JavaScript Code Challenge v3

Introduction

This challenge is used to check your knowledge with modern web technologies. The focused technologies / frameworks are

- HTML 5,
- CSS 3,
- JavaScript (ES 5 / 6) and
- modern MVVM / MVW / MVC frameworks / libraries (preferred AngularJS 1.x / 2.x).

The challenge is separated into two main parts:

- **Part 1: Code fragments**
  General algorithms - your task is to finish the predefined code wrappers, thus they fulfill the requirements.

- **Part 2: Coding challenge**
  Show us your skills - A coding challenge with a broad range, including *design*, *development*, *testing* and *documentation*.

The complete challenge should be manageable within 2 - 4 evenings. Please keep in mind that some of the requirements are just *NICE TO HAVE* - try to finish the tasks you do *"as good as possible"* instead of *"starting as many as possible"*.

Have fun and enjoy the challenge!

1. Part: Code fragments

In this part you should finish several code fragments in the best way you can think of. Each code fragment consists of a description, a code wrapper and a small test suite. As described in section "*3. Finalize the challenge"* you should create one file per code fragment and add your solution to the final archive you send us.

1.1. Sum of the two smallest numbers

Description

Given an array of numbers, you should compute the sum of the two smallest number values. You can assume that the array only contains numbers, is never undefined and the array has min length of 2.

Fragment

```
function sumSmallestNumbers(numbers) {
  // your code
  return 0;
}
```

Test suite

```
describe('sumSmallestNumbers()', () => {

  it('should return the sum of the two smallest numbers', () => {
    expect(sumSmallestNumbers([1, 2, 3, 4])).toEqual(3);
    expect(sumSmallestNumbers([6, 7, 56, 2, 9, 34, 3])).toEqual(5);
    expect(sumSmallestNumbers([4, 4])).toEqual(8);
    expect(sumSmallestNumbers([5, 38, 15, 1, 1, -19, 9])).toEqual(-18);
    expect(sumSmallestNumbers([1, 1, 1, 1])).toEqual(2);
  });

});
```

1.2. Check if the amount of X and O in a string is the same

Description

Given an input string, you should check if the string contains the same amount of 'x' and 'o'. The case doesn't matter - if the amount is equal, return a `true` otherwise return `false`.

Fragment

```
function sameXOAmount(str) {
  // your code
  return true;
}
```

Test suite

```
describe('sameXOAmount()', () => {

  it('should return `true` if number of `X` and `O` is equal', () => {
    expect(sameXOAmount('xXoO')).toEqual(true);
    expect(sameXOAmount('aAxXXbBoOo')).toEqual(true);
    expect(sameXOAmount('abc')).toEqual(true);
  });

  it('should return `false` if number of `X` and `O` is not equal', () => {
    expect(sameXOAmount('OaAxXbBoO')).toEqual(false);
    expect(sameXOAmount('xgXoXsdxOxz')).toEqual(false);
    expect(sameXOAmount('aaAmmMxMM')).toEqual(false);
  });

});
```

1.3. Get the number which occurs an odd number of times

Description

Given an array of numbers, you should find the number which occurs an odd number of times within the array. You can assume that there is always just one number with an odd amount. Furthermore the array is never undefined and contains only numbers.

Fragment

```
function findOddAmount(numbers) {
  // your code
  return 0;
}
```

Test suite

```
describe('findOddAmount()', () => {

  it('should return the number which occurs with an odd frequency', () => {
    expect(findOddAmount([1, 2, 2, 3, 3])).toEqual(1);
    expect(findOddAmount([8, 8, 7, 7, 7])).toEqual(7);
    expect(findOddAmount([10, 3, 3, 10, 6, 10, 6, 1, 1])).toEqual(10);
  });

});
```

2. Part: Coding challenge

2.1. Task

The goal of this coding challenge is to create a component / module which can be used to show **notification messages** within a SPA. Those notifications should be used to display information which give the user a feedback, but do not disturb him at the current work. One example would be a chat application, where the notification module displays a slight notification that someone has logged in. The module should be encapsulated, thus it could be shared across multiple projects. The final result should be included into a nice example page, where the features of the notification module can be tested in.

2.2 Requirements

The requirements for the notification module are separated into *MUST HAVE* and *NICE TO HAVE*. You should focus on the *MUST HAVE* requirements to pass this challenge. Furthermore try to solve the requirements *"as good as possible"* instead of *"starting as many as possible"*.

2.2.1 Must have

1. The notification module should display the notifications as overlay in front of the page.
2. The notifications shouldn't interrupt the user - they should be displayed in an area where the user notices the messages, but does not lose the focus on his work.
3. The notifications should have a header and a body.
4. The notification module should support three different notification categories: `"info", "warning" and "error"`
5. The notifications should have a different appearance based on the categories.
6. The notifications should be closable.

2.2.2 Nice to have

1. The notifications with the category `"info"` should be closed automatically after 90 seconds.
2. The notification module should display max 5 notifications at the same time.
3. If the max amount of notifications is reached, the notification module should combine the oldest notification into one group, thus the max amount is satisfied again.

2.3 Topics

The realization of this challenge is separated into four small topics which you should work on. You don’t have to care about the order, but you should be aware to work on each different topic.

- Design
- Development
- Test
- Documentation

2.3.1 Design

Create a draft / mockup with one of your favorite tools. The following questions should be answered within the mockup:

- How should the notifications look like (single & grouped notification)?
- Where should the notifications appear?
- How do new notifications appear?
- How should the user interact with the notifications?

2.3.2 Development

Create a standalone prototype based on a modern MVVM / MVC / MVW framework (preferred AngularJS), HTML5 and CSS3. **Write a simple SPA where the notification module is included and can be tested in**. Try to use multiple development tools like:

- Package / Dependency managers - like NPM / bower
- CSS preprocessors - like SASS / LESS
- Build tools - like gulp / grunt

The following topics should be done during the development:

- Create a basic notification module
- Implement the requirements ;)
- Create an example page, where the module can be tested on
    - Input fields for header & body section
    - Category selector
    - Button to simulate the notification

2.3.3 Test

Write unit tests to check the functionality of your application. Reduce the test cases to the main services / controllers. Ensure that the test coverage within these is about 80%. Try to use test tools / runners like Jasmine and Karma.

2.3.4 Documentation

Document the resulting module / code with your favorite documentation generator / tool (like for AngularJS ngDoc or dgeni). Try to document the code as good as possible, thus other developers might be able to work on your code too.

---

3. Finalize the challenge

The result of your challenge should be packed into one archive, including the following data / information:

- **Part 1:** Code fragments (folder name `"fragments"`)
    - one file for each task / solution
- **Part 2:** Coding Challenge (folder name `"challenge"`)
    - designed mockups as JPEG / PNG / PDF
    - all project sources including the tests and documentation (if you use bower / npm, remove all dependencies which can be loaded automatically)
    - README file, containing information about how to use your result

Afterwards send your results to jana@riskident.com.

Thank you!