# Assignment 2 (12.5%)
# Writing a complete client-side application

T-213-VEFF, Web programming I, 2025-1

Reykjavik University - Department of Computer Science, Menntavegi 1, 101 Reykjavík

**Deadline:** Friday, February 28th 2024, 23:59

This is the second assignment in Web Programming I, with the topic of developing a Simon Says web application based on the design created in Assignment 1. Part of this assignment is also to deploy the front end to a web server.

**This assignment has to be done individually, no group work is permitted.**

## 1 Overview

In this assignment, you will develop a Simon Says web application based on the design created in assignment 1. The assignment is divided into the following tasks:

1. **User interaction:** Implement clickable pads and keyboard support (*Q, W, A, S*) to play sounds.
2. **Game logic:** Fetch sequences from the backend, play them with correct timing, and validate user input.
3. **High score tracking:** Display and update the high score dynamically.
4. **Replay and reset:** Allow users to replay the current sequence and reset the game when refreshing the page.
5. **Deployment:** Deploy the frontend to a website (e.g., using Netlify) and share the deployment URL by replacing the placeholder in the *myDeploymentUrl.txt* file. Instructions for the deployment process can be found in Canvas, I will also show you the process in the Friday lecture on February 21st, which will also be available on recording after the class.

You are provided with a start pack (*assignment2_starter_pack*) which you should use as a starting point for your assignment.

# 2 Use Cases and Requirements

The focus in this assignment should be on the logic.

## 2.1 Use Cases

The following use cases must be implemented:

1. **Game initialization:**
   - When the page loads, the game state is reset by contacting the backend.
   - The game starts in an idle state (it should **not** be possible to click on any play buttons), ready for user interaction.
2. **Playing the game:**
   - When the start button is pressed the game is started.
   - The frontend plays the sequence by lighting up pads and playing sounds with reasonable interval.
   - The tone style that is played must be the one that is selected in the dropdown menu.
   - The player must repeat the sequence by clicking the game-pads or using the keyboard (*Q, W, A, S*).
3. **Validating user input:**
   - Once the player completes the sequence, their input is sent to the backend for validation.
   - If the input is correct, the game progresses with an extended sequence.
   - If the input is incorrect, a failure message (modal) appears, allowing the player to start a new game.
4. **High score tracking:**
   - The high score is retrieved from the backend when the game starts.
   - If the player achieves a new high score, it updates dynamically on the screen.
5. **Replay functionality:**
   - A Replay button allows the player to replay the current level's sequence.
   - This helps players memorize the pattern before attempting to repeat it.
6. **Game reset on refresh:**
   - When the page is refreshed, the game state is reset by contacting the backend.
   - The game always starts from level 1 after a page reload.

## 2.2 Technical Requirements

In addition to the use cases, the following technical requirements must be followed:

1. The application must consist of a single HTML file, named *index.html*.
2. The application must not trigger a full page reload—all interactions should happen dynamically.
3. The HTML file must be valid HTML5, passing the W3C Validator (https://validator.w3.org).
4. All CSS must be placed in a separate `style.css` file, and all JavaScript in a separate `app.js` file.
5. No inline CSS or JavaScript is allowed, except for JavaScript function calls in HTML event attributes.
6. Tone Mapping: Each pad must correspond to a specific note as follows:
   - `pad-red (Q)` → `C4`
   - `pad-yellow (W)` → `D4`
   - `pad-green (A)` → `E4`
   - `pad-blue (S)` → `F4`
7. The state of the buttons (Start and Replay) should depend on what is happening in the game, e.g. initially Start should be enabled and Replay should be disabled, but once the game has started it should be reversed.
8. Allowed External Libraries:
   - **Axios** (for AJAX HTTP requests).
   - **Tone.js** (for playing tones).
   - No other libraries are allowes.
9. JavaScript Best Practices:
   - Use *const* and *let*, respectively. When a variable should not be mutated, use *const*. If a variable must be mutated, use *let*. You should **not** be using *var* in this assignment.
   - When defining functions, use *arrow functions* unless you have a good reason not to.
   - When dealing with *asynchronous* functions you should opt for the *async/await* syntax in favor of *promises* for better readability.
   - You should be using *forEach* loops and other inbuilt array functions such as *map* instead of a "traditional" for-loops.
10. There are no restrictions on the ECMAScript (JavaScript) version.
11. The game must be deployed to a public website (e.g., Netlify) and the deployment URL must be submitted in *myDeploymentUrl.txt*.

You are free to design the application in any way you like, as long as it supports the named use cases, and closely follows the requirements.

## 3 Starting up the backend (Rest API)

Below you have all the steps necessary to run the backend. I will also show you how to start up the backend in the Friday lecture on February 14th, which will also be available on recording after the class.

1. It is necessary to install **node.js**, and it is also good to install an software to test and use the backend, e.g. **Postman**. Links for node.js and Postman can be found in Canvas on the "Useful Links and Tools" page.

2. Navigate to **assignment2_supplement_material/assignment2_backend** folder in a terminal window (can be done e.g. in terminal via VSCode).

3. Install the NPM packages listed in package.json by running the npm install command (npm i does the same thing). This will fetch all the required packages needed to run the backend.

4. To start up the process, you can run the "npm start" command. The process will now print out a message telling you it has started:

   ```
   > simon_says@1.0.0 start
   > node ./index.js
   Simon Says app running on port 3000
   ```

5. Your backend is now running http://localhost:3000 and can be accessed via Postman, cURL or your website.

The backend supports a number of endpoints ("actions") that are relevant to this assignment. These are:

1. **PUT** /api/v1/game-state
   A PUT request to this URL resets the game state, ensuring that:
   - The game starts from **level 1**.
   - The sequence is cleared.
   - The high score remains unchanged.

2. **POST** /api/v1/game-state/sequence
   A POST request to this URL validates the user's input sequence against the expected sequence for the current level. The request body must contain:
   - **sequence**: an array representing the tones played by the user.

   If the input is correct, the response includes the new level and updates the high score if applicable. If incorrect, the response indicates failure, allowing the user to retry.

In general, the different endpoints return 2xx status codes if successful, and 4xx/5xx status codes if unsuccessful. The POST method returns the created object, if successful. To call any of the endpoints, you send an HTTP request with the right HTTP method (verb) and the right request body to "your" backend url with the added endpoint path . Note that GET requests do not have a request body, while POST requires it. Additionally, it might be a good idea to start trying out the different requests using Postman or cUrl.

## 4 Supplementary Material

The supplemental material includes various resources to help you with this task. In the folder **assignment2_supplementary_files** you will find the following:

1. Example (named **axios.html**) that demonstrates how to perform a successful GET request to the backend using the Axios library. You can adapt this example to send requests to the other endpoints of the given backend as well.

2. File (named **tune.html**) that demonstrates how to play a tune with the Tone.js library.

3. Postman collection (named **assignment2.postman_collection.json**) which you can import into Postman to test out the API (backend).

In the folder assignment2_starter_pack you will eventually find various files that you should use as a base for you solution.

## 5 Submission

The assignment is submitted via Gradescope and deployed to the web. On Gradescope: submit the following files:

- **index.html** (this file is provided in the starter pack, but you will need to alter this file to solve this assignment)

- **app.js** (this file is provided in the starter pack as an empty file, add your Javascript code in this file to solve this assignment)

- **myDeploymentUrl.txt** (this file is provided in the starter pack, with a "dummy" url, you need to replace this url with the url to your website before turning in your code on Gradscope).

- **style.css** (this file is provided in the start pack, there should be no need to change this file to solve this assignment)

- **information.png** (this file is provided in the start pack, there should be no need to change this file to solve this assignment)

- **oops.jpg** (this file is provided in the start pack, there should be no need to change this file to solve this assignment)

Submitting extra files of folder will result in point deduction. **Submissions will not be accepted after the deadline.**

## 6 Grading and point deductions

On the next page you can see the criteria for grading, this list is not exhaustive but gives you an idea of how grading will be for the project.

| Criteria | Point Deduction |
|---|---|
| **Use Case 1: Game initialization (2 Point).** When the page is loaded, the game state is reset by contacting the backend. The game starts in an idle state (it should not be possible to click on any play buttons), ready for user interaction. | Up to -2 points, depending on the severity. |
| **Use Case 2: Playing the game (2 Points).** When the user presses the start button, the frontend plays the sequence by lighting up pads and playing sounds with reason- able interval. The player must then be able to repeat the sequence using the mouse or keyboard (*Q, W, A, S*). | Up to -2 points, depending on the severity. |
| **Use Case 3: Validating user input (2 Points).** When the player completes a sequence, their input is sent to the backend for validation. If the sequence is correct, the game progresses to the next level. If incorrect, a failure message (modal) appears, allowing the player to retry. | Up to -2 points, depending on the severity. |
| **Use Case 4: High score tracking (1 Point).** The high score is displayed and updated dynamically when a new record is achieved. | Up to -1 points, depending on the severity. |
| **Use Case 5: Replay functionality (2 Points).** The replay button allows the player to replay the current level's sequence without progressing. | Up to -2 points, depending on the severity. |
| **Use Case 6: Game reset on refresh (1 Point).** When the page is refreshed, the game state resets by contacting the backend, ensuring the game starts from level 1. After the restart the game should be in idle state (it should not be possible to click on any play buttons), ready for user interaction. | Up to -1 points, depending on the severity. |
| **HTML Validation:** The HTML document shall validate as an HTML 5 document without errors in the W3C validator (https://validator.w3.org), using automatic detection for the document type. | The HTML document does not validate as correct HTML in the W3C HTML Validator: -1 point per validation error (max -3 points). |
| **Other:** Other issues (using `var` instead of `let`/`const`, regular for-loops instead of array methods, not using arrow functions, using the `promise` syntax instead of `async/await`, site not deployed). | Point deduction depending on severity. |