Summer Fellowship Report

On

Extending XCos on Cloud platform

Submitted by

Shantanu Tripathi

Under the guidance of

Mr. Sunil Shetye

IIT Bombay

August 17, 2022

Acknowledgment

I am grateful to everyone who has helped me in completing this project successfully. I would like to thank **Mr.Sunil Shetye** and the entire FOSSEE team at IIT Bombay, for providing me with this wonderful opportunity to work on this project and to would like to thank him for being a constant support and pointing me out in the right direction. His insights have been a key in helping me complete the project well within the stipulated deadline. I'd also like to thank him for his guidance and invaluable suggestions. Last but not the least, I truly appreciate all our fellow interns' efforts to help me out, with the problems we faced, and making my internship experience a memorable and an enjoyable one.

Contents

Chapter 1

Introduction

1.1 Objective

Xcos is a graphical editor to design hybrid dynamical systems models. Models can be designed, loaded, saved, compiled and simulated. Ergonomic and efficient solution for industrial and academics needs, Xcos provides functionalities for modeling of mechanical systems, hydraulic circuits, control systems, etc. Xcos-On-Cloud ports these core functionalities to a browser only version of Xcos. Shifting Xcos to browser increases its industrial and educational use as user will be able to use it anytime from anywhere without going into the trouble of installing a software. The aim of this project include rendering of charts using HighCharts and using EventSource for handling various events.

1.2 Approach

Using EventSource API for handling of events and using condtitional cases to handle each event. HighChart library to display of graphs using data from the Scilab at backend. Serializing the data into React state and using UseEffect to update charts. Defining the layout of the charts using conditionals and React reflow for constant update of layout. Google sign in for user authentication.

Chapter 2

Technical Specification

2.1 EventSource

An EventSource instance opens a persistent connection to an HTTP server, which sends events in text/event-stream format. Once the connection is opened, incoming messages from the server are delivered to your code in the form of events. Unlike WebSockets, server-sent events are unidirectional; that is, data messages are delivered in one direction, from the server to the client (such as a user's web browser). The connection remains open until closed by calling `EventSource.close()`. [1]

2.2 HighCharts

Highcharts is a pure JavaScript based charting library meant to enhance web applications by adding interactive charting capability. Highcharts provides a wide variety of charts. For example, line charts, spline charts, area charts, bar charts, pie charts and so on. [5]

2.3 Google Authentication (OAuth)

Google OAuth 2.0 allows users to share specific data with an application while keeping their usernames, passwords, and other information private.This OAuth 2.0 flow is called the implicit grant flow. It is designed for applications that access APIs only while the user is present at the application. These applications are not able to store confidential information. [4]

Chapter 3

Environment Setup

- Switch to no-backend branch (no-backend-windows branch for windows), this runs the server without installation of Scilab. Then we can work only on the frontend part.

- For the backend part to work, install python virtual environment and install all the dependencies once. Later on, we can just run the pyenv server every time and input the command **python manage.py runserver** to run the server in the blocks directory.

- Change the directory to eda-frontend and install the node modulese using the command **npm install** and then run the server using the command **npm start**

- Install **nginx** to run both the servers python and node on the same port. Configure the server section in **nginx config** file to do this.

Chapter 4

Events Received

The instruction event will be of three types of data. Depending on the data received, take the following actions:

```
                                                        # Comments
data: addChart id=uniqueId type=chartType other-parameters   # add (render) a chart to the page
data: addData id=chartId other-parameters                    # add (render) a point to that chart
data: reset                                                  # clear the page. remove all added charts
```

The `addChart` instruction will lead to adding of a new graph on the page. The `addData` instruction will add a point to the specific chart identified by chartId which is unique for all the charts. The `Reset` instruction will clear the page and remove all the added charts but won't close the connection.

```
event: instruction
data: addChart id=1 type=bar xMin=0 xMax=10 yMin=-30 yMax=30

event: instruction
data: addChart id=2 type=curve xMin=0 xMax=20 yMin=-10 yMax=10

event: instruction
data: addData id=1 x=2 y=6

event: instruction
data: addData id=2 x=3 y=1

event: instruction
data: reset

event: instruction
data: addChart id=1 type=curve xMin=0 xMax=10 yMin=-30 yMax=30

event: done
data: none
```

For the above code, the final page will have one charts containing no points.

Desired Layout

1. Layout of Charts

Add 1st chart (as before).

```
|        Chart 1        |              # Add first chart of full width
```

Add 2nd chart in the same row.

```
|  Chart 1  |  Chart 2  |              # Each chart will be 50% of page width
```

When 3th chart is to be added, change to 2-row layout.

```
|  Chart 1  |  Chart 2  |
|        Chart 3        |              # Add third chart of full width on 2nd row
```

Add 4th chart in the same row.

```
|  Chart 1  |  Chart 2  |
|  Chart 3  |  Chart 4  |              # Each chart will be 50% of page width
```

When 5th chart is to be added, change to 3-row layout.

```
|  Chart 1  |  Chart 2  |
|  Chart 3  |  Chart 4  |
|        Chart 5        |              # Add fifth chart of full width on 3nd row
```

Chapter 5

Event Handling

```
const sse = new EventSource('/api/' + streamingUrl, { withCredentials: true })
sse.addEventListener('log', e => {
  ++loglines

  const data = e.data.split(' ')
```

The Variable `sse` is made to store the event received from the backend

```
  const block = parseInt(data[0])
  const figureId = (block === 2) ? data[4] : data[2] // For CMSCOPE
  let noOfGraph
  if (block === 5 || block === 10) { // For 3D-SCOPE blocks
    noOfGraph = data[11]
  } else if (block === 11) { // For BARXY
    noOfGraph = data[12]
  } else {
    noOfGraph = data[10]
  }
```

Extracting the data from the variable `data` and using conditional cases for the identification of the specific case.

Setting the parameters required by HighCharts using conditionals.

```
  // process data for 3D-SCOPE blocks
  createNewChart3d(figureId, noOfGraph, xmin, xmax, ymin, ymax, zmin, zmax, typeChart, titleText, alpha, theta)
} else if (block < 5 || block === 9 || block === 11 || block === 12 || block === 23) {
  // sink block is not CSCOPXY
  createNewChart(figureId, noOfGraph, xmin, xmax, ymin, ymax, typeChart, titleText, colorAxis)
  range.current[chartIdList.current[figureId]] = parseFloat(xmax)
}
```

Function call for specific cases and passing the data Received.

```
    chartIdList.current[id] = chartIdCount.current
    const datapoint = {
      datapointId: id,
      datapointType: typeChart,
      datapointTitle: titleText,
      datapointXMin: xmin,
      datapointYMin: ymin,
      datapointXMax: xmax,
      datapointYMax: ymax,
      datapointPointRange: pointRange,
      datapointLineWidth: lineWidth,
      datapointPointWidth: pointWidth,
      datapointDataClasses: colorAxis
    }
    datapointsRef.current[chartIdCount.current] = datapoint
    chartIdCount.current = chartIdCount.current + 1
    setNoOfGraphs(nog => nog + 1)
  }
```

Variable `datapoint` to store and render the chart using these values.

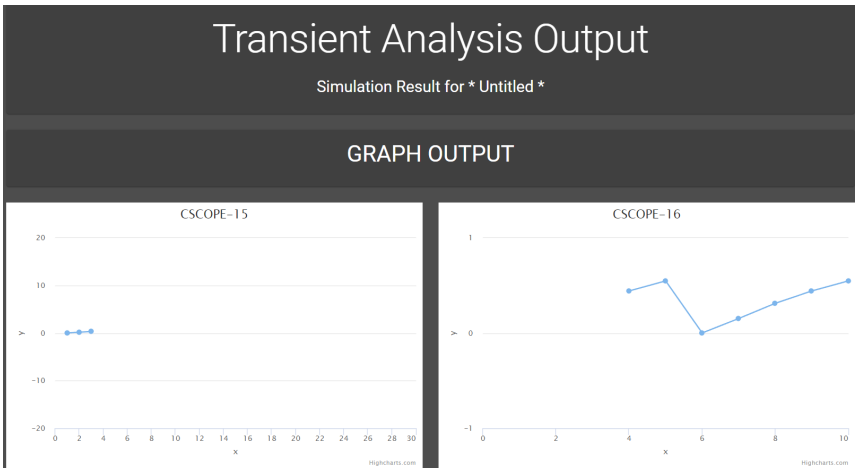
Chapter 6

Layout Handling

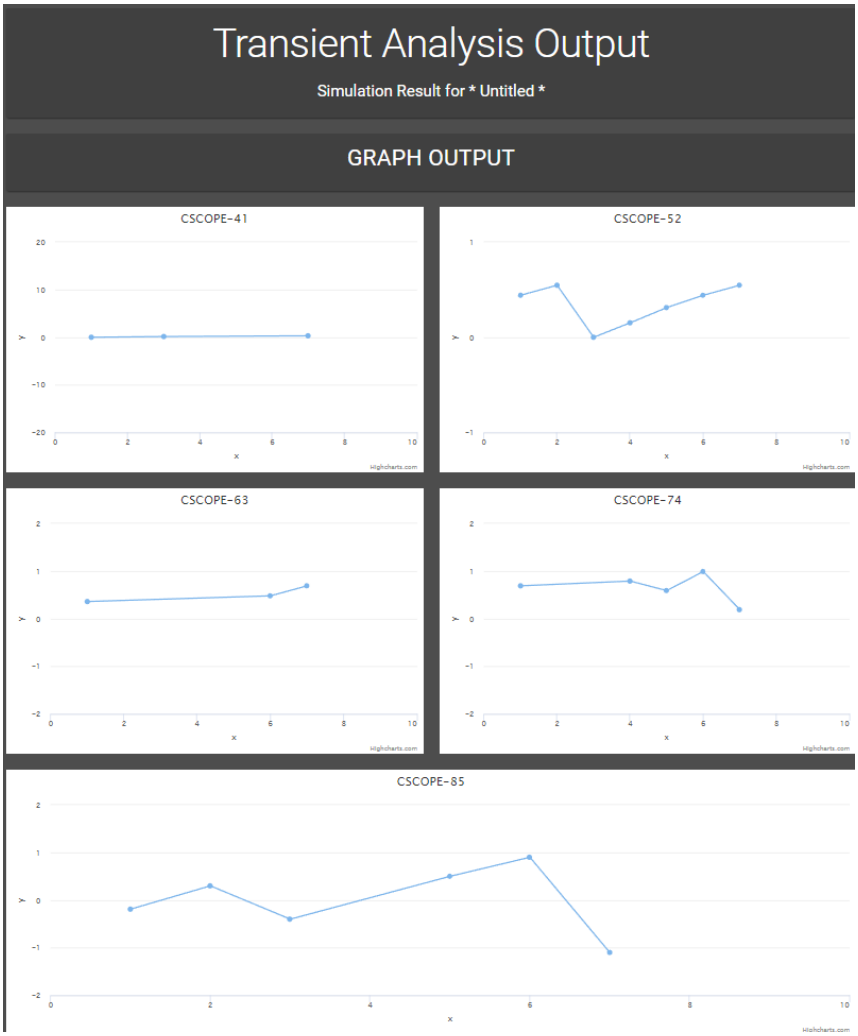
```
        <Grid item xs={12} sm={12}>
          <Paper className={classes.paper}>
            <Typography variant='h4' align='center' gutterBottom>
              GRAPH OUTPUT
            </Typography>
          </Paper>
        </Grid>
        {
          noOfGraphs !== 0
            ? datapointsRef.current.map((element, i, arr) => {
              const gridSize = (arr.length - 1 === i && i % 2 === 0) ? 12 : 6
              return (
                <Grid item key={i} xs={gridSize}>
                  <Graph
                    ref={el => { graphsRef.current[i] = el }}
                    datapoint={element}
                  />
                </Grid>
              )
            })
            : <div />
        }
        </>
      : (result.isGraph === 'false') ? <span>{typography1}</span> : <span />
  }
```

Using conditions and number of graphs to specify the position and get the desired layout of charts.

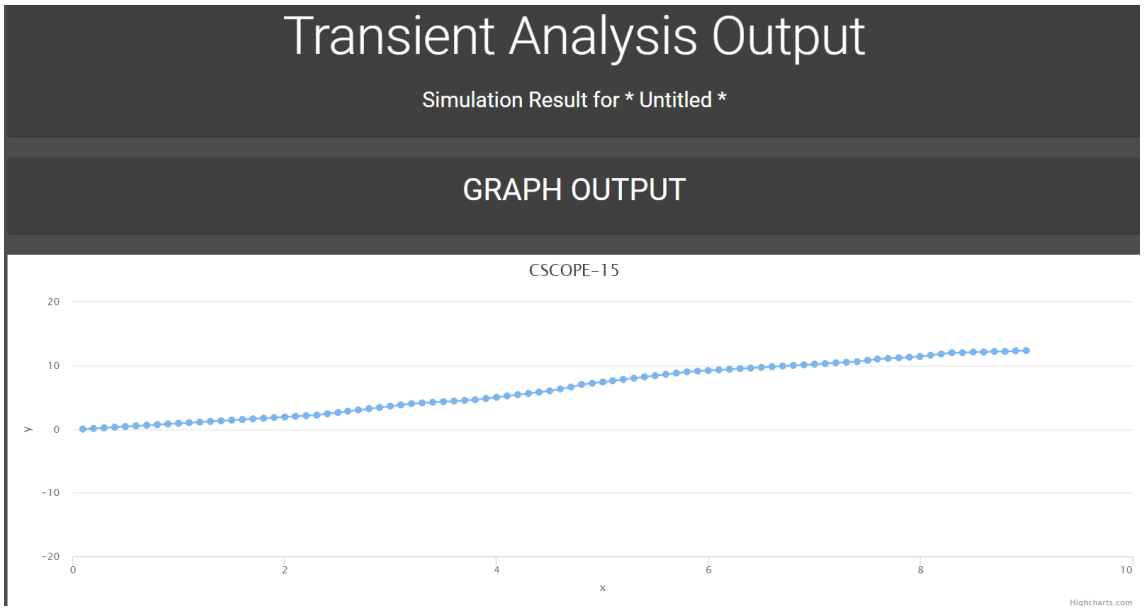
Generalization Of Graphs



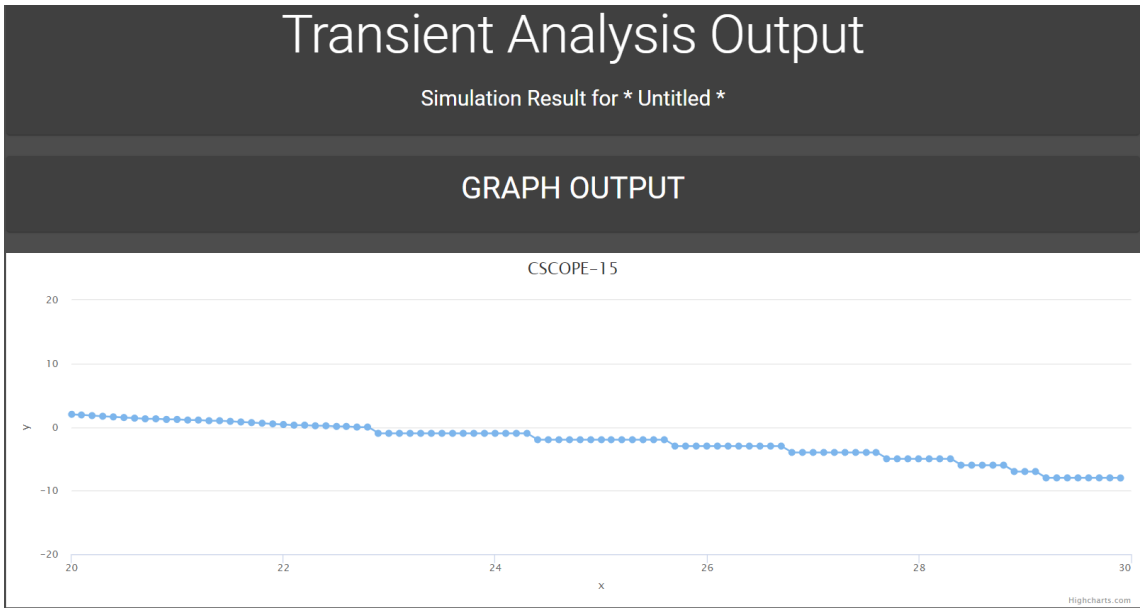
Earlier the code was working for only two number of graphs. Generalizing the incoming Events and mapping them using the `datapointId` has displayed n-number of graphs as shown below.



Shifting Of Graph

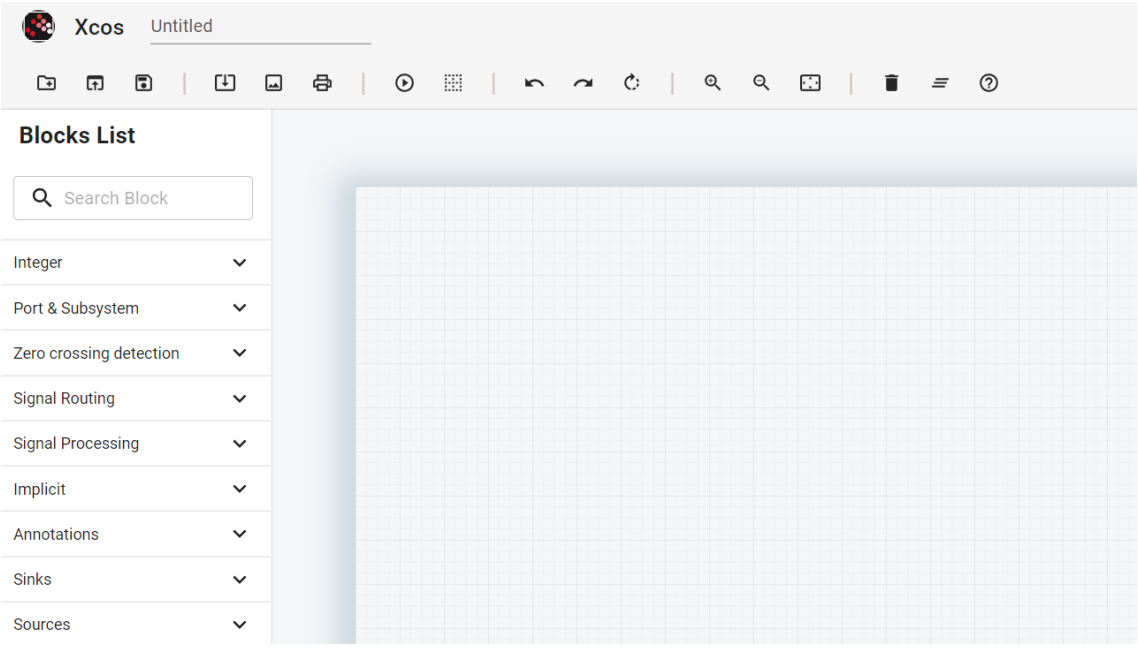


The initial graph with no-shifting of x-axis regardless of receiving points greater than 10

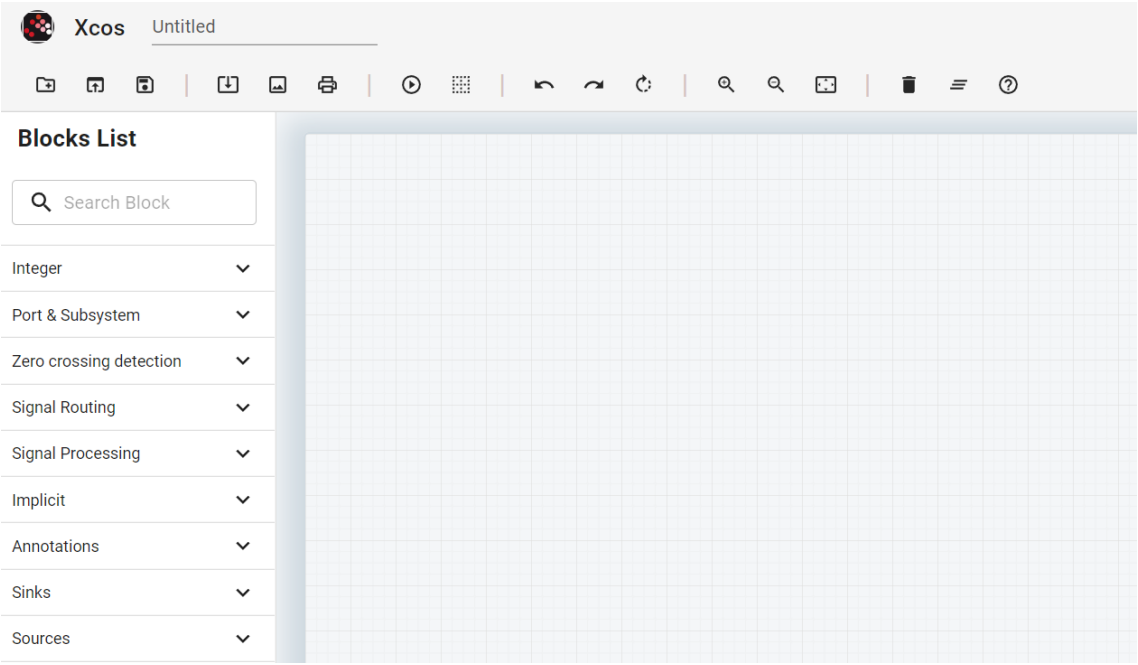


Shifting of graph on receiving point greater than current x-max.

Canvas Fix

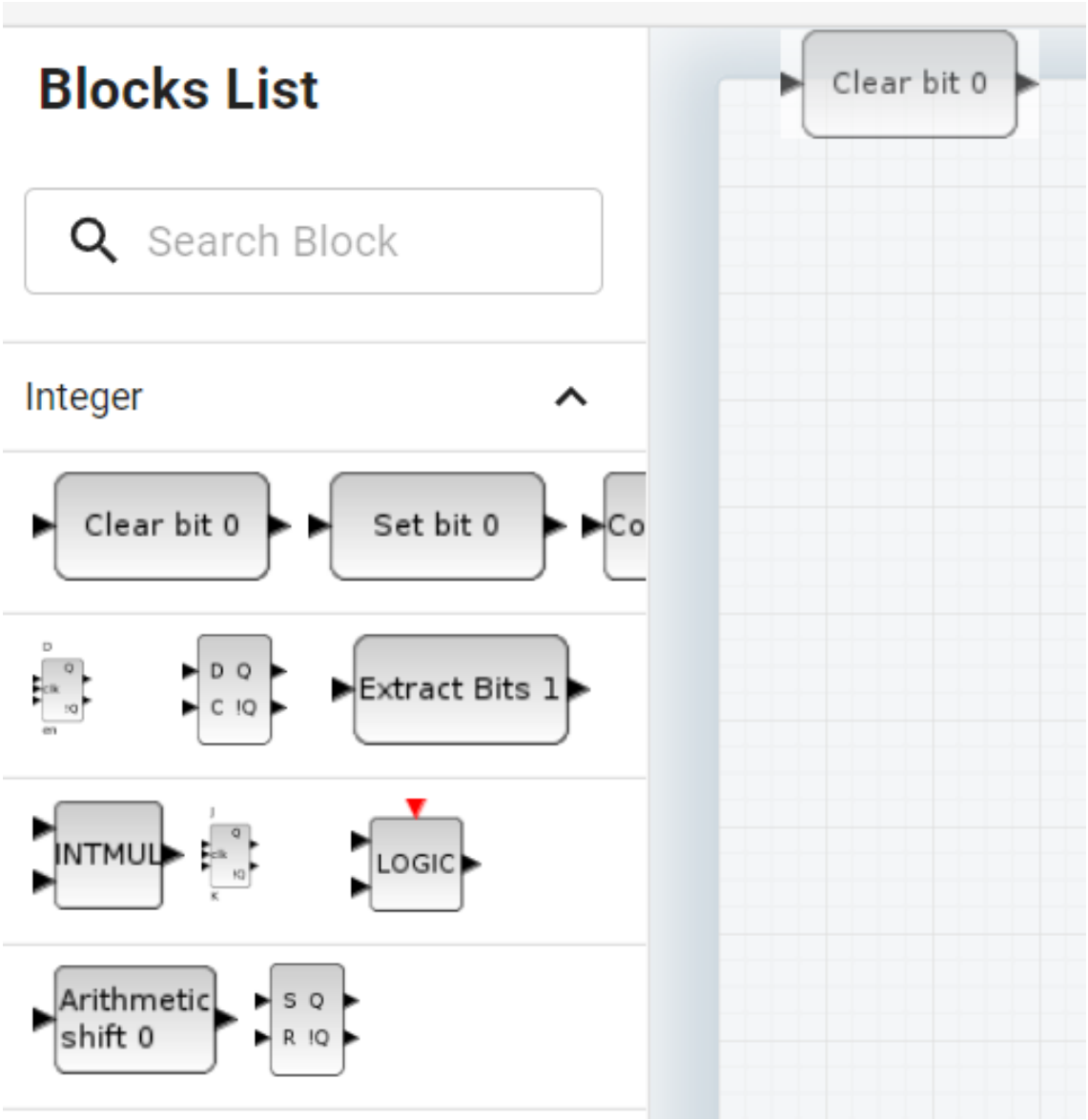


The initial canvas with area-gap between the sidebar and the field.



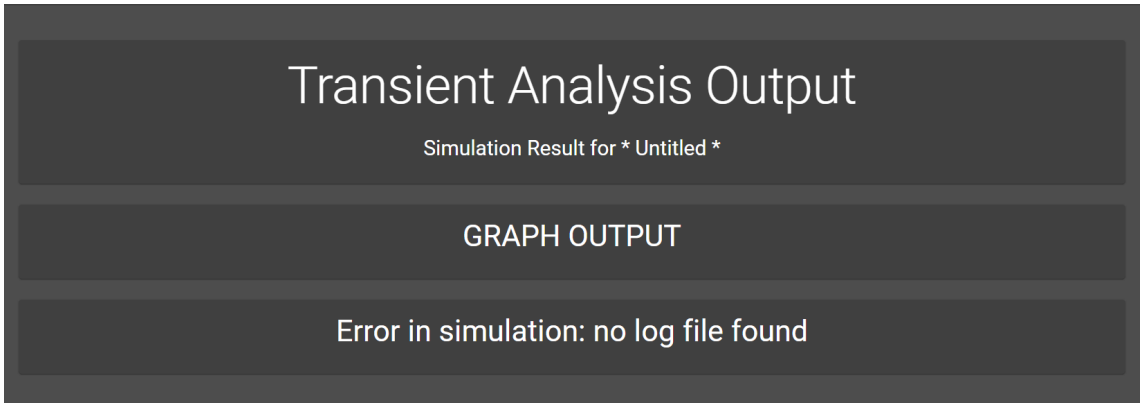
The canvas with no area-gap between the sidebar and the field.

Component Fix



The component being dragged was initially invisible.

Empty Log Fix



Initally the user had to wait even if the log file was empty, now the code detects the empty log file and displays the specific message to the user.

EventSource Close Connection

```
sse.addEventListener('duplicate', e => {
  printloglines()
  console.log('duplicate', e)
}, false)
sse.addEventListener('DONE', () => {
  printloglines()
  console.log('DONE')
  sse.close()
  setGraphStatusDone()
}, false)
sse.addEventListener('ERROR', e => {
  printloglines()
  console.log('ERROR', e)
  setError('Error in simulation: ' + e.data)
  sse.close()
}, false)
sse.addEventListener('MESSAGE', e => {
  printloglines()
  console.log('MESSAGE', e)
  sse.close()
}, false)
```

Using `addEventListener` for identification of done and closing the EventSource connection.

Chapter 7

Git and Github

Git is a fast, scalable, distributed revision control system with an unusually rich command set that provides both high-level operations and full access to internals.[2]

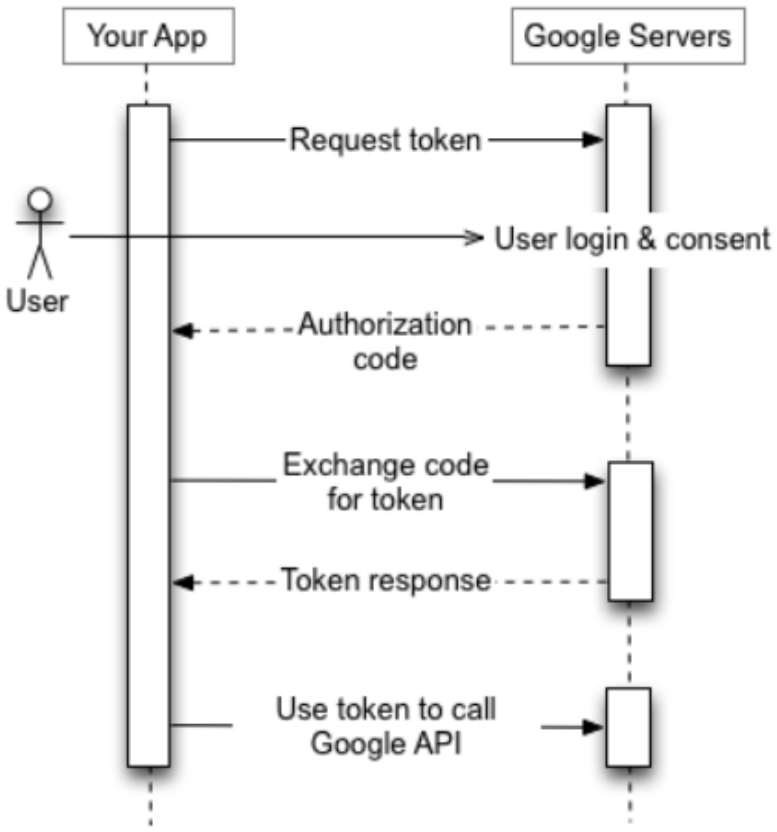
GitHub is a code hosting platform for version control and collaboration. It lets us and others work together on projects from anywhere.[3]

In this project there were three branches namely no-backend, no-backend-windows and Xcosblocks. Most of the work was done on the no-backend-windows branch as it didn't require the installation of Scilab, the changes made on this branch were merged to the other branches. This was a great experience teaching the importance of changing the branch depending on the system environment and accessibility to different packages. Overall, the code was maintained and merged to all the branches.

Chapter 8

Google OAuth

Google APIs use the OAuth 2.0 protocol for authentication and authorization. Google supports common OAuth 2.0 scenarios such as those for web server, client-side, installed and limited-input device applications.
To begin, We obtain OAuth 2.0 client credentials from the Google API Console. Then our client application requests an access token from the Google authorization server, extracts a token from the response, and sends the token to the Google API that we want to access. For an interactive demonstration of using OAuth 2.0 with Google (including the option to use our own client credentials)



```
function Loginpage () {
  const onSuccess = (res) => {
    console.log('[Login Success] currentUser:', res.profileObj)
  }

  const onFailure = (res) => {
    console.log('[Login failed] res:', res)
  }
  return (
        <div>
        <Googlelogin
            clientId={clientId}
            buttonText='Login'
            onSuccess={onSuccess}
            onFailure={onFailure}
            cookiePolicy={'single_host_origin'}
            style={{ marginTop: '100px' }}
            isSignedIn={true}
      />
        </div>
  )
}

export default Loginpage
```

Using the Google provided login button for easy access. [4]

Reference

[1] *EventSource - web apis: MDN*. URL: https://developer.mozilla.org/en-US/docs/Web/API/EventSource.

[2] *Git documentation*. URL: https://git-scm.com/docs/git.

[3] *Github Documentation*. URL: https://docs.github.com/en/get-started/quickstart/hello-world.

[4] *Google OAuth 2.0*. URL: https://developers.google.com/identity/protocols/oauth2.

[5] *Highcharts documentation: Highcharts*. URL: https://www.highcharts.com/docs/index.