# Source: app.js

```
/* global dict:readonly */
/*
 * Audio
 * newPM is a whistle
 * GenericNotify is a chime (light/dark theme)
 * tournament is clapping (congrats)
 * lowTime is a low bell
 */
/*
 * Phases: idle, work, short break, long break (lowercase)
 *
 * i = idle, W = work, b = short break, B = long break
 *
 * i-W-b-W-b-W-b-W-B (1 cycle)
 * i-0-1-0-1-0-1-0-2
 */
let lang;
let phase = 'idle';         // idle, work, short break, long break, stopped
let workLength = 900;             // work time (seconds)   15 mins (900)
let shortBreakLength = 300;       // short break time      5 mins  (300)
let longBreakLength = 1500;        // long break time       25 mins (1500)
let timer;                  // Represents the interval change of 1s
let secondsRemaining = 0;   // Displays on timer
let MMSS;                   // {string} MM:SS format
let tasksDone = 0;          // the number of tasks finished
let pomosDone = 0;          // Number of pomo 'work' phases completed.
let taskCount = 0;          // Used to keep track of all active tasks
var uniqueID = 1;           // Used to assign uniqueID's when deleting specific tasks
let savedTasks = [];
```

```
let volume = 25;
let theme;                  // Potato, Dark, Light, undefined (Capitalized)
let mute = false;           // whether the volume is muted
let animation = true;           // show dancing potatoes or still potatoes

const MAX_POTATOES_COMPLETED = 15;
/**
 * Adds event listeners to the congratulations screen so a user can click anywhere outside the screen
 * To close it.
 * Loads the user's locally saved theme and language.
 * If it is the user's first time, the instructions menu will automatically show up.
 * Otherwise, the user's tasks and saved settings will be loaded from local storage.
 */
window.onload = function () {
    let congrats = document.getElementById('congratsScreen');

    //make addTask execute on enter press
    window.addEventListener('keypress', function (e) {
        //if the pressed key is the enter key
        if (e.key == 'Enter') {
            addTask();
        }
    });

    // When the user clicks anywhere outside of the modal, close the congrats
    // screen
    window.onclick = function (event) {
        if (event.target == congrats) {
            hide('congratsScreen');
        }
    }

    /**
```

```
     * Function that stores themes, languages, and adds tasks back into the
     * website
     */
    // Load's users theme and laguage and settings
    loadTheme();
    loadLang();
    if (window.localStorage.getItem('returning') == 'true') {
        loadTasks();
        if(window.localStorage.getItem('savedSettings') == 'true') {
            loadSettings();
        }
    } else {
        show('instructionsMenu');
        next();
        window.localStorage.setItem('returning', true);
    }
}

/**
 * Play the audio from break to work
 * 'breakToWorkAudio', 'workToBreakAudio', 'victoryAudio'
 *
 * @param {string} id The audio block involved with the sound called
 */
function playAudio(id) {
    volume = (+localStorage.getItem('volume'));
    // Check if the volume is muted
    if (volume == 0 || mute == 'true') {
        return;
    }
    const audioObj = document.getElementById(id);
    audioObj.volume = volume / 100;
    audioObj.play();
}

// /**
//  * Flips the value of mute from string 'true' or 'false'.
//  */
// function toggleMute() {
//     mute = '' + !(mute === 'true');
//     changeMuteIcon();
//     window.localStorage.setItem('mute', mute);
// }

/**
 * Changes the icon and ARIA of the mute volume.
 */
function changeMuteIcon() {
    let volumeIcon = document.getElementById('volumeIcon');
    if (mute == 'true') {
        if (theme == 'Dark') {
            volumeIcon.src = 'img/volume-mute-dark.png';
        } else {
            volumeIcon.src = 'img/volume-mute.png';
        }
        volumeIcon.alt = dict['unmute'][lang];
    } else {
        if (theme == 'Dark') {
            volumeIcon.src = 'img/volume-dark.png';
        } else {
            volumeIcon.src = 'img/volume.png';
        }
        volumeIcon.alt = dict['mute'][lang];
    }
}

```

```javascript
// /**
//  * Disables or enables the dancing potato animation.
//  * @returns Flag if animations are turned on or off
//  */
// function toggleAnimation() {
//     animation = '' + !(animation === 'true');
//     if (animation == 'true') {
//         document.getElementById('animationBtn').innerText = dict['disableAnimation'][lang];
//     } else {
//         hidePotatoes();
//         document.getElementById('animationBtn').innerText = dict['enableAnimation'][lang];
//     }
//     localStorage.setItem('animation', animation);
//     return animation;
// }

/**
 * Sets the input times when the cycle isn't in progress.
 * @param {string} phase The phase to set the input times
 * @returns {number} The input time in seconds associated with the phase
 */
function setInputTimes(phase) {
    let minutes = document.getElementById(phase + 'Min').value;
    let seconds = document.getElementById(phase + 'Sec').value;

    return (+minutes) * 60 + (+seconds);
}

// /**
//  * Checks each input after user leaves the input to see if they entered a number
//  * beyond the min and max. If so, change the overflowed number to min/max.
//  * Locally stores every input, so each place that uses this should take from local storage.
//  * @param {number} input
//  */
// function checkValue(input) {
//     let inputValue = document.getElementById(input).value;
//     if (inputValue < 0) {
//         inputValue = 0;
//     } else if (input == 'volume' && inputValue > 100) {
//         inputValue = 100;
//     } else if (input != 'volume' && inputValue >= 60) {
//         inputValue = 59;
//     }
//     document.getElementById(input).value = inputValue;
//     localStorage.setItem(input, inputValue);
// }

/**
 * Precondition: there must be at least one task in order to start the timer.
 * Starts the timer and decrements the timer's MM:SS every second.
 * Will save the settings after every time the start button is pressed and hides the
 * options to give the user a more focused screen.
 * After every work phase, the timer will switch to displaying the break phase and its time.
 * If the user completes all of their tasks while the timer is running,
 * then the timer stops and a congratulations screen is shown.
 */
function start() {
    if (taskCount == 0) {
        return;
    }

    phase = 'work';

    // console.log('Setting input times');
    workLength = setInputTimes('work');
    shortBreakLength = setInputTimes('short');
    longBreakLength = setInputTimes('long');

    saveSettings();
    window.localStorage.setItem('savedSettings', true);
    secondsRemaining = setTimeRemaining();

    document.getElementById('start').innerHTML = dict['stop'][lang];
    document.getElementById('start').onclick = function () { confirmationPrompt('Stop'); };
    document.getElementById('phaseDisplay').innerHTML = dict['phase'][phase][lang];

    //hide task container
    hideOptions();

    if (taskCount > 0) {
        timer = setInterval(function () {
            // once all the tasks have ended, clear the timer and show congrats screen
            if (taskCount == tasksDone) {
                clearInterval(timer);
                phase = 'idle';
                // Update the phase
                document.getElementById('phaseDisplay').innerHTML = dict['phase'][phase][lang];
                showOptions();
                displayCongrats();
                stop();
                deleteAllTasks();
            } else {
                // Display the time MM:SS
                MMSS = convertSeconds(secondsRemaining);
                document.title = setPageTitle(MMSS)+dict['title'][lang];
                document.getElementById('timerDisplay').innerHTML = MMSS;
                secondsRemaining--;

                if (secondsRemaining < 0) {
                    if (phase == 'work') {
                        playAudio('workToBreakAudio');
                        showOptions();
                    }
                    if (phase != 'work') {
                        playAudio('breakToWorkAudio');
                        hideOptions();
                    }
                    updatePhase();
                    secondsRemaining = setTimeRemaining();
                    document.getElementById('phaseDisplay').innerHTML = dict['phase'][phase][lang];
                    // To change to dark background, need to create a new class
                    const background = document.getElementById('background');
                    if (theme == 'Potato' && phase == 'short break' || phase == 'long break') {
                        background.classList.replace('potatoWork', 'potatoBreak');
                    } else if (theme == 'Potato') {
                        background.classList.replace('potatoBreak', 'potatoWork');
                    }
                }
            }
        }, 1000); //update the timer every second
    }
}


/**
 * Converts the seconds in the remaining time to the format {min}:{sec}
 * MM:SS
 *
 * @param {string} secondsRemaining
 */
function convertSeconds(secondsRemaining) {
    minutes = Math.floor(secondsRemaining / 60);
```

```javascript
    seconds = secondsRemaining - (60 * minutes);

    var timerString = '';
    if (minutes < 10) { timerString = '0'; }
    timerString += minutes + ':';
    if (seconds < 10) { timerString += '0'; }
    timerString += seconds;
    return timerString;
}

/**
 * Update the phase and number of tasks complete.
 */
function updatePhase() {
    const circle = document.getElementById('circleTimer');

    if (phase == 'work') {
        pomosDone++;
        localStorage.setItem('pomosDone', pomosDone);
        if (theme == 'Potato') {
            circle.className = 'circlePotato';
            if (animation == true || animation == 'true') {
                showPotatoes();
            }
        }

        if (pomosDone % 4 != 0) {
            // If the pomos completed is less than 4 (1-3)
            phase = 'short break';
            // document.getElementById('cycleNum').innerText = (pomosDone % 4) + ' / 4';

            if (theme == 'Potato') {
                circle.className = 'circlePotatoBreak';
```

```javascript
            }
        } else {
            // If the pomos completed is divisible by 4
            phase = 'long break';
            if (theme == 'Potato') {
                if (lang == 'ko' || lang == 'zh') {
                    circle.className = 'circlePotatoBreakAsian';
                } else {
                    circle.className = 'circlePotatoBreak';
                }
            }
        }
    } else {
        if (phase == 'long break') {
            if (theme == 'Potato') {
                if (lang == 'ko' || lang == 'zh') {
                    circle.className = 'circlePotatoBreakAsian';
                } else {
                    circle.className = 'circlePotatoBreak';
                }
                hidePotatoes();
            }
        }
        phase = 'work';
        if (theme == 'Potato') {
            circle.className = 'circlePotato';
        }
    }
}

/**
 * Checks what the current timer state is from
 * 'work', 'short break', or 'long break'
 * to know what the timer should start counting down with.
 *
 * @return {number} The time remaining for the current timer state.
 */
function setTimeRemaining() {
    return (phase == 'work') ? workLength :
        (phase == 'short break') ? shortBreakLength :
            longBreakLength;
}

/**
 *  Appends a pomotato to the congrats screen
 *  for each pomo done and displays the congrats screen.
 */
function displayCongrats() {
    document.getElementById('potatoImgs').innerHTML = '';

    // Output potato images to the congrats screen (limit amount of potatoes)
    // Max Potatoes: 16
    for (let i = 0; i < pomosDone; i++) {
        // Prevent Potatoes images from overcrowding screen
        if (i > MAX_POTATOES_COMPLETED) {
            break;
        }
        let potato = document.createElement('img');

        if (animation == 'true') {
            potato.src = 'img/potato-dance.gif';
            potato.alt = dict['potatoDance'][lang];
        } else {
            potato.src = 'img/pomotato.png';
            potato.alt = dict['pomotato'][lang];
        }
        document.getElementById('potatoImgs').appendChild(potato);
    }
    document.getElementById("congratsText").innerHTML = dict['congrats1'][lang] + pomosDone + dict['congrats2'][lang];
    playAudio('victoryAudio');
    show('congratsScreen');
}

/**
 * Hides all of the dancing potato gifs.
 */
function hidePotatoes() {
    document.getElementById('cycle0').style.display = 'none';
    document.getElementById('cycle1').style.display = 'none';
    document.getElementById('cycle2').style.display = 'none';
    document.getElementById('cycle3').style.display = 'none';
}

/**
 * Shows a number of dancing potatoes based on the pomosDone.
 */
function showPotatoes() {
    document.getElementById('cycle' + pomosDone % 4).style.display = 'inline';
}

/**
 * Sets the title element for users to see remaining time off-page.
 *
 * @param {string} MMSS 'MM:SS' form
 * @return {string} New page title
 */
function setPageTitle(MMSS) {
    let phaseSymbol;
    switch (phase) {
        case 'work':
            phaseSymbol = ' Work - ';
            break;
        case 'short break':
        case 'long break':
            phaseSymbol = ' Break - ';
            break;
        case 'stopped':
            phaseSymbol = ' Stopped - '
            break;
        default:
            phaseSymbol = ' - ';
    }

    return MMSS + phaseSymbol;
}



/**
 * Resets the pomodoro cycle to the beginning.
 */
function stop() {
    // console.log('stop the timer and reset everything');
    clearInterval(timer);
    let bg = document.getElementById('background');
    if (theme == 'Potato' && phase != 'work') {
        document.getElementById('circleTimer').className = 'circlePotato';
        bg.classList.replace('potatoBreak', 'potatoWork');
    }
    hidePotatoes();
    phase = 'idle';
    document.getElementById('timerDisplay').innerHTML = '- - : - -';
```

```
    document.getElementById('phaseDisplay').innerHTML = dict['phase'][phase][lang];

    document.getElementById('start').innerHTML = dict['start'][lang];
    document.getElementById('start').onclick = start;

    //tasksDone = 0;
    pomosDone = 0;
    localStorage.setItem('pomosDone', pomosDone);

    uniqueID = 1;
    hide('prompt');
    showOptions();
}

/**
 * Adds a non-blank task to the list of tasks.
 */
function addTask() {
    const task = document.getElementById('enterTask').value;
    document.getElementById('enterTask').value = '';
    if (task != '') {
        createTask(task);
        // console.log('Created task with ID ' + uniqueID);
        // console.log('Task count: ' + taskCount);
    }
}

/**
 * Creates a userTask in the taskListContainer.
 * This does not display a task on the main page.
 * A userTask is identified with a unique numerical ID.
 * Has four child elements: mark to mark as done, pin a copy to main page,
 * delete from task list and main page, if pinned, and task content.
 *
 * @event addTask()
 * @param {string} text The task the user entered.
 */
function createTask(text) {
    let taskList = document.getElementById('taskListContainer');
    let newTask = document.createElement('div');
    newTask.className = 'userTask';
    newTask.id = uniqueID;

    let markBtn = document.createElement('button');
    markBtn.className = 'transparent';
    markBtn.setAttribute('aria-label', dict['markBtn'][lang]);

    let pinBtn = document.createElement('button');
    pinBtn.classList.add('transparent'); //, 'smallIcon');
    pinBtn.setAttribute('aria-label', dict['pinBtn'][lang]);

    let delBtn = document.createElement('button');
    delBtn.classList.add('transparent', 'smallIcon');
    delBtn.setAttribute('aria-label', dict['delBtn'][lang]);

    if (theme == 'Dark') {
        markBtn.innerHTML = '<div class="markCircle markDark" id="mark-' + uniqueID + '"></div>';
        pinBtn.innerHTML = '<img src="img/unpinned-dark.png" id="pin-' + uniqueID + '">';
        delBtn.innerHTML = '<img src="img/delete-task-dark.png" class="delete" id="del-' + uniqueID + '">';
    } else {
        markBtn.innerHTML = '<div class="markCircle markLight" id="mark-' + uniqueID + '"></div>';
        pinBtn.innerHTML = '<img src="img/unpinned.png" id="pin-' + uniqueID + '">';
        delBtn.innerHTML = '<img src="img/delete-task.png" class="delete" id="del-' + uniqueID + '">';
    }

    markBtn.onclick = function () {
        markedTask = document.getElementById('mark-' + newTask.id);
        if (markedTask.classList.contains('markFill')) {
            unmark(newTask.id);
        } else {
            markDone(newTask.id);
        }
    };

    pinBtn.onclick = function () {
        origTask = document.getElementById('pin-' + newTask.id);
        pinnedTask = document.getElementById(newTask.id + '-copy');
        if (!pinnedTask) {
            createPinnedTask(text, newTask.id);
            if (theme == 'Dark') {
                origTask.src = 'img/pinned-dark.png';
                console.log('dark pin');
            } else {
                origTask.src = 'img/pinned.png';
            }
        } else {
            if (theme == 'Dark') {
                origTask.src = 'img/unpinned-dark.png';
            } else {
                origTask.src = 'img/unpinned.png';
            }
            unpinTask(newTask.id);
        }
    };

    delBtn.onclick = function () {
        deleteTask(newTask.id);
    }

    let content = document.createElement('p');
    content.id = 'p' + uniqueID;
    content.innerHTML = text;

    let ariaSkip = document.createElement('a');
    ariaSkip.href = '#' + (uniqueID + 1);
    ariaSkip.className = 'ariaSkipTask';
    ariaSkip.innerText = dict['skip'][lang];
    newTask.appendChild(ariaSkip);

    taskCount++;
    const taskBtn = document.getElementById('taskBtn');
    taskBtn.innerHTML = dict['tasks'][lang] + ' (' + tasksDone + '/' + taskCount + ')';
    taskBtn.style.width = "fit-content";

    savedTasks.push(text);
    // console.log(JSON.stringify(savedTasks));
    localStorage.setItem('savedTasks', JSON.stringify(savedTasks));
    // console.log(localStorage.getItem("savedTasks"));

    newTask.appendChild(markBtn);
    newTask.appendChild(pinBtn);
    newTask.appendChild(content);
    newTask.appendChild(delBtn);
    taskList.appendChild(newTask);

    if (taskCount == 1) {
        createPinnedTask(text, uniqueID);
        if (theme == 'Dark') {
            document.getElementById('pin-' + uniqueID).src = 'img/pinned-dark.png';
        } else {
            document.getElementById('pin-' + uniqueID).src = 'img/pinned.png';
        }
```

```
    }

    notifyUser('addTask');
    return uniqueID++;
}

/**
 * Creates 'pinned' userTask in the mainTasks container.
 * This display an existing task on the main page.
 * A pinned task is identified as '#pin' where # is the uniqueID.
 * Inherits the four userTask components,
 * The eventListener for pin is different.
 *
 * @param {string} text     A copy of the user's task.
 * @param {string} uniqueID The existing task's id.
 */
function createPinnedTask(text, uniqueID) {
    let mainTasks = document.getElementById('mainTasks');
    let pinTask = document.createElement('div');
    pinTask.classList.add('userTask', 'pinnedTask');
    pinTask.id = uniqueID + '-copy';

    let markBtn = document.createElement('button');
    markBtn.className = 'transparent';
    markBtn.setAttribute('aria-label', dict['markBtn'][lang]);

    let pinBtn = document.createElement('button');
    pinBtn.classList.add('transparent'); //, 'smallIcon');
    pinBtn.setAttribute('aria-label', dict['pinBtn'][lang]);

    // let delBtn = document.createElement('button');
    // delBtn.classList.add('transparent', 'smallIcon');
    // delBtn.setAttribute('aria-label', 'Delete this Task');

```

```javascript
    if (theme == 'Dark') {
        markBtn.innerHTML = '<div class="markCircle markDark" id="mark-' + uniqueID + '-copy"></div>';
        pinBtn.innerHTML = '<img src="img/pinned-dark.png">';
        //    delBtn.innerHTML = '<img src="img/delete-task-dark.png" class="delete" id="del-'+uniqueID+'-copy">';
    } else {
        markBtn.innerHTML = '<div class="markCircle markLight" id="mark-' + uniqueID + '-copy"></div>';
        pinBtn.innerHTML = '<img src="img/pinned.png">';
        //    delBtn.innerHTML = '<img src="img/delete-task.png" class="delete" id="del-'+uniqueID+'-copy">';
    }

    markBtn.onclick = function () {
        markedTask = document.getElementById('mark-' + uniqueID);
        if (markedTask.classList.contains('markFill')) {
            unmark(uniqueID);
        } else {
            markDone(uniqueID);
        }
    };

    pinBtn.onclick = function () {
        unpinTask(uniqueID);
    };

    // delBtn.onclick = function() {
    //     deleteTask(uniqueID);
    // }

    let content = document.createElement('p');
    content.innerHTML = text;

    let ariaSkip = document.createElement('a');
    ariaSkip.href = '#' + (uniqueID + 1) + '-copy';
    ariaSkip.className = 'ariaSkipTask';
```

```javascript
    ariaSkip.innerText = dict['skip'][lang];
    pinTask.appendChild(ariaSkip);

    pinTask.appendChild(markBtn);
    pinTask.appendChild(pinBtn);
    pinTask.appendChild(content);
    // pinTask.appendChild(delBtn);

    mainTasks.appendChild(pinTask);

    if (document.getElementById('mark-' + uniqueID).classList.contains('markFill')) {
        document.getElementById('mark-' + uniqueID + '-copy').classList.add('markFill');
    }
    setPinnedSkip();
    notifyUser('pinTask');
    return uniqueID;
}

/**
 * Visually marks a task if a user completes the task.
 * This affects the task list and main display, if possible.
 * Increments the number of tasks completed.
 * @param {string} uniqueID The existing task's (task list) id.
 */
function markDone(uniqueID) {
    let originalTask = document.getElementById(uniqueID);
    document.getElementById('mark-' + uniqueID).classList.add('markFill');
    let pinnedTask = document.getElementById(uniqueID + '-copy');

    if (pinnedTask) {
        document.getElementById('mark-' + uniqueID + '-copy').classList.add('markFill');
    }
    tasksDone++;
    originalTask.setAttribute('marked', 'true');
    const taskBtn = document.getElementById('taskBtn');
    taskBtn.innerHTML = dict['tasks'][lang] + ' (' + tasksDone + '/' + taskCount + ')';
    // console.log('Tasks done: ' + tasksDone);
    notifyUser('mark');
}

/**
 * Visually unmarks a task if a user did not complete the task.
 * This affects the task list and main display, if possible.
 * Decrements the number of tasks complete.
 * @param {string} uniqueID The existing task's (task list) id.
 */
function unmark(uniqueID) {
    let originalTask = document.getElementById(uniqueID);
    document.getElementById('mark-' + uniqueID).classList.remove('markFill');
    let pinnedTask = document.getElementById(uniqueID + '-copy');

    if (pinnedTask) {
        document.getElementById('mark-' + uniqueID + '-copy').classList.remove('markFill');
    }
    tasksDone--;
    originalTask.setAttribute('marked', 'false');
    const taskBtn = document.getElementById('taskBtn');
    taskBtn.innerHTML = dict['tasks'][lang] + ' (' + tasksDone + '/' + taskCount + ')';
    // console.log('Tasks done: ' + tasksDone);
    notifyUser('unmark');
}

/**
 * Unpins a task from the main display by deleting the pinned copy.
 * @param {string} uniqueID The existing task's (task list) id.
 *
 * @example Unpin pinned task '1pin' calls function with '1pin'
 */
function unpinTask(uniqueID) {
    let pinnedTask = document.getElementById(uniqueID + '-copy');
    const mainTasks = document.getElementById('mainTasks');
    mainTasks.removeChild(pinnedTask);
    if (theme == 'Dark') {
        document.getElementById('pin-' + uniqueID).src = 'img/unpinned-dark.png';
    } else {
        document.getElementById('pin-' + uniqueID).src = 'img/unpinned.png';
    }
    notifyUser('unpinTask');
    setPinnedSkip();
}

/**
 * Deletes a task from both the task list and the main display, if possible.
 * Decreases the number of tasks by one.
 * @param {string} uniqueID The existing task's (task list) id.
 *
 * @example Delete pinned task '1pin' calls function with '1'.
 */
function deleteTask(uniqueID) {
    let taskText = document.getElementById('p' + uniqueID).innerText;

    const pinnedTask = document.getElementById(uniqueID + '-copy');

    if (document.getElementById(uniqueID).getAttribute('marked') == 'true') {
        tasksDone--;
    }

    if (pinnedTask) {
        const mainTasks = document.getElementById('mainTasks');
        mainTasks.removeChild(pinnedTask);
        // console.log('Deleted a pinned task.');
    }
    const taskListContainer = document.getElementById('taskListContainer');
    taskListContainer.removeChild(document.getElementById(uniqueID));
    taskCount--;

    const taskBtn = document.getElementById('taskBtn');
    taskBtn.innerHTML = dict['tasks'][lang] + ' (' + tasksDone + '/' + taskCount + ')';

    if (taskCount == 0) {
        taskBtn.innerHTML = dict['tasks'][lang];
    }

    // ARIA
    setARIASkip();

    savedTasks.splice(savedTasks.indexOf(taskText), 1);
    localStorage.setItem('savedTasks', JSON.stringify(savedTasks));

    notifyUser('delTask');
    // console.log('Task count: ' + taskCount);
}

/**
 * Deletes all of the tasks from both the tsak list and main display, if possible.
 * Resets taskCount to 0 and uniqueID to 1.
 * @event deleteAll text
 */
function deleteAllTasks() {
    const taskListContainer = document.getElementById('taskListContainer');
    while (taskListContainer.firstChild) {
        taskListContainer.removeChild(taskListContainer.lastChild);
```

```javascript
    }

    const mainTasks = document.getElementById('mainTasks');
    while (mainTasks.firstChild) {
        mainTasks.removeChild(mainTasks.lastChild);
    }

    taskCount = 0;
    tasksDone = 0;
    uniqueID = 1;
    const taskBtn = document.getElementById('taskBtn');

    if (taskCount == 0) {
        taskBtn.innerHTML = dict['tasks'][lang];
    }
    savedTasks = [];
    localStorage.removeItem('savedTasks');
    hide('prompt');
    notifyUser('deleteAll');
    // console.log('Deleted all tasks.');
    // console.log('Task Count: ' + taskCount);
}

/**
 * @event deleteTask()
 * Updates all userTasks in the task list so their ARIA skip links will link
 * To the next task based on the next task's ID.
 */
function setARIASkip() {
    let taskListContainer = document.getElementById('taskListContainer');
    let userTaskList = taskListContainer.children; // <a list of userTask nodes
    for (let i = 0; i < userTaskList.length - 1; i++) {
        let userTask = userTaskList[i]; // The first userTask
        let nextTask = userTaskList[i + 1];
        userTask.firstChild.href = '#' + nextTask.id;
    }
}

/**
 * @event unpinTask()
 * Updates all userTasks in the main task list so their ARIA skip links will link
 * To the next task based on the next task's ID.
 */
function setPinnedSkip() {
    let mainTasks = document.getElementById('mainTasks');
    let pinnedTaskList = mainTasks.children;
    for (let i = 0; i < pinnedTaskList.length - 1; i++) {
        let pinnedTask = pinnedTaskList[i];
        let nextTask = pinnedTaskList[i + 1];
        pinnedTask.firstChild.href = '#' + nextTask.id;
    }
}

/**
 * Confirms a user's action to prevent major accidents.
 * @param {string} action The action to confirm. Either 'Reset' or 'Delete' all.
 */
function confirmationPrompt(action) {
    // console.log('prompt');
    show('prompt');
    let message = document.getElementById('confirmMessage');
    let confirmBtn = document.getElementById('confirm');

    if (action == 'Stop') {
        message.innerHTML = dict['confirmReset'][lang];
        confirmBtn.onclick = stop;
```

```javascript
    } else if (action == 'Delete') {
        message.innerHTML = dict['confirmDeleteAll'][lang];
        confirmBtn.onclick = deleteAllTasks;
    }
}

/**
 * Shows an element by changing its display to block.
 * @param {string} id The id of the element to show.
 */
function show(id) {
    const elem = document.getElementById(id);
    // console.log('showing');
    elem.classList.replace('hidden', 'showing');
}

/**
 * Hides an element by changing its display to none.
 * also saves settings if the element to be hidden is the settings menu
 * @param {string} id The id of the element to hide.
 */
function hide(id) {
    const elem = document.getElementById(id);
    if(id == 'settingsMenu') {
        saveSettings();
    }
    // console.log('hiding');
    elem.classList.replace('showing', 'hidden');
}

/**
 * @event stop()
 * Shows the various options and buttons available to the user.
 * Triggers when a user presses the stop button or reaches the congrats screen.
 */
function showOptions() {
    document.getElementById('help').classList.replace('opacityHide', 'opacityShow');
    document.getElementById('settingsIcon').classList.replace('opacityHide', 'opacityShow');
    document.getElementById('enterTask').classList.replace('opacityHide', 'opacityShow');
    document.getElementById('taskAdder').classList.replace('opacityHide', 'opacityShow');
    document.getElementById('taskBtn').classList.replace('opacityHide', 'opacityShow');

    // let options = document.getElementsByClassName('opacityHide');
    // for(let i = 0; i < options.length; i++) {
    //     console.log(options);
    //     options[i].classList.remove('opacityHide');
    //     options[i].classList.add('opacityShow');
    // }
}

/**
 * @event start()
 * Hides the various options and buttons available to the user.
 * Triggers when a user clicks the start button.
 */
function hideOptions() {
    document.getElementById('help').classList.replace('opacityShow', 'opacityHide');
    document.getElementById('settingsIcon').classList.replace('opacityShow', 'opacityHide');
    document.getElementById('enterTask').classList.replace('opacityShow', 'opacityHide');
    document.getElementById('taskAdder').classList.replace('opacityShow', 'opacityHide');
    document.getElementById('taskBtn').classList.replace('opacityShow', 'opacityHide');
    // let options = document.getElementsByClassName('opacityShow');
    // for(let i = 0; i < options.length; i++) {
    //     console.log(options);
    //     options[i].classList.remove('opacityShow');
    //     options[i].classList.add('opacityHide');
    // }
}


var page = 0;
/**
 * Goes to the previous page of the instructions menu
 */
function back() {
    if (page <= 1) {
        return;
    }
    --page;
    let topic = document.getElementById('instrTopic');
    topic.innerText = dict[page][topic.id][lang];
    let content = document.getElementById('instrContent');
    content.innerText = dict[page][content.id][lang];
    if (page != 4) {
        content.classList.add('leftAlign');
    }
    document.getElementById('page').innerText = page + ' / 4';
    document.getElementById('next').innerHTML = dict['next'][lang];

}

/**
 * Goes to the next page of the instructions menu
 */
function next() {
    document.getElementById('next').innerHTML = dict['next'][lang];
    if (page >= 4) {
        hide('instructionsMenu');
        page = 0;
        // document.getElementById('next').innerHTML = ;
        return;
    }
    page++;
    let topic = document.getElementById('instrTopic');
    topic.innerText = dict[page][topic.id][lang];
    let content = document.getElementById('instrContent');
    content.innerText = dict[page][content.id][lang];
    document.getElementById('page').innerText = page + ' / 4';
    if (page == 4) {
        content.classList.remove('leftAlign');
        content.innerText = content.innerText +
            'Alexis Chen\nElizabeth Cho\nKevin Jang\nMarco Kuan\nAhmad Milad\nRohan Patel\nMiaoqiu Sun\nJessie Zou\n';
        if (theme == 'Dark') {
            content.innerHTML = content.innerHTML + '<a href="https://github.com/AlexisChen99/cse110-w21-group4"><img src="img/GitHub-Mark-Light-32px.png alt="GitHub"></a>';
        } else {
            content.innerHTML = content.innerHTML + '<a href="https://github.com/AlexisChen99/cse110-w21-group4"><img src="img/GitHub-Mark-32px.png" alt="GitHub"></a>';
        }
        document.getElementById('next').innerHTML = dict['close'][lang];
    }
}
/**
 * Creates a notification for the user based on what action the user just did
 * @param {string} action the action the user did
 * @returns  the action the user did
 */
function notifyUser(action) {
    let notif = document.getElementById('notificationBar');
    notif.innerText = dict['notification'][action][lang];
    setTimeout(function () {
        notif.innerText = '';
    }, 3000);
    return action;
}


/**
 * Loads a user's last theme and settings selected locally.
 */
function loadTheme() {
    switch (window.localStorage.getItem('theme')) {
        case 'Potato':
            changeTheme('Potato');
            break;
        case 'Dark':
            changeTheme('Dark');
            break;
        case 'Light':
            changeTheme('Light');
            break;
        default:
            //console.log('no previous theme');
            changeTheme('Potato');
    }
}

/**
 * Changes the theme and stores the new theme locally.
 * @event loadTheme()
 * @event button
 * @param {string} newTheme The theme to change to.
 */
function changeTheme(newTheme) {
    window.localStorage.setItem('theme', newTheme);
    theme = newTheme;
    const body = document.getElementById('background');
    body.className = 'theme' + newTheme;

    if (newTheme == 'Potato') {
        body.classList.add('potatoWork');
    } else {
        hidePotatoes();
    }

    const circle = document.getElementById('circleTimer');
    circle.className = 'circle' + newTheme;

    if (newTheme == 'Dark') {
        let settingsIcon = document.getElementById('settingsIcon');
        settingsIcon.classList.replace('settingsLight', 'settingsDark');

        let volumeIcon = document.getElementById('volumeIcon');
        if (volume != 0) {
            volumeIcon.src = 'img/volume-dark.png';
        } else {
            volumeIcon.src = 'img/volume-mute-dark.png';
        }

        let buttons = document.getElementsByClassName('mainButton');
        for (let i = 0; i < buttons.length; i++) {
            buttons[i].classList.add('darkButton');
        }

        let transparentBtns = document.getElementsByClassName('transparent');
        for (let i = 0; i < transparentBtns.length; i++) {
            transparentBtns[i].classList.remove('textDark');
        }

        let menus = document.getElementsByClassName('menu');
        for (let i = 0; i < menus.length; i++) {
            menus[i].classList.add('themeDark');
            if (menus[i].classList.contains('menuLight')) {
                menus[i].classList.remove('menuLight');
            }
        }

        let prompt = document.getElementById('prompt');
        prompt.classList.add('themeDark');
        if (prompt.classList.contains('themeLight')) {
            prompt.classList.remove('themeLight');
        }

        let congrats = document.getElementById('congratsContent');
        congrats.classList.add('modalDark');
        congrats.classList.remove('modalLight', 'modalPotato');

        let userTasks = document.getElementsByClassName('userTask');
        for (let i = 0; i < userTasks.length; i++) {
            userTasks[i].children[1].firstChild.classList.replace('markLight', 'markDark');
            let pinSrc = userTasks[i].children[2].firstChild.src;
            if (pinSrc.includes('img/pinned.png')) {
                userTasks[i].children[2].firstChild.src = 'img/pinned-dark.png';
            } else {
                userTasks[i].children[2].firstChild.src = 'img/unpinned-dark.png';
            }
            if (userTasks[i].children[4]) {
                userTasks[i].children[4].firstChild.src = 'img/delete-task-dark.png';
            }
        }
    } else if (newTheme == 'Potato' || newTheme == 'Light') {
        let settingsIcon = document.getElementById('settingsIcon');
        settingsIcon.classList.replace('settingsDark', 'settingsLight');

        let volumeIcon = document.getElementById('volumeIcon');
        if (volume != 0) {
            volumeIcon.src = 'img/volume.png';
        } else {
            volumeIcon.src = 'img/volume-mute.png';
        }

        let buttons = document.getElementsByClassName('mainButton');
        for (let i = 0; i < buttons.length; i++) {
            if (buttons[i].classList.contains('darkButton')) {
                buttons[i].classList.remove('darkButton');
            }
        }

        let transparentBtns = document.getElementsByClassName('transparent');
        for (let i = 0; i < transparentBtns.length; i++) {
            transparentBtns[i].classList.add('textDark');
        }

        let menus = document.getElementsByClassName('menu');
        for (let i = 0; i < menus.length; i++) {
            menus[i].classList.add('menuLight');
            if (menus[i].classList.contains('themeDark')) {
                menus[i].classList.remove('themeDark');
            }
        }

        let prompt = document.getElementById('prompt');
        prompt.classList.add('themeLight');
        if (prompt.classList.contains('themeDark')) {
            prompt.classList.remove('themeDark');
        }

        let congrats = document.getElementById('congratsContent');
        congrats.classList.remove('modalDark');
        if(theme == 'Light') {
            congrats.classList.add('modalLight');
        } else {
            congrats.classList.add('modalPotato');
        }

        let userTasks = document.getElementsByClassName('userTask');
        for (let i = 0; i < userTasks.length; i++) {
            // console.log('changing tasks');
            userTasks[i].children[1].firstChild.classList.replace('markDark', 'markLight');
            let pinSrc = userTasks[i].children[2].firstChild.src;
            if (pinSrc.includes('img/pinned-dark.png')) {
                userTasks[i].children[2].firstChild.src = 'img/pinned.png';
            } else {
                userTasks[i].children[2].firstChild.src = 'img/unpinned.png';
            }
            if (userTasks[i].children[4]) {
                userTasks[i].children[4].firstChild.src = 'img/delete-task.png';
            }
        }
    }

    if (theme == 'Potato') {
        show('animationBtn');
    } else {
        hide('animationBtn');
        hidePotatoes();
    }
    //hide('settingsMenu');
}

```

```javascript
/**
 * @event button
 * Changes the chosen language of Potato Timer
 * @param {string} selectedLang the language the user wishes to see potatotimer in
 */
function setLang(selectedLang) {
    window.localStorage.setItem('lang', selectedLang);
    window.location.reload();
}

/**
 * Changes all of the elements of the DOM into the proper language.
 * Stores the new language in local storage.
 * The default language is English.
 */
function loadLang() {
    let savedLang = window.localStorage.getItem('lang');
    if (savedLang == null) {
        // console.log("No saved language detected. Your browser's language is: " + navigator.language);
        if (navigator.language.includes('es')) {
            lang = 'es';
        } else if (navigator.language.includes('zh')) {
            lang = 'zh';
        } else if (navigator.language == 'ko') {
            lang = 'ko';
        } else {
            lang = 'en';
        }
        window.localStorage.setItem('lang', lang);
    } else {
        lang = savedLang;
    }

    document.documentElement.lang = lang; // <HTML> tag
    document.title = dict['title'][lang];
    document.getElementById('title').innerText = dict['title'][lang];
    document.getElementById('help').setAttribute('aria-label', dict['help'][lang]);
    document.getElementById('settingsIcon').setAttribute('aria-label', dict['openSettings'][lang]);
    document.getElementById('phaseDisplay').innerText = dict['phase']['idle'][lang];
    document.getElementById('start').innerText = dict['start'][lang];
    document.getElementById('taskBtn').innerText = dict['tasks'][lang];
    document.getElementById('enterTask').placeholder = dict['enterTask'][lang];
    document.getElementById('taskAdder').innerText = dict['add'][lang];

    document.getElementById('settingsTitle').innerText = dict['settings'][lang];
    document.getElementById('closeSettings').innerText = dict['close'][lang];
    document.getElementById('selectTheme').innerText = dict['selectTheme'][lang];
    document.getElementById('lightTheme').innerText = dict['lightTheme'][lang];
    document.getElementById('darkTheme').innerText = dict['darkTheme'][lang];
    document.getElementById('potatoTheme').innerText = dict['potatoTheme'][lang];
    document.getElementById('workTime').innerText = dict['workTime'][lang];
    document.getElementById('shortTime').innerText = dict['shortBreak'][lang];
    document.getElementById('longTime').innerText = dict['longBreak'][lang];
    // document.getElementById('cycleLength').innerText = dict['cycleLength'][lang];
    document.getElementById('volumeTitle').innerText = dict['volume'][lang];

    document.getElementById('tasksTitle').innerText = dict['tasks'][lang];
    document.getElementById('taskHelp').innerText = dict['taskHelp'][lang];
    document.getElementById('closeTasks').innerText = dict['close'][lang];
    let close = document.getElementsByClassName('ariaClose');
    for (let i = 0; i < close.length; i++) {
        close[i].innerText = dict['close'][lang];
    }
    document.getElementById('deleteAll').innerText = dict['deleteAll'][lang];

    document.getElementById('confirm').innerText = dict['confirm'][lang];
    document.getElementById('cancel').innerText = dict['cancel'][lang];
    document.getElementById('congratsTitle').innerText = dict['congratsTitle'][lang];
    document.getElementById('instrTitle').innerText = dict['instructions'][lang];
    document.getElementById('back').innerText = dict['back'][lang];
    document.getElementById('next').innerText = dict['next'][lang];
    if (animation) {
        document.getElementById('animationBtn').innerText = dict['disableAnimation'][lang];
    } else {
        document.getElementById('animationBtn').innerText = dict['enableAnimation'][lang];
    }
    if (lang == 'es') {
        document.getElementById('settingsTitle').style.fontSize = "32px";
        document.getElementById('closeSettings').style.fontSize = "17px";
        var elements = document.getElementsByClassName('fieldLabel');
        for (var i = 0; i < elements.length; i++) {
            var element = elements[i];
            element.style.fontSize = "16.5px";
        }
    }
    document.getElementById('cycle0').innerText = dict['potatoDance'][lang];
    document.getElementById('cycle1').innerText = dict['potatoDance'][lang];
    document.getElementById('cycle2').innerText = dict['potatoDance'][lang];
    document.getElementById('cycle3').innerText = dict['potatoDance'][lang];
    document.getElementById('notificationBar').innerText = dict['notification']['welcome'][lang];

}

/**
 * Loads the tasks from local storage and creates them again.
 */
function loadTasks() {
    let savedTasks = JSON.parse(localStorage.getItem('savedTasks'));
    if (!savedTasks) {
        return;
    }

    for (let i = 0; i < savedTasks.length; i++) {
        createTask(savedTasks[i]);
    }

    //Pomo Data
    if (localStorage.getItem('pomosDone') != null) {
        pomosDone = localStorage.getItem('pomosDone');
    } else {
        console.log('no previous pomos');
    }
}

/**
 * Loads all of the user-custom settings in the settings menu.
 */
function loadSettings() {
    document.getElementById('workMin').value = (+localStorage.getItem('workMin'));
    document.getElementById('workSec').value = (+localStorage.getItem('workSec'));
    document.getElementById('shortMin').value = (+localStorage.getItem('shortMin'));
    document.getElementById('shortSec').value = (+localStorage.getItem('shortSec'));
    document.getElementById('longMin').value = (+localStorage.getItem('longMin'));
    document.getElementById('longSec').value = (+localStorage.getItem('longSec'));
    document.getElementById('volume').value = (+localStorage.getItem('volume'));
    mute = window.localStorage.getItem('mute');
    changeMuteIcon();
    animation = window.localStorage.getItem('animation');
    if (animation == 'true') {
        document.getElementById('animationBtn').innerText = dict['disableAnimation'][lang];
    } else {
        document.getElementById('animationBtn').innerText = dict['enableAnimation'][lang];
    }
}

/**
 * @event closeSettings The close button(s) on settings is pressed.
 * @event start()
 * Stores all of the current settings into localStorage.
 */
function saveSettings() {
    //timer phase settings
    localStorage.setItem('workMin', document.getElementById('workMin').value);
    localStorage.setItem('workSec', document.getElementById('workSec').value);
    localStorage.setItem('shortMin', document.getElementById('shortMin').value);
    localStorage.setItem('shortSec', document.getElementById('shortSec').value);
    localStorage.setItem('longMin', document.getElementById('longMin').value);
    localStorage.setItem('longSec', document.getElementById('longSec').value);
    //volume settings
    localStorage.setItem('volume', document.getElementById('volume').value);
    localStorage.setItem('mute', mute);
    //animation settings
    localStorage.setItem('animation', animation);
}

/**
 * (For Testing)
 * Manually sets the phase.
 * @param {string} newPhase The phase to change to.
 */
function setPhase(newPhase) {
    phase = newPhase;
}

module.exports = {
     setPhase,
     convertSeconds,
     setTimeRemaining,
     setPageTitle
  }
```