# BUILDING AN OPTIONS CHAIN

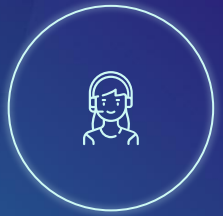
And finding Implied Volatility

# OUR TEAM

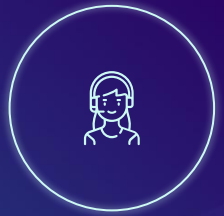
ANANYA SENGUPTA

SHREYA MOILY

TRISHA GHOSH



SHRAVANI GUCHHAIT



ANISHA SAH

# TABLE OF CONTENTS

# OBJECTIVES

The objective of building this website is to provide a platform or application that allows users to access and analyze options data for a particular underlying asset, such as stocks, indices, or commodities. Option chains then display a list of available options contracts for the selected asset, including their strike prices, expiration dates, and other relevant details.

1. Accessibility: Provide users with an easy-to-use interface to access and view option chains for various assets. This includes the ability to filter and search for specific options based on different criteria.

2. Real time data: Integrated with a reliable data source to ensure that the option chains tool reflects the latest market data. Real-time updates are crucial for accurate analysis and decision-making.

# PROBLEM STATEMENT

The candidates will be provided with a market data stream over TCP/IP which will contain the market data structure as given below. Your goal is to process the market data and calculate Implied volatility (IV), etc. and display as an options chain screen (e.g. https://www.nseindia.com/option-chain ) as a webpage. The following points should be catered.

- Highlight the "in the money" options and "out of money" options differently as shown in above example.
- There must be a selection of underlying and different expiries.
- The options chain should work in real-time. As the market data changes, the fields should be recalculated and refreshed on screen without having to reload on the browser.

# 01

## CONCEPTS IMPLEMENTED

# CONCEPT OF IMPLIED VOLATILITY

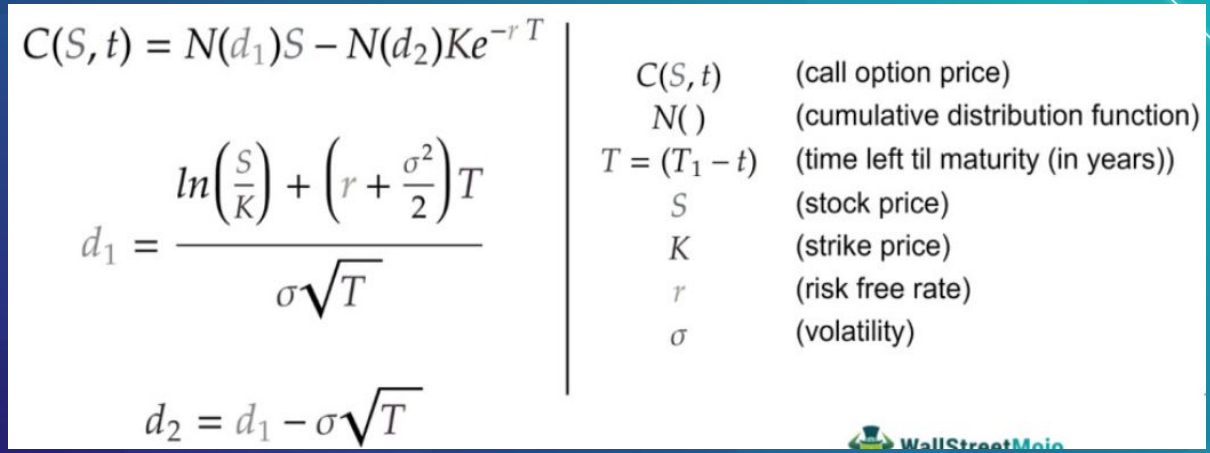
- Implied volatility, it is a significant concept in options trading. Implied volatility represents the market's expectation of the future volatility of the underlying asset, as implied by the prices of the options contracts.

- The objective of finding implied volatility is to help options traders assess the relative expensiveness or cheapness of options. By comparing the implied volatility of different options contracts, traders can identify potential opportunities for buying or selling options based on their volatility expectations.

# IMPLIED VOLATILITY CALCULATION

- First we need to gather the inputs of the Black and Scholes model, such as spot price, strike price, Last Traded Price, the time to expire, and the risk-free rate, option(call or put) from the data received from the feedback-play. Jar file.
- Then input the above data in the Black and Scholes Model.
- Once the above steps are completed, start doing an iterative search by trial and error.

# BLACK SCHOLES FORMULA




- The Black-Scholes formula is a mathematical model used to calculate the theoretical price of options.
- The formula takes into account several variables, including the current stock price, the option's strike price, the time to expiration, the risk-free interest rate, and the volatility of the underlying stock.
- It calculates the fair value of a European-style option, which can only be exercised at expiration.
- Investors can estimate the fair price of an option, which helps them make informed decisions about buying or selling options.

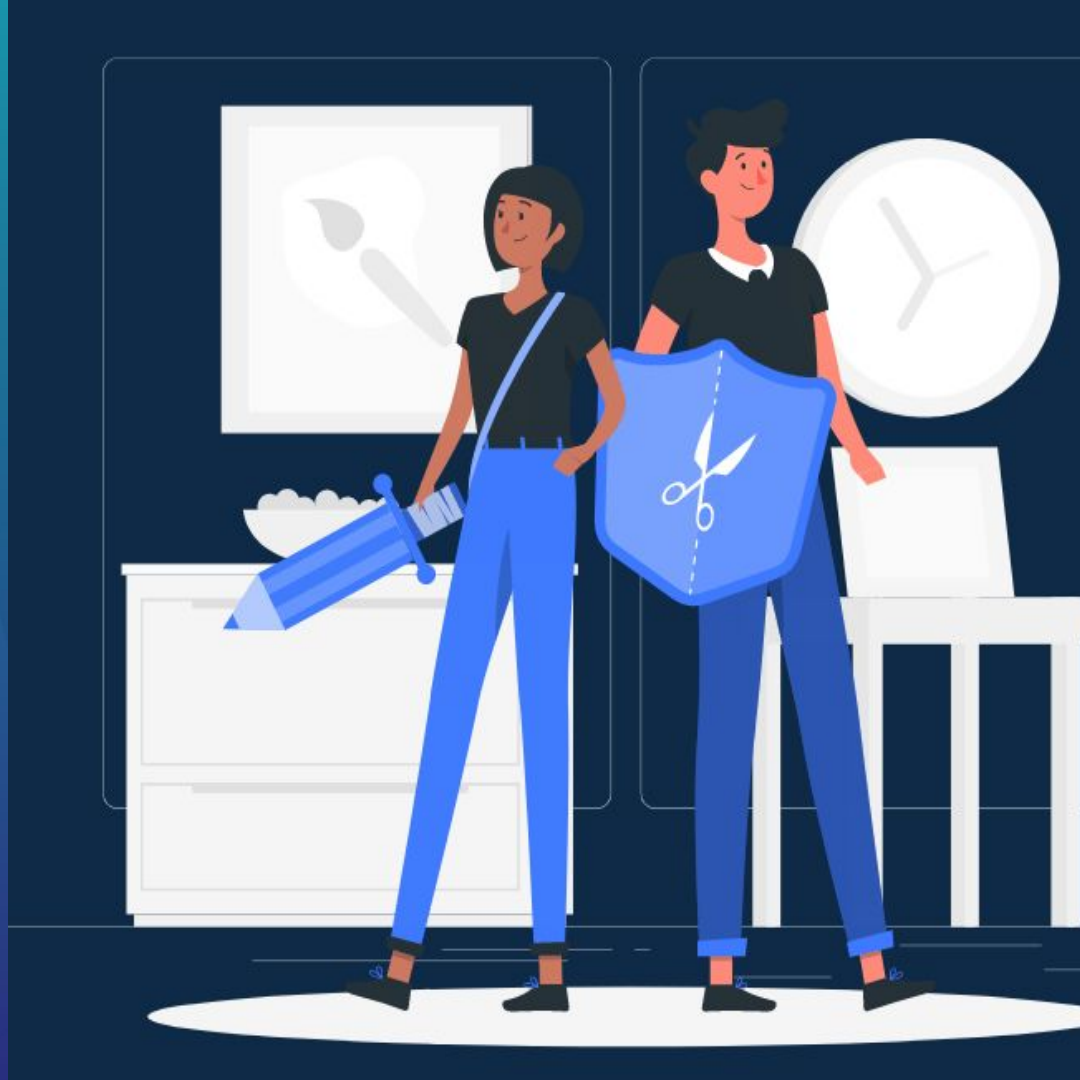
# WEB-SOCKET CONNECTION

- A WebSocket connection is a persistent, bidirectional communication channel between a client and a server, allowing real-time data exchange. It provides efficient and low-latency communication by eliminating the need for repeated requests.
- They offer a more responsive and interactive user experience compared to traditional request-response-based communication protocols like HTTP.

- **Port number for server to backend connection : 9011**
- **Port number for backend to frontend connection : 8000**

# 02

# CODE SNIPPETS



# WEB-SOCKET CONNECTION

## On server.js side

- We have 2 ports port1 and port2.
- Then we establish a TCP/IP connection with the Java server on port2 which is the backend.
- The server on port1 listens on the frontend.
- Then we create a Websocket server which establishes and closes connection.
- When there is a connection between the front-end and the backend, a new client is added to the set.
- When the websocket handles a ‘close’ event, the client is removed from the set.

```
const port1 = 8000;
const port2 = 9011;

const serverClient = net.createConnection({ port: port2 }, () => {
  console.log('Connected to Java server!');
  serverClient.write(Buffer.from([1]));
});
```

```
// Creating a WebSocket server
const wss = new WebSocket.Server({ noServer: true });

wss.on('connection', (ws) => {
    console.log('WebSocket connection between frontend-backend established');


    clients.add(ws);


    ws.on('close', () => {
        console.log('WebSocket between frontend- backend connection closed');

        clients.delete(ws);
    });
});
```

# WEB-SOCKET CONNECTION

```
useEffect(() => {

  const socket = new WebSocket('ws://localhost:8000');

  socket.onopen = () => {
      console.log('WebSocket connection establisheds');
  };

  socket.onmessage = (event) => {
      const newData = event.data;
      const receivedData = JSON.parse(newData)
      fetchData(receivedData);
      setData((prevData) => [...prevData, receivedData]);
  };

  socket.onclose = () => {
      console.log('WebSocket connection closed');
  };

  return () => {
      socket.close();
  };

}, []);
```

## On DataDisplay.js side

- Here we create new websocket on port 8000.
- When the websocket encounters an open event, connection is established and then data is being fetched from the backend port i.e. port 9011.
- Then when the websocket encounters a close event, connection is closed.

# BLACK SCHOLES FORMULA

- A blackScholes function is created which returns the price of the option.

- Here, s = spot Price
  k = strike Price
  t = time to Maturity
  v = volatility
  r = Interest rate
  callPut = option

```
function blackScholes(s, k, t, v, r, callPut)
{
  var price = null;
  var w = (Math.log(s / k) + (r + (Math.pow(v, 2) / 2)) * t) / (v * Math.sqrt(t));
  if(callPut === "call")
  {
    price = s * stdNormCDF(w) - k * Math.pow(Math.E, -1 * r * t) * stdNormCDF(w - v * Math.sqrt(t));
  }
  else // put
  {
    price = k * Math.pow(Math.E, -1 * r * t) * stdNormCDF(v * Math.sqrt(t) - w) - s * stdNormCDF(-w);
  }
  return price;
}
```

# USING BLACK SCHOLES FORMULA TO FIND IMPLIED VOLATILITY

```
function getImpliedVolatility(expectedCost, s, k, t, r, callPut, estimate)
{
  estimate = estimate || .1;
  var low = 0;
  var high = Infinity;

  for(var i = 0; i < 100; i++)
  {
    var actualCost = bs.blackScholes(s, k, t, estimate, r, callPut);
    // comparing the price
    if(expectedCost * 100 == Math.floor(actualCost * 100))
    {
      break;
    }
    else if(actualCost > expectedCost)
    {
      high = estimate;
      estimate = (estimate - low) / 2 + low
    }
    else
    {
      low = estimate;
      estimate = (high - estimate) / 2 + estimate;
      if(!isFinite(estimate)) estimate = low * 2;
    }
  }
  return estimate;
}
```

- The getImpliedVolatility function estimates the implied volatility by doing an iterative search using for loop by trial and error method.

- Where, expectedCost = LTP
  s = spot Price
  k = strike Price
  t = time to Maturity
  v = volatility
  r = Interest rate
  callPut = option

# UPDATING VALUES OF CALL AND PUT IMPLIED VOLATILITY

```
      </tr>
    </thead>
    <tbody>
      {data.map((item) => (

        <tr key={item.id}>
          <td class ="bg-yellow">{item.symbol}</td>
          <td class ="bg-yellow">{item.callVol}</td>
          <td class ="bg-yellow">{item.callImpliedVolatility ? item.callImpliedVolatility * 100 : '-'}</td>
          <td class ="bg-yellow">{item.callLtp}</td>
          <td class ="bg-yellow">{item.callBidQ}</td>
          <td class ="bg-yellow">{item.callBidP}</td>
          <td class ="bg-yellow">{item.callAskP}</td>
          <td class ="bg-yellow">{item.callAskQ}</td>
          <td>{item.strikePrice}</td>
          <td>{item.putBidQ}</td>
          <td>{item.putBidP}</td>
          <td>{item.putAskP}</td>
          <td>{item.putAskQ}</td>
          <td>{item.putLtp}</td>
          <td>{item.putImpliedVolatility ? item.putImpliedVolatility * 100 : '-'}</td>
          <td>{item.putVol}</td>
        </tr>
      )

      )}
    </tbody>
  </table>
);
```

Where,

callImpliedVolatility ⇒ Implied Volatility for call Option

putImpliedVolatility ⇒ Implied Volatility for put Option

# 03

# OUTPUT SNIPPETS

# SNIPPETS- 1. HOW TO RUN

1.To establish connection from the server:

```
hackathon\code>java -classpath feed-play-1.0.jar hackathon.player.Main dataset.csv 9011
```

```
Client added from /0:0:0:0:0:0:0:1
Client disconnected /0:0:0:0:0:0:0:1
Client added from /0:0:0:0:0:0:0:1
```

2. To start the backend:

```
hackathon\code>node server.js
```

3. Starting the frontend:

```
hackathon\code\edel>npm start
```

# 2. DATA RECEIVED FROM BACKEND


```
{
  packetLength: '124',
  tradingSymbol: 'ALLBANKS13JUL2345000PE',
  sequenceNumber: '13767',
  timestamp: '1688375447540',
  lastTradedPrice: '99085',
  lastTradedQuantity: '25',
  volume: '25',
  bidPrice: '100225',
  bidQuantity: '250',
  askPrice: '100530',
  askQuantity: '125',
  openInterest: '10125',
  previousClosePrice: '9975',
  previousOpenInterest: '5800'
}
```


(Before further processing)

# 3. DATA BUFFER RECEIVED FROM SERVER


```
packetLength is  124
<Buffer 7c 00 00 00 41 4c 4c 42 41 4e 4b 53 32 30 4a 55 4c 32 33 34 39 35 30 30 43 45 00 00 00 00 00 00 00 00 3a 1c 00 00 00
 00 00 00 a2 34 10 17 89 01 00 00 ... 65486 more bytes>
```

# 4. DATA SENT TO FRONTEND FROM BACKEND


The data is sent in JSON format from the back-end to the front-end.

# 5. FINAL TABLE WITH CALCULATED IMPLIED VOLATILITY

## NSE Option Chain - Team Data-Sirens

Select Symbol: ALLBANKS Select Expiry: All Expiries

| | | CALLS | | | | | | | | PUTS | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Underlying | Expiry | Volume | IV | LTP | BidQty | BID | ASK | AskQty | STRIKE | BidQty | BID | ASK | AskQty | LTP | IV | Volume |
| ALLBANKS | 27JUL23 | 0 | - | 0 | 0 | 0 | 0 | 0 | 31500 | 0 | 2 | 2 | 0 | 2 | 45.859375 | 0 |
| ALLBANKS | 27JUL23 | 0 | - | 0 | 0 | 0 | 0 | 0 | 33000 | 0 | 5.09 | 5.09 | 0 | 5.09 | 43.759765625 | 0 |
| ALLBANKS | 26OCT23 | 0 | - | 0 | 0 | 0 | 0 | 0 | 33000 | 0 | 38 | 38 | 0 | 38 | 26.092529296875007 | 0 |
| ALLBANKS | 27JUL23 | 0 | - | 0 | 0 | 0 | 0 | 0 | 34000 | 0 | 22 | 22 | 0 | 22 | 47.28759765625 | 0 |
| ALLBANKS | 27JUL23 | 0 | - | 0 | 0 | 0 | 0 | 0 | 34500 | 0 | 3.8 | 3.8 | 0 | 3.8 | 36.484375 | 0 |
| ALLBANKS | 26OCT23 | 0 | - | 0 | 0 | 0 | 0 | 0 | 34500 | 0 | 300 | 300 | 0 | 300 | 34.19097900390625 | 0 |
| ALLBANKS | 27JUL23 | 0 | - | 8865 | 25 | 8865.65 | 8943.35 | 25 | 35000 | 0 | 6.65 | 6.65 | 0 | 6.65 | 36.6748046875 | 0 |
| ALLBANKS | 27JUL23 | 0 | - | 8214.7 | 25 | 8273.2 | 8716.04 | 500 | 35500 | 0 | 8 | 8 | 0 | 8 | 35.390625 | 0 |
| ALLBANKS | 27JUL23 | 0 | - | 7970.7 | 25 | 7878.7 | 7943.2 | 25 | 36000 | 4300 | 9.05 | 9.8 | 925 | 9.3 | 33.947825395614686 | 0 |
| ALLBANKS | 26OCT23 | 0 | - | 0 | 0 | 0 | 0 | 0 | 36000 | 0 | 81.95 | 81.95 | 0 | 81.95 | 22.221984863281254 | 0 |
| ALLBANKS | 06JUL23 | 0 | 163.203125 | 7555.55 | 50 | 7359.95 | 7494.45 | 125 | 36500 | 16900 | 2.4 | 2.45 | 3900 | 2.4 | 98.4765625 | 1700 |
| ALLBANKS | 27JUL23 | 0 | - | 7522.8 | 750 | 7247.9 | 7516.05 | 25 | 36500 | 0 | 9 | 9 | 0 | 9 | 31.757812500000004 | 0 |
| ALLBANKS | 31AUG23 | 0 | - | 0 | 0 | 0 | 0 | 0 | 36500 | 45 | 21.3 | 35 | 15 | 20.25 | 22.672119140624996 | 0 |
| ALLBANKS | 06JUL23 | 0 | 135.44921875000003 | 7012.1 | 25 | 6831.65 | 7115.8 | 750 | 37000 | 225 | 2.54 | 2.6 | 3225 | 2.54 | 92.42187500000001 | 225 |
| ALLBANKS | 27JUL23 | 0 | - | 7035.5 | 50 | 6894.65 | 6979.35 | 25 | 37000 | 0 | 9.75 | 9.75 | 0 | 9.75 | 30.019531250000004 | 0 |

# REDUCING LOAD ON THE FRONTEND:

- Implemented two classes on the backend that handle data processing before sending it to the frontend.
- These classes help in organizing and manipulating the data efficiently, ensuring that the frontend receives optimized and ready-to-use data.

**Classes:**

- Represents a row of processed data.
- Properties: symbol, underlying, expiry, callTimestamp, putTimestamp, strikePrice, callLtp, putLtp, callLtq, putLtq, callVol, putVol, callAskP, putAskP, callAskQ, putAskQ, callBidP, callBidQ, putBidQ, putBidP, callOI, putOI, callPCP, putPCP, callPOI, putPOI.
- Constructor: Initializes the properties.

```
class stock {
    constructor(symbol, ul, ts, ltp) {
        // console.log("constructed stock for symbol: ", symbol);
        this.symbol = symbol;
        this.underlying = ul;
        this.timestamp = ts;
        this.ltp = ltp;
    }
```

# DATA UPDATION WITHOUT RELOADING

- Method: update(): Updates the row's properties if the provided timestamps are greater than the existing ones.

**Stock:**

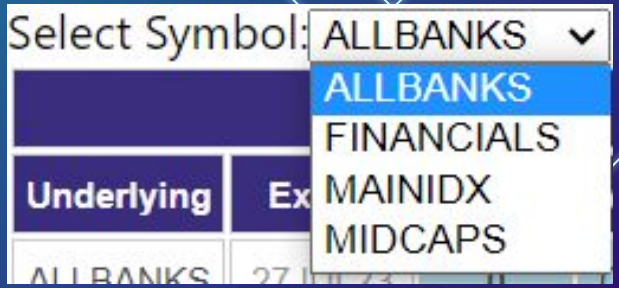
- Represents a stock entry.
- Properties: symbol, underlying, timestamp, ltp.
- Constructor: Initializes the properties.
- Method: update(): Updates the stock's properties if the provided timestamp is greater than the existing one.

```
    update(ts,ltp) {

        if(BigInt(this.timestamp) < BigInt(ts)){
            console.log("updAted stock for symbol: ", this.symbol);
            this.timestamp = ts;
            this.ltp = ltp;
        }
    }
}
```

# SELECTION OF UNDERLYING AND DIFFERENT EXPIRIES

Local Data Storage: All the required data is stored on the frontend, eliminating the need for conventional GET requests to the backend for filtering.



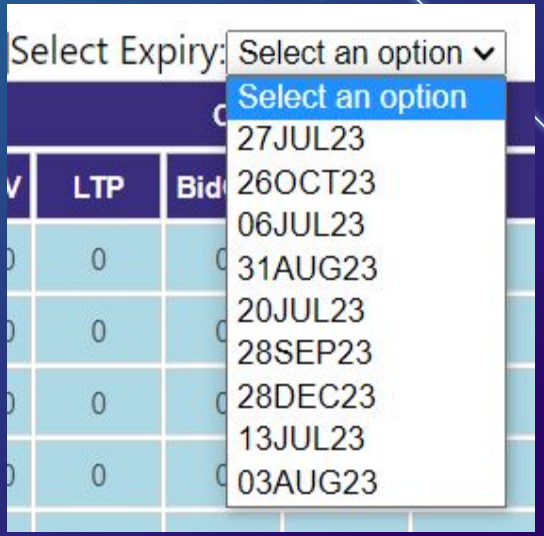

**Symbol Filtering:**

- A map (stockPrices) is used to store the available symbols as keys.
- The dropdown menu is populated dynamically by iterating over the keys of stockPrices and generating <option> elements.
- When a symbol is selected from the dropdown, it triggers the handleDropdownChange function

# SELECTION OF UNDERLYING AND DIFFERENT EXPIRIES

**Expiry Filtering:**

- Another map (expiry) is used to store the available expiry dates for each selected symbol.
- The dropdown menu for expiry is populated based on the selected symbol by retrieving the corresponding values from expiry using selectedValue.
- The values are converted into an array using Array.from() and then mapped to generate <option> elements.
- The selected expiry value is captured by the handleDropdownChangeExp function.



# HIGHLIGHTING THE "IN THE MONEY" OPTIONS AND "OUT OF MONEY" OPTIONS DIFFERENTLY

Function getCellClassNameCalls(strikePrice):

- It compares the strikePrice with the strikeThreshold value.
- If the strikePrice is less than the strikeThreshold, it returns the CSS class, which applies a left color (lightblue) to the cell, otherwise it returns an empty string, indicating no special color should be applied to the cell.

Function getCellClassNamePuts(strikePrice):

- It compares the strikePrice with the strikeThreshold value.
- If the strikePrice is greater than the strikeThreshold, it returns the CSS class, which applies a color (lightblue) to the cell, otherwise it returns an empty string, indicating no special color should be applied to the cell.

**NSE Option Chain - Team Data-Sirens**

Select Symbol: MAINIDX Select Expiry: 27JUN24

| | | CALLS | | | | | | | | PUTS | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Underlying | Expiry | Volume | IV | LTP | BidQty | BID | ASK | AskQty | STRIKE | BidQty | BID | ASK | AskQty | LTP | IV | Volume |
| MAINIDX | 27JUN24 | 0 | - | 0 | 0 | 0 | 0 | 0 | 14000 | 0 | 41.05 | 41.05 | 0 | 41.05 | - | 0 |
| MAINIDX | 27JUN24 | 0 | - | 0 | 0 | 0 | 0 | 0 | 15000 | 0 | 72.25 | 72.25 | 0 | 72.25 | - | 0 |
| MAINIDX | 27JUN24 | 0 | - | 0 | 0 | 0 | 0 | 0 | 16000 | 0 | 180 | 180 | 0 | 180 | - | 0 |
| MAINIDX | 27JUN24 | 0 | - | 0 | 0 | 0 | 0 | 0 | 17000 | 0 | 340.8 | 340.8 | 0 | 340.8 | - | 0 |
| MAINIDX | 27JUN24 | 0 | 16.964721679687504 | 2001 | 0 | 2001 | 2001 | 0 | 18000 | 0 | 475 | 475 | 0 | 475 | - | 0 |
| MAINIDX | 27JUN24 | 0 | 15.379943847656252 | 1601 | 0 | 1601 | 1601 | 0 | 18500 | 0 | 0 | 0 | 0 | 0 | - | 0 |
| MAINIDX | 27JUN24 | 0 | 13.518218994140629 | 1200.05 | 0 | 1200.05 | 1200.05 | 0 | 19000 | 0 | 780 | 780 | 0 | 780 | - | 0 |
| MAINIDX | 27JUN24 | 0 | 7.3609924316406286 | 320 | 0 | 320 | 320 | 0 | 20000 | 0 | 1500 | 1500 | 0 | 1500 | - | 0 |

# 04

## CONCLUSION

# CONCLUSION

- We implemented an options chain website which displays the option chain screen, processes the market data which was provided by the market data stream over TCP/IP and calculates the Implied Volatility.
- The options chain works in real-time. As the market data changes, the fields are recalculated and refreshed on screen without having to reload on the browser.
- It also has a provision of symbol filtering, expiry date filtering and highlights in the money options and out of the money options differently which provides ease of use.

# THANK YOU!