Development

```
<table style="width:100%">
  <tr>
    <th>time</th>
    <th id="a1">sunday</th>
    <th id="a2">monday</th>
    <th id="a3">tuesday</th>
    <th id="a4">wednesday</th>
    <th id="a5">thursday</th>
    <th id="a6">friday</th>
    <th id="a7">saturday</th>
  </tr>
  <tr id="8am">
    <td>8:00 am</td>
    <td contenteditable="true"></td>
    <td contenteditable="true"></td>
    <td contenteditable="true"></td>
    <td contenteditable="true"></td>
    <td contenteditable="true"></td>
    <td contenteditable="true"></td>
    <td contenteditable="true"></td>
  </tr>
```

This table has more <tr> with <td> elements from 8:30 am to 8:00 pm (omitted to avoid redundancy). Displays weekly calendar cells containing tasks, description, and subject. Editable cells enable client to edit tasks and reduce user-error.

```
<div class="webRow">
  <div class="column side">
    <h2>focus while working</h2><br>
      <iframe style="border-radius:12px" src="https://open.spotify.com/embed/playlist/3hPeTyReRDrbVqtUHvwfSp?utm_source=generator"
              width="100%" height="380" frameBorder="0" allowfullscreen="" allow="autoplay; clipboard-write; encrypted-media; fullscreen; picture-in-picture">
      </iframe>
  </div>
```

```
<div class="column side">
  <h2>get motivated while taking a break</h2><br>
    <iframe style="border-radius:12px" src="https://open.spotify.com/embed/playlist/37i9dQZF1DXdxcBWuJkbcy?utm_source=generator"
            width="100%" height="380" frameBorder="0" allowfullscreen="" allow="autoplay; clipboard-write; encrypted-media; fullscreen; picture-in-picture">
    </iframe>
</div>
```

Inline frames, embedding two Spotify playlists—one for motivation and another for focus—allows client to access focus and motivational playlists on website.

```
<div class="webRow">
  <div class="column side">
    <h2>7/11 breathing exercise</h2><br>
      <p>breathe in for 7 seconds</p><br>
      <p>breathe out for 11 seconds</p><br>
      <p>repeat for 12 to 15 repetitions</p>
  </div>
```

```
<div class="column side">
  <h2>hot chocolate breathing exercise</h2>
  <br><p>hold up your hands as if you're holding a mug of hot chocolate right under your face</p>
  <br><p>imagine that you are smelling the hot chocolate by taking a deep breath</p>
  <br><p>blow out the hot chocolate by exhaling</p>
  <br><p>repeat</p>
</div>
```

```
<br><p>stuck? let's <a href="https://www.cognifit.com/brain-games" target="_blank">play!</a></p>
```

Breathing exercises and small games are accessible on/from webpage. Play link open in new tab in case client wants to leave it open and use it conveniently with task manager tab.

```
var task = [];
        function Task(day, time, subject, taskInfo) {
        this.dayy = day;
        this.timee = time;
        this.subjectt= subject;
        this.taskInfoo = taskInfo; }
```

Declared variable Task represents any task. Task() handles the parameters distinguishing each task and constructs their objects. Client can save tasks with their respective values.

```
function setProperties() {
    var e = document.getElementById('daySelector');
    var daySelectorValue = e.options[e.selectedIndex].text;
    var f = document.getElementById('timeSelector');
    var timeSelectorValue = f.options[f.selectedIndex].text;

    var a = daySelectorValue;
    var b = timeSelectorValue;
    var c = (document.getElementById("Subject").value);
    var d = (document.getElementById("taskInfo").value);

    var newTask = new Task(a, b, c, d);
            task.push(newTask);

    var x = document.getElementById("daySelector").value;
    var z = document.getElementById("Subject").value;
    var w = document.getElementById("taskInfo").value;

    document.getElementById(document.getElementById("timeSelector").value).children[parseInt(x)].innerHTML = w + ' regarding/for ' + z;
```

Function assigns values to tasks. Client can input day, time, subject, and description of each task and display it in calendar.

```
    var i;
    var txt = '';
    for (i = 0; i < task.length; i++) {
        var total = task[i];
        txt += total.totalList()+'<br><br>';
        document.getElementById('outputTasks').innerHTML = txt;
  }
        }
  //This function handles the task objects returned and how they're organized when the list of total tasks is displayed.
  Task.prototype.totalList = function() {
        return 'you have to ' + this.taskInfoo + ' regarding/for ' + this.subjectt + ' on/by ' + this.dayy + ' at ' + this.timee;
        }
```

Establishes variables representing total tasks. Loops through all of the task instances in ascending order to output tasks chronologically as text. Separated by breaks to appear as the intended to-do list.

```
var a = new Date();
var b = a.getDay();
var c = a.getHours();
var d = a.getMinutes();

var trs = document.getElementsByTagName('tr');
```

Variable a stores the current day and time elements. Declaring variables b, c, and d to store specific day, hours, and minutes values. Allows conditional statements for client to visualize current day and time, as follows.

```
if (b==0) {
   var i;
    for (i=0;i<trs.length; i++){
 trs[i].children[1].style.backgroundColor = "#9EE4A1"} }

else if (b==1) {
   var i;
    for (i=0;i<trs.length; i++){
 trs[i].children[2].style.backgroundColor = "#9EE4A1"} }

else if (b==2) {
   var i;
    for (i=0;i<trs.length; i++){
 trs[i].children[3].style.backgroundColor = "#9EE4A1"} }

else if (b==3) {
   var i;
    for (i=0;i<trs.length; i++){
 trs[i].children[4].style.backgroundColor = "#9EE4A1"} }

else if (b==4) {
   var i;
    for (i=0;i<trs.length; i++){
 trs[i].children[5].style.backgroundColor = "#9EE4A1"} }

else if (b==5) {
   var i;
    for (i=0;i<trs.length; i++){
 trs[i].children[6].style.backgroundColor = "#9EE4A1"} }

else {
   var i;
    for (i=0;i<trs.length; i++){
 trs[i].children[7].style.backgroundColor = "#9EE4A1"} }
```

Loops from Sunday to Saturday to highlight column of current day (second child). Client can visualize when tasks are due, relative to current day.

```
if (c == 8) {
    if (d<30) {document.getElementById("8am").children[b+1].style.backgroundColor = "#31C437"}
    else {document.getElementById("8:30am").children[b+1].style.backgroundColor = '#31C437'}
}
else if (c==9)
    {if (d<30) {document.getElementById("9am").children[b+1].style.backgroundColor='#31C437'}
    else {document.getElementById('9:30am').children[b+1].style.backgroundColor='#31C437'}}

else if (c==10)
    {if (d<30) {document.getElementById("10am").children[b+1].style.backgroundColor='#31C437'}
    else {document.getElementById('10:30am').children[b+1].style.backgroundColor='#31C437'}}

else if (c==11)
    {if (d<30) {document.getElementById("11am").children[b+1].style.backgroundColor='#31C437'}
    else {document.getElementById('11:30am').children[b+1].style.backgroundColor='#31C437'}}

else if (c==12)
    {if (d<30) {document.getElementById("12pm").children[b+1].style.backgroundColor='#31C437'}
    else {document.getElementById('12:30pm').children[b+1].style.backgroundColor='#31C437'}}

else if (c==13)
    {if (d<30) {document.getElementById("1pm").children[b+1].style.backgroundColor='#31C437'}
    else {document.getElementById('1:30pm').children[b+1].style.backgroundColor='#31C437'}}

else if (c==14)
    {if (d<30) {document.getElementById("2pm").children[b+1].style.backgroundColor='#31C437'}
    else {document.getElementById('2:30pm').children[b+1].style.backgroundColor='#31C437'}}

else if (c==15)
    {if (d<30) {document.getElementById("3pm").children[b+1].style.backgroundColor='#31C437'}
    else {document.getElementById('3:30pm').children[b+1].style.backgroundColor='#31C437'}}

else if (c==16)
    {if (d<30) {document.getElementById("4pm").children[b+1].style.backgroundColor='#31C437'}
    else {document.getElementById('4:30pm').children[b+1].style.backgroundColor='#31C437'}}

else if (c==17)
    {if (d<30) {document.getElementById("5pm").children[b+1].style.backgroundColor='#31C437'}
    else {document.getElementById('5:30pm').children[b+1].style.backgroundColor='#31C437'}}

else if (c==18)
    {if (d<30) {document.getElementById("6pm").children[b+1].style.backgroundColor='#31C437'}
    else {document.getElementById('6:30pm').children[b+1].style.backgroundColor='#31C437'}}

else if (c==19)
    {if (d<30) {document.getElementById("7pm").children[b+1].style.backgroundColor='#31C437'}
    else {document.getElementById('7:30pm').children[b+1].style.backgroundColor='#31C437'}}

else if (c==20)
    {if (d<30) {document.getElementById("8pm").children[b+1].style.backgroundColor='#31C437';}}

else {document.getElementById('outsideHours').innerHTML = "it is currently outside of your working hours!";}
```

Highlights cell with current time using getMinutes() and getHours(). Loops through whole hours from 8 AM until 8 PM, inclusive and checks if minutes are less than 30. If so, rounds down to whole hour. If minutes are 30 or more, highlights cell with current half hour increment. Client can visualize tasks relative to highlighted time. If current time isn't between 8 AM and 8 PM, else statement displays text reminding client that it's outside their working hours.

```
function deleteArray() {
    window.location.reload();
}
```

Function called when reset calendar button is clicked, allowing the client to reset all tasks with one click. To make code more compact, the function reloads the webpage. Has room for extension by deleting the actual array instead of reloading the page, to prevent timer from resetting.

```
function deleteTask() {
    var x = document.getElementById("daySelectorD").value;
    var e = document.getElementById('daySelectorD');
    var daySelectorDvalue = e.options[e.selectedIndex].text;
    var f = document.getElementById('timeSelectorD');
    var timeSelectorDvalue = f.options[f.selectedIndex].text;
    var a = daySelectorDvalue;
    var b = timeSelectorDvalue;
    document.getElementById(document.getElementById("timeSelectorD").value).children[parseInt(x)].innerHTML = "";

for (i=0; i<task.length; i++) {
    var i;
    var total = task[i];
        if (total.dayy === a){
            if (total.timee === b){
                task.splice (i, 1);
                document.getElementById('outputTasks').innerHTML = "";

        }
}

}
}
```

Handles the event when the deleteTask button is clicked. Variables order ensures that deleted task are deleted both from the table and to-do list. Used similar loop to the one in setProperties() so I can use splice method and ensure deletion of task from table and to-do list too so client can delete individual tasks.

```
const progressBar = document.querySelector(".edgeOfTimer"),
    minuteArea = document.querySelector("#minutes"),
    secondArea = document.querySelector("#seconds"),
    edit = document.querySelector("#edit"),
```

```
    startStop = document.querySelector("#startAndStop");
let minutes = document.querySelector("#minutes").innerHTML,
    seconds = document.querySelector("#seconds").innerHTML,
    progress = null,
    startingPoint = 0,
    endingPoint = parseInt(minutes) * 60 + parseInt(seconds),
    progressSpeed = 1000,
    degreesDivided = 360 / endingPoint,
```

Get start/stop button using const declaration and declare minutes and seconds as let variables, allowing their display to be altered so client is aware of the time.

```
    toggleEdits = false,
    remainingSeconds = 0,
    remainingMinutes = 0;
```

Declare toggleEdits as let variable to manage edit button. It's set to false to establish default timer 25 minutes—ideal for ADHD patients. Allows client to use that time duration the most.

```
function progressTrack() {
    startingPoint++;

    remainingSeconds = Math.floor((endingPoint - startingPoint) % 60);
    remainingMinutes = Math.floor((endingPoint - startingPoint) / 60);
```

Function tracks timer progress and progress bar. It calculates the remaining time and how many minutes and seconds it is. Determines how many degrees of the progress bar are allocated per second. Enables visualization of client's progress.

```
  secondArea.innerHTML = remainingSeconds.toString().length == 2 ? remainingSeconds : `0${remainingSeconds}`;
  minuteArea.innerHTML = remainingMinutes.toString().length == 2 ? remainingMinutes : `0${remainingMinutes}`;
```

Length allows updating timer's numerical display so that client is aware of time.

```
  progressBar.style.background = `conic-gradient(
      #D62A2A ${startingPoint * degreesDivided}deg,
      #17171a ${startingPoint * degreesDivided}deg
      )`;
  if (startingPoint == endingPoint) {
      progressBar.style.background = `conic-gradient(
              #2AD62C 360deg,
              #2AD62C 360deg
        )`;
      clearInterval(progress);
      startStop.innerHTML = "start";
      progress = null;
      startingPoint = 0;
  }
```

To handle progress bar, I used conic-gradient() function because radial-gradient() function would be unnecessary as only the edge of the gradient is visible. I styled its color depending on timer's progress status (in-progress/paused). DOM is updated accordingly, visually indicating client's progress.

```
function startStopProgress() {
    if (!progress) {
        progress = setInterval(progressTrack, progressSpeed);
    } else {
        clearInterval(progress);
        progress = null;
        startingPoint = 0;
        progressBar.style.background = `conic-gradient(
                #000000 360deg,
                #000000 360deg
          )`;
    }
}
```

Function called when start/stop button is clicked. If there's no progress, assumed that start button is displayed and setInterval() method tracks progress. Otherwise, timer is running and that the STOP text is displayed on the button, which uses the clearInterval method. That is when the progress bar resets to original color. Client can visually see that timer stopped.

```
function reset() {
    if (progress) {
        clearInterval(progress);
    }
    minutes = document.querySelector("#minutes").innerHTML;
    seconds = document.querySelector("#seconds").innerHTML;
    toggleEdits = false;
    minuteArea.contentEditable = false;
    minuteArea.style.borderBottom = `none`;
    secondArea.contentEditable = false;
    secondArea.style.borderBottom = `none`;
    progress = null;
    startingPoint = 0;
    endingPoint = parseInt(minutes) * 60 + parseInt(seconds);
    degreesDivided = 360 / endingPoint;
    progressBar.style.background = `conic-gradient(
                #000000 360deg,
                #000000 360deg
          )`;
}
```

Function called in the event of an onblur of minute and second display and when edit button is unclicked. Thus, timer values reset visually to start newly assigned timer interval.Used the querySelector() method because I needed the first elements of the minutes and seconds. Allows the client to resume timer and visually see timer is no longer editable.

```
startStop.onclick = function () {
    if (startStop.innerHTML === "start") {
        if (!(parseInt(minutes) === 0 && parseInt(seconds) === 0)) {
            startStop.innerHTML = "stop";
            startStopProgress();
        } else {
            alert("please enter numerical values that are greater than zero!");
        }
    } else {
        startStop.innerHTML = "start";
        startStopProgress();
    }
};
```

Handles click event of start/stop button. To make code more compact, I used the same startStopProgress function for either start/stop status. If button displays “start,” function ensures that input is nonzero. Then, start/stop button's text changes from “start” to “stop.” Additionally, numerical display progresses as startStopProgress is called, changing the timer and progress bar display. If both minutes and seconds are zero, user is alerted to prevent user error as specified in success criteria. If, when clicked, button displays “stop” (timer in-progress), then the same startStopProgress function resets the progress and the start/stop button. Client can visualize timer’s status (paused or in-progress).

```
edit.onclick = function () {
    if (!toggleEdits) {
        toggleEdits = true;
        minuteArea.contentEditable = true;
        minuteArea.style.borderBottom = `5px dashed #ffffff`;
        secondArea.contentEditable = true;
        secondArea.style.borderBottom = `5px dashed #ffffff`;
    } else {
        reset();
    }
};
```

Handles click event of edit button. Timer’s editability is determined by boolean values, allowing client to manage time in customizable manner. If being edited, minutes and seconds display is underlined. Using such visual indication like client requested, this styling illustrates ability of client to edit timer directly from numerical display.

```
minuteArea.onblur = function () {
    reset();
};

secondArea.onblur = function () {
    reset();
};
```

For the minutes and seconds elements, I used onblur event rather than onfocusout event because onblur event doesn’t bubble, which satisfies client's non-distracting, minimal design. Reset() handles newly edited values to update time elements, allowing user to customize time intervals.

```
function saveTextAsFile(){
    var savedNotes = document.getElementById("inputSavedNotes").value;
    var savedNotesAsBlob = new Blob([savedNotes], {type:"text/plain"});
    var savedNotesAsURL = window.URL.createObjectURL(savedNotesAsBlob);
    var savedName = document.getElementById("inputSavedName").value;

    var downloadLink = document.createElement("a");
    downloadLink.download = savedName;
    downloadLink.innerHTML = "Download File";
    downloadLink.href = savedNotesAsURL;
    downloadLink.onclick = destroyClickedElement;
    downloadLink.style.display = "none";
    document.body.appendChild(downloadLink);

    downloadLink.click();
}

function destroyClickedElement(event){
    document.body.removeChild(event.target);
}

function loadFileAsText(){
    var fileToLoad = document.getElementById("fileToLoad").files[0];

    var fileReader = new FileReader();
    fileReader.onload = function(fileLoadedEvent){
        var textFromFileLoaded = fileLoadedEvent.target.result;
        document.getElementById("inputSavedNotes").value = textFromFileLoaded;
    };
    fileReader.readAsText(fileToLoad, "UTF-8");
}
```

Functions allow user to save and edit text files to/from computer. saveTextAsFile() uses the download attribute of the element a to download input of textarea element. Allows the client a lot of available character length, as desired, in txt format for universality. The target property of the event interface refers to the object onto which the event was dispatched. loadFileAsText() is finally called at the event of file loading. It gets the text content of the loaded file and inputs it as if the user wrote it in the text area, which allows client to edit notes as desired. I used UTF-8 because it covers a wide variety of Unicode characters and, thus, allows client to write freely.