Documentation for ‘MYdata’:

Spring 2023 CAPSTONE:

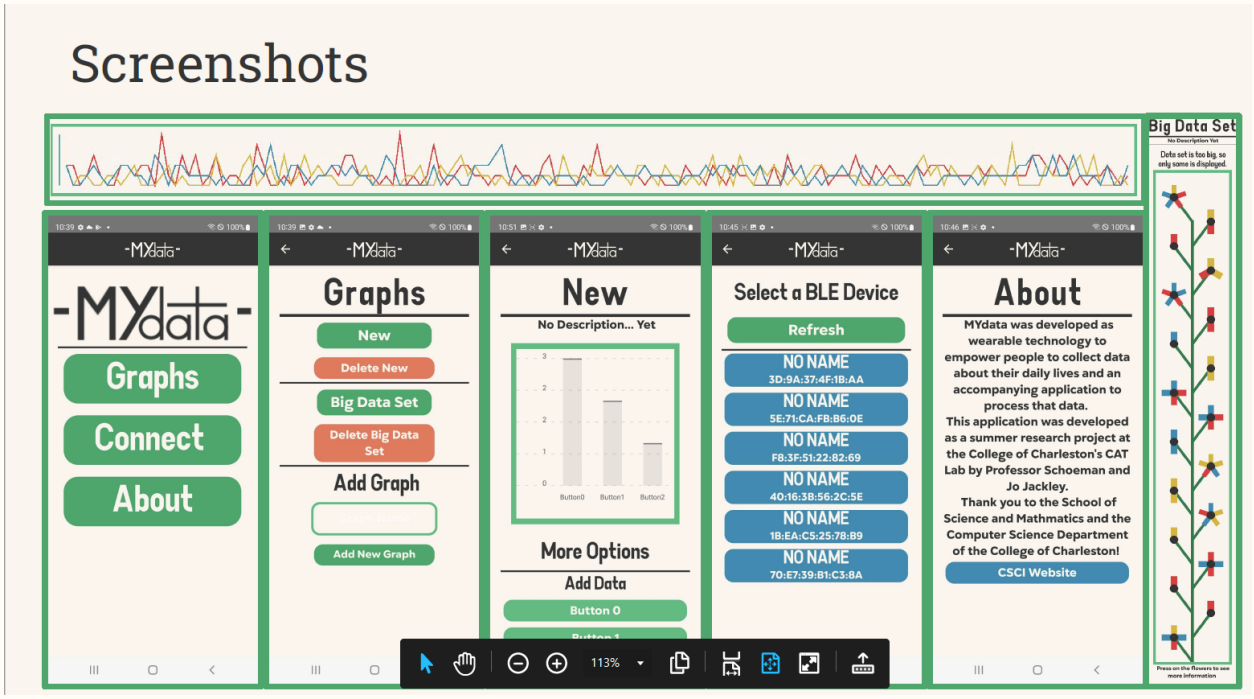
Team: Sarah Greene, Alex Alsheimer, Drew Smuniewski, Emma Satterfield, Lucas Fancher

Project Description: Our main tasks were to update the app’s UI, as well as add new graphs and functionality.

NOTE: Recommended to use Document Outline for easier navigation of this document.

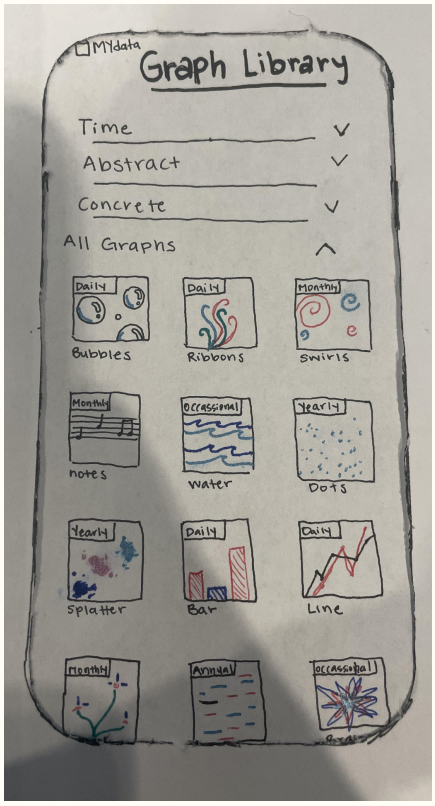
Previous MYdata Documentation:

Previous MYdata Layout:

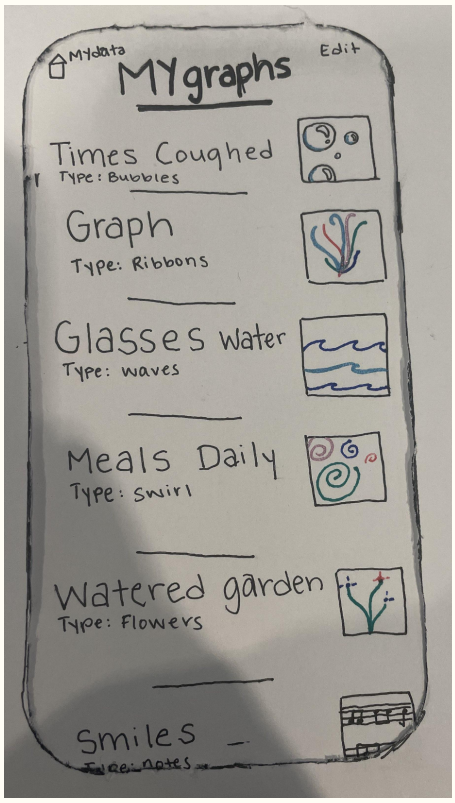


(Image from presentation from Jan. 2023)

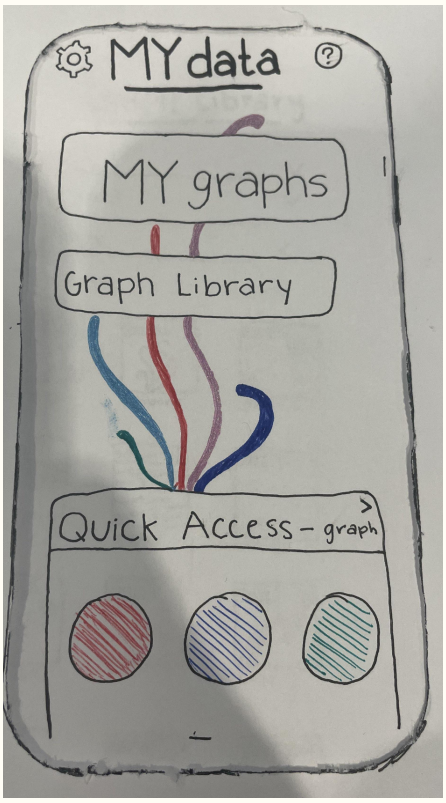
Paper Prototype V.1:



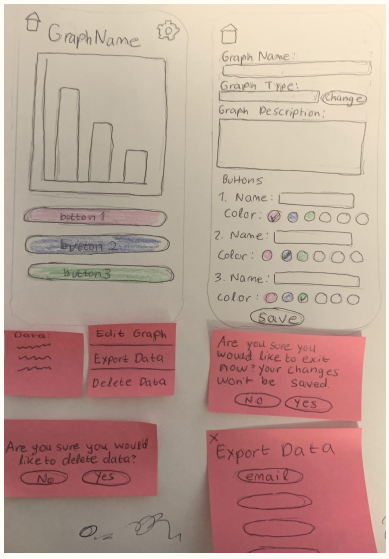
Graph Library Screen



MYgraphs Screen

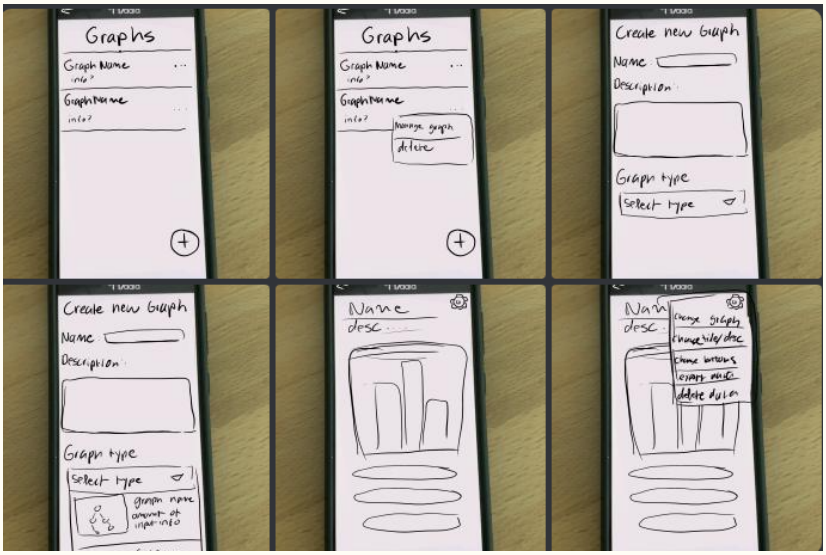


Homepage



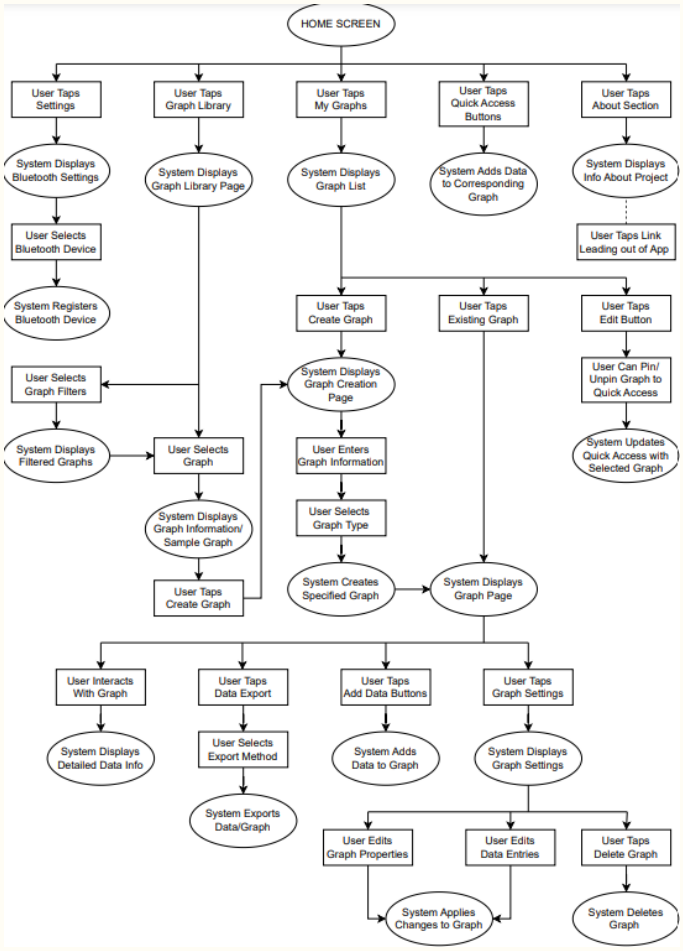
Graph Creation Screen

Prototype V.2:

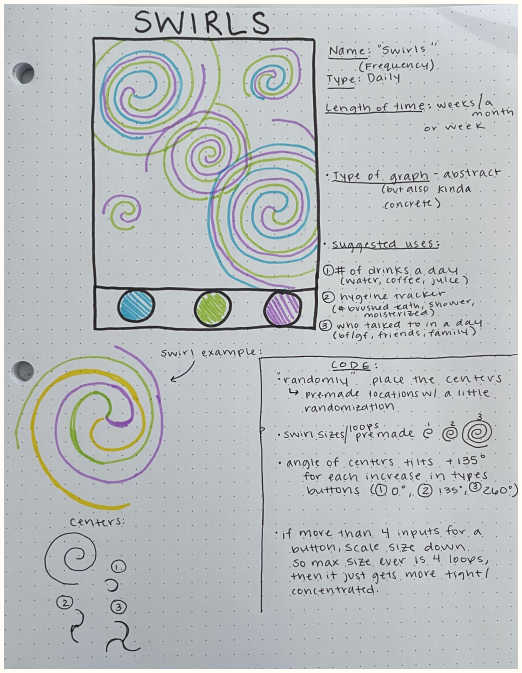


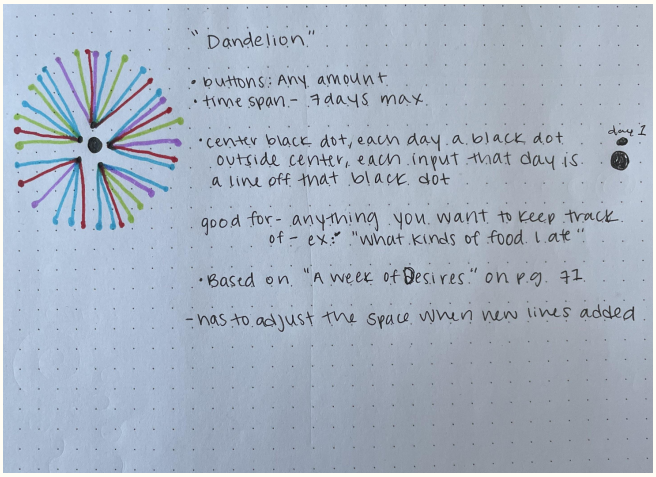
Higher Fidelity Versions

Updated Workflow:



New Graph Paper Prototypes:





"Swirls"

"Dandelion"

Code Breakdown/ Explanation:

The code breakdown/ explanation will be more of a quick guide on how to add graphs and an example of what the code from one graph looks like then the last section will be an installation and run guide.

NOTE: As of the time of writing this guide (4/20/2023), there are TWO computers located in the Innovation Center on the second floor with everything already installed and ready to go.

Quick Guide:

How to Add a Graph into App.js:

Add code corresponding to the graph in the similar style of already listed graphs in the sections under "//User Graph Imports", "const graphOptions =", and add a description for the graph under "const graphDescriptions"

EXAMPLE (ALL HIGHLIGHTED TEXT- do not copy and paste this whole code, the example will not work and is filled with placeholders):

```
// User Graph Imports
import BarGraph from "./components/GraphLibraries/barGraph.js";
import HeatMap from './components/GraphLibraries/heatGraph.js';
import ButtonOrderGraph from "./components/GraphLibraries/buttonOrderPressed.js"
import TimesPerDay from './components/GraphLibraries/timePressedEachDay.js';
import FlowerGraph from './components/GraphLibraries/FlowerGraph.js';
import Triskelion from './components/GraphLibraries/triskelion.js';
import ChessClock from './components/GraphLibraries/chessClock.js';
import ButtonPressedPerDay from './components/GraphLibraries/EachButtonPressedPerDay.js';
import Dandelion from "./components/GraphLibraries/dandelion.js";
import (GRAPH NAME) from '(GRAPH LOCATION)';

const BleManagerModule = NativeModules.BleManager;
const bleEmitter = new NativeEventEmitter(BleManagerModule);

//Switches between the two styles, set as either lightStyles or darkStyles
```

```
let styles=lightStyles;

//used to add new graphs
// graphOptions is the list of graphs that can be selected
const graphOptions =[
  <Picker.Item label="Set Graph Type" value="NoDataYet" style={styles.pickerDropdown}/>,
  <Picker.Item label="Bar Graph" value="BarGraph" style={styles.pickerDropdown}/>,
  <Picker.Item label="Heat Map" value="HeatMap" style={styles.pickerDropdown}/>,
  <Picker.Item label="Button Order" value="ButtonOrderGraph" style={styles.pickerDropdown}/>,
  <Picker.Item label="Times Buttons Were Pressed Each Day" value="TimesPerDay" style={styles.pickerDropdown}/>,
  <Picker.Item label="Flowers" value="Flower" style={styles.pickerDropdown} />,
  <Picker.Item label="Knot" value="Triskelion" style={styles.pickerDropdown} />,
  <Picker.Item label="Chess Clock" value="ChessClock" style={styles.pickerDropdown}/>,
  <Picker.Item label="Total Buttons Pressed per Day" value="ButtonPressedPerDay" style={styles.pickerDropdown}/>,
  <Picker.Item label="Dandelion" value="Dandelion" style={styles.pickerDropdown} />,
  <Picker.Item label= “(GRAPH NAME)” value= “(GRAPH VALUE)” style = {styles.pickerDropdown} />,
]

// graphDescriptions is the graph's descriptions
const graphDescriptions = [
  <Text style={styles.tinyText} name="NoDataYet"> No Graph Selected. </Text>,
  <Text style={styles.tinyText} name="BarGraph"> Displays the number of times each button is pressed. </Text>,
  <Text style={styles.tinyText} name="HeatMap"> Displays the number of times all buttons are pressed for each day.
</Text>,
  <Text style={styles.tinyText} name="ButtonOrderGraph"> Displays the order of button pushes. </Text>,
  <Text style={styles.tinyText} name="TimesPerDay"> Displays when buttons are pressed each day.</Text>,
  <Text style={styles.tinyText} name="Flower"> Displays the buttons pressed each day. </Text>,
  <Text style={styles.tinyText} name="Triskelion"> Displays the number of button pairs pressed. </Text>,
  <Text style={styles.tinyText} name="ChessClock"> Displays the duration between the same button being
pressed.</Text>,
  <Text style={styles.tinyText} name="ButtonPressedPerDay"> Displays the number of times each button was pressed
each day </Text>,
  <Text style={styles.tinyText} name="Dandelion"> Displays the buttons pressed over a week. </Text>,
  <Text style={styles.tinyText} name=” (GRAPH NAME)” > (DESCRIPTION OF USE) </Text>,
]
// DON'T FORGET TO ADD TO GRAPHSWITCH IN GRAPHS AS WELL
```

EXAMPLE OF A GRAPH CODE (using the Dandelion graph as example):

```
// default imports
import React, {useEffect, useState, Suspense } from 'react'
import {View,Text, Dimensions,TouchableWithoutFeedback} from 'react-native';
// React-Native-Chart-Kit import; simple graphs
import {LineChart,BarChart,PieChart,ProgressChart,ContributionGraph,StackedBarChart,} from "react-native-chart-kit";
// React-Native-Svg import
import Svg,
{Circle,Ellipse,G,TSpan,TextPath,Path,Polygon,Polyline,Line,Rect,Use,Image,Symbol,Defs,LinearGradient,RadialGradient,Stop,ClipPath
,Pattern,Mask,} from 'react-native-svg';
// Table stuff
import { Table, TableWrapper, Row, Rows, Col, Cols, Cell } from 'react-native-table-component';
// Global Variables
import GLOBAL from "../global.js";

// all colors
let backgroundColor="#faf5ef";
let highlight="#63ba83";
let midtone = "#4ea66d";
let dark="#343434";
```

```
let warning = "#E07A5F";
let pink="#c740d6";
let red="#d64040";
let yellow="#d6b640";
let green="#4ea66d";
let blue="#438ab0";
let teal="#43b0a9";
let indigo="#6243b0";

let colorArray = [red, blue, yellow];


export default class Dandelion extends React.Component {

  // State of the class, data stored in here
  // @param: props, the props passed in from the the parent class
  constructor(props) {
    super(props);
    this.state = {
      isLoading: true,
      numberOfDays:0,
      endDate:new Date("2017-04-01"),
      title:"Whatever you want",
      graphData: [],
      dateList: [],
      dateListTotals: [],
      tableHead: [' Column 1', ' Column 2',],
      tableData: [['row 1', 'row 1'], ['row 2', 'row 2']]
    }
  }

  // used to check if props have updated
  previousProps;
  // if you integrate data point descriptions, you may use this to store the descriptionData
  descriptionList=[];

  // When the passed in value changes, this is called
  // Updates the state for the graph
  componentDidUpdate(prevProps, prevState) {
    if (prevProps !== this.props || this.state.isLoading) {
      let tempGraphData=this.DataProcessing(this.props.rawData);
      let newTableData=this.ChangeTableData(tempGraphData);
      this.setState({
        isLoading:false,
        graphData:tempGraphData,
        tableData:newTableData,
        title:this.props.rawData.Title,
      });
      this.previousProps=this.props;
    }
  }

  // This is Called on load
  // Updates the state for the graph
  componentDidMount(){
    let tempGraphData=this.DataProcessing(GLOBAL.ITEM);
    let newTableData=this.ChangeTableData(tempGraphData);
    this.setState({
      isLoading:false,
      graphData:tempGraphData,
      tableData:newTableData,
      title:GLOBAL.ITEM.Title,
    });
  }

  // This is used to manually reload the state
  updateData(){
      this.setState({
        isLoading:true,
      });
  }

  reformatDate = (rawJson, i, j) => {
```

```
    let tempJson = {};
    let date = new Date(rawJson.data[j].Year, rawJson.data[j].Month+1, rawJson.data[j].Day, rawJson.data[j].Hour,
rawJson.data[j].Minutes, rawJson.data[j].Seconds, rawJson.data[j].Milliseconds);
    tempJson.data = date;
    tempJson.button = i;
    tempJson.buttonName = rawJson.ButtonName;
    return tempJson;
  }

  // processes the data
  DataProcessing = (rawJson) => {
    //TODO: Write code to parse the data
    let dataArray = rawJson.Data;
    let allData = [];
    for(let i = 0 ; i < dataArray.length;i++){
      for(let j = 0; j < dataArray[i].data.length;j++){
        allData.push(this.reformatDate(dataArray[i],i,j));
      }
    };

    if (allData.length > 0) {
      this.quickSort(allData, 0, (allData.length - 1));
    }

    let TempDates = [];
    for (let i = 0; i < allData.length; i++) {
      TempDates.push(allData[i].data.toDateString());
    }

    let dates = [...new Set(TempDates.map((date) => date))];
    let dateTotals = [];

    for (let i = 0; i < dates.length; i++) {
      let count = 0;
      for (let j = 0; j < TempDates.length; j++) {
        if (TempDates[j] === dates[i]) {
          count++;
        }
      }
      dateTotals.push(count);
    }

    this.setState({
      dateList:dates,
      dateListTotals:dateTotals,
    });

    return allData;
  }

  // Changes TableData for BarGraph
  // tempData by default is the parsed data from DataProcessing
  ChangeTableData = (tempData) => {
    let tableDataClone=[];
    // TODO: write the code to return the table's data
    return tableDataClone;
  }

  swap=(arr,xp, yp)=>{
    var temp = arr[xp];
    arr[xp] = arr[yp];
    arr[yp] = temp;
  }

  partition = (arr, low, high) => {
    let pivot = arr[high];
    let i = (low - 1);

    for (let j = low; j <= high - 1; j++) {
      if (arr[j].data < pivot.data) {
        i++;
        this.swap(arr, i, j);
      }
```

```
    }
    this.swap(arr, i + 1, high);
    return (i + 1);
  }

  quickSort = (arr, low, high) => {
    if (low < high) {
      let pi = this.partition(arr, low, high);
      this.quickSort(arr, low, pi - 1);
      this.quickSort(arr, pi + 1, high);
    }
  }

  render() {
    return (
      <View style={this.props.styles.container}>
        <View style={this.props.styles.border}>
          <GetDandelion
            data={this.state.graphData}
            dateList={this.state.dateList}
            dateTotals={this.state.dateListTotals}
            colors={colorArray}
            styles={this.props.styles}>
          </GetDandelion>
        </View>
      </View>
    );
  }
}

function GetDandelion({data, dateList, dateTotals, colors, styles}) {
  let days = [];
  let spokes = [];
  let entries = [];
  let width = Dimensions.get("window").width*.75;
  let height = width;

  let dateLocations = [];

  let circleX = width/2;
  let circleY = height/2;

  let radiusInner = width * .1;
  let radiusOuter = width * .45;

  let prev = -1;
  for (let i = 0; i < dateList.length; i++) {
    let min = prev + 1;
    let max = prev + dateTotals[i];

    let minAngle = min * ((2 * Math.PI) / data.length);
    let maxAngle = max * ((2 * Math.PI) / data.length);
    let finalAngle = ((maxAngle - minAngle) / 2) + minAngle;

    let dayX = (Math.sin(finalAngle) * radiusInner) + circleX;
    let dayY = (Math.cos(finalAngle) * radiusInner) + circleY;

    dateLocations.push([dayX, dayY]);
    days.push(<Circle cx={dayX} cy={dayY} r={width*0.02} fill={dark}/>)

    prev = max;
  }


  for (let i = 0; i < data.length; i++) {
    let angle = i * ((2 * Math.PI) / data.length);
    let dataX = (Math.sin(angle) * radiusOuter) + circleX;
    let dataY = (Math.cos(angle) * radiusOuter) + circleY;

    entries.push(<Circle cx={dataX} cy={dataY} r={width*0.01} fill={colors[data[i].button]}/>);

    const dateMatch = (element) => element === date;
    let date = data[i].data.toDateString();
```

```
    let index = dateList.findIndex(dateMatch)
    let dateCoordinates = dateLocations[index];
    let dayX = dateCoordinates[0];
    let dayY = dateCoordinates[1];

    spokes.push(<Line x1={dataX} y1={dataY} x2={dayX} y2={dayY} strokeWidth={1.5} stroke={colors[data[i].button]}/>);
  }

  return (
    <View>
      <Svg height={height} width={width} key={"Dandelion"}>
        <Circle cx={circleX} cy={circleY} r={width*.04} fill={dark}/>
        {entries}
        {spokes}
        {days}
      </Svg>
    </View>
  );
}
```

Installation Guide:

Before Installation:

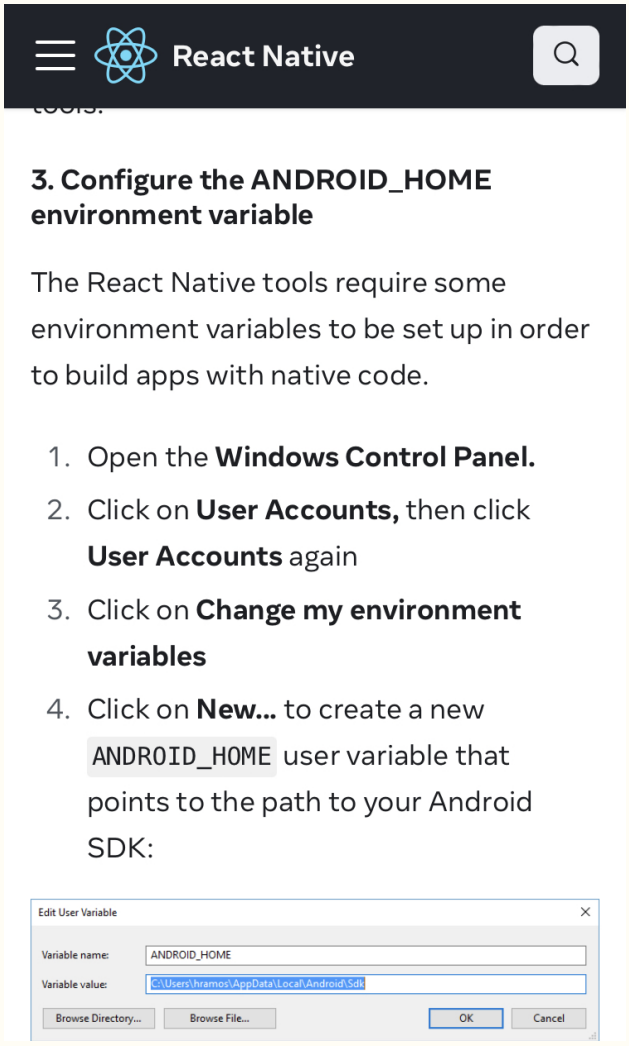
1. Make sure you have a copy of Microsoft VS Code and a copy of the MYdata files.
2. (If you are going to use a virtual device aka you do not have a windows phone to test the code on) Make sure the computer allows virtualization. This setting can be toggled in the BIOS.

Installation:

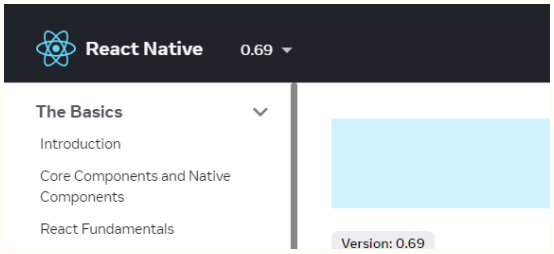
1. Install Chocolatey Package Manager.
2. Through Chocolatey, install Java JDK 11 (or OpenJDK 11) and Node.js (can be done through command prompt starting commands with “choco” - detailed in THIS guide).
3. Install Android Studio Electric Eel with all addons/ packages.
4. Within Android Studio, set up a new virtual device. It should be running Android SDK version ‘R’(11).
5. Within VS Code, open a terminal window and type in the npm/npx codes (listed in the guide) to install React Native CLI. The run-android command can be used to run MYdata’s app.js file in the virtual Android machine.

Additional Notes:

- You may have to configure the ANDROID_HOME environment variable (on certain computers) - (pictured left)



- MYdata was made on React Native 0.69, so you need to include the settings listed under that version. Make sure you switch the version on the React Native website on the dropdown to get the correct dependencies (pictured below).



Capstone Spring 2023- Results:

Members of the MYdata Spring 2023 Capstone: Alex Alsheimer, Lucas Fancher, Sarah Greene, Emma Satterfield, and Drew Smuniewski,

Future Ideas:

Ideas that were liked that we didn't get a chance to implement.

- Color selection in the settings. Allowing for customization of colors for the user with some preset options based off of enjoyed aesthetics and common color blindness combinations for increased accessibility.
- Quick access on the home screen of the app: At the bottom of the home screen, there would be an area for quick access buttons (with colors associated to the currently selected graph's button's colors). This would be to quickly input data (that could be later edited), so that the user could input data in the smallest amount of clicks to be discrete and time saving. (Think about if someone wanted to enter data for every time they saw something with the color blue in one day- they would be accessing this app A LOT. We should make it feel the least like a chore as possible by decreasing the amount of clicks they would need to enter data).
- Lots of UI changes (mainly the ones in our Paper Prototype seen above).