

Source: app.js

```
1.  /* global dict:readonly */
2.  /*
3.   * Audio
4.   * newPM is a whistle
5.   * GenericNotify is a chime (Light/dark theme)
6.   * tournament is clapping (congrats)
7.   * LowTime is a Low bell
8.   */
9.  /*
10.   * Phases: idle, work, short break, Long break (Lowercase)
11.   *
12.   * i = idle, W = work, b = short break, B = Long break
13.   *
14.   * i-W-b-W-b-W-b-W-B (1 cycle)
15.   * i-0-1-0-1-0-1-0-2
16.   */
17.  let lang;
18.  let phase = 'idle';      // idle, work, short break, Long break, stopped
19.  let workLength = 900;    // work time (seconds)   15 mins (900)
20.  let shortBreakLength = 300; // short break time    5 mins  (300)
21.  let longBreakLength = 1500; // Long break time     25 mins (1500)
22.  let timer;              // Represents the interval change of 1s
23.  let secondsRemaining = 0; // Displays on timer
24.  let MMSS;              // {string} MM:SS format
25.  let tasksDone = 0;      // the number of tasks finished
26.  let pomosDone = 0;      // Number of pomo 'work' phases completed.
27.  let taskCount = 0;      // Used to keep track of all active tasks
28.  var uniqueID = 1;      // Used to assign uniqueID's when deleting specific tasks
29.  let savedTasks = [];

30.  let volume = 25;
31.  let theme;              // Potato, Dark, Light, undefined (Capitalized)
32.  let mute = false;      // whether the volume is muted
33.  let animation = true;   // show dancing potatoes or still potatoes
34.
35.  const MAX_POTATOES_COMPLETED = 15;
36.  /**
37.   * Adds event listeners to the congratulations screen so a user can click anywhere outside the screen
38.   * To close it.
39.   * Loads the user's locally saved theme and language.
40.   * If it is the user's first time, the instructions menu will automatically show up.
41.   * Otherwise, the user's tasks and saved settings will be loaded from local storage.
42.   */
43.  window.onload = function () {
44.    let congrats = document.getElementById('congratsScreen');
45.
46.    //make addTask execute on enter press
47.    window.addEventListener('keypress', function (e) {
48.      //if the pressed key is the enter key
49.      if (e.key == 'Enter') {
50.        addTask();
51.      }
52.    });
53.
54.    // When the user clicks anywhere outside of the modal, close the congrats
55.    // screen
56.    window.onclick = function (event) {
57.      if (event.target == congrats) {
58.        hide('congratsScreen');
59.      }
60.    }
61.
62.    /**
```

```
63.      * Function that stores themes, languages, and adds tasks back into the
64.      * website
65.      */
66.      // Load's users theme and Laguage and settings
67.      loadTheme();
68.      loadLang();
69.      if (window.localStorage.getItem('returning') == 'true') {
70.          loadTasks();
71.          if(window.localStorage.getItem('savedSettings') == 'true') {
72.              loadSettings();
73.          }
74.      } else {
75.          show('instructionsMenu');
76.          next();
77.          window.localStorage.setItem('returning', true);
78.      }
79.  }
80.
81.  /**
82.   * Play the audio from break to work
83.   * 'breakToWorkAudio', 'workToBreakAudio', 'victoryAudio'
84.   *
85.   * @param {string} id The audio block involved with the sound called
86.   */
87.  function playAudio(id) {
88.      volume = (+localStorage.getItem('volume'));
89.      // Check if the volume is muted
90.      if (volume == 0 || mute == 'true') {
91.          return;
92.      }
93.      const audioObj = document.getElementById(id);
94.      audioObj.volume = volume / 100;
95.      audioObj.play();
```

```
96.  }
97.
98.  // /**
99.  //  * Flips the value of mute from string 'true' or 'false'.
100.  //  */
101.  // function toggleMute() {
102.  //     mute = '' + !(mute === 'true');
103.  //     changeMuteIcon();
104.  //     window.localStorage.setItem('mute', mute);
105.  // }
106.
107.  /**
108.   * Changes the icon and ARIA of the mute volume.
109.   */
110.  function changeMuteIcon() {
111.      let volumeIcon = document.getElementById('volumeIcon');
112.      if (mute == 'true') {
113.          if (theme == 'Dark') {
114.              volumeIcon.src = 'img/volume-mute-dark.png';
115.          } else {
116.              volumeIcon.src = 'img/volume-mute.png';
117.          }
118.          volumeIcon.alt = dict['unmute'][lang];
119.      } else {
120.          if (theme == 'Dark') {
121.              volumeIcon.src = 'img/volume-dark.png';
122.          } else {
123.              volumeIcon.src = 'img/volume.png';
124.          }
125.          volumeIcon.alt = dict['mute'][lang];
126.      }
127.  }
128.
```

```
129. // /**
130. //  * Disables or enables the dancing potato animation.
131. //  * @returns Flag if animations are turned on or off
132. //  */
133. // function toggleAnimation() {
134. //     animation = '' + !(animation === 'true');
135. //     if (animation == 'true') {
136. //         document.getElementById('animationBtn').innerText = dict['disableAnimation'][Lang];
137. //     } else {
138. //         hidePotatoes();
139. //         document.getElementById('animationBtn').innerText = dict['enableAnimation'][Lang];
140. //     }
141. //     localStorage.setItem('animation', animation);
142. //     return animation;
143. // }
144.
145. /**
146.  * Sets the input times when the cycle isn't in progress.
147.  * @param {string} phase The phase to set the input times
148.  * @returns {number} The input time in seconds associated with the phase
149.  */
150. function setInputTimes(phase) {
151.     let minutes = document.getElementById(phase + 'Min').value;
152.     let seconds = document.getElementById(phase + 'Sec').value;
153.
154.     return (+minutes) * 60 + (+seconds);
155. }
156.
157. // /**
158. //  * Checks each input after user leaves the input to see if they entered a number
159. //  * beyond the min and max. If so, change the overflowed number to min/max.
160. //  * Locally stores every input, so each place that uses this should take from local storage.
161. //  * @param {number} input
```

```
162. //  */
163. // function checkValue(input) {
164. //     let inputValue = document.getElementById(input).value;
165. //     if (inputValue < 0) {
166. //         inputValue = 0;
167. //     } else if (input == 'volume' && inputValue > 100) {
168. //         inputValue = 100;
169. //     } else if (input != 'volume' && inputValue >= 60) {
170. //         inputValue = 59;
171. //     }
172. //     document.getElementById(input).value = inputValue;
173. //     localStorage.setItem(input, inputValue);
174. // }
175.
176. /**
177.  * Precondition: there must be at least one task in order to start the timer.
178.  * Starts the timer and decrements the timer's MM:SS every second.
179.  * Will save the settings after every time the start button is pressed and hides the
180.  * options to give the user a more focused screen.
181.  * After every work phase, the timer will switch to displaying the break phase and its time.
182.  * If the user completes all of their tasks while the timer is running,
183.  * then the timer stops and a congratulations screen is shown.
184.  */
185. function start() {
186.     if (taskCount == 0) {
187.         return;
188.     }
189.
190.     phase = 'work';
191.
192.     // console.log('Setting input times');
193.     workLength = setInputTimes('work');
194.     shortBreakLength = setInputTimes('short');
```

```
195.     longBreakLength = setInputTimes('long');
196.
197.     saveSettings();
198.     window.localStorage.setItem('savedSettings', true);
199.     secondsRemaining = setTimeRemaining();
200.
201.     document.getElementById('start').innerHTML = dict['stop'][lang];
202.     document.getElementById('start').onclick = function () { confirmationPrompt('Stop'); };
203.     document.getElementById('phaseDisplay').innerHTML = dict['phase'][phase][lang];
204.
205.     //hide task container
206.     hideOptions();
207.
208.     if (taskCount > 0) {
209.         timer = setInterval(function () {
210.             // once all the tasks have ended, clear the timer and show congrats screen
211.             if (taskCount == tasksDone) {
212.                 clearInterval(timer);
213.                 phase = 'idle';
214.                 // Update the phase
215.                 document.getElementById('phaseDisplay').innerHTML = dict['phase'][phase][lang];
216.                 showOptions();
217.                 displayCongrats();
218.                 stop();
219.                 deleteAllTasks();
220.             } else {
221.                 // Display the time MM:SS
222.                 MMSS = convertSeconds(secondsRemaining);
223.                 document.title = setPageTitle(MMSS)+dict['title'][lang];
224.                 document.getElementById('timerDisplay').innerHTML = MMSS;
225.                 secondsRemaining--;
226.
227.                 if (secondsRemaining < 0) {
```

```
228.                     if (phase == 'work') {
229.                         playAudio('workToBreakAudio');
230.                         showOptions();
231.                     }
232.                     if (phase != 'work') {
233.                         playAudio('breakToWorkAudio');
234.                         hideOptions();
235.                     }
236.                     updatePhase();
237.                     secondsRemaining = setTimeRemaining();
238.                     document.getElementById('phaseDisplay').innerHTML = dict['phase'][phase][lang];
239.                     // To change to dark background, need to create a new class
240.                     const background = document.getElementById('background');
241.                     if (theme == 'Potato' && phase == 'short break' || phase == 'long break') {
242.                         background.classList.replace('potatoWork', 'potatoBreak');
243.                     } else if (theme == 'Potato') {
244.                         background.classList.replace('potatoBreak', 'potatoWork');
245.                     }
246.                 }
247.             }
248.         }, 1000); //update the timer every second
249.     }
250. }
251.
252.
253. /**
254.  * Converts the seconds in the remaining time to the format {min}:{sec}
255.  * MM:SS
256.  *
257.  * @param {string} secondsRemaining
258.  */
259. function convertSeconds(secondsRemaining) {
260.     minutes = Math.floor(secondsRemaining / 60);
```

```
261.     seconds = secondsRemaining - (60 * minutes);
262.
263.     var timerString = '';
264.     if (minutes < 10) { timerString = '0'; }
265.     timerString += minutes + ':';
266.     if (seconds < 10) { timerString += '0'; }
267.     timerString += seconds;
268.     return timerString;
269. }
270.
271. /**
272.  * Update the phase and number of tasks complete.
273.  */
274. function updatePhase() {
275.     const circle = document.getElementById('circleTimer');
276.
277.     if (phase == 'work') {
278.         pomosDone++;
279.         localStorage.setItem('pomosDone', pomosDone);
280.         if (theme == 'Potato') {
281.             circle.className = 'circlePotato';
282.             if (animation == true || animation == 'true') {
283.                 showPotatoes();
284.             }
285.         }
286.
287.         if (pomosDone % 4 != 0) {
288.             // If the pomos completed is less than 4 (1-3)
289.             phase = 'short break';
290.             // document.getElementById('cycleNum').innerText = (pomosDone % 4) + ' / 4';
291.
292.             if (theme == 'Potato') {
293.                 circle.className = 'circlePotatoBreak';
```

```
294.         }
295.     } else {
296.         // If the pomos completed is divisible by 4
297.         phase = 'long break';
298.         if (theme == 'Potato') {
299.             if (lang == 'ko' || lang == 'zh') {
300.                 circle.className = 'circlePotatoBreakAsian';
301.             } else {
302.                 circle.className = 'circlePotatoBreak';
303.             }
304.         }
305.     }
306. } else {
307.     if (phase == 'long break') {
308.         if (theme == 'Potato') {
309.             if (lang == 'ko' || lang == 'zh') {
310.                 circle.className = 'circlePotatoBreakAsian';
311.             } else {
312.                 circle.className = 'circlePotatoBreak';
313.             }
314.             hidePotatoes();
315.         }
316.     }
317.     phase = 'work';
318.     if (theme == 'Potato') {
319.         circle.className = 'circlePotato';
320.     }
321. }
322. }
323.
324. /**
325.  * Checks what the current timer state is from
326.  * 'work', 'short break', or 'Long break'
```

```
327.  * to know what the timer should start counting down with.
328.  *
329.  * @return {number} The time remaining for the current timer state.
330.  */
331.  function setTimeRemaining() {
332.      return (phase == 'work') ? workLength :
333.          (phase == 'short break') ? shortBreakLength :
334.              longBreakLength;
335.  }
336.
337.  /**
338.   * Appends a pomotato to the congrats screen
339.   * for each pomo done and displays the congrats screen.
340.   */
341.  function displayCongrats() {
342.      document.getElementById('potatoImgs').innerHTML = '';
343.
344.      // Output potato images to the congrats screen (limit amount of potatoes)
345.      // Max Potatoes: 16
346.      for (let i = 0; i < pomosDone; i++) {
347.          // Prevent Potatoes images from overcrowding screen
348.          if (i > MAX_POTATOES_COMPLETED) {
349.              break;
350.          }
351.          let potato = document.createElement('img');
352.
353.          if (animation == 'true') {
354.              potato.src = 'img/potato-dance.gif';
355.              potato.alt = dict['potatoDance'][lang];
356.          } else {
357.              potato.src = 'img/pomotato.png';
358.              potato.alt = dict['pomotato'][lang];
359.          }
```

```

360.          document.getElementById('potatoImgs').appendChild(potato);
361.      }
362.      document.getElementById("congratsText").innerHTML = dict['congrats1'][lang] + pomosDone + dict['congrats2'][lang];
363.      playAudio('victoryAudio');
364.      show('congratsScreen');
365.  }
366.
367.  /**
368.   * Hides all of the dancing potato gifs.
369.   */
370.  function hidePotatoes() {
371.      document.getElementById('cycle0').style.display = 'none';
372.      document.getElementById('cycle1').style.display = 'none';
373.      document.getElementById('cycle2').style.display = 'none';
374.      document.getElementById('cycle3').style.display = 'none';
375.  }
376.
377.  /**
378.   * Shows a number of dancing potatoes based on the pomosDone.
379.   */
380.  function showPotatoes() {
381.      document.getElementById('cycle' + pomosDone % 4).style.display = 'inline';
382.  }
383.
384.  /**
385.   * Sets the title element for users to see remaining time off-page.
386.   *
387.   * @param {string} MMSS 'MM:SS' form
388.   * @return {string} New page title
389.   */
390.  function setPageTitle(MMSS) {
391.      let phaseSymbol;
392.      switch (phase) {
```

```
393.         case 'work':
394.             phaseSymbol = ' Work - ';
395.             break;
396.         case 'short break':
397.         case 'long break':
398.             phaseSymbol = ' Break - ';
399.             break;
400.         case 'stopped':
401.             phaseSymbol = ' Stopped - '
402.             break;
403.         default:
404.             phaseSymbol = ' - ';
405.     }
406.
407.     return MMSS + phaseSymbol;
408. }
409.
410.
411.
412. /**
413.  * Resets the pomodoro cycle to the beginning.
414.  */
415. function stop() {
416.     // console.log('stop the timer and reset everything');
417.     clearInterval(timer);
418.     let bg = document.getElementById('background');
419.     if (theme == 'Potato' && phase != 'work') {
420.         document.getElementById('circleTimer').className = 'circlePotato';
421.         bg.classList.replace('potatoBreak', 'potatoWork');
422.     }
423.     hidePotatoes();
424.     phase = 'idle';
425.     document.getElementById('timerDisplay').innerHTML = '- - : - -';
```

```
426.     document.getElementById('phaseDisplay').innerHTML = dict['phase'][phase][lang];
427.
428.     document.getElementById('start').innerHTML = dict['start'][lang];
429.     document.getElementById('start').onclick = start;
430.
431.     //tasksDone = 0;
432.     pomosDone = 0;
433.     localStorage.setItem('pomosDone', pomosDone);
434.
435.     uniqueID = 1;
436.     hide('prompt');
437.     showOptions();
438. }
439.
440. /**
441.  * Adds a non-blank task to the list of tasks.
442.  */
443. function addTask() {
444.     const task = document.getElementById('enterTask').value;
445.     document.getElementById('enterTask').value = '';
446.     if (task != '') {
447.         createTask(task);
448.         // console.log('Created task with ID ' + uniqueID);
449.         // console.log('Task count: ' + taskCount);
450.     }
451. }
452.
453. /**
454.  * Creates a userTask in the taskListContainer.
455.  * This does not display a task on the main page.
456.  * A userTask is identified with a unique numerical ID.
457.  * Has four child elements: mark to mark as done, pin a copy to main page,
458.  * delete from task list and main page, if pinned, and task content.
```

```
459.  *
460.  * @event addTask()
461.  * @param {string} text The task the user entered.
462.  */
463.  function createTask(text) {
464.      let taskList = document.getElementById('taskListContainer');
465.      let newTask = document.createElement('div');
466.      newTask.className = 'userTask';
467.      newTask.id = uniqueID;
468.
469.      let markBtn = document.createElement('button');
470.      markBtn.className = 'transparent';
471.      markBtn.setAttribute('aria-label', dict['markBtn'][lang]);
472.
473.      let pinBtn = document.createElement('button');
474.      pinBtn.classList.add('transparent'); //, 'smallIcon');
475.      pinBtn.setAttribute('aria-label', dict['pinBtn'][lang]);
476.
477.      let delBtn = document.createElement('button');
478.      delBtn.classList.add('transparent', 'smallIcon');
479.      delBtn.setAttribute('aria-label', dict['delBtn'][lang]);
480.
481.      if (theme == 'Dark') {
482.          markBtn.innerHTML = '<div class="markCircle markDark" id="mark-' + uniqueID + '"></div>';
483.          pinBtn.innerHTML = '';
484.          delBtn.innerHTML = '';
485.      } else {
486.          markBtn.innerHTML = '<div class="markCircle markLight" id="mark-' + uniqueID + '"></div>';
487.          pinBtn.innerHTML = '';
488.          delBtn.innerHTML = '';
489.      }
490.
491.      markBtn.onclick = function () {
```

```
492.          markedTask = document.getElementById('mark-' + newTask.id);
493.          if (markedTask.classList.contains('markFill')) {
494.              unmark(newTask.id);
495.          } else {
496.              markDone(newTask.id);
497.          }
498.      };
499.
500.      pinBtn.onclick = function () {
501.          origTask = document.getElementById('pin-' + newTask.id);
502.          pinnedTask = document.getElementById(newTask.id + '-copy');
503.          if (!pinnedTask) {
504.              createPinnedTask(text, newTask.id);
505.              if (theme == 'Dark') {
506.                  origTask.src = 'img/pinned-dark.png';
507.                  console.log('dark pin');
508.              } else {
509.                  origTask.src = 'img/pinned.png';
510.              }
511.          } else {
512.              if (theme == 'Dark') {
513.                  origTask.src = 'img/unpinned-dark.png';
514.              } else {
515.                  origTask.src = 'img/unpinned.png';
516.              }
517.              unpinTask(newTask.id);
518.          }
519.      };
520.
521.      delBtn.onclick = function () {
522.          deleteTask(newTask.id);
523.      }
524.
```



```
525.     let content = document.createElement('p');
526.     content.id = 'p' + uniqueID;
527.     content.innerHTML = text;
528.
529.     let ariaSkip = document.createElement('a');
530.     ariaSkip.href = '#' + (uniqueID + 1);
531.     ariaSkip.className = 'ariaSkipTask';
532.     ariaSkip.innerText = dict['skip'][lang];
533.     newTask.appendChild(ariaSkip);
534.
535.     taskCount++;
536.     const taskBtn = document.getElementById('taskBtn');
537.     taskBtn.innerHTML = dict['tasks'][lang] + ' (' + tasksDone + '/' + taskCount + ')';
538.     taskBtn.style.width = "fit-content";
539.
540.     savedTasks.push(text);
541.     // console.Log(JSON.stringify(savedTasks));
542.     localStorage.setItem('savedTasks', JSON.stringify(savedTasks));
543.     // console.Log(localStorage.getItem("savedTasks"));
544.
545.     newTask.appendChild(markBtn);
546.     newTask.appendChild(pinBtn);
547.     newTask.appendChild(content);
548.     newTask.appendChild(delBtn);
549.     taskList.appendChild(newTask);
550.
551.     if (taskCount == 1) {
552.         createPinnedTask(text, uniqueID);
553.         if (theme == 'Dark') {
554.             document.getElementById('pin-' + uniqueID).src = 'img/pinned-dark.png';
555.         } else {
556.             document.getElementById('pin-' + uniqueID).src = 'img/pinned.png';
557.         }
```

```
558.     }
559.
560.     notifyUser('addTask');
561.     return uniqueID++;
562. }
563.
564. /**
565.  * Creates 'pinned' userTask in the mainTasks container.
566.  * This display an existing task on the main page.
567.  * A pinned task is identified as '#pin' where # is the uniqueID.
568.  * Inherits the four userTask components,
569.  * The eventListener for pin is different.
570.  *
571.  * @param {string} text      A copy of the user's task.
572.  * @param {string} uniqueID The existing task's id.
573.  */
574. function createPinnedTask(text, uniqueID) {
575.     let mainTasks = document.getElementById('mainTasks');
576.     let pinTask = document.createElement('div');
577.     pinTask.classList.add('userTask', 'pinnedTask');
578.     pinTask.id = uniqueID + '-copy';
579.
580.     let markBtn = document.createElement('button');
581.     markBtn.className = 'transparent';
582.     markBtn.setAttribute('aria-label', dict['markBtn'][lang]);
583.
584.     let pinBtn = document.createElement('button');
585.     pinBtn.classList.add('transparent'); //, 'smallIcon');
586.     pinBtn.setAttribute('aria-label', dict['pinBtn'][lang]);
587.
588.     // Let delBtn = document.createElement('button');
589.     // delBtn.classList.add('transparent', 'smallIcon');
590.     // delBtn.setAttribute('aria-label', 'Delete this Task');
591.
```

```
592.     if (theme == 'Dark') {
593.         markBtn.innerHTML = '<div class="markCircle markDark" id="mark-' + uniqueID + '-copy"></div>';
594.         pinBtn.innerHTML = '';
595.         //     delBtn.innerHTML = '';
596.     } else {
597.         markBtn.innerHTML = '<div class="markCircle markLight" id="mark-' + uniqueID + '-copy"></div>';
598.         pinBtn.innerHTML = '';
599.         //     delBtn.innerHTML = '';
600.     }
601.
602.     markBtn.onclick = function () {
603.         markedTask = document.getElementById('mark-' + uniqueID);
604.         if (markedTask.classList.contains('markFill')) {
605.             unmark(uniqueID);
606.         } else {
607.             markDone(uniqueID);
608.         }
609.     };
610.
611.     pinBtn.onclick = function () {
612.         unpinTask(uniqueID);
613.     };
614.
615.     // delBtn.onclick = function() {
616.     //     deleteTask(uniqueID);
617.     // }
618.
619.     let content = document.createElement('p');
620.     content.innerHTML = text;
621.
622.     let ariaSkip = document.createElement('a');
623.     ariaSkip.href = '#' + (uniqueID + 1) + '-copy';
624.     ariaSkip.className = 'ariaSkipTask';
```

```
625.     ariaSkip.innerText = dict['skip'][lang];
626.     pinTask.appendChild(ariaSkip);
627.
628.     pinTask.appendChild(markBtn);
629.     pinTask.appendChild(pinBtn);
630.     pinTask.appendChild(content);
631.     // pinTask.appendChild(delBtn);
632.
633.     mainTasks.appendChild(pinTask);
634.
635.     if (document.getElementById('mark-' + uniqueID).classList.contains('markFill')) {
636.         document.getElementById('mark-' + uniqueID + '-copy').classList.add('markFill');
637.     }
638.     setPinnedSkip();
639.     notifyUser('pinTask');
640.     return uniqueID;
641. }
642.
643. /**
644.  * Visually marks a task if a user completes the task.
645.  * This affects the task list and main display, if possible.
646.  * Increments the number of tasks completed.
647.  * @param {string} uniqueID The existing task's (task list) id.
648.  */
649. function markDone(uniqueID) {
650.     let originalTask = document.getElementById(uniqueID);
651.     document.getElementById('mark-' + uniqueID).classList.add('markFill');
652.     let pinnedTask = document.getElementById(uniqueID + '-copy');
653.
654.     if (pinnedTask) {
655.         document.getElementById('mark-' + uniqueID + '-copy').classList.add('markFill');
656.     }
657.     tasksDone++;
```

```
658.     originalTask.setAttribute('marked', 'true');
659.     const taskBtn = document.getElementById('taskBtn');
660.     taskBtn.innerHTML = dict['tasks'][lang] + ' (' + tasksDone + '/' + taskCount + ')';
661.     // console.Log('Tasks done: ' + tasksDone);
662.     notifyUser('mark');
663. }
664.
665. /**
666.  * Visually unmarks a task if a user did not complete the task.
667.  * This affects the task list and main display, if possible.
668.  * Decrements the number of tasks complete.
669.  * @param {string} uniqueID The existing task's (task list) id.
670.  */
671. function unmark(uniqueID) {
672.     let originalTask = document.getElementById(uniqueID);
673.     document.getElementById('mark-' + uniqueID).classList.remove('markFill');
674.     let pinnedTask = document.getElementById(uniqueID + '-copy');
675.
676.     if (pinnedTask) {
677.         document.getElementById('mark-' + uniqueID + '-copy').classList.remove('markFill');
678.     }
679.     tasksDone--;
680.     originalTask.setAttribute('marked', 'false');
681.     const taskBtn = document.getElementById('taskBtn');
682.     taskBtn.innerHTML = dict['tasks'][lang] + ' (' + tasksDone + '/' + taskCount + ')';
683.     // console.Log('Tasks done: ' + tasksDone);
684.     notifyUser('unmark');
685. }
686.
687. /**
688.  * Unpins a task from the main display by deleting the pinned copy.
689.  * @param {string} uniqueID The existing task's (task list) id.
690.  *
```

```
691.  * @example Unpin pinned task '1pin' calls function with '1pin'
692.  */
693. function unpinTask(uniqueID) {
694.     let pinnedTask = document.getElementById(uniqueID + '-copy');
695.     const mainTasks = document.getElementById('mainTasks');
696.     mainTasks.removeChild(pinnedTask);
697.     if (theme == 'Dark') {
698.         document.getElementById('pin-' + uniqueID).src = 'img/unpinned-dark.png';
699.     } else {
700.         document.getElementById('pin-' + uniqueID).src = 'img/unpinned.png';
701.     }
702.     notifyUser('unpinTask');
703.     setPinnedSkip();
704. }
705.
706. /**
707.  * Deletes a task from both the task list and the main display, if possible.
708.  * Decreases the number of tasks by one.
709.  * @param {string} uniqueID The existing task's (task list) id.
710.  *
711.  * @example Delete pinned task '1pin' calls function with '1'.
712.  */
713. function deleteTask(uniqueID) {
714.     let taskText = document.getElementById('p' + uniqueID).innerText;
715.
716.     const pinnedTask = document.getElementById(uniqueID + '-copy');
717.
718.     if (document.getElementById(uniqueID).getAttribute('marked') == 'true') {
719.         tasksDone--;
720.     }
721.
722.     if (pinnedTask) {
723.         const mainTasks = document.getElementById('mainTasks');
```

```
724.     mainTasks.removeChild(pinnedTask);
725.     // console.log('Deleted a pinned task.');
726. }
727. const taskListContainer = document.getElementById('taskListContainer');
728. taskListContainer.removeChild(document.getElementById(uniqueID));
729. taskCount--;
730.
731. const taskBtn = document.getElementById('taskBtn');
732. taskBtn.innerHTML = dict['tasks'][lang] + ' (' + tasksDone + '/' + taskCount + ')';
733.
734. if (taskCount == 0) {
735.     taskBtn.innerHTML = dict['tasks'][lang];
736. }
737.
738. // ARIA
739. setARIASkip();
740.
741. savedTasks.splice(savedTasks.indexOf(taskText), 1);
742. localStorage.setItem('savedTasks', JSON.stringify(savedTasks));
743.
744. notifyUser('delTask');
745. // console.log('Task count: ' + taskCount);
746. }
747.
748. /**
749.  * Deletes all of the tasks from both the tsak List and main display, if possible.
750.  * Resets taskCount to 0 and uniqueID to 1.
751.  * @event deleteAll text
752.  */
753. function deleteAllTasks() {
754.     const taskListContainer = document.getElementById('taskListContainer');
755.     while (taskListContainer.firstChild) {
756.         taskListContainer.removeChild(taskListContainer.lastChild);
```

```
757.     }
758.
759.     const mainTasks = document.getElementById('mainTasks');
760.     while (mainTasks.firstChild) {
761.         mainTasks.removeChild(mainTasks.lastChild);
762.     }
763.
764.     taskCount = 0;
765.     tasksDone = 0;
766.     uniqueID = 1;
767.     const taskBtn = document.getElementById('taskBtn');
768.
769.     if (taskCount == 0) {
770.         taskBtn.innerHTML = dict['tasks'][lang];
771.     }
772.     savedTasks = [];
773.     localStorage.removeItem('savedTasks');
774.     hide('prompt');
775.     notifyUser('deleteAll');
776.     // console.log('Deleted all tasks.');
777.     // console.log('Task Count: ' + taskCount);
778. }
779.
780. /**
781.  * @event deleteTask()
782.  * Updates all userTasks in the task List so their ARIA skip Links will Link
783.  * To the next task based on the next task's ID.
784.  */
785. function setARIASkip() {
786.     let taskListContainer = document.getElementById('taskListContainer');
787.     let userTaskList = taskListContainer.children; // <a List of userTask nodes
788.     for (let i = 0; i < userTaskList.length - 1; i++) {
789.         let userTask = userTaskList[i]; // The first userTask
```

```
790.         let nextTask = userTaskList[i + 1];
791.         userTask.firstChild.href = '#' + nextTask.id;
792.     }
793. }
794.
795. /**
796.  * @event unpinTask()
797.  * Updates all userTasks in the main task list so their ARIA skip links will link
798.  * To the next task based on the next task's ID.
799.  */
800. function setPinnedSkip() {
801.     let mainTasks = document.getElementById('mainTasks');
802.     let pinnedTaskList = mainTasks.children;
803.     for (let i = 0; i < pinnedTaskList.length - 1; i++) {
804.         let pinnedTask = pinnedTaskList[i];
805.         let nextTask = pinnedTaskList[i + 1];
806.         pinnedTask.firstChild.href = '#' + nextTask.id;
807.     }
808. }
809.
810. /**
811.  * Confirms a user's action to prevent major accidents.
812.  * @param {string} action The action to confirm. Either 'Reset' or 'Delete' all.
813.  */
814. function confirmationPrompt(action) {
815.     // console.log('prompt');
816.     show('prompt');
817.     let message = document.getElementById('confirmMessage');
818.     let confirmBtn = document.getElementById('confirm');
819.
820.     if (action == 'Stop') {
821.         message.innerHTML = dict['confirmReset'][lang];
822.         confirmBtn.onclick = stop;
```

```
823.     } else if (action == 'Delete') {
824.         message.innerHTML = dict['confirmDeleteAll'][lang];
825.         confirmBtn.onclick = deleteAllTasks;
826.     }
827. }
828.
829. /**
830.  * Shows an element by changing its display to block.
831.  * @param {string} id The id of the element to show.
832.  */
833. function show(id) {
834.     const elem = document.getElementById(id);
835.     // console.log('showing');
836.     elem.classList.replace('hidden', 'showing');
837. }
838.
839. /**
840.  * Hides an element by changing its display to none.
841.  * also saves settings if the element to be hidden is the settings menu
842.  * @param {string} id The id of the element to hide.
843.  */
844. function hide(id) {
845.     const elem = document.getElementById(id);
846.     if(id == 'settingsMenu') {
847.         saveSettings();
848.     }
849.     // console.log('hiding');
850.     elem.classList.replace('showing', 'hidden');
851. }
852.
853. /**
854.  * @event stop()
855.  * Shows the various options and buttons available to the user.
```

```
856.      * Triggers when a user presses the stop button or reaches the congrats screen.
857.      */
858.      function showOptions() {
859.          document.getElementById('help').classList.replace('opacityHide', 'opacityShow');
860.          document.getElementById('settingsIcon').classList.replace('opacityHide', 'opacityShow');
861.          document.getElementById('enterTask').classList.replace('opacityHide', 'opacityShow');
862.          document.getElementById('taskAdder').classList.replace('opacityHide', 'opacityShow');
863.          document.getElementById('taskBtn').classList.replace('opacityHide', 'opacityShow');
864.
865.          // Let options = document.getElementsByClassName('opacityHide');
866.          // for(let i = 0; i < options.length; i++) {
867.          //     console.log(options);
868.          //     options[i].classList.remove('opacityHide');
869.          //     options[i].classList.add('opacityShow');
870.          // }
871.      }
872.
873.      /**
874.       * @event start()
875.       * Hides the various options and buttons available to the user.
876.       * Triggers when a user clicks the start button.
877.       */
878.      function hideOptions() {
879.          document.getElementById('help').classList.replace('opacityShow', 'opacityHide');
880.          document.getElementById('settingsIcon').classList.replace('opacityShow', 'opacityHide');
881.          document.getElementById('enterTask').classList.replace('opacityShow', 'opacityHide');
882.          document.getElementById('taskAdder').classList.replace('opacityShow', 'opacityHide');
883.          document.getElementById('taskBtn').classList.replace('opacityShow', 'opacityHide');
884.          // Let options = document.getElementsByClassName('opacityShow');
885.          // for(let i = 0; i < options.length; i++) {
886.          //     console.log(options);
887.          //     options[i].classList.remove('opacityShow');
888.          //     options[i].classList.add('opacityHide');
```

```
889.          // }
890.      }
891.
892.
893.      var page = 0;
894.      /**
895.       * Goes to the previous page of the instructions menu
896.       */
897.      function back() {
898.          if (page <= 1) {
899.              return;
900.          }
901.          --page;
902.          let topic = document.getElementById('instrTopic');
903.          topic.innerText = dict[page][topic.id][lang];
904.          let content = document.getElementById('instrContent');
905.          content.innerText = dict[page][content.id][lang];
906.          if (page != 4) {
907.              content.classList.add('leftAlign');
908.          }
909.          document.getElementById('page').innerText = page + ' / 4';
910.          document.getElementById('next').innerHTML = dict['next'][lang];
911.
912.      }
913.
914.      /**
915.       * Goes to the next page of the instructions menu
916.       */
917.      function next() {
918.          document.getElementById('next').innerHTML = dict['next'][lang];
919.          if (page >= 4) {
920.              hide('instructionsMenu');
921.              page = 0;
```

```
922.         // document.getElementById('next').innerHTML = ;
923.         return;
924.     }
925.     page++;
926.     let topic = document.getElementById('instrTopic');
927.     topic.innerText = dict[page][topic.id][lang];
928.     let content = document.getElementById('instrContent');
929.     content.innerText = dict[page][content.id][lang];
930.     document.getElementById('page').innerText = page + ' / 4';
931.     if (page == 4) {
932.         content.classList.remove('leftAlign');
933.         content.innerText = content.innerText +
934.             'Alexis Chen\nElizabeth Cho\nKevin Jang\nMarco Kuan\nAhmad Milad\nRohan Patel\nMiaoqiu Sun\nJessie Zou\n';
935.         if (theme == 'Dark') {
936.             content.innerHTML = content.innerHTML + '<a href="https://github.com/AlexisChen99/cse110-w21-group4"></a>';
937.         } else {
938.             content.innerHTML = content.innerHTML + '<a href="https://github.com/AlexisChen99/cse110-w21-group4"></a>';
939.         }
940.         document.getElementById('next').innerHTML = dict['close'][lang];
941.     }
942. }
943. /**
944.  * Creates a notification for the user based on what action the user just did
945.  * @param {string} action the action the user did
946.  * @returns the action the user did
947.  */
948. function notifyUser(action) {
949.     let notif = document.getElementById('notificationBar');
950.     notif.innerText = dict['notification'][action][lang];
951.     setTimeout(function () {
952.         notif.innerText = '';
953.     }, 3000);
954.     return action;
```

```
955. }
956.
957.
958. /**
959.  * Loads a user's last theme and settings selected locally.
960.  */
961. function loadTheme() {
962.     switch (window.localStorage.getItem('theme')) {
963.         case 'Potato':
964.             changeTheme('Potato');
965.             break;
966.         case 'Dark':
967.             changeTheme('Dark');
968.             break;
969.         case 'Light':
970.             changeTheme('Light');
971.             break;
972.         default:
973.             //console.log('no previous theme');
974.             changeTheme('Potato');
975.     }
976. }
977.
978. /**
979.  * Changes the theme and stores the new theme locally.
980.  * @event LoadTheme()
981.  * @event button
982.  * @param {string} newTheme The theme to change to.
983.  */
984. function changeTheme(newTheme) {
985.     window.localStorage.setItem('theme', newTheme);
986.     theme = newTheme;
987.     const body = document.getElementById('background');
```

```
988. body.className = 'theme' + newTheme;
989.
990. if (newTheme == 'Potato') {
991.     body.classList.add('potatoWork');
992. } else {
993.     hidePotatoes();
994. }
995.
996. const circle = document.getElementById('circleTimer');
997. circle.className = 'circle' + newTheme;
998.
999. if (newTheme == 'Dark') {
1000.     let settingsIcon = document.getElementById('settingsIcon');
1001.     settingsIcon.classList.replace('settingsLight', 'settingsDark');
1002.
1003.     let volumeIcon = document.getElementById('volumeIcon');
1004.     if (volume != 0) {
1005.         volumeIcon.src = 'img/volume-dark.png';
1006.     } else {
1007.         volumeIcon.src = 'img/volume-mute-dark.png';
1008.     }
1009.
1010.     let buttons = document.getElementsByClassName('mainButton');
1011.     for (let i = 0; i < buttons.length; i++) {
1012.         buttons[i].classList.add('darkButton');
1013.     }
1014.
1015.     let transparentBtns = document.getElementsByClassName('transparent');
1016.     for (let i = 0; i < transparentBtns.length; i++) {
1017.         transparentBtns[i].classList.remove('textDark');
1018.     }
1019.
1020.     let menus = document.getElementsByClassName('menu');
```

```
1021.     for (let i = 0; i < menus.length; i++) {
1022.         menus[i].classList.add('themeDark');
1023.         if (menus[i].classList.contains('menuLight')) {
1024.             menus[i].classList.remove('menuLight');
1025.         }
1026.     }
1027.
1028.     let prompt = document.getElementById('prompt');
1029.     prompt.classList.add('themeDark');
1030.     if (prompt.classList.contains('themeLight')) {
1031.         prompt.classList.remove('themeLight');
1032.     }
1033.
1034.     let congrats = document.getElementById('congratsContent');
1035.     congrats.classList.add('modalDark');
1036.     congrats.classList.remove('modalLight', 'modalPotato');
1037.
1038.     let userTasks = document.getElementsByClassName('userTask');
1039.     for (let i = 0; i < userTasks.length; i++) {
1040.         userTasks[i].children[1].firstChild.classList.replace('markLight', 'markDark');
1041.         let pinSrc = userTasks[i].children[2].firstChild.src;
1042.         if (pinSrc.includes('img/pinned.png')) {
1043.             userTasks[i].children[2].firstChild.src = 'img/pinned-dark.png';
1044.         } else {
1045.             userTasks[i].children[2].firstChild.src = 'img/unpinned-dark.png';
1046.         }
1047.         if (userTasks[i].children[4]) {
1048.             userTasks[i].children[4].firstChild.src = 'img/delete-task-dark.png';
1049.         }
1050.     }
1051. } else if (newTheme == 'Potato' || newTheme == 'Light') {
1052.     let settingsIcon = document.getElementById('settingsIcon');
1053.     settingsIcon.classList.replace('settingsDark', 'settingsLight');
1054. }
```



```
1055.     let volumeIcon = document.getElementById('volumeIcon');
1056.     if (volume !== 0) {
1057.         volumeIcon.src = 'img/volume.png';
1058.     } else {
1059.         volumeIcon.src = 'img/volume-mute.png';
1060.     }
1061.
1062.     let buttons = document.getElementsByClassName('mainButton');
1063.     for (let i = 0; i < buttons.length; i++) {
1064.         if (buttons[i].classList.contains('darkButton')) {
1065.             buttons[i].classList.remove('darkButton');
1066.         }
1067.     }
1068.
1069.     let transparentBtns = document.getElementsByClassName('transparent');
1070.     for (let i = 0; i < transparentBtns.length; i++) {
1071.         transparentBtns[i].classList.add('textDark');
1072.     }
1073.
1074.     let menus = document.getElementsByClassName('menu');
1075.     for (let i = 0; i < menus.length; i++) {
1076.         menus[i].classList.add('menuLight');
1077.         if (menus[i].classList.contains('themeDark')) {
1078.             menus[i].classList.remove('themeDark');
1079.         }
1080.     }
1081.
1082.     let prompt = document.getElementById('prompt');
1083.     prompt.classList.add('themeLight');
1084.     if (prompt.classList.contains('themeDark')) {
1085.         prompt.classList.remove('themeDark');
1086.     }
1087.
```

```
1088.     let congrats = document.getElementById('congratsContent');
1089.     congrats.classList.remove('modalDark');
1090.     if(theme === 'Light') {
1091.         congrats.classList.add('modalLight');
1092.     } else {
1093.         congrats.classList.add('modalPotato');
1094.     }
1095.
1096.     let userTasks = document.getElementsByClassName('userTask');
1097.     for (let i = 0; i < userTasks.length; i++) {
1098.         // console.log('changing tasks');
1099.         userTasks[i].children[1].firstChild.classList.replace('markDark', 'markLight');
1100.         let pinSrc = userTasks[i].children[2].firstChild.src;
1101.         if (pinSrc.includes('img/pinned-dark.png')) {
1102.             userTasks[i].children[2].firstChild.src = 'img/pinned.png';
1103.         } else {
1104.             userTasks[i].children[2].firstChild.src = 'img/unpinned.png';
1105.         }
1106.         if (userTasks[i].children[4]) {
1107.             userTasks[i].children[4].firstChild.src = 'img/delete-task.png';
1108.         }
1109.     }
1110. }
1111.
1112. if (theme === 'Potato') {
1113.     show('animationBtn');
1114. } else {
1115.     hide('animationBtn');
1116.     hidePotatoes();
1117. }
1118. //hide('settingsMenu');
1119. }
1120.
```

```
1121.  /**
1122.   * @event button
1123.   * Changes the chosen Language of Potato Timer
1124.   * @param {string} selectedLang the language the user wishes to see potatotimer in
1125.   */
1126.  function setLang(selectedLang) {
1127.      window.localStorage.setItem('lang', selectedLang);
1128.      window.location.reload();
1129.  }
1130.
1131.  /**
1132.   * Changes all of the elements of the DOM into the proper language.
1133.   * Stores the new Language in Local storage.
1134.   * The default Language is English.
1135.   */
1136.  function loadLang() {
1137.      let savedLang = window.localStorage.getItem('lang');
1138.      if (savedLang == null) {
1139.          // console.log("No saved Language detected. Your browser's Language is: " + navigator.Language);
1140.          if (navigator.language.includes('es')) {
1141.              lang = 'es';
1142.          } else if (navigator.language.includes('zh')) {
1143.              lang = 'zh';
1144.          } else if (navigator.language == 'ko') {
1145.              lang = 'ko';
1146.          } else {
1147.              lang = 'en';
1148.          }
1149.          window.localStorage.setItem('lang', lang);
1150.      } else {
1151.          lang = savedLang;
1152.      }
1153.  }
```

```
1154.      document.documentElement.lang = lang; // <HTML> tag
1155.      document.title = dict['title'][lang];
1156.      document.getElementById('title').innerText = dict['title'][lang];
1157.      document.getElementById('help').setAttribute('aria-label', dict['help'][lang]);
1158.      document.getElementById('settingsIcon').setAttribute('aria-label', dict['openSettings'][lang]);
1159.      document.getElementById('phaseDisplay').innerText = dict['phase']['idle'][lang];
1160.      document.getElementById('start').innerText = dict['start'][lang];
1161.      document.getElementById('taskBtn').innerText = dict['tasks'][lang];
1162.      document.getElementById('enterTask').placeholder = dict['enterTask'][lang];
1163.      document.getElementById('taskAdder').innerText = dict['add'][lang];
1164.
1165.      document.getElementById('settingsTitle').innerText = dict['settings'][lang];
1166.      document.getElementById('closeSettings').innerText = dict['close'][lang];
1167.      document.getElementById('selectTheme').innerText = dict['selectTheme'][lang];
1168.      document.getElementById('lightTheme').innerText = dict['lightTheme'][lang];
1169.      document.getElementById('darkTheme').innerText = dict['darkTheme'][lang];
1170.      document.getElementById('potatoTheme').innerText = dict['potatoTheme'][lang];
1171.      document.getElementById('workTime').innerText = dict['workTime'][lang];
1172.      document.getElementById('shortTime').innerText = dict['shortBreak'][lang];
1173.      document.getElementById('longTime').innerText = dict['longBreak'][lang];
1174.      // document.getElementById('cycleLength').innerText = dict['cycleLength'][lang];
1175.      document.getElementById('volumeTitle').innerText = dict['volume'][lang];
1176.
1177.      document.getElementById('tasksTitle').innerText = dict['tasks'][lang];
1178.      document.getElementById('taskHelp').innerText = dict['taskHelp'][lang];
1179.      document.getElementById('closeTasks').innerText = dict['close'][lang];
1180.      let close = document.getElementsByClassName('ariaClose');
1181.      for (let i = 0; i < close.length; i++) {
1182.          close[i].innerText = dict['close'][lang];
1183.      }
1184.      document.getElementById('deleteAll').innerText = dict['deleteAll'][lang];
1185.
1186.      document.getElementById('confirm').innerText = dict['confirm'][lang];
```

```
1187.     document.getElementById('cancel').innerText = dict['cancel'][lang];
1188.     document.getElementById('congratsTitle').innerText = dict['congratsTitle'][lang];
1189.     document.getElementById('instrTitle').innerText = dict['instructions'][lang];
1190.     document.getElementById('back').innerText = dict['back'][lang];
1191.     document.getElementById('next').innerText = dict['next'][lang];
1192.     if (animation) {
1193.         document.getElementById('animationBtn').innerText = dict['disableAnimation'][lang];
1194.     } else {
1195.         document.getElementById('animationBtn').innerText = dict['enableAnimation'][lang];
1196.     }
1197.     if (lang == 'es') {
1198.         document.getElementById('settingsTitle').style.fontSize = "32px";
1199.         document.getElementById('closeSettings').style.fontSize = "17px";
1200.         var elements = document.getElementsByClassName('fieldLabel');
1201.         for (var i = 0; i < elements.length; i++) {
1202.             var element = elements[i];
1203.             element.style.fontSize = "16.5px";
1204.         }
1205.     }
1206.     document.getElementById('cycle0').innerText = dict['potatoDance'][lang];
1207.     document.getElementById('cycle1').innerText = dict['potatoDance'][lang];
1208.     document.getElementById('cycle2').innerText = dict['potatoDance'][lang];
1209.     document.getElementById('cycle3').innerText = dict['potatoDance'][lang];
1210.     document.getElementById('notificationBar').innerText = dict['notification']['welcome'][lang];
1211.
1212. }
1213.
1214. /**
1215.  * Loads the tasks from local storage and creates them again.
1216.  */
1217. function loadTasks() {
1218.     let savedTasks = JSON.parse(localStorage.getItem('savedTasks'));
1219.     if (!savedTasks) {
```

```
1220.         return;
1221.     }
1222.
1223.     for (let i = 0; i < savedTasks.length; i++) {
1224.         createTask(savedTasks[i]);
1225.     }
1226.
1227.     //Pomo Data
1228.     if (localStorage.getItem('pomosDone') != null) {
1229.         pomosDone = localStorage.getItem('pomosDone');
1230.     } else {
1231.         console.log('no previous pomos');
1232.     }
1233. }
1234.
1235. /**
1236.  * Loads all of the user-custom settings in the settings menu.
1237.  */
1238. function loadSettings() {
1239.     document.getElementById('workMin').value = (+localStorage.getItem('workMin'));
1240.     document.getElementById('workSec').value = (+localStorage.getItem('workSec'));
1241.     document.getElementById('shortMin').value = (+localStorage.getItem('shortMin'));
1242.     document.getElementById('shortSec').value = (+localStorage.getItem('shortSec'));
1243.     document.getElementById('longMin').value = (+localStorage.getItem('longMin'));
1244.     document.getElementById('longSec').value = (+localStorage.getItem('longSec'));
1245.     document.getElementById('volume').value = (+localStorage.getItem('volume'));
1246.     mute = window.localStorage.getItem('mute');
1247.     changeMuteIcon();
1248.     animation = window.localStorage.getItem('animation');
1249.     if (animation == 'true') {
1250.         document.getElementById('animationBtn').innerText = dict['disableAnimation'][lang];
1251.     } else {
1252.         document.getElementById('animationBtn').innerText = dict['enableAnimation'][lang];
```

```
1253.     }
1254. }
1255.
1256. /**
1257.  * @event closeSettings The close button(s) on settings is pressed.
1258.  * @event start()
1259.  * Stores all of the current settings into LocalStorage.
1260.  */
1261. function saveSettings() {
1262.     //timer phase settings
1263.     localStorage.setItem('workMin', document.getElementById('workMin').value);
1264.     localStorage.setItem('workSec', document.getElementById('workSec').value);
1265.     localStorage.setItem('shortMin', document.getElementById('shortMin').value);
1266.     localStorage.setItem('shortSec', document.getElementById('shortSec').value);
1267.     localStorage.setItem('longMin', document.getElementById('longMin').value);
1268.     localStorage.setItem('longSec', document.getElementById('longSec').value);
1269.     //volume settings
1270.     localStorage.setItem('volume', document.getElementById('volume').value);
1271.     localStorage.setItem('mute', mute);
1272.     //animation settings
1273.     localStorage.setItem('animation', animation);
1274. }
1275.
1276. /**
1277.  * (For Testing)
1278.  * Manually sets the phase.
1279.  * @param {string} newPhase The phase to change to.
1280.  */
1281. function setPhase(newPhase) {
1282.     phase = newPhase;
1283. }
1284.
1285. module.exports = {
```

```
1286.     setPhase,
1287.     convertSeconds,
1288.     setTimeRemaining,
1289.     setpageTitle
1290. }
```