**New Skill I learned**

For my program, I had to implement a timer for the test, to countdown and calculate the score of the user. This feature wasn’t taught in the advanced higher course so I had to research ways to do it myself.

**Timer**

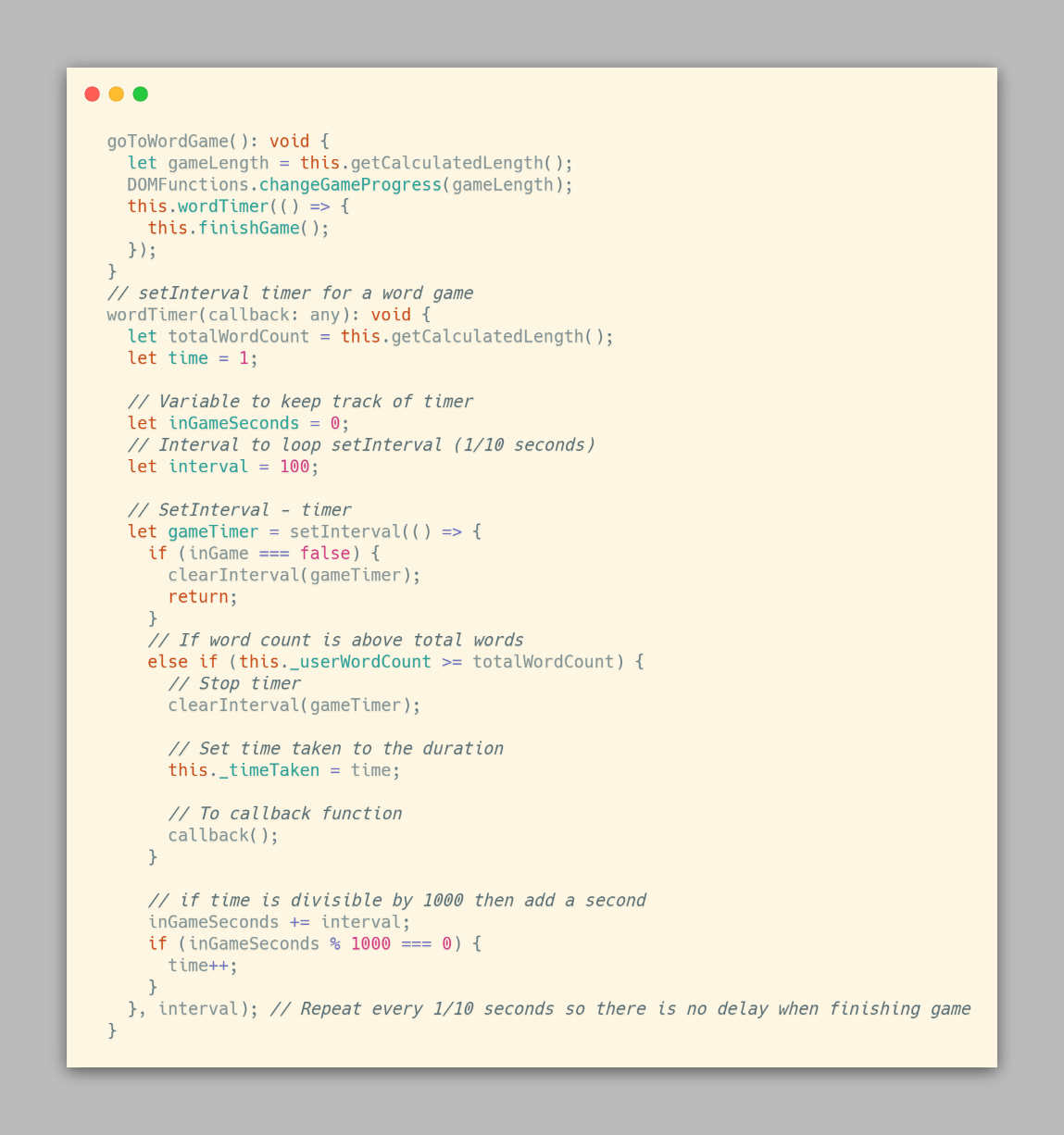
I found that the best way to do this is using a setInterval function, with a callback function when it’s finished. setInterval is a function that is executed repeatedly with a fixed delay between each call. When a “callback()” is called, the timer will stop and return to where it was called from. This effectively makes it so a setInterval with a delay of 1000 milliseconds will act as a per second timer. I used this to create a timer with 1000 milliseconds, which will update the time value each time it is called. When the test is finished, it sends a callback to the function, which will stop the test and move on to calculate scores and display the finish screen.

References

<https://developer.mozilla.org/en-US/docs/Web/API/WindowOrWorkerGlobalScope/setInterval>

<https://stackoverflow.com/questions/30427882/make-a-timer-using-setinterval>

Evidence

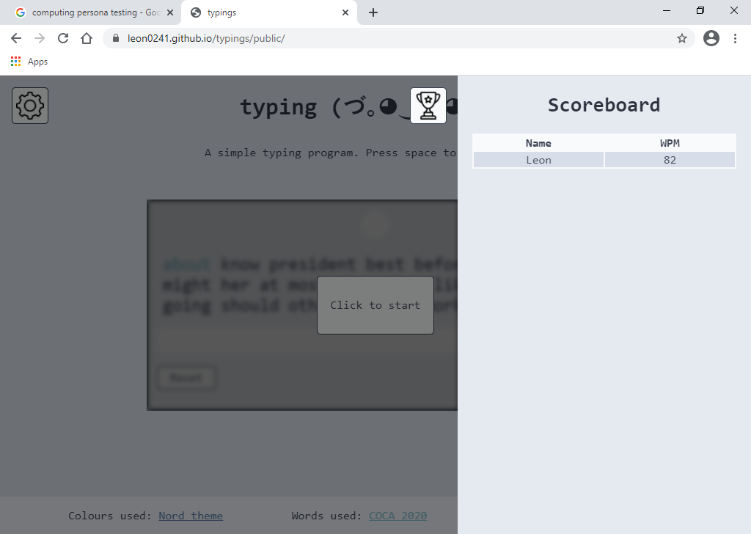
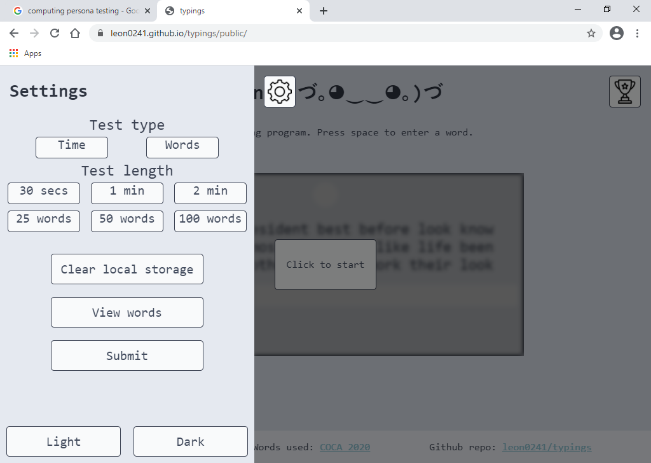


**Ongoing Testing**

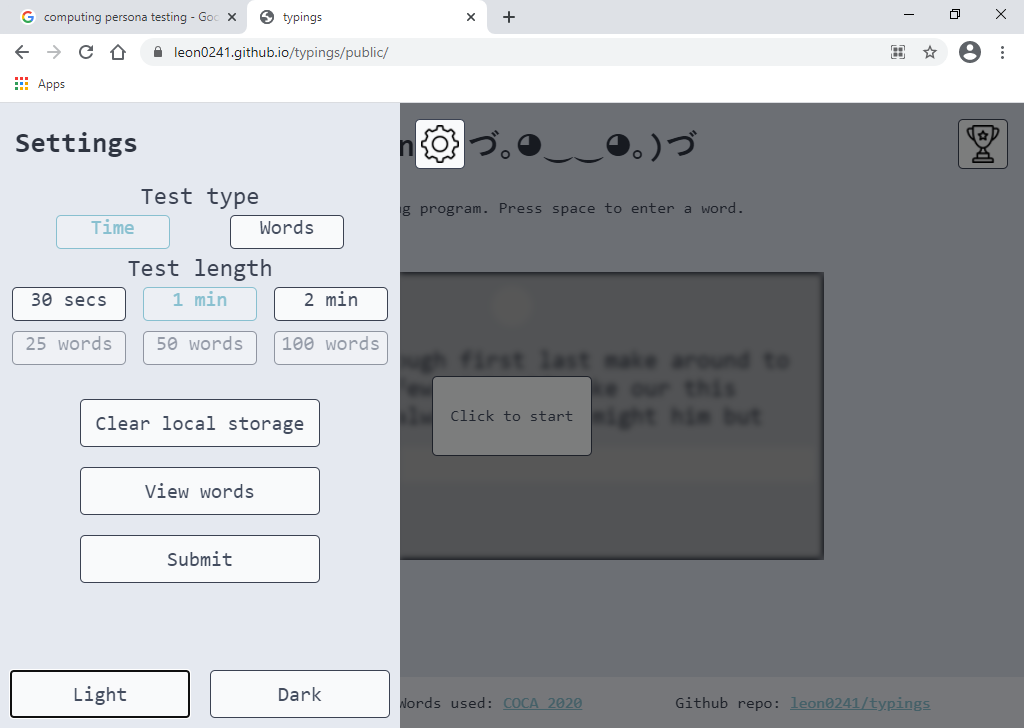
|  |  |  |  |
| --- | --- | --- | --- |
| Test | Outcome | Solution | Reference |
| Previous word highlighting | Not working | Only highlight second/third word so it doesn't go to a value of -1 |  |
| Line deletion when you get to end of line | Deleting every word instead of a row | Use offsetTop(container) instead of getBoundingClientRect(window) | <https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement/offsetTop> |
| Displaying words typed in the countdown | changing every second instead of when a word is typed | Put the counter in wordCount instead of the setInterval loop | MDN Web docs |
| Buttons changing settings | Buttons don't do anything | convert the variables to int, instead of a string "0" for example |  |
| Submit button works properly | Submit button will reload the page | Add return false to the end, so it doesn't refresh | Stack overflow |
| LocalStorage will save a score | LocalStorage can't store the user score | Data needs to be parsed into string format, and deparsed later | https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage |
| LocalStorage will save scores properly | LocalStorage breaks with 0 scores | Add a condition to check the length of localStorage is above 0 |  |
| Socreboard will update with new scores | Scoreboard looks for an undefined value | Set the key to search to index - 1 as the length is 1 based, while the array is 0 based |  |
| Bubble sort sorts properly | Program freezing | Bool checking if data is still being swapped had to be strongly compared with === |  |
| View word button works correctly | returns TypeError: "x" is not a function | Change onclick function to be different from the ID | https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Errors/Not\_a\_function |
| Submitting your name works properly | If name is empty, it is stored empty | Add a dummy name "Anon" for anonymous users | https://developer.mozilla.org/en-US/docs/Web/API/Node/textContent |
| LocalStorage will save settings | LocalStorage doesn't work with settings | Instead of the scoreboard working from length of localStorage, make it work from its own counter |  |

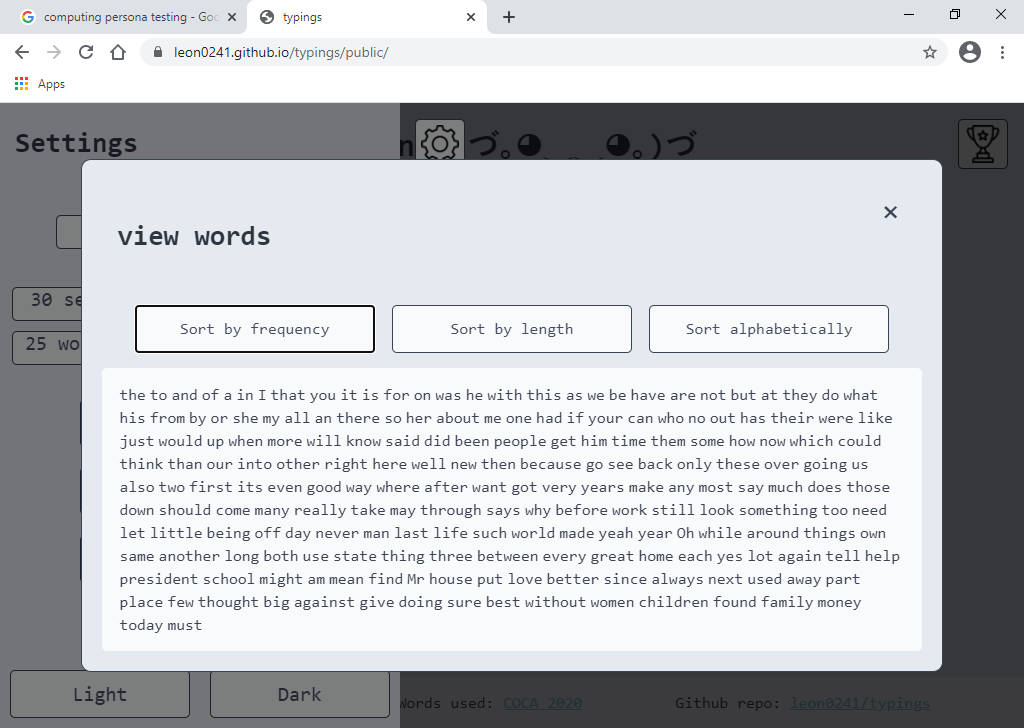
Component testing

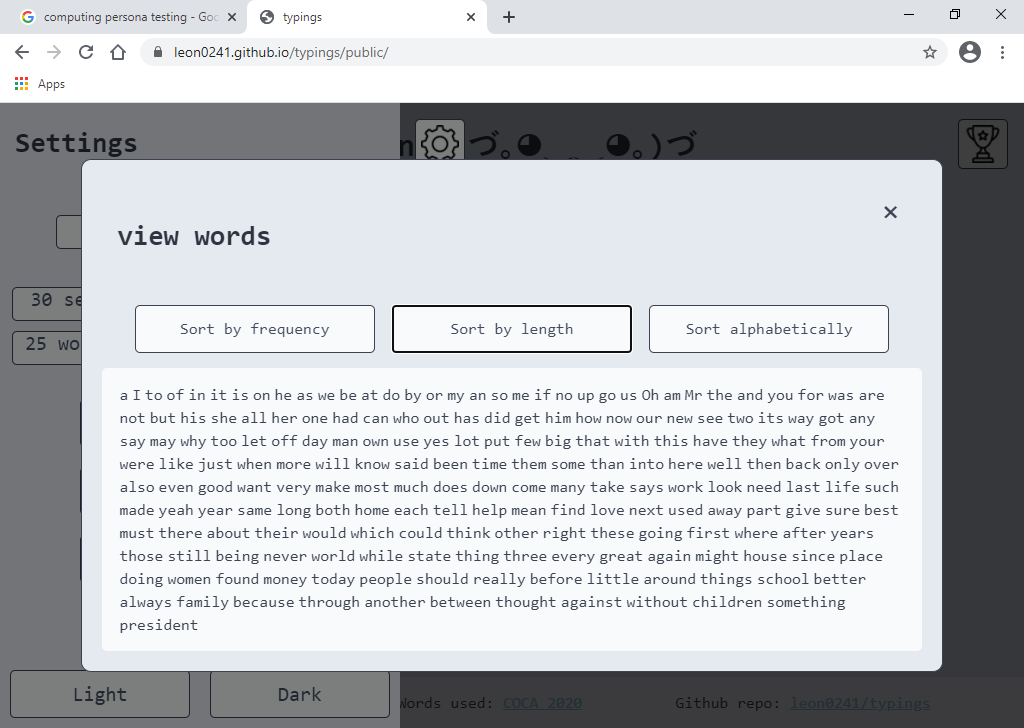
|  |  |  |  |
| --- | --- | --- | --- |
| Test | Component | Description | Successful? |
| C1 | Sidebar buttons | Check that the buttons open the sidebar properly. This should disable the rest of the screen from being able to be clicked | Yes |
| C2 | Settings input validation | The selection boxes work as intended. The test/time should gray out depending on which is selected | Yes |
| C3 | View words validation | The alphabetical, length and frequency sort work as intended, and sort the words in the correct order | Yes |
| C4 | Scoreboard validation | The scoreboard displays properly, and sorts user scores properly | Yes |
| C5 | Settings validation | Form will submit when submit button is pressed | Yes |
| C6 | Test run validation | 1. Typing field works properly and will delete word after spacebar is typed 2. Game countdown works properly, will start counting down when you start typing and will count down accordingly 3. Words will highlight red or green depending on if the word is correct | Yes |
| C7 | Test finish validation | WPM and Accuracy will display correctly, Name input box will save input on reload, retry/exit buttons work properly | Yes |

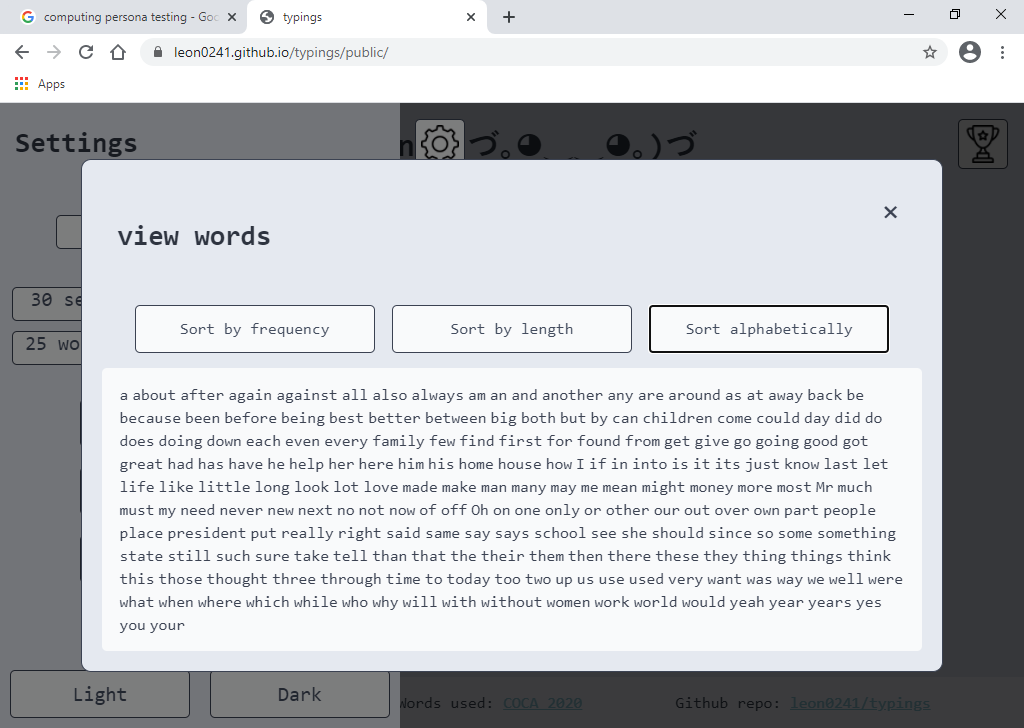
Component test – C1

Component test – C2

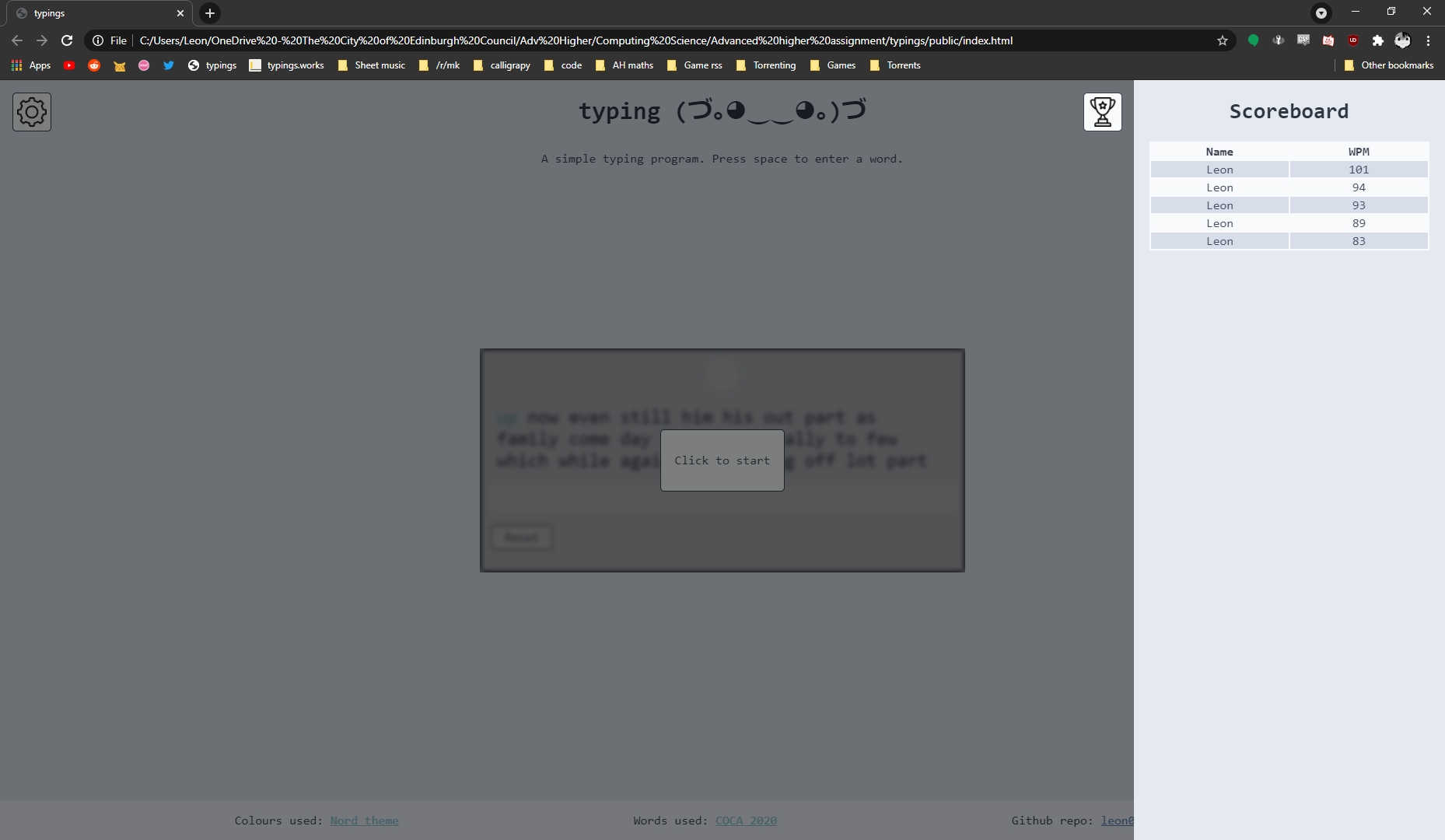


Component Test – C3

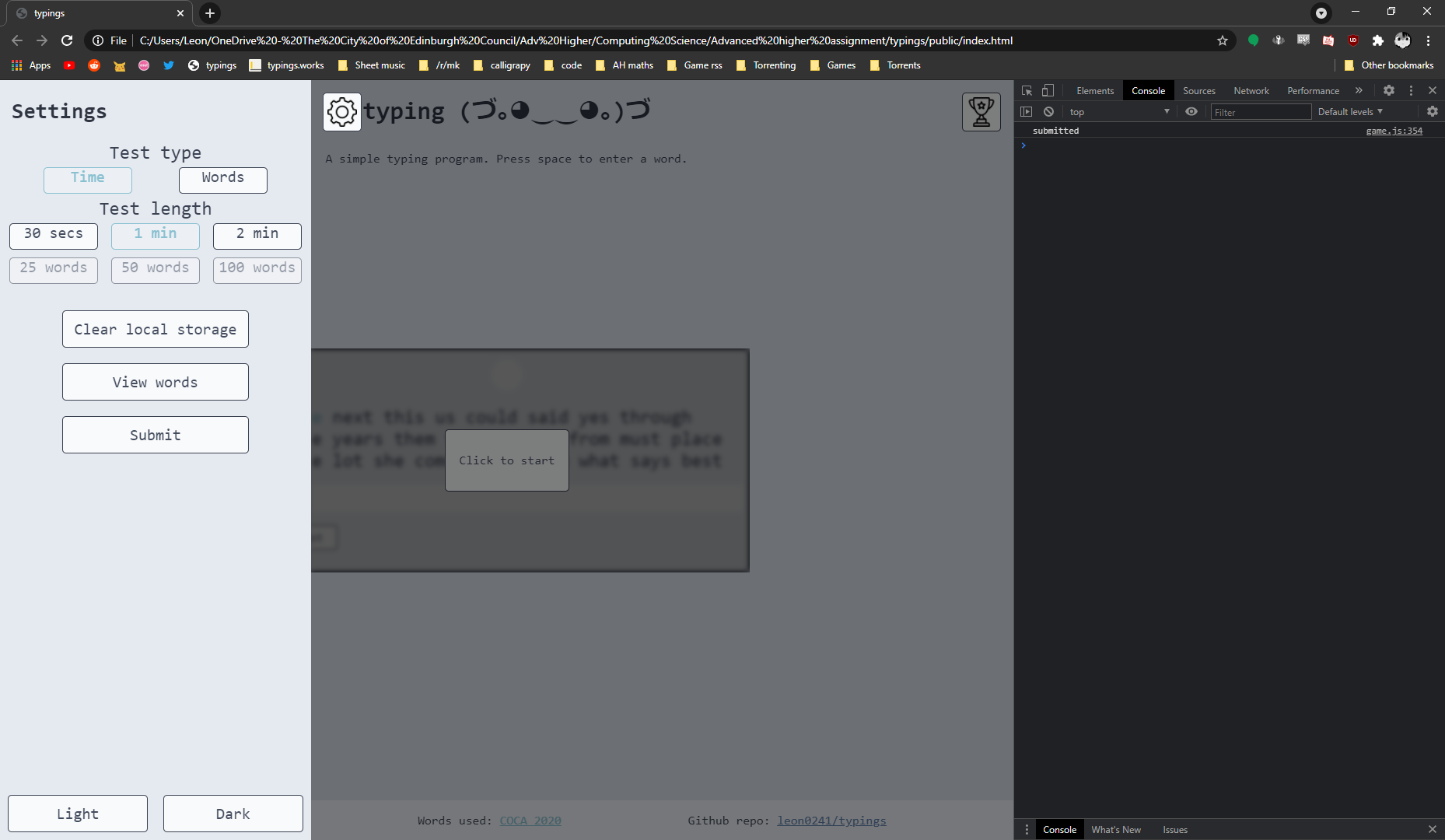


Component Test – C3

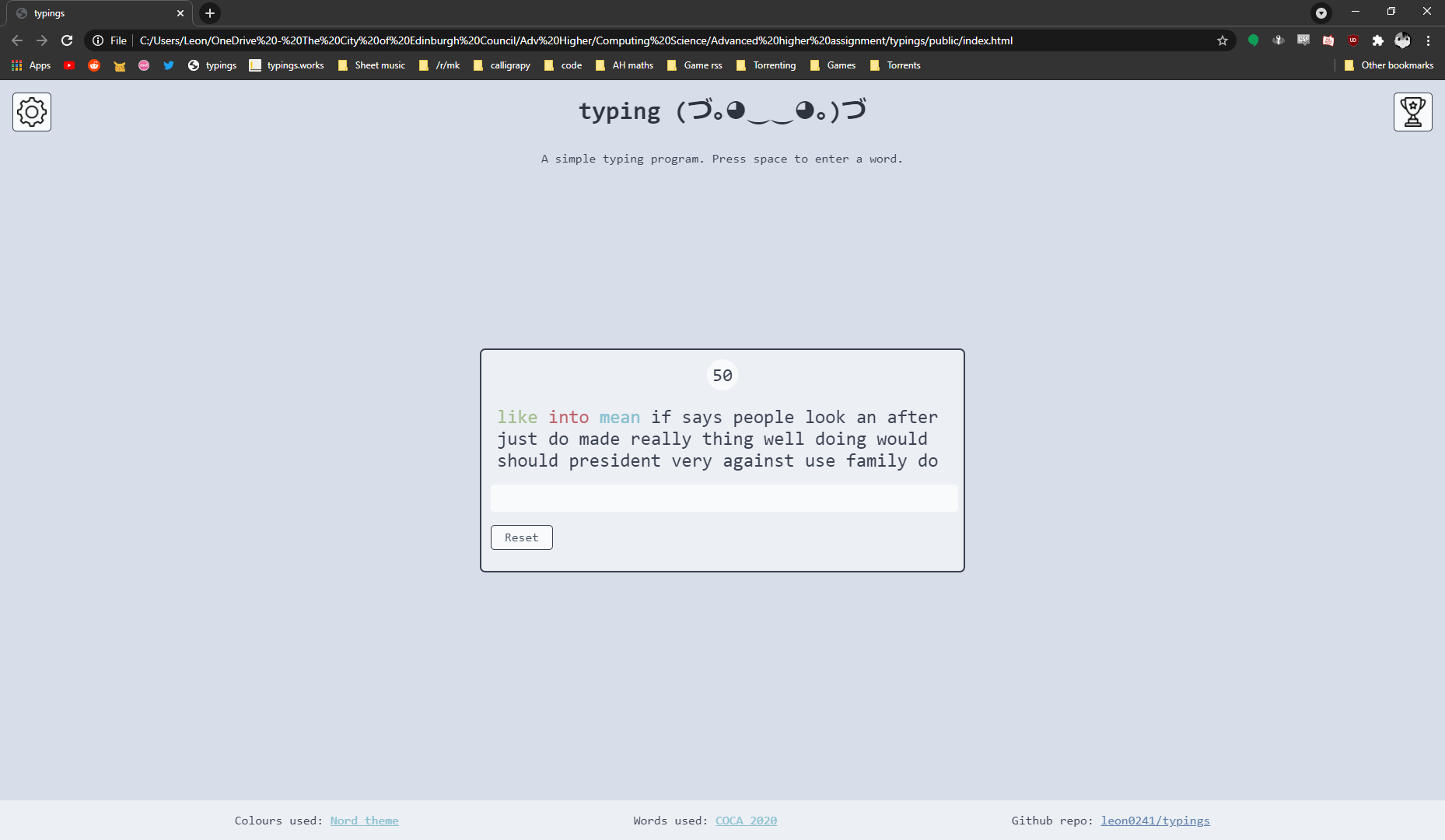
Component Test – C4



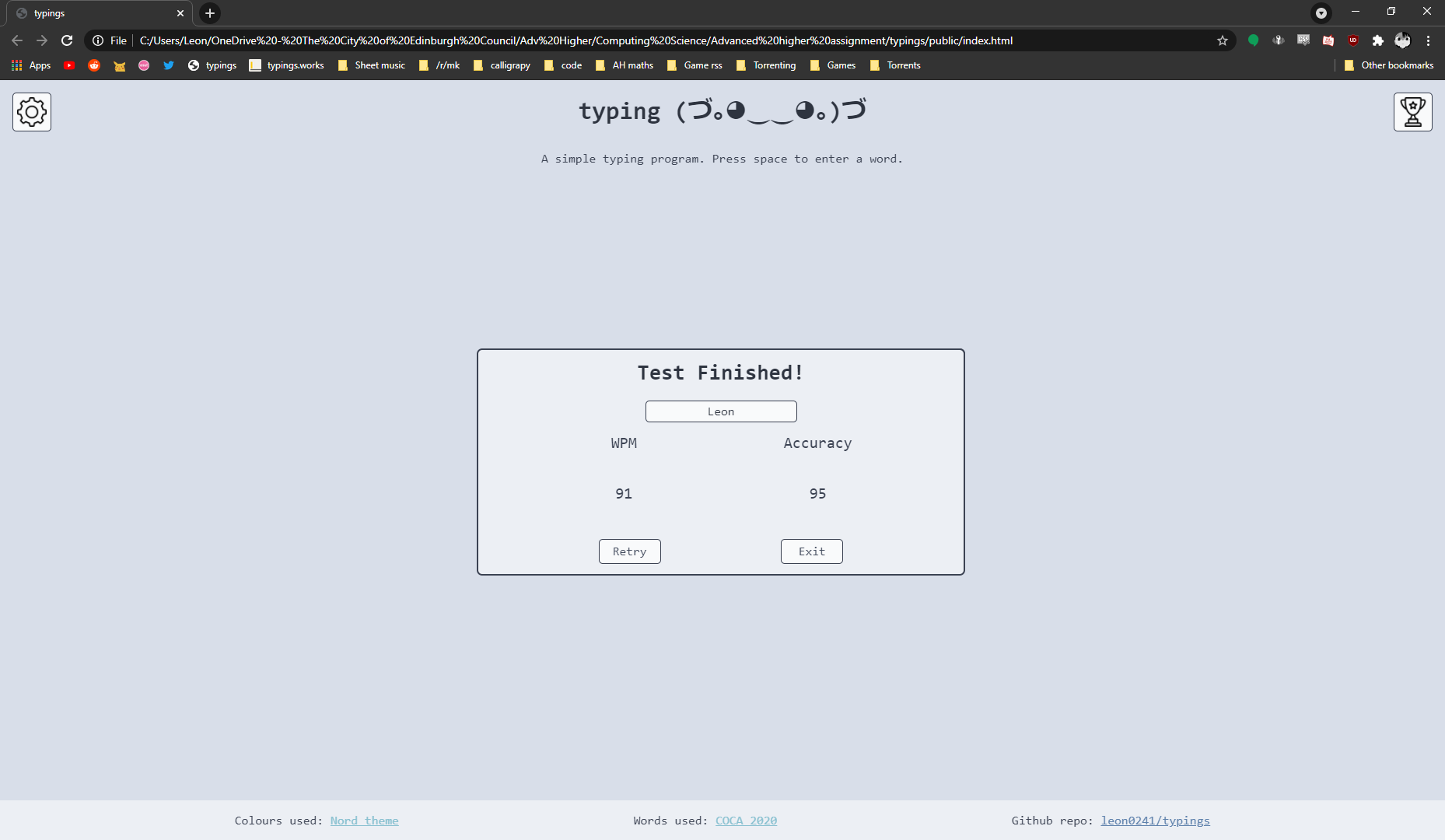
Component Test – C5 (Temporary console log)



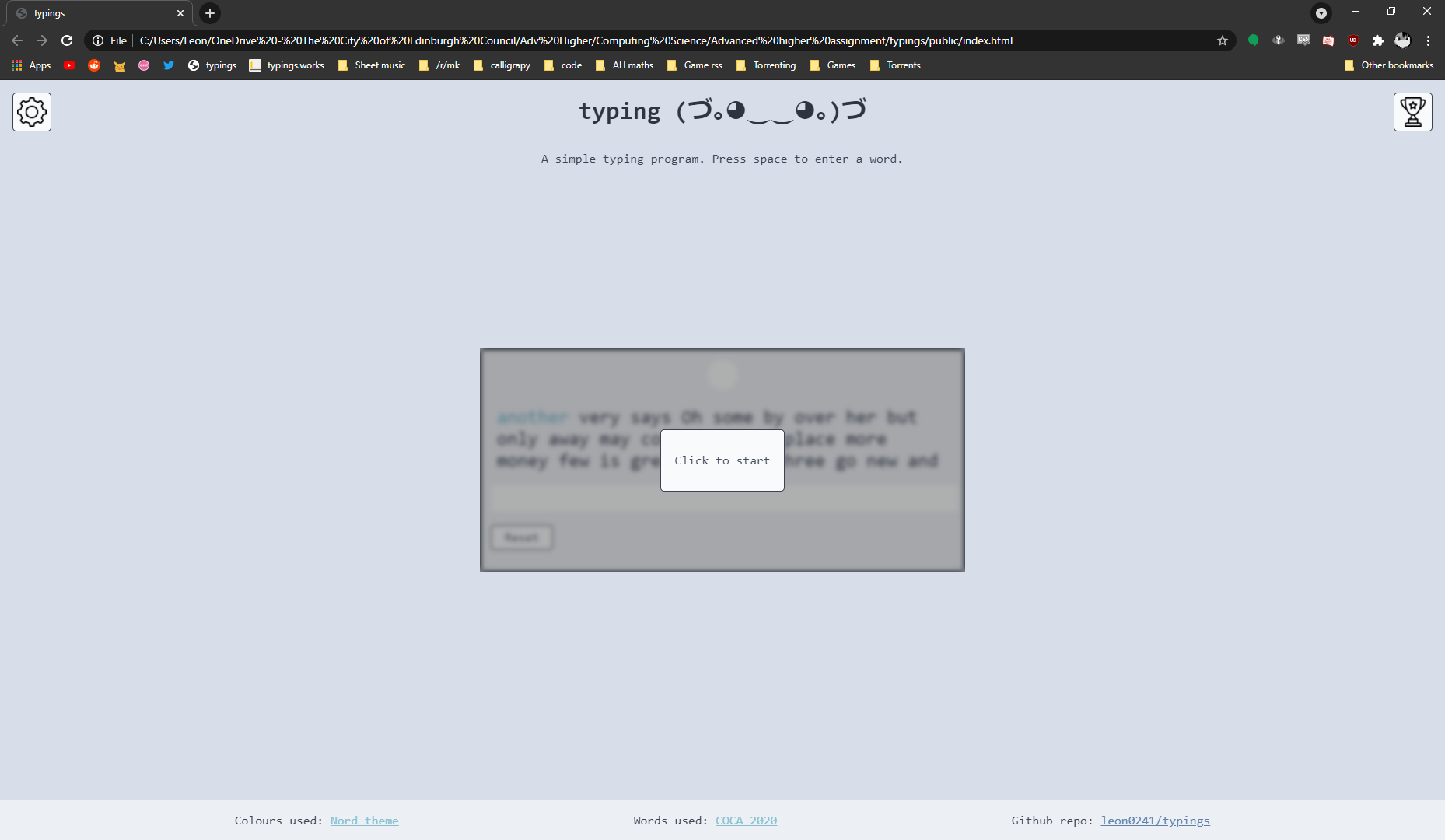
Component Test – C6



Component Test – C7( Name has been kept)



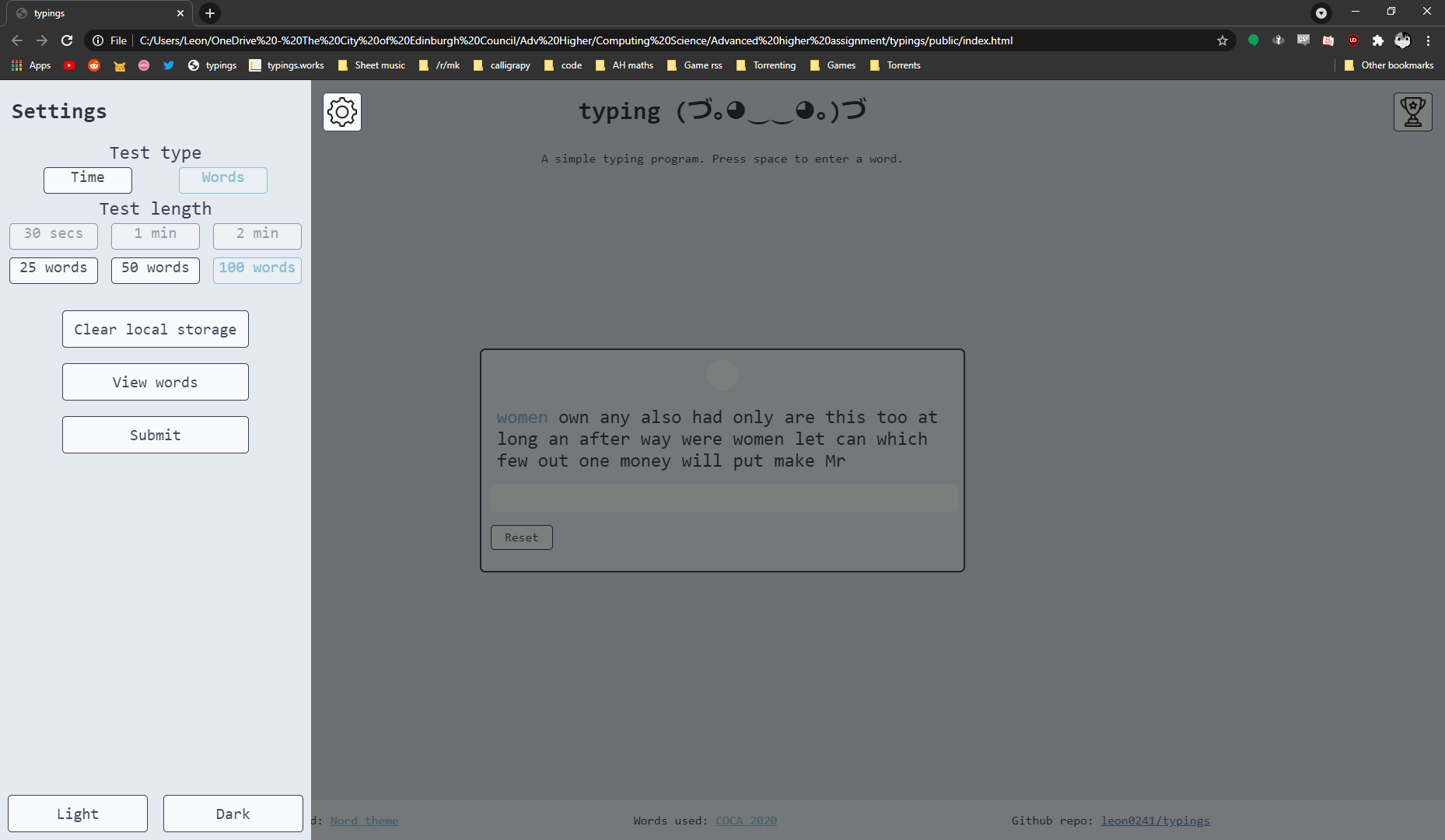
Exit button takes you to start



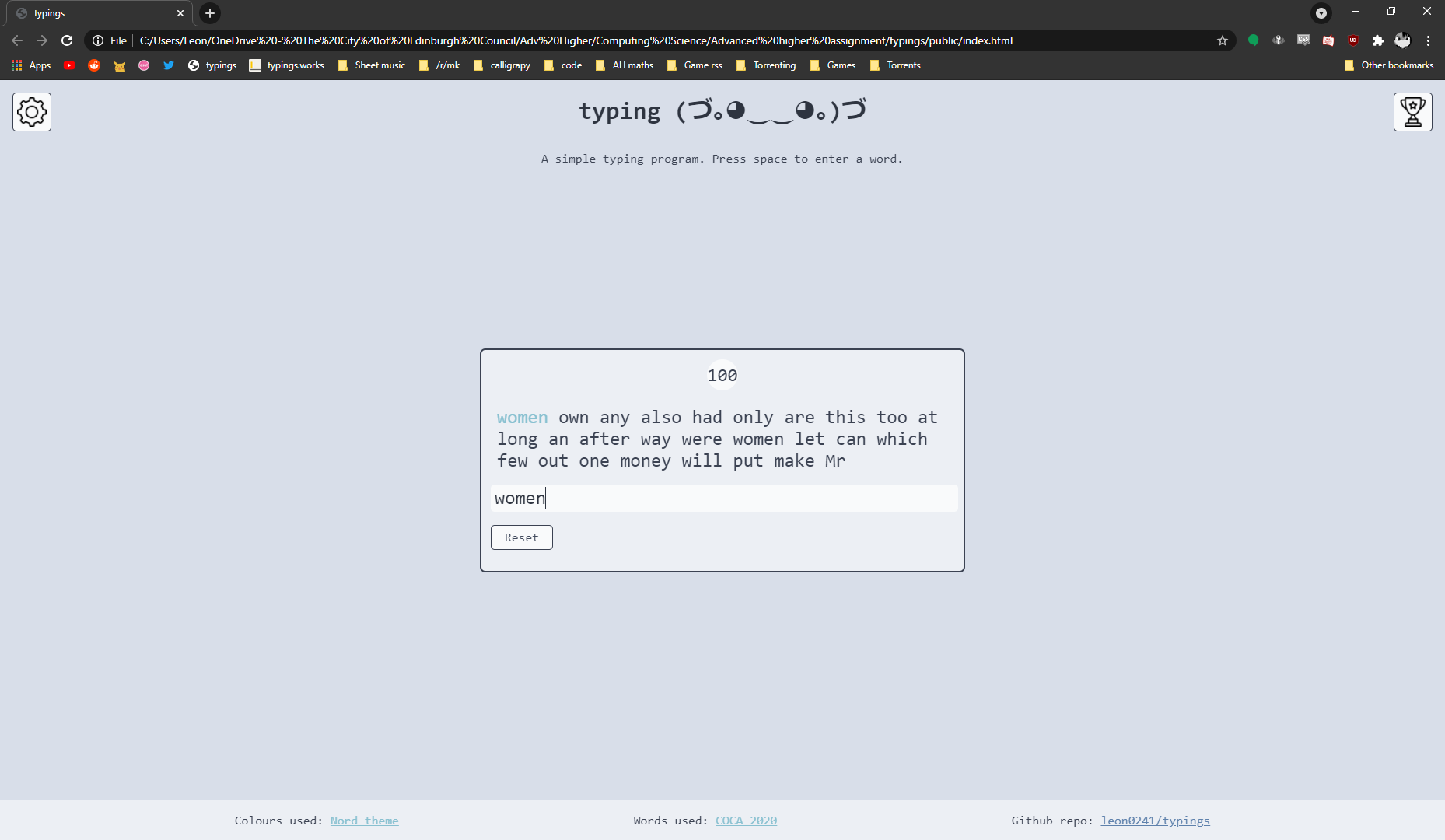
Integration Testing

|  |  |  |  |
| --- | --- | --- | --- |
| Test | Component | Description | Successful? |
| I1 | Settings work properly | Settings will work properly on submit, and update the site even after reload | Yes |
| I2 | Program works properly | Program will run through, and save a user score into localStorage | Yes |
| I3 | Scoreboard displays and sorts properly | Scoreboard will display all user submitted scores sorted from the highest wpm down | Yes |
| I4 | Clear local data works properly | Clearing local storage will remove any user preferences, and scores | Yes |

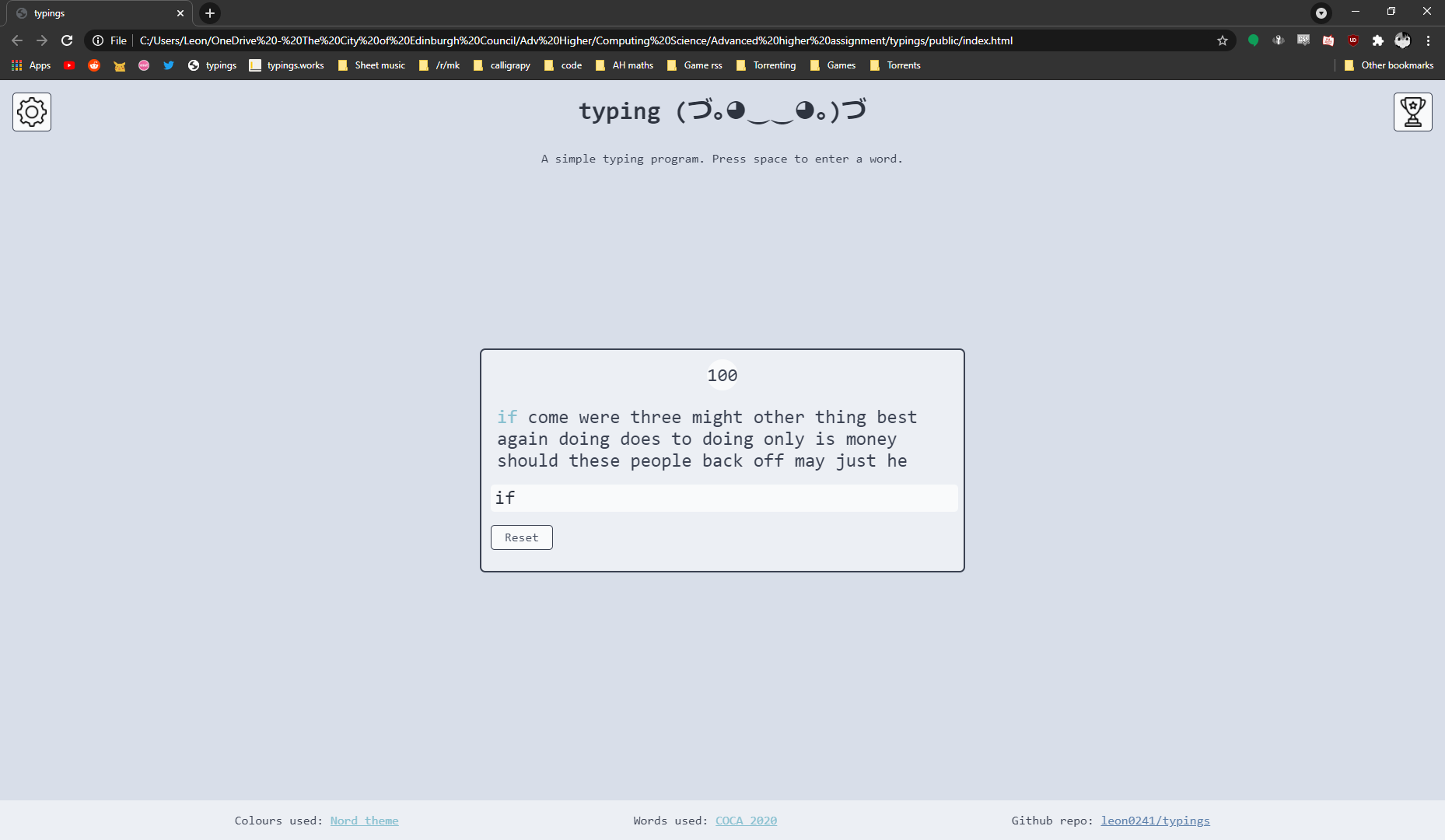
Integration Test – I1



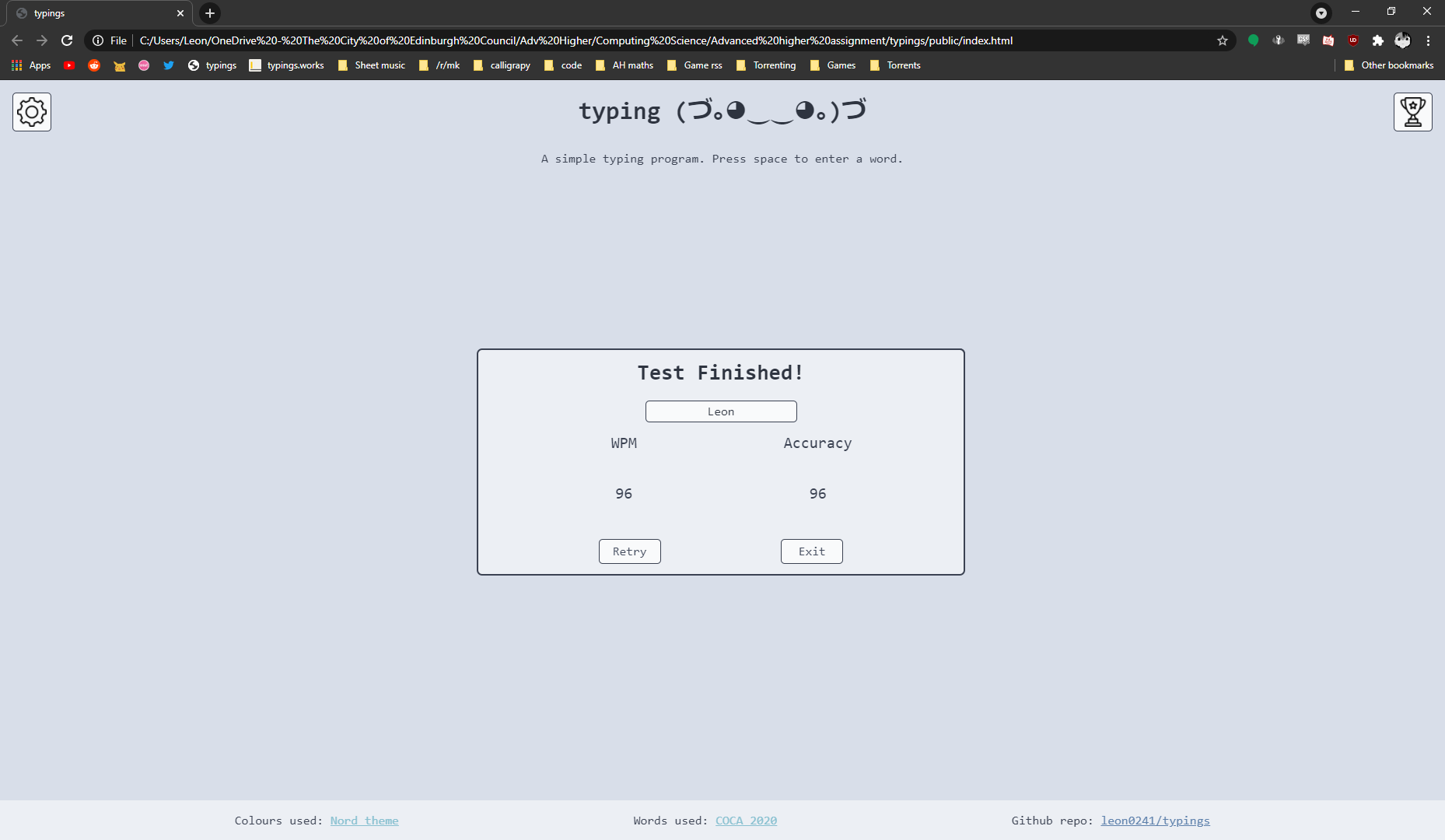
Submits, and leads to screenshot below



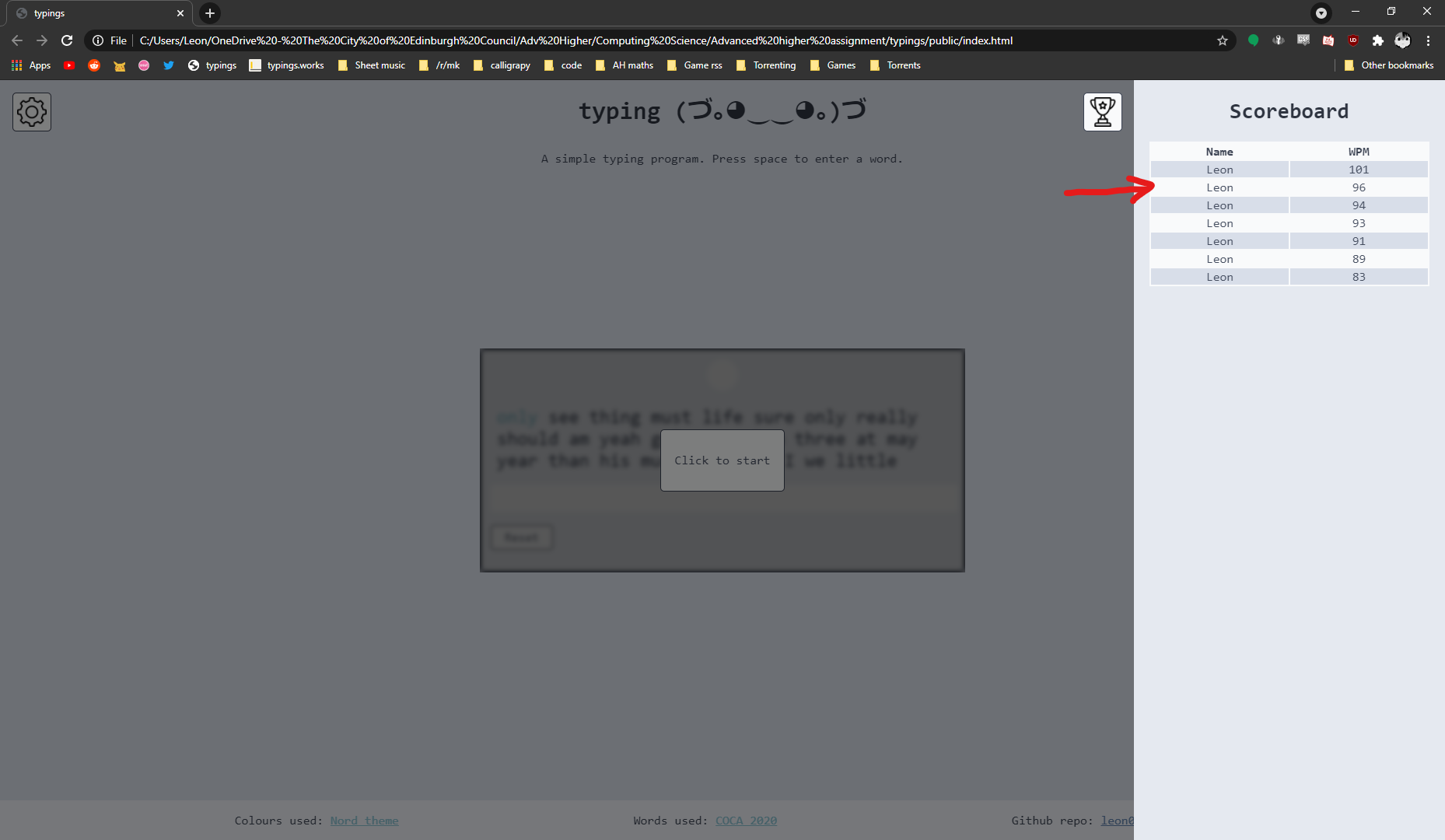
Settings on reload



Integration Test – I2

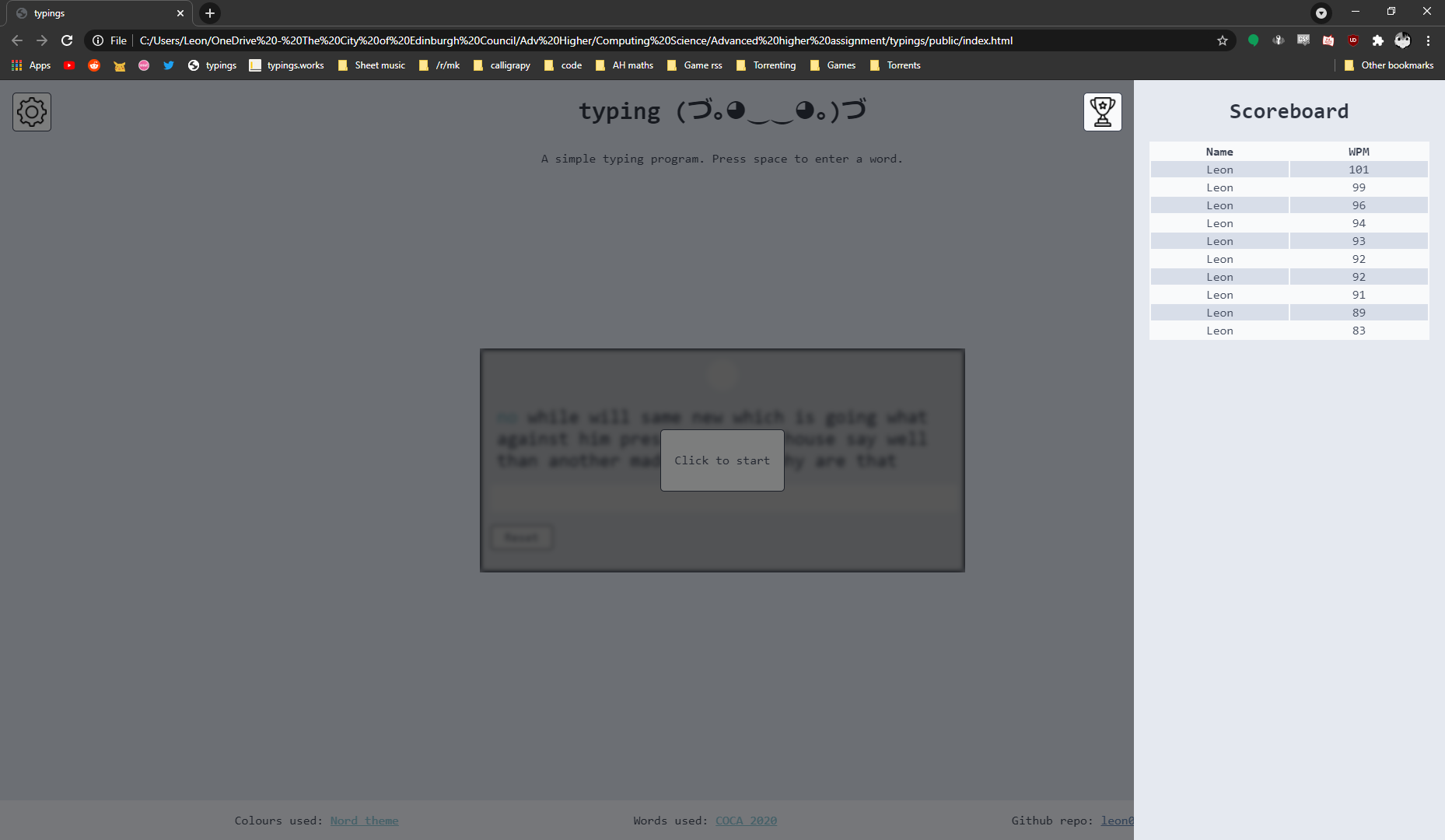


Updated Scoreboard



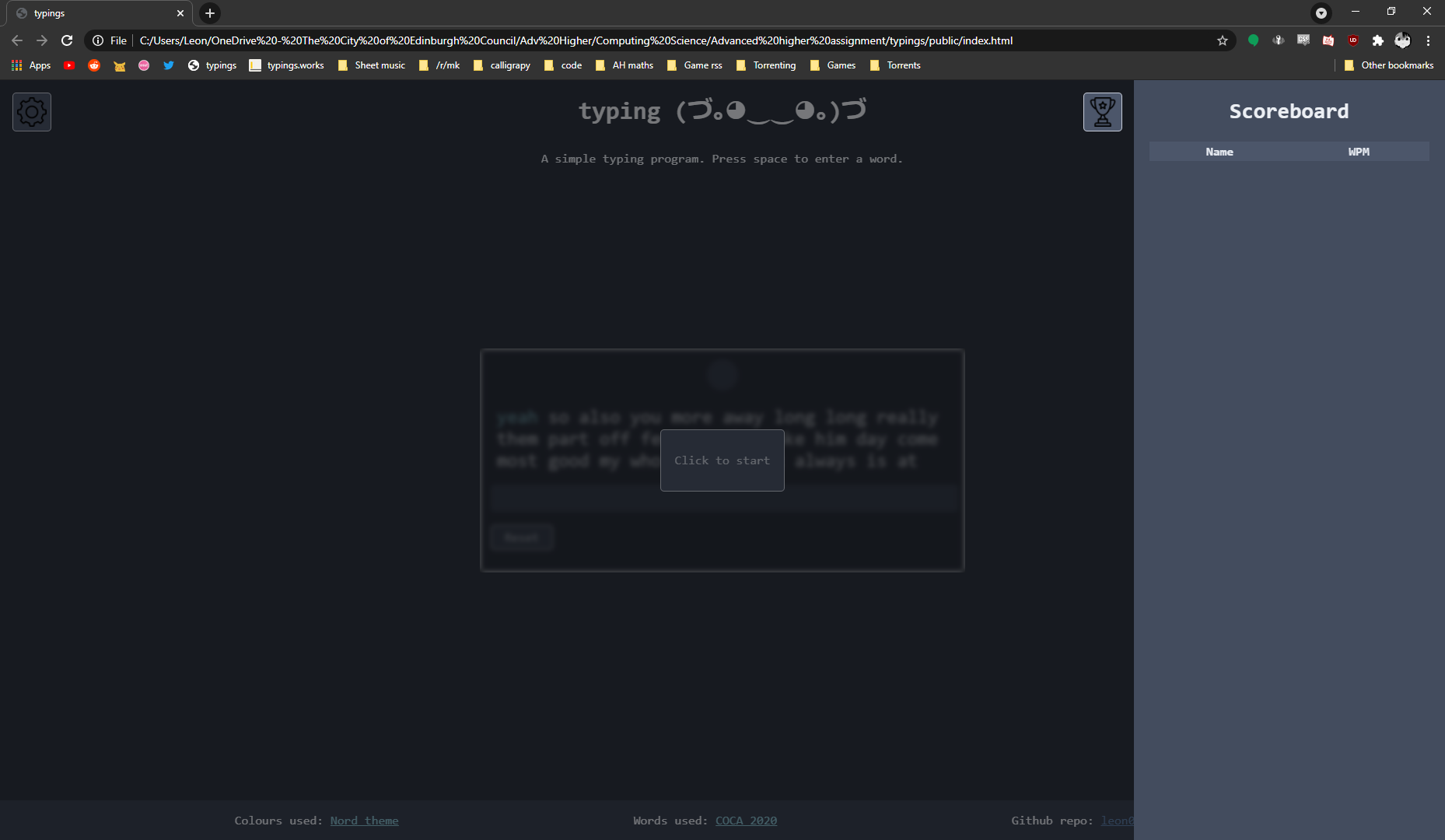
Integration Test – I3

10 scores, sorted properly



Integration Test – I4

Upon localStorage clear, all settings and scores are reset



**Code Evidence**

**index.html**

<!DOCTYPE html>

<html lang="en" dir="ltr">

  <head>

    <meta charset="utf-8" />

    <title>typings</title>

*<!-- Import stylesheet -->*

    <link rel="stylesheet" type="text/css" href="styles/styles.min.css" />

*<!-- Import scripts -->*

    <script type="text/javascript" src="scripts/words.min.js" defer></script>

    <script type="text/javascript" src="scripts/main.min.js" defer></script>

    <script type="text/javascript" src="scripts/game.min.js" defer></script>

  </head>

  <body class="dark">

*<!-- Overlay for nav bars -->*

    <div id="overlay"></div>

*<!-- Modal to view words -->*

    <div id="wordModal" class="modal">

*<!-- Modal contents -->*

      <div id="modalContent">

*<!-- Modal Header -->*

        <header id="modalHeader" class="modalSection">

          <h2>view words</h2>

          <span id="modalExit">&times;</span>

        </header>

*<!-- Modal sort buttons -->*

        <section id="modalButtons" class="modalSection">

          <button id="freqSort" class="modalButton">Sort by frequency</button>

          <button id="lengthSort" class="modalButton">Sort by length</button>

          <button id="alphaSort" class="modalButton">

            Sort alphabetically

          </button>

        </section>

*<!-- Modal words -->*

        <section id="modalWords" class="modalSection">

          <p>Click a button to view words...</p>

*<!-- Empty and will be filled with DOM -->*

        </section>

      </div>

    </div>

*<!-- Game window -->*

    <main>

*<!-- On screen text -->*

      <section id="pageText">

*<!-- Header-->*

        <header id="topText">

*<!-- Title -->*

          <h1>typing (づ｡◕‿‿◕｡)づ</h1>

*<!-- Description -->*

          <p>A simple typing program. Press space to enter a word.</p>

        </header>

*<!-- Footer -->*

        <footer id="bottomText">

*<!-- Colours credit - nord theme -->*

          <p>

            Colours used:

            <a href="https://www.nordtheme.com/">Nord theme</a>

          </p>

*<!-- Word list credit - COCA -->*

          <p>

            Words used:

            <a href="https://www.wordfrequency.info/">COCA 2020</a>

          </p>

*<!-- Github repo link -->*

          <p>

            Github repo:

            <a href="https://github.com/leon0241/typings">leon0241/typings</a>

          </p>

        </footer>

      </section>

*<!-- Settings nav bar -->*

      <nav id="settingsNav" class="navbar">

*<!-- Form with settings -->*

        <form

          name="settingsForm"

          id="settingsForm"

          onsubmit="newGame(this);return false"

        >

*<!-- Header -->*

          <h2 id="settingsHeader">Settings</h2>

*<!-- Test type -->*

          <label for="test\_type" class="radioHeader">Test type</label>

          <section class="settingsRadio" id="typeSelector">

*<!-- Time -->*

            <input type="radio" name="test\_type" id="radiotime" value="0" />

            <label for="radiotime" class="radioSelector">Time</label>

*<!-- Words -->*

            <input type="radio" name="test\_type" id="radiowords" value="1" />

            <label for="radiowords" class="radioSelector">Words</label>

          </section>

*<!-- Test Length - Time -->*

          <label for="time\_length" class="radioHeader">Test length</label>

          <section class="settingsRadio" id="timeSelector">

*<!-- 30 seconds -->*

            <input

              type="radio"

              name="time\_length"

              id="radio30s"

              class="lengthRadio"

              value="0"

            />

            <label for="radio30s" class="radioSelector">30 secs</label>

*<!-- 1 minute -->*

            <input

              type="radio"

              name="time\_length"

              id="radio1m"

              class="lengthRadio"

              value="1"

            />

            <label for="radio1m" class="radioSelector">1 min</label>

*<!-- 2 minutes -->*

            <input

              type="radio"

              name="time\_length"

              id="radio2m"

              class="lengthRadio"

              value="2"

            />

            <label for="radio2m" class="radioSelector">2 min</label>

          </section>

*<!-- Test Length - Words -->*

          <section class="settingsRadio" id="wordSelector">

*<!-- 25 words -->*

            <input

              type="radio"

              name="word\_length"

              id="radio25w"

              class="lengthRadio"

              value="0"

            />

            <label for="radio25w" class="radioSelector">25 words</label>

*<!-- 50 words -->*

            <input

              type="radio"

              name="word\_length"

              id="radio50w"

              class="lengthRadio"

              value="1"

            />

            <label for="radio50w" class="radioSelector">50 words</label>

*<!-- 100 words -->*

            <input

              type="radio"

              name="word\_length"

              id="radio100w"

              class="lengthRadio"

              value="2"

            />

            <label for="radio100w" class="radioSelector">100 words</label>

          </section>

          <br />

*<!-- Clear local storage button -->*

          <button

            type="button"

            class="settingsMenuButton"

            onclick="localStorage.clear()"

          >

            Clear local storage

          </button>

*<!-- Open modal button -->*

          <button type="button" class="settingsMenuButton" id="openModal">

            View words

          </button>

*<!-- Submit button -->*

          <button id="submitButton" class="settingsMenuButton" type="submit">

            Submit

          </button>

*<!-- Theme buttons -->*

          <div id="themeButtons">

*<!-- Light theme -->*

            <button type="button" class="settingsMenuButton" id="lightButton">

              Light

            </button>

*<!-- Dark theme -->*

            <button type="button" class="settingsMenuButton" id="darkButton">

              Dark

            </button>

          </div>

        </form>

      </nav>

*<!-- Button to open settings -->*

      <button type="button" id="settingsButton" class="navButton">

*<!-- Settings button -->*

        <img

          src="resources/images/settings.png"

          class="navIcon"

          alt="Settings"

        />

*<!-- Tooltip -->*

        <span class="tooltip">Settings</span>

      </button>

*<!-- Full screen container -->*

      <section id="screenContainer">

*<!-- Game size container -->*

        <div id="gameContainer">

*<!-- Game window -->*

          <div id="game">

*<!-- Game time countdown -->*

            <section class="gameArea" id="gameStatsArea">

              <div id="statBox">

                <span class="gameStat" id="gameProgress"></span>

              </div>

            </section>

*<!-- Game word display -->*

            <section class="gameArea" id="gameWordArea">

*<!-- Empty and will be filled with DOM -->*

            </section>

*<!-- Game typing field -->*

            <section class="gameArea" id="gameTypingArea">

              <input type="text" id="gameTypingField" />

            </section>

*<!-- Game reset button -->*

            <section class="gameArea" id="gameResetArea">

              <button type="button" onclick="resetGame()" id="gameResetButton">

                Reset

              </button>

            </section>

          </div>

*<!-- Start overlay window -->*

          <div id="startOverlay" class="gameOverlay">

            <button id="startButton" type="button">Click to start</button>

          </div>

*<!-- Finish overlay window -->*

          <div id="finishOverlay" class="gameOverlay">

*<!-- Form with information -->*

            <form id="finishForm" onsubmit="submitFinishForm();return false">

*<!-- Title -->*

              <section class="finishArea" id="finishTitleArea">

                <h2>Test Finished!</h2>

              </section>

*<!-- Name typing field -->*

              <section class="finishArea" id="finishNameArea">

                <input

                  type="text"

                  class="finishTypingField"

                  name="name"

                  id="finishTypingField"

                />

              </section>

*<!-- WPM and Acc display -->*

              <section class="finishArea" id="finishStatArea">

*<!-- WPM label -->*

                <span class="finishStat StatText" id="finishWPM"> WPM </span>

*<!-- Accuracy label -->*

                <span class="finishStat StatText" id="finishAcc">

                  Accuracy

                </span>

*<!-- WPM count -->*

                <span class="finishStat StatNum" id="gameWPM" name="WPM">

*<!-- Empty and will be filled with DOM -->*

                </span>

*<!-- Accuracy count -->*

                <span

                  class="finishStat StatNumber"

                  id="gameAccuracy"

                  name="Acc"

                >

*<!-- Empty and will be filled with DOM -->*

                </span>

              </section>

*<!-- Hidden form inputs to submit info -->*

              <section class="finishArea" id="finishHiddenForms">

*<!-- WPM -->*

                <input

                  type="text"

                  name="wpm"

                  id="hiddenWpm"

                  class="hiddenField"

                />

*<!-- Accuracy -->*

                <input

                  type="text"

                  name="acc"

                  id="hiddenAcc"

                  class="hiddenField"

                />

              </section>

*<!-- Submit buttons -->*

              <section class="finishArea" id="finishRetryArea">

*<!-- Submit and retry -->*

                <button

                  type="submit"

                  class="finishButton"

                  onclick="finishedReset(false)"

                >

                  Retry

                </button>

*<!-- Submit and exit -->*

                <button

                  type="submit"

                  class="finishButton"

                  onclick="finishedReset(true)"

                >

                  Exit

                </button>

              </section>

            </form>

          </div>

        </div>

      </section>

*<!-- Button to open scores -->*

      <button type="button" id="scoresButton" class="navButton">

*<!-- Button image -->*

        <img src="resources/images/trophy.png" class="navIcon" alt="Scores" />

*<!-- Button tooltip -->*

        <span class="tooltip" id="scoreTooltip">Scoreboard</span>

      </button>

*<!-- Scores nav bar -->*

      <nav id="scoresNav" class="navbar">

*<!-- Title -->*

        <h2>Scoreboard</h2>

*<!-- Table with values -->*

        <table id="scoreboard">

*<!-- Headers -->*

          <thead>

            <tr>

              <th class="scoreHeader">Name</th>

              <th class="scoreHeader">WPM</th>

            </tr>

          </thead>

*<!-- Scores -->*

          <tbody id="scorebody">

*<!-- Empty and will be filled with DOM -->*

          </tbody>

        </table>

      </nav>

    </main>

  </body>

</html>

*<!-- <div>Icons made by <a*

*href="https://www.freepik.com"*

*title="Freepik">Freepik</a> from*

*<a href="https://www.flaticon.com/"*

*title="Flaticon">www.flaticon.com*

*</a></div> -->*

**game.ts**

*/\*================*

*|    CLASSES     |*

*================\*/*

*// Basic game options - type, length, words array*

class GameSettings {

  \_name: string;

  \_type: number;

  \_length: number;

  readonly \_words: string[];

*// ==Constructor==*

  constructor(type: number, length: number, words: string[]) {

    this.\_name = "";

*// Length of game (0,1,2)*

    this.\_length = length;

*// Type of game(time = 0, words = 1)*

    this.\_type = type;

*// Array of top 200 words*

    this.\_words = words;

  }

*// ==Class Getters==*

  get name(): string {

    return this.\_name;

  }

*// 0 = lowest, 1 = standard, 2 = high*

  get length(): number {

    return this.\_length;

  }

*// 0 = time, 1 = words*

  get type(): number {

    return this.\_type;

  }

*// ==Class Setters==*

  set length(value: number) {

    this.\_length = value;

  }

  set type(value: number) {

    this.\_type = value;

  }

*// ==Class Functions==*

*// Gets the Length in terms of time/number of words*

  getCalculatedLength(): number {

    let output: number;

*// Concatenates type and length into a string*

    let part1 = this.\_type.toString();

    let part2 = this.\_length.toString();

    let switcher = part1.concat(part2);

*// Switch statement to go to each option*

    switch (switcher) {

*// Time(30s, 60s, 120s)*

      case "00": {

        output = 30;

        break;

      }

      case "01": {

        output = 60;

        break;

      }

      case "02": {

        output = 120;

        break;

      }

*// Words(25 words, 50 words, 100 words)*

      case "10": {

        output = 25;

        break;

      }

      case "11": {

        output = 50;

        break;

      }

      case "12": {

        output = 100;

        break;

      }

    }

    return output;

  }

*// Reads the name from the input box and stores it*

  setName(): void {

*// Read input element*

    let textbox: HTMLInputElement = document.querySelector(

      "#finishTypingField"

    );

*// If name is empty, set name to "Anon"*

    let name = textbox.value;

    if (name === "") {

      name = "Anon";

    }

*// Set name to name variable*

    this.\_name = name;

  }

}

*// Methods used to modify values by the player throughout the game*

class UserGame extends GameSettings {

  \_calculatedStats: number[];

  \_characters: number;

  \_gameWords: string[];

  \_timeTaken: number;

  \_userWordCount: number;

  \_wordErrors: number;

*// ==Constructor==*

  constructor(...args: [number, number, string[]]) {

    super(...args);

*// WPM and Accuracy of player*

    this.\_calculatedStats = [0, 0];

*// Number of characters typed*

    this.\_characters = 0;

*// Array of words in the game*

    this.\_gameWords = [];

*// Time taken to complete game*

    this.\_timeTaken = 0;

*// Number of times spacebar pressed*

    this.\_userWordCount = 0;

*// Number of errors made*

    this.\_wordErrors = 0;

  }

*// ==Class Getters==*

  get characters(): number {

    return this.\_characters;

  }

  get calculatedStats(): number[] {

    return this.\_calculatedStats;

  }

  get gameWords(): string[] {

    return this.\_gameWords;

  }

  get timeTaken(): number {

    return this.\_timeTaken;

  }

  get userWordCount(): number {

    return this.\_userWordCount;

  }

*// Returns the current word*

  get word(): string {

    return this.\_gameWords[this.\_userWordCount - 1];

  }

*// ==Class Setters==*

  set calculatedStats(value: number[]) {

    this.\_calculatedStats = value;

  }

  set timeTaken(value: number) {

    this.\_timeTaken = value;

  }

*// Adds 1 to the number of characters typed*

*// char is the keyCode of the typed character*

  incrementCharacters(char: number): void {

*// 8: backspace, 16: shift, 17: ctrl, 20: caps, 46: delete*

    const options = [8, 16, 17, 20, 46];

*// if the index of char is not in the array then go*

*// alternate way of multiple or statements*

    if (options.indexOf(char) === -1) {

      this.\_characters += 1;

    }

  }

*// Adds 1 to word count*

  incrementWordCount(): void {

    this.\_userWordCount += 1;

  }

*// Adds 1 to errors*

  incrementWordErrors(): void {

    this.\_wordErrors += 1;

  }

*// ==Class Functions==*

*// Changes length and type when settings are edited*

*//TODO: Settings DOM to call this function*

  editGameData(x: string, y: string): void {

    let type: number = parseInt(x, 10);

    let length: number = parseInt(y, 10);

    this.\_type = type;

    this.\_length = length;

  }

*// Creates an array of 30 with random words*

  initialiseArray(): void {

*// Creates temp variable, maybe faster to assign to this.gameWords before*

    let tempGameWords = [];

    for (let i = 0; i < 50; i++) {

*// Random integer from 0 to length of array and assigns*

      let randint = Math.floor(Math.random() \* words.length);

      tempGameWords[i] = this.\_words[randint];

    }

    this.\_gameWords = tempGameWords;

  }

*// Checks if the word is correct or not*

  wordCheck(): void {

*// Removes the spacebar from your input word*

    let inputWord = gameTypingField.value.trim();

    let wordComparison = this.word;

*// Defines the item to change the class of*

    let nodeItem = document.getElementById("previousWord");

*// Checks if the input word is the same as the actual word*

    if (inputWord === wordComparison) {

      DOMFunctions.setAnswer(true, nodeItem);

    } else {

      this.incrementWordErrors();

      DOMFunctions.setAnswer(false, nodeItem);

    }

*// Checking if next word is on next line, and deletes the first line*

*// Sets DOMRect of the next word, will test if it is on the next line or not*

    DOMFunctions.checkRow();

  }

*// Appends a new word to the araray*

  newWord(): void {

    let gameWords = this.\_gameWords;

*// Random number up to the length of total words array*

    let randint = Math.floor(Math.random() \* words.length);

    gameWords.push(words[randint]);

*// Goes to updateWords with array gameWords, position length of gameWords - 1 (for 0 counting array)*

    DOMFunctions.updateWords(this.\_gameWords, this.\_gameWords.length - 1);

  }

*// Calculates WPM and accuracy of the player*

  calculateStats(): void {

*// Variable initialisation*

    let chars = this.\_characters;

    let time = this.\_timeTaken;

    let errors = this.\_wordErrors;

    let totalWords = this.\_userWordCount;

*// netWords: The number of correct words (assuming a word is 5 letters)*

    let netWords = chars / 5 - errors;

*// Value to divide to get words per minute*

    let timeFactor = time / 60;

*// Calculate WPM*

    let netWPM =

      netWords < 0

        ? *//Account for error where most words that appear are less than 5 letters resulting in negative WPM*

          (totalWords - errors) / timeFactor

        : netWords / timeFactor;

*// Calculate accuracy*

    let accuracy = ((totalWords - errors) / totalWords) \* 100;

*// Rounds each result to the nearest integer*

    let tempStats = [netWPM, accuracy];

    let stats = [];

    tempStats.forEach((element) => stats.push(Math.round(element)));

    this.\_calculatedStats = stats;

  }

*// Reset all the stats to 0*

  resetStats(): void {

    this.\_calculatedStats = [0, 0];

    this.\_characters = 0;

    this.\_gameWords = [""];

    this.\_timeTaken = 0;

    this.\_userWordCount = 0;

    this.\_wordErrors = 0;

    DOMFunctions.position = 0;

  }

}

*// Methods that are used to control the game*

class GameFunctions extends UserGame {

  constructor(...args: [number, number, string[]]) {

    super(...args);

  }

*// Starts the game*

  startGame(): void {

    inGame = true;

*// Go to timedgame or wordGame depending on type variable*

    let gameType =

      this.\_type === 0 ? this.goToTimedGame() : this.goToWordGame(); *// TODO: maybe fix this idk*

  }

*// Callback function for timed game*

  goToTimedGame(): void {

*// Change progress to timer*

    DOMFunctions.changeGameProgress(this.getCalculatedLength());

*// setInterval callback function*

    this.timeTimer(() => {

*// Finish game*

      this.finishGame();

    });

  }

*// setInterval timer for a timed game*

  timeTimer(callback: any): void {

*// Set duration to length*

    let duration = this.getCalculatedLength();

*// Init time*

    let time = 1;

*// SetInterval - timer with tick of 1000 seconds*

    let gameTimer = setInterval(() => {

*// On reset button i think*

      if (inGame === false) {

        clearInterval(gameTimer);

        return;

      }

*// if timer over the max time*

      else if (time >= duration) {

*// Stop timer*

        clearInterval(gameTimer);

*// Set time taken to the duration*

        this.\_timeTaken = duration;

*// To callback function*

        callback();

      }

*// Change the on screen timer*

      DOMFunctions.changeGameProgress(duration - time);

*// Add 1 to time*

      time++;

    }, 1000);

  }

*// Callback function for word game*

  goToWordGame(): void {

*// Change progress to words remaining*

    let gameLength = this.getCalculatedLength();

    DOMFunctions.changeGameProgress(gameLength);

*// setInterval callback function*

    this.wordTimer(() => {

*// Finish game*

      this.finishGame();

    });

  }

*// setInterval timer for a word game*

  wordTimer(callback: any): void {

    let totalWordCount = this.getCalculatedLength();

    let time = 1;

*// Variable to keep track of timer*

    let inGameSeconds = 0;

*// Interval to loop setInterval (1/10 seconds)*

    let interval = 100;

*// SetInterval - timer*

    let gameTimer = setInterval(() => {

      if (inGame === false) {

        clearInterval(gameTimer);

        return;

      }

*// If word count is above total words*

      else if (this.\_userWordCount >= totalWordCount) {

*// Stop timer*

        clearInterval(gameTimer);

*// Set time taken to the duration*

        this.\_timeTaken = time;

*// To callback function*

        callback();

      }

*// if time is divisible by 1000 then add a second*

      inGameSeconds += interval;

      if (inGameSeconds % 1000 === 0) {

        time++;

      }

    }, interval); *// Repeat every 1/10 seconds so there is no delay when finishing game*

  }

*// On game finish*

  finishGame(): void {

*// Set in game to false*

    inGame = false;

*// Calculate the user stats*

    this.calculateStats();

*// Show finish overlay*

    DOMFunctions.showFinish();

*// Put stats in hidden form element*

    DOMFunctions.appendFormData();

  }

*// Functions and methods called after a word is typed*

  goToNextWord(): void {

    DOMFunctions.updateNodeList();

*// Increase word count*

    this.incrementWordCount();

    this.newWord();

*// Increase position (this is used for DOM styling)*

    DOMFunctions.incrementPosition();

*// Highlights the current word*

    DOMFunctions.highlightCurrentWord();

*// Checks: if word is correct, if word is last on its line*

    this.wordCheck();

*// Clears the value of the field*

    gameTypingField.value = "";

*//debugger;*

    if (this.\_type === 1) {

      DOMFunctions.changeGameProgress(

        this.getCalculatedLength() - this.\_userWordCount

      );

    }

  }

}

*/\*================*

*|   Functions    |*

*================\*/*

*// Function when button is submitted*

function newGame(that: any) {

*// Get type from form*

  let type: string = that.test\_type.value;

*// Get length from form, depending on the type*

  let length: string =

    type === "0" ? that.time\_length.value : that.word\_length.value;

*// If the type is empty, put default type*

  if (type === "") {

    type = defOpt[0].toString();

  }

*// If the length is empty, put default length*

  if (length === "") {

    length = defOpt[1].toString();

  }

*// Change the game data to the values in the form*

  Game.editGameData(type, length);

*// Change the game settings in the class*

  DOMFunctions.setSettings(type, length);

*// Reset game function*

  resetGame();

}

*// Function when reset button is pressed*

function resetGame() {

*// Reset game stats*

  Game.resetStats;

*// Set inGame and clicked globals to false*

  inGame = false;

  clicked = false;

*// Empty the value of the typing field*

  gameTypingField.value = "";

*// Empty the progress field*

  DOMFunctions.changeGameProgress("");

*//Initialise the array*

  Game.initialiseArray();

*// Show the array in the DOM*

  DOMFunctions.showArray(Game.gameWords);

}

*// Function when exit or retry button is changed*

function finishedReset(exit) {

*// Read name from input box and set it*

  Game.setName();

*// Hide finish overlay*

  DOMFunctions.hideFinish();

*// Submit score to local storage*

  Scores.submitToLocalStorage(Game.name, Game.calculatedStats[0]);

*// Update the scoreboard*

  Scores.updateScoreboard();

*// Reset game function*

  resetGame();

*// Check if exit is true and show the start overlay*

  if (exit === true) {

    DOMFunctions.showStart();

  }

}

*// On mouse click on typing field*

gameTypingField.onclick = () => {

*// Sets condition to true so if a key is pressed the game will start*

  if (inGame === false) {

    clicked = true;

  }

};

*// On character pressed in the typing field*

gameTypingField.onkeydown = (e) => {

*// Check if clicked is true, and start game if met*

  if (clicked === true) {

    Game.startGame();

    clicked = false;

  }

*// Check if inGame is true before doing any calculations*

  if (inGame === true) {

*// Increment the character count with keycode of typed letter*

    Game.incrementCharacters(e.keyCode);

*// If spacebar is pressed => function go to next word*

    if (e.keyCode == 32) {

      Game.goToNextWord();

    }

  }

};

*/\*================*

*|     GAME       |*

*================\*/*

*// Check if in game*

let inGame = false;

*// Check if clicked*

let clicked = false;

*// Default options (time, 1 minute)*

let defOpt = [0, 1];

*// Define new classes*

let Game = new GameFunctions(defOpt[0], defOpt[1], words); *// Words = 1, time = 0*

let DOMFunctions = new DOMManipulation();

let Scores = new Scoreboard();

*// On window load*

window.onload = (e) => {

*// Check if the length is empty - no settings*

  if (localStorage.length === 0) {

*// Set the defualt settings*

    DOMFunctions.setSettings(defOpt[0].toString(), defOpt[1].toString());

  }

*// Check if the localStorage isn't empty - import settings*

  else if (localStorage.length > 0) {

*// Get the settings from localStorage*

    let importSettings: string[] = DOMFunctions.getSettings();

*// Change the settings in the game*

    Game.editGameData(importSettings[0], importSettings[1]);

*// Set the theme*

    setTheme(importSettings[2]);

*// Check if the localStorage is over 3 - scores stored*

    if (localStorage.length > 3) {

*// Initialise the user score index with an offset of 3*

      Scores.initIndex(3);

*// Initialise the scoreboard in DOM*

      Scores.initScoreboard();

    }

  }

*// empty the game typing field*

  gameTypingField.value = "";

*// Initialise array*

  Game.initialiseArray();

*// Show the array*

  DOMFunctions.showArray(Game.gameWords);

};

*//TODO: reset needs to clear input box*

*//TODO: submit needs to reset the game*

**main.ts**

*/\*=====================*

*|    DOM Variables    |*

*=====================\*/*

*// Body*

const body: HTMLElement = document.querySelector("body");

const overlay: HTMLElement = document.getElementById("overlay");

const settingsButton: HTMLElement = document.getElementById("settingsButton");

const scoresButton: HTMLElement = document.getElementById("scoresButton");

*// Game*

const game: HTMLElement = document.getElementById("game");

const gameProgress: HTMLElement = game.querySelector("#gameProgress");

const gameWordArea: HTMLElement = game.querySelector("#gameWordArea");

const gameTypingField: HTMLInputElement = game.querySelector(

  "#gameTypingField"

);

*// Start overlay*

const startOverlay: HTMLElement = document.getElementById("startOverlay");

*// Finish overlay*

const finishOverlay: HTMLElement = document.getElementById("finishOverlay");

const finishForm: HTMLFormElement = finishOverlay.querySelector("#finishForm");

const nameInput: HTMLInputElement = finishForm.querySelector(

  "#finishTypingField"

);

const gameWPM: HTMLElement = finishForm.querySelector("#gameWPM");

const gameAccuracy: HTMLElement = finishForm.querySelector("#gameAccuracy");

const hiddenWPMInput: HTMLInputElement = finishForm.querySelector("#hiddenWpm");

const hiddenAccInput: HTMLInputElement = finishForm.querySelector("#hiddenAcc");

*// Settings navbar*

const settingsNav: HTMLElement = document.getElementById("settingsNav");

const wordSelector: HTMLElement = settingsNav.querySelector("#wordSelector");

const timeSelector: HTMLElement = settingsNav.querySelector("#timeSelector");

*// Scores navbar*

const scoresNav: HTMLElement = document.getElementById("scoresNav");

const scoreboard: HTMLTableElement = scoresNav.querySelector("#scoreboard");

const scorebody: HTMLTableSectionElement = scoreboard.querySelector(

  "#scorebody"

);

*// View words modal*

const modal: HTMLElement = document.getElementById("wordModal");

const modalWords: HTMLElement = modal.querySelector("#modalWords");

*// Creates a http request to submit form (NOT IN USE)*

*// function submitFinishForm(): void {*

*//   // // Gets form with the new data*

*//   // let newFinishForm: HTMLFormElement = document.querySelector("#finishForm")*

*//   // // Sets up new http request*

*//   // let http = new XMLHttpRequest();*

*//   // // Setup variable with form data object*

*//   // const formData = new FormData(newFinishForm);*

*//   // // Open request method: post, route: finish, true idk*

*//   // http.open("POST", "/finish", true);*

*//   // // Send form data*

*//   // http.send(formData);*

*// }*

*/\*==================*

*|  Onclick events  |*

*===================\*/*

let openToggle = false;

*// Settings button onclick*

settingsButton.onclick = () => {

  if (openToggle === false) {

    sidebarDOM(settingsButton, settingsNav, scoresButton, "open");

    openToggle = true;

  } else {

    sidebarDOM(settingsButton, settingsNav, scoresButton, "close");

    openToggle = false;

  }

};

*// Scores button onclick*

scoresButton.onclick = () => {

  if (openToggle === false) {

    sidebarDOM(scoresButton, scoresNav, settingsButton, "open");

    openToggle = true;

  } else {

    sidebarDOM(scoresButton, scoresNav, settingsButton, "close");

    openToggle = false;

  }

};

*// Set dom of sidebar*

function sidebarDOM(button, nav, button2, state): void {

*// List of elements to add class*

  let elementList = [button, nav, overlay];

*// Check if state is open*

  if (state === "open") {

*// Add open class to each element in list*

    elementList.forEach((element) => {

      element.classList.add("open");

    });

*// Set the z index of the opposite button so it gets hidden by overlay*

    button2.style.zIndex = "2";

*// If state is closed*

  } else {

*// Remove open class to each element in the list*

    elementList.forEach((element) => {

      element.classList.remove("open");

    });

*// Set the z index of the opposite button back to normal*

    button2.style.zIndex = "4";

  }

}

*// Light mode button onclick*

document.getElementById("lightButton").onclick = () => {

  setTheme("light");

};

*// Dark mode button onclick*

document.getElementById("darkButton").onclick = () => {

  setTheme("dark");

};

*// Set theme to either light or dark*

function setTheme(theme: string): void {

  switch (theme) {

    case "light": {

*// Change theme key in localStorage*

      localStorage.setItem("theme", "light");

*// Replace body class*

      body.classList.replace("dark", "light");

*// Set class theme*

      DOMFunctions.theme = "light";

      break;

    }

    case "dark": {

*// Change theme key in localStorage*

      body.classList.replace("light", "dark");

*// Replace body class*

      DOMFunctions.theme = "dark";

*// Set class theme*

      localStorage.setItem("theme", "dark");

      break;

    }

  }

}

*// On start button onclick*

document.getElementById("startButton").onclick = () => {

  DOMFunctions.showBackdrop();

};

*// Open modal button onclick*

document.getElementById("openModal").onclick = () => {

  modal.style.display = "flex";

};

*// Type radio, time button onclick*

document.getElementById("radiotime").onclick = () => {

  chooseTimeWords(wordSelector, timeSelector);

};

*// Type radio, word button onclick*

document.getElementById("radiowords").onclick = () => {

  chooseTimeWords(timeSelector, wordSelector);

};

*// Disable the length for the type not in use*

function chooseTimeWords(a: HTMLElement, b: HTMLElement) {

*// Add deselcted class*

  a.classList.add("deselected");

*// Selects all radio buttons inside the group with*

  let group = a.querySelectorAll(".lengthRadio");

*// Disable each element*

  group.forEach((element: any) => {

    element.disabled = true;

    element.checked = false;

  });

*// If the other class is deselected*

  if (b.classList.contains("deselected")) {

*// Remove class*

    b.classList.remove("deselected");

*// Undisable each element in group*

    let group = b.querySelectorAll(".lengthRadio");

    group.forEach((element: any) => {

      element.disabled = false;

    });

  }

}

*// Modal exit onclick*

document.getElementById("modalExit").onclick = () => {

  modal.style.display = "none";

};

*// Modal Frequency sort onclick*

document.getElementById("freqSort").onclick = () => {

  let arr = words;

  writeWords(arr);

};

*// Modal Length sort onclick*

document.getElementById("lengthSort").onclick = () => {

*// Sort words by length*

  let arr = sortWordsByLength();

  writeWords(arr);

};

*// Modal alphabetical sort onclick*

document.getElementById("alphaSort").onclick = () => {

*// Copy array - pass by value*

  let wordDummy = [...words];

*// Sorting algorithm*

  let arr = wordDummy.sort((a, b) => {

    return a.toLowerCase().localeCompare(b.toLowerCase());

  });

  writeWords(arr);

};

*/\*=================*

*|    Functions    |*

*==================\*/*

*//Write modal words with providedarray*

function writeWords(arr: string[]): void {

*// Clear inner html of field*

  modalWords.innerHTML = "";

*// Repeats for all words*

  for (let i = 0; i < arr.length; i++) {

*// Insert span and append to field*

    let appenderSpan = document.createElement("span");

    appenderSpan.classList.add("modalWord");

    appenderSpan.textContent = arr[i];

    modalWords.appendChild(appenderSpan);

  }

}

*// Sort words by length function*

function sortWordsByLength(): string[] {

  let lengthArr: [string, number][] = [];

*// Define array with [word, word length]*

  for (let i = 0; i < words.length; i++) {

    let element = words[i];

    lengthArr[i] = [element, element.length];

  }

*// Insertion sort the array*

  let sortedArr = insertionSort2d(lengthArr);

*// Reverse array*

  sortedArr.reverse;

*// Redefine array as word only*

  let finalArr: string[] = [];

  for (let i = 0; i < words.length; i++) {

    finalArr[i] = sortedArr[i][0];

  }

  return finalArr;

}

*// Insertion sort*

function insertionSort2d(list: [string, number][]) {

*// Sorting algoritm*

  let max = list.length;

  for (let i = 1; i < max; i++) {

    let j = i;

*//While list[j - 1] is larger than list[j]*

    while (j > 0 && list[j - 1][1] > list[j][1]) {

*// Swap values*

      let buffer = list[j - 1];

      list[j - 1] = list[j];

      list[j] = buffer;

*// Decrement J*

      j--;

    }

  }

  return list;

}

*// Bubble sort*

function bubbleSort2d(list: [string, number][]) {

*// Swapped check*

  let swapped = true;

  let max = list.length;

  while (swapped === true) {

*// Preset swap check*

    swapped = false;

*// Repeat all items in list*

    for (let i = 1; i < max; i++) {

*// If list[i - 1] is larger than list[i]*

      if (list[i - 1][1] > list[i][1]) {

*// Swap items*

        let buffer = list[i - 1];

        list[i - 1] = list[i];

        list[i] = buffer;

*// Swapped condition to true*

        swapped = true;

      }

    }

  }

  return list;

}

*/\*===============*

*|    Classes    |*

*================\*/*

*// Scoreboard and any functions related to it*

class Scoreboard {

  \_scores: [string, number][];

  \_index: number;

  constructor() {

*// Users scores*

    this.\_scores = [];

*// localStorage index*

    this.\_index = 0;

  }

*// Getter methods*

  get strLen(): string {

    return this.\_index.toString();

  }

  get scores(): [string, number][] {

    return this.\_scores;

  }

*// Setter methods*

  set scores(value: [string, number][]) {

    this.\_scores = value;

  }

*// Other methods*

*// Initialise index with length - offset(stored settings)*

  initIndex(offset) {

    this.\_index = localStorage.length - offset;

  }

*// Add new score to scores array*

  addNewScore(value: [string, number]) {

    this.\_scores.push(value);

  }

*// Submit wpm and acc to a new localStorage value*

  submitToLocalStorage(name: string, wpm: number): void {

*// Define object*

    let stat = { name, wpm };

*// Set item with index, and string format of the object*

    localStorage.setItem(this.strLen, JSON.stringify(stat));

*// Increase index*

    this.\_index += 1;

  }

*// Parse an item from json string to dictionary*

  parseItem(index: number): [string, number] {

*// Get value from the index*

    let raw = localStorage.getItem(index.toString());

*// Get parsed value*

    let parsed = JSON.parse(raw);

*// Store parsed values into array*

    let score: any = Object.values(parsed);

*// Return array*

    return score;

  }

*// Initialise scoreboard DOM on startup*

  initScoreboard() {

    let arr = [];

*// Set array with parsed items of all localStorage values*

    let len = this.\_index;

    for (let i = 0; i < len; i++) {

*// Format: [string, number]*

      arr[i] = this.parseItem(i);

    }

*// Insertion sort the values*

    let sortedArr = insertionSort2d(arr);

*// Reverse values from high to low*

    sortedArr.reverse();

*// Set scores variable to the sorted array*

    this.\_scores = sortedArr;

*// Empty scoreboard table*

    scorebody.innerHTML = "";

*// Insert row for each value in array*

    for (let i = 0; i < sortedArr.length; i++) {

      this.insertRow(sortedArr[i]);

    }

  }

*// Update scoreboard when a test is done*

  updateScoreboard() {

*// Parse item of index - 1(zero-based)*

    let item = this.parseItem(this.\_index - 1);

*// Add new score with item*

    this.addNewScore(item);

*// Bubble sort the values (partially sorted)*

    let sortedArr = bubbleSort2d(this.\_scores);

*// Reverse the array*

    sortedArr.reverse();

*//Set scores variable to the sorted array*

    this.scores = sortedArr;

*// Empty scoreboard table*

    scorebody.innerHTML = "";

*// Insert row for each value in array*

    for (let i = 0; i < sortedArr.length; i++) {

      this.insertRow(sortedArr[i]);

    }

  }

*// Insert a row into the table*

  insertRow(array: [string, number]) {

*// Set new row in the table*

    let newRow = scorebody.insertRow(-1);

*// create new node for each column*

    for (let i = 0; i < 2; i++) {

*// InsertCell at i*

      let cell = newRow.insertCell(i);

*// Text element*

      let text = document.createTextNode(array[i].toString());

*// Append text element to the cell*

      cell.appendChild(text);

    }

  }

}

*// Any functions that require DOM manipulation*

class DOMManipulation {

  \_position: number;

  \_nodeList: NodeListOf<HTMLElement>;

  \_theme: string;

  constructor() {

*// How far a word is down a line*

    this.\_position = 0;

*// List of all words*

    this.\_nodeList = document.querySelectorAll(".typingWord");

*// User theme*

    this.\_theme = "dark";

  }

*// == Class getters==*

  get position(): number {

    return this.\_position;

  }

  get nodeList() {

    return this.\_nodeList;

  }

  get area() {

    return gameWordArea;

  }

  get theme() {

    return this.\_theme;

  }

*// ==Class Setters==*

  set position(value: number) {

    this.\_position = value;

  }

  set theme(value: string) {

    this.\_theme = value;

  }

*// ==Class Functions==*

*// updates the node list*

  updateNodeList(): void {

    this.\_nodeList = gameWordArea.querySelectorAll(".typingWord");

  }

*// Updates the area and puts in the specified word in the array*

*// Array gamewords, position i*

  updateWords(gameWords: string[], i: number): void {

    let appenderSpan = document.createElement("span");

    appenderSpan.classList.add("typingWord");

    appenderSpan.textContent = `${gameWords[i]} `;

    gameWordArea.appendChild(appenderSpan);

  }

*// Adds 1 to position*

  incrementPosition(): void {

    this.\_position += 1;

  }

*// Shows the words on screen and sets starting word as highlight*

  showArray(gameWords: string[]): void {

    let area = gameWordArea;

    area.innerHTML = "";

*// Creates new spans with text from gamewords[]*

*// Repeats 50 times for some overflow*

    for (let i = 0; i < 50; i++) {

*// Goes to updateWords with array gameWords, position i*

      this.updateWords(gameWords, i);

    }

*// Set first word with .typingword as the highlight word*

    let nodeItem = area.querySelector(".typingWord");

    nodeItem.id = "highlightWord";

    this.updateNodeList();

  }

*// Sets the highlight id - triggers on spacebar pressed*

  highlightCurrentWord(): void {

*// Setting local variables for each item needed*

    let position = this.\_position;

*// List of words*

    let nodeList = this.\_nodeList;

*// Word just typed*

    let nodeItem = nodeList.item(position);

*// Add classes and IDs to each of the items*

    nodeItem.id = "highlightWord";

*// Add id for the previous word, and remove the id from the second last word*

*// If statement so the first position doesn't return an error*

    if (position > 0) {

*// Last word typed*

      let previousItem = nodeList.item(position - 1);

      previousItem.id = "previousWord";

      previousItem.classList.add("completedWord");

*// Second last word typed - If for same reason as above*

      if (position > 1) {

        let backItem = nodeList.item(position - 2);

        backItem.removeAttribute("id");

      }

    }

  }

*// Bool in from wordCheck() and changes class if the word was right or wrong*

  setAnswer(checkedWord: boolean, nodeItem: HTMLElement): void {

    if (checkedWord) {

      nodeItem.classList.add("correctWord");

    } else {

      nodeItem.classList.add("wrongWord");

    }

  }

*// Check if the word is on the second row, and delete the first row if it is*

  checkRow() {

    let nodeList = this.nodeList;

    let item = nodeList.item(this.position);

    let offset = item.offsetTop;

*/\* Checks if the y coordinate of the span relative to the div is more than 107(next row) and deletes the row \*/*

    if (offset > item.offsetHeight) {

      this.deleteRow(this.position);

*//Set the position back to 0*

      this.position = 0;

    }

  }

*// Deletes a row - called from wordCheck*

  deleteRow(position: number): void {

    let nodeList = this.\_nodeList;

*// Remove each span less than the position*

    for (let i = 0; i < position; i++) {

      let selectedSpan = nodeList.item(i);

      selectedSpan.remove();

    }

  }

*// Display stats to finish screen spans*

  displayStats(): void {

    let wpm = Game.calculatedStats[0];

    let time = Game.calculatedStats[1];

    gameWPM.textContent = wpm.toString();

    gameAccuracy.textContent = time.toString();

  }

*// Sets the timer/word countdown to value*

  changeGameProgress(value: any): void {

*// If type is number, convert to string*

    if (typeof value === "number") {

      value = value.toString();

    }

*// Set game progress span text content to the value*

    gameProgress.textContent = value;

  }

*// Show starting display*

  showStart(): void {

    startOverlay.classList.remove("hide");

  }

*// Show main display*

  showBackdrop(): void {

    startOverlay.classList.add("hide");

  }

*// Show finish screen*

  showFinish(): void {

    finishOverlay.style.display = "inline";

    this.displayStats();

  }

*// Hide finish screen*

  hideFinish(): void {

    finishOverlay.style.display = "none";

  }

*// Set hidden field values to WPM and Accuracy*

  appendFormData(): void {

    hiddenWPMInput.value = Game.\_calculatedStats[0].toString();

    hiddenAccInput.value = Game.\_calculatedStats[1].toString();

  }

*// Set the settings into variables in localStorage*

  setSettings(type, length): void {

    localStorage.setItem("type", type);

    localStorage.setItem("length", length);

    localStorage.setItem("theme", this.\_theme);

  }

*// Get the settings from localStorage and return an array with them*

  getSettings(): string[] {

    let type = localStorage.getItem("type");

    let length = localStorage.getItem("length");

    let theme = localStorage.getItem("theme");

    return [type, length, theme];

  }

}

**words.ts**



**styles.scss**

*// Import themes and variables (partials folder)*

@use "partials/variables" as \*;

@use "partials/themes" as \*;

$menuSize: 25rem;

*/\*==============*

*|    Mixins    |*

*==============\*/*

*// Display flex and center item*

@mixin center\_flex\_element {

  display: flex;

  justify-content: center;

  align-items: center;

}

*// Center text*

@mixin center\_text {

  text-align: center;

  vertical-align: middle;

}

*// Set width and height to 100%*

@mixin full\_screen {

  @include width\_height(100%);

}

*// Set width and height to values.*

*//If height not specified, set both to same value*

@mixin width\_height($width, $height: $width) {

  width: $width;

  height: $height;

}

*// Style for container*

@mixin container\_style {

  border: var(--containerBorder);

  border-radius: 7px;

  background-color: var(--gameBackground);

}

*// Set style for border of buttons*

@mixin border\_style {

  border: var(--buttonBorder);

}

*// Hover effect for button*

@mixin button\_hover {

  background-color: var(--buttonHover);

  color: var(--hoverCol);

  border: 1px solid var(--hoverCol);

}

*// Styling of buttons*

@mixin button\_style {

  @include border\_style;

  background-color: var(--focusBackground);

  color: var(--smallText);

  border-radius: 5px;

  transition: all 0.3s;

*// On hover*

  &:hover {

    @include button\_hover;

  }

}

*// Create a half transparent overlay across whole screen*

@mixin blocking\_overlay {

  background-color: rgba(0, 0, 0, 0.5);

  display: none;

  position: fixed;

  @include full\_screen;

}

*// Set item to top right corner*

@mixin top\_right {

  top: 0px;

  right: 0px;

}

*/\*===============*

*|    Colours    |*

*===============\*/*

*// Set transition of all*

\* {

  transition: background-color 0.5s, color 1s;

}

*// General body elements*

body {

*// Game container*

  #screenContainer {

    background-color: var(--backgroundColor);

  }

*// Modal content*

  #modalContent {

    background-color: var(--navBackground);

  }

*// Tooltip for nav buttons*

  .navButton {

    .tooltip {

      background-color: var(--mediumText);

      color: var(--gameBackground);

    }

  }

*// Title and subtitle*

  h1,

  h2 {

    color: var(--highlightText);

  }

*// Text and labels*

  p,

  label {

    color: var(--mediumText);

  }

*// Navbar*

  nav {

    background-color: var(--navBackground);

  }

*// Input typing boxes*

  input {

    background-color: var(--focusBackground);

  }

}

*// Game Container*

#gameContainer {

*// Main game container*

  #game {

    @include container\_style;

*// all of the game grid separator elements*

    .gameArea {

      background-color: var(--gameBackground);

    }

*// Stat box*

    #statBox {

      background-color: var(--focusBackground);

      color: var(--mediumText);

    }

*// Game word area*

    #gameWordArea {

*// Any word spans inside it*

      .typingWord {

        font-size: 1.5rem;

        text-overflow: clip;

        color: var(--typingWord);

        transition: color 0.2s;

      }

*// Current word*

      #highlightWord {

        color: var(--highlightWord);

      }

*// Correct words*

      .correctWord {

        color: var(--correctWord);

      }

*// Wrong words*

      .wrongWord {

        color: var(--wrongWord);

      }

    }

*// Game typing area*

    #gameTypingArea {

*// User typing field*

      #gameTypingField {

        color: var(--highlightText);

      }

    }

  }

*// Start overlay container*

  #startOverlay {

*// 'press to start' button*

    #startButton {

      @include button\_style;

    }

  }

*// Finish overlay container*

  #finishOverlay {

    background-color: var(--gameBackground);

    @include container\_style;

*// Name input area*

    #finishNameArea {

*// Name input field*

      #finishTypingField {

        color: var(--highlightText);

      }

    }

*// WPM and accuracy display*

    #finishStatArea {

*// Fields inside it*

      .finishStat {

        color: var(--mediumText);

        font-size: 1.25rem;

      }

    }

  }

}

*// Settings form*

#settingsForm {

*// Radio button sections*

  .settingsRadio {

*// Radio button label*

    .radioSelector {

      @include button\_style;

    }

*// Radio button*

    input {

*// Label when checked*

      &:checked + label {

        @include button\_hover;

      }

    }

  }

*// Other buttons in the settings*

  .settingsMenuButton {

    @include button\_style;

  }

}

*// Scoreboard nav bar*

#scoresNav {

*// Scoreboard table*

  #scoreboard {

    background-color: var(--focusBackground);

*// Header element*

    .scoreHeader {

      color: var(--mediumText);

    }

*// Row elements*

    #scorebody > tr {

      color: var(--smallText);

*// Odd element*

      &:nth-child(odd) {

        background-color: var(--scoreboard1);

      }

*// Even element*

      &:nth-child(even) {

        background-color: var(--scoreboard2);

      }

    }

  }

}

*// Word modal*

#wordModal {

*// Word modal contents*

  #modalContent {

*// Word model words field*

    #modalWords {

      background-color: var(--focusBackground);

*// Actual word spans*

      .modalWord {

        color: var(--mediumText);

      }

    }

  }

}

*/\*=============*

*|    Fonts    |*

*=============\*/*

\* {

  font-family: Consolas;

  font-size: 16px;

}

body {

  h1 {

    font-size: 2rem;

  }

  h2 {

    font-size: 1.75rem;

  }

  .radioHeader {

    font-size: 1.5rem;

    font-weight: 100;

  }

  .radioSelector {

    font-size: 1.25rem;

  }

}

*/\*============================*

*|    General body styling    |*

*============================\*/*

*// Set html to full height*

html {

  height: 100%;

}

*// Body*

body {

  height: 100%;

  margin: 0;

  padding: 0;

  position: static;

*// Word modal*

  #wordModal {

    @include blocking\_overlay;

    overflow: auto;

    justify-content: center;

  }

*// Settings overlay*

  #overlay {

    @include blocking\_overlay;

  }

*// Main body elements*

  main {

    @include full\_screen;

    position: relative;

    display: flex;

    flex-direction: column;

*// Main body text*

    #pageText {

      display: flex;

      flex-direction: column;

      justify-content: space-between;

      height: 100%;

*// Text at the top*

      #topText {

        h1 {

          margin-top: 1rem;

        }

      }

*// Footer*

      #bottomText {

        position: relative;

        background-color: var(--gameBackground);

        display: flex;

        flex-direction: row;

        justify-content: space-evenly;

        z-index: 2;

*// P element inside*

        p {

          color: var(--mediumText);

*// Link inside p*

          & > a {

            color: var(--hoverCol);

*// On visited*

            &:visited {

              color: var(--selectedCol);

            }

          }

        }

      }

    }

*// Container containing game*

    #screenContainer {

      @include full\_screen;

      @include center\_flex\_element;

      flex: 1 1 0;

      position: absolute;

    }

  }

}

*/\*=============*

*|    Main     |*

*=============\*/*

main {

  h1 {

    text-align: center;

  }

  p {

    text-align: center;

  }

}

*/\*=======================*

*|    Sidebar buttons    |*

*=======================\*/*

main {

*// Buttons to activate navbar*

  .navButton {

    @include button\_style;

    @include width\_height(50px);

    position: absolute;

    margin: 1rem;

    padding: 0;

*// On focus*

    &:focus {

      outline: none;

    }

*// Set size of icon*

    .navIcon {

      height: 80%;

      margin: 10%;

    }

*// Tooltip*

    .tooltip {

      @include width\_height(120px, 50%);

      margin: 20% 5px auto;

      visibility: hidden;

      text-align: center;

      padding: 5% 0;

      border-radius: 5px;

      position: absolute;

      z-index: 10;

      opacity: 0;

      transition: opacity 0.3s;

    }

*// Set scores tooltip to the right*

    #scoreTooltip {

      right: 105%;

    }

*// Tooltip visible when hovered*

    &:hover .tooltip {

      visibility: visible;

      opacity: 1;

    }

  }

*// Put scores button to top right*

  #scoresButton {

    @include top\_right;

  }

*// Rotate transition for settings button*

  #settingsButton {

    img {

      transition: transform 0.3s;

    }

    &:hover img {

      transform: rotate(45deg);

    }

  }

}

*/\*===============*

*|    Sidebar    |*

*===============\*/*

body {

*// Navbars*

  .navbar {

    @include width\_height(0, 100%);

    display: flex;

    flex-direction: column;

    position: fixed;

    overflow: hidden;

    opacity: 0;

    transition: width 0.3s, opacity 0.15s;

  }

*// Put scores nav in top right*

  #scoresNav {

    @include top\_right;

  }

*// With class open*

  .open {

*// Margin left of button*

    &#settingsButton {

      margin-left: calc(#{$menuSize} + 15px);

    }

*// Width of navbar*

    &#settingsNav {

      width: $menuSize;

      opacity: 1;

    }

*// Margin right of button*

    &#scoresButton {

      margin-right: calc(#{$menuSize} + 15px);

    }

*// Width of button*

    &#scoresNav {

      width: $menuSize;

      opacity: 1;

    }

*// Overlay visibility*

    &#overlay {

      display: block;

    }

  }

}

*/\*=====================*

*|    Settings form    |*

*=====================\*/*

#settingsForm {

  display: flex;

  flex-direction: column;

  height: 100%;

*// Settings header*

  #settingsHeader {

    padding-left: 15px;

  }

*// Header for radio buttons*

  .radioHeader {

    text-align: center;

  }

*// Radio buttons group*

  .settingsRadio {

    display: flex;

    flex-direction: row;

    justify-content: space-evenly;

*// Radio button input invisible*

    input {

      position: fixed;

      opacity: 0;

      pointer-events: none;

    }

*// Radio button label*

    .radioSelector {

      @include center\_text;

      @include width\_height(7rem, 2rem);

      margin: 5px;

    }

*// lable of disabled button*

    .lengthRadio:disabled + label {

      opacity: 0.5;

      pointer-events: none;

    }

  }

*// Other buttons in menu*

  .settingsMenuButton {

    @include width\_height(15rem, 3rem);

    margin: 10px;

    font-size: 1.25rem;

    align-self: center;

  }

*// Theme buttons at bottom*

  #themeButtons {

    margin-top: auto;

    display: flex;

    flex-direction: row;

    justify-content: space-evenly;

  }

}

*/\*========================*

*|    View words modal    |*

*========================\*/*

#wordModal {

  @include center\_flex\_element();

*// Modal content*

  #modalContent {

    border-radius: 10px;

    padding: 20px;

    position: absolute;

    @include border\_style;

    @include width\_height(80%, auto);

*// Animation come down*

    animation-name: animatetop;

    animation-duration: 0.4s;

*// Sections in modal*

    .modalSection {

      padding: 15px;

    }

*// Header section*

    #modalHeader {

      display: flex;

      flex-direction: row;

      text-align: center;

*// Exit button*

      #modalExit {

        margin-left: auto;

        font-size: 2rem;

        height: 2rem;

        width: 2rem;

        border-radius: 50%;

        color: var(--mediumText);

        text-align: center;

        vertical-align: middle;

        line-height: 2rem;

*// On hover*

        &:hover {

          background-color: var(--focusBackground);

          color: var(--highlightText);

        }

      }

    }

*// Buttons section*

    #modalButtons {

      display: flex;

      justify-content: space-evenly;

*// Modal buttons*

      .modalButton {

        @include button\_style;

        @include width\_height(15rem, 3rem);

      }

    }

*// Words section*

    #modalWords {

      display: flex;

      flex-wrap: wrap;

      border-radius: 5px;

*// Modal word*

      .modalWord {

        padding: 2px;

      }

    }

  }

}

*// Animate top*

@keyframes animatetop {

  from {

    opacity: 0;

  }

  to {

    opacity: 1;

  }

}

*/\*==================*

*|    Scoreboard    |*

*==================\*/*

#scoresNav {

  align-items: center;

*// Scoreboard table*

  #scoreboard {

    width: 90%;

    table-layout: fixed;

*// Scoreboard body*

    #scorebody {

*// Element in body*

      tr {

        text-align: center;

        width: 50%;

      }

    }

  }

}

*/\*==================*

*|    Z-indexing    |*

*==================\*/*

html {

*// Main elements above normal*

  #gameContainer,

  #topText {

    z-index: 2;

  }

*// Overlay above main elements*

  #overlay {

    z-index: 3;

  }

*// Navbar above overlay*

  .navbar,

  .navButton {

    z-index: 4;

  }

*// Modal above navbar*

  #wordModal {

    z-index: 5;

  }

}

*/\*===================*

*|    Game window    |*

*===================\*/*

#gameContainer {

  @include width\_height(39rem, 18rem);

  position: absolute;

*// Game grid making*

  #game {

*// lh = line height including bottom e.g y*

    $lh: 7 / 6;

*// Text size = 4.5 (3 columns) \* line height + 5/6(no idea lol)*

    $text: 4.5rem \* $lh + 5/6;

*// Regular grids to 2.5rem*

    $grids: 2.5rem;

    @include full\_screen;

    display: grid;

    grid-template-rows: $grids $text $grids $grids;

    gap: 0.75rem;

    padding: 0.75rem;

    box-sizing: border-box;

  }

}

*// Game*

#game {

*// Stats area*

  #gameStatsArea {

    @include center\_flex\_element;

*// Box inside stats area*

    #statBox {

      @include width\_height(2.5rem, 100%);

      display: flex;

      border-radius: 50%;

      justify-content: center;

*// Span inside box*

      #gameProgress {

        line-height: 2.5rem;

        font-size: 1.5rem;

      }

    }

  }

*// Game words*

  #gameWordArea {

    position: relative;

    overflow: hidden;

    padding: 0.5rem;

    padding-bottom: 0.5rem + 1rem \* 1/6;

  }

*// Game typing area*

  #gameTypingArea {

    display: block;

*// Typing field*

    #gameTypingField {

      @include width\_height(100%, calc(100% - 5px));

      border: 0;

      margin: 0;

      padding: 0;

      border-radius: 5px;

      outline: none;

      padding-left: 5px;

      font-size: 1.5rem;

    }

  }

*// Reset area*

  #gameResetArea {

*//Reset button*

    #gameResetButton {

      @include button\_style;

      @include width\_height(5rem, 80%);

    }

  }

}

*/\*============================*

*|    Start/finish overlay    |*

*============================\*/*

#gameContainer {

*// Overlays*

  .gameOverlay {

    @include full\_screen;

    @include top\_right;

    position: absolute;

  }

*// Start overlay*

  #startOverlay {

    @include center\_flex\_element;

    background-color: rgba(0, 0, 0, 0.3);

*// Blur backdrop doesn't work on firefox*

    backdrop-filter: blur(3px);

    opacity: 1;

    height: 100%;

*// On hidden*

    &.hide {

      backdrop-filter: blur(0);

      opacity: 0;

      height: 0;

*// Transition doesn't really work but whatever*

      transition: backdrop-filter 0.3s, opacity 0.03s, height 0s 0.3s;

    }

*// Start button*

    #startButton {

      @include width\_height(10rem, 5rem);

      @include center\_text;

    }

  }

*// Finish overlay*

  #finishOverlay {

    display: none;

*// finish "form"*

    form {

      display: grid;

      grid-template-rows: 20% 15% auto 20%;

      height: 100%;

*// Title of form*

      #finishTitleArea {

        @include center\_flex\_element;

      }

*// Name input area*

      #finishNameArea {

        @include center\_flex\_element;

*// Typing field*

        #finishTypingField {

          margin: 0;

          height: 1.5rem;

          outline: 0;

          @include border\_style;

          border-radius: 5px;

          text-align: center;

        }

      }

*// Stats area*

      #finishStatArea {

        display: grid;

        padding: 0 10%;

        grid-template-columns: 50% 50%;

*// Title area smaller than actual number*

        grid-template-rows: 30% 70%;

*// Grid items*

        .finishStat {

          justify-self: center;

          align-self: center;

        }

      }

*// Hidden forms*

      #finishHiddenForms {

        display: none;

      }

*// Retry/Exit buttons*

      #finishRetryArea {

        display: flex;

        justify-content: space-evenly;

        align-items: center;

*// Button*

        .finishButton {

          @include button\_style;

          @include width\_height(5rem, 2rem);

        }

      }

    }

  }

}

**\_colors.scss**



**\_themes.scss**

*// Import colours*

@use "colors" as \*;

*//https://www.nordtheme.com/docs/colors-and-palettes*

*// Light mode*

.light {

*// Text*

  --highlightText: #{$nord0};

  --mediumText: #{$nord1};

  --smallText: #{$nord2};

*// Background colours*

  --backgroundColor: #{$nord4};

  --navBackground: #{$nord5};

  --gameBackground: #{$nord6};

  --focusBackground: #{$polarbg};

*// Button colours*

  --hoverCol: #{$nord8};

  --selectedCol: #{$nord-10};

  --borderOutline: #{$nord1};

  --buttonHover: #{$nord6};

*// Container colours*

  --containerBorder: 2px solid var(--borderOutline);

  --buttonBorder: 1px solid var(--borderOutline);

*// Scoreboard odd/even*

  --scoreboard1: #{$nord4};

  --scoreboard2: #{$polarbg};

*// Word game colours*

  --typingWord: #{$nord1};

  --highlightWord: #{$nord8};

  --correctWord: #{$nord-14};

  --wrongWord: #{$nord-11};

}

*// Dark mode*

.dark {

*// Text*

  --highlightText: #{$nord6};

  --mediumText: #{$nord5};

  --smallText: #{rgba($nord4, 0.87)};

*// Background colours*

  --backgroundColor: #{$nord0};

  --gameBackground: #{$nord1};

  --navBackground: #{$nord2};

  --focusBackground: #{$nord3};

*// Button colours*

  --hoverCol: #{$nord8};

  --selectedCol: #{$nord-10};

  --borderOutline: #{rgba($nord4, 0.87)};

  --buttonHover: #{$nord2};

*// Container colours*

  --containerBorder: 2px solid var(--borderOutline);

  --buttonBorder: 1px solid var(--borderOutline);

*// Scoreboard odd/even*

  --scoreboard1: #{$nord1};

  --scoreboard2: #{$nord3};

*// Word game colours*

  --typingWord: #{$nord4};

  --highlightWord: #{$nord8};

  --correctWord: #{$nord-14};

  --wrongWord: #{$nord-11};

}

**Game.min.js**

class GameSettings{constructor(type,length,words){this.\_name="",this.\_length=length,this.\_type=type,this.\_words=words}get name(){return this.\_name}get length(){return this.\_length}get type(){return this.\_type}set length(value){this.\_length=value}set type(value){this.\_type=value}getCalculatedLength(){let output,part1=this.\_type.toString(),part2=this.\_length.toString(),switcher;switch(part1.concat(part2)){case"00":output=30;break;case"01":output=60;break;case"02":output=120;break;case"10":output=25;break;case"11":output=50;break;case"12":output=100}return output}setName(){let textbox,name=document.querySelector("#finishTypingField").value;""===name&&(name="Anon"),this.\_name=name}}class UserGame extends GameSettings{constructor(...args){super(...args),this.\_calculatedStats=[0,0],this.\_characters=0,this.\_gameWords=[],this.\_timeTaken=0,this.\_userWordCount=0,this.\_wordErrors=0}get characters(){return this.\_characters}get calculatedStats(){return this.\_calculatedStats}get gameWords(){return this.\_gameWords}get timeTaken(){return this.\_timeTaken}get userWordCount(){return this.\_userWordCount}get word(){return this.\_gameWords[this.\_userWordCount-1]}set calculatedStats(value){this.\_calculatedStats=value}set timeTaken(value){this.\_timeTaken=value}incrementCharacters(char){const options=[8,16,17,20,46];-1===options.indexOf(char)&&(this.\_characters+=1)}incrementWordCount(){this.\_userWordCount+=1}incrementWordErrors(){this.\_wordErrors+=1}editGameData(x,y){let type=parseInt(x,10),length=parseInt(y,10);this.\_type=type,this.\_length=length}initialiseArray(){let tempGameWords=[];for(let i=0;i<50;i++){let randint=Math.floor(Math.random()\*words.length);tempGameWords[i]=this.\_words[randint]}this.\_gameWords=tempGameWords}wordCheck(){let inputWord=gameTypingField.value.trim(),wordComparison=this.word,nodeItem=document.getElementById("previousWord");inputWord===wordComparison?DOMFunctions.setAnswer(!0,nodeItem):(this.incrementWordErrors(),DOMFunctions.setAnswer(!1,nodeItem)),DOMFunctions.checkRow()}newWord(){let gameWords=this.\_gameWords,randint=Math.floor(Math.random()\*words.length);gameWords.push(words[randint]),DOMFunctions.updateWords(this.\_gameWords,this.\_gameWords.length-1)}calculateStats(){let chars=this.\_characters,time=this.\_timeTaken,errors=this.\_wordErrors,totalWords=this.\_userWordCount,netWords=chars/5-errors,timeFactor=time/60,netWPM,accuracy,tempStats,stats=[];[netWords<0?(totalWords-errors)/timeFactor:netWords/timeFactor,(totalWords-errors)/totalWords\*100].forEach(element=>stats.push(Math.round(element))),this.\_calculatedStats=stats}resetStats(){this.\_calculatedStats=[0,0],this.\_characters=0,this.\_gameWords=[""],this.\_timeTaken=0,this.\_userWordCount=0,this.\_wordErrors=0,DOMFunctions.position=0}}class GameFunctions extends UserGame{constructor(...args){super(...args)}startGame(){inGame=!0;let gameType=0===this.\_type?this.goToTimedGame():this.goToWordGame()}goToTimedGame(){DOMFunctions.changeGameProgress(this.getCalculatedLength()),this.timeTimer(()=>{this.finishGame()})}timeTimer(callback){let duration=this.getCalculatedLength(),time=1,gameTimer=setInterval(()=>{!1!==inGame?(time>=duration&&(clearInterval(gameTimer),this.\_timeTaken=duration,callback()),DOMFunctions.changeGameProgress(duration-time),time++):clearInterval(gameTimer)},1e3)}goToWordGame(){let gameLength=this.getCalculatedLength();DOMFunctions.changeGameProgress(gameLength),this.wordTimer(()=>{this.finishGame()})}wordTimer(callback){let totalWordCount=this.getCalculatedLength(),time=1,inGameSeconds=0,interval=100,gameTimer=setInterval(()=>{!1!==inGame?(this.\_userWordCount>=totalWordCount&&(clearInterval(gameTimer),this.\_timeTaken=time,callback()),inGameSeconds+=100,inGameSeconds%1e3==0&&time++):clearInterval(gameTimer)},100)}finishGame(){inGame=!1,this.calculateStats(),DOMFunctions.showFinish(),DOMFunctions.appendFormData()}goToNextWord(){DOMFunctions.updateNodeList(),this.incrementWordCount(),this.newWord(),DOMFunctions.incrementPosition(),DOMFunctions.highlightCurrentWord(),this.wordCheck(),gameTypingField.value="",1===this.\_type&&DOMFunctions.changeGameProgress(this.getCalculatedLength()-this.\_userWordCount)}}function newGame(that){let type=that.test\_type.value,length="0"===type?that.time\_length.value:that.word\_length.value;""===type&&(type=defOpt[0].toString()),""===length&&(length=defOpt[1].toString()),Game.editGameData(type,length),DOMFunctions.setSettings(type,length),resetGame()}function resetGame(){Game.resetStats,inGame=!1,clicked=!1,gameTypingField.value="",DOMFunctions.changeGameProgress(""),Game.initialiseArray(),DOMFunctions.showArray(Game.gameWords)}function finishedReset(exit){Game.setName(),DOMFunctions.hideFinish(),Scores.submitToLocalStorage(Game.name,Game.calculatedStats[0]),Scores.updateScoreboard(),resetGame(),!0===exit&&DOMFunctions.showStart()}gameTypingField.onclick=()=>{!1===inGame&&(clicked=!0)},gameTypingField.onkeydown=e=>{!0===clicked&&(Game.startGame(),clicked=!1),!0===inGame&&(Game.incrementCharacters(e.keyCode),32==e.keyCode&&Game.goToNextWord())};let inGame=!1,clicked=!1,defOpt=[0,1],Game=new GameFunctions(defOpt[0],defOpt[1],words),DOMFunctions=new DOMManipulation,Scores=new Scoreboard;window.onload=e=>{if(0===localStorage.length)DOMFunctions.setSettings(defOpt[0].toString(),defOpt[1].toString());else if(localStorage.length>0){let importSettings=DOMFunctions.getSettings();Game.editGameData(importSettings[0],importSettings[1]),setTheme(importSettings[2]),localStorage.length>3&&(Scores.initIndex(3),Scores.initScoreboard())}gameTypingField.value="",Game.initialiseArray(),DOMFunctions.showArray(Game.gameWords)};

**main.min.js**

const body=document.querySelector("body"),overlay=document.getElementById("overlay"),settingsButton=document.getElementById("settingsButton"),scoresButton=document.getElementById("scoresButton"),game=document.getElementById("game"),gameProgress=game.querySelector("#gameProgress"),gameWordArea=game.querySelector("#gameWordArea"),gameTypingField=game.querySelector("#gameTypingField"),startOverlay=document.getElementById("startOverlay"),finishOverlay=document.getElementById("finishOverlay"),finishForm=finishOverlay.querySelector("#finishForm"),nameInput=finishForm.querySelector("#finishTypingField"),gameWPM=finishForm.querySelector("#gameWPM"),gameAccuracy=finishForm.querySelector("#gameAccuracy"),hiddenWPMInput=finishForm.querySelector("#hiddenWpm"),hiddenAccInput=finishForm.querySelector("#hiddenAcc"),settingsNav=document.getElementById("settingsNav"),wordSelector=settingsNav.querySelector("#wordSelector"),timeSelector=settingsNav.querySelector("#timeSelector"),scoresNav=document.getElementById("scoresNav"),scoreboard=scoresNav.querySelector("#scoreboard"),scorebody=scoreboard.querySelector("#scorebody"),modal=document.getElementById("wordModal"),modalWords=modal.querySelector("#modalWords");let openToggle=!1;function sidebarDOM(button,nav,button2,state){let elementList=[button,nav,overlay];"open"===state?(elementList.forEach(element=>{element.classList.add("open")}),button2.style.zIndex="2"):(elementList.forEach(element=>{element.classList.remove("open")}),button2.style.zIndex="4")}function setTheme(theme){switch(theme){case"light":localStorage.setItem("theme","light"),body.classList.replace("dark","light"),DOMFunctions.theme="light";break;case"dark":body.classList.replace("light","dark"),DOMFunctions.theme="dark",localStorage.setItem("theme","dark")}}function chooseTimeWords(a,b){let group;if(a.classList.add("deselected"),a.querySelectorAll(".lengthRadio").forEach(element=>{element.disabled=!0,element.checked=!1}),b.classList.contains("deselected")){let group;b.classList.remove("deselected"),b.querySelectorAll(".lengthRadio").forEach(element=>{element.disabled=!1})}}function writeWords(arr){modalWords.innerHTML="";for(let i=0;i<arr.length;i++){let appenderSpan=document.createElement("span");appenderSpan.classList.add("modalWord"),appenderSpan.textContent=arr[i],modalWords.appendChild(appenderSpan)}}function sortWordsByLength(){let lengthArr=[];for(let i=0;i<words.length;i++){let element=words[i];lengthArr[i]=[element,element.length]}let sortedArr=insertionSort2d(lengthArr);sortedArr.reverse;let finalArr=[];for(let i=0;i<words.length;i++)finalArr[i]=sortedArr[i][0];return finalArr}function insertionSort2d(list){let max=list.length;for(let i=1;i<max;i++){let j=i;for(;j>0&&list[j-1][1]>list[j][1];){let buffer=list[j-1];list[j-1]=list[j],list[j]=buffer,j--}}return list}function bubbleSort2d(list){let swapped=!0,max=list.length;for(;!0===swapped;){swapped=!1;for(let i=1;i<max;i++)if(list[i-1][1]>list[i][1]){let buffer=list[i-1];list[i-1]=list[i],list[i]=buffer,swapped=!0}}return list}settingsButton.onclick=()=>{!1===openToggle?(sidebarDOM(settingsButton,settingsNav,scoresButton,"open"),openToggle=!0):(sidebarDOM(settingsButton,settingsNav,scoresButton,"close"),openToggle=!1)},scoresButton.onclick=()=>{!1===openToggle?(sidebarDOM(scoresButton,scoresNav,settingsButton,"open"),openToggle=!0):(sidebarDOM(scoresButton,scoresNav,settingsButton,"close"),openToggle=!1)},document.getElementById("lightButton").onclick=()=>{setTheme("light")},document.getElementById("darkButton").onclick=()=>{setTheme("dark")},document.getElementById("startButton").onclick=()=>{DOMFunctions.showBackdrop()},document.getElementById("openModal").onclick=()=>{modal.style.display="flex"},document.getElementById("radiotime").onclick=()=>{chooseTimeWords(wordSelector,timeSelector)},document.getElementById("radiowords").onclick=()=>{chooseTimeWords(timeSelector,wordSelector)},document.getElementById("modalExit").onclick=()=>{modal.style.display="none"},document.getElementById("freqSort").onclick=()=>{let arr;writeWords(words)},document.getElementById("lengthSort").onclick=()=>{let arr;writeWords(sortWordsByLength())},document.getElementById("alphaSort").onclick=()=>{let wordDummy,arr;writeWords([...words].sort((a,b)=>a.toLowerCase().localeCompare(b.toLowerCase())))};class Scoreboard{constructor(){this.\_scores=[],this.\_index=0}get strLen(){return this.\_index.toString()}get scores(){return this.\_scores}set scores(value){this.\_scores=value}initIndex(offset){this.\_index=localStorage.length-offset}addNewScore(value){this.\_scores.push(value)}submitToLocalStorage(name,wpm){let stat={name:name,wpm:wpm};localStorage.setItem(this.strLen,JSON.stringify(stat)),this.\_index+=1}parseItem(index){let raw=localStorage.getItem(index.toString()),parsed=JSON.parse(raw),score;return Object.values(parsed)}initScoreboard(){let arr=[],len=this.\_index;for(let i=0;i<len;i++)arr[i]=this.parseItem(i);let sortedArr=insertionSort2d(arr);sortedArr.reverse(),this.\_scores=sortedArr,scorebody.innerHTML="";for(let i=0;i<sortedArr.length;i++)this.insertRow(sortedArr[i])}updateScoreboard(){let item=this.parseItem(this.\_index-1);this.addNewScore(item);let sortedArr=bubbleSort2d(this.\_scores);sortedArr.reverse(),this.scores=sortedArr,scorebody.innerHTML="";for(let i=0;i<sortedArr.length;i++)this.insertRow(sortedArr[i])}insertRow(array){let newRow=scorebody.insertRow(-1);for(let i=0;i<2;i++){let cell=newRow.insertCell(i),text=document.createTextNode(array[i].toString());cell.appendChild(text)}}}class DOMManipulation{constructor(){this.\_position=0,this.\_nodeList=document.querySelectorAll(".typingWord"),this.\_theme="dark"}get position(){return this.\_position}get nodeList(){return this.\_nodeList}get area(){return gameWordArea}get theme(){return this.\_theme}set position(value){this.\_position=value}set theme(value){this.\_theme=value}updateNodeList(){this.\_nodeList=gameWordArea.querySelectorAll(".typingWord")}updateWords(gameWords,i){let appenderSpan=document.createElement("span");appenderSpan.classList.add("typingWord"),appenderSpan.textContent=`${gameWords[i]} `,gameWordArea.appendChild(appenderSpan)}incrementPosition(){this.\_position+=1}showArray(gameWords){let area=gameWordArea,nodeItem;area.innerHTML="";for(let i=0;i<50;i++)this.updateWords(gameWords,i);area.querySelector(".typingWord").id="highlightWord",this.updateNodeList()}highlightCurrentWord(){let position=this.\_position,nodeList=this.\_nodeList,nodeItem;if(nodeList.item(position).id="highlightWord",position>0){let previousItem=nodeList.item(position-1);if(previousItem.id="previousWord",previousItem.classList.add("completedWord"),position>1){let backItem;nodeList.item(position-2).removeAttribute("id")}}}setAnswer(checkedWord,nodeItem){checkedWord?nodeItem.classList.add("correctWord"):nodeItem.classList.add("wrongWord")}checkRow(){let nodeList,item=this.nodeList.item(this.position),offset;item.offsetTop>item.offsetHeight&&(this.deleteRow(this.position),this.position=0)}deleteRow(position){let nodeList=this.\_nodeList;for(let i=0;i<position;i++){let selectedSpan;nodeList.item(i).remove()}}displayStats(){let wpm=Game.calculatedStats[0],time=Game.calculatedStats[1];gameWPM.textContent=wpm.toString(),gameAccuracy.textContent=time.toString()}changeGameProgress(value){"number"==typeof value&&(value=value.toString()),gameProgress.textContent=value}showStart(){startOverlay.classList.remove("hide")}showBackdrop(){startOverlay.classList.add("hide")}showFinish(){finishOverlay.style.display="inline",this.displayStats()}hideFinish(){finishOverlay.style.display="none"}appendFormData(){hiddenWPMInput.value=Game.\_calculatedStats[0].toString(),hiddenAccInput.value=Game.\_calculatedStats[1].toString()}setSettings(type,length){localStorage.setItem("type",type),localStorage.setItem("length",length),localStorage.setItem("theme",this.\_theme)}getSettings(){let type,length,theme;return[localStorage.getItem("type"),localStorage.getItem("length"),localStorage.getItem("theme")]}}

**styles.min.css**

.light{--highlightText:#2E3440;--mediumText:#3B4252;--smallText:#434C5E;--backgroundColor:#D8DEE9;--navBackground:#E5E9F0;--gameBackground:#ECEFF4;--focusBackground:#f9fafb;--hoverCol:#88C0D0;--selectedCol:#5E81AC;--borderOutline:#3B4252;--buttonHover:#ECEFF4;--containerBorder:2px solid var(--borderOutline);--buttonBorder:1px solid var(--borderOutline);--scoreboard1:#D8DEE9;--scoreboard2:#f9fafb;--typingWord:#3B4252;--highlightWord:#88C0D0;--correctWord:#A3BE8C;--wrongWord:#BF616A}.dark{--highlightText:#ECEFF4;--mediumText:#E5E9F0;--smallText:rgba(216, 222, 233, 0.87);--backgroundColor:#2E3440;--gameBackground:#3B4252;--navBackground:#434C5E;--focusBackground:#4C566A;--hoverCol:#88C0D0;--selectedCol:#5E81AC;--borderOutline:rgba(216, 222, 233, 0.87);--buttonHover:#434C5E;--containerBorder:2px solid var(--borderOutline);--buttonBorder:1px solid var(--borderOutline);--scoreboard1:#3B4252;--scoreboard2:#4C566A;--typingWord:#D8DEE9;--highlightWord:#88C0D0;--correctWord:#A3BE8C;--wrongWord:#BF616A}\*{transition:background-color .5s,color 1s}body #screenContainer{background-color:var(--backgroundColor)}body #modalContent{background-color:var(--navBackground)}body .navButton .tooltip{background-color:var(--mediumText);color:var(--gameBackground)}#gameContainer #finishOverlay,#gameContainer #game,#gameContainer #game .gameArea{background-color:var(--gameBackground)}body h1,body h2{color:var(--highlightText)}body label,body p{color:var(--mediumText)}body nav{background-color:var(--navBackground)}body input{background-color:var(--focusBackground)}#gameContainer #game{border:var(--containerBorder);border-radius:7px}#gameContainer #game #statBox{background-color:var(--focusBackground);color:var(--mediumText)}#gameContainer #game #gameWordArea .typingWord{font-size:1.5rem;text-overflow:clip;color:var(--typingWord);transition:color .2s}#gameContainer #game #gameWordArea #highlightWord{color:var(--highlightWord)}#gameContainer #game #gameWordArea .correctWord{color:var(--correctWord)}#gameContainer #game #gameWordArea .wrongWord{color:var(--wrongWord)}#gameContainer #finishOverlay #finishNameArea #finishTypingField,#gameContainer #game #gameTypingArea #gameTypingField{color:var(--highlightText)}#gameContainer #startOverlay #startButton{border:var(--buttonBorder);background-color:var(--focusBackground);color:var(--smallText);border-radius:5px;transition:all .3s}#gameContainer #startOverlay #startButton:hover{background-color:var(--buttonHover);color:var(--hoverCol);border:1px solid var(--hoverCol)}#gameContainer #finishOverlay{border:var(--containerBorder);border-radius:7px}#gameContainer #finishOverlay #finishStatArea .finishStat{color:var(--mediumText);font-size:1.25rem}#settingsForm .settingsRadio .radioSelector{border:var(--buttonBorder);background-color:var(--focusBackground);color:var(--smallText);border-radius:5px;transition:all .3s}#settingsForm .settingsRadio .radioSelector:hover{background-color:var(--buttonHover);color:var(--hoverCol);border:1px solid var(--hoverCol)}#settingsForm .settingsRadio input:checked+label{background-color:var(--buttonHover);color:var(--hoverCol);border:1px solid var(--hoverCol)}#settingsForm .settingsMenuButton{border:var(--buttonBorder);background-color:var(--focusBackground);color:var(--smallText);border-radius:5px;transition:all .3s}#settingsForm .settingsMenuButton:hover{background-color:var(--buttonHover);color:var(--hoverCol);border:1px solid var(--hoverCol)}#scoresNav #scoreboard{background-color:var(--focusBackground)}#scoresNav #scoreboard .scoreHeader{color:var(--mediumText)}#scoresNav #scoreboard #scorebody>tr{color:var(--smallText)}#scoresNav #scoreboard #scorebody>tr:nth-child(odd){background-color:var(--scoreboard1)}#scoresNav #scoreboard #scorebody>tr:nth-child(even){background-color:var(--scoreboard2)}body #overlay,body #wordModal{background-color:rgba(0,0,0,.5);display:none;position:fixed;height:100%;width:100%}#wordModal #modalContent #modalWords{background-color:var(--focusBackground)}#wordModal #modalContent #modalWords .modalWord{color:var(--mediumText)}\*{font-family:Consolas;font-size:16px}body h1{font-size:2rem}body h2{font-size:1.75rem}body .radioHeader{font-size:1.5rem;font-weight:100}body .radioSelector{font-size:1.25rem}html{height:100%}body{height:100%;margin:0;padding:0;position:static}body #wordModal{overflow:auto;justify-content:center}body main{width:100%;height:100%;position:relative;display:flex;flex-direction:column}body .navbar,body main #pageText{display:flex;flex-direction:column;height:100%}body main #pageText{justify-content:space-between}body main #pageText #topText h1{margin-top:1rem}body main #pageText #bottomText{position:relative;background-color:var(--gameBackground);display:flex;flex-direction:row;justify-content:space-evenly;z-index:2}body main #pageText #bottomText p{color:var(--mediumText)}body main #pageText #bottomText p>a{color:var(--hoverCol)}body main #pageText #bottomText p>a:visited{color:var(--selectedCol)}body main #screenContainer{width:100%;height:100%;display:flex;justify-content:center;align-items:center;flex:1 1 0;position:absolute}main h1,main p{text-align:center}main .navButton{border:var(--buttonBorder);background-color:var(--focusBackground);color:var(--smallText);border-radius:5px;transition:all .3s;width:50px;height:50px;position:absolute;margin:1rem;padding:0}main .navButton:hover{background-color:var(--buttonHover);color:var(--hoverCol);border:1px solid var(--hoverCol)}main .navButton:focus{outline:0}main .navButton .navIcon{height:80%;margin:10%}main .navButton .tooltip{width:120px;height:50%;margin:20% 5px auto;visibility:hidden;text-align:center;padding:5% 0;border-radius:5px;position:absolute;z-index:10;opacity:0;transition:opacity .3s}main .navButton #scoreTooltip{right:105%}#gameContainer .gameOverlay,body #scoresNav,main #scoresButton{top:0;right:0}main .navButton:hover .tooltip{visibility:visible;opacity:1}main #settingsButton img{transition:transform .3s}main #settingsButton:hover img{transform:rotate(45deg)}body .navbar{width:0;position:fixed;overflow:hidden;opacity:0;transition:width .3s,opacity .15s}#game #gameTypingArea,body .open#overlay{display:block}body .open#scoresNav,body .open#settingsNav{width:25rem;opacity:1}body .open#settingsButton{margin-left:calc(25rem + 15px)}body .open#scoresButton{margin-right:calc(25rem + 15px)}#settingsForm{display:flex;flex-direction:column;height:100%}#settingsForm #settingsHeader{padding-left:15px}#settingsForm .radioHeader{text-align:center}#settingsForm .settingsRadio{display:flex;flex-direction:row;justify-content:space-evenly}#gameContainer #finishOverlay,#gameContainer #finishOverlay form #finishHiddenForms{display:none}#settingsForm .settingsRadio input{position:fixed;opacity:0;pointer-events:none}#settingsForm .settingsRadio .radioSelector{text-align:center;vertical-align:middle;width:7rem;height:2rem;margin:5px}#settingsForm .settingsRadio .lengthRadio:disabled+label{opacity:.5;pointer-events:none}#settingsForm .settingsMenuButton{width:15rem;height:3rem;margin:10px;font-size:1.25rem;align-self:center}#settingsForm #themeButtons{margin-top:auto;display:flex;flex-direction:row;justify-content:space-evenly}#wordModal{display:flex;justify-content:center;align-items:center}#wordModal #modalContent{border-radius:10px;padding:20px;position:absolute;border:var(--buttonBorder);width:80%;height:auto;animation-name:animatetop;animation-duration:.4s}#wordModal #modalContent .modalSection{padding:15px}#wordModal #modalContent #modalHeader{display:flex;flex-direction:row;text-align:center}#wordModal #modalContent #modalHeader #modalExit{margin-left:auto;font-size:2rem;height:2rem;width:2rem;border-radius:50%;color:var(--mediumText);text-align:center;vertical-align:middle;line-height:2rem}#wordModal #modalContent #modalHeader #modalExit:hover{background-color:var(--focusBackground);color:var(--highlightText)}#wordModal #modalContent #modalButtons{display:flex;justify-content:space-evenly}#wordModal #modalContent #modalButtons .modalButton{border:var(--buttonBorder);background-color:var(--focusBackground);color:var(--smallText);border-radius:5px;transition:all .3s;width:15rem;height:3rem}#wordModal #modalContent #modalButtons .modalButton:hover{background-color:var(--buttonHover);color:var(--hoverCol);border:1px solid var(--hoverCol)}#wordModal #modalContent #modalWords{display:flex;flex-wrap:wrap;border-radius:5px}#wordModal #modalContent #modalWords .modalWord{padding:2px}@keyframes animatetop{from{opacity:0}to{opacity:1}}#scoresNav{align-items:center}#scoresNav #scoreboard{width:90%;table-layout:fixed}#scoresNav #scoreboard #scorebody tr{text-align:center;width:50%}html #gameContainer,html #topText{z-index:2}html #overlay{z-index:3}html .navButton,html .navbar{z-index:4}html #wordModal{z-index:5}#gameContainer{width:39rem;height:18rem;position:absolute}#gameContainer #game{width:100%;height:100%;display:grid;grid-template-rows:2.5rem 6.0833333333rem 2.5rem 2.5rem;gap:.75rem;padding:.75rem;box-sizing:border-box}#game #gameStatsArea{display:flex;justify-content:center;align-items:center}#game #gameStatsArea #statBox{width:2.5rem;height:100%;display:flex;border-radius:50%;justify-content:center}#game #gameStatsArea #statBox #gameProgress{line-height:2.5rem;font-size:1.5rem}#game #gameWordArea{position:relative;overflow:hidden;padding:.5rem;padding-bottom:.6666666667rem}#game #gameTypingArea #gameTypingField{width:100%;height:calc(100% - 5px);border:0;margin:0;padding:0 0 0 5px;border-radius:5px;outline:0;font-size:1.5rem}#game #gameResetArea #gameResetButton{border:var(--buttonBorder);background-color:var(--focusBackground);color:var(--smallText);border-radius:5px;transition:all .3s;width:5rem;height:80%}#game #gameResetArea #gameResetButton:hover{background-color:var(--buttonHover);color:var(--hoverCol);border:1px solid var(--hoverCol)}#gameContainer .gameOverlay{width:100%;height:100%;position:absolute}#gameContainer #startOverlay{display:flex;justify-content:center;align-items:center;background-color:rgba(0,0,0,.3);backdrop-filter:blur(3px);opacity:1;height:100%}#gameContainer #startOverlay.hide{backdrop-filter:blur(0);opacity:0;height:0;transition:backdrop-filter .3s,opacity 30ms,height 0s .3s}#gameContainer #startOverlay #startButton{width:10rem;height:5rem;text-align:center;vertical-align:middle}#gameContainer #finishOverlay form{display:grid;grid-template-rows:20% 15% auto 20%;height:100%}#gameContainer #finishOverlay form #finishNameArea,#gameContainer #finishOverlay form #finishTitleArea{display:flex;justify-content:center;align-items:center}#gameContainer #finishOverlay form #finishNameArea #finishTypingField{margin:0;height:1.5rem;outline:0;border:var(--buttonBorder);border-radius:5px;text-align:center}#gameContainer #finishOverlay form #finishStatArea{display:grid;padding:0 10%;grid-template-columns:50% 50%;grid-template-rows:30% 70%}#gameContainer #finishOverlay form #finishStatArea .finishStat{justify-self:center;align-self:center}#gameContainer #finishOverlay form #finishRetryArea{display:flex;justify-content:space-evenly;align-items:center}#gameContainer #finishOverlay form #finishRetryArea .finishButton{border:var(--buttonBorder);background-color:var(--focusBackground);color:var(--smallText);border-radius:5px;transition:all .3s;width:5rem;height:2rem}#gameContainer #finishOverlay form #finishRetryArea .finishButton:hover{background-color:var(--buttonHover);color:var(--hoverCol);border:1px solid var(--hoverCol)}