JS Advanced: Exam Preparation 2

Link on Judge: https://judge.softuni.org/Contests/3296/Js-Advanced-Final-Retake-Exam

Problem 1. Service

Environment Specifics

Please, be aware that every JS environment may **behave differently** when executing code. Certain things that work in the browser are not supported in **Node.js**, which is the environment used by **Judge**.

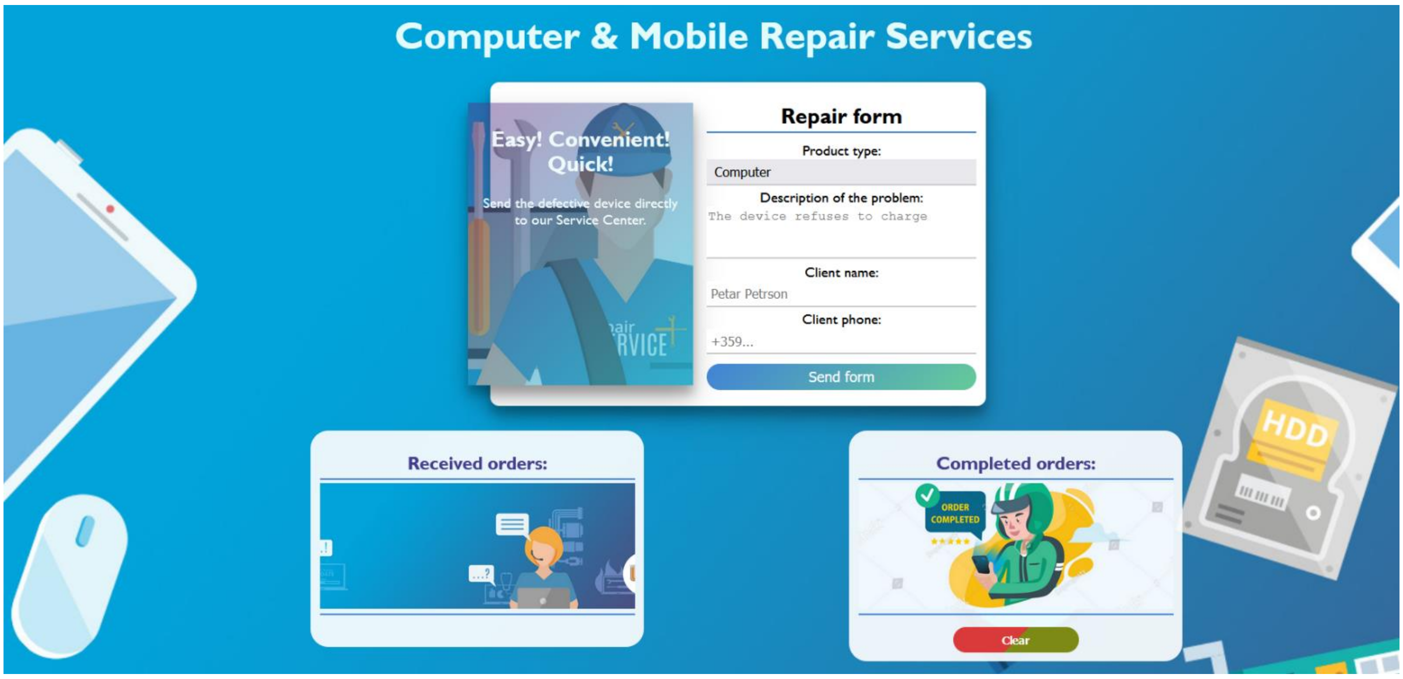
The following actions are **NOT** supported:

- `.forEach()` with **NodeList** (returned by `querySelector()` and `querySelectorAll()`)
- `.forEach()` with **HTMLCollection** (returned by `getElementsByClassName()` and `element.children`)
- Using the **spread-operator** (`...`) to convert a **NodeList** into an array
- `append()` in Judge (use only `appendChild()`)
- `replaceWith()` in Judge
- `replaceAll()` in Judge
- `closest()` in Judge
- `replaceChildren()`
- Always turn the collection into a **JS array** (forEach, forOf, et.)

If you want to perform these operations, you may use `Array.from()` to first convert the collection into an array.

Use the provided skeleton to solve this problem.

Note: You **can't** and you have no permission to **change** directly the given HTML code (index.html file).

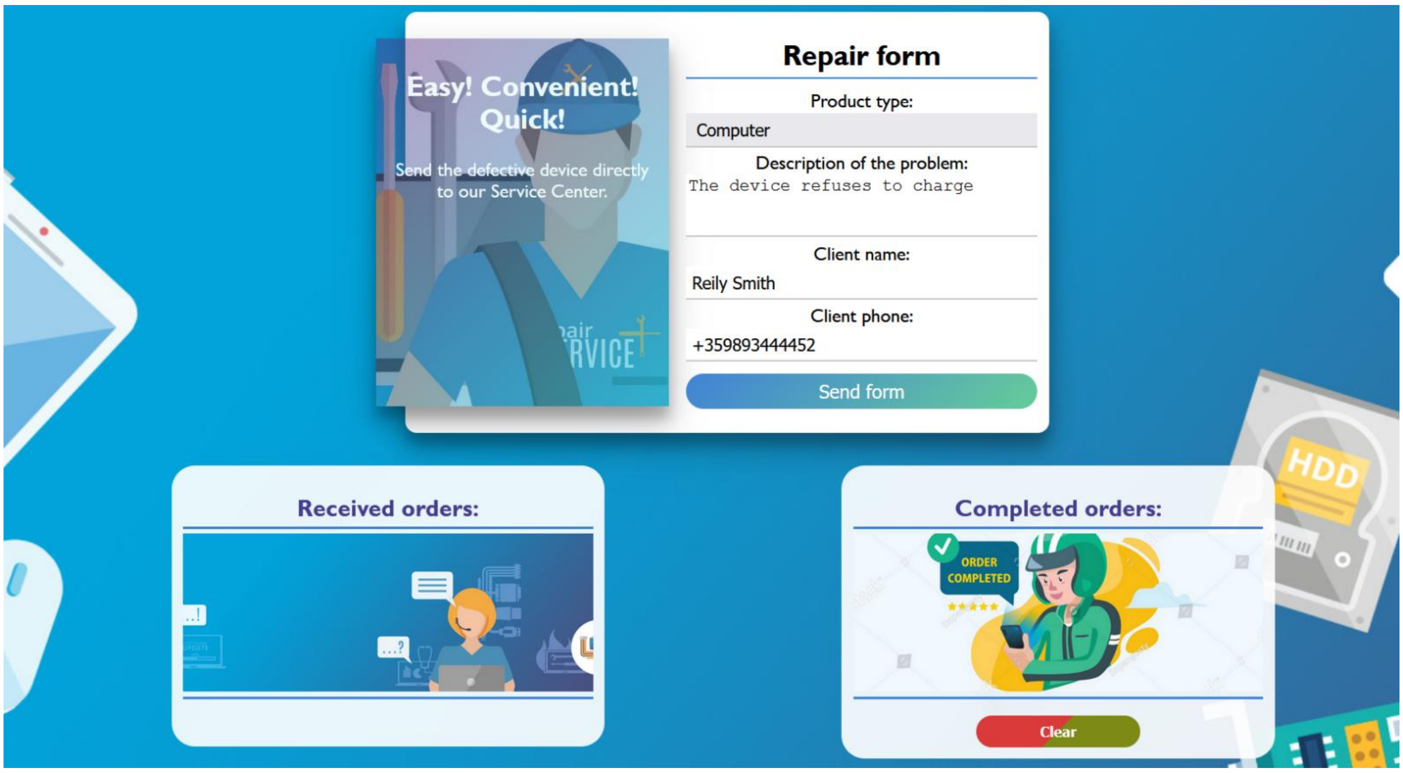


Your Task

Write the missing JavaScript code to make the **Service** work as expected:

- All fields **(description, client name, and client phone)** are **filled with the correct input**
    - **Description, client name,** and **client phone** are **non-empty strings**. If any of them are empty, the program should not do anything.
    - **Note** that the possible values for the Product type are two - **Phone** and **Computer**. To see which **drop-down menu** option is **selected**, read its parent's properties: **value.**

1. Getting the information from the repair form



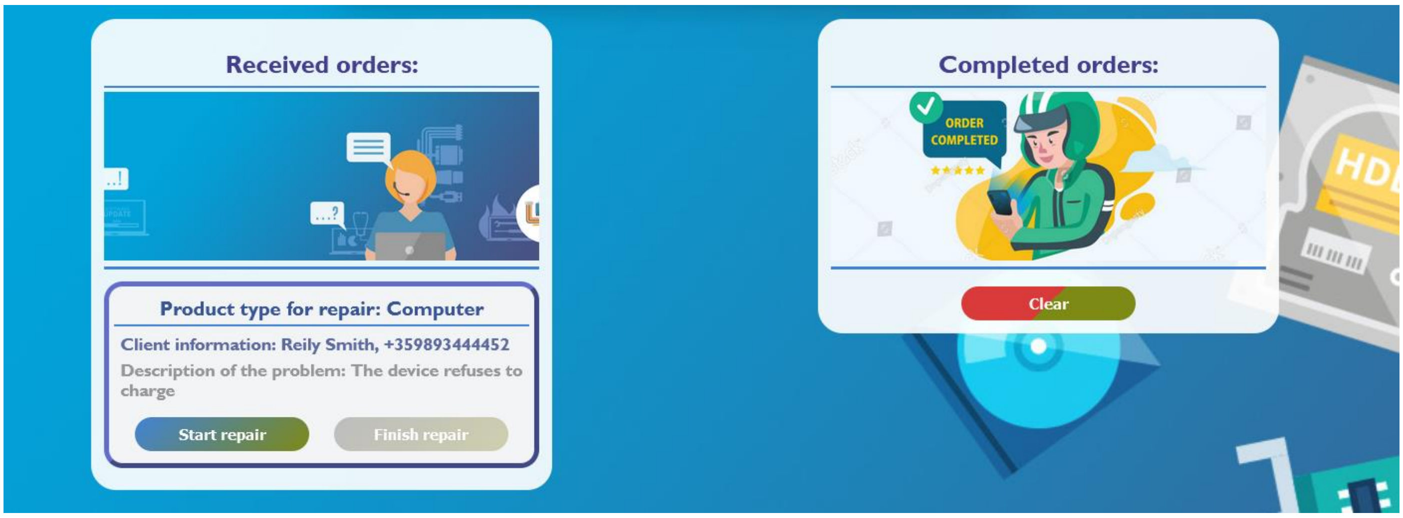
- When you click the **["Send form"]** button, the information from the input fields must be added to the **section** with the **id "received-orders"** and **then clear input fields**.
- The HTML structure looks like this:

```
<section id="received-orders">
    <h2>Received orders:</h2>
    <img src="./style/img/slider_2.jpg">

    <div class="container">
        <h2>Product type for repair: Computer</h2>
        <h3>Client information: Reily Smith, +359893444452</h3>
        <h4>Description of the problem: The device refuses to charge</h4>
        <button class="start-btn">Start repair</button>
        <button class="finish-btn" disabled>Finish repair</button>
    </div>

</section>
```

Note: When an order is successfully added, the button **["Finish repair"]** must be **disabled**, as the order cannot be completed if it has not started (Once the button is **disabled**, its color will turn gray).

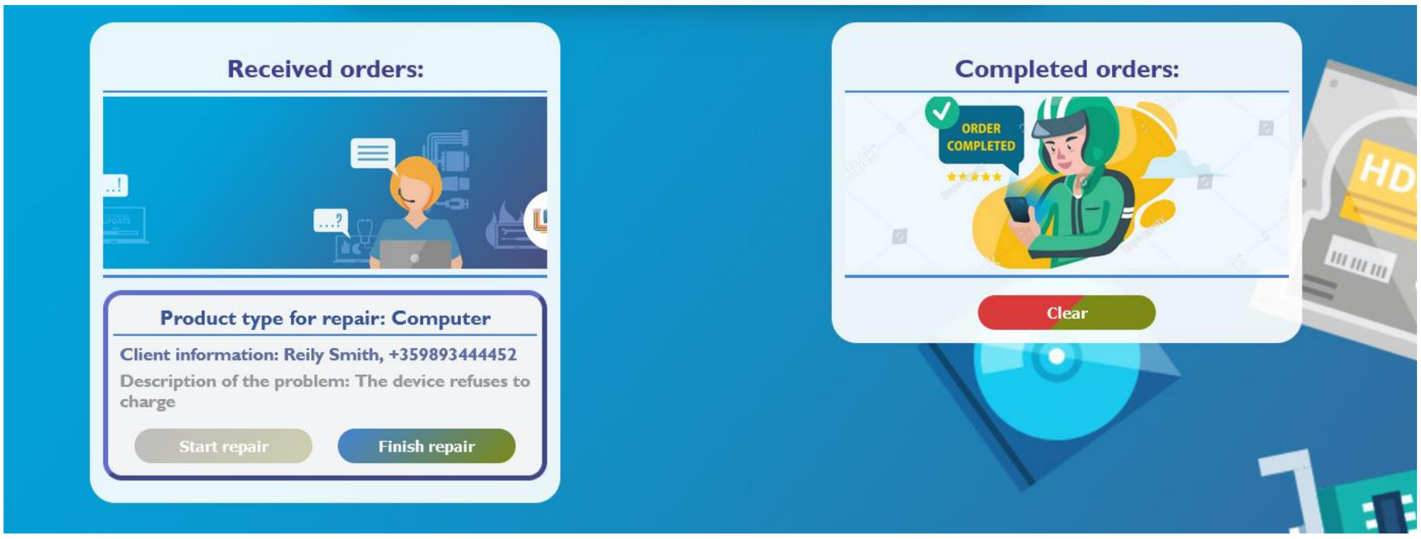


Product type for repair: Computer

Client information: Reily Smith, +359893444452

Description of the problem: The device refuses to charge

- When the **["Start repair"]** button is clicked, repair on the device begins. Since the process has already started, the worker will not be allowed to restart it, so the **["Start repair"]** button must be **disabled.** (Once the button is **disabled**, its color will turn gray).
- Button **["Finish repair"]** must become activated.

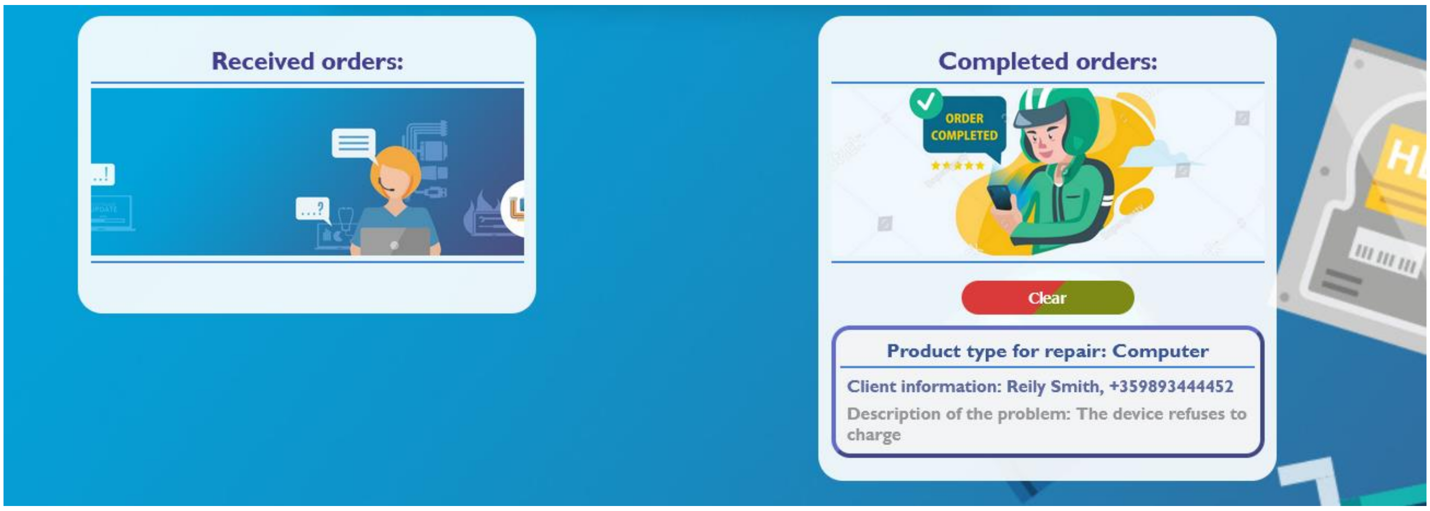


Product type for repair: Computer

Client information: Reily Smith, +359893444452

Description of the problem: The device refuses to charge

- When the **["Finish repair"]** button is clicked, repair on the device is complete. Therefore, you need to move the current device in the **section** with the **id "completed-orders"**.



Product type for repair: Computer

Client information: Reily Smith, +359893444452

Description of the problem: The device refuses to charge

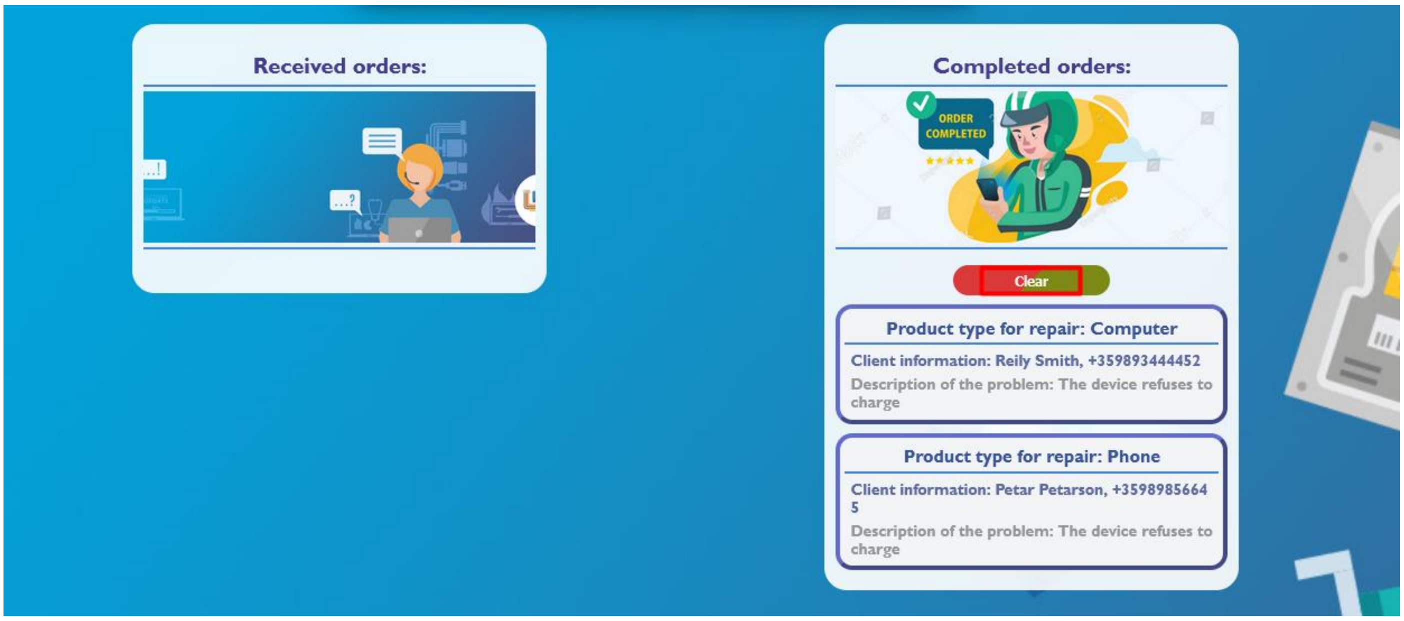
- The HTML structure looks like this:

```
<section id="completed-orders">
    <h2>Completed orders:</h2>
    <img src="./style/img/completed-order.png">
    <button class="clear-btn">Clear</button>

   <div class="container">
       <h2>Product type for repair: Computer</h2>
       <h3>Client information: Reily Smith, +359893444452</h3>
       <h4>Description of the problem: The device refuses to charge</h4>
   </div>

</section>
```

- When you click the **["Clear"]** button, you must **remove** all added **div** elements with **class "container"** from the section **Completed orders**.


Received orders:

Completed orders:

Product type for repair: Computer

Client information: Reily Smith, +359893444452

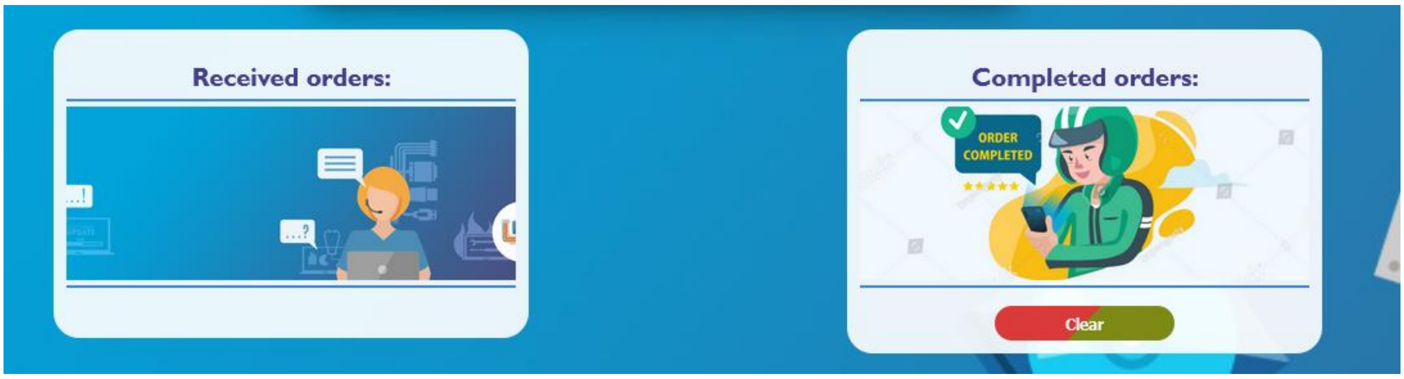
Description of the problem: The device refuses to charge

Product type for repair: Phone

Client information: Petar Petarson, +3598985664 5

Description of the problem: The device refuses to charge

- The HTML structure looks like this:


Received orders:

Completed orders:

```
<section id="completed-orders">
    <h2>Completed orders:</h2>
    <img src="./style/img/completed-order.png">
    <button class="clear-btn">Clear</button>

</section>
```

Problem 2. Vegetable store

```
class VegetableStore{
   //TODO Implement this class
}
```

Write a **class Vegetable store**, which supports the described functionality below.

Functionality

Constructor

Should have these **3** properties:

- `owner - string`
- `location - string`
- `availableProducts - empty array`

At the initialization of the **VegetableStore** class, the **constructor** accepts the **owner** and **location.**

Hint: You can add more properties to help you finish the task.

loadingVegetables (vegetables)

This method makes loading of the products in the store. The method takes 1 argument: **vegetables (array of strings)**.

- **Every element** into this array is information about vegetable in the format:

  `"{type} {quantity} {price}"`

  - They are separated by a single space. The **quantity** and **price** are per unit kilogram. **Example**:

    `["Okra 2.5 3.5", "Beans 10 2.8", "Celery 5.5 2.2"…]`

- If the **type** of the current vegetable is already present in **`availableProducts`** array, add the new quantity to the old one and update the old price per kilogram **only if** the current one is **higher**.
- Otherwise, should **add** the vegetable, with properties: **`{type, quantity, price}`** to the **`availableProducts`** **array**.
- In all cases, you must **finally return a string** in the following format:

  `` `Successfully added {type1}, {type2}, …{typeN}` ``

  Note: When returning the **string**, keep in mind that the different **types** of **vegetables must** be:

  - **Unique** - for instance:
    - **"Successfully added Okra, Beans, Celery"** - is a correctly returned string
    - **"Successfully added Okra, Beans, Okra"** - is not a correctly returned string
  - **Separated** by **comma** and **space (`, `)**

buyingVegetables (selectedProducts)

With this method, customers can buy products from the store. The method takes 1 argument: **selectedProducts (array of strings)**.

- **Every element** in this array is information about the selected vegetables in the format:

  `"{type} {quantity}"`

- For each element of the array **`selectedProducts`**, check:
  - If the **type** of the current vegetable is not present in **`availableProducts`** array, an error with the following message should be **thrown**:

```
`{type} is not available in the store, your current bill is ${totalPrice}.`
```

        - **totalPrice -** is the total price of all customer's **purchases**, if there are **no** purchases yet the **value** should **be 0.00.**

    - If the **quantity** selected by the customer for a given vegetable **is greater** than the quantity recorded in the array **`availableProducts`**, an error with the following message should be **thrown**:

`` `The quantity {quantity} for the vegetable {type} is not available in the store, your current bill is ${totalPrice}.` ``

        - **totalPrice -** is the total price of all customer's **purchases**, if there are **no** purchases yet the **value** should **be 0.00.**

    - Otherwise, if the above conditions are not met, you have to **calculate** the **price** for the given vegetable by **multiplying** the price per kilogram for the **given type** by the **quantity** desired by the customer. Then reduce the quantity recorded in the **`availableProducts`** array.
    - **Note: Add** a **variable** that will calculate the **total price** obtained from the individual prices of **each** vegetable in the array.

- Finally, you need to **return** the string in the following format:

  `` `Great choice! You must pay the following amount ${totalPrice}.` ``

Note: The **`totalPrice`** must be rounded to the second decimal point and **before** the **price** must have a **dollar sign** (**$**).

rottingVegetable (type, quantity)

With this method, the freshness of the vegetables in the store is preserved, removing the rotting vegetables. The method takes 2 arguments:

- **`type (string)`**
- **`quantity (number)`**

- If the submitted **type** is not present in the **`availableProducts`** array, an error with the following message should be **thrown**:

  `` `{type} is not available in the store.` ``

- If the submitted **quantity is greater** than the quantity recorded in the **`availableProducts`** array, then the **value** of the quantity in the array becomes **zero,** and **return** the **following string:**

  `` `The entire quantity of the {type} has been removed.` ``

- Otherwise, reduce the **quantity** recorded in the array **`availableProducts`** with the quantity obtained as a parameter, and **return** the string in the following format:

  `` `Some quantity of the {type} has been removed.` ``

revision ()

- This method **returns all** available **products** in the store in the following format:
  - The first line shows the following message:

    "Available vegetables:"
  - On the new line, display information about each vegetable sorted in **ascending** order of **price**:

    `{type}-{quantity}-${price}`
  - The last line shows the following message:

    `The owner of the store is {owner}, and the location is {location}.`

Example

Input 1
`let vegStore = new VegetableStore("Jerrie Munro", "1463 Pette Kyosheta, Sofia");`
`console.log(vegStore.loadingVegetables(["Okra 2.5 3.5", "Beans 10 2.8", "Celery 5.5 2.2", "Celery 0.5 2.5"]));`

Output 1
Successfully added Okra, Beans, Celery

Input 2
`let vegStore = new VegetableStore("Jerrie Munro", "1463 Pette Kyosheta, Sofia");`
`console.log(vegStore.loadingVegetables(["Okra 2.5 3.5", "Beans 10 2.8", "Celery 5.5 2.2", "Celery 0.5 2.5"]));`
`console.log(vegStore.buyingVegetables(["Okra 1"]));`
`console.log(vegStore.buyingVegetables(["Beans 8", "Okra 1.5"]));`
`console.log(vegStore.buyingVegetables(["Banana 1", "Beans 2"]));`

Output 2
Successfully added Okra, Beans, Celery
Great choice! You must pay the following amount $3.50.
Great choice! You must pay the following amount $27.65.

```
Uncaught Error: Banana is not available in the store, your current bill
is $0.00.
```

Input 3

```
let vegStore = new VegetableStore("Jerrie Munro", "1463 Pette Kyosheta,
Sofia");
console.log(vegStore.loadingVegetables(["Okra 2.5 3.5", "Beans 10 2.8",
"Celery 5.5 2.2", "Celery 0.5 2.5"]));
console.log(vegStore.rottingVegetable("Okra", 1));
console.log(vegStore.rottingVegetable("Okra", 2.5));
console.log(vegStore.buyingVegetables(["Beans 8", "Okra 1.5"]));
```

Output 3

```
Successfully added Okra, Beans, Celery
Some quantity of the Okra has been removed.
The entire quantity of the Okra has been removed.
Uncaught Error: The quantity 1.5 for the vegetable Okra is not available
in the store, your current bill is $22.40.
```

Input 4

```
let vegStore = new VegetableStore("Jerrie Munro", "1463 Pette Kyosheta,
Sofia");
console.log(vegStore.loadingVegetables(["Okra 2.5 3.5", "Beans 10 2.8",
"Celery 5.5 2.2", "Celery 0.5 2.5"]));
console.log(vegStore.rottingVegetable("Okra", 1));
console.log(vegStore.rottingVegetable("Okra", 2.5));
console.log(vegStore.buyingVegetables(["Beans 8", "Celery 1.5"]));
console.log(vegStore.revision());
```

Output 4

```
Successfully added Okra, Beans, Celery
Some quantity of the Okra has been removed.
The entire quantity of the Okra has been removed.
Great choice! You must pay the following amount $26.15.
Available vegetables:
Celery-4.5-$2.5
Beans-2-$2.8
Okra-0-$3.5
The owner of the store is Jerrie Munro, and the location is 1463 Pette
Kyosheta, Sofia.
```

Problem 3. Unit Testing

Your Task

Using **Mocha** and **Chai** write **JS Unit Tests** to test a variable named **companyAdministration**, which represents an object. You may use the following code as a template:

```
describe("Tests …", function() {
    describe("TODO …", function() {

        it("TODO …", function() {
            // TODO: …
        });
     });

     // TODO: …
});
```

The object that should have the following functionality:

`hiringEmployee (name, position, yearsExperience)` - A function that accepts three parameters: **string**, **string**, and **number**.

- If the value of the string **`position`** is different from "**Programmer**", **throw** an error: **`` `We are not looking for workers for this position.` ``**
- To be hired, the **employee** must meet the **following requirement**:
    - If the **`yearsExperience`** are **greater** than or equal to **3**, **return** the string:
      `` `{name} was successfully hired for the position {position}.` ``
- Otherwise, if the above conditions are not met, **return** the following message:

    `` `{name} is not approved for this position.` ``

  - There is **no** need for **validation** for the **input**, you will always be given string, string, and number.

- **`calculateSalary (hours)`** **-** A function that accepts one parameter: **number**.
  - Workers in this company receive **equal** pay per **hour** and this is **BGN 15**.
  - You need to **calculate** the salary by **multiplying** the pay **for one hour** by the number of **`hours`**.
  - **Also**, if the employee has been working for **more than 160 hours**, he must receive an additional **BGN 1000 bonus.**
  - Finally, **return** the employee's salary.
  - You need to validate the input, if the **`hours`** are not a **number**, or are a **negative** number, **throw** an error: "**`Invalid hours`**".

- **`firedEmployee (employees, index)`** **-** A function that accepts an array and number.
  - The **`employees`** array will store the names of its employees (["**Petar**", "**Ivan**", "**George**"…]).
  - You must **remove** an **element** (employee) from the **array** that is located on the **index** specified as a parameter.
  - Finally, **return** the changed array of employees as a string, **joined** by a **comma** and a **space**.
  - There is a need for validation for the input, an **array** and index may not always be valid. In case of submitted **invalid** parameters, **throw** an error "**`Invalid input`**":
    - If passed **`employees`** parameter is not an array.
    - If the **`index`** is not a number and is outside the limits of the array.

JS Code

To ease you in the process, you are provided with an implementation that meets all of the specification requirements for the **companyAdministration** object:

`companyAdministration.js`

```
const companyAdministration = {

    hiringEmployee(name, position, yearsExperience) {
        if (position == "Programmer") {
            if (yearsExperience >= 3) {
                return `${name} was successfully hired for the position
${position}.`;
            } else {
                return `${name} is not approved for this position.`;
            }
        }
        throw new Error(`We are not looking for workers for this position.`);
    },
    calculateSalary(hours) {

        let payPerHour = 15;
        let totalAmount = payPerHour * hours;

        if (typeof hours !== "number" || hours < 0) {
            throw new Error("Invalid hours");
        } else if (hours > 160) {
            totalAmount += 1000;
        }
        return totalAmount;
    },
    firedEmployee(employees, index) {

        let result = [];

        if (!Array.isArray(employees) || !Number.isInteger(index) || index < 0 ||
index >= employees.length) {
            throw new Error("Invalid input");
        }
        for (let i = 0; i < employees.length; i++) {
            if (i !== index) {
                result.push(employees[i]);
            }
        }
        return result.join(", ");
    }}
```

Submission

Submit your tests inside a **`describe()`** statement, as shown above.