# JS Advanced: Exam Preparation 1

**Link to contest**: <https://judge.softuni.org/Contests/3395/JS-Advanced-Exam>

# Problem 1. Mails Delivery

**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**.

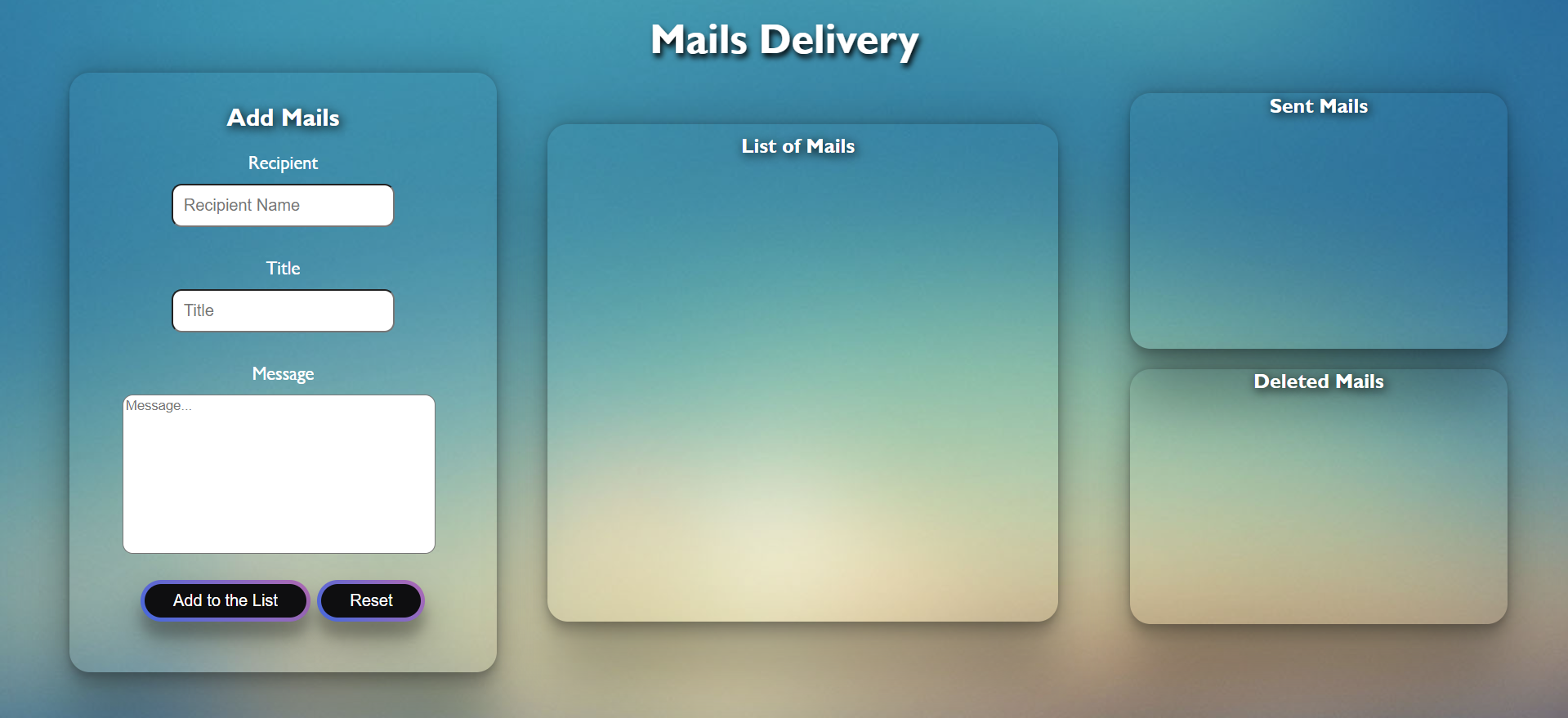
The following actions are **NOT** supported:

* **.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).



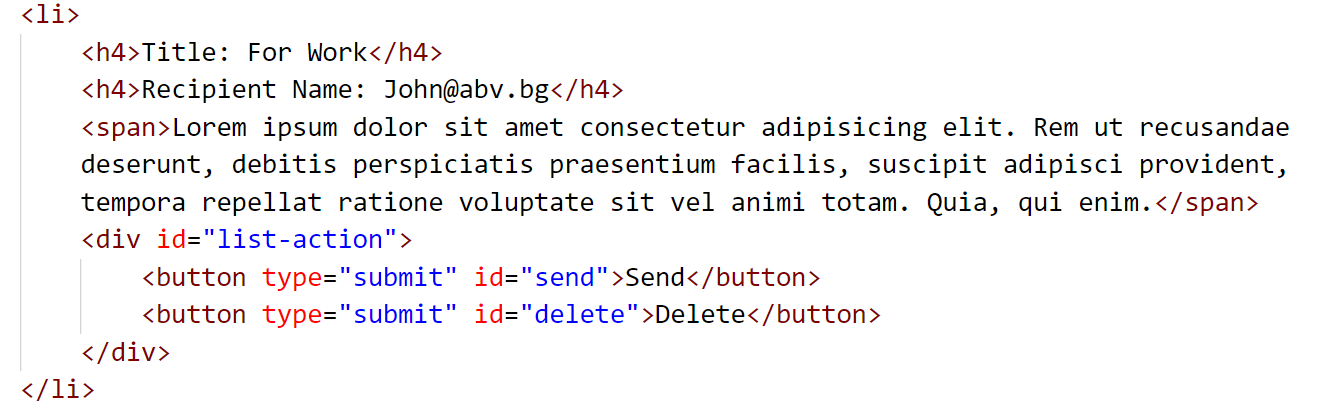
**Your Task**

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

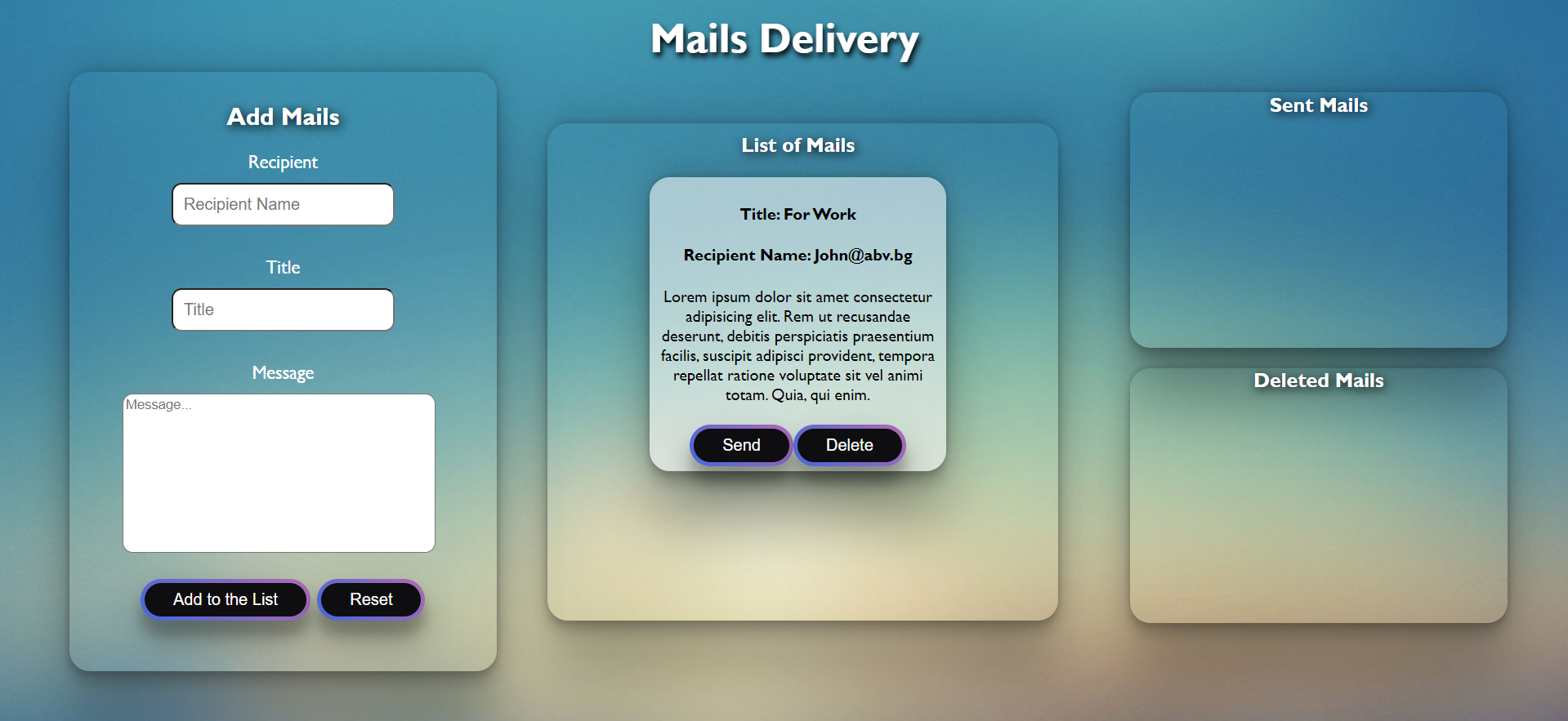
# Getting the information from the form



* **All fields** are **non**-**empty** **strings**. If any of them are empty, the program should not do anything.
* When you click the [**"**Add to the List**"**] button, the information from the input fields must be added to the ul in div with class ="list-mails", and inputs must be cleared. The structure must be:



* The Title and Recipient Name must be saved in the h4 tag and the message in the span tag.
* The Buttons - Send and Delete - are in div with id="list-action".



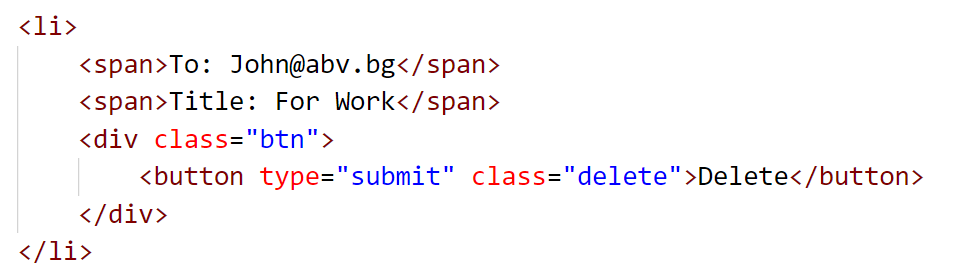
* When you click **["Reset"]**, the information from the inputs should be **cleared**, without any other changes.

# Send Mails

* When the **["Send"]** button is clicked, the information must be sent to the **Sent Mails** and the **li** tag should be deleted from the **List of Mails**.



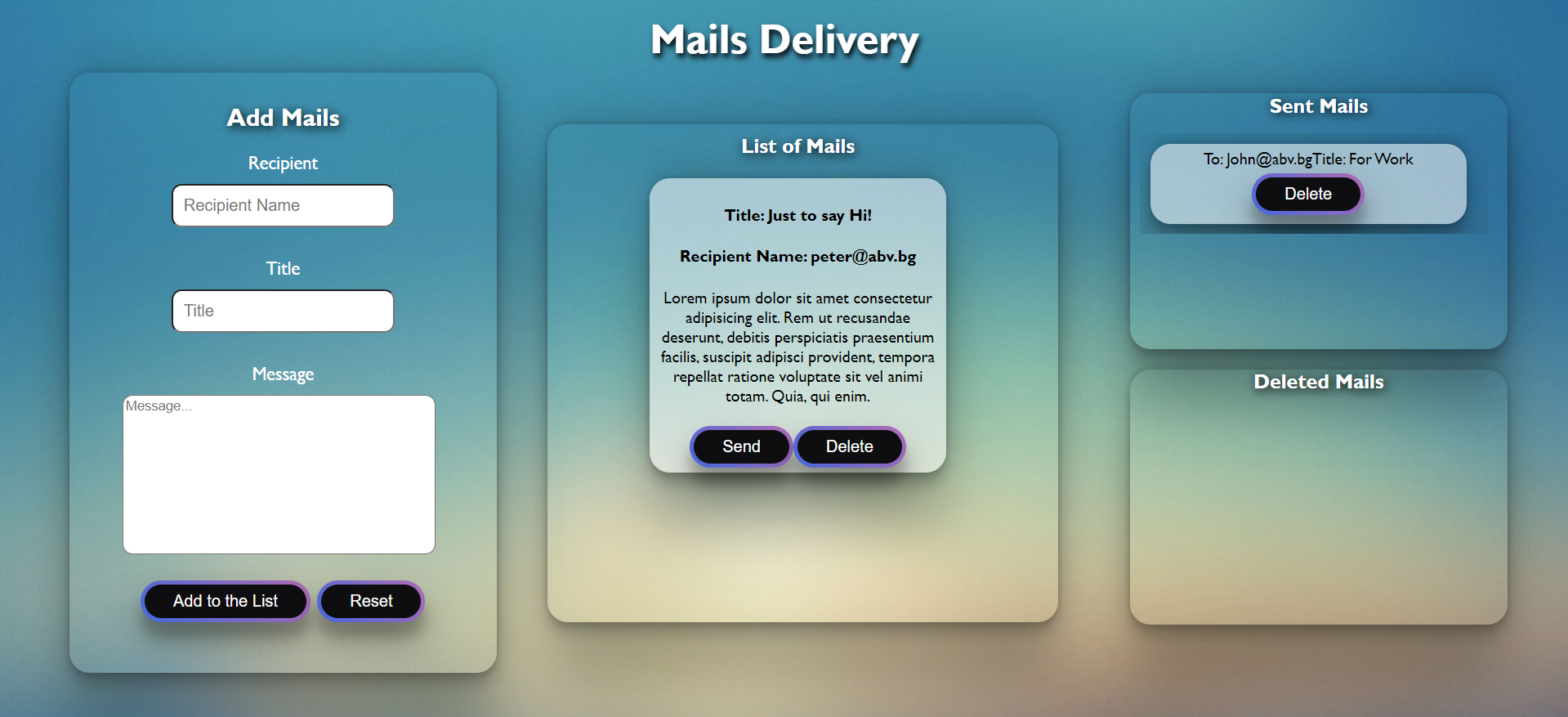
* The **structure** must be:

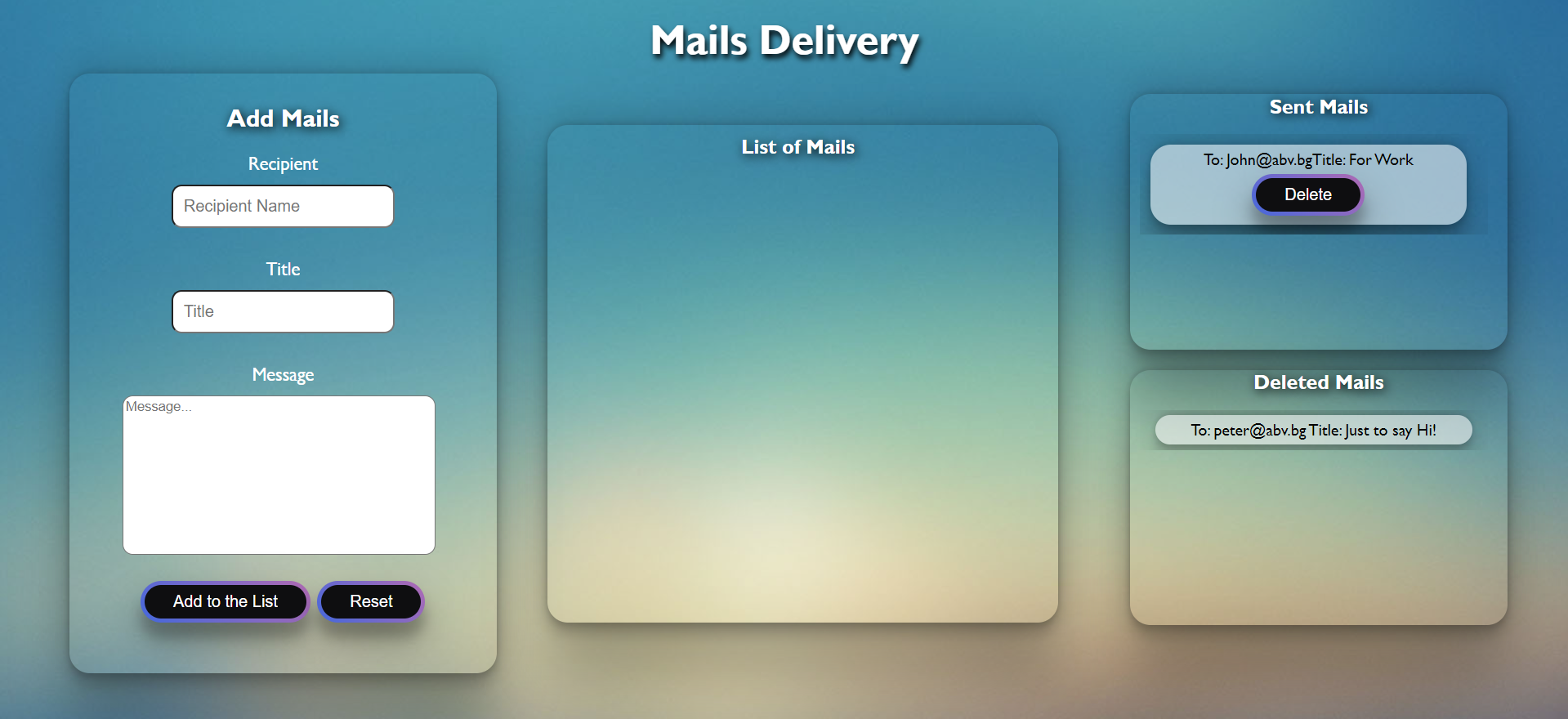


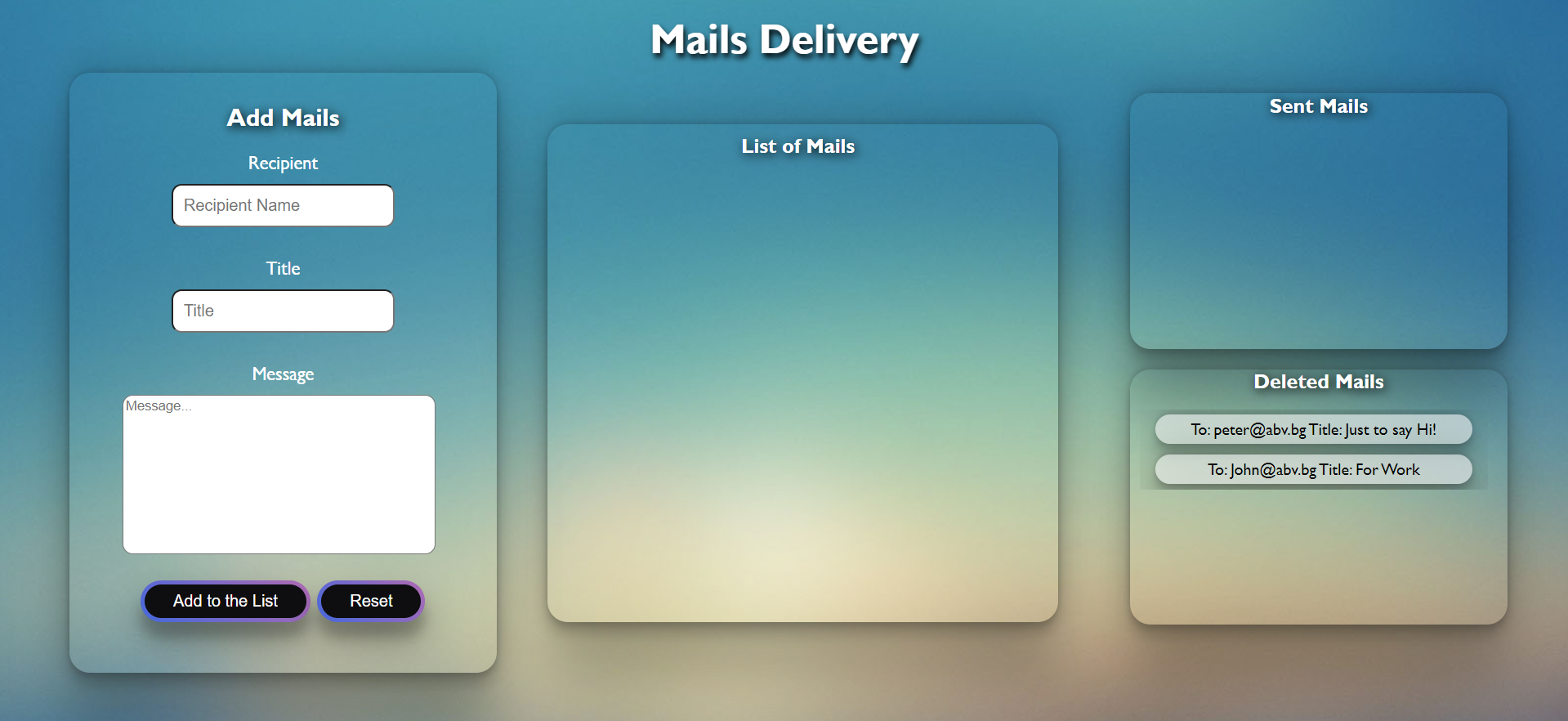
* The Title and Recipient Name must both be saved in the span tag.
* The Button, Delete, is in div with class="btn".

# Delete Mails

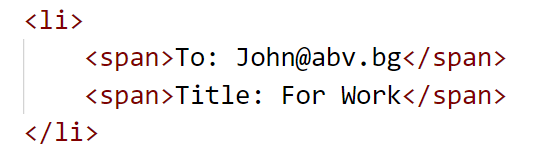
* When you click the **["Delete"]** button, the information on either the **List of mails** or the **Sent mails** div, the information must be sent to the **Delete Mails** and the record should be deleted from the **List of Mails** or **Sent Mails**.



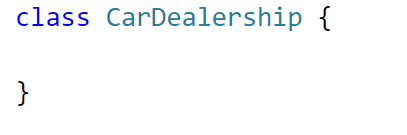




* The **structure** must be:



## Problem 2. Car Dealership



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

**Functionality**

**Constructor**

Should have these **4** properties:

* **name –** String
* **availableCars –** Array
* **soldCars –** Array
* **totalIncome –** **default: 0**

**At the initialization of the CarDealership class,** the **constructor** accepts the **name.** The **totalIncome** has a **default value of 0!** The rest of the properties must be **empty!**

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

**addCar (model, horsepower, price, mileage) -** This method should **add a new car** to the dealership. The method accepts **4 arguments:**

* If any of the following requirements is **NOT fulfilled**, an **error** with the following message should be **thrown**: **"Invalid input!"**
  + **Model** – non-empty string;
  + **Horsepower** – positive integer number;
  + **Price** – positive number;
  + **Mileage** – positive number.

**Hint**: Zero is also a positive number.

* Otherwise, you should **add the car**, with properties: **{model, horsepower, price, mileage}** to the **availableCars** array and **return**:

**"New car added: {model} - {horsepower} HP - {mileage} km - {price}$"**

* When **returning** the result, the **Mileage and Price** mustbe **rounded to the second decimal point!**

**sellCar (model, desiredMileage) –** This method should **search for a car** with the given **model** in the **availableCars** array, and then **sell** it. Accepts **2 arguments**.

* If a car with the given **model** cannot be found, an error with the following message should be **thrown**:

**"{model} was not found!"**

* If you **find the car with the given model**, you should look up its **mileage**. The person who wants to buy it has a simple request. He is looking for a car with a **mileage** that is **less or equal** to his **desired mileage**. To ensure the sale of the car you must make a bargain:
  + If the **found** car’s mileage is **less than or equal to** the **desiredMileage** – the price stays the same!
  + If the **difference** between the **car’s mileage** and the **desiredMileage** is less or equal to **40.000 km** – the price gets **deducted by 5%**!
  + If the **difference** between the **car’s mileage** and the **desiredMileage** is more than **40.000 km** – the price gets **deducted by 10%**!
* You should **remove** the car from the **availableCars** array and **add** it to the **soldCars** array in the following format: **{model, horsepower, soldPrice}**
* Finally, you must add the **soldPrice** to the **totalIncome** and return:

**"{model} was sold for {soldPrice}$"**

**Note: soldPrice** must be **rounded** to the second decimal point!

**currentCar ()** - This method should just return all available cars separated by a new line in format:

**"-Available cars:**

**---{model} - {horsepower} HP - {mileage} km - {price}$**

**---{model} - {horsepower} HP - {mileage} km - {price}$"**

**Note: mileage** and **price** mustbe **rounded** to the second decimal point!

* If there are **no available** cars, just return:

**"There are no available cars"**

**salesReport (criteria)** – This method accepts 1 argument. It should **sort** the sold cars, **based on a given criterion**. The two possible criteria are – **"horsepower"** or **"model"**

* If the given criteria **do not match** either of the possible criteria, an **error** with the following message should be **thrown**:

**"Invalid criteria!"**

* If the given criteria is **"horsepower"** – the sold **cars** must be **sorted** by their **horsepower** in **descending** **order**;
* If the given criteria is **"model"** – the sold cars must be **sorted alphabetically** by their **model**;
* Finally, **return** **all sorted** sold cars **separated** by **a new line** in format:

**"-{dealershipName} has a total income of {totalIncome}$**

**-{soldCarsCount} cars sold:**

**---{model} - {horsepower} HP - {price}$**

**---{model} - {horsepower} HP - {price}$"**

**…**

**Note: totalIncome and price must be rounded to the second decimal point!**

## Example

|  |
| --- |
| **Input 1** |
| **let dealership = new CarDealership('SoftAuto');**  **console.log(dealership.addCar('Toyota Corolla', 100, 3500, 190000));**  **console.log(dealership.addCar('Mercedes C63', 300, 29000, 187000));**  **console.log(dealership.addCar('', 120, 4900, 240000));** |

|  |
| --- |
| **Output 1** |
| New car added: Toyota Corolla - 100 HP - 190000.00 km - 3500.00$  New car added: Mercedes C63 - 300 HP - 187000.00 km - 29000.00$  Uncaught Error Error: Invalid input! |

|  |
| --- |
| **Input 2** |
| **let dealership = new CarDealership('SoftAuto');**  **dealership.addCar('Toyota Corolla', 100, 3500, 190000);**  **dealership.addCar('Mercedes C63', 300, 29000, 187000);**  **dealership.addCar('Audi A3', 120, 4900, 240000);**  **console.log(dealership.sellCar('Toyota Corolla', 230000));**  **console.log(dealership.sellCar('Mercedes C63', 110000));** |

|  |
| --- |
| **Output 2** |
| Toyota Corolla was sold for 3500.00$  Mercedes C63 was sold for 26100.00$ |

|  |
| --- |
| **Input 3** |
| **let dealership = new CarDealership('SoftAuto');**  **dealership.addCar('Toyota Corolla', 100, 3500, 190000);**  **dealership.addCar('Mercedes C63', 300, 29000, 187000);**  **dealership.addCar('Audi A3', 120, 4900, 240000);**  **console.log(dealership.currentCar());** |

|  |
| --- |
| **Output 3** |
| -Available cars:  ---Toyota Corolla - 100 HP - 190000.00 km - 3500.00$  ---Mercedes C63 - 300 HP - 187000.00 km - 29000.00$  ---Audi A3 - 120 HP - 240000.00 km - 4900.00$ |

|  |
| --- |
| **Input 4** |
| **let dealership = new CarDealership('SoftAuto');**  **dealership.addCar('Toyota Corolla', 100, 3500, 190000);**  **dealership.addCar('Mercedes C63', 300, 29000, 187000);**  **dealership.addCar('Audi A3', 120, 4900, 240000);**  **dealership.sellCar('Toyota Corolla', 230000);**  **dealership.sellCar('Mercedes C63', 110000);**  **console.log(dealership.salesReport('horsepower'));** |

|  |
| --- |
| **Output 4** |
| -SoftAuto has a total income of 29600.00$  -2 cars sold:  ---Mercedes C63 - 300 HP - 26100.00$  ---Toyota Corolla - 100 HP - 3500.00$ |

# Problem 3. Rent Car

### Your Task

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

* searchCar(shop, model) - A function that accepts two parameters (one array and one string):
  + The function checks whether the submitted string **model** is present in the **shop (**example: **["Volkswagen", "BMW", "Audi"])**, and return number of matching elements and the model of the car in the message: **`There is ${findModel.length} car of model ${model} in the catalog!`**;
  + There is a need for validation of the input, a shop and a model mаy not always be valid. In case of submitted invalid parameters, **throw** an error **"Invalid input!";**
  + If there are no matching elements, the function **throw** an error: **'There are no such models in the catalog!'**
* **calculatePriceOfCar**(model, days) - A function that accepts two parameters (string and number):
  + There is a need for validation of the input, a **model,** and **days** mаy not always be valid. In case of submitted invalid parameters, **throw** an error **"Invalid input!";**
  + The function returns the model and the price it will cost for renting a car for the given days: **`You choose ${model} and it will cost $${cost}!`**;
  + Otherwise, if there is no such model, the function **throw** an error: **'No such model in the catalog!'.**
* **checkBudget**(**costPerDay, days, budget**) - A function that accepts three parameters (numbers):
  + There is a need for validation of the input, a **costPerDay**, **days, and a budget** mаy not always be valid. In case of submitted invalid parameters, **throw** an error **"Invalid input!";**
  + If the budget is bigger or equal to cost, function return: **`You rent a car!`;**
  + If the budget is less than cost, the function returns the message: **'You need a bigger budget!'.**

### JS Code

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

|  |
| --- |
| rentCar.js |
| const rentCar = {  searchCar(shop, model) {  let findModel = [];  if (Array.isArray(shop) && typeof model == 'string') {  for (let i = 0; i < shop.length; i++) {  if (model == shop[i]) {  findModel.push(shop[i]);  }  }  if (findModel.length !== 0) {  return `There is ${findModel.length} car of model ${model} in the catalog!`;  } else {  throw new Error('There are no such models in the catalog!')  }  } else {  throw new Error('Invalid input!')  }  },  calculatePriceOfCar (model, days) {  let catalogue = {  Volkswagen: 20,  Audi: 36,  Toyota: 40,  BMW: 45,  Mercedes: 50  };  if (typeof model == 'string' && Number.isInteger(days)) {  if (catalogue.hasOwnProperty(model)) {  let cost = catalogue[model] \* days;  return `You choose ${model} and it will cost $${cost}!`  } else {  throw new Error('No such model in the catalog!')  }  } else {  throw new Error('Invalid input!')  }  },  checkBudget(costPerDay, days, budget) {  if (!Number.isInteger(costPerDay) || !Number.isInteger(days) || !Number.isInteger(budget)) {  throw new Error('Invalid input!');  } else {  let cost = costPerDay \* days;  if (cost <= budget) {  return `You rent a car!`  } else {  return 'You need a bigger budget!'  }  }  }  } |