Table of Contents

1. Overview

This document is the description of the code challenge to implement a Sudoku Solver as a client-server solution with Angular application. This document describes the Algorithm applied in the application, the coding strategy, the standers folder structure and summary of each files of the application, the documentation of REST API's endpoints and the report of code quality analyzed and generated by ESlint linting tool.

2. Problem statement

Create a Sudoku-Solver (solver, not generator) for a standard 9x9 Sudoku

This should be a small client-server solution with Angular App in the frontend which communicates with the Backend via REST-API. You can write the backend API in any language you like (NodeJS, for example).

The following features need to be implemented in the app:

- Provide a function to get a random puzzle from the server (first API endpoint) and display it. You can simply provide 10 or so sample puzzles from internet and store them on the server side, and then randomly choose one when calling the endpoint.
- Allow the user to populate the rest of the fields (he should not be able to change the values of the pre-populated fields loaded from the server). Also provide a reset function (to clear the user-entered values)
- Provide a function (second API endpoint) to check the submitted solution and visually mark errors (if any)
- Provide a function (third API endpoint) to automatically solve the puzzle in the current form (when not all fields are filled) and display the solution if one exists, or display errors otherwise.

Everything can run locally (on two different localhost ports), you don't need to worry about authorization, https, etc. Please store the solution on your GitHub or BitBucket account (whichever you may have). You can use any established Sudoku-solving algorithm, as long as you can explain how it is working.

3. Source code location

Sudoku-solver repository location: https://github.com/rinkeshpatel22/sudoku-solver

This repository includes below folders:

1. client-app - Client-side application in Angular 8
2. server-app - Server-side application in Node.js and Express.js
3. apidoc - includes HTML page with REST API endpoint documentations
4. ESlint-reports - Code quality analysis and reports of both client and server app.

4. Technologies / tools used

Server Side Application : Node.js, Express.js, ES6, Eslint
Client Side Application : Angular8, TypeScript, ES6, Rxjs, Bootstrap, ESlint, Tslint, SCSS

5. Setup in local environment

5.1 Server-Side Application

- Locate serve-app folder in node cmd
  - npm install
- Start the application
  - npm start (or node index.js)
- Port 3000 has been set in appConfig.js file
- ESlint code quality check and generate report.
  - npm run eslint-report (Please find report.html file in serve-app folder)

5.2 Client-Side apllication

- Locate client-app folder in node cmd
  - npm install
- Start the application
  - npm start (or ng serve -o)
- Port 4200 will be ready.
- Please find asset/appConfiguration.json file to configure API endpoint.
- ESlint Code quality check and generate report:
  - npm run eslint-report (Please find report.html file in client-app folder)
- The project in created and managed by using Angular CLI.

6. Folder Structure of Appplication

6.1 Server-Side Application folder structure with description of files

```
server-app
  |        index.js - Main application page that start and listen the server.
  |
  +---app
  |  +---controllers
  |  |      sudokuController.js  - Controls request-response
  |  |
  |  +---libs
  |  |      responseGeneratorLib.js - Generate API response in common format
  |  |      sudokuChekerLib.js - Logic for validation of Sudoku values.
  |  |      sudokuGeneratorLib.js – Returns a random puzzle from an array.
  |  |      sudokuSolverLib.js - Logic to solve the Sudoku.
  |  |
  |   \- ---routes
  |        sudokuRoute.js - Rotes the Client-side requests to desired controller.
  |
  \- ---config
        appConfig.js - Configuration file to set port and environment.
```

6.2 Client-side application folder structure with description of files.

```
src
 |  index.html
 |  styles.scss
 |
 +---app
 |   |  app.component.html
 |   |  app.component.ts
 |   |
 |   +---core
 |   |   |  core.module.ts – ensures that services remains singleton
 |   |   |
 |   |   +---constants
 |   |   |      app-constants.ts – constants used in application
 |   |   |      message-constants.ts – message constants to display on screen
 |   |   |
 |   |   +---interfaces
 |   |   |      sudoku-request.ts – interface for API request type
 |   |   |      sudoku-response.ts – Interface for API response type
 |   |   |
 |   |   \---services
 |   |          app-configuration.service.ts – gets configures API base url
 |   |          http-interceptor.service.ts – intercepts API error
 |   |          loader.service.ts – loading spinner strat-stop control
 |   |          sudoku.service.ts – REST API calls
 |   |
 |   +---shared
 |   |   |  shared.module.ts – import and exports shared components
 |   |   |
 |   |   +---header
 |   |   |      header.component.ts – display header of the application
 |   |   |
 |   |   \---loader
 |   |          loader.component.ts – display loading spinner on API call wait
 |   |
 |   +---styles
 |   |      header.scss – css styles for header
 |   |      loader.scss – css styles for loading spinner
 |   |      sudoku.scss – css styles for Sudoku board
 |   |      variables.scss – css variables used in application
 |   |
 |   \---sudoku
 |       |  sudoku.module.ts
 |       |
 |       \---sudoku
 |              sudoku.component.ts – display and manage puzzle board
 |
 \---assets
       appConfiguration.json – API Base URL can be configured from here.
```

7. REST API documentation

Please find “apidoc” folder in GitHub repository and run the “index.html” file.

Location: https://github.com/rinkeshpatel22/sudoku-solver/tree/master/apidoc

1. GET SUDOKU
2. CHECK SUDOKU
3. SOLVE SUDOKU

Sudoku - Get Puzzle

1.0.0

GET

```
http://localhost:3000/api/v1/getSudoku
```

Success-Response:

```
{
"error"  : false,
"status" : "200",
"message": "Get Sudoku successfull",
"data"   :  '[6,null,null,1,5,7,null,null,null,3,null,null,2,null,4,null,9,null,null,1,null,null,null,6,null,4,null,2,
}
```

Error-Response:- Get Sudoku Failed: | Error-Response:- Internal Server Error:

```
{
    "error": true,
    "status": "403",
    "message": "Get Sudoku failed",
    "data": {}
}
```

Sudoku - Check Puzzle

1.0.0

POST

```
http://localhost:3000/api/v1/checkSudoku
```

Parameter

Field	Type	Description
sudoku	array	Entered values to check valid or not. (body params)(required)

Request-Example:

```
{
 "sudoku": '[[6,7,2,8,9,4,5,3,1],[5,4,9,1,6,3,8,2,3],[8,3,1,5,2,7,6,4,9],[2,8,4,9,1,6,7,5,3],[3,9,6,4,7,5,2,1,8],[1,5,
}
```

Success-Response:- Invalid entires found: | Success-Response:- No invalid entires found:

```
{
    "error":false,
    "status":"200",
    "message":"Invalid entires found",
    "data":'[7,14,17,35]'
}
```

Sudoku - Solve Puzzle

1.0.0

POST

```
http://localhost:3000/api/v1/solveSudoku
```

Parameter

Field	Type	Description
sudoku	array	Unsolved Puzzle to solve. (body params)(required)

Request-Example:

```
{
 "sudoku": '[[null,null,null,null,9,null,5,null,1],[null,null,null,null,null,null,null,2,null],[8,3,null,null,2,null,nu
 }
```

Success-Response:- Solved:

```
{
"error"  : false,
"status" : "200",
"message": "Sudoku solved successfully",
"data"   :  '[[6,7,2,8,9,4,5,3,1],[5,4,9,1,6,3,8,2,7],[8,3,1,5,2,7,6,4,9],[2,8,4,9,1,6,7,5,3],[3,9,6,4,7,5,2,1,8],[1,5
}
```

8. Code quality analysis report

8.1 Serve-Side Application code quality analysis

Used ESlint tool to check code quality and generated the report as below:

ESLint Report - Success

Summary

0 problems

Details

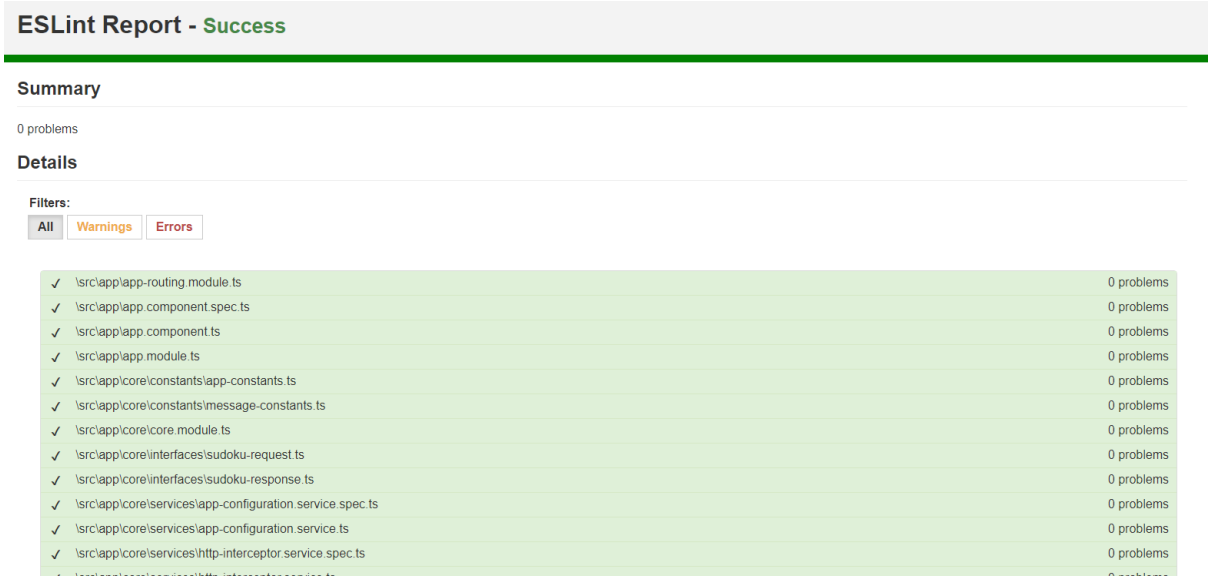
Filters: All | Warnings | Errors

✓	\app\controllers\sudokuController.js	0 problems
✓	\app\libs\responseGeneratorLib.js	0 problems
✓	\app\libs\sudokuChekerLib.js	0 problems
✓	\app\libs\sudokuGeneratorLib.js	0 problems
✓	\app\libs\sudokuSolverLib.js	0 problems
✓	\app\routes\sudokuRoute.js	0 problems

- Command to run ESLint analysis:
  - npm run eslint-report
  - after successful run, the report.html file will be generated in server-app folder

8.2 Client-Side application code quality analysis

Used ESlint tool to check code quality and generated the report as below:



- Command to run ESLint analysis:
  - npm run eslint-report
  - after successful run, the report.html file will be generated in client-app folder
- Also checked with tslint which is pre-installed when the project was created using Angular CLI:
  - npm run lint (or ng lint)

```
PS E:\projects\sudoku-solver\client-app> npm run lint

> ng lint

Linting "client-app"...
All files pass linting.
PS E:\projects\sudoku-solver\client-app> npm run eslint-report

> client-app@0.0.0 eslint-report E:\projects\sudoku-solver\client-app
> eslint --ext .ts src/app -f node_modules/eslint-detailed-reporter/lib/detailed.js --o report.html

PS E:\projects\sudoku-solver\client-app>
```

9. Backtracking Algorithm to solve Sudoku

The basic idea behind backtracking algorithm is to start finding the solution step by step with possible values and if any value found as invalid then back track to the previous step and go with another possible value. Repeat this process until the solution not found.

In Sudoku we can follow the same. We can start from an empty square to fill possible value from 1 to 9 which is exist in same row, same column and residing 3 X 3 square of that empty square. We can fill all empty squares one by one with possible values and if any value found invalid then back track to the previous square and so the same process again until all empty squares are filled with valid values.

Algorithm steps:

1. Find list of empty positions
2. Iterate the list of empty position
3. Tack first empty position and assume its value as 1
4. Check the assumed value is valid or not by checking current row, column and 3 X 3 square.
5. If value is valid then fill that value to the empty position and go for next empty position.
6. If the value is not valid then check for next value incrementing by 1 until valid value not found.
7. If all possible values (1 to 9) checked and no valid value found for the position then track back to the previous position and do the same process again.
8. Repeat the above steps until all positions are filled with valid values.

9.1 Code to solve Sudoku

In code the algorithm is divided into four functions:

1. saveEmptyPositions()

2. check3X3Square()

3. checkValue()

4. solveSudoku()

1. saveEmptyPositions

Takes puzzle array as an argument and returns a list of empty elements of the puzzle array.

```
37  // Keep empty positions in one array
38  let saveEmptyPositions = (sudoku) => {
39    let emptyPositions = [];
40    sudoku.forEach((rows, rowIndex) => rows.forEach((element, columnIndex) => {
41      if (element === null) {
42        emptyPositions.push([rowIndex, columnIndex]);
43      }
44    }));
45    return emptyPositions;
46  };
```

2. check3x3Square

Takes the possible value, row and column of empty position and puzzle array as an argument and returns true if the value is already exists in 3 X 3 square of empty position.

```
56   // Check if the value is already exists in current 3X3 square or not
57   let check3x3Square = (sudoku, column, row, value) => {
58     let rowCorner = Math.floor(row / 3) * 3;
59     let columnCorner = Math.floor(column / 3) * 3;
60     let rows = [rowCorner, rowCorner + 1, rowCorner + 2];
61     let columns = [columnCorner, columnCorner + 1, columnCorner + 2];
62     return rows.some(row => columns.some(column => sudoku[row][column] === value));
63   };
```

3. checkValue

Takes the possible value, row and column of empty position and puzzle array as an argument and returns false if the value is already exists in row, column or 3 X 3 square else returns true.

```
48   // Check if the VALUE is already exists in current row, column or 3X3 square
49   let checkValue = (sudoku, column, row, value) => {
50     let isColumnIncludesThisValue = sudoku.some(row => row[column] === value);
51     let isRowIncludesThisValue = sudoku[row].includes(value);
52     let is3X3SquareIncludesThisValue = check3x3Square(sudoku, column, row, value);
53     return (!isColumnIncludesThisValue && !isRowIncludesThisValue && !is3X3SquareIncludesThisValue);
54   };
```

4. solveSudoku

Takes puzzle array as an argument, get list of empty positions by calling its function and iterate all positions one by one, set possible values and call checkValue. If already exixts then try for other value from 1 to 9. If all possible values are invalid then back track to previous empty position and repeat the same logic.

```
1  let solveSudoku = (sudoku) => {
2    return new Promise((resolve, reject) => {
3      try {
4        var emptyPositions = saveEmptyPositions(sudoku);
5        let limit = 9, i = 0, row, column, value, found;
6
7        while (i < emptyPositions.length) {
8          row = emptyPositions[i][0];
9          column = emptyPositions[i][1];
10         value = sudoku[row][column] + 1;  // Try the first value
11         found = false;
12
```

```
            while (!found && value <= limit) {
              // If a valid value is found, mark found true, set the position to the value,
              // and move to the next position until the limit
              if (checkValue(sudoku, column, row, value)) {
                found = true;
                sudoku[row][column] = value;
                i++;
              } else {
                value++; // If value is not valid then increment the value by 1 and check
                         // until valid value not found.
              }
            }
```

```
            // If all possible values (1 to 9) checked and not found valid value
            // then move back track to the previous position
            if (found === false) {
              sudoku[row][column] = null;
              i--; // Back track
            }
          }
          resolve(sudoku);
        } catch (error) {
          reject(error);
        }
      });
    };
```

9.2 Code to check Sudoku

To check Sudoku iterate each square of Sudoku board from 1 to 81 and check the value of the square already exists in current row, column or 3 X 3 square or not. If exists then save the index in array and return the array.

```
// check the current row if the value repeats
if (sudoku[rowIndex].some((element, index) => element === value && index !== columnIndex)) {
    errorIndexList.push(indexCounter);
}
```

```
// check the current column if the value repeats
if (sudoku.some((row, index) => row[columnIndex] === value && index !== rowIndex)) {
    errorIndexList.push(indexCounter);
}
```

```
// Check the current 3X3 square if the value repeats
let rowCorner = Math.floor(rowIndex / 3) * 3;
let columnCorner = Math.floor(columnIndex / 3) * 3;
let rows = [rowCorner, rowCorner + 1, rowCorner + 2];
let columns = [columnCorner, columnCorner + 1, columnCorner + 2];
if (rows.some(row => columns.some(column => sudoku[row][column] === value && column !== columnIndex)
    errorIndexList.push(indexCounter);
}
```

10. Summary

This Sudoku Solver application has been implemented with trying to keep the number of iterations as minimum as possible to achieve an optimal solution. Used latest ES6 coding syntax to get benefit of ES6 over native JavaScript code. Kept all logical codes in try-catch block to handle unexpected errors. Used observables and promises wherever required for smooth and uninterrupted functioning over asynchronous functions. User centralized error handling using http-interceptor. Used interfaces to get benefits of Typescript strict typing and get and resolve early compile time error rather than runtime errors. Used Rxjs Subject to broadcast indication of loading spinner start and stop to keep user-interface smooth and user friendly while the client application is in waiting for API response. Implemented the HTML page as responsive to compatible in all viewports.