SC-T-213-VEFF - Web Programming I

Project Assignment 2 – Client-side Application

Reykjavik University | Deadline: **22nd February 2019, 23:59**

The topic of this assignment is: **Writing a complete client-side application**.

1 Overview

In this assignment, you will develop a Sudoku web application. You will communicate with an existing server-side application that generates Sudoku boards of different difficulties, write the code that displays the board and allows users to fill it in, and write the logic that checks whether or not the current board violates the Sudoku rules.

This assignment can be done **individually, or as a group of three students**. No groups of 2 students or more than 3 are permitted.

In the following sections, the different parts of the application are outlined. In Section 2, the Sudoku rules and terminology are explained. How to access the Sudoku backend is explained in Section 3. The actual task together with requirements and constraints is described in Section 4.

Note: If you are new to HTML, CSS and JavaScript, this task might seem daunting for you. To get started, try to break the problem down into pieces and look into each problem separately. For instance, your first starting point should be to try contacting the backend to get a Sudoku board. Then, continue with the next step, and so forth.

2 Sudoku Rules

Sudoku is a logic puzzle in which you have to place numbers from 1 to 9 in the squares of a 9x9 grid. In an initial Sudoku board, some numbers are present, while the remaining cells are left empty. An example of such a board is depicted in Figure 1.

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Figure 1: An initial Sudoku board.

A Sudoku board is further divided into nine **boxes** (also referred to as sub-grids or blocks), numbered from box 1 to box 9 as in Figure 2.

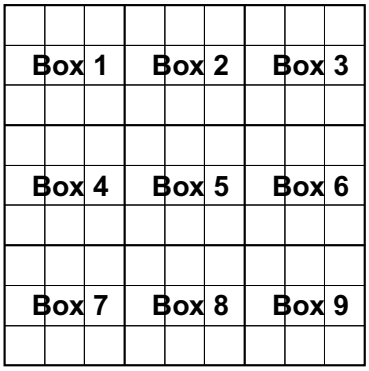

Figure 2: Box numbering in Sudoku.

A Sudoku is valid if the following rules are fulfilled:

1. Each cell contains a number between 1 and 9.
2. Every number is unique in its row.
3. Every number is unique in its column.
4. Every number is unique in its box.

For example, consider the cell marked red in Figure 3. The value in the cell can not be 3, 9, or 6, since those values are already in other cells in the same row. It cannot be 8, 6, or 3, since those values exist in other cells in the same column. Finally, it cannot be 8, 4, 7, since those values exist in the same box (Box 4 in this case).

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Figure 3: Sudoku rules example.

3 Sudoku Backend

For this assignment, we use an existing backend that can generate initial Sudoku boards of three different difficulties: easy, medium, hard. The backend is available at `https://veff213-sudoku.herokuapp.com/`. Whenever this website is not available, it means that the backend is down and cannot be reached.

To generate a new Sudoku, you send a POST request to `https://veff213-sudoku.herokuapp.com/api/v1/sudoku`. In the request body, you have to include the parameter *difficulty* with one of the

three values *easy*, *medium*, or *hard*. In the supplementary material, you find a general example (named *axios.html*) on how to do an AJAX POST request (with a body) in Axios. You can adapt this example to send requests to the Sudoku backend.

Depending on the supplied parameter value, the backend returns an initial Sudoku board of varying difficulty. The return value is an object named *board* which has the following attributes:

- *difficulty* - the difficulty of the returned board.
- *dateGenerated* - the date at which the board was generated on the server.
- *_id* - a unique id identifying the Sudoku board.
- *boxes* - an array of size 9, containing the 9 boxes of the Sudoku board. boxes[0] contains Box 1, boxes[1] contains Box 2, and so forth. Each box contains another array of size 9, which contains the values of this particular box. The values are sorted from top left to bottom right, where empty values are encoded as a dot ".". For example, the box with the marked cell in Figure 3 would be found in boxes[3] (since it is Box 4) and contain the following (string) values:
  - boxes[3][0]=8
  - boxes[3][1]=.
  - boxes[3][2]=.
  - boxes[3][3]=4
  - boxes[3][4]=.
  - boxes[3][5]=.
  - boxes[3][6]=7
  - boxes[3][7]=.
  - boxes[3][8]=.

4 Requirements and Constraints

The focus in this assignment should be on the logic. Since we are not studying design, the look of the application is secondary and does not give any extra points. However, the application needs to have a basic usability. For instance, the Sudoku board should be displayed using the same layout as in the example figures in Section 2 (using 9x9 input fields that are aligned in the right way). It is NOT sufficient to have a single text field in which all cell values are written using the comma notation.

The following five use cases shall be supported by the application:

1. The user first selects a difficulty (between easy, medium, and hard). Then, she presses the generate button. A Sudoku is received from the backend with the desired difficulty and displayed in the board. Cells that do not hold a value (i.e., a dot in the data structure) shall be left empty.

2. The user fills in empty cells or modifies them. She is not able to modify cells which were assigned a number on the initial board (these shall be marked in a grey shade). Whenever she clicks the validate button, the current board content is validated using the rules described in Section 2. The validation shall mark empty cells yellow, and cells that contain a number that violates the rules red. All other cells shall keep their original colour. Cells that were on the initial board shall not be marked. After five seconds, the marking is reset.

3. Whenever a board is validated that is complete (all cells have a value) and correct (no cell violates a rule), a success message shall be displayed to the user.
4. Whenever the backend does not return a board (i.e., the backend is not reachable or, for some reason[1], returns an error), the fitting default board from Appendix 4 shall be displayed instead.
5. Whenever the generate button is pressed again, the current board shall be cleared and a new Sudoku board requested and displayed (as described in Point 1).

The following constraints apply (which are aimed to make testing of your application easier for us):

1. All buttons named above shall use button tags
2. The generate button shall have id *generateButton*
3. The difficulty shall be selected using a dropdown menu (select tag) with id *difficultySelector*. The options within that menu shall have values *easy*, *medium*, and *hard* for the three difficulties.
4. The validate button shall have id *validateButton*
5. The sudoku id of the current board (returned by the backend) shall be displayed in a paragraph with id *sudokuId*. Use -1 for default boards. You are allowed to hide this paragraph from the user
6. Each cell in the Sudoku board shall be an input tag with type number. Each cell shall have id *cellXY*, where X is the array index of the box, and y is the array index of the cell, i.e., boxes[x][y]
7. The success message for Use case 3 shall be displayed in a div with id *resultMsg*.

In addition to the use cases and constraints, the following requirements shall be closely followed:

1. The application uses only one HTML file named *sudoku.html*.
2. The application shall never cause the HTML to reload.
3. 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.
4. The CSS document shall validate as correct CSS with the *CSS Level 3 + SVG* profile in the W3C CSS Validator (`https://jigsaw.w3.org/css-validator`).
5. The web site shall have basic responsive capabilities. That means that relative lengths should be used, so that the application re-sizes when the browser width changes. Absolute lengths (e.g., px) are allowed for margins and paddings only. The application shall be usable between 800px and 1600px width (that is, it should display properly within those bounds).
6. All CSS code shall be placed in its own CSS file, all JavaScript code in its own JS file. No inline CSS or JavaScript is allowed (with the exception of JavaScript function calls from HTML events).
7. External CSS files (e.g., Bootstrap) are allowed.
8. External JS libraries/frameworks (e.g., jQuery) are NOT allowed, with the exception of Axios.
9. There are no restrictions on the ECMAScript (JavaScript) version.

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

[1] Note that wrong usage of the backend is a mistake - not a valid reason to use the default boards.

Submission

The lab is submitted via Canvas. Submit a zip file containing your *sudoku.html* file and all additional resources needed (e.g., CSS files, JS files, images).

Appendix A: Default Sudoku Boards

In the following, we list three default Sudoku boards, one for each difficulty. Whenever the backend is unavailable or erroneous, the right one of these default boards shall be used (the one corresponding to the chosen difficulty). Figure 4 shows the easy default board, Figure 5 the medium default board, and Figure 6 the hard default board

5	6	4	8	7	2	3	9	1
		3		1				
2		1	3	9				5
4	2	9			8	7	1	3
6	5	7	2	3	1	8	4	9
3	1	8	9	4	7	5	2	6
		6	4	2	3		5	8
	3	5	7	8	9	2	6	4
8	4	2	1			9	3	7

Figure 4: Difficulty: Easy

8	7		4	5		2	1	
	4			2		8	5	
6	2	5		1			9	
7	6		9	3	1	5	4	8
5		4	8	6	2	3		1
	8		5		7	9	6	2
2		7		9	4			5
9	5	8	6	7	3	1		4
4		6	2		5			

Figure 5: Difficulty: Medium

4						5	3	9
9				4		6		1
						7		4
	9	6		7	8	2	5	3
	4	7	5		2	9	1	6
			1	9	6	8	4	7
		1		8		4		2
	8	4				3		5
2				5	4	1	7	8

Figure 6: Difficulty: Hard