JS Front-end: Exam Preparation 1

Link to contest: https://judge.softuni.org/Contests/4807/JS-Front-End-Regular-Exam-18-August-2024

01. Superhero Alliance

You are the leader of a newly formed Superhero Alliance, dedicated to protecting the city from nefarious villains and supernatural threats. Your task is to assemble a team of superheroes with extraordinary abilities and manage their heroic actions. Here's how you'll proceed:

On the first line of input, you will receive an integer **n** – the number of superheroes you can recruit. Following this, the details of each superhero will be provided in the next **n** lines. Each superhero's details include their superhero name, their superpower, and their current energy level in the following format:

`"{superhero name}-{superpower}-{energy}"`

- The energy level indicates how much power they have to use their abilities.
- a superhero can have a **maximum** of **100 energy**
- if the superhero's energy is **0** or **less**, he cannot perform his superpowers

After assembling your team, you will receive a series of commands, each on a new line, separated by " * ", until the "**Evil Defeated!**" command is given. Here are the actions that superheroes can undertake:

`"Use Power * {superhero name} * {superpower} * {energy required}"`

- Check if the superhero has the specified superpower and enough energy. If both conditions are met, they perform the superpower and reduce their energy accordingly, then **print**:

  `"{superhero name} has used {superpower} and now has {remaining energy} energy!"`

- If the superhero does not have the specified **superpower** or does not have enough **energy**, **print**:

  `"{superhero name} is unable to use {superpower} or lacks energy!"`

`"Train * {superhero name} * {training energy}"`

- Increase the superhero's energy by the specified amount (**training energy**). If this action would exceed the **maximum** energy level of 100, set the energy to 100. **Print**:

  `"{superhero name} has trained and gained {energy gained} energy!"`

- If the current energy is **already** 100, **print**:

  `"{superhero name} is already at full energy!"`

`"Learn * {superhero name} * {new superpower}"`

- If the superhero already knows the specified **superpower**, **print:**

  `"{superhero name} already knows {new superpower}."`

- Otherwise, add the new superpower to their list of abilities. **Print**:

  `"{superhero name} has learned {new superpower}!"`

Input

- On the first line of the standard input, you will receive an integer **n.**
- On the following **n** lines, the superheroes themselves will follow with their **superpower** and **energy** separated by a dash in the following format.

- You will be receiving different **commands**, each on a new line, separated by **" * "** until the **"Evil Defeated!"** command is given.

Output

- After processing all commands until "**Evil Defeated!**", print each superhero in the following format:

  "Superhero: {superhero name}

  - Superpowers: {superpower 1, superpower 2, ...}

  - Energy: {current energy}"

Constraints

- All superhero names will be unique.
- All given commands will be valid and within the constraints.

Examples

Input	Output
(["3", "Iron Man-Repulsor Beams,Flight-80", "Thor-Lightning Strike,Hammer Throw-10", "Hulk-Super Strength-60", "Use Power * Iron Man * Flight * 30", "Train * Thor * 20", "Train * Hulk * 50", "Learn * Hulk * Thunderclap", "Use Power * Hulk * Thunderclap * 70", "Evil Defeated!"])	Iron Man has used Flight and now has 50 energy! Thor has trained and gained 20 energy! Hulk has trained and gained 40 energy! Hulk has learned Thunderclap! Hulk has used Thunderclap and now has 30 energy! Superhero: Iron Man - Superpowers: Repulsor Beams, Flight - Energy: 50 Superhero: Thor - Superpowers: Lightning Strike, Hammer Throw - Energy: 30 Superhero: Hulk - Superpowers: Super Strength, Thunderclap - Energy: 30

Input	Output
(["2", "Iron Man-Repulsor Beams,Flight-20", "Thor-Lightning Strike,Hammer Throw-100", "Train * Thor * 20", "Use Power * Iron Man * Repulsor Beams * 30", "Evil Defeated!"])	Thor is already at full energy! Iron Man is unable to use Repulsor Beams or lacks energy! Superhero: Iron Man - Superpowers: Repulsor Beams, Fligh - Energy: 20 Superhero: Thor - Superpowers: Lightning Strike, Hammer Throw - Energy: 100

Input	Output
(["2", "Iron Man-Repulsor Beams,Flight-100", "Thor-Lightning Strike,Hammer Throw-50", "Train * Thor * 20", "Learn * Thor * Hammer Throw", "Use Power * Iron Man * Repulsor Beams * 30", "Evil Defeated!"])	Thor has trained and gained 20 energy! Thor already knows Hammer Throw. Iron Man has used Repulsor Beams and now has 70 energy! Superhero: Iron Man - Superpowers: Repulsor Beams, Flight - Energy: 70 Superhero: Thor - Superpowers: Lightning Strike, Hammer Throw - Energy: 70

02. Historic Events

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:

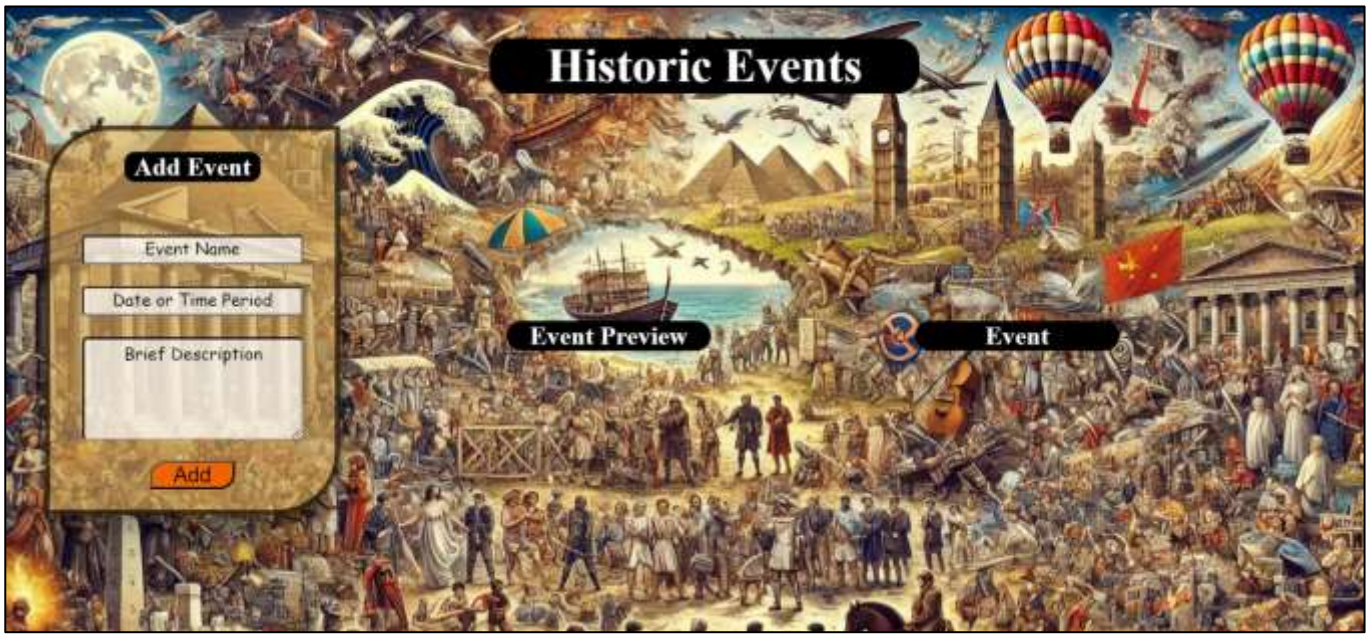
- `.forEach()` with **NodeList** (returned by `querySelector()` and `querySelectorAll()`)
- `.forEach()` with **HTMLCollection** (returned by `getElementsByClassName()` and `element.children`)
- using the **spread-operator** (`...`) to convert a **NodeList** into an array

- `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 **Historic Events** application work as expected:

- **Event Name, Date or Time Period,** and **Brief Description** should be **non-empty strings**. If any of them are empty, the program should not do anything.

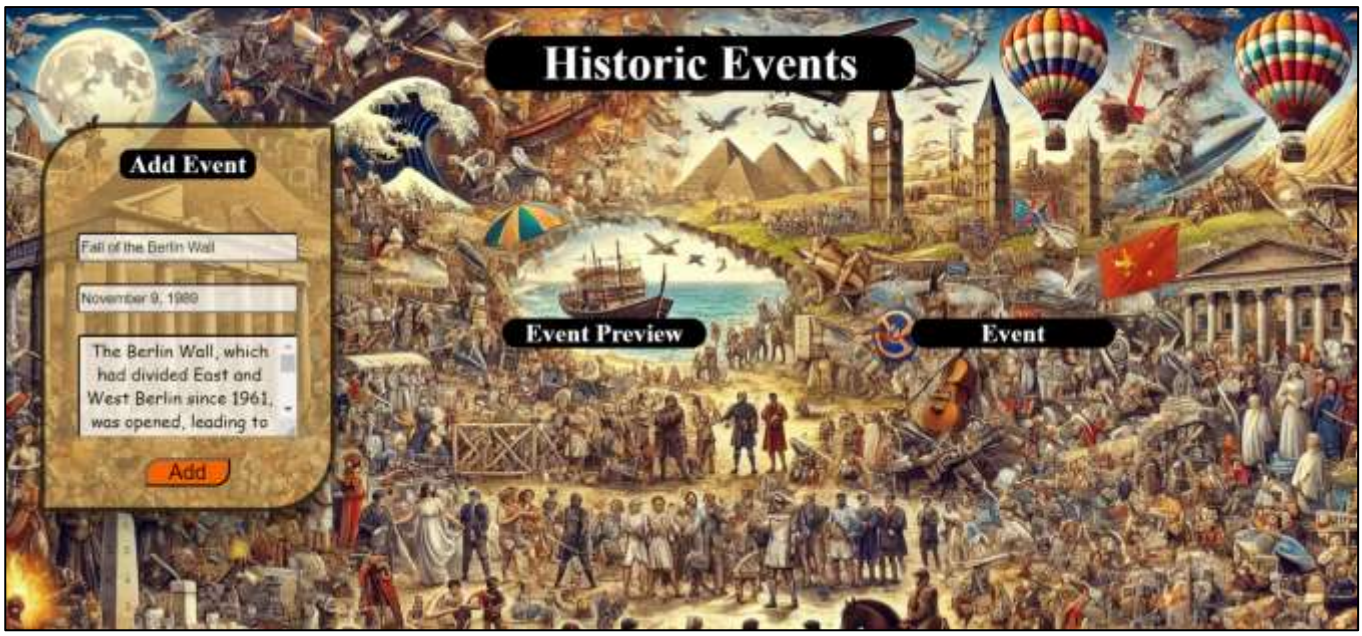
Getting the information from the form

- When you click the **`[Add]`** button, the information from the input fields must be added to the **`<ul>`** with the **`id "preview-list".`**
- The input fields should then be cleared, and the **`[Add]`** button must be **disabled**.
- At the same time the **`[Edit]`** and the **`[Next]`** buttons must be **added**.
- The HTML structure should look like this:

```
<ul id="preview-list">
  <li> flex
    <article> flex
      <p>Fall of the Berlin Wall</p>
      <p>November 9, 1989</p>
      <p>
        The Berlin Wall, which had divided East and West Berlin since
        1961, was opened, leading to the reunification of Germany. This
        event marked the symbolic end of the Cold War and the division
        between the communist East and the capitalist West.
      </p>
    </article>
    <div class="buttons"> flex
      <button class="edit-btn">Edit</button>
      <button class="next-btn">Next</button>
    </div>
  </li>
</ul>
```

Edit Info

- When the **[Edit]** button is clicked, the information from the **Event Preview** must be sent to the input fields and the record should be deleted from the ** "preview-list".**
- The **[Add]** button must be **enabled** again.

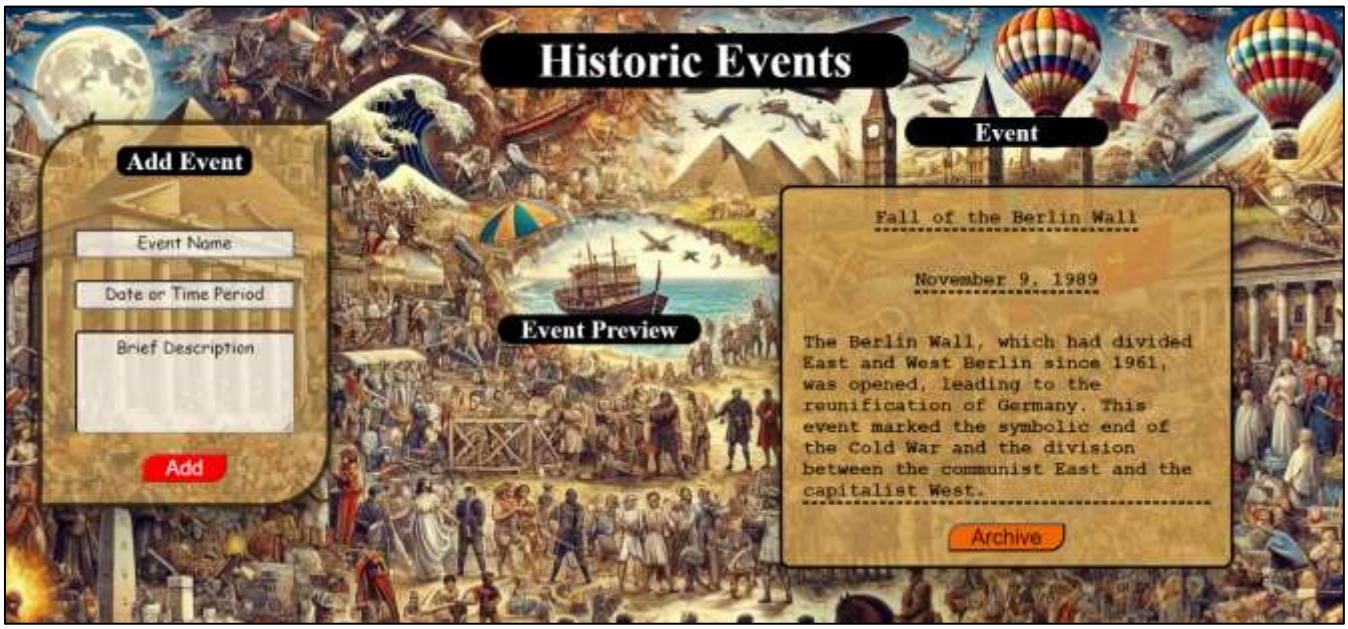


- After editing the information, add a new item to the **`<ul>`** with the updated information.

Next

- When you click the **`[Next]`** button, the record must be **deleted** from the **`<ul>`** with **`id "preview-list"`** and appended to the **`<ul>`** with **`id "archive-list"`**.
- The **buttons [Edit]** and **[Next]** should be removed from the **`<li>`** element and the **button [Archive]** should be added.
- The HTML structure should look like this:

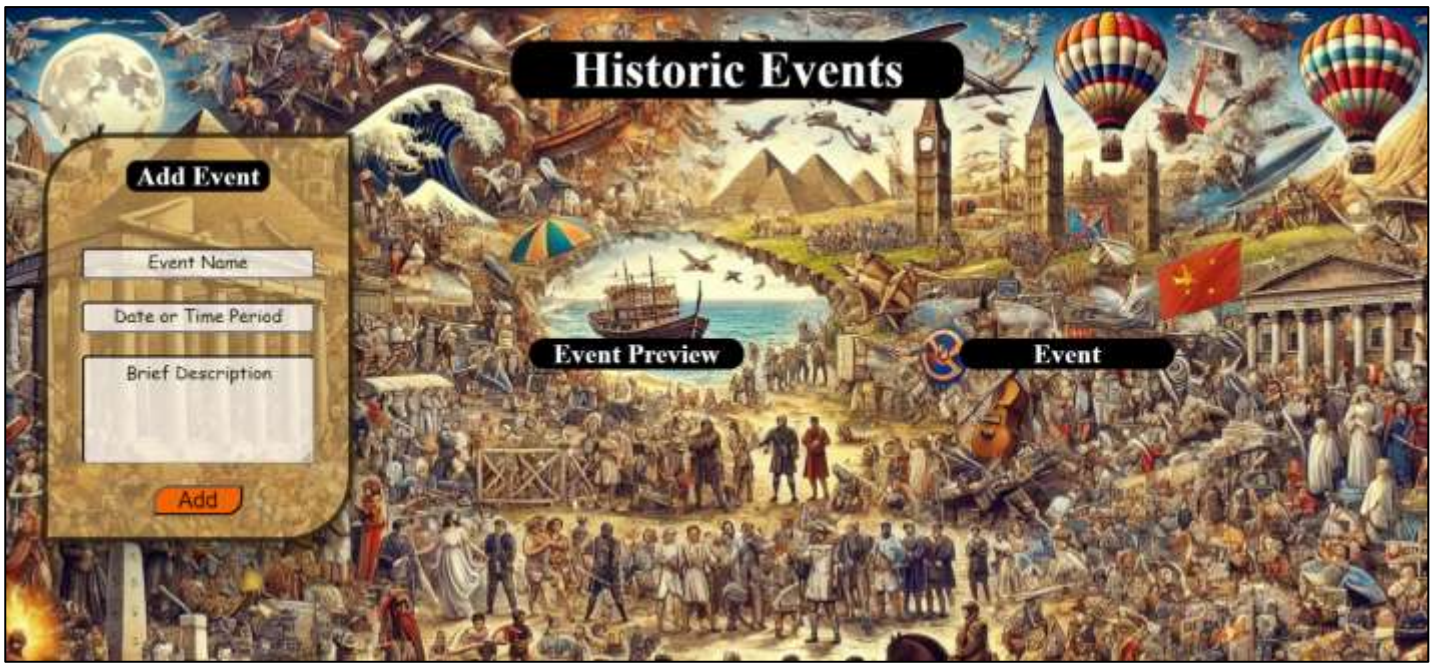
```
<h2>Event</h2>
<ul id="archive-list">
  <li> flex
    <article> flex
      <p>Fall of the Berlin Wall</p>
      <p>November 9, 1989</p>
      <p>
        The Berlin Wall, which had divided East and West Berlin since
        1961, was opened, leading to the reunification of Germany. This
        event marked the symbolic end of the Cold War and the division
        between the communist East and the capitalist West.
      </p>
    </article>
    <button class="archive-btn">Archive</button>
  </li>
</ul>
```



Archive Event

When you click the **`[Archive]`** button, the record must be **deleted** from the **`<ul>`** with **the `id "archive-list"`**.

- The **`[Add]`** button must be **enabled** again.



Submission

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

03.Match Memories

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 step matches. 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
  Match Memories Tests
    √ Load Matches (371ms)
    √ Add match (494ms)
    √ Edit match (Has Input) (436ms)
    √ Edit match (Makes API Call) (448ms)
    √ Delete match (320ms)


5 passing (3s)
```

Endpoints

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

Load Matches

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

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

```html
<ul id="list">
  <li class="match">
    <div class="info">
      <p>England</p>
      <p>2-1</p>
      <p>Slovakia</p>
    </div>
    <div class="btn-wrapper">
      <button class="change-btn">Change</button>
      <button class="delete-btn">Delete</button>
    </div>
  </li>
</ul>
```

Add a Match

Clicking the **[Add Match]** button should send a **POST** request to the server, creating a new Record record with the **host**, **score and guest** from the input values. After a successful creation, you should send another **GET** request to fetch all the matches including the **newly added one**. You should also **clear all the input fields** after the creation!

Edit a Match

Clicking the **[Change]** button should populate the info into the input fields above. The **[Edit Match]** button **in the form** should be activated and the **[Add Match]** one should be deactivated.

After clicking the [**Edit Match]** button in the form, you should send a **PUT** request to the server to **modify the Host, Score and Guest** 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 Match]** button should be deactivated and the **[Add Match]** one should be activated.

Delete

Clicking the **[Delete]** 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 Matches **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.