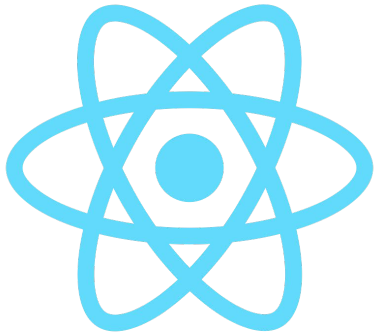
# DisConvReact

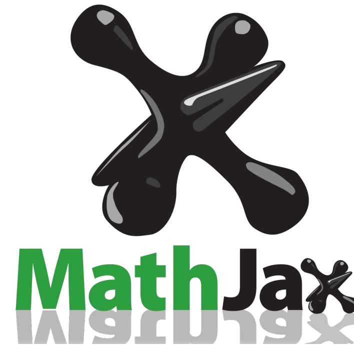
Spring 2020

# Introductions

- Sean Crowley
  - 4th year CS
  - Javascript, C, C++, Java, and Python
- Camille Bossut
  - 3rd year CS
  - Javascript, C, C++, Java, Python, Matlab
- Elias Reta
  - 2nd year CM
  - Java, Python, HTML, CSS
- Suma Cherkadi
  - 2nd year CS
  - Java, Python, C, Scala, Javascript, ReactJS
- Zihuan Wu(Mark)
  - 3rd year CS
  - C, Java, Python and R

# Tools Used



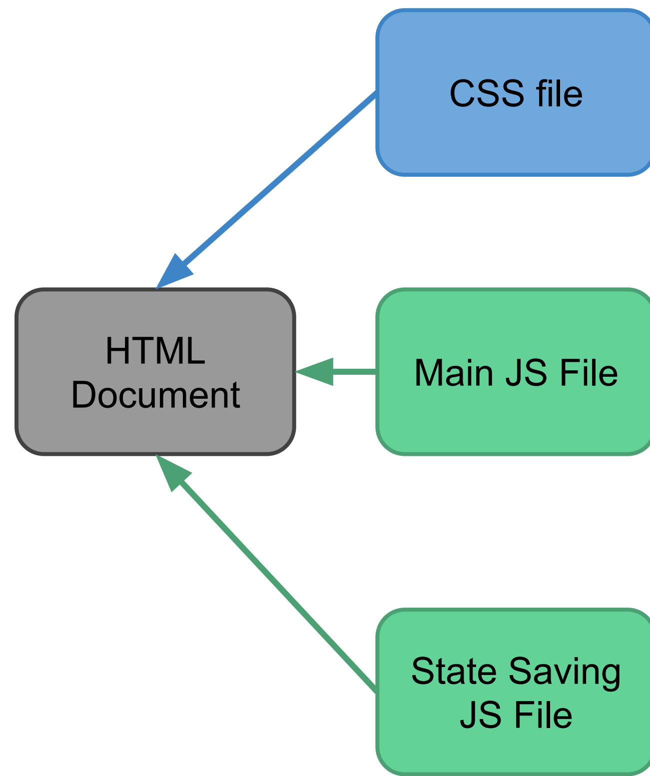


# Project Plan

- Integrated old **DisConvDemo** GUI with React and NodeJS
  - Current function types: Constant, Sinusoid, Exponential
- Added database access using **Express**
  - Enabled state saving and restoring
- Created **React and Express hierarchy** template for future teams

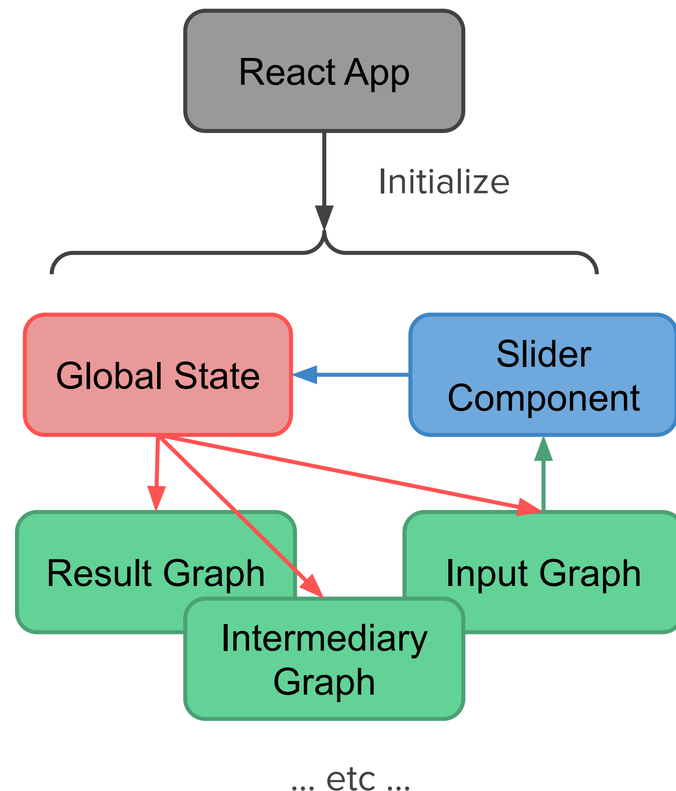
# Motivation

- Issues with previous design:
- Plain Javascript and HTML didn't work well for saving state
  - Required use of PHP (outdated)
- Code lacked organization and structure



# Motivation

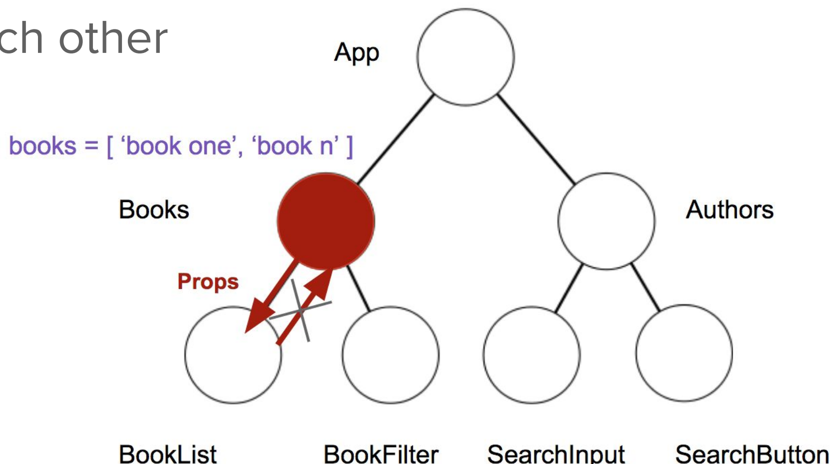
- React is better for *code organization* and *version control*
- React and NodeJS allow us to use *Express for database elements*
- ITS has long-term goal to make a GUI template in React
  - React format for GUIs is preferable to test this out



# What is REACT?

```
<App>
  <NavBar />
  <Header title="ReactJS Academy" />
  <BookFilter />
  <BookList />
  <Footer />
</App>
```

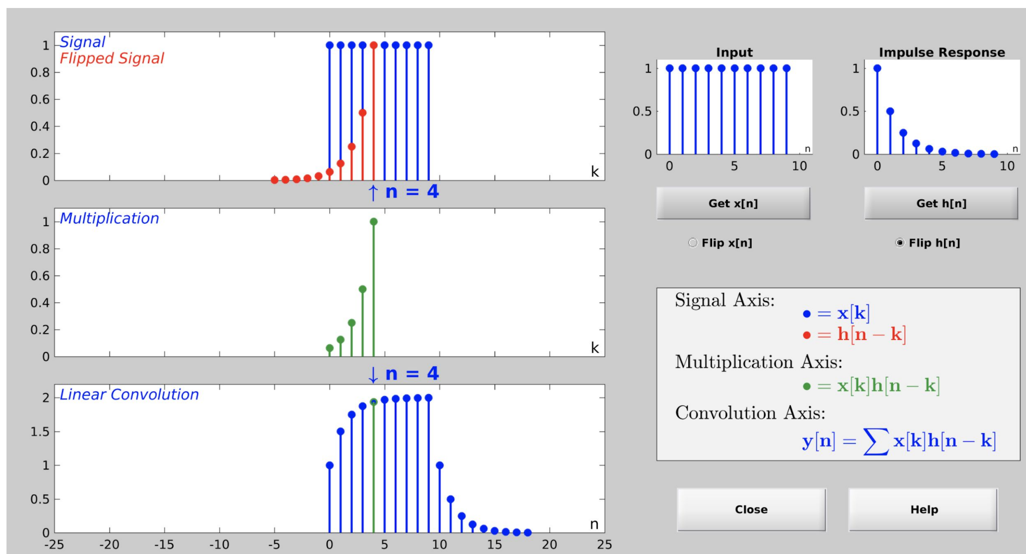
- A JavaScript library that creates UIs
- Components are independent from each other
- Component Organization
    - Props and States

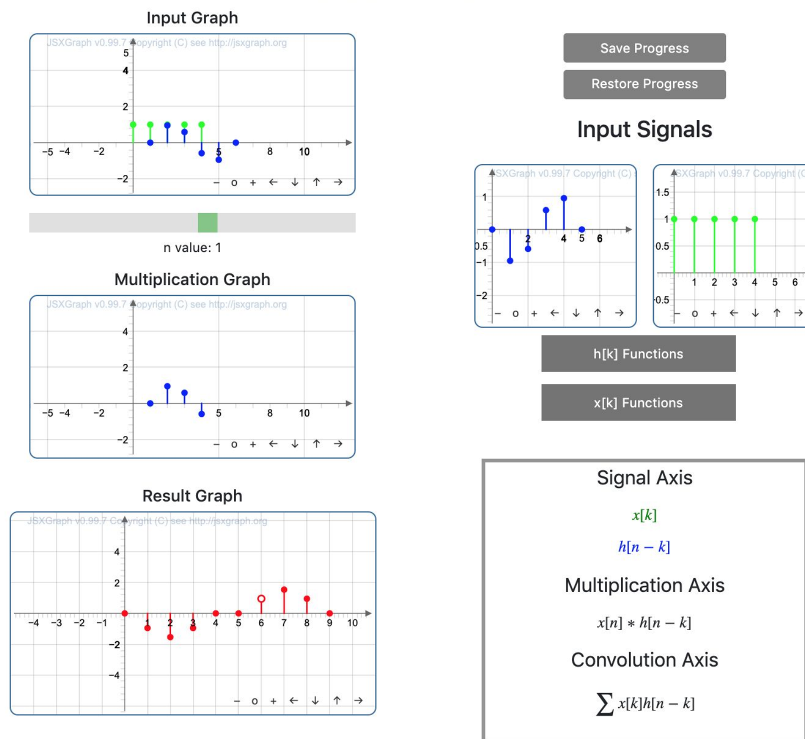


# Frontend

- Improve code structure
  - Remove hard-coded values, have an organized file system
- Adapt code structure to React format
  - Code should be organized into react components
- Add user input and display formulas
  - User input should have the same options as the matlab GUI

# Frontend




Matlab

Last Semester

# Frontend — Elias

Replaced hard-coded functions with

More versatile Signal Functions

- User Signal(pictured)
- Exponential
- Sine/Cosine
- Constant Signal

Written in Javascript

```
function userSignal(userData, backingArray) {
  for (let j = 0; j <= userData.length; j++) {
    if (userData.charAt(j) != ",") {
      try {
        var input = userData.charAt(j)
        var intInput = parseInt(input)
        backingArray.push(intInput)
      }
      catch {
        throw new Error('Invalid Input');
      }
      finally {
        continue
      }
    }
  }
  return backingArray
}
```

# Frontend — Elias

Other Signal Functions

- User Signal
- Exponential
- Sine/Cosine
- Constant Signal

```
export function sinSignal(array, num_samples, amp, freq, phase) {
    for (var n = 0; n < num_samples; n++) {
        array[n] = amp * Math.sin(n * freq + phase);
    }
}

export function expSignal(array, num_samples, scalingFactor, expConstant) {
    for (let i = 0; i <= num_samples; i++) {
        array[i] = scalingFactor * Math.pow(expConstant, i);
    }
}

export function constSignal(array, num_samples, constant) {
    for (let i = 0; i <= num_samples; i++) {
        array[i] = constant;
    }
}
```

# Frontend — Elias

- Used **Mathjax** and **algebra.js** libraries to
- improve display of functions

MathJax

$$f(a) = \frac{1}{2\pi i} \oint_\gamma \frac{f(z)}{z-a} dz$$

```
const sineFunc = 'f(x) = asin(bx + c) + d'
const expFunc = 'f(n) = ca ^ n - nd'
```

```
return (
    <div>
      <MathJax.Context input='ascii'>
                    <div>
                        <MathJax.Node inline>{ sineFunc }</MathJax.Node>
                        <MathJax.Node inline>{ expFunc }</MathJax.Node>
                    </div>
                </MathJax.Context>
```

# Frontend — Camille

- Create a structure based off of React components
  - Component rendered for each graph type
- Create global state variable
  - One state for the graphs to remain updated
  - One to store in the database (only essential information)

# Frontend — Camille

- Use jsxgraph-react-js library to incorporate JSX plots
  - Require an init function to be passed in
  - Save a reference to the JSX Board within that function to the global state

```
InputsOverlap = (brd) => {
    Graph_State.inputs_board = brd;
    var input1_stems = brd.create('curve',[[0], [0]], {strokeWidth:2, color:"blue",
    input1_stems.updateDataArray = function() {
        this.dataX = [];
        this.dataY = [];

        for (var n = 0; n <  Graph_State.input1_x.length; n++) {
            var x = n +  Graph_State.sliderval/1;
            var y =  Graph_State.input1_x[n];
            // alert(y);

            this.dataX.push(x);  // Start of a stem
            this.dataY.push(0);
            this.dataX.push(x);  // End of stem
            this.dataY.push(y);
            this.dataX.push( NaN ); // Interrupt the curve
            this.dataY.push( NaN );
        }
    };
```

# Frontend — Camille

- Use jsxgraph-react-js library to incorporate JSX plots
  - Require an init function to be passed in
  - Save a reference to the JSX Board within that function to the global state

```
var Graph_State = {
    input1_x : [1,1,1,1,1],
    input2_x : [1,1,1,1,1],
    result : [],
    diff : 0,
    sliderval : 5,
    inputs_board: null,
    mult_board: null,
    res_board:null,
    slider: null
};
```

# Frontend — Camille

- Values in the Store_State variable are used to initialize the stem plots
  - This makes it easier to restore state

```
var Store_State = {
    userID: 2,
    func_1: func.CONSTANT,
    len_1 : 5,
    amp_scale_1 : 1,
    freq_const_1 : 1,
    phase_1 : 0,
    func_2: func.CONSTANT,
    len_2 : 10,
    amp_scale_2 : 2,
    freq_const_2 : 0.68,
    phase_2 : 0
};
```

```
// console.log(Store_State.len_1);
if(Store_State.func_1 == 3) {
    sinSignal(Graph_State.input1_x, Store_State.len_1, Store_State.amp_scale_1, Store_State.freq_const_1, Store_State.phase_1);
} else if(Store_State.func_1 == 2) {
    expSignal(Graph_State.input1_x, Store_State.len_1, Store_State.amp_scale_1, Store_State.freq_const_1);
} else {
    constSignal(Graph_State.input1_x, Store_State.len_1, Store_State.freq_const_1);
}
```

# Frontend — Mark

- Get User Input
  - Use Drop_Down_List.js to gather all user input signals
  - Selections for both functions and relevant parameters

```
else {
  return (
      <form onSubmit={this.handleSubmit}>
          <label>
              Function Type:
              <select value={this.state.value} onChange={this.handleChange}>
                  <option value="const">Constant</option>
                  <option value="sin">Sinusoid</option>
                  <option value="exp">Exponential</option>
              </select>
          </label>
          {/* <input type="submit" value="Submit" /> */}
      </form>
  );
```

```
<label>
    <br></br>
    Phase
    <input type="text" onChange={this.handlePhaseChange}>
    </input>
</label>
<label>
    <br></br>
    Amplitude
    <input type="text" onChange={this.handleAmpScaleChange}>
    </input>
</label>
<label>
    <br></br>
    Number of Samples
    <input type="text" onChange={this.handleSamplesChange}>
    </input>
```

# Frontend - Mark

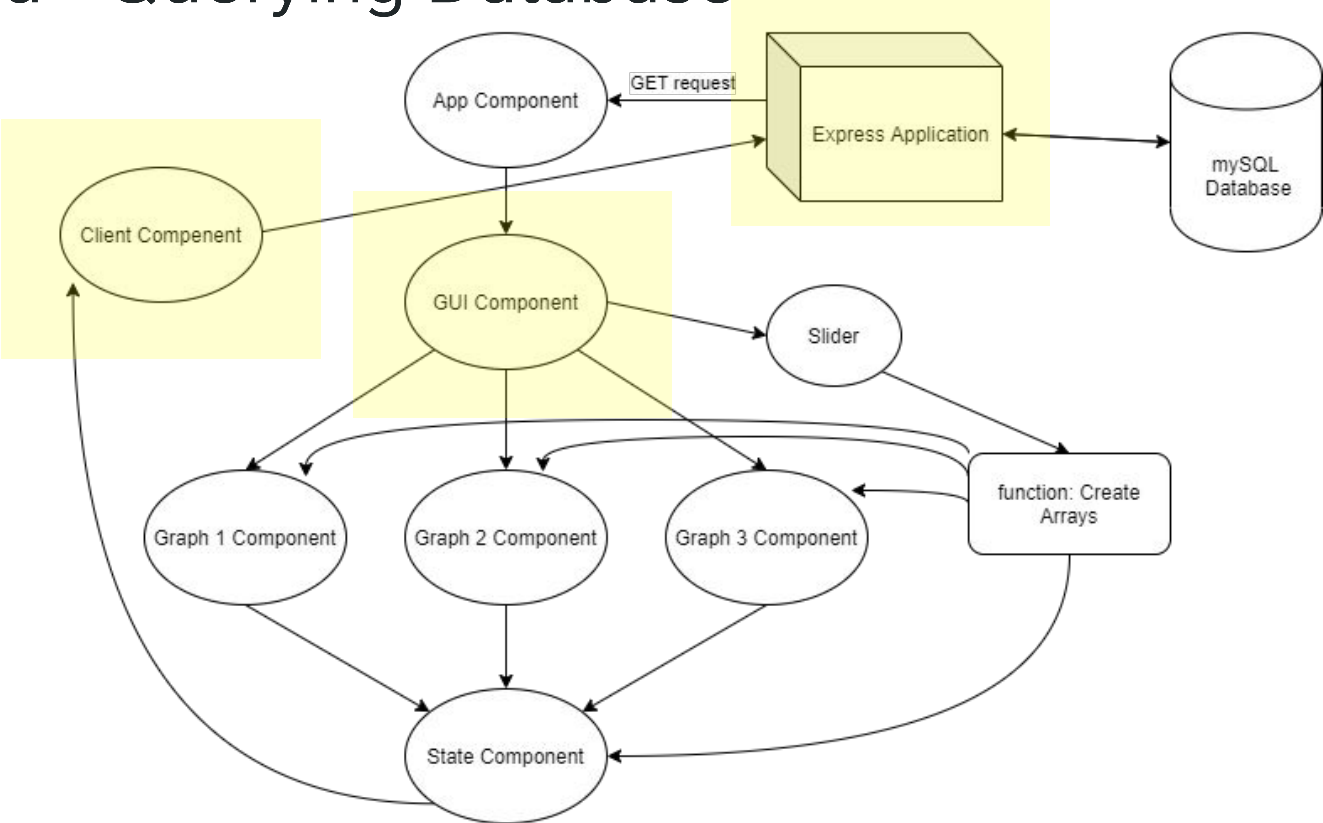
- Utilize User Input
  - Modify variables in Store_State.js
  - Make function call to update the array
  - Display array on the frontend

```
handleFreqConstChange(event) {
    Store_State.freq_const_2 = event.target.value;
    if(this.state.value == "const") {
        constSignal(Graph_State.input2_x, Store_State.len_2, Store_State.freq_const_2);
    } else if(this.state.value == "sin") {
        sinSignal(Graph_State.input2_x, Store_State.len_2, Store_State.freq_const_2);
    } else if(this.state.value == "exp") {
        expSignal(Graph_State.input2_x, Store_State.len_2, Store_State.freq_const_2);
    }
    Graph_State.inputs_board.update();
    Graph_State.mult_board.update();
    convolve();
    Graph_State.res_board.update();
```

# Backend

- Update the lab to use an Express backend rather than PHP
- Create a “blackbox” for database reads and writes that can be easily utilized by the Frontend
  - saveState: POST request to save the Store_State object
  - restoreState: Calls getParams to retrieve data via UserID. If the call to the database is resolved the Store_State is restored accordingly.

# Backend - Querying Database



# Backend - GUI Uses Client.js

```
saveState() {
    Client.saveState(Store_State);
}
```

```
restoreState() {
    let data = {userID: Store_State.userID};
    Client.getParams(data).then(params =>
        {   console.log(params[0]);
```

Frontend can use Client file as a blackbox to submit and retrieve data (in this case function parameters) from the database

# Backend - Client Interacts With Express Application

- Client file handles all CRUD functions
  - getParams(): reads saved parameters from a given userID
  - saveState(): passes parameter to Express application to be saved in database

```
return fetch( input: '/arrays/add',  init: {
    method: 'POST',
    headers: {'Content-Type': 'application/json'},
    body: JSON.stringify(data)
```

# Backend - Express Queries Database

- Based on the route used by the Client function, the Express application performs a:
  - SELECT command to read data
  - INSERT INTO command to write data

```
db.query( sql: "SELECT * FROM params WHERE userID = " + req.body.userID,
    values: function (err, result, fields) {
```

*Demo!*

# Conclusion

- We achieved:
  - Porting the initial GUI over to react
  - Saving a GUI state to the database
- Improvements:
  - Improve layout and add more user input options
  - Allow user to be able to update the data they want to save in the database

*Questions?*