

Solution Model:

-Part 1:

- Hercules emulator for emulating Mainframe on Windows, running MVS3.8 OS and Vista TN3270 command line.
- I found more resources for running Cobol, so generating 5 $\sin^2 x$ values, where $x = [1,5]$ from Cobol program.
- Send these values to the frontend.

-Part 2:

- For N-tier architecture, I used 2 tier Model-Controller architecture using NodeJS and SQLite3 using JavaScript, a 3rd Generation language.
- I stored values of $\cos x$ from $[1,5]$ (model) and upon request by the frontend for individual values, extract it from the database and square it to get $\cos^2 x$. Send this value to the frontend.

-Part 3:

- I used React Native to create an app, which I ran on Android emulator.
- For microservices, I am using websockets to connect to the node server and planned to use FTP for data transmission to Mainframe.
- I designed it as such to have a button, upon pressing which:
 - i) It sends data 1,2,3,4,5 on a channel to NodeJS server at an interval of 1 second each. After each data packet emit, it receives the $\cos^2 x$ value for 1,2,3,4 &5, which it displays.
 - ii) Then it sends request to mainframe, and all the values are received at once. Display these on screen.
 - iii) Sum the values for $x=1,2,3,4$ &5. Averaging the result, we get the desired value, 1.

The repository has been hosted on GitHub at <https://github.com/zed-shawn/EDA>

1. Code Blocks:

a) COBOL code with JCL:

```
000001 //SINEVAL JOB (COBOL),
000002 //          'Ali Zeeshan',MSGCLASS=H,
000003 //          CLASS=A,
000004 //          REGION=8M,TIME=1440,
000005 //          MSGLEVEL=(1,1)
000006 //STEP01 EXEC PROG=SINVAL
000007 //STEPLIB DD DSN=CUST.PR.LOADLIB,DISP=SHR
000008 //SYSPRINT DD SYSOUT=*
000009 //SYSOUT DD SYSOUT=A
000010 //SYSIN DD DUMMY

000014      30 IDENTIFICATION DIVISION.
000017      40 PROGRAM-ID. 'SINVAL'
000018      50 ***

000020      70 DATA DIVISION.
000021      80 WORKING-STORAGE SECTION.
000022      90          01 VAL1 PIC S9(2)V9(2) VALUE 1.00.
000023     100          01 VAL2 PIC S9(2)V9(2) VALUE 2.00.
000024     110          01 VAL3 PIC S9(2)V9(2) VALUE 3.00.
000025     120          01 VAL4 PIC S9(2)V9(2) VALUE 4.00.
000026     130          01 VAL5 PIC S9(2)V9(2) VALUE 5.00.
000027     140          01 RES1 PIC S9(1)V9(10).
000028     150          01 RES2 PIC S9(1)V9(10).
000029     160          01 RES3 PIC S9(1)V9(10).
000030     170          01 RES4 PIC S9(1)V9(10).
000031     180          01 RES5 PIC S9(1)V9(10).
000032     190 ***

000033     200 PROCEDURE DIVISION.
000034     210          COMPUTE RES1= FUNCTION SIN(VAL1)
000035     220          MULTIPLY RES1 BY RES1.
000036     230          DISPLAY '1,'RES1.
000037     240
000038     250          COMPUTE RES2= FUNCTION SIN(VAL2)
000039     260          MULTIPLY RES2 BY RES2.
000040     270          DISPLAY '2,'RES2.
000041     280
000042     290          COMPUTE RES3= FUNCTION SIN(VAL3)
000043     300          MULTIPLY RES3 BY RES3.
000044     310          DISPLAY '3,'RES3.
000045     320
000046     330          COMPUTE RES4= FUNCTION SIN(VAL4)
000047     340          MULTIPLY RES4 BY RES4.
000048     350          DISPLAY '4,'RES4.
000049     360
000050     370          COMPUTE RES5= FUNCTION SIN(VAL5)
000051     380          MULTIPLY RES5 BY RES5.
```

```
000052    390      DISPLAY '5','RES5.
000054    410      STOP RUN.
```

b) NodeJS code:

```
const express = require("express");
const socket = require("socket.io");
const sqlite3 = require("sqlite3").verbose();

const PORT = 5000;
const app = express();
const server = app.listen(PORT, function () {
  console.log(`Listening on port ${PORT}`);
  console.log(`http://localhost:${PORT}`);
});

app.use(express.static("public"));

app.get("/", function (req, res) {
  res.render("index", {});
});

// Socket setup
const io = socket(server);

io.on("connection", function (socket) {
  console.log("Made socket connection");

  socket.on("values", (data) => {
    console.log(data);
    searchAndEmit(data);
  });
});

let db = new sqlite3.Database("./cos_val.db", sqlite3.OPEN_READONLY, (err) => {
  if (err) {
    console.error(err.message);
  }
  console.log("Connected to the the database.");
});

const searchAndEmit = (number) => {
  let query = `SELECT value FROM Cos WHERE key = ${number}`;

  db.all(query, [], (err, rows) => {
    if (err) {
      throw err;
    }
    rows.forEach((row) => {
      console.log(row.value);
      sqVal= Math.pow(row.value, 2)
      var appendData = {
```

```

        number: number,
        value: sqVal,
    });

    let sendData = JSON.stringify(appendData);
    io.emit("values", sendData);
  });
});
};

```

c) React Native code:

```

import React, { useEffect, useState } from "react";
import { StyleSheet, Text, View, Button, FlatList } from "react-native";
import io from "socket.io-client";

const socket = io("http://192.168.0.101:5000", {
  transports: ["websocket"],
  reconnection: false,
});

try {
  socket.on("connect", () => {
    if (socket.connected === true) {
      console.log("Connected to Node Server");
    } // true
  });
} catch (error) {
  console.log("Could not connect", error);
}

class DataPacket {
  constructor(number, value) {
    this.number = number;
    this.value = value;
  }
}

export default function App() {
  const [cosineValue, setCosineValue] = useState([
    {
      number: "Value of X",
      value: "Value of cos^2 x",
    },
  ]);

  const pressHandler = () => {
    console.log("press handler");
    setCosineValue(() => [
      {
        number: "Value of X",
        value: "Value of cos^2 x",
      },
    ],
  );
}

```

```

]);
socket.emit("values", 1);
setTimeout(() => {
  socket.emit("values", 2);
}, 300);
setTimeout(() => {
  socket.emit("values", 3);
}, 600);
setTimeout(() => {
  socket.emit("values", 4);
}, 900);
setTimeout(() => {
  socket.emit("values", 5);
}, 1200);
};

useEffect(() => {
  socket.on("values", (data) => {
    console.log(data);
    const receivedMessage = JSON.parse(data);
    const newDataPacket = new DataPacket(
      receivedMessage.number.toString(),
      receivedMessage.value.toString()
    );
    setCosineValue((cosineValue) => [...cosineValue, newDataPacket]);
    console.log(newDataPacket);
  });
}, []);

const renderValues = (itemData) => {
  return (
    <View style={styles.renderTile}>
      <View style={styles.tileComponent}>
        <Text>{itemData.item.number}</Text>
      </View>
      <View style={styles.tileComponent}>
        <Text>{itemData.item.value}</Text>
      </View>
    </View>
  );
};

return (
  <View style={styles.container}>
    <View style={styles.headerBox}>
      <Text style={styles.headingText}>
        Testing the expression  $\sin^2 x + \cos^2 x = 1$ 
      </Text>
      <Button title="Initiate the Test" color="red" onPress={pressHandler} />
    </View>
    <View>
      <FlatList
        renderItem={renderValues}

```

```

    data={cosineValue}
    keyExtractor={(item, index) => item.number}
  />
</View>
</View>
);
}

```

2. A) Experience with 3 gen of code:

Mainframe: This was the most difficult by far. It took me a lot of time to realize what Mainframe and MVS are, and to setup and navigate them. Working with COBOL and JCL was also new. But the most difficult to plan was how to receive and send data from the outside world, as the documentation for mainframes are very rare, and often in languages of the expert. I had planned to use FTP to communicate with the frontend, but unfortunately, I couldn't reach that stage as I had some error in my JCL header of the code, which caused runtime failure.

JavaScript: I have been working with JS for some time now, so it was already familiar. For things which are new, like FTP, documentation was easy to find and use.

I did not use any other language, as JavaScript is from the same generation as Java (which was suggested to use), and the frontend part was also done using JavaScript.

B) Differences between the systems:

Mainframe is a legacy system, so naturally it is hard to get started with initially. However, I was thoroughly impressed by its resource management efficiency, and the failsafe measures implemented on mainframes. It can be seen why mainframes are still used by corporate giants.

On the other hand, node and react are modern frameworks, which can be implemented on any scale. However, I feel the uptime and reliability of data is higher in the case of mainframes, as they are precisely designed for those applications.

C) Difficulties faced & possible solutions:

1. Error with JCL:

- I was not able to execute COBOL program in mainframe, as I faced error with JCL header. I compiled the COBOL code on windows, and it compiled with correct values.

Solution:

Better understanding of JCL, as it was mostly a syntax or library error.

2. Connectivity with Mainframe:

- It was a major confusion as to how to network data with mainframe. I was able to find two methods- FTP & virtual Ethernet ports. I planned to go via the FTP route but couldn't reach the stage. I am unsure if I would've been able to implement the connection.

Solution:

More worktime with mainframe, help from mainframe forums.

Screenshots:

```
REVEDIT SYS2.JCLLIB(PRIMCOB3) - 1.04 Columns 00001 00072
Command ==> Scroll ==> CS
***** ****Zap****Autosave***** Top of Data *****
000001 //SINEVAL JOB (COBOL),
000002 // 'Ali Zeeshan',MSGCLASS=H,
000003 // CLASS=A,
000004 // REGION=8M,TIME=1440,
000005 // MSGLEVEL=(1,1)
000006 //STEP01 EXEC PROG=SINVAL
000007 //STEPLIB DD DSN=CUST.PR.LOADLIB,DISP=SHR
000008 //SYSPRINT DD SYSOUT=*
000009 //SYSOUT DD SYSOUT=A
000010 //SYSIN DD DUMMY
000011 //
000012 //
000013 //
000014 10 ***
000015 20 ***
000016 30 IDENTIFICATION DIVISION.
000017 40 PROGRAM-ID. 'SINVAL'
000018 50 ***
000019 60 ***
000020 70 DATA DIVISION.
000021 80 WORKING-STORAGE SECTION.
000022 90 01 VAL1 PIC S9(2)V9(2) VALUE 1.00.
000023 100 01 VAL2 PIC S9(2)V9(2) VALUE 2.00.
000024 110 01 VAL3 PIC S9(2)V9(2) VALUE 3.00.
000025 120 01 VAL4 PIC S9(2)V9(2) VALUE 4.00.
000026 130 01 VAL5 PIC S9(2)V9(2) VALUE 5.00.
000027 140 01 RES1 PIC S9(1)V9(10).
000028 150 01 RES2 PIC S9(1)V9(10).
000029 160 01 RES3 PIC S9(1)V9(10).
000030 170 01 RES4 PIC S9(1)V9(10).
000031 180 01 RES5 PIC S9(1)V9(10).
000032 190 ***
000033 200 PROCEDURE DIVISION.
000034 210 COMPUTE RES1= FUNCTION SIN(VAL1)
000035 220 MULTIPLY RES1 BY RES1.
000036 230 DISPLAY '1',RES1.
000037 240
000038 250 COMPUTE RES2= FUNCTION SIN(VAL2)
000039 260 MULTIPLY RES2 BY RES2.
000040 270 DISPLAY '2',RES2.
```

Cobol program on Vista 3270

```
HERCULES - System Status: GREEN
08.39.22 JOB 4 $HASP373 SINEVAL STARTED - INIT 1 - CLASS A - SYS TK4-
08.39.22 JOB 4 IEF403I SINEVAL - STARTED - TIME=08.39.22
08.39.22 JOB 4 IEF453I SINEVAL - JOB FAILED - JCL ERROR - TIME=08.39.22
08.39.22 JOB 4 $HASP395 SINEVAL ENDED
08.39.22 $HASP309 INIT 1 INACTIVE ***** C=A
08.40.33 STC 164 $HASP150 MF1 ON PRINTER1 216 LINES
08.40.33 STC 164 IRB101I MF/1 REPORT AVAILABLE FOR PRINTING
08.40.33 $HASP160 PRINTER1 INACTIVE - CLASS=A
08.44.16 JOB 5 $HASP100 SINEVAL ON INTRDR Ali Zeeshan
08.44.16 JOB 5 $HASP373 SINEVAL STARTED - INIT 1 - CLASS A - SYS TK4-
08.44.16 JOB 5 IEF403I SINEVAL - STARTED - TIME=08.44.16
08.44.16 JOB 5 IEF453I SINEVAL - JOB FAILED - JCL ERROR - TIME=08.44.16
08.44.16 JOB 5 $HASP395 SINEVAL ENDED
08.44.16 $HASP309 INIT 1 INACTIVE ***** C=A
08.46.11 JOB 6 $HASP100 SINEVAL ON INTRDR Ali Zeeshan
08.46.11 JOB 6 $HASP373 SINEVAL STARTED - INIT 1 - CLASS A - SYS TK4-
08.46.11 JOB 6 IEF403I SINEVAL - STARTED - TIME=08.46.11
08.46.11 JOB 6 IEF453I SINEVAL - JOB FAILED - JCL ERROR - TIME=08.46.11
08.46.11 JOB 6 $HASP395 SINEVAL ENDED
08.46.11 $HASP309 INIT 1 INACTIVE ***** C=A
08.55.33 STC 164 $HASP150 MF1 ON PRINTER1 216 LINES
08.55.33 STC 164 IRB101I MF/1 REPORT AVAILABLE FOR PRINTING
08.55.33 $HASP160 PRINTER1 INACTIVE - CLASS=A
09.03.15 JOB 7 $HASP100 SINEVAL ON INTRDR Ali Zeeshan
09.03.15 JOB 7 IEF452I SINEVAL JOB NOT RUN - JCL ERROR
09.10.33 STC 164 $HASP150 MF1 ON PRINTER1 216 LINES
09.10.33 STC 164 IRB101I MF/1 REPORT AVAILABLE FOR PRINTING
09.10.33 $HASP160 PRINTER1 INACTIVE - CLASS=A
09.10.35 $HASP000 OK
09.25.33 STC 164 $HASP150 MF1 ON PRINTER1 216 LINES
09.25.33 STC 164 IRB101I MF/1 REPORT AVAILABLE FOR PRINTING
09.25.33 $HASP160 PRINTER1 INACTIVE - CLASS=A
herc ==>>>>
CP00 PSW=070E000000000000 24.W..... instcnt 906,282,808; mips 0.080; I/O
```

Runtime error log on MVS

```
app.js > ...
1 const express = require("express");
2 const socket = require("socket.io");
3 const sqlite3 = require("sqlite3").verbose();

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS F:\EDA\Assignment1\node> node app.js
Listening on port 5000
http://localhost:5000
Connected to the the database.
Made socket connection
1
0.540302306
2
-0.416146837
3
-0.989992497
4
-0.653643621
5
-0.283662186
[]
```

NodeJS server, emitting values of $\cos^2 x$

```
Appjs > ...
1 import React, { useEffect, useState } from "react";
2 import { StyleSheet, Text, View, Button, FlatList } from "react-native";
3 import io from "socket.io-client";

Trying to open the project on Android...
Opening on Android device
Expo Press ? to show a list of all available commands.
Finished building JavaScript bundle in 2408ms.
Running application on AOSP on IA Emulator.
Connected to Node Server
press handler
{"number":1,"value":0.29192658186891757}
DataPacket {
  "number": "1",
  "value": "0.29192658186891757",
}
{"number":2,"value":0.17317818994510456}
DataPacket {
  "number": "2",
  "value": "0.17317818994510456",
}
{"number":3,"value":0.9800851441162951}
DataPacket {
  "number": "3",
  "value": "0.9800851441162951",
}
{"number":4,"value":0.42724998327399155}
DataPacket {
  "number": "4",
  "value": "0.42724998327399155",
}
{"number":5,"value":0.08046423576629859}
DataPacket {
  "number": "5",
  "value": "0.08046423576629859",
}
}
```

Value of X	Value of $\cos^2 x$
1	0.29192658186891757
2	0.17317818994510456
3	0.9800851441162951
4	0.42724998327399155
5	0.08046423576629859

React Native log, with app running on Android emulator