JS Front-end: Exam Preparation 1

Link to contest: https://judge.softuni.org/Contests/4138

Problem 1 – MotoGP Race

You are in the middle of a highly competitive MotoGP race. Let's determine who the best rider is! The input format and the actions the riders can perform are described below.

The first line of the input should contain an integer 'n' - the number of riders participating in the race. The next 'n' lines should provide the details of each rider, including their fuel capacity and current position in the race. Each rider's details should be separated by a pipe (|) and follow this format:

"{rider}|{fuel capacity}|{position}"

Note: A rider's fuel capacity can have a maximum **value** of 100%.

After adding the riders, you can start the race. You will receive different commands, each on a new line and separated by " - ", until the "**Finish**" command is given.

The actions the riders can perform during the race are as follows:

"StopForFuel – {rider} – {minimum fuel} – {changed position}"

- If the rider has **less** than minimum fuel, he needs to a make pit stop. **Print** this message:
  - **"{rider} stopped to refuel but lost his position, now he is {changed position}."**
- If the rider has enough fuel **print**:
  - **"{rider} does not need to stop for fuel!"**

"Overtaking – {rider 1} – {rider 2}"

- If rider 1 is positioned to the left of rider 2, it means that rider 1 is ahead of rider 2 in the race, **swap** the **position** of the two riders. Then, **print** the following:

  "{rider 1} overtook {rider 2}!"

"EngineFail – {rider} – {laps left}"

- If the rider's engine fails, remove him from the race and **print**:

  "{rider} is out of the race because of a technical issue, {laps left} laps before the finish."

Input

- On the first line of the standard input, you will receive an integer **n**
- On the following **n** lines, the riders themselves will follow with their **fuel capacity** and **position** in percentages**,** separated by a pipe in the following format
- You will be receiving different **commands**, each on a new line, separated by **" – "**, until the **"Finish"** command is given

Output

- Every command should **print its own template sentence**, after that **print** all riders who are **finished** the race, in the following format:

```
"{rider}
  Final position: {position}"
```

Constraints

- The **names** of the riders will **always** be **unique**.
- All given **commands** will be **valid**.

Examples

Input	Output

`(["3",`
`"Valentino Rossi|100|1",`
`"Marc Marquez|90|2",`
`"Jorge Lorenzo|80|3",`
`"StopForFuel - Valentino Rossi - 50 - 1",`
`"Overtaking - Marc Marquez - Jorge Lorenzo",`
`"EngineFail - Marc Marquez - 10",`
`"Finish"])`
	`Valentino Rossi does not need to stop for fuel!`
	`Marc Marquez overtook Jorge Lorenzo!`
	`Marc Marquez is out of the race because of a technical issue, 10 laps before the finish.`
	`Valentino Rossi`
	`  Final position: 1`
	`Jorge Lorenzo`
	`  Final position: 2`

Input	**Output**

`(["4",`
`"Valentino Rossi|100|1",`
`"Marc Marquez|90|3",`
`"Jorge Lorenzo|80|4",`
`"Johann Zarco|80|2",`
`"StopForFuel - Johann Zarco - 90 - 5",`
`"Overtaking - Marc Marquez - Jorge Lorenzo",`
`"EngineFail - Marc Marquez - 10",`
`"Finish"])`
	`Johann Zarco stopped to refuel but lost his position, now he is 5.`
	`Marc Marquez overtook Jorge Lorenzo!`
	`Marc Marquez is out of the race because of a technical issue, 10 laps before the finish.`
	`Valentino Rossi`
	`  Final position: 1`
	`Jorge Lorenzo`
	`  Final position: 3`
	`Johann Zarco`
	`  Final position: 5`

Problem 2. Scholarship

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**.

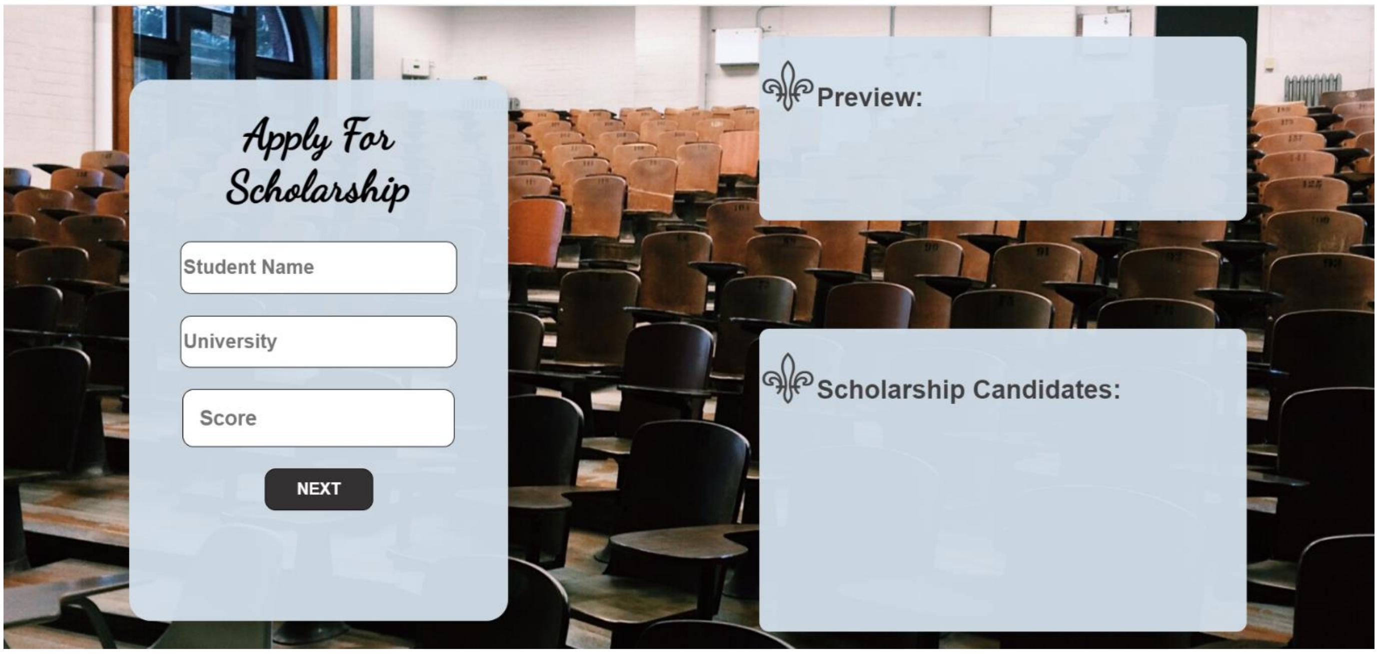
The following actions are **NOT** supported:

- `.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()` (use only `appendChild()`)
- `prepend()`
- `replaceWith()`
- `replaceAll()`
- `closest()`
- `replaceChildren()`

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 **Scholarship** work as expected:

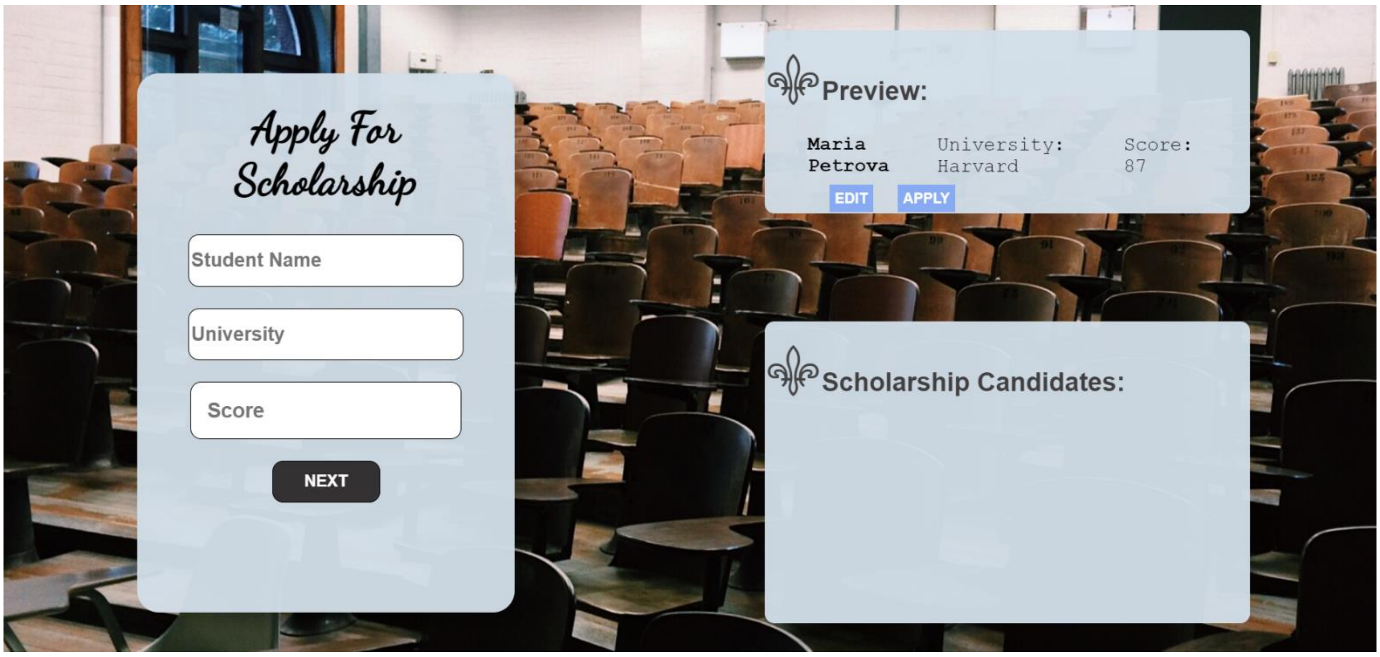
- **Student Name, University,** and **Score** should be **non-empty strings**. If any of them are empty, the program should not do anything.

1. Getting the information from the form

When you click the **[Next]** button, the information from the input fields must be added to the **`<ul>`** with the **id `"preview-list"`**, **[Next]** button must be **disabled** and **the input fields should be cleared**.

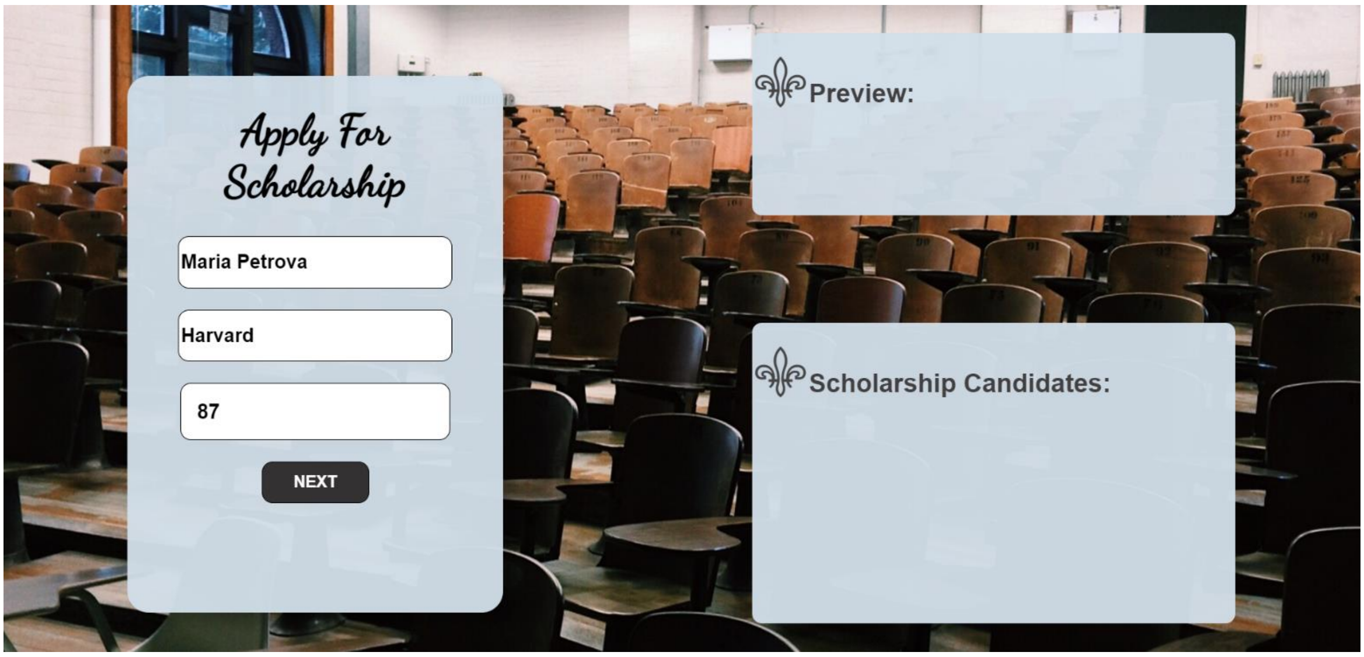
The HTML structure should look like this:

```
▼<ul id="preview-list">
  ▼<li class="application">
    ▼<article> flex
       <h4>Maria Petrova</h4>
       <p>University: Harvard</p>
       <p>Score: 87</p>
     </article>
     <button class="action-btn edit">edit</button>
     <button class="action-btn apply">apply</button>
   </li>
 </ul>
```

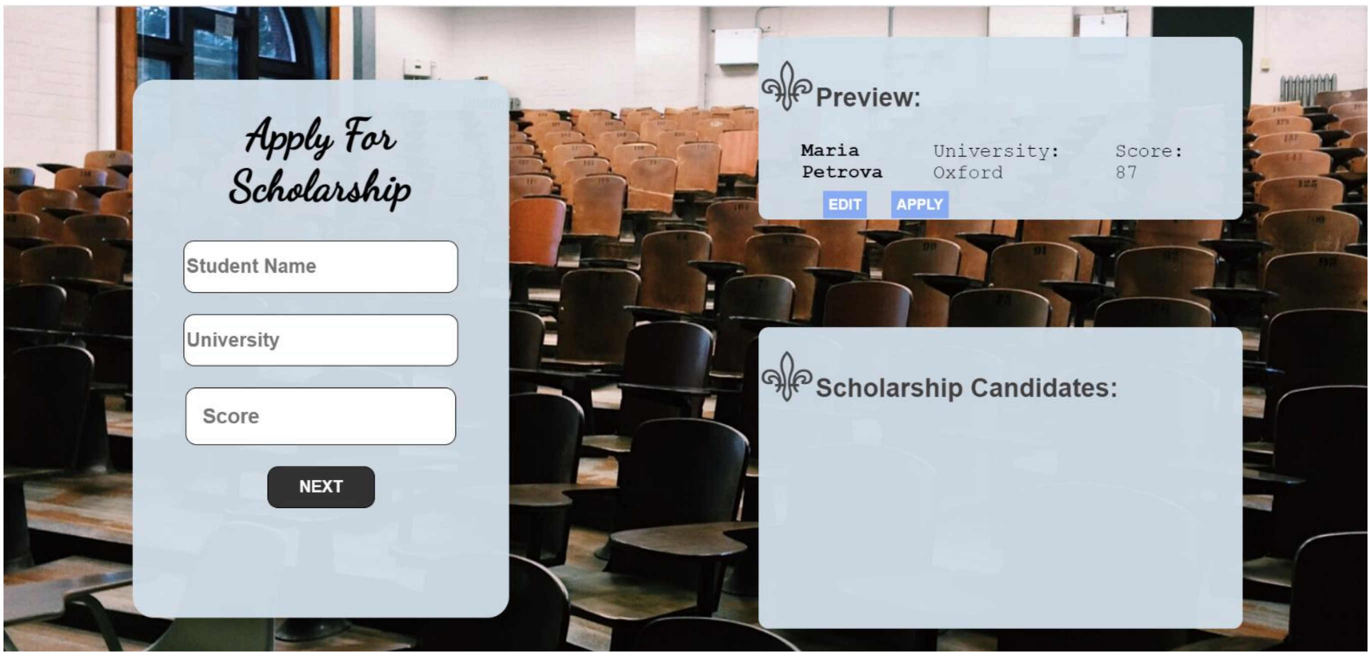


2. Edit information

When the **[Edit]** button is clicked, the information from the post must be sent to the input fields on the left side and the record should be deleted from the **`<ul>` `"preview-list"`** and **[Next]** button must be **enabled** again.

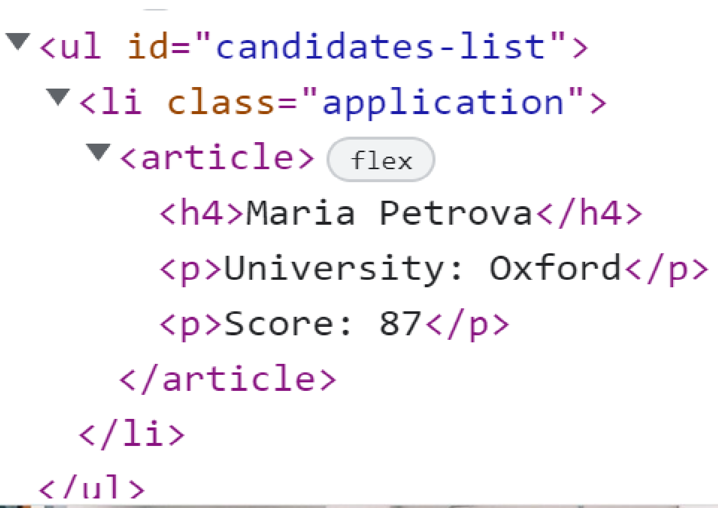


After editing the information, add a new item to the **`<ul>`** with the updated information.

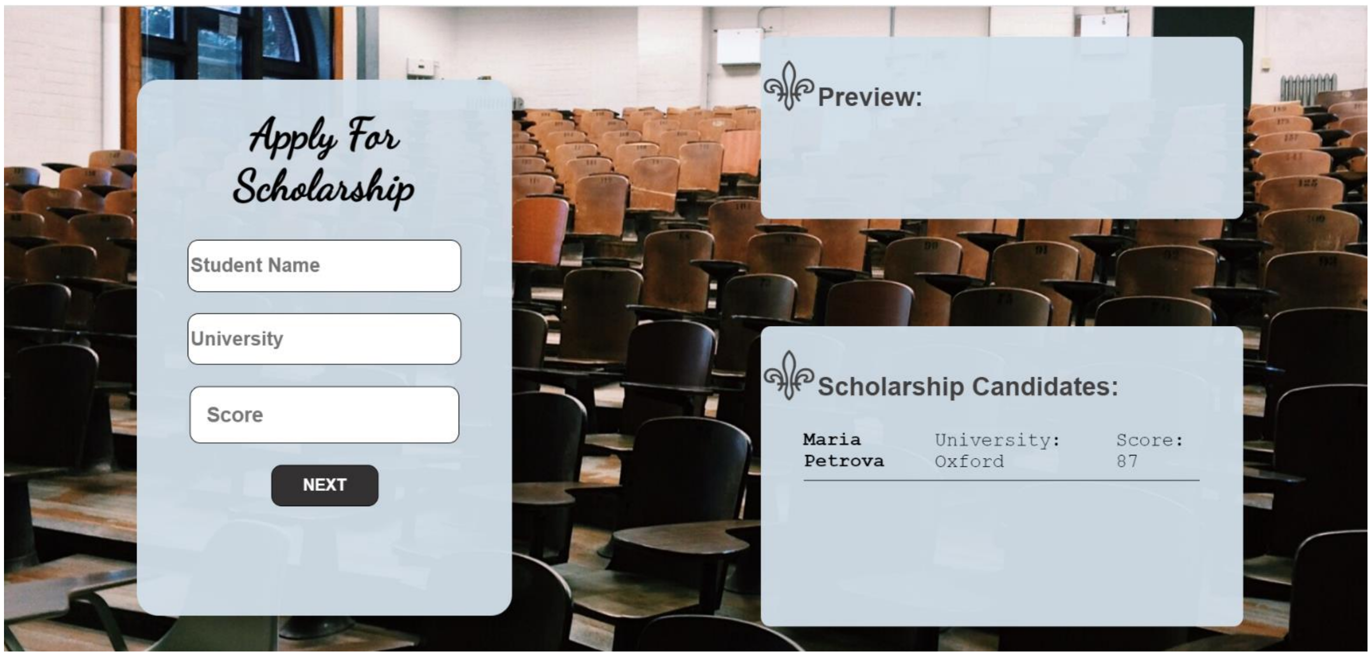


3. Apply for Scholarship

When you click the **[Apply]** button, the task must be **deleted** from the `<ul>` with **id** `"preview-list"` and appended to the `<ul>` with **id** `"candidates-list"`.
The **buttons [Edit]** and **[Apply]** should be removed from the `<li>` element and **[Next]** button must be **enabled** again.



```html
▼<ul id="candidates-list">
  ▼<li class="application">
    ▼<article> flex
        <h4>Maria Petrova</h4>
        <p>University: Oxford</p>
        <p>Score: 87</p>
      </article>
    </li>
  </ul>
```



Maria Petrova   University: Oxford   Score: 87

Submission

Submit only your **solve()** function.

Problem 3 – Vacation Schedule

Working with Remote Data

For the solution of some of the following tasks, you will need to use an up-to-date version of the **local REST service** provided in the lesson's resources archive. You can [read the documentation here](#).

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()` (use only `appendChild()`)
- `prepend()`
- `replaceWith()`
- `replaceAll()`
- `closest()`
- `replaceChildren()`

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

Requirements

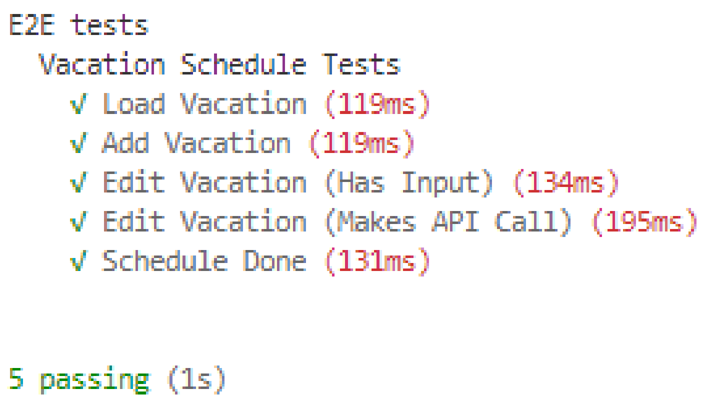
Write a JS program that can load, create, remove and edit a list of scheduled vacations. You will be given an HTML template to which you must bind the needed functionality.

First, you need to install all dependencies using the **`npm install`** command

Then, you can start the front-end application with the **`npm start`** command

You also must start the ***server.js*** file in the ***server*** folder using the **`node server.js`** command in another console **(BOTH THE CLIENT AND THE SERVER MUST RUN AT THE SAME TIME)**.

At any point, you can open up another console and run **`npm test`** to test the **current state** of your application. It's preferable for **all of your tests to pass locally** before you submit to the Judge platform, like this:



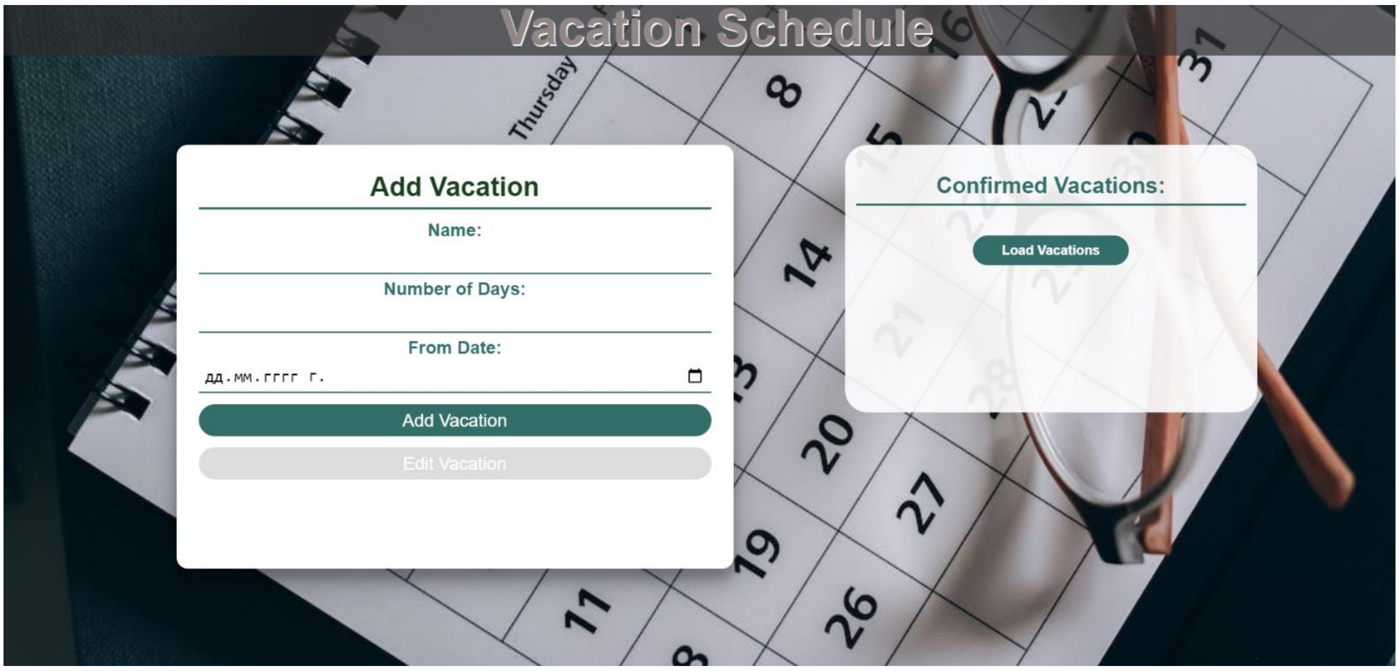
```
E2E tests
  Vacation Schedule Tests
    √ Load Vacation (119ms)
    √ Add Vacation (119ms)
    √ Edit Vacation (Has Input) (134ms)
    √ Edit Vacation (Makes API Call) (195ms)
    √ Schedule Done (131ms)


5 passing (1s)
```

Endpoints

- http://localhost:3030/jsonstore/tasks/
- http://localhost:3030/jsonstore/tasks/:id

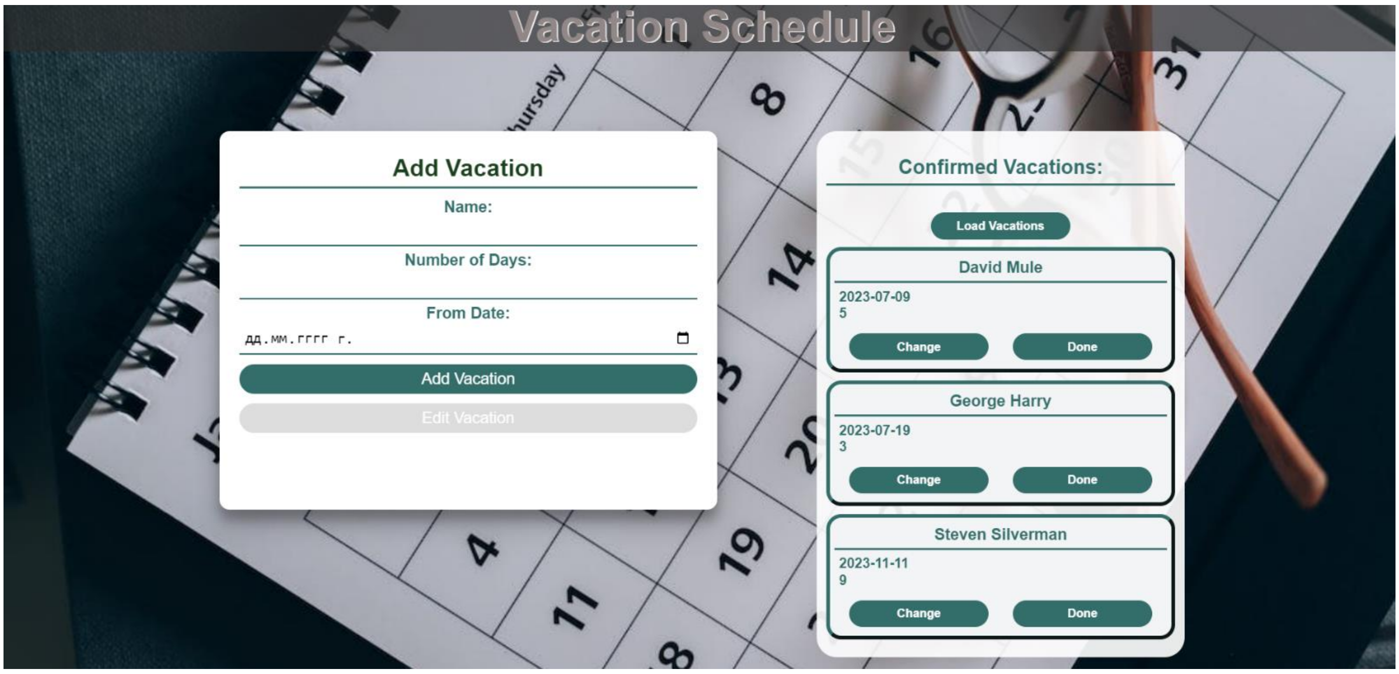
Load Vacations



Clicking the **[Load Vacations]** button should send a **GET** request to the server to fetch **all courses** from your local database. You must add each task to the **`<div>`** with **`id="list"`. [Edit Vacation]** button should be deactivated.

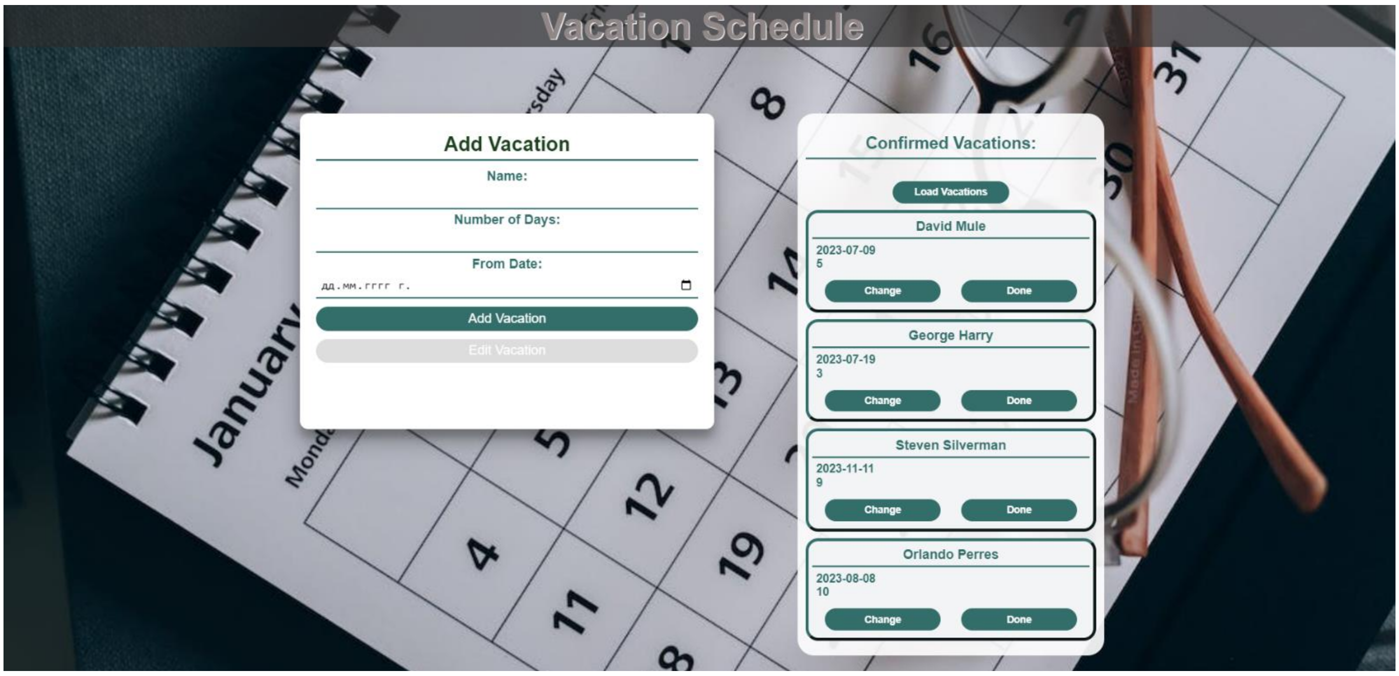
Each vacation has the following **HTML structure**:

```
▼ <div class="container">
    <h2>David Mule</h2>
    <h3>2023-07-09</h3>
    <h3>5</h3>
    <button class="change-btn">Change</button>
    <button class="done-btn">Done</button>
  </div>
```



Add a Vacation

Clicking the **[Add Vacation]** button should send a **POST** request to the server, creating a new scheduled vacation with the **name**, **days count, and starting date** from the input values. After a successful creation, you should send another **GET** request to fetch all the scheduled vacations, including the **newly added one** into the **Confirmed Vacations** column. You should also **clear all the input fields** after the creation!

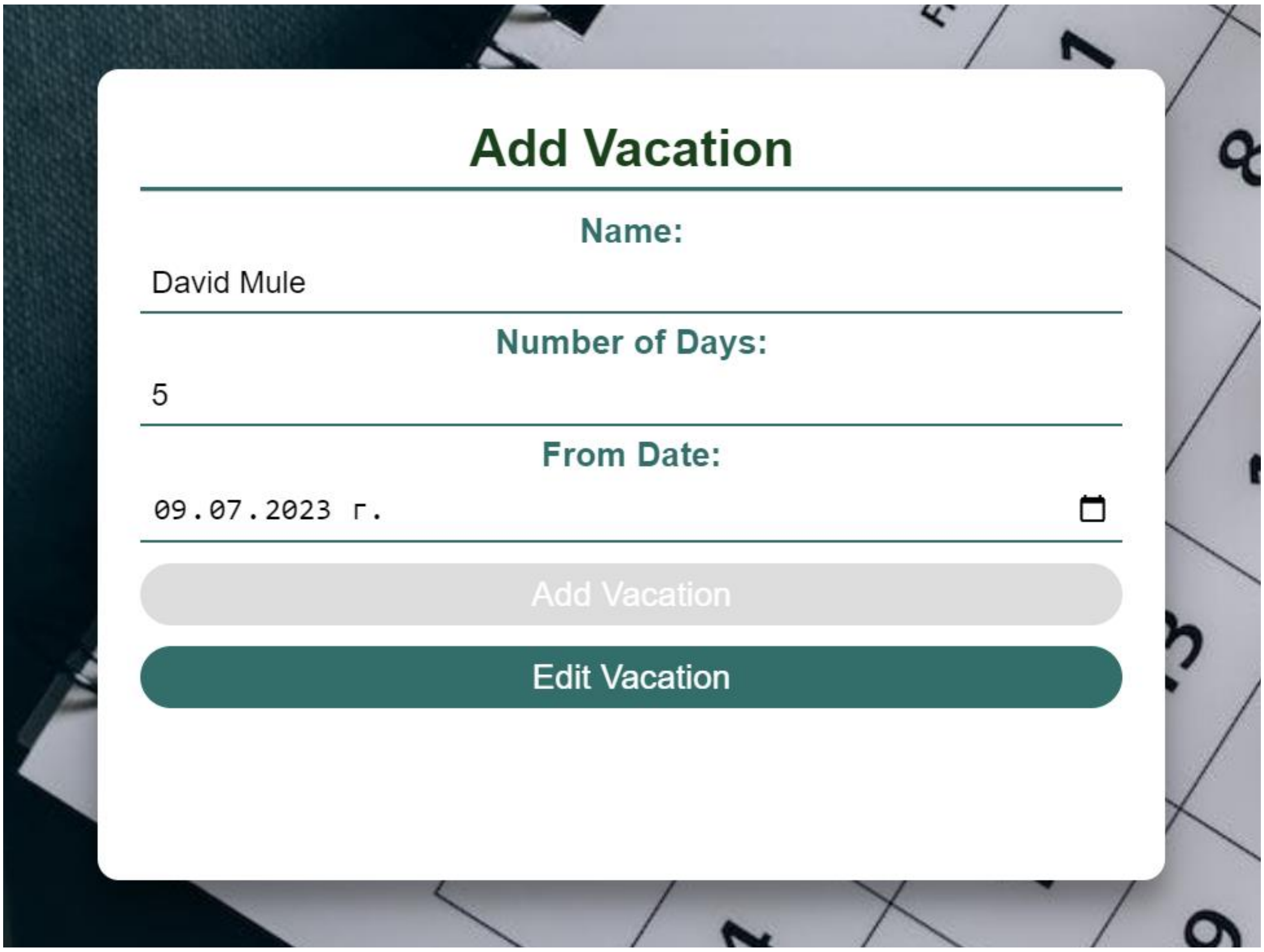


Edit a Vacation

Clicking the **[Change]** button on a record should remove the record from the DOM structure and the information about the task should be populated into the input fields above. The **[Edit Vacation]** button **in the form** should be activated and the **[Add Vacation]** one should be deactivated.

After clicking the [**Edit Vacation]** button in the form, you should send a **PUT** request to the server to **modify the name, number of days and the date** of the changed item. After the successful request, you should **fetch the items**

again and see that the changes have been made. After that, the **[Edit Vacation]** button should be deactivated and the **[Add Vacation]** one should be activated.



Add Vacation

Name:

David Mule

Number of Days:

5

From Date:

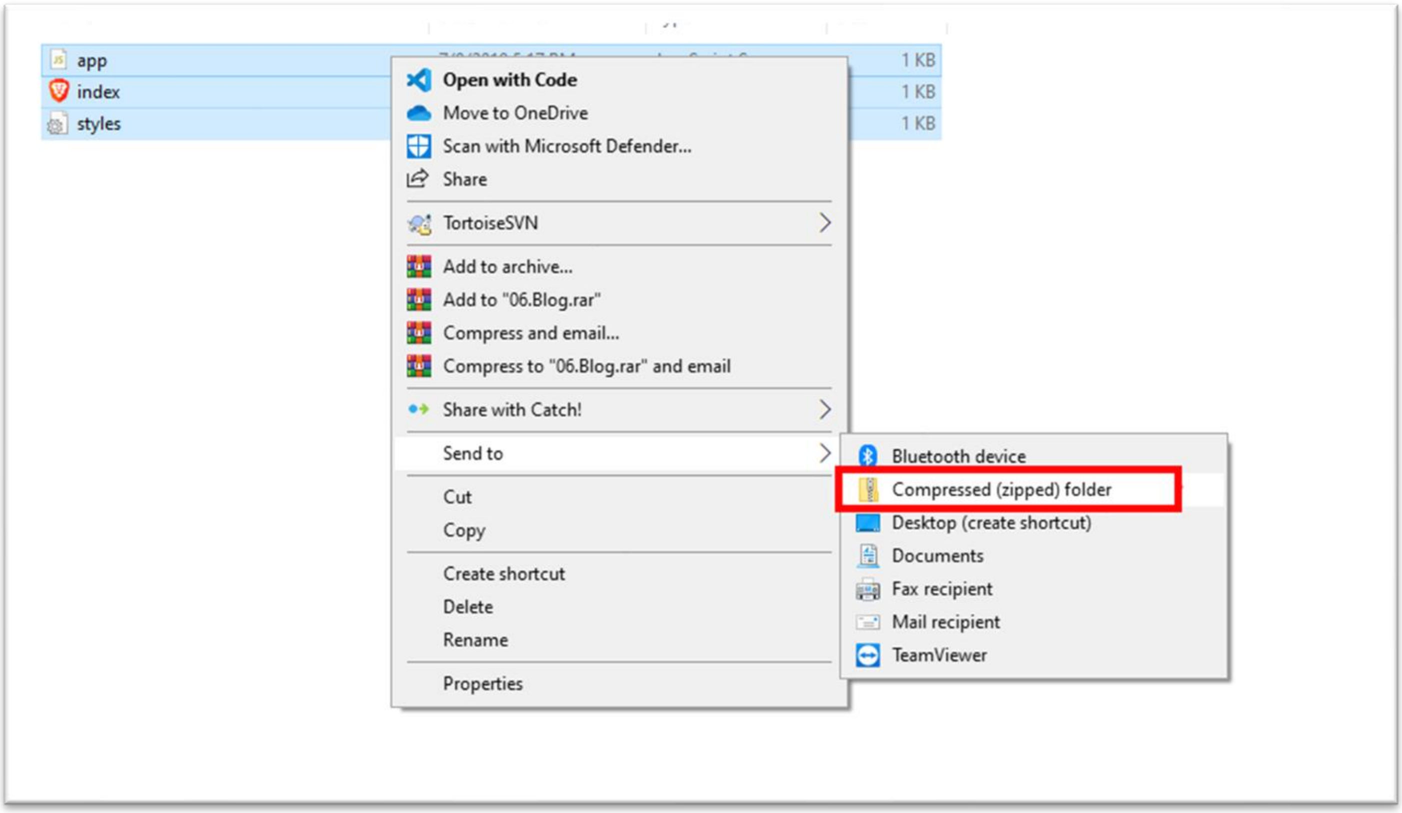
09.07.2023 г.

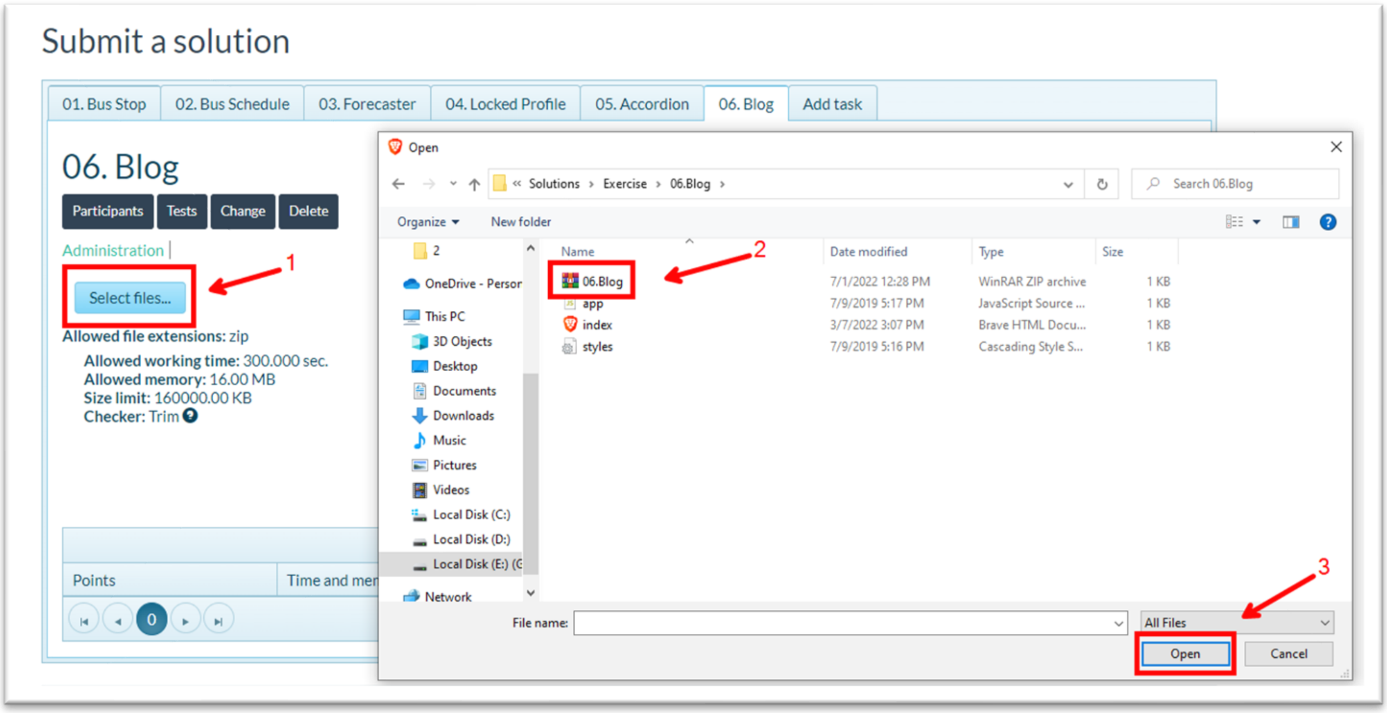
Mark as Done

Clicking the **[Done]** button should send a **DELETE** request to the server and remove the item from your local database. After you've removed it successfully, **fetch** the items **again**.

Submitting Your Solution

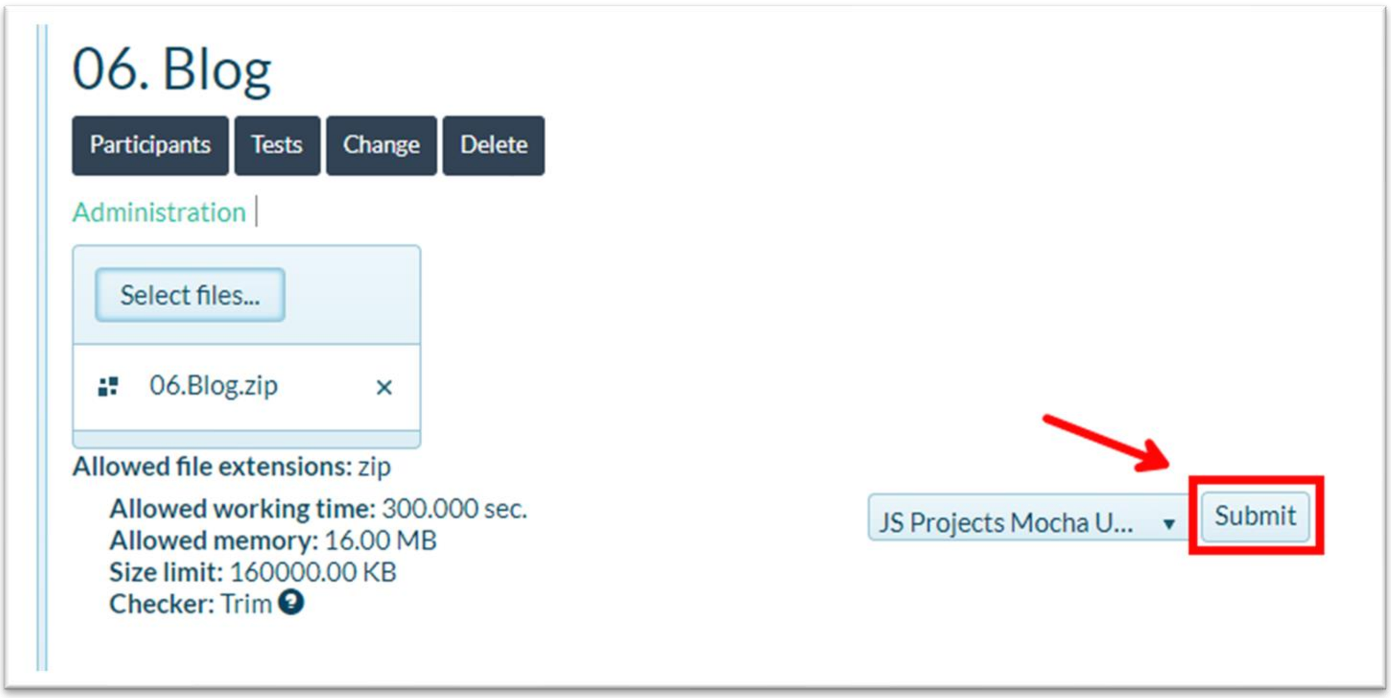
Select the content of your working folder (the given resources). Exclude the *node_modules* & *tests* folders. Archive the rest into a **ZIP** file and upload the archive to Judge.





Allowed file extensions: zip
Allowed working time: 300.000 sec.
Allowed memory: 16.00 MB
Size limit: 160000.00 KB
Checker: Trim

Name	Date modified	Type	Size
06.Blog	7/1/2022 12:28 PM	WinRAR ZIP archive	1 KB
app	7/9/2019 5:17 PM	JavaScript Source ...	1 KB
index	3/7/2022 3:07 PM	Brave HTML Docu...	1 KB
styles	7/9/2019 5:16 PM	Cascading Style S...	1 KB



Allowed file extensions: zip
Allowed working time: 300.000 sec.
Allowed memory: 16.00 MB
Size limit: 160000.00 KB
Checker: Trim