# JS Advanced: Exam Preparation 2

**Link on Judge:** <https://judge.softuni.org/Contests/3367/Js-Advanced-Final-Exam>

# Problem 1. Work Process

**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**.

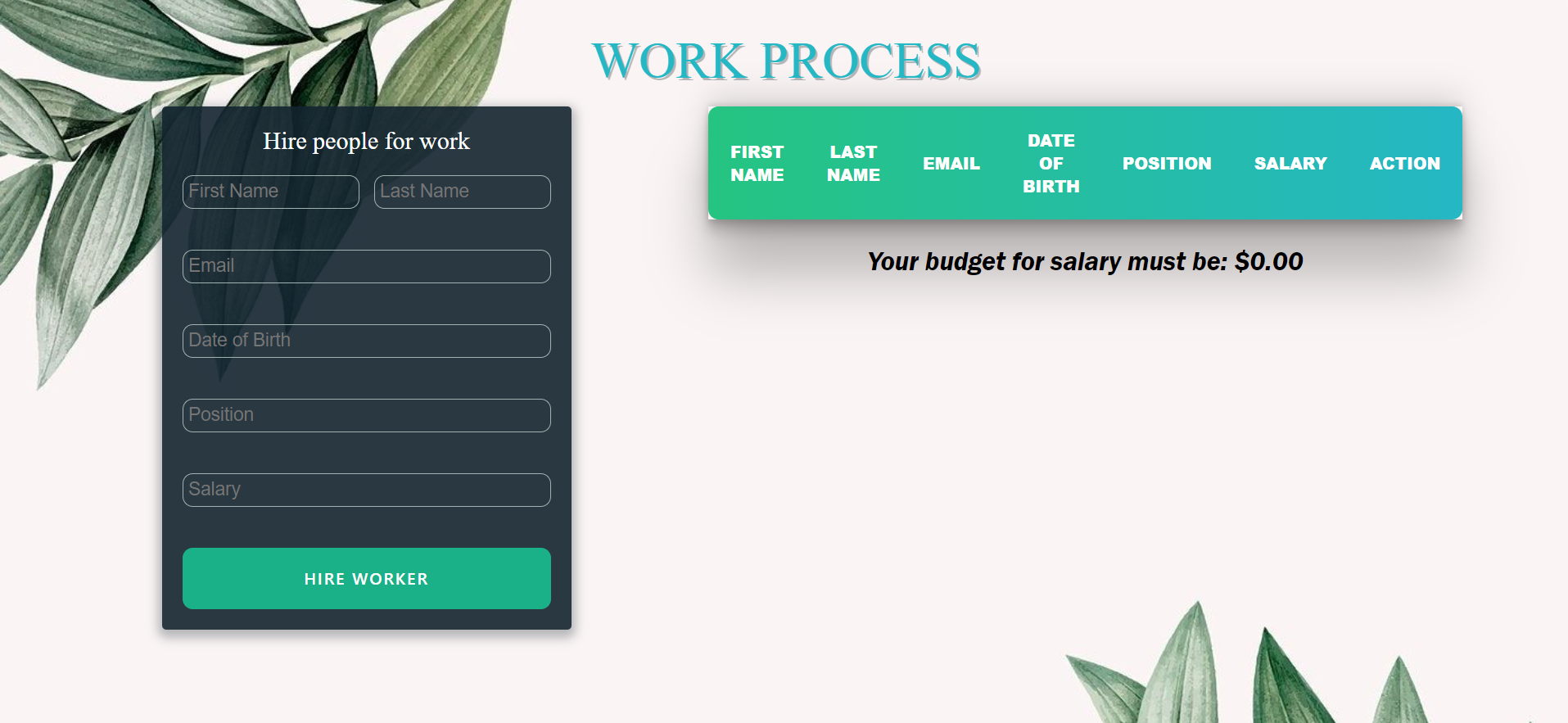
The following actions are **NOT** supported:

* **.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.**

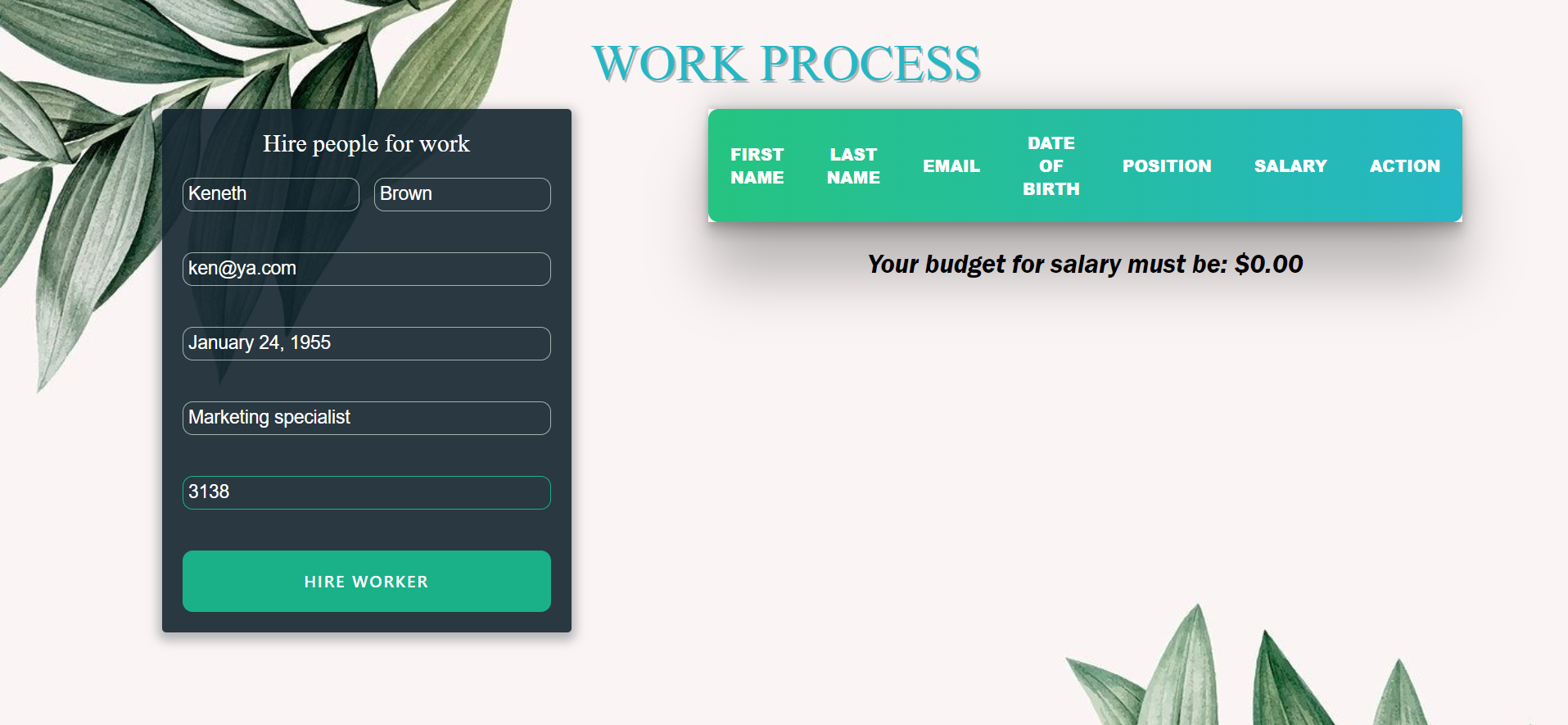
**Note**: You **can't** and you have no permission to **change** directly the given HTML code (index.html file).



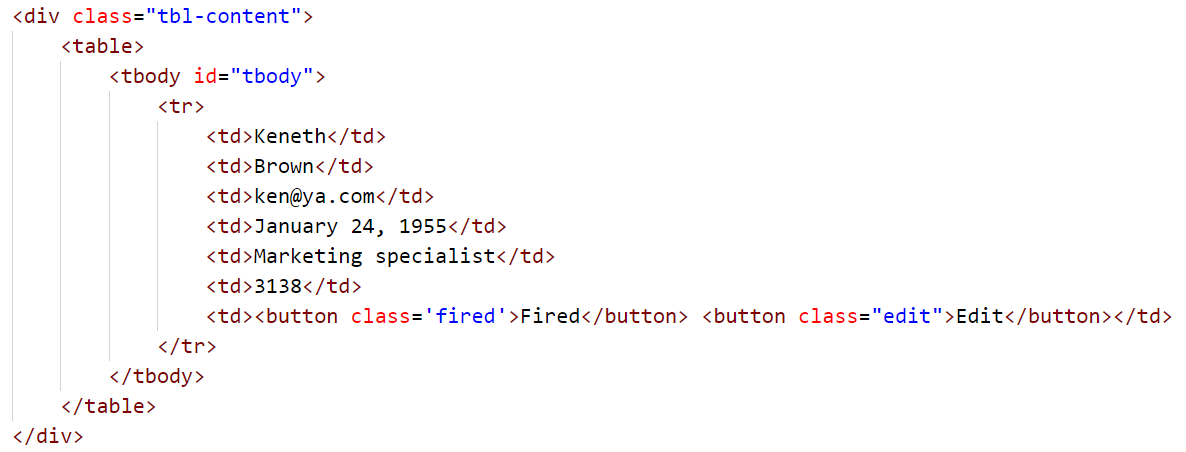
**Your Task**

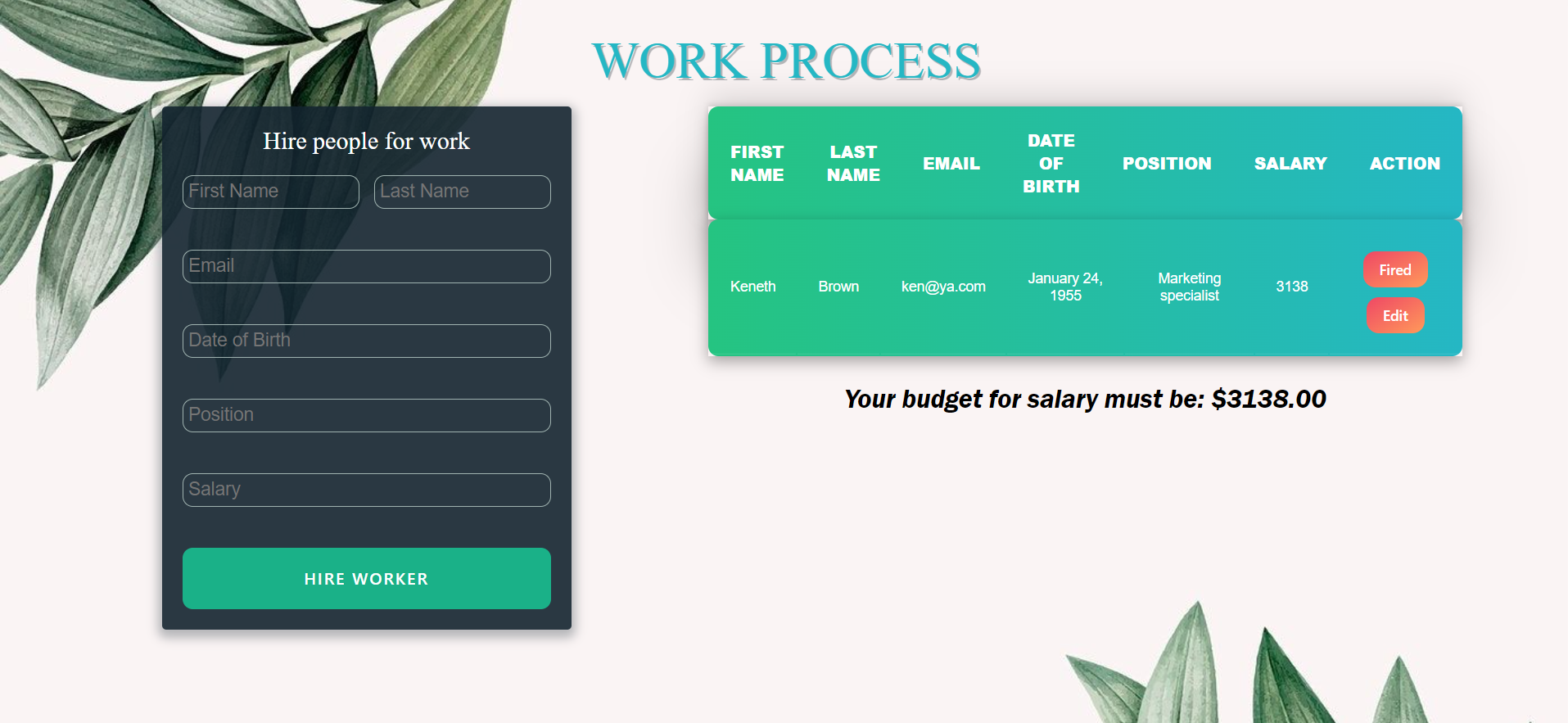
**Write the missing JavaScript code** to make the **Work Process** work as expected:

# Getting the information from the hired form

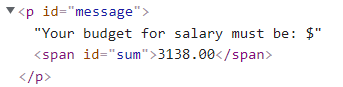


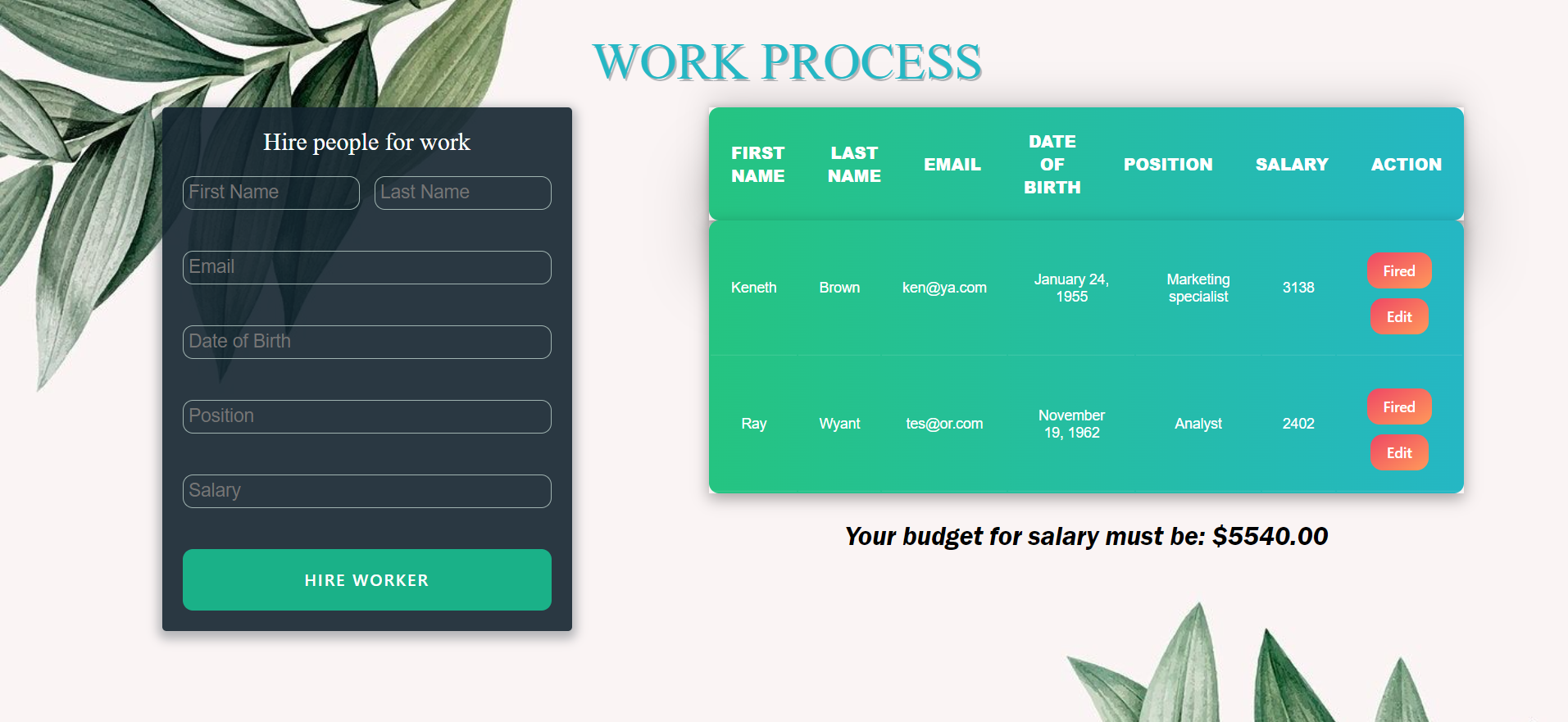
* **All fields** are **non**-**empty** **strings**. If any of them are empty, the program should not do anything.
* When you click the **["**Hire Worker**"**] button, the information from the input fields must be added to the **table** and **then clear input fields**.
* Each input must be saved in **td** tag and the end must be added buttons – **Fired** and **Edit.** Look at the example below.
* The HTML structure looks like this:





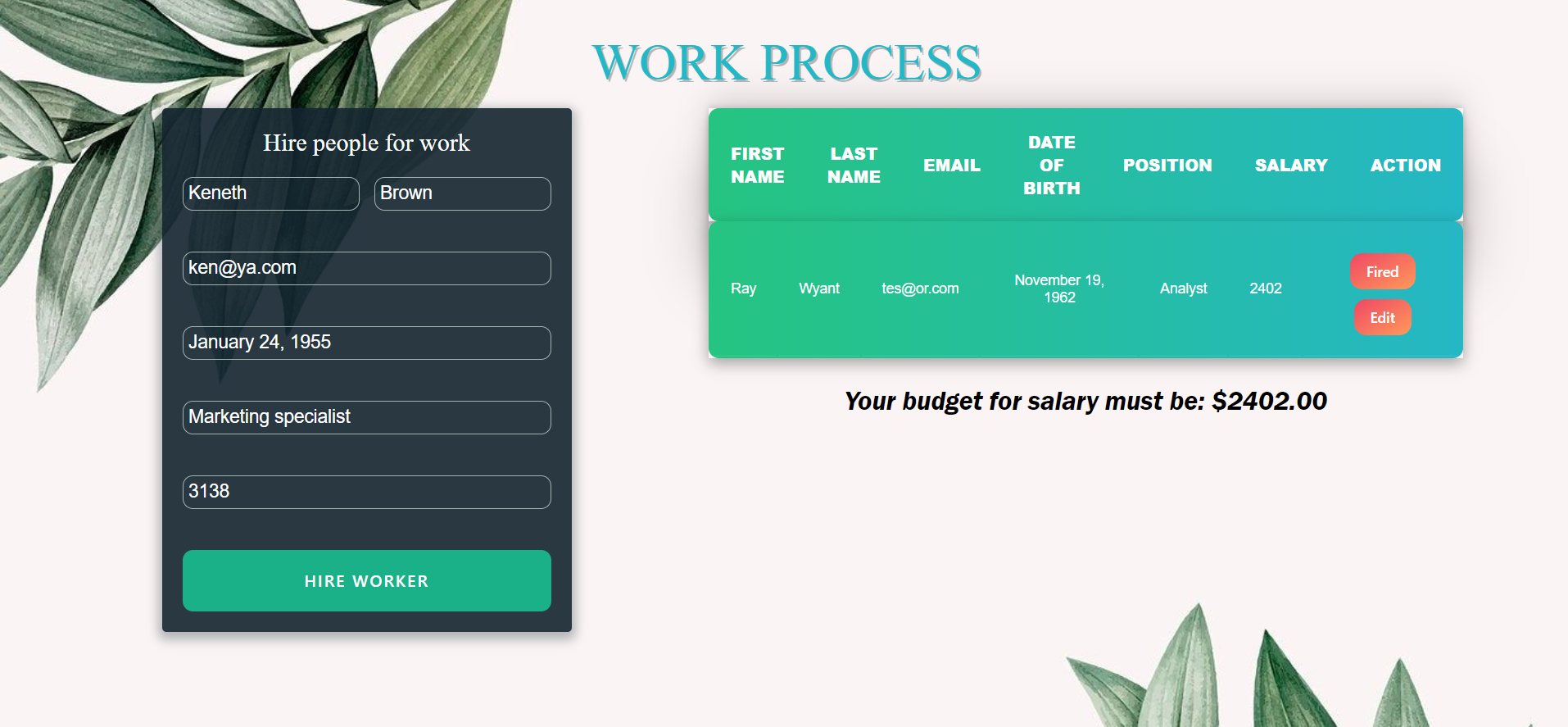
* When the record is savedthe **salary** must be **added** to the **budget** message. The sum should be rounded to the second decimal number:



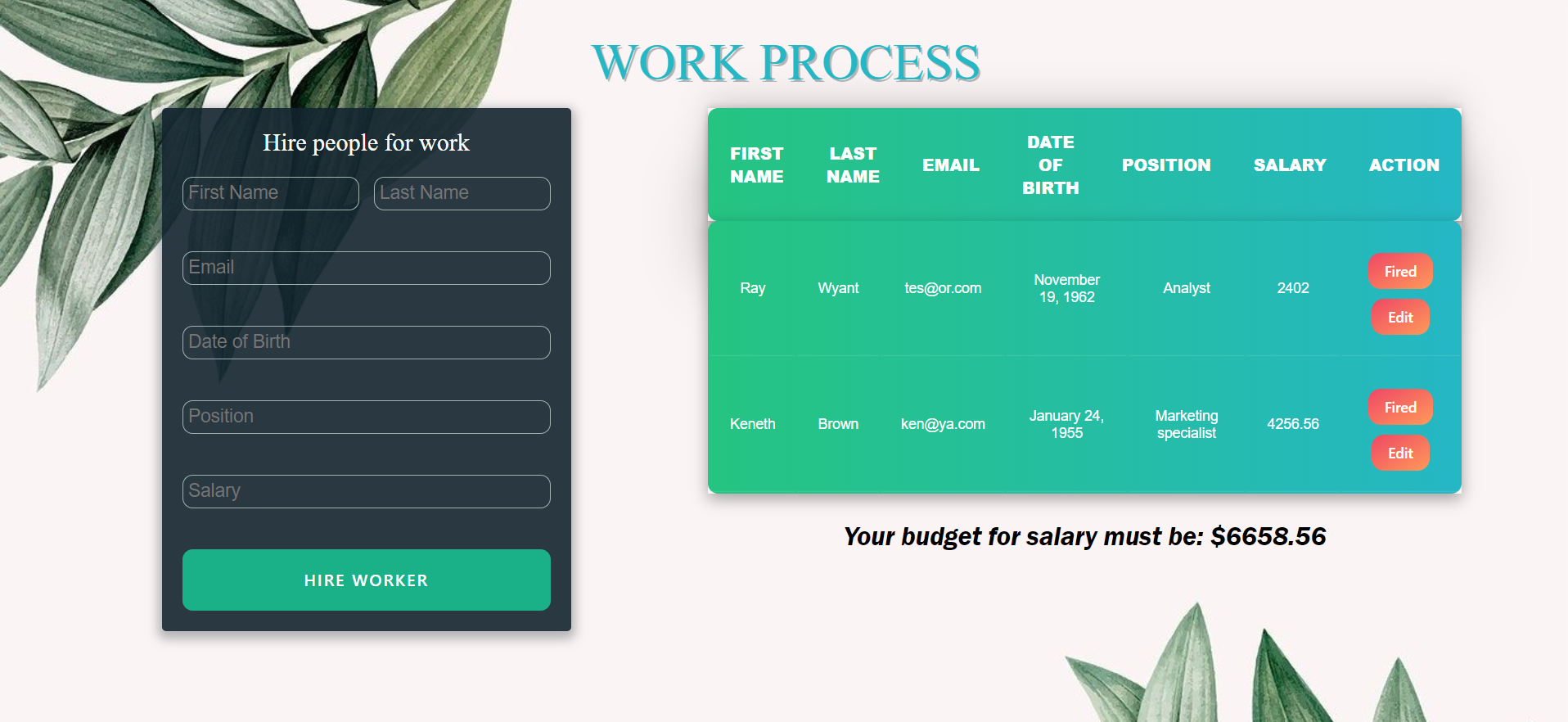


# Edit information for workers

* When the **["Edit"]** button is clicked, the information from the table must be sent to the input’s fields and the record should be deleted from the table.
* The **salary** for the **edit** worker should be **taken out** from the **budget**.

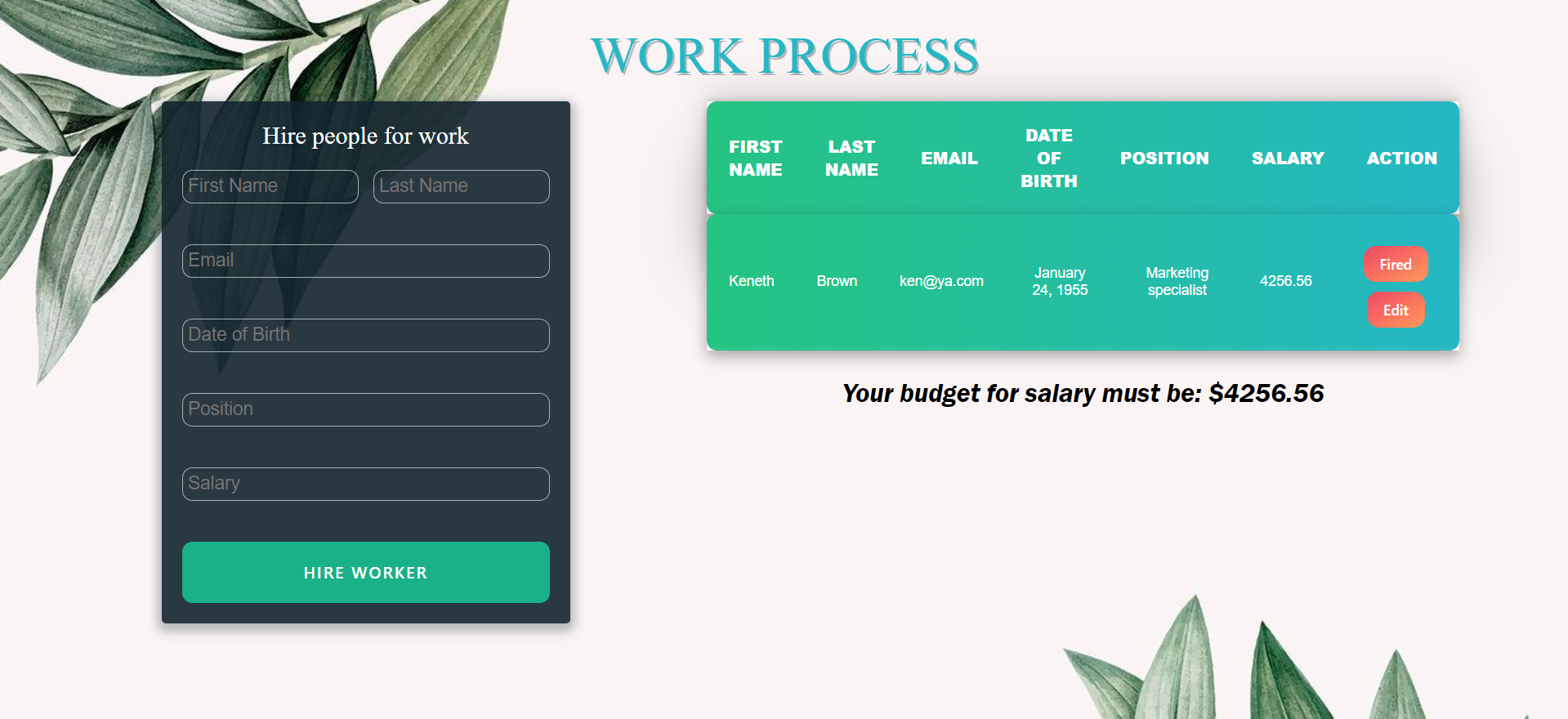


* After editing the information make a new record to the table with updated information.

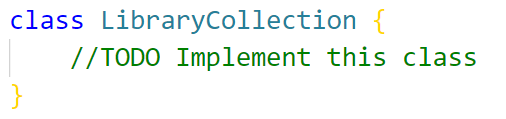


# Fired workers

* When you click the **["Fired"]** button, the record must be **deleted** from the **table**;
* The **salary** for the fired worker should be **taken out** from the **budget**.



## Problem 2. Library Collection



Write a class **LibraryCollection**, which implements the following functionality:

**Functionality**

**Constructor**

Should have these **2** properties:

* **capacity –** Number
* **books –** Array (empty)

**At the initialization of the LibraryCollection class,** the **constructor** accepts the **capacity.**

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

**addBook (bookName, bookAuthor)**

The **bookName** and **bookAuthor** are of type **string**.

* If there's **not enough space in the collection** for the book, **throw an Error**:

**"Not enough space in the collection."**

* Otherwise, this function should add the book, with the properties: **bookName**, **bookAuthor**, **payed**: **default false**, to the books array and **return**:

**"The {bookName}, with an author {bookAuthor}, collect."**

**payBook( bookName )**

* If the book is not found, **throw an Error**:

**"{bookName} is not in the collection."**

* If the book has already paid, **throw an Error**:

**"{bookName} has already been paid."**

* Otherwise, this function set paid to true on the found book and **return**:

**"{bookName} has been successfully paid."**

**removeBook(bookName)**

* If the book is **not found**, throw **an Error:**

**"The book, you're looking for, is not found."**

* If the book **hasn't paid**, throw **an Error**:

**"{bookName} need to be paid before removing from the collection."**

* Otherwise, this function should **remove** the book from the array and **return**:

**"{bookName} remove from the collection."**

**getStatistics(bookAuthor)**

This **method** can be called **with one parameter** or **without** any.

If **no** parameter is provided, the method should **return** the full information of the library

* At the first line:

**"The book collection has { emptySlots } empty spots left."**

* On the lines, display information about each book, **sorted alphabetically ascending**, by their **bookName**:

**"{bookName} == {bookAuthor} - {Has Paid / Not Paid}."**

If the method is called with a parameter for **bookAuthor**:

* **Return** only the information about the book from the given **bookAuthor**:

**"{bookName} == {bookAuthor} - {Has Paid / Not Paid}."**

* If there is **no such author’s book** found, **throw an Error**:

**"{bookAuthor} is not in the collection."**

## Example

|  |
| --- |
| **Input 1** |
| **const *library* = new** LibraryCollection**(2)**  ***console*.log(*library*.addBook('In Search of Lost Time', 'Marcel Proust'));**  ***console*.log(*library*.addBook('Don Quixote', 'Miguel de Cervantes'));**  ***console*.log(*library*.addBook('Ulysses', 'James Joyce'));** |

|  |
| --- |
| **Output 1** |
| The In Search of Lost Time, with an author Marcel Proust, collect.  The Don Quixote, with an author Miguel de Cervantes, collect.  Not enough space in the collection. |

|  |
| --- |
| **Input 2** |
| **const *library* = new** LibraryCollection**(2)**  ***library*.addBook('In Search of Lost Time', 'Marcel Proust');**  ***console*.log(*library*.payBook('In Search of Lost Time'));**  ***console*.log(*library*.payBook('Don Quixote'));** |

|  |
| --- |
| **Output 2** |
| In Search of Lost Time has been successfully paid.  Don Quixote is not in the collection. |

|  |
| --- |
| **Input 3** |
| **const *library* = new** LibraryCollection**(2)**  ***library*.addBook('In Search of Lost Time', 'Marcel Proust');**  ***library*.addBook('Don Quixote', 'Miguel de Cervantes');**  ***library*.payBook('Don Quixote');**  ***console*.log(*library*.removeBook('Don Quixote'));**  ***console*.log(*library*.removeBook('In Search of Lost Time'));** |

|  |
| --- |
| **Output 3** |
| Don Quixote remove from the collection.  In Search of Lost Time need to be paid before removing from the collection. |

|  |
| --- |
| **Input 4** |
| **const *library* = new** LibraryCollection**(2)**  ***console*.log(*library*.addBook('Don Quixote', 'Miguel de Cervantes'));**  ***console*.log(*library*.getStatistics('Miguel de Cervantes'));** |

|  |
| --- |
| **Output 4** |
| The Don Quixote, with an author Miguel de Cervantes, collect.  Don Quixote == Miguel de Cervantes - Not Paid. |

|  |
| --- |
| **Input 5** |
| **const *library* = new** LibraryCollection**(5)**  ***library*.addBook('Don Quixote', 'Miguel de Cervantes');**  ***library*.payBook('Don Quixote');**  ***library*.addBook('In Search of Lost Time', 'Marcel Proust');**  ***library*.addBook('Ulysses', 'James Joyce');**  ***console*.log(*library*.getStatistics());** |

|  |
| --- |
| **Output 5** |
| The book collection has 2 empty spots left.  Don Quixote == Miguel de Cervantes - Has Paid.  In Search of Lost Time == Marcel Proust - Not Paid.  Ulysses == James Joyce - Not Paid. |

# Problem 3. Flower Shop

### Your Task

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

* calcPriceOfFlowers(flower, price, quantity) - A function that accepts three parameters (one string and two numbers):
  + The function returns the multiplies price and quantity;
  + There is a need for validation of the input, a flower, a price and a quantity mаy not always be valid. In case of submitted invalid parameters, **throw** an error **"Invalid input!";**
  + The result is rounded to the second digits after the decimal point.
* checkFlowersAvailable(flower, gardenArr) - A function that accepts two parameters (string and array):
  + There is no need for validation for input, you will always be given a string and an array;
  + The array (gardenArr) includes all available flowers (example: ["Rose", "Lily", "Orchid"]);
  + The function checks whether the submitted string **flower** is present in the array **gardenArr**;
  + If present in the array, the function **return: `The ${flower} are available!`**;
  + Otherwise, the function **return: `The ${flower} are sold! You need to purchase more!`**.
* sellFlowers(gardenArr, space) - A function that accepts two parameters (array and number):
  + The gardenArr array will store the names of flowers(["Rose", "Lily", "Orchid"]);
  + You must **remove** an **element** from the array that is located on the **space** specified as a parameter;
  + There is a need for validation for the input, an array and space may not always be valid. In case of submitted **invalid** parameters, **throw** an error **"Invalid input!"**;
  + Finally, **return** the changed array of flowers as a string, joined by **" / "**.

### JS Code

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

|  |
| --- |
| flowerShop.js |
| const flowerShop = {  calcPriceOfFlowers(flower, price, quantity) {  if (typeof flower != 'string' || !Number.isInteger(price) || !Number.isInteger(quantity)) {  throw new Error('Invalid input!');  } else {  let result = price \* quantity;  return `You need $${result.toFixed(2)} to buy ${flower}!`;  }  }  checkFlowersAvailable(flower, gardenArr) {  if (gardenArr.indexOf(flower) >= 0) {  return `The ${flower} are available!`;  } else {  return `The ${flower} are sold! You need to purchase more!`;  }  },  sellFlowers(gardenArr, space) {  let shop = [];  let i = 0;  if (!Array.isArray(gardenArr) || !Number.isInteger(space) || space < 0 || space >= gardenArr.length) {  throw new Error('Invalid input!');  } else {  while (gardenArr.length > i) {  if (i != space) {  shop.push(gardenArr[i]);  }  i++  }  }  return shop.join(' / ');  }  } |