JS Advanced: Exam Preparation 2

Link to contest: https://judge.softuni.org/Contests/3519

Problem 1. Car Dealers

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**.

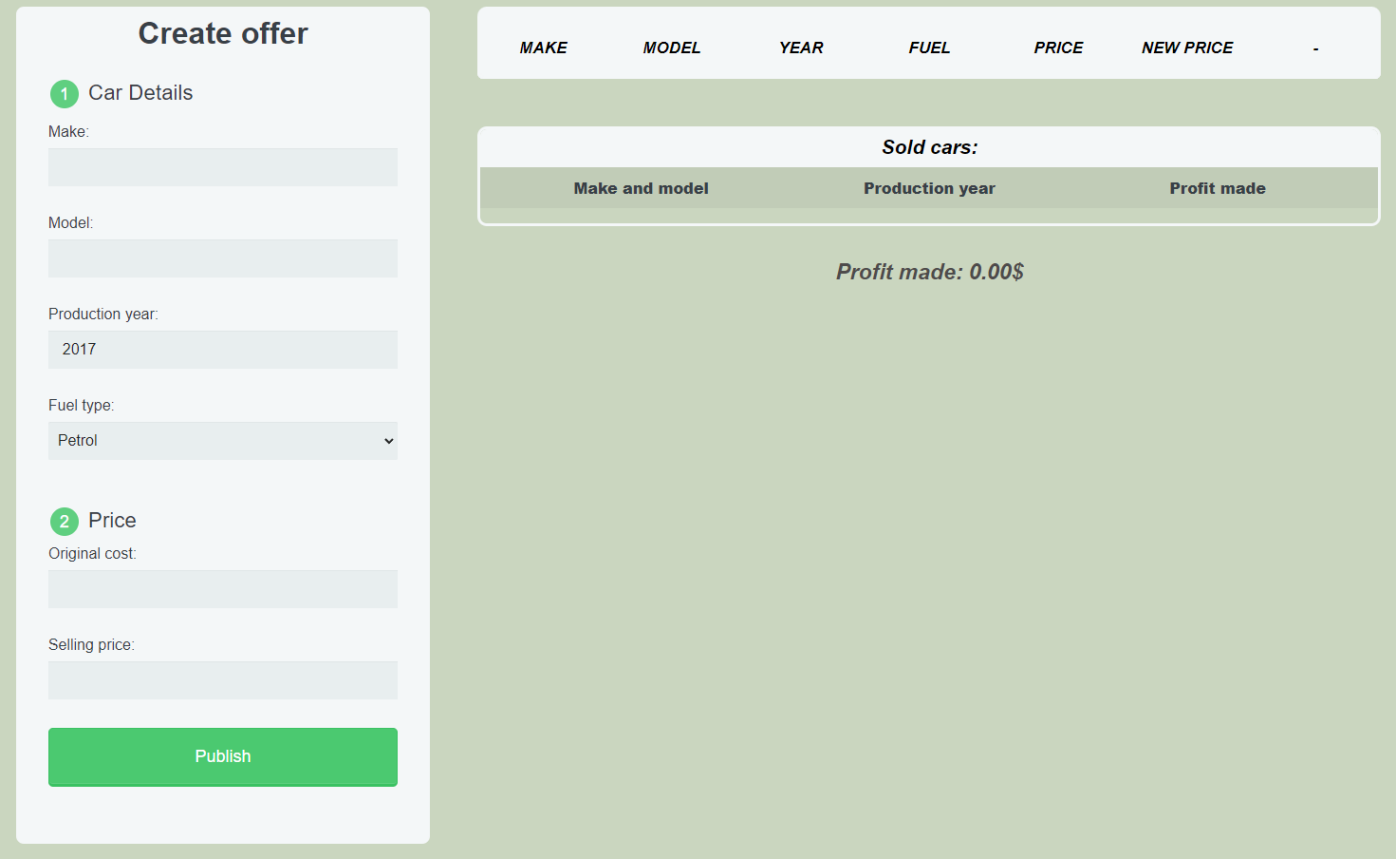
The following actions are **NOT** supported:

- `.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
- `replaceChildren()` in Judge
- `replaceAll()` in Judge
- `closest()` in Judge

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 **Car Dealer** work as expected:

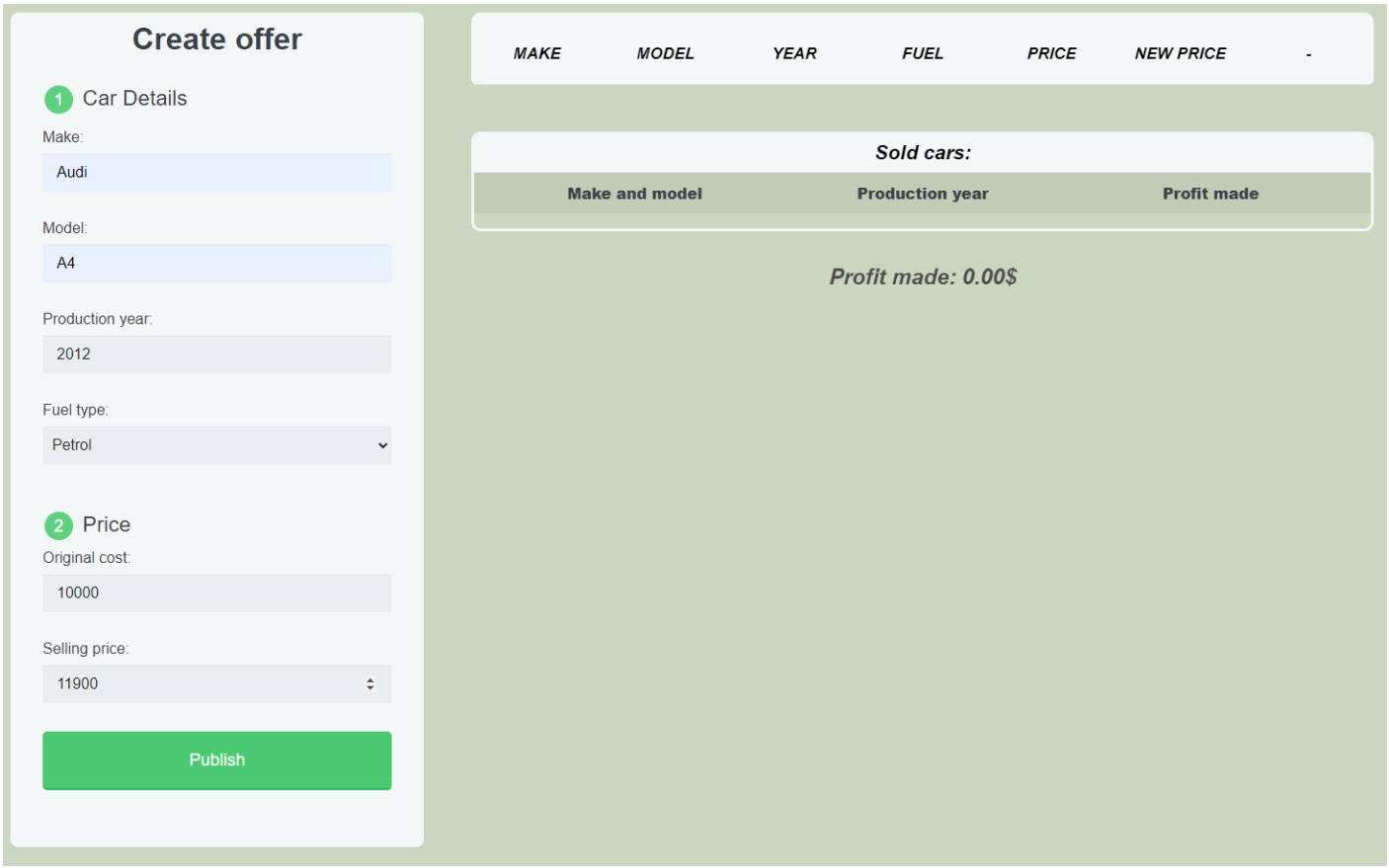
All fields **(make, model, year, fuel, original-cost** and **selling-price)** are **filled with the correct input** and **selling price** should have bigger value than **original price**

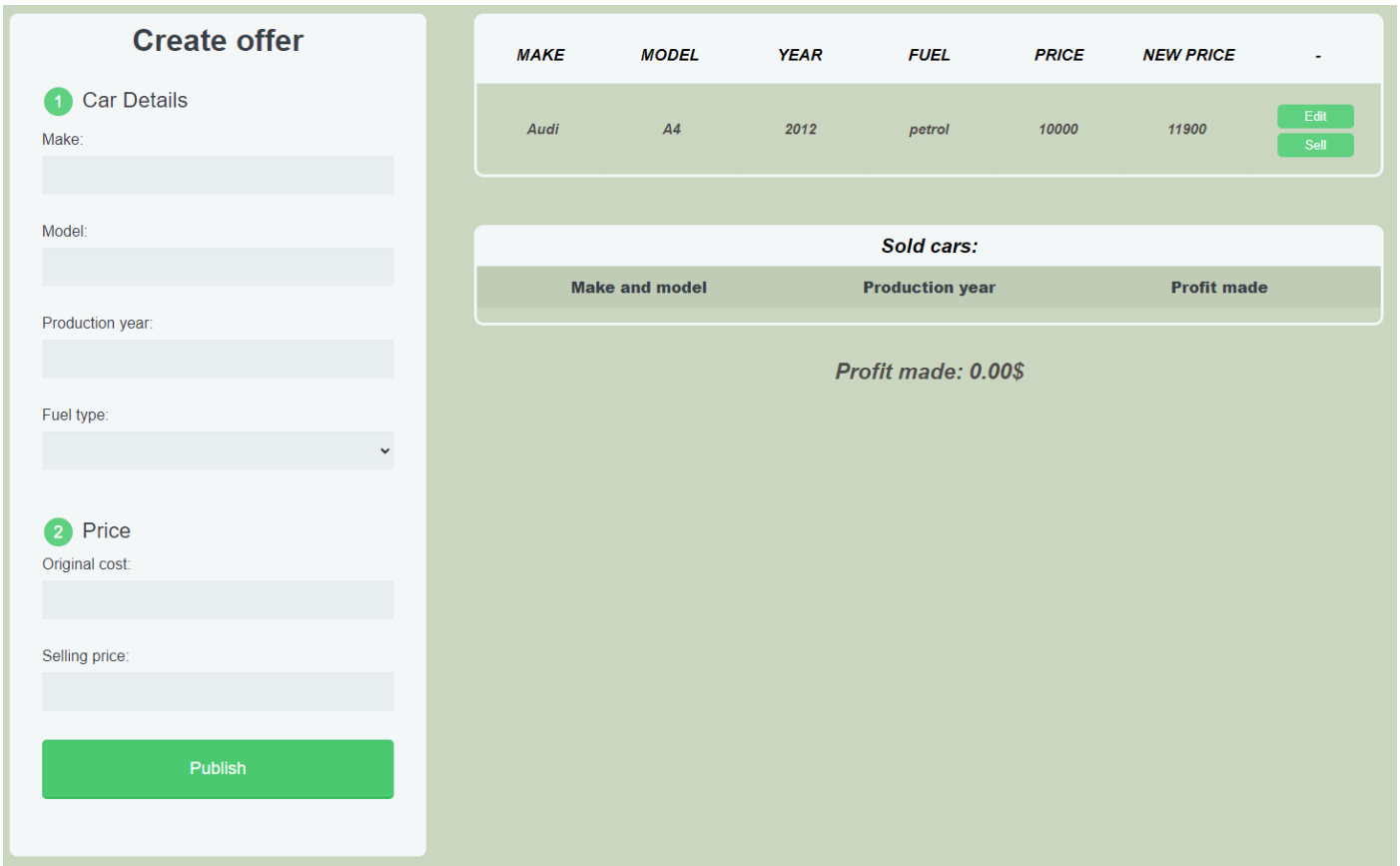
- **Make, model, year, fuel, original-cost** and **selling price** are **non-empty strings**. If any of them are empty, or **original price** is bigger than **selling price** the program should not do anything.

1. Getting the information from the form

When you click the **["Publish"] button,** the information from the input fields must be added to the **tbody** with the **id "table-body"**. **Then, clear all input fields**.
The HTML structure looks like this:

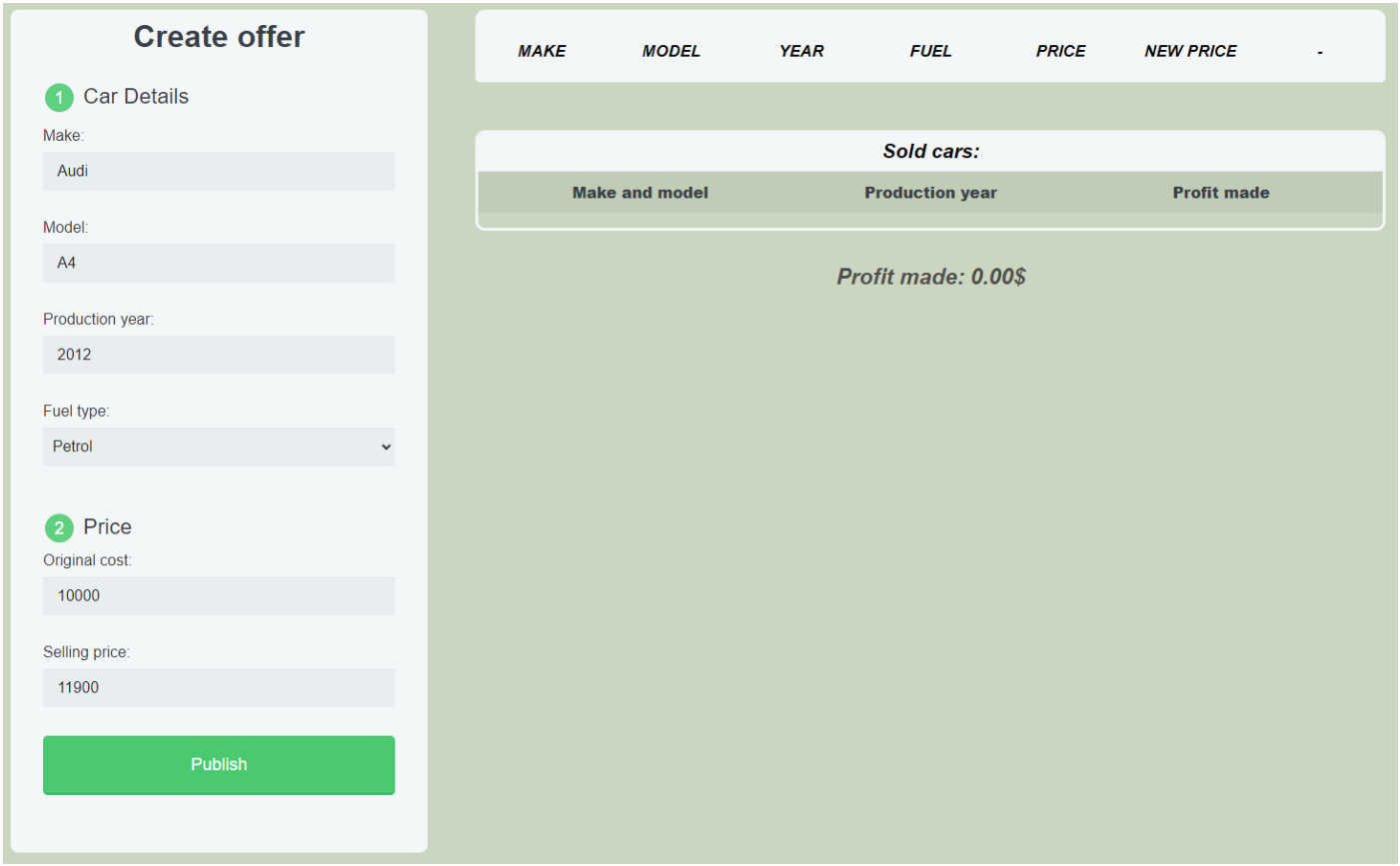
```
<tbody id="table-body">
    <tr class="row">
      <td>Audi</td>
      <td>A4</td>
      <td>2012</td>
      <td>petrol</td>
      <td>10000</td>
      <td>11900</td>
      <td>
        <button class="action-btn edit">Edit</button>
        <button class="action-btn sell">Sell</button>
      </td>
    </tr>
</tbody>
```



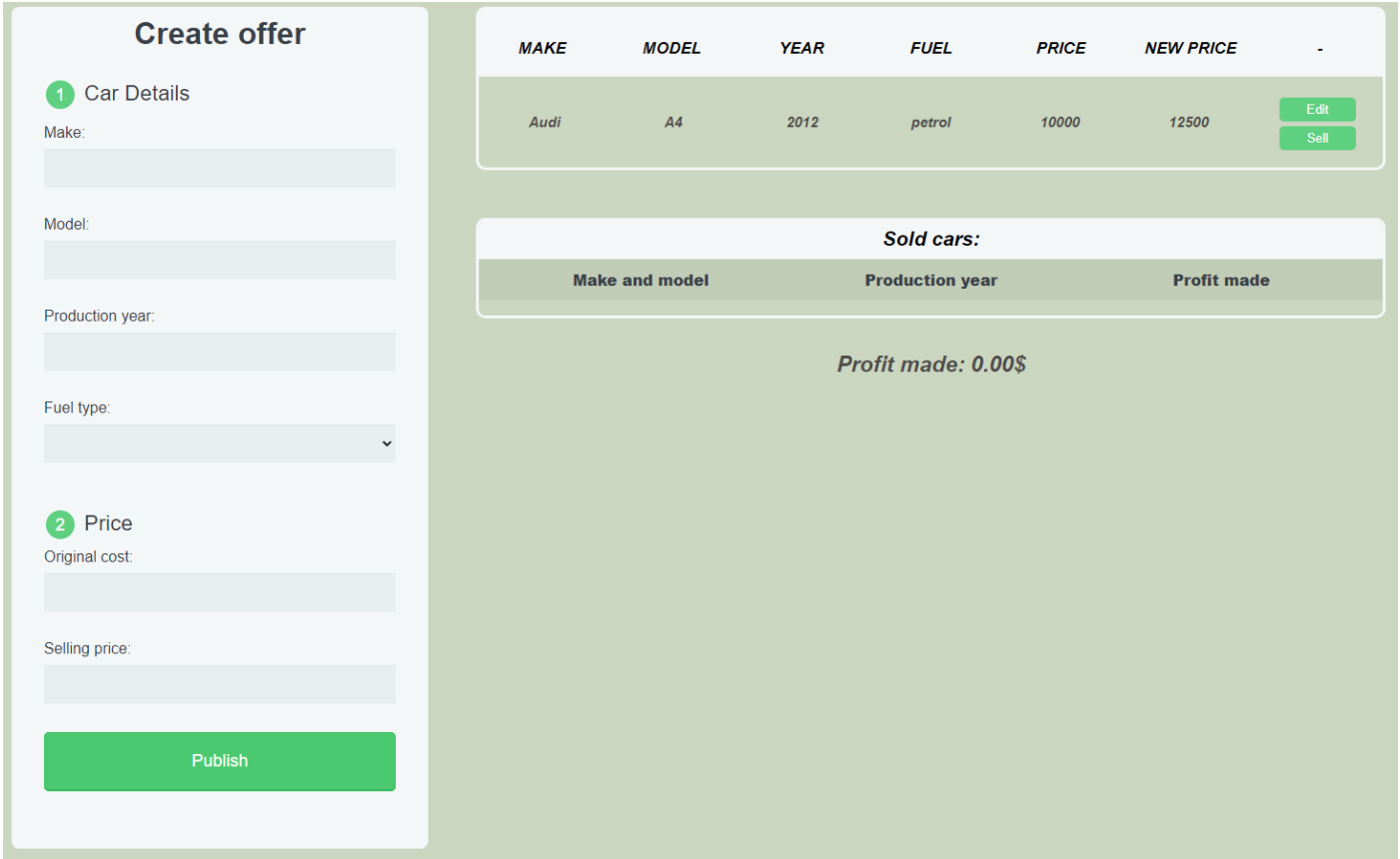


2. Edit information for posts

When the **["Edit"]** button is clicked, the information from the post must be sent to the input fields and the record should be deleted from the **tbody** with the **id "table-body"**.



After editing the information make a new record to the **tbody** with updated information.



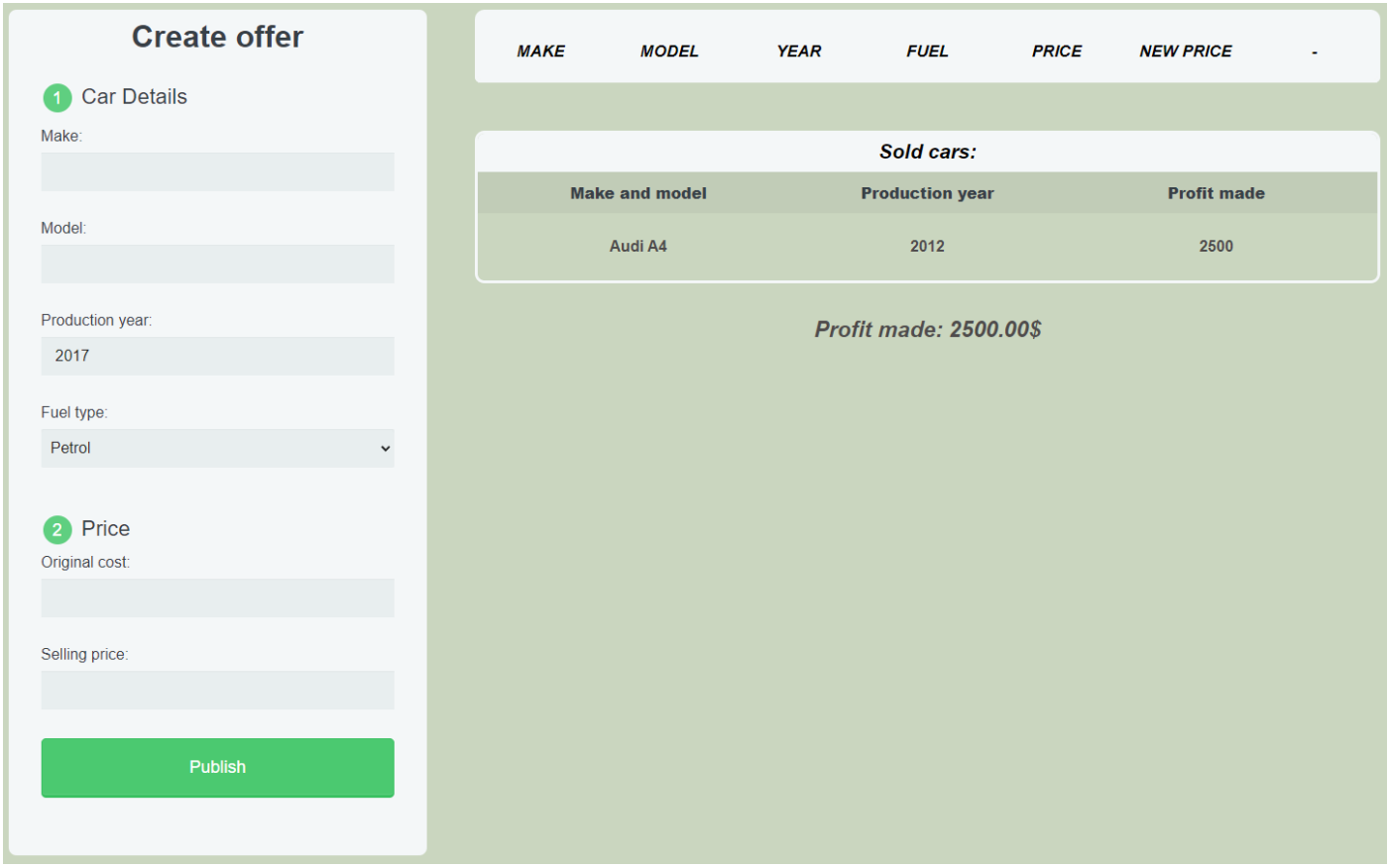
3. Sell car

When you click the **["Sell"]** button, the record must be **deleted** from the **tbody** with **id "table-body"** and a new record must be appended to the **ul** with the **id "cars-list"**
The new record should be as the following:

- First span should include both car **Make** and **Model** as whole string and separated by a single space
- Second span should include the **Production year**
- Third span should include the **difference** between the **Selling price** and **Original price**

```
<ul id="cars-list">
  <li class="each-list">
    <span>Audi A4</span>
    <span>2012</span>
    <span>2500</span>
  </li>
</ul>
```

Total profit made will be the **sum** from all sold cars profits which should be **rounded** to the **second digit** after the decimal point and should be displayed in **strong** with **id "profit"**



Problem 2. Smart Hike

```
class SmartHike {
    //TODO: implement this class
}
```

Implement a **class SmartHike**, which supports the functionality described below.

Functionality

Constructor

The constructor has **4** properties:

- **username – a string**
- **goals – an empty object**
- **listOfHikes – an empty array**
- **resources – 100**

At the initialization of the **SmartHike** class, the **constructor** accepts only the **username!**

The **goals** property is an **object**, representing a key-value pair of a peak's name and its altitude.

Methods

addGoal (peak, altitude)

This method adds a new goal to the **goals** object. The method accepts 2 arguments:

- **peak – a string**
- **altitude – a number**

If the **goal exists in the goals object**, **return** the following message:

`${peak} has already been added to your goals`

Otherwise, **add the new goal** to the goals object and **return** the following message:

`You have successfully added a new goal - ${peak}`

hike (peak, time, difficultyLevel)

Accept 3 arguments:

- **peak – a string**
- **time – a number**

- `difficultyLevel – "hard" or "easy"`

If the peak doesn't exist in the goals object, **throw new Error**:

```
`${peak} is not in your current goals`
```

If the peak exists in the goals object but the **resources** are already **0**, **throw new Error**:

```
"You don't have enough resources to start the hike"
```

Afterwards, find the **difference** between the current **resources** and the **time**, multiplied by 10. If the difference is a negative number, **return** the following message:

```
"You don't have enough resources to complete the hike"
```

Otherwise extract the used resources from all resources and add the hike to the **list of hikes** in the following format: { peak, time, difficultyLevel }

Return the following message:

```
`You hiked ${peak} peak for ${time} hours and you have ${resources}% resources left`
```

rest (time)

Accept 1 argument:

- `time – a number`

Add the time for rest multiplied by 10 to the resources.

If the **resources are more than or equal to 100**, set them to 100 and **return** the following message:

```
`Your resources are fully recharged. Time for hiking!`
```

Otherwise, **return** the following message:

```
`You have rested for ${time} hours and gained ${time*10}% resources`
```

showRecord (criteria)

Accept 1 argument:

- `criteria – a string`

This method **returns information** based on the criteria. The three possible types of criteria are: **"hard"**, **"easy"**, **"all".**

If the **list of hikes is empty**, return the following message:

`${username} has not done any hiking yet`

Find all hikes from the **list of hikes** depending on the given criterion **"hard" or "easy"** and find the best hike - the hike with the shortest time**.** If there are more than one hike with the same amount of time, list the **first** one added in the **list of hikes**.

Return the following message:

`${username}'s best ${criteria} hike is ${peak} peak, for ${time} hours`

If there is no hike with the given difficulty level, return:

`${username} has not done any ${criteria} hiking yet`

If the criterion is **"all", return** all hikes from the **list of hikes** array in following format:

- On first line show the following message:

  "All hiking records:"

- On the following lines, display information about each hike:

  `${username} hiked ${peak} for ${time} hours`

Examples

Input 1

`const user = new SmartHike('Vili');`
`console.log(user.addGoal('Musala', 2925));`
`console.log(user.addGoal('Rui', 1706));`
`console.log(user.addGoal('Musala', 2925));`

Output 1
You have successfully added a new goal - Musala You have successfully added a new goal - Rui Musala has already been added to your goals

Input 2

```
const user = new SmartHike('Vili');
console.log(user.addGoal('Musala', 2925));
console.log(user.addGoal('Rui', 1706));
console.log(user.hike('Musala', 8, 'hard'));
console.log(user.hike('Rui', 3, 'easy'));
console.log(user.hike('Everest', 12, 'hard'));
```

Output 2
You have successfully added a new goal - Musala You have successfully added a new goal - Rui You hiked Musala peak for 8 hours and you have 20% resources left You don't have enough resources to complete the hike Uncaught Error: Everest is not in your current goals

Input 3

`const user = new SmartHike('Vili');`
`console.log(user.addGoal('Musala', 2925));`
`console.log(user.hike('Musala', 8, 'hard'));`
`console.log(user.rest(4));`
`console.log(user.rest(5));`

Output 3
You have successfully added a new goal - Musala You hiked Musala peak for 8 hours and you have 20% resources left You have rested for 4 hours and gained 40% resources Your resources are fully recharged. Time for hiking!

Input 4
`const user = new SmartHike('Vili');` `console.log(user.showRecord('all'));`

Output 4
`Vili has not done any hiking yet`

Input 5

`const user = new SmartHike('Vili');`
`user.addGoal('Musala', 2925);`
`user.hike('Musala', 8, 'hard');`
`console.log(user.showRecord('easy'));`
`user.addGoal('Vihren', 2914);`
`user.hike('Vihren', 4, 'hard');`
`console.log(user.showRecord('hard'));`
`user.addGoal('Rui', 1706);`
`user.hike('Rui', 3, 'easy');`
`console.log(user.showRecord('all'));`

Output 5

`Vili has not done any easy hiking yet`
 	
`Vili's best hard hike is Musala peak, for 8 hours`
 	
`All hiking records:`
 	
`Vili hiked Musala for 8 hours`

Problem 3. Car Service

Your Task

Using **Mocha** and **Chai** write **JS Unit Tests** to test a variable named **carService**, 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:

`isItExpensive (issue)` - A function that accepts one parameter: **string**.

- If the value of the parameter **`issue`** is equal to "**`Engine`**" or "**`Transmission`**", return: **`` `The issue with the car is more severe and it will cost more money` ``**
- Otherwise, if the above conditions are not met, **return** the following message:

  `` `The overall price will be a bit cheaper` ``
- There is **no** need for **validation** for the **input**, you will always be given a string.

`discount (numberOfParts, totalPrice)` - A function that accepts two parameters: **number** and **number**.

- Percentage of discount based on the **`numberOfParts:`**
  - **15%** when **`numberOfParts`** is higher than 2 and smaller or equal to 7
  - **30%** when **`numberOfParts`** is higher than 7
- You need to **calculate** and **return** the **price** you will save, depending on the **discount**.

- If the **`numberOfParts`** is smaller or equal to **2,** return**:**

  `"You cannot apply a discount"`
- Otherwise, calculate the discount and **return** the following message:

  `` `Discount applied! You saved ${result}$` ``
- You need to validate the input, if the **`numberOfParts`** and **`totalPrice`** are not a **number**, **throw** an error: "**`Invalid input`**"

`partsToBuy (partsCatalog, neededParts)` - A function that accepts two arrays.

- The **`partsCatalog`** array will store the parts and the price for them ([{ **part**: "**blowoff valve**", **price**: **145** }, { **part: "coil springs", price: 230** } ...])
- The **`neededParts`** array will store the parts that you need to buy (["**blowoff valve**", "**injectors**" ...])
- You must iterate through both the arrays and calculate the **total price** of the **parts** that are equal to the **`neededParts`**.
- If **`partsCatalog`** is empty**,** return **`0`**

- Finally, **return** the total price of all parts needed.
- There is a need for validation for the input, may not always be valid. In case of submitted **invalid** parameters, **throw** an error "**`Invalid input`**":
  - If passed **`partsCatalog`** or **`neededParts`** parameters are not an arrays.

JS Code

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

carService.js

```
const carService = {
  isItExpensive(issue) {
    if (issue === "Engine" || issue === "Transmission") {
      return `The issue with the car is more severe and it will cost more money`;
    } else {
      return `The overall price will be a bit cheaper`;
    }
  },
  discount(numberOfParts, totalPrice) {
    if (
      typeof numberOfParts !== "number" ||
      typeof totalPrice !== "number"
    ) {
      throw new Error("Invalid input");
    }

    let discountPercentage = 0;

    if (numberOfParts > 2 && numberOfParts <= 7) {
      discountPercentage = 15;
    } else if (numberOfParts > 7) {
      discountPercentage = 30;
    }

    let result = (discountPercentage / 100) * totalPrice;

    if (numberOfParts <= 2) {
      return "You cannot apply a discount";
    } else {
      return `Discount applied! You saved ${result}$`;
    }
  },
  partsToBuy(partsCatalog, neededParts) {
    let totalSum = 0;

    if (!Array.isArray(partsCatalog) || !Array.isArray(neededParts)) {
      throw new Error("Invalid input");
    }
    neededParts.forEach((neededPart) => {
      partsCatalog.map((obj) => {
```

```
        if (obj.part === neededPart) {
          totalSum += obj.price;
        }
      });
    });

    return totalSum;
  },
};
```