Satellite Antenna Tracker: Web Application

Owner: David Santos
Course: ECEN 403
Team Member: Hunter B. , Michael R.

Table of Contents

Specifications

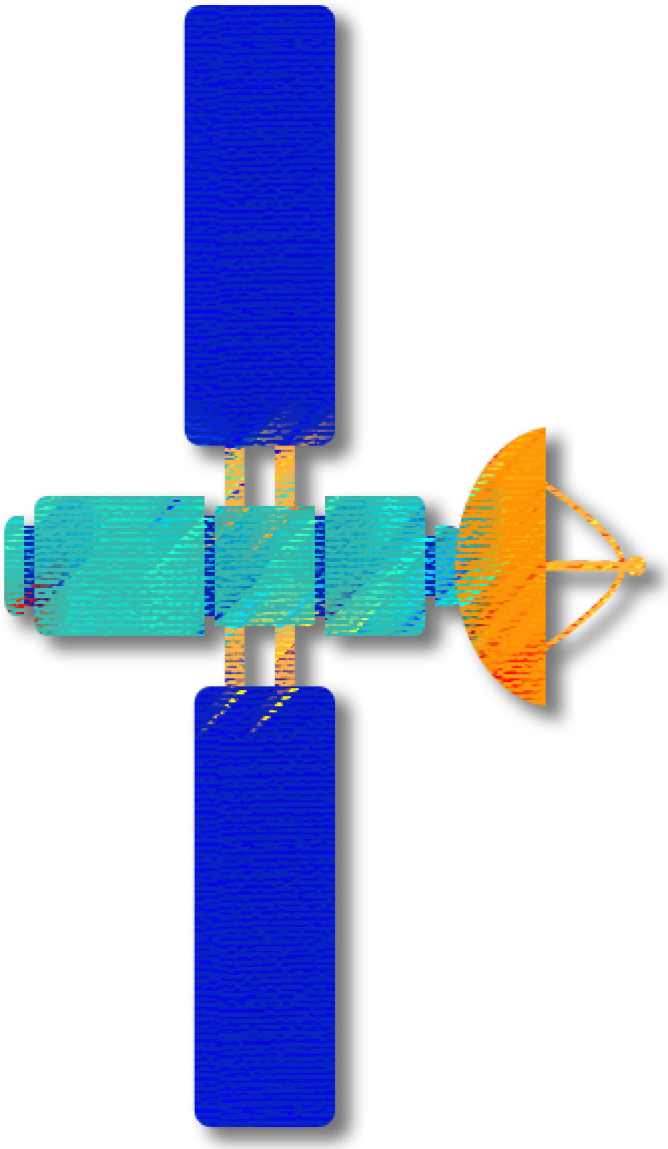
- User should be able to choose a Satellite from database
- App should retrieve and format orbital data for mcu
- User should be able to automatically/manually control rotator remotely
- User should be able to visualize relative position of observer/satellite

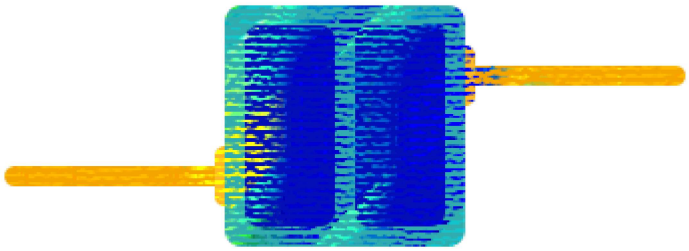
Execution Log

Date	Task
Feb 27	Begun working on toolbar
Feb 27	Added Search Bar, Control Menu Icon, Connect Button
March 3	Created method for searching for satellites and dropdown suggestions
March 4	Added control menu dropdown
March 4	Added control menu switch/fields
March 4	Added Dialog Box to connect to MCU
March 5	Added IPaddress verification
March 5	Created Method for fetching orbital data
March 5	Styled Toolbar
March 5	Styled Dialog Box
March 10	Created Dialog Box Graphic
March 14	Figured out how to send props to React children

March 14	Built test esp8266 server
March 14	Created fetching for mcu server
March 14	Finished Connect/Disconnect Button
March 14	created helper to create Satellite object
March 17	finished Satellite helpers (orbital functions)
March 17	created helper for rotator object
March 18	Signed up for Arcgis developer account and began mercator map
March 20	Added Zoom button and stylized base map
March 20	Finished satellite and rotator graphics in adobe illustrator
March 24	Added MapSettings Button/Logic
March 25	Started creating map helper to add/render graphics
March 25	figured out how to create /plot paths on arcgis
April 1	Finished “get” functions for map
April 3	Figured out how to import png files into map
April 7	Added Rotator Icon
April 8	Started Validation Plan for UI
April 10	Started Validation Plan for usability
April 18	Finished Validation

Graphics





TEST ID	Catagory	Method	Description	Test Cases	Steps	Expected Results/Behavior	Actual Results	Pass/Fail
1	Search Bar	handleSearchTextChange()	handles changes to the search input ar based on the new search text.	"ISS" "Cap" "ZZZ"	Type test case in search bar	"ISS (Zarya)" "CAPE-3" ""	"ISS (Zarya)" "CAPE-3" ""	Pass Pass Pass
2		handleSuggestionClick()	handles a suggestion click event by up setting the search text to the selected s	"ISS"	1. Type in test case in search bar 2. Click A Suggestion	console.log(Satellite != null) => true console.log(SearchText) => ISS (Zarya)	True ISS (Zarya)	Pass Pass
3		handleClickAway()	This function clears the suggestions an away and did not select a satellite.	"ISS"	1. Type in test case in search bar 2. Click away without selecting a satellite	console.log(suggestions) => undefined	undefined	Pass
4	Rotator Connector	handleConnect()	attempts to connect to a submitted IP a the rotator object if successful, and dis messages accordingly.	"slakdgo" "102.2.4.2.4" "123.123.123.123" "192.168.50.100"	1. Click "Connect a Rotator" 2. setup server to 192.168.50.100 3. enter test case 4. Press "Connect"	"Invalid IP" "Invalid IP" "Timed Out" "Connected Sucessfully"	"Invalid IP" "Invalid IP" "Timed Out" "Connected Sucessfully"	Pass Pass Pass Pass
5		handleCancel()	clears the IP address input field, closes any errors.	N/A	1. Click "Connect a Rotator" 2. Press "Cancel"	console.log(openDialog) => false console.log(error) => ""	false ""	Pass Pass
6		handleClick()	opens a dialog box for connecting to a connected rotator, but prevents disconr currently tracking.	isTracking= False connected = False isTracking= False connected = True isTracking= True connected = True	1. reach test case state 2. press "Connect/Disconnect"	Dialog Box Opens Disconnects Rotator Nothing	Dialog Box Opens Disconnects Rotator Nothing	Pass Pass Pass
7		handleInputChange()	This function sets the IP address state change event.	"123.123.123"	1. Press "Connect" 2. type in test case into field	console.log(ipAddress) => "123.123.123"	"123.123.123"	Pass
8	Mercator Map	handleToggleMapSettings()	opens or closes a map settings popup to the current event target.	N/A	1. Mount application 2. Click Map Setting Icon (wrench) 3. Click the Icon again to Close	Map Settings pops up to the left of the I Popup closes	Map Settings pops up to the left of the Popup closes	Pass Pass
9	MapSettings	handleClose()	closes the map settings popup by settir	N/A				
10		handleShowGridChange()	changes the visibility of a grid layer and showGrid.	N/A	1. Mount application 2. Click Map Setting Icon (wrench) 3. Click "Grid" to show 4. Click "Grid" to Close	grid shows on switch right grid dissapears on switch left	grid shows on switch grid dissapears on switch	Pass Pass
11		handleShowDayNightChange()	toggles the visibility of the day/night ter corresponding state variable.	N/A	1. Mount application 2. Click Map Setting Icon (wrench) 3. Click "DayNight" to show 4. Click "DayNight" to Close	terminator shows on switch right terminator dissapears on switch left	terminator shows on switch terminator dissapears on switch	Pass Pass
12		handleShowGroundTrackChan	toggles the visibility of the ground track graphic is not null.	N/A	1. Mount application 2. Type in known satellite in search bar 3. Click on suggestion 2. Click Map Setting Icon (wrench) 3. Click "Ground Track" to show 4. Click "Grid" to Close	Ground track shows on switch right Ground track dissapears on switch left	Ground track shows on switch right Ground track dissapears on switch lef	Pass Pass
13	Map Helper	getGround()	generates a coordinate path to compute satellite based on given coordinates.	Longitude =100 Latitude = 45 Longitude = 500 Latitude = -2993	1. Run MapHelper.test.js	Coordinates of polyline that intersect with Satellite "Out of Range" Error	Coordinates of polyline that intersect with Satellite "Out of Range" Error	Pass Pass
14		getGrid()	generates a grid of coordinates with sp and longitude.	N/A		Coordinates of Latitude/Longitude at 15 degree interval	Coordinates of Latitude/Longitude at degree interval	Pass
15		getDayNight()	calculates and returns the coordinates separating day and night) on Earth at t	N/A		Coordinates of day/night terminator retu	Coordinates of day/night terminator re	Pass
16		renderGrid()	renders a grid on a map viewport using returns the grid graphic.	N/A		Grid shows on map	Grid shows on map	Pass
17		renderDayNight()	creates and adds a day/night graphic to	N/A		Terminator shows on map	Terminator shows on map	Pass
18		renderGroundTrack()	renders a ground track graphic on an E longitude and latitude coordinates.	Longitude =100 Latitude = 45 Longitude = 500 Latitude = -2993		Polyline rendered intersecting (100,45) "Out of Range" Error	Polyline rendered intersecting (100,45 "Out of Range" Error	Pass Pass
19		renderObserver()	renders an observer graphic on an Esri longitude and latitude coordinates, and	Longitude =100 Latitude = 45 Longitude = 500 Latitude = -2993		Marker rendered at (100,45) "Out of Range" Error	Marker rendered at (100,45) "Out of Range" Error	Pass Pass
20		renderSatellite()	enders a satellite on a map view using creating a graphic with a specific symt view's graphics layer.	Longitude =100 Latitude = 45 Longitude = 500 Latitude = -2993		Satellite Icon rendered at (100,45) "Out of Range" Error	Satellite Icon rendered at (100,45) "Out of Range" Error	Pass Pass
21	Helper	getSatellite()	retrieves orbital data from Celestrak for name, constructs a satellite object with name, time, longitude, latitude, altitude, orbital data (OMM), and returns the sat	Name = ISS (Zarya) Name = aksfhsdf	1. Run Helper.test.js	console.log(Satellite != null) => true console.log(Satellite != null) => false	true false	Pass Pass
22		getRotator()	retrieves rotator information from a give constructs a rotator object with relevar IP address, call sign, model, local time look angles (initialized as null), and trac rotator object; if the IP address is inval appropriate error messages are return	"slakdgo" "102.2.4.2.4" "123.123.123.123" "192.168.50.100"		console.log(Rotator != null) => false console.log(Rotator != null) => false console.log(Rotator != null) => false console.log(Rotator != null) => true	false false false true	Pass Pass Pass Pass
23		fetchOrbitalData()	fetches orbital data for a satellite from specified format (either TLE or JSON), the satellite name and format, sends a returns the fetched data as either TLE depending on the format requested; an the fetch are logged to the console.	Name = ISS (Zarya) Format = "JSON" Name = aksfhsdf Fortmat = sdfgs		OBJECT_NAME: "ISS (ZARYA)" OBJECT_ID: "1998-067A" EPOCH: "2023-04-25T12:46:22.350432" MEAN_MOTION: 15.50274552 ECCENTRICITY: 0.0005628 INCLINATION: 51.6404 RA_OF_ASC_NODE: 229.4846 ARG_OF_PERICENTER: 240.7961 MEAN_ANOMALY: 207.2997 EPHEMERIS_TYPE: 0 CLASSIFICATION_TYPE: "U" NORAD_CAT_ID: 25544 ELEMENT_SET_NO: 999 REV_AT_EPOCH: 39362 BSTAR: 0.00048547 MEAN_MOTION_DOT: 0.00027629 MEAN_MOTION_DDOT: 0 Error	OBJECT_NAME: "ISS (ZARYA)" OBJECT_ID: "1998-067A" EPOCH: "2023-04-25T12:46:22.350432" MEAN_MOTION: 15.50274552 ECCENTRICITY: 0.0005628 INCLINATION: 51.6404 RA_OF_ASC_NODE: 229.4846 ARG_OF_PERICENTER: 240.7961 MEAN_ANOMALY: 207.2997 EPHEMERIS_TYPE: 0 CLASSIFICATION_TYPE: "U" NORAD_CAT_ID: 25544 ELEMENT_SET_NO: 999 REV_AT_EPOCH: 39362 BSTAR: 0.00048547 MEAN_MOTION_DOT: 0.00027629 MEAN_MOTION_DDOT: 0 Error	Pass Pass
24		fetchJson()	fetches JSON data from a specified IP request type, with an optional payload, data; it includes a timeout mechanism longer than 10 seconds, and throws ar invalid.	"slakdgo" "102.2.4.2.4" "123.123.123.123" "192.168.50.100"		"Invalid IP Error" "Invalid IP Error" "IP Timeout Error" Name: "David's Rotator" IP: "192.168.50.100" Call_Sign: "W5QZ" Model: "Yaesu G-5500" Time: "21:58:18" Longitude: "30.6280" Latitude: "-96.3344" ObserverGd: Object Look_Angles: null isTracking: false Template: Object	"Invalid IP Error" "Invalid IP Error" "IP Timeout Error" Name: "David's Rotator" IP: "192.168.50.100" Call_Sign: "W5QZ" Model: "Yaesu G-5500" Time: "21:58:18" Longitude: "30.6280" Latitude: "-96.3344" ObserverGd: Object Look_Angles: null isTracking: false Template: Object	Pass Pass Pass Pass
25		getPositionGd()	calculates the geodetic coordinates (latitude, longitude, and altitude) of a sa surface based on its Earth-Centered In the current date and time.	x: 3049.86548, y: -472.53814, z: -6332.4052		longitude: -0.84379 latitude: 0.23418 height: 35856.7679	longitude: -0.84379 latitude: 0.23418 height: 35856.7679	Pass
26		getSatRec()	takes a Two-Line Element (TLE) string the line 1 and line 2 TLE data, converts object using the twoline2satrec library,	RIGIDSPHERE 2 (LCS 4) 05398U 71067E 23115.5114748 0001221 00000+0 41465-3 0 99 5398 87.6149 296.8478 006482 852 284.6547 14.347658967089		{error: 0, satnum: "05398", epochyr: 23, epochdays: 115.51147481, ndot: 0.00001221…}	{error: 0, satnum: "05398", epochyr: 23, epochdays: 115.51147481, ndot: 0.00001221…}	Pass
27		getLookAngles()	calculates and returns the azimuth, ele of a satellite with respect to a given obs (observerGd), using the satellite record between Earth-centered inertial (ECI) a coordinate systems.	{error: 0, satnum: "05398", epochyr: 23, epochdays: 115.51147481, ndot: 0.00001221..		longitude: 2.7705218839347348, latitude: -0.03913181951360097, height: 768.8038068041697}	longitude: 2.7705218839347348, latitude: -0.03913181951360097, height: 768.8038068041697}	Pass
28		isValid()	takes an input string representing an IP and uses a regular expression to valida returning a boolean value indicating its	"slakdgo" "102.2.4.2.4" "123.123.123.123" "192.168.50.100"		false false true true	false false true true	Pass Pass Pass Pass

29		getLocalTime()	calculates and returns the local time at (in degrees), by adjusting the current difference from Greenwich Mean Time object representing the local time.	N/A		Time: "22:04:16"	Time: "22:04:16"	Pass
30		getTimeUTC()	calculates and returns the current Coor Time (UTC) based on the Greenwich M and the observer's geographic coordina the current date and time to account fo Date object representing the UTC time.	N/A		12:52:55	12:52:55	Pass
31	Dropdown Menu	handleToggleMenu()	handles a menu toggle event and displ the control menu based on whether a been selected.	Satellite = null Rotator = null Satellite != null Rotator = null Satellite = null Rotator != null Satellite != null Rotator != null	1. Reach the test case state by connecting to rotator/selecting satellite 2. Click Control Menu button in the top left corner (Slider Icon)	" Please select a Satellite and connect a rotator to begin tracking" " Please select a Satellite and connect a rotator to begin tracking" " Please select a Satellite and connect a rotator to begin tracking" Control Menu Opens	" Please select a Satellite and connect a rotator to begin tracking" " Please select a Satellite and connect a rotator to begin tracking" " Please select a Satellite and connect a rotator to begin tracking" Control Menu Opens	Pass Pass Pass Pass
32		handleCloseMenu()	handles the event to close a control me and error messages related to azimuth	Azimuth = 120 Elevation = 10 Azimuth = 1210 Elevation = 110	1. Connect a rotator and choose a satellite 2. Click Control Menu button in the top left corner (Slider Icon) 3. enter test case into text field 4. click away from control menu	Azimuth = 120 Elevation = 10 Azimuth = "" Elevation = ""	Azimuth = 120 Elevation = 10 Azimuth = "" Elevation = ""	Pass Pass
33		handleCommand()	sends a start or stop tracking command whether the tracking is already on or of success or error message accordingly. of the satellite object or azimuth/elevat whether auto or manual tracking mode	isTracking = False command = "Start" isTracking = False command = "Stop" isTracking = False command = "" isTracking = True command = "Start"	1. Connect a rotator and choose a satellite 2. Click Control Menu button in the top left corner (Slider Icon) 3. click "autoTrack" 4. reach test case state by using commands	"Rotator has started auto-tracking" Client Request:: StartAuto "Rotator has stopped tracking" Client Request:: Stop "HTTP/1.1 404 Not Found" No Change	"Rotator has started auto-tracking" Client Request:: StartAuto "Rotator has stopped tracking" Client Request:: Stop "HTTP/1.1 404 Not Found" No Change	Pass Pass Pass Pass
34		handleCloseSnack()	clears the snack message popup and c	N/A	1. Click Control Menu button in the top left corner (Slider Icon)	Snack bar message displays and disap	Snack bar message displays and disa	Pass
35		handleAzimuth()	validates and sets the azimuth input va if it is out of range.	Azimuth = 120 Elevation = 10 Azimuth = 1210 Elevation = 110	1. Connect a rotator and choose a satellite 2. Click Control Menu button in the top left corner (Slider Icon) 3. enter test case	True False	True False	Pass Pass
36		handleElevation()	validates and sets the elevation input v if it is out of range.	Azimuth = 120 Elevation = 10 Azimuth = 1210 Elevation = 11330	1. Connect a rotator and choose a satellite 2. Click Control Menu button in the top left corner (Slider Icon) 3. enter test case	True False	True False	Pass Pass
37		handleToggleAutoTrack()	toggles the state of the tracking control and elevation input values.	isTracking = False Azimuth = 120 Elevation = 10 isTracking = True	1. Connect a rotator and choose a satellite 2. Click Control Menu button in the top left corner (Slider Icon) 3. click Autotrack Switch	AutoTrack Switch Toggles On Azimuth = "" Elevation = "" AutoTrack Switch Toggles Off	AutoTrack Switch Toggles On Azimuth = "" Elevation = "" AutoTrack Switch Toggles Off	Pass Pass

APPENDIX A: Application Code

1.index.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <link
      rel="icon"
      href="./src/Components/Assets/favicon.ico"
      type="image/x-icon"
    />
    <title>Satellite Tracker</title>
  </head>
  <body>
    <div id="root"></div>
    <script src="./src/App.js"></script>
  </body>
</html>
```

2. App.js

```
import React, { useState } from "react";
import { createRoot } from "react-dom/client";
import DropdownMenu from "./Components/DropdownMenu";
import SearchBar from "./Components/SearchBar";
import RotatorConnector from "./Components/RotatorConnector";
import MercatorMap from "./Components/MercatorMap";
import { AppBar, Toolbar } from "@material-ui/core";

// render main app components; toolbar and the map
function App() {
  //intialize state object variables
  const [Satellite, setSatellite] = useState(null);
  const [Rotator, setRotator] = useState(null);

  return (
    <div>
      <AppBar
        position="fixed"
        color="default"
        elevation={1}
        style={{ minHeight: 64 }}
      >
        <Toolbar style={{ flexGrow: 1, justifyContent: "space-between" }}>
          <div style={{ display: "flex", flex: 1 }}>
            <DropdownMenu Rotator={Rotator} Satellite={Satellite} />
            <SearchBar
              Rotator={Rotator}
              Satellite={Satellite}
              setSatellite={setSatellite}
            />
          </div>
          <div style={{ justifyContent: "flex-end" }}>
            <RotatorConnector
              setRotator={setRotator}
              Rotator={Rotator}
              Satellite={Satellite}
            />
          </div>
        </Toolbar>
      </AppBar>
```

```
      <MercatorMap Rotator={Rotator} Satellite={Satellite} />
    </div>
  );
}

createRoot(document.getElementById("root")).render(<App />);
```

3. SearchBar.js

```
import React, { useState } from "react";
import satellites from "./Satellites.json";
import { getSatellite, getLookAngles } from "./Helper";

import {
  InputBase,
  List,
  ListItem,
  ListItemText,
  ClickAwayListener
} from "@material-ui/core";

function SearchBar({ setSatellite, Satellite, Rotator }) {
  const [searchText, setSearchText] = useState("");
  const [suggestions, setSuggestions] = useState([]);

  const handleSearchTextChange = (event) => {
    // save search input
    const newSearchText = event.target.value;
    setSearchText(newSearchText);

    // compute new suggestions based on the search input
    const newSuggestions = satellites
      .filter((satellite) =>
        satellite.OBJECT_NAME.toLowerCase().includes(
          newSearchText.toLowerCase()
        )
      )
      .slice(0, 5);
    setSuggestions(newSuggestions);
  };

  const handleSuggestionClick = (suggestion) => async () => {
    // clear suggestions
    setSuggestions([]);

    // update satellite object
    const satellite = await getSatellite(suggestion.OBJECT_NAME);

    if (typeof satellite === "object") {
```

```
    // set look angles
    if (Boolean(Rotator) && Rotator.Look_Angles == null) {
      Rotator.Look_Angles = getLookAngles(
        Rotator.ObserverGd,
        satellite.SatRec
      );
    }
    //set satellite
    await setSatellite(satellite);
    console.log("Satellite Selected: " + satellite.Name);
    setSearchText(suggestion.OBJECT_NAME);
  }
};

const handleClickAway = () => {
  // clear suggestions
  setSuggestions([]);

  // clear search bar if user didn't choose a satellite
  if (Satellite == null) {
    setSearchText("");
  }
};

return (
  <ClickAwayListener onClickAway={handleClickAway}>
    <div style={{ width: "50%", position: "relative" }}>
      <InputBase
        placeholder="Search for satellite... "
        value={searchText}
        onChange={handleSearchTextChange}
        style={{
          margin: "0 0 0 25px",
          padding: "0 0 0 5px",
          borderBottom: "1px solid grey",
          width: "100%",
          display: "flex"
        }}
      />
      {suggestions.length > 0 && (
        <List
          style={{
            position: "absolute",
```

```
              top: "70px",
              left: "25px",
              zIndex: 1,
              backgroundColor: "#fff",
              borderRadius: 4,
              boxShadow: "0px 8px 16px 0px rgba(0,0,0,0.2)",
              width: "100%",
              padding: 0,
              margin: 0
            }}
          >
            {suggestions.map((suggestion) => (
              <ListItem
                button
                key={suggestion.OBJECT_NAME}
                onClick={handleSuggestionClick(suggestion)}
              >
                <ListItemText primary={suggestion.OBJECT_NAME} />
              </ListItem>
            ))}
          </List>
        )}
      </div>
    </ClickAwayListener>
  );
}

export default SearchBar;
```

4. RotatorConnector.js

```
import React, { useState } from "react";
import { getRotator, getLookAngles } from "./Helper";
import {
  Button,
  Dialog,
  DialogTitle,
  DialogContent,
  DialogActions,
  TextField,
  CircularProgress
} from "@material-ui/core";
import RotatorIcon from "./Assets/RotatorIcon";
function RotatorConnector({ setRotator, Satellite, Rotator }) {
  const [connected, setConnected] = useState(false);
  const [openDialog, setOpenDialog] = useState(false);
  const [ipAddress, setIpAddress] = useState("");
  const [loading, setLoading] = useState(false);
  const [error, setError] = useState("");

  const handleConnect = async () => {
    setError("");
    // begin connection procedure
    setLoading(true);

    // attempt to connect to submitted ip address
    const rotator = await getRotator(ipAddress);

    if (typeof rotator === "object") {
      //set looking angles
      if (Boolean(Satellite) && rotator.Look_Angles == null) {
        rotator.Look_Angles = getLookAngles(
          rotator.ObserverGd,
          Satellite.SatRec
        );
      }

      //set rotator object
      setRotator(rotator);
      console.log("Rotator Connected: " + rotator.Name);
      console.log(rotator);
```

```
    // display success message and close Dialog Box
    setConnected(true);
    setLoading(false);
    setTimeout(() => {
      setOpenDialog(false);
    }, 1000);
  } else {
    setIpAddress("");
    setError(rotator);
    setLoading(false);
  }
};

const handleCancel = () => {
  // clear IP address input field
  setIpAddress("");
  // close dialog Box
  setOpenDialog(false);
  // clear errors
  setError("");
};

const handleClick = () => {
  if (!connected) {
    // open Dialog Box
    setOpenDialog(true);
  } else if (Rotator.isTracking) {
    // prevent user from disconnecting while tracking
    console.log("Please stop tracking before disconnecting rotator");
  } else {
    // disconnect from rotator
    setConnected(false);
    setRotator(null);
    setIpAddress("");
  }
};

const handleInputChange = (event) => {
  setIpAddress(event.target.value);
};

return (
```

```
<div
  style={{
    display: "flex",
    alignItems: "center",
    justifyContent: "flex-end"
  }}
>
  <Button
    variant="contained"
    color={connected ? "secondary" : "primary"}
    onClick={handleClick}
    disabled={loading}
  >
    {connected ? "Disconnect" : "Connect Rotator"}
  </Button>

  <Dialog
    open={openDialog}
    onClose={handleCancel}
    aria-labelledby="form-dialog-title"
  >
    {!loading & !connected ? <RotatorIcon /> : ""}

    {!connected && (
      <DialogTitle id="form-dialog-title">
        {loading ? "Connecting..." : "Connect to Rotator"}
      </DialogTitle>
    )}
    <DialogContent>
      {connected && "Connection Successful!"}
      {!loading && !connected && (
        <TextField
          autoFocus
          margin="dense"
          label="IP Address"
          value={ipAddress}
          onChange={handleInputChange}
          fullWidth
          error={Boolean(error)}
          helperText={error}
          InputProps={{
            style: {
              borderColor: error ? "red" : ""
```

```
                }
              }}
            />
          )}
        </DialogContent>
        <DialogActions
          style={{ justifyContent: "center", alignItems: "center" }}
        >
          {loading && <CircularProgress />}
          {!loading && !connected && (
            <>
              <Button onClick={handleCancel} color="primary">
                Cancel
              </Button>
              <Button onClick={handleConnect} color="primary">
                Connect
              </Button>
            </>
          )}
        </DialogActions>
      </Dialog>
    </div>
  );
}

export default RotatorConnector;
```

5. MercatorMap.js

```
import React, { useState, useEffect, useRef } from "react";
import { createRoot } from "react-dom/client";
import { loadModules } from "esri-loader";
import { Button } from "@material-ui/core";
import { Build as BuildIcon } from "@material-ui/icons";
import {
  renderObserver,
  renderSatellite,
  renderGrid,
  renderDayNight,
  renderGroundTrack
} from "./MapHelper";
import MapSettings from "./MapSettings";
import "./Map.css";

function MercatorMap({ Satellite, Rotator }) {
  const MapElement = useRef(null);
  const [anchorEl, setAnchorEl] = useState(null);
  const [gridGraphic, setGridGraphic] = useState(null);
  const [groundTrackGraphic, setGroundTrackGraphic] = useState(null);
  const [dayNightGraphic, setDayNightGraphic] = useState(null);
  const [showGroundTrack, setShowGroundTrack] = useState(false);

  useEffect(() => {
    let view;

    loadModules(
      [
        "esri/config",
        "esri/views/MapView",
        "esri/Map",
        "esri/Graphic",
        "esri/widgets/Popup",
        "esri/geometry/Mesh",
        "esri/layers/GraphicsLayer"
      ],
      {
        css: true
      }
    ).then(([esriConfig, MapView, Map, Graphic]) => {
```

```
      esriConfig.apiKey =

"AAPK74b83cbed7c040a4912534f0a31191eccop4MGikdDa5odq5tNJeSsCrLgGBy2Tp_I_7cO
fpmGxeEfDZbPxCOi2xlOx9o7AF";
      // create map
      const map = new Map({
        basemap: "hybrid"
      });

      view = new MapView({
        container: MapElement.current,
        map: map,
        zoom: 4,
        constraints: {
          minZoom: 4
        }
      });

      //setup Grid
      renderGrid(view).then((graphic) => {
        setGridGraphic(graphic);
      });

      //setup Day/Night Terminator
      renderDayNight(view).then((graphic) => {
        setDayNightGraphic(graphic);
      });

      //setup rotator
      if (Rotator !== null) {
        var point = renderObserver(view, Rotator);
      }

      // setup groundtrack and satellite
      if (Satellite !== null) {
        var point = renderSatellite(view, Satellite);
        renderGroundTrack(view, Satellite.Longitude,
Satellite.Latitude).then(
          (graphic) => {
            setShowGroundTrack(true);
            setGroundTrackGraphic(graphic);
          }
        );
```

```
      }

      // open/close map settings popup
      const handleToggleMapSettings = (event) => {
        setAnchorEl(event.currentTarget);
      };

      // create and add map settings popup to viewport
      var node = document.createElement("div");
      view.ui.add(node, "top-right");

      createRoot(node).render(
        <Button
          onClick={handleToggleMapSettings}
          style={{ backgroundColor: "white", width: "34px", minWidth: "0px"
}}
        >
          <BuildIcon style={{ width: "100%" }} />
        </Button>
      );

      // move the zoom buttons to the top right
      view.ui.move("zoom", "top-right");
    });

    return () => {
      if (view) {
        view.destroy();
      }
    };
  }, [Satellite, Rotator]);

  return (
    <div
      style={{
        height: "calc(100% - 64px)",
        width: "100%",
        position: "absolute",
        top: 64,
        left: 0
      }}
      ref={MapElement}
    >
```

```
      <div style={{ position: "absolute", right: "100px" }}>
        <MapSettings
          anchorEl={anchorEl}
          setAnchorEl={setAnchorEl}
          gridGraphic={gridGraphic}
          groundTrackGraphic={groundTrackGraphic}
          dayNightGraphic={dayNightGraphic}
          showGroundTrack={showGroundTrack}
          setShowGroundTrack={setShowGroundTrack}
        />
      </div>
    </div>
  );
}

export default MercatorMap;
```

6. MapSettings.js

```
import React, { useState } from "react";
import {
  Popover,
  List,
  ListItem,
  ListItemText,
  Switch
} from "@material-ui/core";

function MapSettings({
  anchorEl,
  setAnchorEl,
  gridGraphic,
  dayNightGraphic,
  groundTrackGraphic,
  showGroundTrack,
  setShowGroundTrack
}) {
  const open = Boolean(anchorEl);
  const [showGrid, setShowGrid] = useState(false);
  const [showDayNight, setShowDayNight] = useState(false);
```

```
const handleClose = () => {
  setAnchorEl(null);
};

const handleShowGridChange = () => {
  // change grid layer visibility
  setShowGrid(!showGrid);
  gridGraphic.visible = !gridGraphic.visible;
};

const handleShowDayNightChange = () => {
  // change day/night terminator layer visibility
  setShowDayNight(!showDayNight);
  dayNightGraphic.visible = !dayNightGraphic.visible;
};

const handleShowGroundTrackChange = (event) => {
  // change ground track layer visibility is satellite is chosen
  if (groundTrackGraphic != null) {
    setShowGroundTrack(!showGroundTrack);
    groundTrackGraphic.visible = !groundTrackGraphic.visible;
  }
};

return (
  <Popover
    open={open}
    anchorEl={anchorEl}
    onClose={handleClose}
    anchorOrigin={{
      vertical: "top",
      horizontal: "left"
    }}
    transformOrigin={{
      vertical: "top",
      horizontal: "left"
    }}
    style={{ marginLeft: "-50px" }}
  >
    <List>
      <ListItem>
        <ListItemText primary="Grid" />
```

```
          <Switch
            checked={showGrid}
            onChange={handleShowGridChange}
            color="primary"
          />
        </ListItem>
        <ListItem>
          <ListItemText primary="Day/Night" />
          <Switch
            checked={showDayNight}
            onChange={handleShowDayNightChange}
            color="primary"
          />
        </ListItem>
        <ListItem>
          <ListItemText primary="Ground Track" />
          <Switch
            checked={showGroundTrack}
            onChange={handleShowGroundTrackChange}
            color="primary"
          />
        </ListItem>
      </List>
    </Popover>
  );
}

export default MapSettings;
```

7. MapHelper.js

```
import { loadModules } from "esri-loader";
const SatelliteIcon = require("./Assets/SatelliteIcon.png");
const CubeSatIcon = require("./Assets/CubeSat.png");
const MarkerIcon = require("./Assets/MarkerIcon.png");

////////////////////////////////////////////////////////////
//////////////            Path Creators           //////////////
////////////////////////////////////////////////////////////

export function getGround(x1, y1) {
  try {
    if (x1 < -180 || x1 > 180 || y1 < -90 || y1 > 90) {
      throw new Error("Latitude/Longitude Out of Range");
    }
    const paths = [];
    //compute phase needed to intersect with satellite
    const b = Math.acos(y1 / 90) * (180 / Math.PI) - 1 * x1;
    var y;
    // generate coordinate path
    for (let i = 0; i < 1000; i++) {
      const x = -180 + (i * 360) / 1000;
      if (Math.abs(y1) < 30) {
        y = 30 * Math.cos((1 * x + b) * (Math.PI / 180));
      } else if (Math.abs(y1) < 60) {
        y = 60 * Math.cos((1 * x + b) * (Math.PI / 180));
      } else {
        y = 70 * Math.cos((1 * x + b) * (Math.PI / 180));
      }
      paths.push([x, y]);
    }
    return paths;
  } catch (error) {
    console.log(error);
    return null;
  }
}

export function getGrid() {
  const grid = [];
```

```
  // generate latitudes
  for (let lat = -75; lat <= 75; lat += 15) {
    const line = [];
    for (let lon = -180; lon <= 180; lon += 1) {
      line.push([lon, lat]);
    }
    grid.push(line);
  }

  // generate longitudes
  for (let lat = -180; lat <= 180; lat += 15) {
    const line = [];
    for (let lon = -180; lon <= 180; lon += 15) {
      line.push([lat, lon]);
    }
    grid.push(line);
  }

  return grid;
}

export function getDayNight() {
  const numPoints = 100;
  const points = [];

  // get the current date and time
  const now = new Date();

  // calculate the current julian date
  const julianDate =
    now.getTime() / 86400000 - now.getTimezoneOffset() / 1440 + 2440587.5;

  // calculate the position of the Sun
  const meanLongitude = (280.46 + 0.9856474 * julianDate) % 360;
  const meanAnomaly = 357.528 + 0.9856003 * julianDate;
  const eclipticLongitude =
    meanLongitude +
    1.915 * Math.sin((meanAnomaly * Math.PI) / 180) +
    0.02 * Math.sin((2 * meanAnomaly * Math.PI) / 180);
  const obliquity = 23.439 - 0.0000004 * julianDate;
  const rightAscension =
    (Math.atan2(
      Math.cos((obliquity * Math.PI) / 180) *
```

```
        Math.sin((eclipticLongitude * Math.PI) / 180),
      Math.cos((eclipticLongitude * Math.PI) / 180)
    ) *
      180) /
    Math.PI;
  const declination =
    (Math.asin(
      Math.sin((obliquity * Math.PI) / 180) *
        Math.sin((eclipticLongitude * Math.PI) / 180)
    ) *
      180) /
    Math.PI;

  // calculate the position of the terminator
  const latitudes = [];
  const longitudes = [];

  //add boundary
  latitudes.push(90);
  longitudes.push(-180);
  latitudes.push(90);
  longitudes.push(180);

  for (let i = 0; i <= numPoints; i++) {
    const theta = (2 * Math.PI * i) / numPoints;
    const longitude = rightAscension - (theta * 180) / Math.PI;
    const latitude =
      (Math.atan(-Math.cos(theta) / Math.tan((declination * Math.PI) /
180)) *
        180) /
      Math.PI;
    latitudes.push(latitude);
    longitudes.push(longitude);
  }

  // construct coordinates array
  for (let i = 0; i < numPoints; i++) {
    points.push([longitudes[i], latitudes[i]]);
  }
  return points;
}

//////////////////////////////////////////////////////////////
```

```
//////////////        Component Renders           //////////////
////////////////////////////////////////////////////////////////

export function renderGrid(view) {
  return loadModules(["esri/Graphic"], {
    css: true
  }).then(([Graphic]) => {
    // create line geometry
    const polyline = {
      type: "polyline",
      paths: getGrid()
    };

    // create symbol
    const lineSymbol = {
      type: "simple-line",
      color: [100, 200, 200],
      width: 1
    };

    // create graphic
    const gridGraphic = new Graphic({
      geometry: polyline,
      symbol: lineSymbol,
      visible: false
    });

    // add graphic to viewport
    view.graphics.add(gridGraphic);

    return gridGraphic;
  });
}

export function renderDayNight(view) {
  return loadModules(["esri/Graphic"], {
    css: true
  }).then(([Graphic]) => {
    // create polygon geometry
    const polygon = {
      type: "polygon",
      rings: getDayNight()
    };
```

```
    // create symbol
    const simpleFillSymbol = {
      type: "simple-fill",
      color: [1, 1, 1, 0.2],
      outline: {
        color: [1, 1, 1, 0.0],
        width: 1
      }
    };

    // create graphic
    const dayNightGraphic = new Graphic({
      geometry: polygon,
      symbol: simpleFillSymbol,
      visible: false
    });

    // add graphic to viewport
    view.graphics.add(dayNightGraphic);

    return dayNightGraphic;
  });
}

export function renderGroundTrack(view, Longitude, Latitude) {
  return loadModules(["esri/Graphic"], {
    css: true
  }).then(([Graphic]) => {
    // create line geometry
    const polyline = {
      type: "polyline",
      paths: getGround(Longitude, Latitude)
    };

    // create symbol
    const lineSymbol = {
      type: "simple-line",
      color: [226, 119, 40],
      width: 4
    };

    // create graphic
```

```
    const groundTrackGraphic = new Graphic({
      geometry: polyline,
      symbol: lineSymbol
    });

    // add graphic to viewport
    view.graphics.add(groundTrackGraphic);

    return groundTrackGraphic;
  });
}

export function renderObserver(view, Rotator) {
  return loadModules(["esri/Graphic"], {
    css: true
  }).then(([Graphic]) => {
    // Plot Observer
    const point = {
      type: "point",
      longitude: parseFloat(Rotator.Longitude),
      latitude: parseFloat(Rotator.Latitude)
    };

    // create symbol
    const markerSymbol = {
      type: "picture-marker",
      url: MarkerIcon,
      width: "60px",
      height: "40px"
    };

    // create graphic
    const Observer = new Graphic({
      geometry: point,
      symbol: markerSymbol,
      attributes: Rotator,
      popupTemplate: Rotator.Template
    });

    // add graphic to viewport
    view.graphics.addMany([Observer]);

    return point;
```

```
  });
}

export function renderSatellite(view, Satellite) {
  return loadModules(["esri/Graphic"], {
    css: true
  }).then(([Graphic]) => {
    // create coordinate point
    const point = {
      type: "point",
      longitude: Satellite.Longitude,
      latitude: Satellite.Latitude
    };
    // create symbol
    let markerSymbol = null;

    if (Satellite.Name.includes("CUBE")) {
      markerSymbol = {
        type: "picture-marker",
        url: CubeSatIcon,
        width: "300",
        height: "100"
      };
    } else {
      markerSymbol = {
        type: "picture-marker",
        url: SatelliteIcon,
        width: "200",
        height: "300"
      };
    }

    // create graphic
    const SatGraphic = new Graphic({
      geometry: point,
      symbol: markerSymbol,
      attributes: Satellite,
      popupTemplate: Satellite.Template
    });

    // add graphic to viewport
    view.graphics.addMany([SatGraphic]);
    return point;
```

```
  });
}
```

8. Helper.js

```
import {
  ecfToLookAngles,
  propagate,
  gstime,
  twoline2satrec,
  eciToEcf,
  eciToGeodetic,
  degreesLong,
  degreesLat
} from "satellite.js";

////////////////////////////////////////////////////////////
////////    Satellite/Rotator Object Constructors     ////////
////////////////////////////////////////////////////////////
export async function getSatellite(satelliteName) {
  // retrieve orbital data from Celestrak
  const OMM = await fetchOrbitalData(satelliteName, "JSON");
  const TLE = await fetchOrbitalData(satelliteName, "TLE");
  const satRec = getSatRec(TLE);
  const positionGd = getPositionGd(satRec);

  // create popup template
  const temp = {
    title: "{Name}",
    content: [
      {
        type: "fields",
        fieldInfos: [
          {
            fieldName: "Longitude",
            label: "Longitude",
            format: {
              digitSeparator: true,
              places: 0
            }
```

```
        },
        {
          fieldName: "Latitude",
          label: "Latitude",
          format: {
            digitSeparator: true,
            places: 0
          }
        },
        {
          fieldName: "Altitude",
          label: "Altitude",
          format: {
            digitSeparator: true,
            places: 0
          }
        },
        {
          fieldName: "Time",
          label: "Time (UTC)",
          format: {
            digitSeparator: true,
            places: 0
          }
        }
      ]
    }
  ]
};

//Construct satellite object
const satellite = {
  Name: satelliteName,
  Time: getTimeUTC(positionGd),
  Longitude: degreesLong(positionGd.longitude),
  Latitude: degreesLat(positionGd.latitude),
  Altitude: positionGd.height,
  SatRec: satRec,
  Omm: OMM,
  Template: temp
};

return satellite;
```

```
}

export async function getRotator(IP) {
  if (isValid(IP)) {
    // retrieve rotator data
    const connectResponse = await fetchJson(IP, "Connect");
    if (connectResponse !== undefined) {
      // create popup template
      const temp = {
        title: "Observer: {Call_Sign}",
        content: [
          {
            type: "fields",
            fieldInfos: [
              {
                fieldName: "Longitude",
                label: "Longitude",
                format: {
                  digitSeparator: true,
                  places: 0
                }
              },
              {
                fieldName: "Latitude",
                label: "Latitude",
                format: {
                  digitSeparator: true,
                  places: 0
                }
              },
              {
                fieldName: "Model",
                label: "Model",
                format: {
                  digitSeparator: true,
                  places: 0
                }
              },
              {
                fieldName: "isTracking",
                label: "Tracking",
                format: {
                  digitSeparator: true,
```

```
                places: 0
              }
            },
            {
              fieldName: "Time",
              label: "Local Time",
              format: {
                digitSeparator: true,
                places: 0
              }
            }
          ]
        }
      ]
    };

    // construct rotator object
    const rotator = {
      Name: connectResponse.NAME,
      IP: connectResponse.IP,
      Call_Sign: connectResponse.CALL_SIGN,
      Model: connectResponse.MODEL,
      Time: getLocalTime(connectResponse.LONGITUDE),
      Longitude: connectResponse.LONGITUDE,
      Latitude: connectResponse.LATITUDE,
      ObserverGd: {
        longitude: connectResponse.LONGITUDE,
        latitude: connectResponse.LATITUDE,
        height: connectResponse.HEIGHT
      },
      Look_Angles: null,
      isTracking: false,
      Template: temp
    };

    return rotator;
  } else {
    return "Request has Timed Out. Please check IP address.";
  }
} else {
  return "Invalid IP address. Please enter a valid IP address.";
}
}
```

```
////////////////////////////////////////////////////////////
//////////////            Data Fetching          /////////////
////////////////////////////////////////////////////////////
function fetchOrbitalData(satelliteName, format) {
  // contruct url from satellite name and format
  const url =
`https://www.celestrak.com/NORAD/elements/gp.php?NAME=${satelliteName}&FORM
AT=${format}`;
  // request document
  return fetch(url)
    .then((response) => response.text())
    .then((data) => {
      if (format === "TLE") {
        // retirn tle string
        return data;
      } else {
        // parse/return JSON into Object
        console.log(JSON.parse(data)[0]);
        return JSON.parse(data)[0];
      }
    })
    .catch((error) => {
      console.error(error);
    });
}

export async function fetchJson(IP, requestType, payload) {
  try {
    if (isValid(IP)) {
      // verify that the ip doesnt timeout
      const timeoutPromise = new Promise((_, reject) =>
        setTimeout(() => reject(new Error("Request timed out")), 10000)
      );

      // construct url
      var url = "";

      if (Boolean(payload)) {
        url = `https://${IP}/${requestType}/r/${payload}`;
      } else {
        url = `https://${IP}/${requestType}/r`;
      }
```

```
      // request json from mcu server
      const response = await Promise.race([fetch(url), timeoutPromise]);

      const data = await response.json();
      console.log(data);
      return data;
    } else {
      // invalid IP address error
      throw new Error("Invalid IP Address");
    }
  } catch (error) {
    console.error(error);
  }
}

//////////////////////////////////////////////////////////////
//////////       Orbital Dynamics/Other          ////////////
//////////////////////////////////////////////////////////////
export function getPositionGd(satRec) {
  // get time
  const gmst = gstime(new Date());

  // propgate position
  const positionEci = propagate(satRec, new Date()).position;
  console.log(eciToGeodetic(positionEci, gmst));
  // convert/return geodetic coordinates
  return eciToGeodetic(positionEci, gmst);
}

function getSatRec(tle) {
  console.log(tle);
  // parse tle lines
  var line1 = tle.split("\n")[1];
  var line2 = tle.split("\n")[2];

  // create satrec object
  const satrec = twoline2satrec(line1, line2);
  console.log(satrec);
  return satrec;
}

export function getLookAngles(observerGd, satRec) {
  // get look angles
```

```
  const gmst = gstime(new Date());
  const positionEci = propagate(satRec, new Date()).position;
  const positionEcf = eciToEcf(positionEci, gmst);
  const lookAngles = ecfToLookAngles(observerGd, positionEcf);
  console.log({
    azimuth: lookAngles.azimuth,
    elevation: lookAngles.elevation,
    rangeSat: lookAngles.rangeSat
  });
  //return ObserverGd Object
  return {
    azimuth: lookAngles.azimuth,
    elevation: lookAngles.elevation,
    rangeSat: lookAngles.rangeSat
  };
}

function isValid(IP) {
  // verify that IP address is the format: [0-256].[0-256].[0-256].[0-256]
  const regex =
/^(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.){3}(?:25[0-5]|2[0-4][0-9]|[
01]?[0-9][0-9]?)$/;
  return regex.test(IP);
}

function getLocalTime(Longitude) {
  // get time
  const date = new Date();

  // calculate offset
  const offset = Longitude / 15;

  // calculate the local time in UTC
  let utcTime = date.getTime() + date.getTimezoneOffset() * 60000;
  utcTime += 3600000 * offset;

  // adjust for daylight saving time
  const dst =
    date.getTimezoneOffset() <
    new Date(Date.UTC(date.getFullYear(), 0, 1)).getTimezoneOffset();
  if (dst) {
    utcTime += 3600000;
  }
```

```
  // convert UTC time to local time zone
  const localTime = new Date(utcTime);

  // format the time as a string
  const hours = localTime.getHours().toString().padStart(2, "0");
  const minutes = localTime.getMinutes().toString().padStart(2, "0");
  const seconds = localTime.getSeconds().toString().padStart(2, "0");
  const timeString = `${hours}:${minutes}:${seconds}`;

  return timeString;
}

function getTimeUTC(positionGd) {
  // get time from longitude
  const gmst = gstime(new Date());
  const now = new Date();
  const lmst = gmst + degreesLong(positionGd.longitude) / 15;
  const time = new Date(now.getTime() + (lmst - gmst) * 3600 * 1000);

  // parse time into hours,min,sec
  const hours = time.getUTCHours().toString().padStart(2, "0");
  const minutes = time.getUTCMinutes().toString().padStart(2, "0");
  const seconds = time.getUTCSeconds().toString().padStart(2, "0");

  // format time as a string
  const timeString = `${hours}:${minutes}:${seconds}`;
  console.log(timeString);
  return timeString;
}
```

9. DropdownMenu.js

```
import React, { useState } from "react";
import Alert from "@material-ui/lab/Alert";
import {
  Button,
  Menu,
  MenuItem,
  FormControlLabel,
  Switch,
  TextField,
  Snackbar
} from "@material-ui/core";
import { fetchJson } from "./Helper";
import { Tune as MenuIcon } from "@material-ui/icons";
import { Stop as StopIcon } from "@material-ui/icons";

function DropdownMenu({ Satellite, Rotator }) {
  const [anchorEl, setAnchorEl] = useState(null);
  const [isAutoTrackOn, setIsAutoTrackOn] = useState(false);
  const [azimuth, setAzimuth] = useState("");
  const [elevation, setElevation] = useState("");
  const [isTracking, setIsTracking] = useState(false);
  const [azimuthError, setAzimuthError] = useState("");
  const [elevationError, setElevationError] = useState("");
  const [Snack, setSnack] = useState(null);

  const handleToggleMenu = (event) => {
    if (Satellite !== null && Rotator !== null) {
      // only open control menu when satellite and rotator are chosen
      setAnchorEl(event.currentTarget);
    } else {
      // display error message
      setAnchorEl(event.currentTarget);
      setSnack({
        message:
          "Please select a satellite and connect a rotator to begin
tracking",
        variant: "warning"
      });
    }
  };
```

```
  const handleCloseMenu = (event) => {
    if (Boolean(azimuthError) || Boolean(elevationError)) {
      // clear input fields
      setAzimuth("");
      setElevation("");
      setAzimuthError("");
      setElevationError("");
    }
    //close control menu
    setAnchorEl(null);
  };

  const handleCommand = async () => {
    if (isAutoTrackOn || (azimuth !== "" && elevation !== ""))
      if (isTracking) {
        // send stop tracking command to mcu
        const response = await fetchJson(Rotator.IP, "Stop");

        if (typeof response === "object") {
          console.log(response.message);
          // notify user with success response
          setSnack({ message: response.message, variant: "success" });
          setIsTracking(false);
          Rotator.isTracking = false;
        } else {
          // // notify user with error response
          setSnack({ message: response.message, variant: "error" });
        }
      } else {
        var response = "";
        setAnchorEl(null);
        if (isAutoTrackOn) {
          // serialize satellite object
          const auto_string = JSON.stringify(Satellite.Omm);
          // send auto-tracking command
          response = await fetchJson(Rotator.IP, "StartAuto", auto_string);
        } else {
          // serialize azimuth/elevation
          const manual_string = azimuth + " " + elevation;
          // send manual tracking command
          response = await fetchJson(Rotator.IP, "StartManual",
manual_string);
```

```
      }
      if (typeof response === "object") {
        console.log(response.message);
        // notify user with success response
        setSnack({ message: response.message, variant: "success" });
        setIsTracking(true);
        Rotator.isTracking = true;
      } else {
        //// notify user with error response
        setSnack({ message: response.message, variant: "error" });
      }
    }
};

const handleCloseSnack = () => {
  // clear snack message popup
  setSnack(null);
  setAnchorEl(null);
};

const handleAzimuth = (value) => {
  //get azimuth from text field
  const azimuthValue = parseFloat(value);
  if (azimuthValue < 0 || azimuthValue > 450) {
    // display error
    setAzimuthError("Azimuth should be between 0 and 450");
    setAzimuth("");
  } else {
    // save text from input field
    setAzimuthError("");
    setAzimuth(value);
  }
};

const handleElevation = (value) => {
  //get elevation from text field
  const elevationValue = parseFloat(value);
  if (elevationValue < 0 || elevationValue > 180) {
    // display error
    setElevationError("Elevation should be between 0 and 180");
  } else {
    // save text from input field
    setElevationError("");
```

```
    setElevation(value);
  }
};
const handleToggleAutoTrack = () => {
  // switch tracking control state
  setIsAutoTrackOn((prevState) => !prevState);
  setAzimuth("");
  setElevation("");
};

return (
  <>
    {!isTracking ? (
      <Button onClick={handleToggleMenu}>
        <MenuIcon />
      </Button>
    ) : (
      <Button onClick={handleCommand}>
        <StopIcon />
      </Button>
    )}

    <Snackbar
      anchorOrigin={{ vertical: "top", horizontal: "left" }}
      open={Boolean(Snack)}
      onClick={handleCloseSnack}
      onClose={handleCloseSnack}
      autoHideDuration={5000}
      style={{ top: "80px" }}
    >
      <div>
        {Boolean(Snack) && (
          <Alert severity={Boolean(Snack) ? Snack.variant : "info"}>
            {Boolean(Snack) ? Snack.message : ""}
          </Alert>
        )}
      </div>
    </Snackbar>

    {!isTracking && !Snack && (
      <Menu
        anchorEl={anchorEl}
        open={Boolean(anchorEl)}
```

```
  onClose={handleCloseMenu}
  style={{ top: "70px" }}
>
  <MenuItem>
    <FormControlLabel
      control={
        <Switch
          checked={isAutoTrackOn}
          onChange={handleToggleAutoTrack}
          name="autoTrack"
          color="primary"
        />
      }
      label="Auto Track"
    />
  </MenuItem>

  <MenuItem>
    <TextField
      label="Azimuth"
      type="number"
      value={azimuth}
      onChange={(e) => handleAzimuth(e.target.value)}
      disabled={isAutoTrackOn}
      onBlur={(e) => handleAzimuth(e.target.value)}
      error={Boolean(azimuthError)}
      helperText={azimuthError}
    />
  </MenuItem>

  <MenuItem>
    <TextField
      label="Elevation"
      type="number"
      value={elevation}
      onChange={(e) => handleElevation(e.target.value)}
      disabled={isAutoTrackOn}
      onBlur={(e) => handleElevation(e.target.value)}
      error={Boolean(elevationError)}
      helperText={elevationError}
    />
  </MenuItem>
```

```
          <MenuItem style={{ justifyContent: "center" }}>
            <Button onClick={handleCommand}>{"start tracking"}</Button>
          </MenuItem>
        </Menu>
      )}
    </>
  );
}

export default DropdownMenu;
```

APPENDIX A: Server Test Code

```
#include <ESP8266WiFi.h>
#include <time.h>
#include <ArduinoJson.h>

#ifndef STASSID
#define STASSID "ASUS-10 2.4G"
#define STAPSK "97955757"
//192.168.50.100

//#define STASSID "DavidPhone"
//#define STAPSK "DavidSantos"
//172.20.10.3
#endif

const char *ssid = STASSID;
const char *pass = STAPSK;

// Set Server Port
BearSSL::WiFiServerSecure server(443);

// Set Server Private Key:
const char server_private_key[] PROGMEM = R"EOF(
-----BEGIN PRIVATE KEY-----
MIIEvgIBADANBgkqhkiG9w0BAQEFAASCBKgwggSkAgEAAoIBAQDYNTVYXPlGmlL2
o7YxFUvpCIsCSbDhNU8YXUz8V2s+s29qlgkMJFl2MSKPFunZ9oefugJ2u+jvlTmh
mwDO2wjb8jN5RIXL8pJ7xOw3lWyNl5p1VPNQc2bDEC4D8mOVg/OvELubrSdsBO9N
HMH93Vxl/DeOfw3jVKnQgrlB8kTfrdnj/SYe7mjOlEkPnmPldnvZhdN9B036aTfz
```

```
l1Tqny+9UewXQ9IpdMYk/H5wCRzumyV9lBLzJCWah1wdWrXcyA/y38iT6nfHnkno
CxhrowknIUUwyT29sotvSxKmEwgCCVoakSwUXmxlg6kpigBKC5KZ94jvE8DnEpQ5
MCGlOvsvAgMBAAECggEABjf5kdISxWoY6skoq16+cTRysx0lfkTHQp8kYLmu9v24
PauVFQlSnWiqYs046rEBk6GJAptDBukW4EQSEOER7ymXO2yyIHyYpAC0/qBIoAX1
RlysJus60G99s1Vcced6sdVcYAzp7lxx+ZjTJ5cfWjuX6XxUDO7Zd4rbAz+RpZFQ
AxmDLMoGT8KTdRLZdPPIW78oXBQrFGXK3ul+WaWrzce/4PFoWLPK9mTUwfNLBasM
quyNhNJ67h2zbATCEgSvhOOGMwSiiq1rt8WWJMCz2nZ/cAayqeAL1KC5uqBIRkbD
amikUkxR1RDN1j0i+uvDa0XgVNz7nZWxPFit5FLLyQKBgQDunFHXiG9Xo2tb6JpJ
IVDpqcF55qEio6nRt6pZJeopQTV3pAUbRUDKcZo3PHsqfB+i/vKeqApAIt/LbFvO
cx/k0BHNevnB9NLwnMNiXo72jw6acGVe/od9dzQ3NqOqbxuanq8nLDyMxvKyeEaP
h+9RPnWQI27Mc0HykoKYJUcf1QKBgQDn9u1on0v9O+Ewj67hWI2xwhlaGT9Ob8D+
G9NHqqEACl0lc3pLll2WFxiDZPtpsVow4ITfzZ0fGvJ1bo+HSlFzCWcGkUoh4o2j
SKiN8PyhlBY2J/52gN8Knj53BlE6ux9+84XXazycNK9DJafGCdo0uOd5k+uCtALE
9MeUf/608wKBgCKaSMQY5QywhkEt2jIAwtsf4w7qgzOiyF3BZNLIzMxuyHIxNwxd
Xqf9EYAxJStkFLJuhoC7ncDvTcUSUb3+tAFaqzpBOzcFahG6qhr524Gl/VgoNIsy
HjPztX7MTa/JrRcfLAdiQuWndEqtZo4qCGqfxCzC9q5viIDE/mtR/J6BAoGBALUf
vU71G5u3duW5O79fqvxC6BLPJfx0DIkeZwJNvHtQnFzS54o/9gFtNUnbvuhPgHIr
zfUHEwmnu+xwEQh9wk2F6Pla1zzQLvJxmJOEuW7dN96/qdUUWZRr7bB4sVKfrD8v
nbGQXXpHNRbEP1XGeaizPKNVGg8rygHKdFCDBPp7AoGBALbAfLpl2i9VtwAOeofK
drTKvUi5L/iA52IdupBx1+nIA2hRVQRnwoo4uSaarEAXg3FuDrLQmcA9FpMBW1eM
d0YH37RaUMZDxmiCkEpFkyqtb4G2/m8UZ1W6B+HwX2AybnWpe3KIwva5Itb9aVFx
EHHyeyYu/+0oPosWVYtNeN/o
-----END PRIVATE KEY-----
)EOF";

// Set Server Certificate
const char server_cert[] PROGMEM = R"EOF(
-----BEGIN CERTIFICATE-----
MIIDnzCCAocCFHWLd/xNg9sbWNql8Yk8TBljv+WYMA0GCSqGSIb3DQEBCwUAMIGL
MQswCQYDVQQGEwJVUzEOMAwGA1UECAwFVGV4YXMxGDAWBgNVBAcMD0NvbGxlZ2Ug
U3RhdGlvbjEMMAoGA1UECgwDQSZNMQ0wCwYDVQQLDARFQ0VOMRQwEgYDVQQDDAtE
YXZpZFNhbnRvczEfMB0GCSqGSIb3DQEJARYQZG1zNDk5NUB0YW11LmVkdTAeFw0y
MzAzMjgxNjUwMTlaFw0yNDAzMjcxNjUwMTlaMIGLMQswCQYDVQQGEwJVUzEOMAwG
A1UECAwFVGV4YXMxGDAWBgNVBAcMD0NvbGxlZ2UgU3RhdGlvbjEMMAoGA1UECgwD
QSZNMQ0wCwYDVQQLDARFQ0VOMRQwEgYDVQQDDAtEYXZpZFNhbnRvczEfMB0GCSqG
SIb3DQEJARYQZG1zNDk5NUB0YW11LmVkdTCCASIwDQYJKoZIhvcNAQEBBQADggEP
ADCCAQoCggEBANg1NVhc+UaaUvajtjEVS+kIiwJJsOE1TxhdTPxXaz6zb2qWCQwk
WXYxIo8W6dn2h5+6Ana76O+VOaGbAM7bCNvyM3lEhcvyknvE7DeVbI2XmnVU81Bz
ZsMQLgPyY5WD868Qu5utJ2wE700cwf3dXGX8N45/DeNUqdCCuUHyRN+t2eP9Jh7u
aM6USQ+eY+V2e9mF030HTfppN/OXVOqfL71R7BdD0il0xiT8fnAJHO6bJX2UEvMk
JZqHXB1atdzID/LfyJPqd8eeSegLGGujCSchRTDJPb2yi29LEqYTCAIJWhqRLBRe
bGWDqSmKAEoLkpn3iO8TwOcSlDkwIaU6+y8CAwEAATANBgkqhkiG9w0BAQsFAAOC
AQEAG8scg+QVUYnxJJ3e9WPnrEwzTUGPNGbLdhULnNNXDxdGY/20Ne64vJmyMh9h
```

```
cGIsJeUUtgWuNSsTC9sWNISsh96NH297AuLpYs4NcbPJaZMB/vApFuI01V6g74WQ
pXd+9Atrgs5W9wTDCUW63QNkF2YDO5jDYG1AYvAFyKf51MZvD6+cMeXhQx7B5V8j
x8OGU5HqrYu/yzgX0/O9cNcS5vxlvUd4yZ6w6OXtvWXBDjTh4kPyTqGWVLz3U1Tz
WDZw81GLkuSc9MIEkJX5yKw0huY8b2OvaDOtErDu5jNMHTPkfSRifvhybCSKdIux
53PU70olAhSLqZmiIDam+FJvSA==
-----END CERTIFICATE-----
)EOF";


#define CACHE_SIZE 5  // Cache Sessions
#define USE_CACHE     // Enable Cache
//#define DYNAMIC_CACHE //Enable Dynamic

#if defined(USE_CACHE) && defined(DYNAMIC_CACHE)
// Dynamically allocated cache.
BearSSL::ServerSessions serverCache(CACHE_SIZE);
#elif defined(USE_CACHE)
// Statically allocated cache.
ServerSession store[CACHE_SIZE];
BearSSL::ServerSessions serverCache(store, CACHE_SIZE);
#endif

void setup() {
  Serial.begin(115200);
  Serial.println();
  Serial.println();

  // Connect Wifi Network
  Serial.print("Connecting to ");
  Serial.println(ssid);
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, pass);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");

  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
```

```
  // Attach Certificate/Key to Server
  BearSSL::X509List *serverCertList = new BearSSL::X509List(server_cert);
  BearSSL::PrivateKey *serverPrivKey = new
BearSSL::PrivateKey(server_private_key);

#ifndef USE_EC
  server.setRSACert(serverCertList, serverPrivKey);
#else
  server.setECCert(serverCertList, BR_KEYTYPE_KEYX | BR_KEYTYPE_SIGN,
serverPrivKey);
#endif

  // Setup Cache
#if defined(USE_CACHE)
  server.setCache(&serverCache);
#endif

  // Start Server
  server.begin();
}

void loop() {

  // Check for Client
  BearSSL::WiFiClientSecure client = server.accept();
  if (!client) { return; }

  // Read Request
  const String request = client.readStringUntil('\r');
  client.flush();

  // Respond to Request
  respond(client, request);

  // End Connection
  client.flush();
  client.stop();
  Serial.printf("Client Disconnected\n");
}


void respond(BearSSL::WiFiClientSecure client, String request) {
  //[TO-DO]: Make RequestType work for all types
```

```
// Initialize Response Variables
//Serial.println("Client Request:: " + request);
String RequestType = request.substring(request.indexOf('/') + 1);
RequestType = RequestType.substring(0,RequestType.indexOf('/'));
Serial.println("Client Request:: " + RequestType);
String Header = "HTTP/1.1 200 OK";
String Methods = "GET";

// Construct a JSON array with the response data
DynamicJsonDocument doc(200);
doc["RequestType"] = RequestType;


//[TO-DO] - respond to "favicon.ico" request
//[TO-DO] - get real coordinates/position from mcu
//[TO-DO] - send payload to mcu if needed
if (RequestType == "Connect") {
  doc["NAME"] = "David's Rotator";
  doc["IP"] = WiFi.localIP();
  doc["CALL_SIGN"] = "W5QZ";
  doc["MODEL"] = "Yaesu G-5500";
  doc["LONGITUDE"] = "30.6280";
  doc["LATITUDE"] = "-96.3344";
  doc["HEIGHT"] = "1";

} else if (RequestType == "Stop"){
  //tell mcu to stop tracking
  //continue once mcu has stopped
  doc["message"] = "Rotator has stopped tracking";

}else if(RequestType == "StartAuto"){
  doc["message"] = "Rotator has started auto-tracking";

}else if(RequestType == "StartManual"){
  doc["message"] = "Rotator has started manual-tracking";

}else {
  doc["message"] = "HTTP/1.1 404 Not Found";
  Header += "HTTP/1.1 404 Not Found";
  Methods = "*";
}

// Serialize the JSON array to a string
```

```
  String jsonString;
  serializeJson(doc, jsonString);

  // Headers
  client.println(Header);
  client.println("Content-Type: application/json");
  client.println("Access-Control-Allow-Origin: *");
  client.println("Access-Control-Allow-Methods: " + Methods);
  client.println("");

  //Send Data
  client.print(jsonString);
}
```