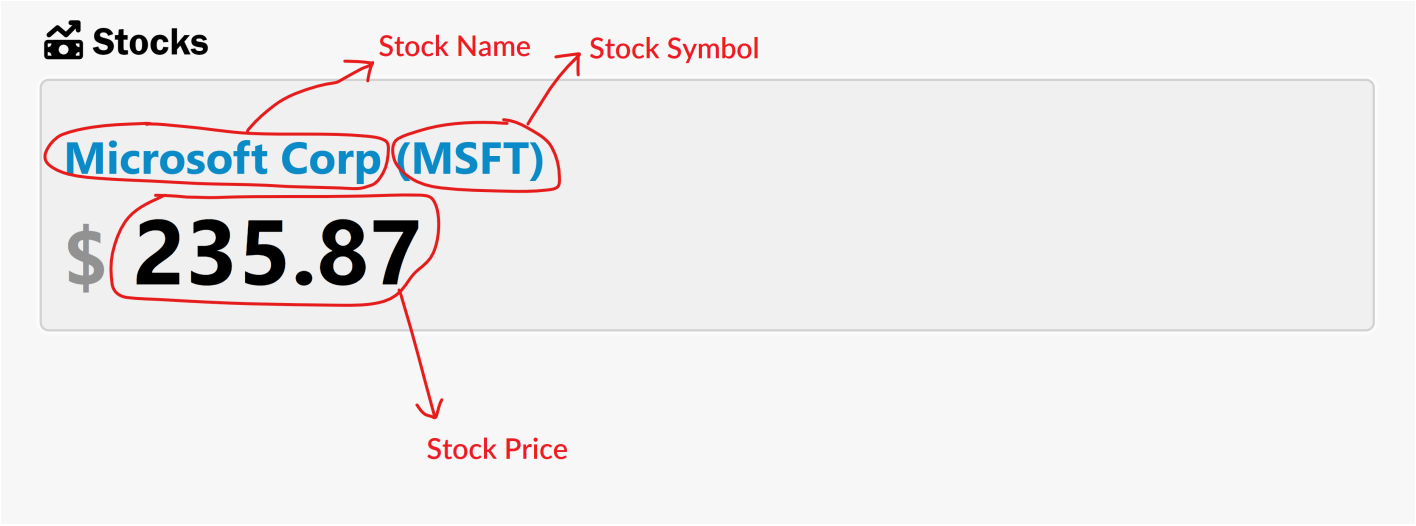
# Requirement

Create an Asp.Net Core Web Application that displays stock price with live updates from "**https://finnhub.io/**".

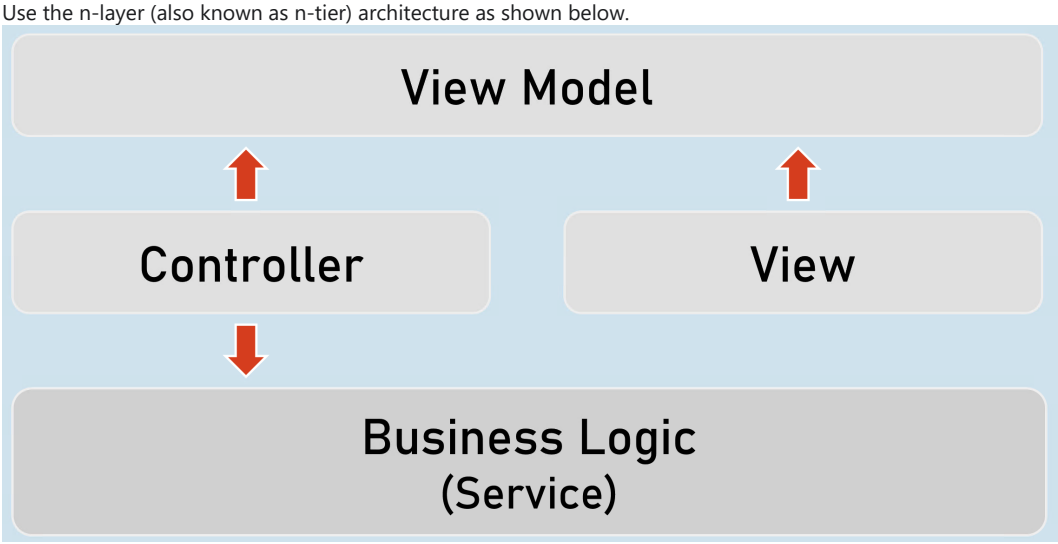
# UI Design

**Trade/Index:**




# Architecture

Use the n-layer (also known as n-tier) architecture as shown below.




# Finnhub.io:

https://finnhub.io is a service provider that provides live stock price information online.

# User-Secrets:

- Register in "**https://finnhub.io/login**" to generate your own token [or] use the token "**cc676uaad3i9rj8tb1s0**" to make requests.
- After registration at finnhub, you can find your API Key (token) at "https://finnhub.io/dashboard".

- You need to store this token in user-secrets on your machine.
- You need to attach the token as a part of request url while making requests to "finnhub.io".

# appsettings:

You will store the following configuration in appsettings.json:

```
"TradingOptions":
{
  "DefaultStockSymbol": "MSFT"
}
```

- The "DefaultStockSymbol" acts as a default stock to fetch details and display stock price.

# FinnhubService:

Create a service interface called '**IFinnhubService**' with following methods:

Task<Dictionary<string, object>?> GetCompanyProfile(string stockSymbol);

Task<Dictionary<string, object>?> GetStockPriceQuote(string stockSymbol);

Implement the above service interface called '**IFinnhubService**' that sends request to the respective url and returns its response as a Dictionary<string, object>.

GetCompanyProfile: https://finnhub.io/api/v1/stock/profile2?symbol={symbol}&token={token}

GetStockPriceQuote: https://finnhub.io/api/v1/quote?symbol={symbol}&token={token}

The service methods such as GetCompanyProfile() and GetStockPriceQuote() return the response data that was actually returned by finnhub.io/api.

# IFinnhubService.GetStockPriceQuote():

The returned data from IFinnhubServiceGetStockPriceQuote() looks like this:

{"c":235.87,"d":9.12,"dp":4.0221,"h":236.6,"l":226.06,"o":226.24,"pc":226.75,"t":1666987204}

**Response Attributes:**

c

Current price

d

Change

dp

Percent change

h

High price of the day

l

Low price of the day

o

Open price of the day

pc

Previous close price

Reference: https://finnhub.io/docs/api/quote

# IFinnhubService.GetCompanyProfile():

The returned data from IFinnhubService.GetCompanyProfile() looks like this:

{"country":"US","currency":"USD","exchange":"NASDAQ NMS - GLOBAL MARKET","finnhubIndustry":"Technology","ipo":"1986-03-13","logo":"https://static2.finnhub.io/file/publicdatany/finnhubimage/stock_logo/MSFT.svg","marketCapitalization":1758286.5806001066,"name":"Micros Corp","phone":"14258828080.0","shareOutstanding":7454.47,"ticker":"MSFT","weburl":"https://www.microsoft.com/en-us"}

**Response Attributes:**

country

Country of company's headquarter.

currency

Currency used in company filings.

exchange

Listed exchange.

finnhubIndustry

Finnhub industry classification.

ipo

IPO date.

logo

Logo image.

marketCapitalization

Market Capitalization.

name

Company name.

phone

Company phone number.

shareOutstanding

Number of oustanding shares.

ticker

Company symbol/ticker as used on the listed exchange.

weburl

Company website.

Reference: https://finnhub.io/docs/api/company-profile2

# StockTrade:

You need to create a view model class called "**StockTrade**".

Create a view model class called "StockTrade" in the Asp.Net Core project with following properties:

string? StockSymbol

string? StockName

double Price

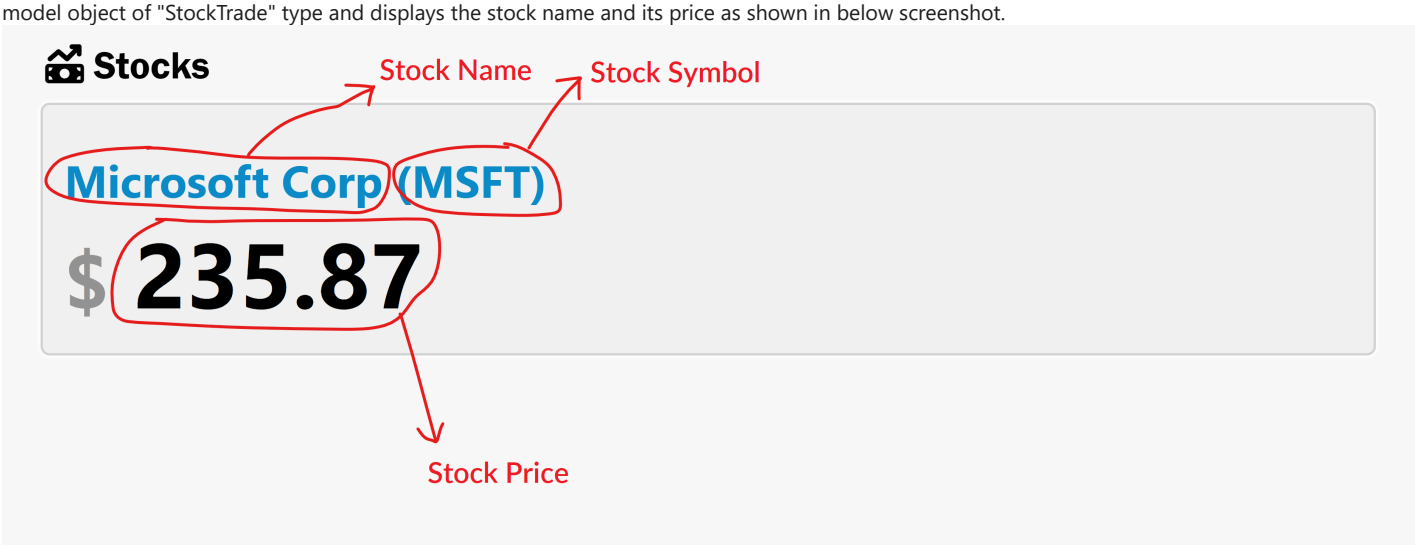
unit Quantity

This class will be used to send model object from controller to the "Trade/Index" view.

# "Trade\Index.cshtml":

You need to create a view called "Trade\Index.cshtml".

Create a view called "Index.cshtml" in "Views\Trade" folder - that is strongly typed to the view model class called "StockTrade". So it receives the model object of "StockTrade" type and displays the stock name and its price as shown in below screenshot.



The view should store the stock symbol (Eg: MSFT) as a hidden element. So that, it can be accessible in the javascript code, while connecting finnhub websocket.

So, you need to write javascript code to connect to finnhub websocket at `**wss://ws.finnhub.io?token=${token}**` url. On receiving each message from the socket, you need update the latest stock price in the UI.

The response from finnhub websocket looks like this:

{"data":[{"p":220.89,"s":"MSFT","t":1575526691134,"v":100}, {"p":220.82,"s":"MSFT","t":1575526691167,"v":15}],"type":"trade"}

**Response attributes:**

type: message type

data: [ list of trades ]

s: symbol of the company

p: Last price

t: UNIX milliseconds timestamp

v: volume (number of orders)

c: trade conditions (if any)

You can see more than one element in the 'data' array in the above response example. But in the output, we need to display only one. You can display either of the prices (or) the highest price in the output.

When the page is closed, you must unsubscribe from the finnhub websocket to avoid any memory leaks.

Overall, your application should refresh the updated price of the stock for every one or two seconds as long as the page runs (of course, only when the market is LIVE i.e. usually 09:30 a.m. to 4 p.m. (ET)).

# Trade Controller:

Create a controller called TradeController that has five action methods called "**Index**()".

The controller has to inject the appsettings called "TradingOptions" (from appsettings.json) and IFinnhubService.

# TradeController.Index():

- It receives HTTP GET request at route "Trade/Index".

- It first gets the "TradingOptions" from appsettings.json using Options pattern.

- The Index() action method invokes the following methods:

1. FinnhubCompanyProfileService.GetCompanyProfile() to fetch stock name, stock symbol and other details.
2. FinnhubStockPriceQuoteService.GetStockPriceQuote() to fetch current stock price.

- And then it creates an object of "StockTrade" model class and fills essential data such as StockSymbol, StockName, Price and Quantity that are read from the return values of above mentioned service methods i.e. "FinnhubService.GetCompanyProfile()" and "FinnhubService.GetStockPriceQuote()".

- Then it sends the same model object to the view.

# Instructions:

1. Create controller(s) with attribute routing.
2. Provide the configuration as service, using Options pattern.
3. Inject the IOptions in the controller.
4. Use CSS styles, layout views, _ViewImports, _ViewStart as per the necessity.
5. The CSS file is provided as downlodable resource for applying essential styles. You can download and use it.
6. Inject IFinnhubService in Controller.
7. Invoke essential service methods in controller.

### Questions for this assignment

Check your source code with Instructor's source code.