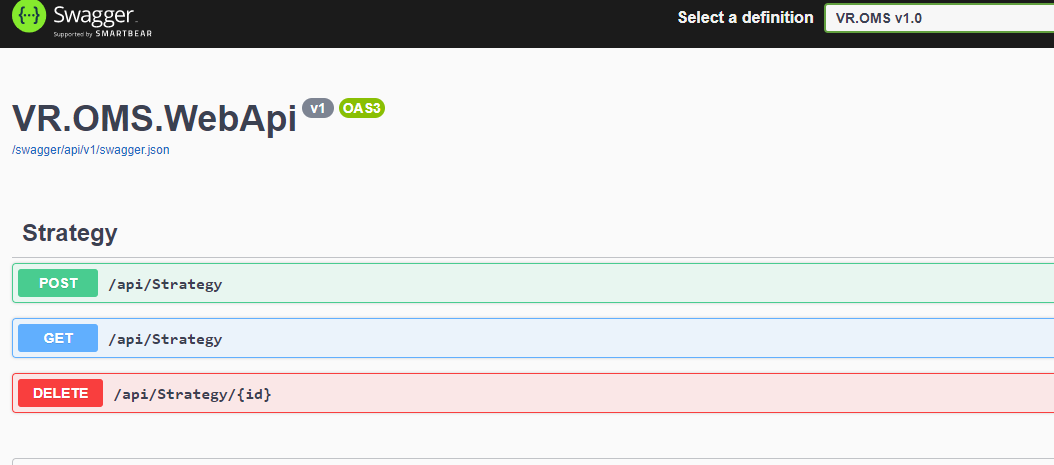
# **How to run the application:**

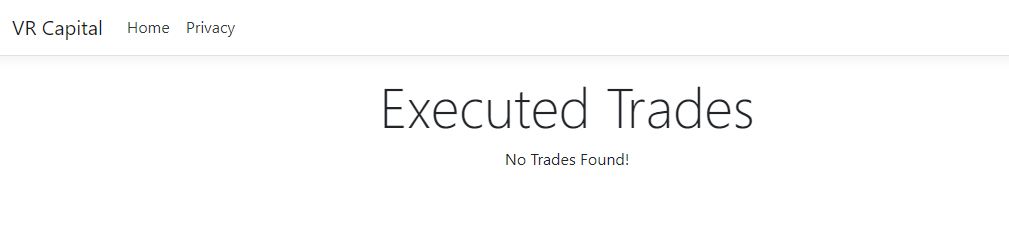
1. Clone the Git repository (<https://github.com/birajbhow/VRCapital.TradingApp>) on local machine
2. Open the solution in visual studio from below location,
   1. For E.g. “C:\Users\BBhow\source\repos\VRCapital.TradingApp\VRCapital.TradingApp.sln”
3. Follow the below steps to run both Api and Web projects together
   1. Right click solution and select “Set Startup Projects…” from the menu
   2. Select options as shown in the image below



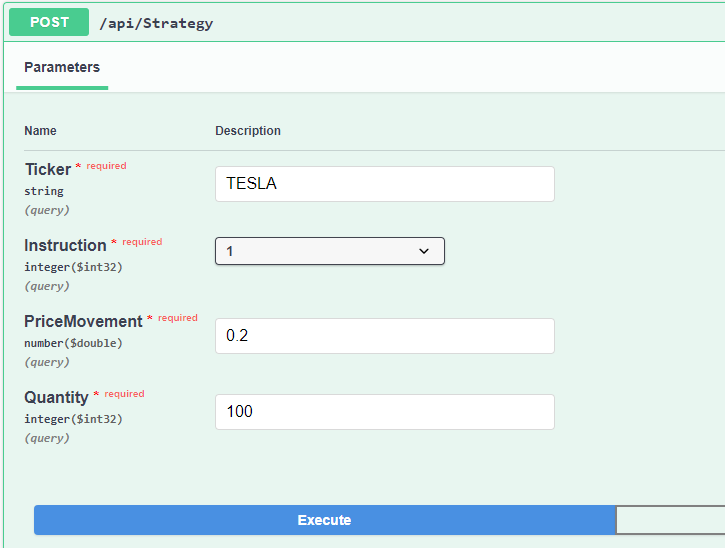
1. Clean and Build the solution.
2. Click Start button in the tool bar to run the applications
3. The Api project will open the swagger UI in the browser window



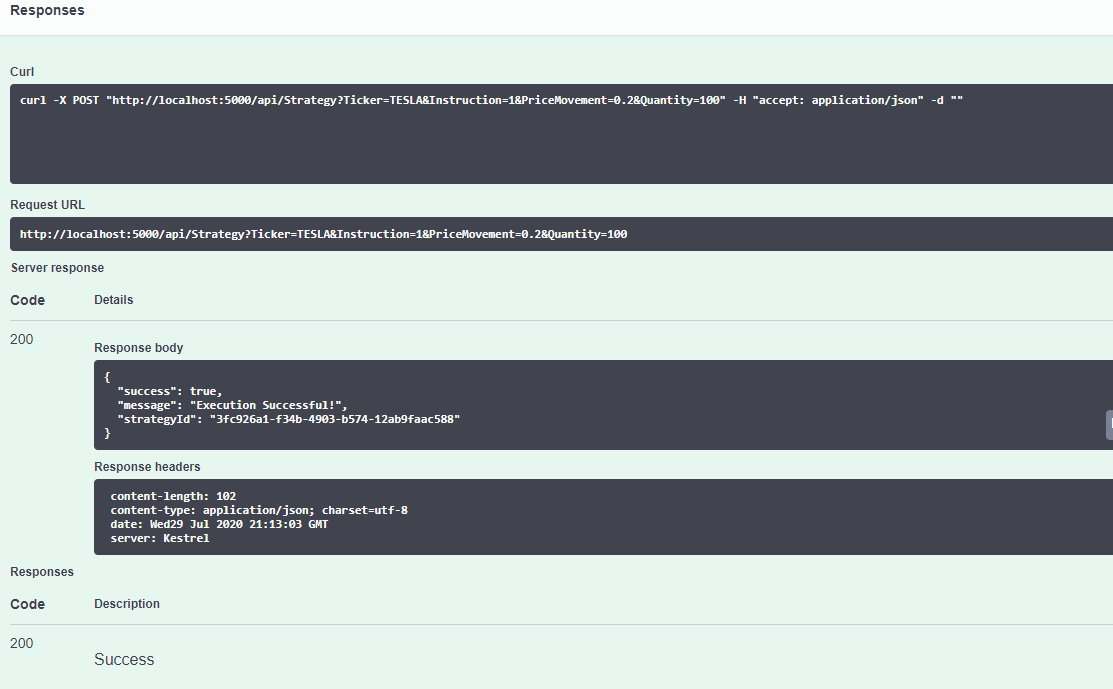
1. The Web project will display initial trade page



1. Register new strategy using the api POST endpoint via swagger UI as shown below,

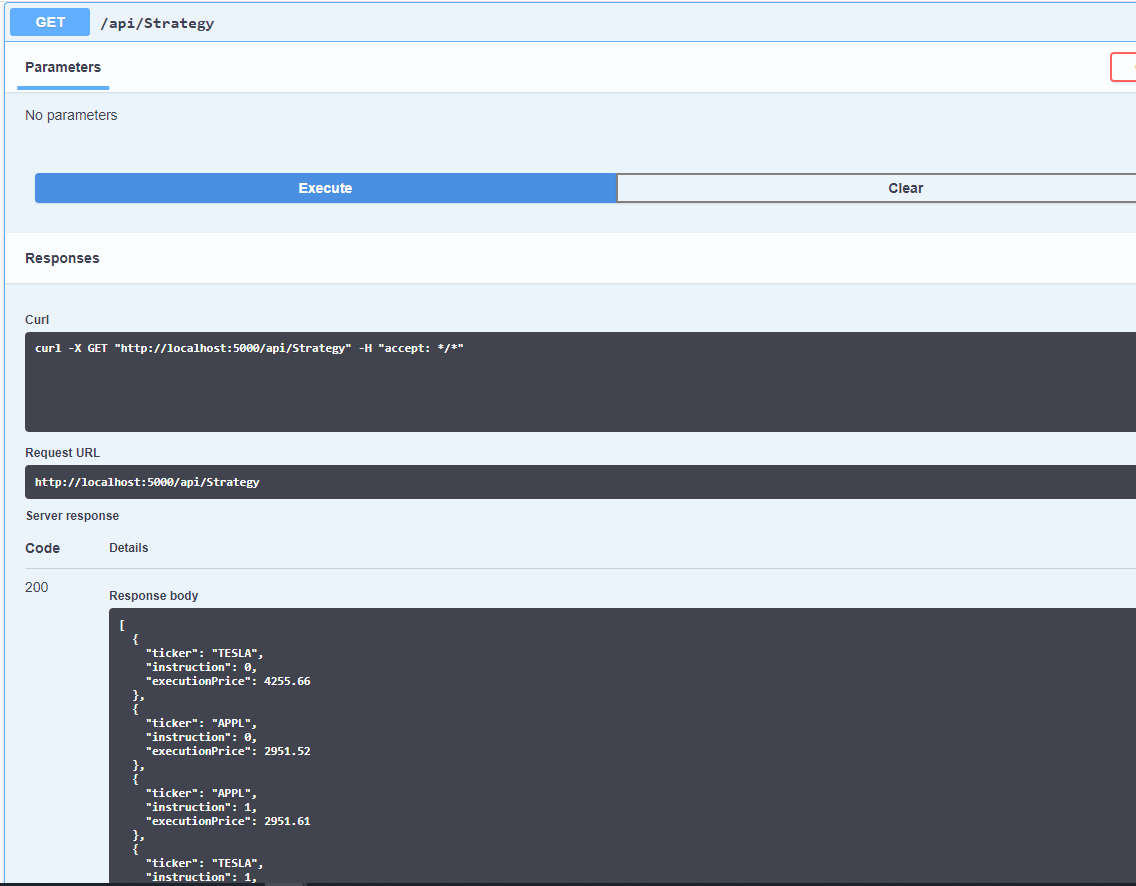


1. The API POST end point response will look like below,

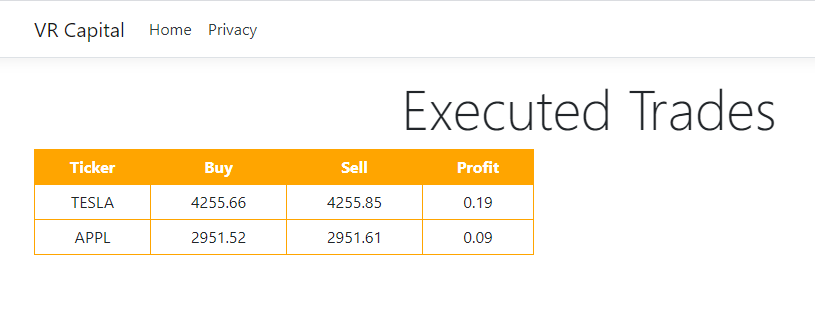


* 1. If the execution of the POST request has failed, it will return success: false

1. Register few more strategies with different ticker and both buy and sell instructions.
   1. Note: If you have not added both Buy and Sell strategies for a ticker, that ticker will not appear in the web UI.
2. Run the API GET endpoint to retrieve all executed strategies. The executed strategies GET response will look like below,



1. The API DELETE endpoint can be run by copying the registered strategy id from the POST endpoint response object.
2. When the API GET endpoint returns few executed strategies, refresh the Web UI page to see the executed trade list as shown below,



# **Source Code Explanations:**

* Tried to write clean code based on SOLID design principles
* Most of the code is self-explanatory but added comments at appropriate places.
* Added some unit tests but room to add lot more.
* The API code is divided into few services as explained below
  + ILocalCache
    - This service helps to put/delete/get objects in local in-memory cache using singleton thread safe object.
    - In production, this service can be replaced by some high throughput distributed cache
  + IStrategyRepository
    - This service is built based on repository design pattern to handle all the strategy add/remove/get operations.
    - Internally it uses ILocalCache service above to store strategies in local api memory cache.
  + IStrategyTradingService
    - This service is wrapper on top of the existing VRTradingService to handle the various exceptions thrown by the different network calls like GetQuote, Buy and Sell.
  + IStrategyManagementService
    - This service runs in the background, checking the registered strategies periodically (1 sec) against the live quote.
    - It encompasses the business logic of whether to execute the strategy or not.
  + StrategyController
    - It is a public interface of the api.
    - It exposes endpoints to register/unregister and retrieve strategies.
* The Web UI is built using a simple MVC web application project template
  + The reason to select this template is my past familiarity with MVC applications and quickly build simple UI within given time frame.
  + It makes a REST calls to retrieve the executed strategies to the strategy trading API running above.
  + The Flurl, fluent REST request library, is used to make the REST calls to the API.
  + It groups the ticker and calculates the profit based on buy and sell strategy transactions
  + Note: For a particular ticker to appear on this screen, both buy and sell strategies must have been executed.