JS Front-end: Exam Preparation 2

Link to contest: https://judge.softuni.org/Contests/3915

01.Problem - Horse Racing

Now that your friend has become a sports journalist, he has to write down the actions that take place on the racetrack. Help him by writing a program that receives commands and prints template sentences.

Input

You will receive an **array** representing the current positions of the horses separated by the pipe symbol: "**|**". The order of the horses is **right to left** (the one on the far right is 1st and the one on the far left is last).

After that, you will be receiving **4 types of commands**. When the program receives "**Finish**", it should stop executing commands.

The commands can be:

- **Retake {overtaking-horse} {overtaken-horse}** – if the overtaking horse is to the left of the overtaken horse, **swap** the **position** of the two horses. Then, **print** the following on the console:
  - **"{overtaking-horse} retakes {overtaken-horse}."**
- **Trouble {horse-name}** – the given horse **drops by one position**, if it's **not** in the **last position already**. If the horse does drop, on the console should be **printed**:
  - **"Trouble for {horse-name} - drops one position."**
- **Rage {horse-name}** – the given horse **rages 2 positions** ahead. If the horse is in second position before the command is given, the horse just goes to the first position. If it's already in the first position, it stays in the first position. Then, on the console should be **printed**:
  - **"{horse-name} rages 2 positions ahead."**
- **Miracle** – the horse in the **last position** gets enormous power and **becomes the first**. Then, on the console should be **printed**:
  - **"What a miracle - {horse-name} becomes first."**

Constraints

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

Output

Every command should **print its own template sentence**. After the program receives "**Finish**", it should print the **updated positions** of the horses, **separated by arrows** ("**->**"):

- **"{horse3}->{horse2}->{horse1}"**

After the updated positions are printed, the **winner should be printed** as well:

- **"The winner is: {horse1}"**

Examples

Input	Output
['Bella|Alexia|Sugar', 'Retake Alexia Sugar', 'Rage Bella', 'Trouble Bella', 'Finish'])	Alexia retakes Sugar. Bella rages 2 positions ahead. Trouble for Bella - drops one position. Sugar->Bella->Alexia The winner is: Alexia

Input	Output
['Onyx|Domino|Sugar|Fiona', 'Trouble Onyx', 'Retake Onyx Sugar', 'Rage Domino', 'Miracle', 'Finish'])	Onyx retakes Sugar. Domino rages 2 positions ahead. What a miracle - Sugar becomes first. Onyx->Fiona->Domino->Sugar The winner is: Sugar

Input	Output
['Fancy|Lilly', 'Retake Lilly Fancy', 'Trouble Lilly', 'Trouble Lilly', 'Finish', 'Rage Lilly'])	Trouble for Lilly - drops one position. Lilly->Fancy The winner is: Fancy

Problem 2. Task Post

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 **Task Post** work as expected:

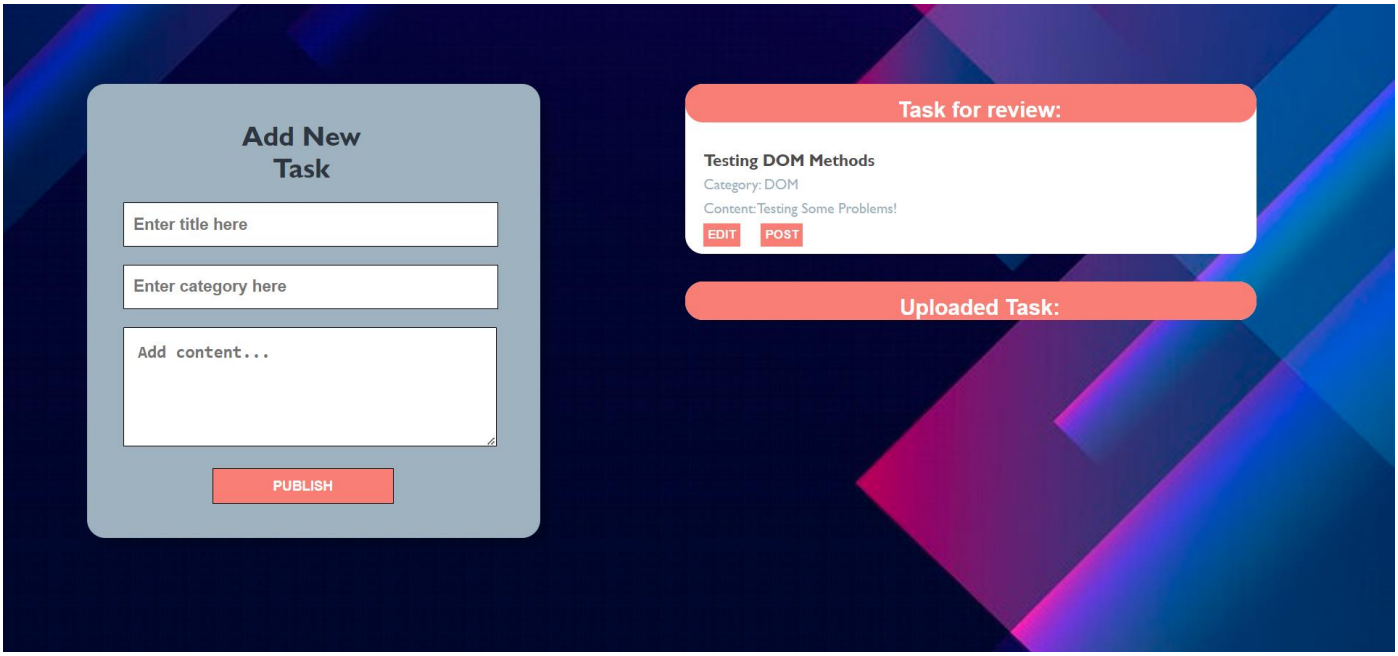
- **Title, category,** and **content** should be **non-empty strings**. If any of them are empty, the program should not do anything.

1. Getting the information from the task

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

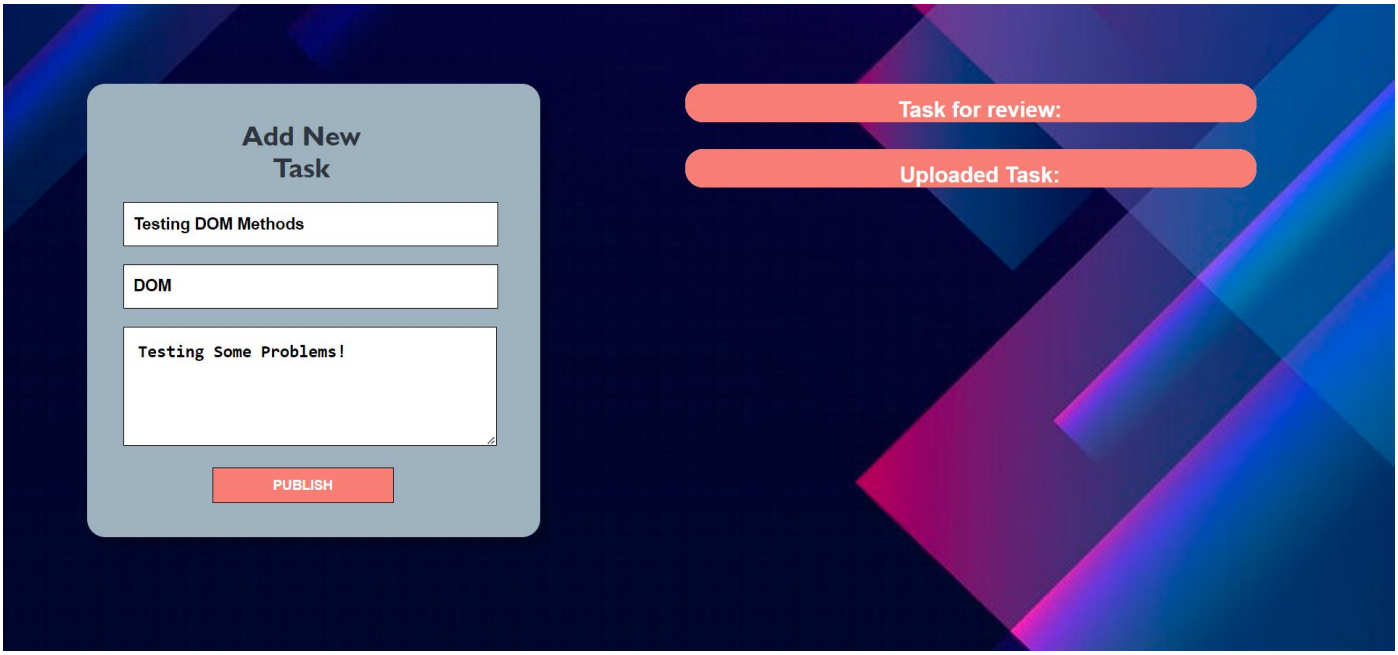
The HTML structure should look like this:

```
<ul id="review-list">
  <li class="rpost">
    <article>
      <h4>Testing DOM Methods</h4>
      <p>Category: DOM</p>
      <p>Content: Testing Some Problems!</p>
    </article>
    <button class="action-btn edit">Edit</button>
    <button class="action-btn post">Post</button>
  </li>
</ul>
```

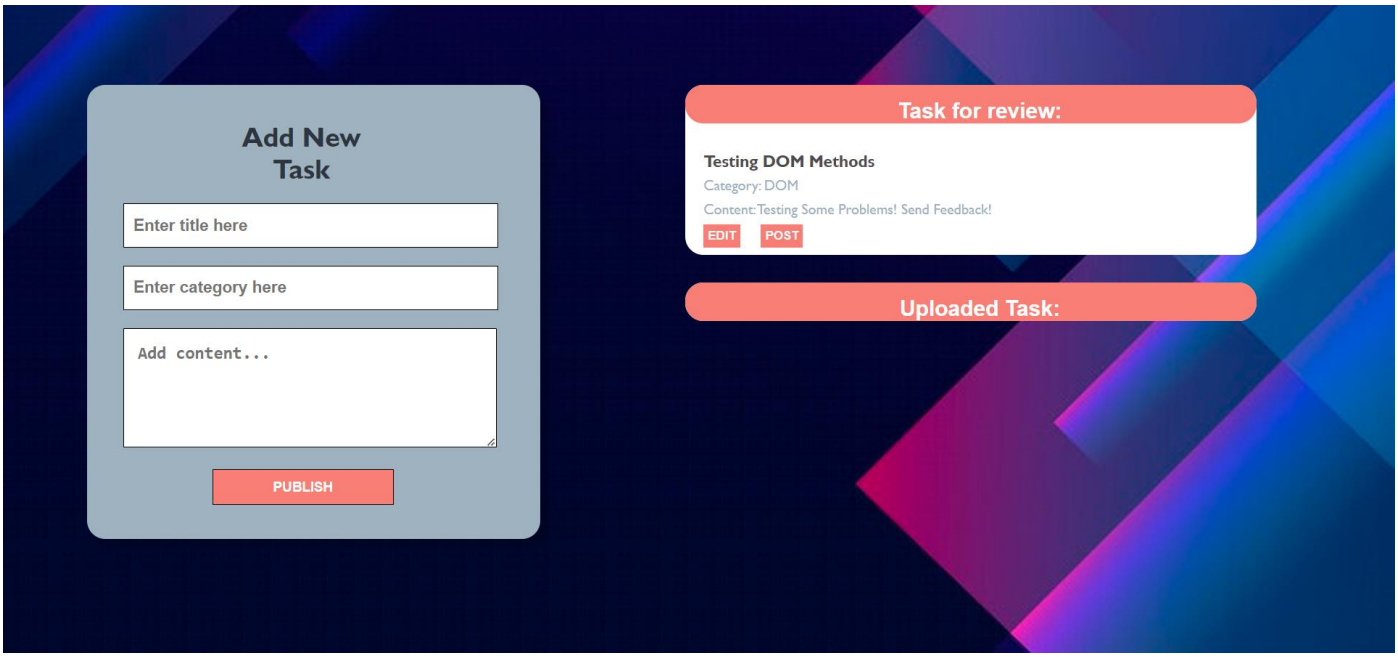


2. Edit information for task

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>` `"review-list"`**.



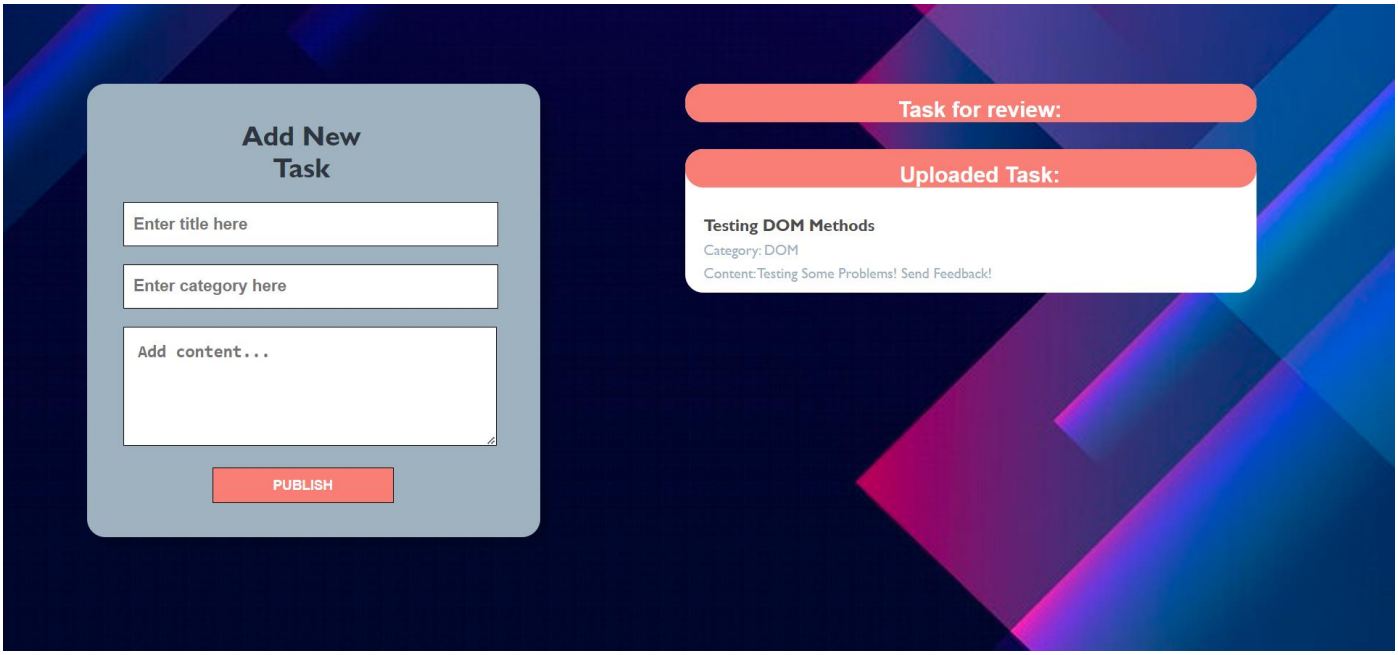
After editing the information, add a new item to the **<ul>** with the updated information.



3. Post Task

When you click the **[Post]** button, the task must be **deleted** from the **<ul>** with **id "review-list"** and appended to the **<ul>** with **id "published-list"**.
The **buttons** [Edit] and [Post] should be removed from the **<li>** element.

```
<ul id="published-list">
  <li class="rpost">
    <article>
      <h4>Testing DOM Methods</h4>
      <p>Category: DOM</p>
      <p>Content: Testing Some Problems! Send Feedback!</p>
    </article>
  </li>
</ul>
```



Problem 3 – Course Planner

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

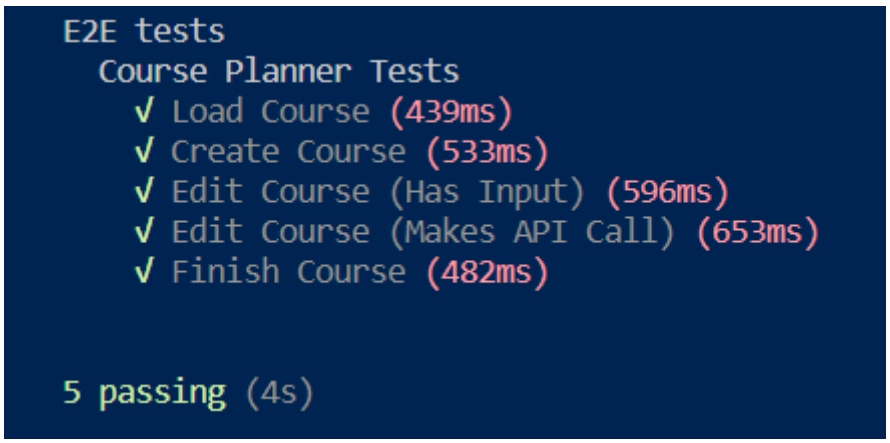
Write a JS program that can load, create, remove and edit a list of courses. You will be given an HTML template to which you must bind the needed functionality.

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)**.

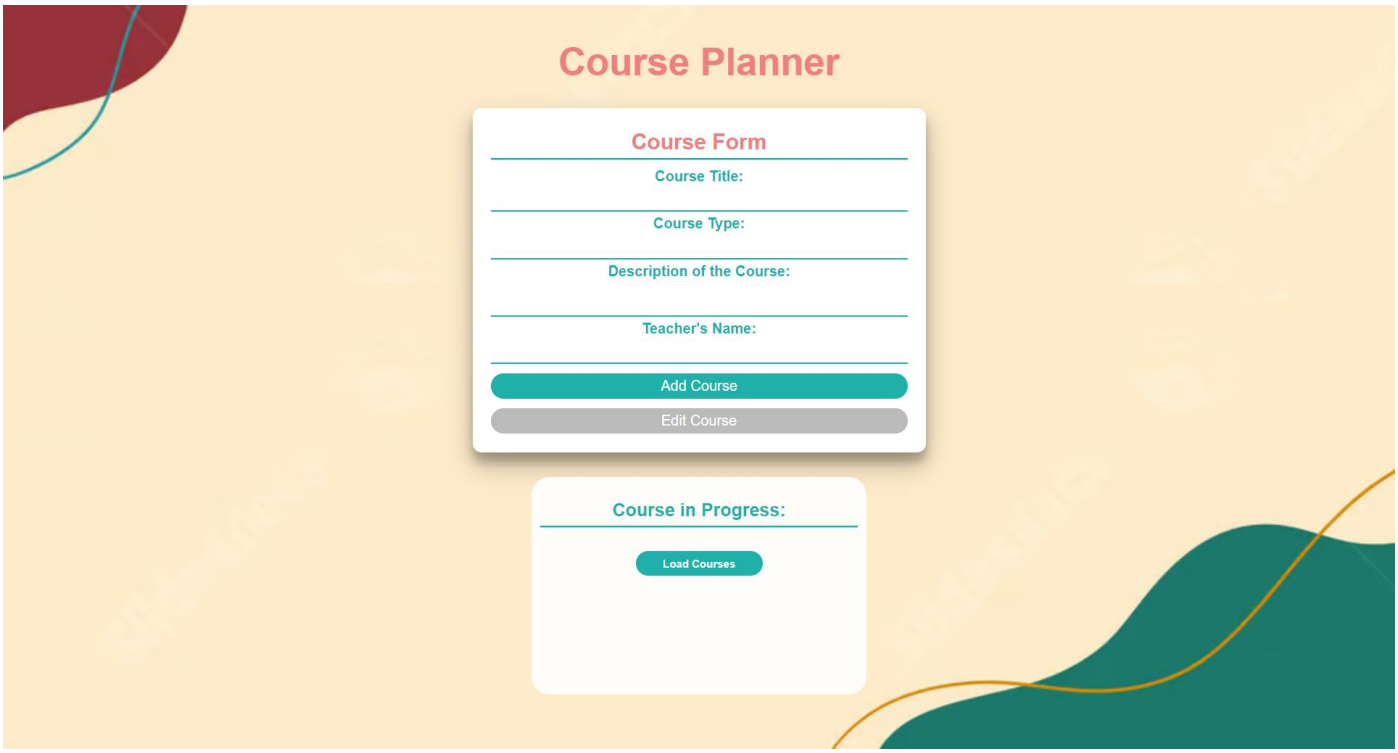
At any point, you can open up another console and run **`npm test`** to test the **current state** of your application. It's preferable for **all of your tests to pass locally** before you submit to the Judge platform, like this:



Endpoints

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

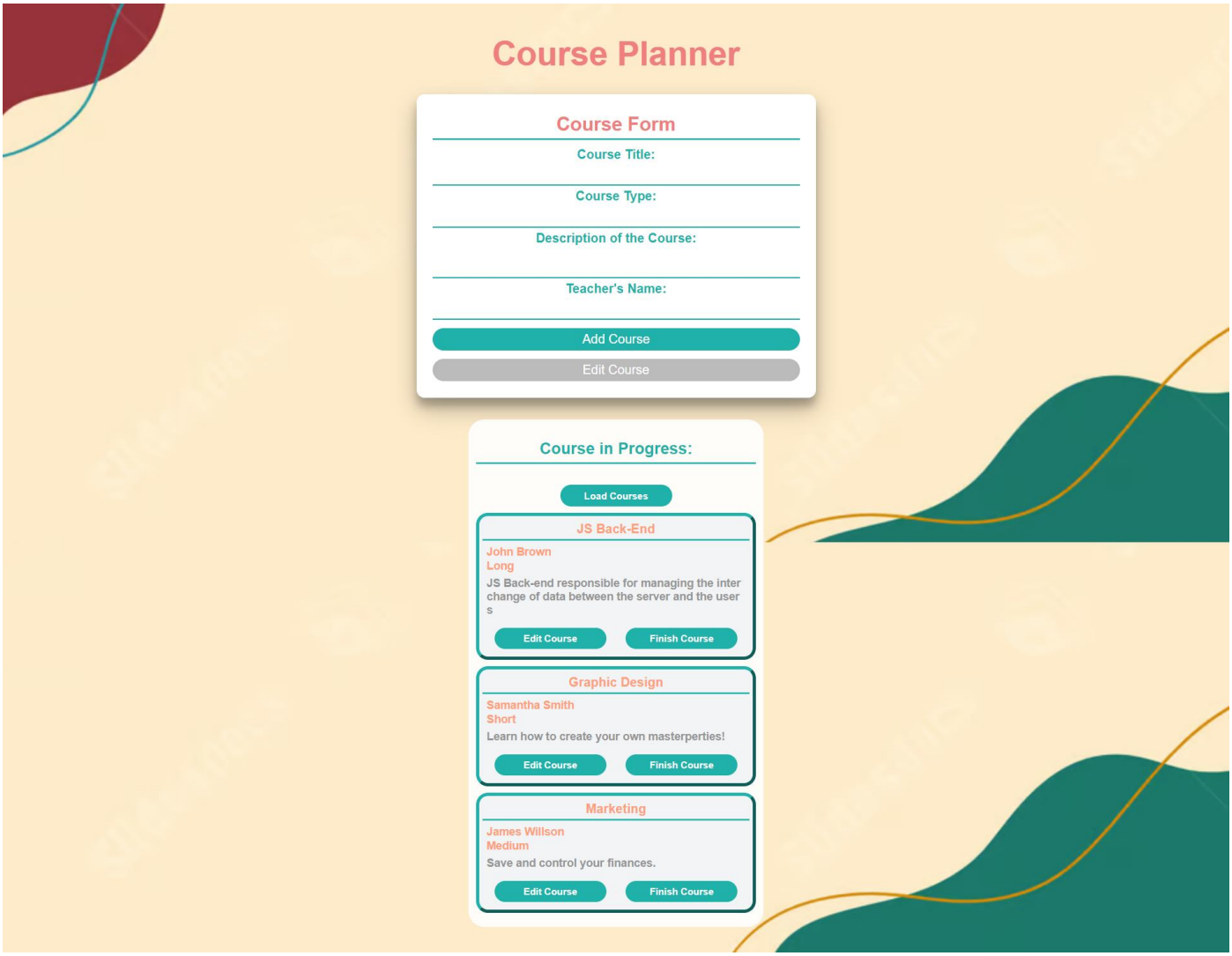
Load Courses



Clicking the **[Load Courses]** 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 Course]** button should be deactivated.

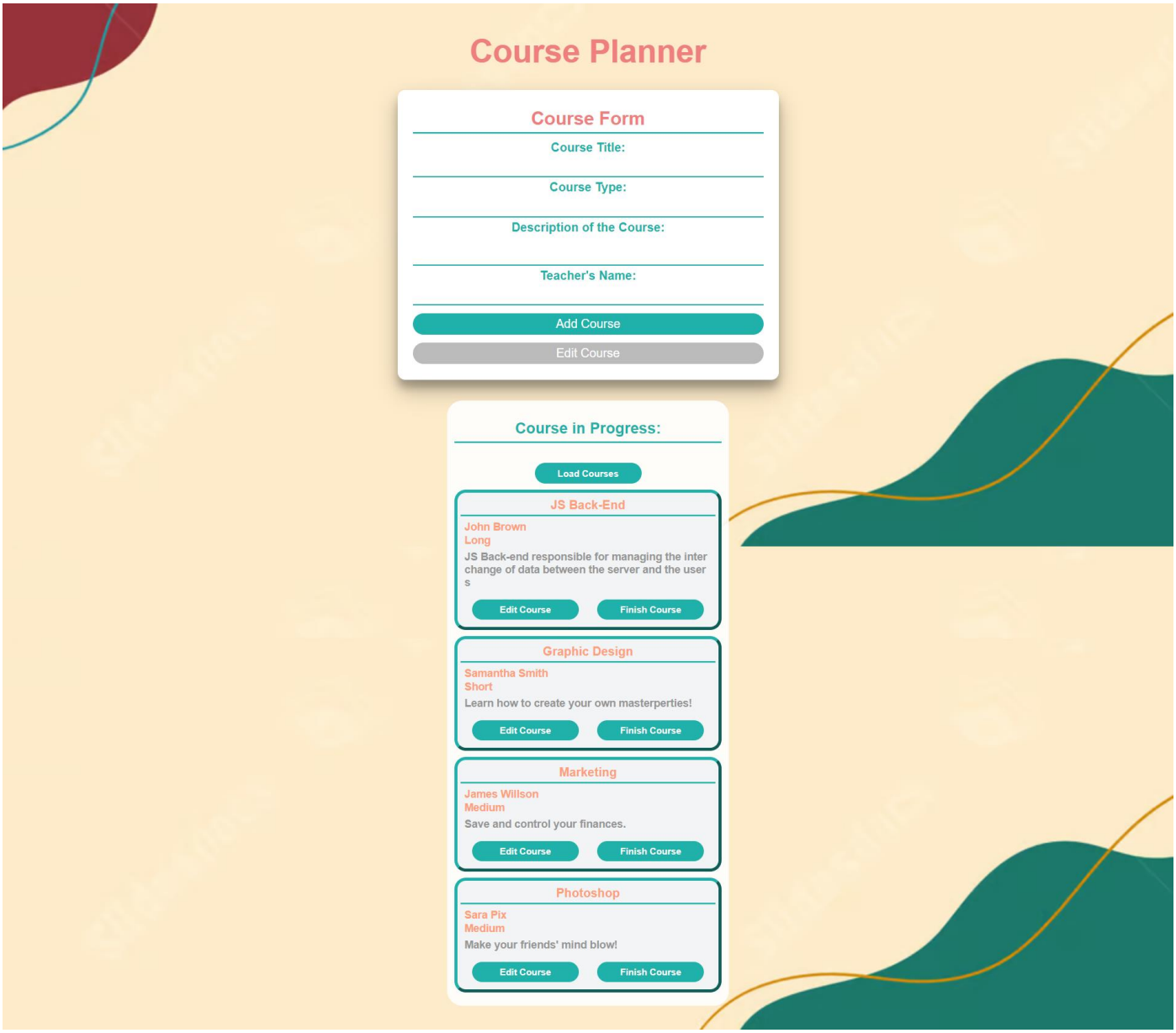
Each course has the following **HTML structure**:

```
<div class="container">
  <h2>JS Back-End</h2>
  <h3>John Brown</h3>
  <h3>Long</h3>
  <h4>
    "JS Back-end responsible for managing the interchange of data between the server and the users"
  </h4>
  <button class="edit-btn">Edit Course</button>
  <button class="finish-btn">Finish Course</button>
</div>
```



Add a Course

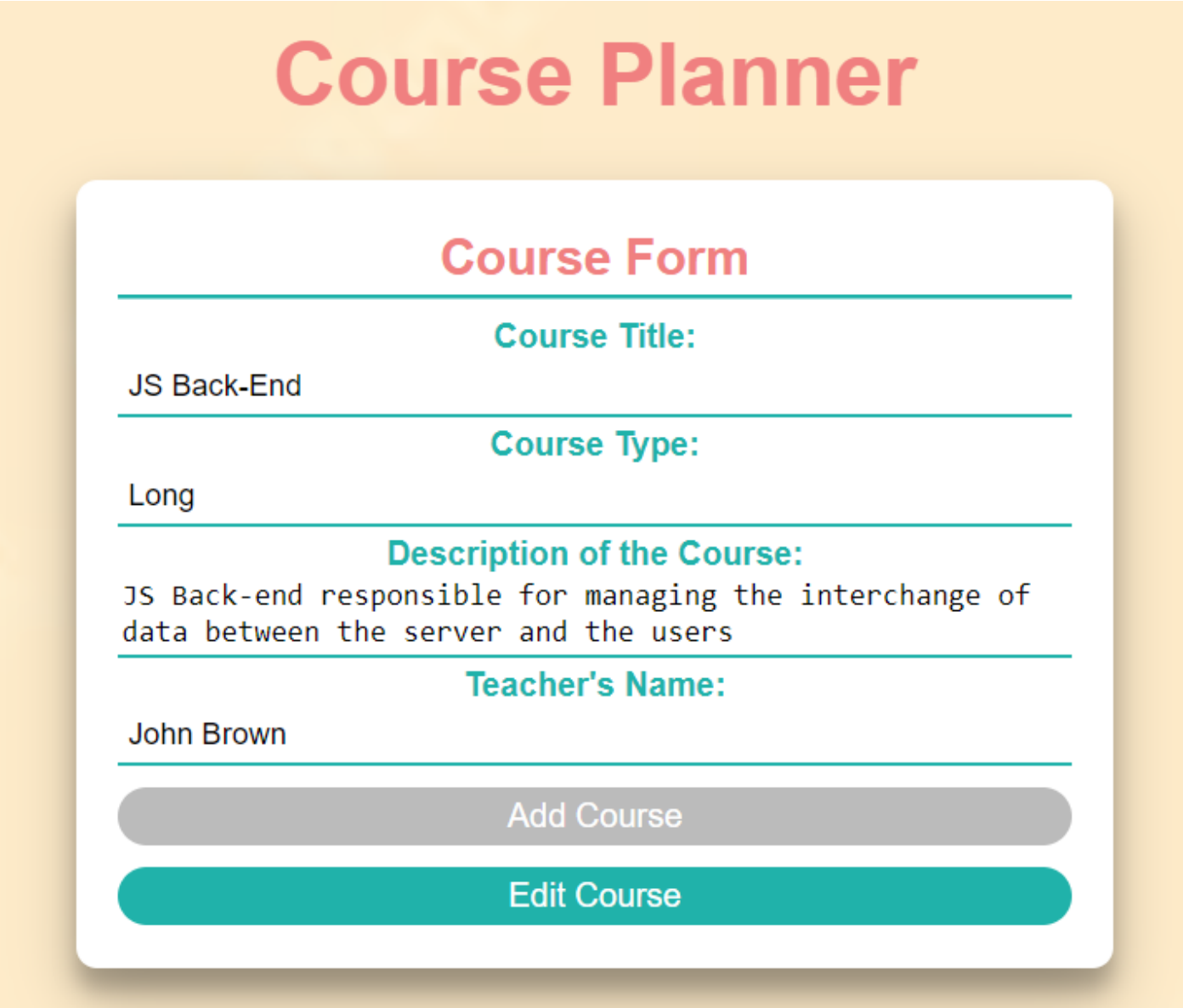
Clicking the **[Add Course]** button should send a **POST** request to the server, creating a new course with the **title**, **type** ("Long", "Medium", or "Short"), **description, , and the teacher's name** from the input values. After a successful creation, you should send another **GET** request to fetch all the courses, including the **newly added one** into the **Course Progress** column. You should also **clear all the input fields** after the creation!



Edit a Course

Clicking the **[Edit Course]** 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 Course]** button in the form should be activated and the **[Add Course]** one should be deactivated.

After clicking the [**Edit Course]** button in the form, you should send a **PUT** request to the server to **modify the title, type, description, and the teacher's name** 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 Course]** button should be deactivated and the **[Add Course]** one should be activated.



Finish a Course

Clicking the **[Finish Course]** 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.