

JS Front-end: Exam Preparation 1

01.Wild West Adventure

You find yourself amidst the gritty and unforgiving terrain of the Wild West, eager to carve out your legend in this vast, lawless frontier. As dawn breaks, you're presented with an opportunity to assemble a posse of the most daring gunslingers and outlaws to ride by your side.

On the first line of the standard input, you will receive an integer n – the number of characters that you can choose for your posse. On the next n lines, these rough-and-tumble characters themselves will follow with their health points (HP) and the number of bullets they carry, separated by a single space in the following format:

"{hero name} {HP} {bullets}"

- **HP** stands for hit points
- a character can have a **maximum** of **100 HP**
- The number of bullets signifies how many shots they can fire before needing to reload.

After you have successfully assembled your posse, the real adventure begins. You will be navigating through a series of commands, each on a **new line**, separated by " - ", until the **"Ride Off Into Sunset"** command is given.

There are several actions that your characters can undertake:

"FireShot - {character name} - {target}"

- If the character has bullets, they fire a shot, reducing their bullet count by one. **Print** this message:

"{character name} has successfully hit {target} and now has {number of bullets left} bullets!"

- If the character does not have bullets to shoot, **print**:
 - **"{character name} doesn't have enough bullets to shoot at {target}!"**

"TakeHit - {character name} - {damage} - {attacker}"

- Reduce the character's HP by the given damage amount. If the character is still standing (their HP is greater than 0), **print**:

"{character name} took a hit for {damage} HP from {attacker} and now has {current HP} HP!"
- If the character has been gunned down, **remove** them from your posse and **print**:

"{character name} was gunned down by {attacker}!"

"Reload - {character name}"

- If they have less than the maximum capacity of bullets (6), the character loads their gun. In this case, **print** the following on the console:

"{character name} reloaded {number of bullets reloaded} bullets!"
- If the gun is already fully loaded, **print**:

"{character name}'s pistol is fully loaded!"

"PatchUp - {character name} - {amount}"

- The character patches up, recovering HP. If this action would bring their HP above the maximum value (100), HP is increased to 100. **Print**:

"{character name} patched up and recovered {amount recovered} HP!"

- If the character is already at full health, **print**:
`"{character name} is in full health!"`

Input

- On the first line of the standard input, you will receive an integer **n**.
- On the following **n** lines, the characters themselves will follow with their **hit points** and **bullets** separated by a space in the following format.
- You will be receiving different **commands**, each on a new line, separated by " - ", until the "**Ride Off Into Sunset**" command is given.

Output

- Print all members of your party who are **still alive**, in the following format:
`"{Character name}`
`HP: {current HP}`
`Bullets: {current bullets}"`

Constraints

- The starting HP/bullets of the characters will be valid, 32-bit integers will never be negative or exceed the respective limits.
- The HP/bullets amounts in the commands will never be negative.
- The hero names in the commands will always be valid members of your posse. No need to check that explicitly.

Examples

Input	Output
<pre>(["2", "Gus 100 0", "Walt 100 6", "FireShot - Gus - Bandit", "TakeHit - Gus - 100 - Bandit", "Reload - Walt", "Ride Off Into Sunset"])</pre>	<pre>Gus doesn't have enough bullets to shoot at Bandit! Gus was gunned down by Bandit! Walt's pistol is fully loaded! Walt HP: 100 Bullets: 6</pre>

Input	Output
<pre>(["2", "Jesse 100 4", "Walt 100 5", "FireShot - Jesse - Bandit", "TakeHit - Walt - 30 - Bandit", "PatchUp - Walt - 20" , "Reload - Jesse", "Ride Off Into Sunset"])</pre>	<pre>Jesse has successfully hit Bandit and now has 3 bullets! Walt took a hit for 30 HP from Bandit and now has 70 HP! Walt patched up and recovered 20 HP! Jesse reloaded 3 bullets! Jesse HP: 100 Bullets: 6 Walt HP: 90 Bullets: 5</pre>

Input	Output
<pre>(["2", "Gus 100 4", "Walt 100 5", "FireShot - Gus - Bandit", "TakeHit - Walt - 100 - Bandit", "Reload - Gus", "Ride Off Into Sunset"])</pre>	<pre>Gus has successfully hit Bandit and now has 3 bullets! Walt was gunned down by Bandit! Gus reloaded 3 bullets! Gus HP: 100 Bullets: 6</pre>

02. Contact List

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 **Contact List** work as expected:

- **Name**, **Phone Number**, and **Category** 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 **[Add]** button, the information from the input fields must be added to the `` with the **id "check-list"** and the input fields should be **cleared**.

The HTML structure should look like this:

```

<h2>Check</h2>
<ul id="check-list">
  <li> <div>
    <article>
      <p>name:Peter</p>
      <p>phone:0888888888</p>
      <p>category:work</p>
    </article>
    <div class="buttons">
      <button class="edit-btn"></button>
      <button class="save-btn"></button>
    </div>
  </li>
</ul>

```



2.Edit Contact

When the **[Edit]** button is clicked, the information from the check must be sent to the input fields on the left side and the record should be deleted from the `` "check-list".



After editing the information, add a new item to the `` with the updated information.



3. Save Contact

When you click the **[Save]** button, the task must be **deleted** from the `` with **id "check-list"** and appended to the `` with **id "contact-list"**.

The **buttons [Edit]** and **[Save]** should be removed from the `` element and the **button [Delete]** should be added.


```

<h2>Contacts</h2>
▼ <ul id="contact-list">
  ▼ <li> flex
    ▼ <article> flex
      <p>name:Peter</p>
      <p>phone:0111111111</p>
      <p>category:work</p>
    </article>
    <button class="del-btn"></button>
  </li>
</ul>

```



4.Delete Contact

When you click the **[Delete]** button, the task must be **deleted** from the **** with **id "contact-list"** .



Submission

Submit only your `solve()` function.

03. My Board Games Collection

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 board games. 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
Board Games Collection
  ✓ Load Games (132ms)
  ✓ Add Game (157ms)
  ✓ Edit Game (Has Input) (150ms)
  ✓ Edit Game (Makes API Call) (196ms)
  ✓ Delete Game (145ms)

5 passing (2s)
```

Endpoints

- <http://localhost:3030/jsonstore/games/>
- <http://localhost:3030/jsonstore/games/:id>

Load Games



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

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

```

▼ <div class="board-game"> flex
  ▼ <div class="content">
    <p>CATAN-Rise of Inkas</p>
    <p>4</p>
    <p>Eurogame</p>
  </div>
  ▼ <div class="buttons-container"> flex
    <button class="change-btn">Change</button>
    <button class="delete-btn">Delete</button>
  </div>
</div>

```



Add a Game

Clicking the **[Add Game]** button should send a **POST** request to the server, creating a new game record with the **name**, **type**, and **max players** from the input values. After a successful creation, you should send another **GET** request to fetch all games records, including the **newly added one**. You should also **clear all the input fields** after the creation!



Edit a Game

Clicking the **[Change]** button on a game the information about the game should be populated into the input fields above. The **[Edit Game]** button in the form should be activated and the **[Add Game]** one should be deactivated.

After clicking the **[Edit Game]** button in the form, you should send a **PUT** request to the server to **modify the name, type** and the **max players** of the changed item. After the successful request, you should **fetch the games again** and see that the changes have been made. After that, the **[Edit Game]** button should be deactivated and the **[Add Game]** one should be activated. You should also **clear all the input fields** after the edit!

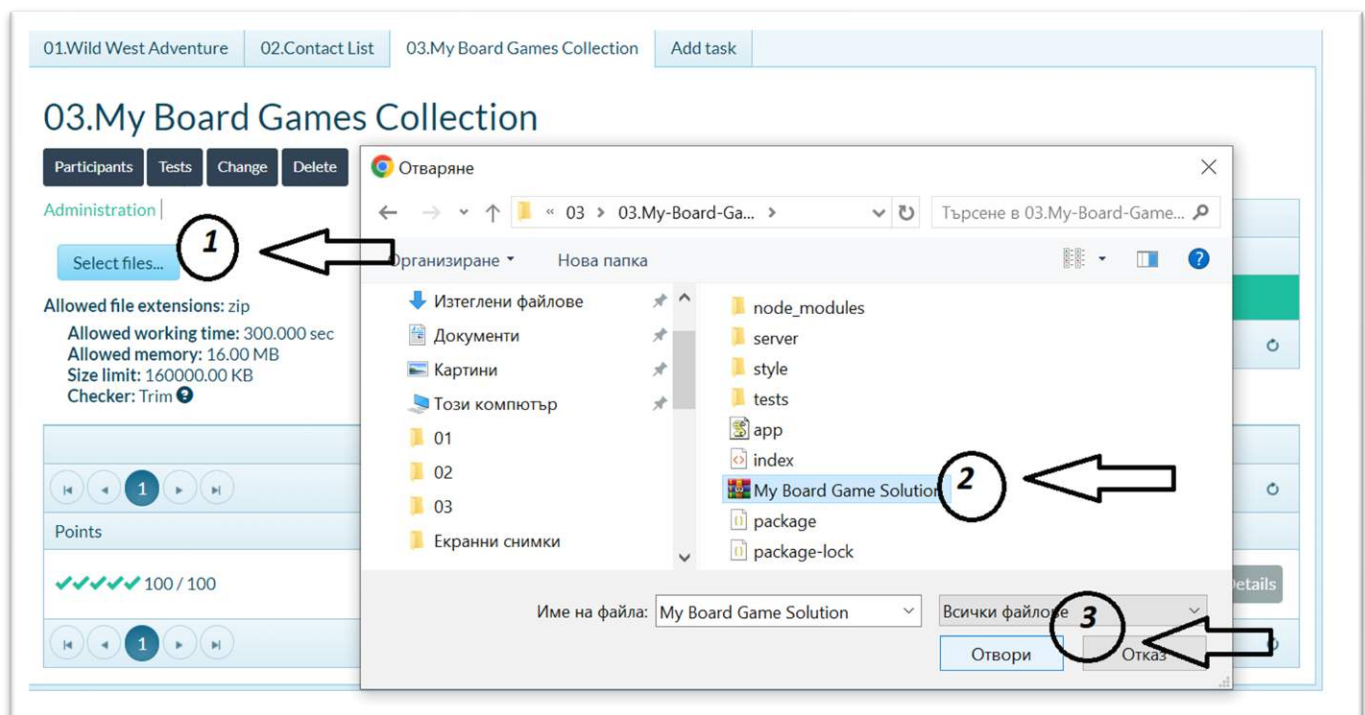
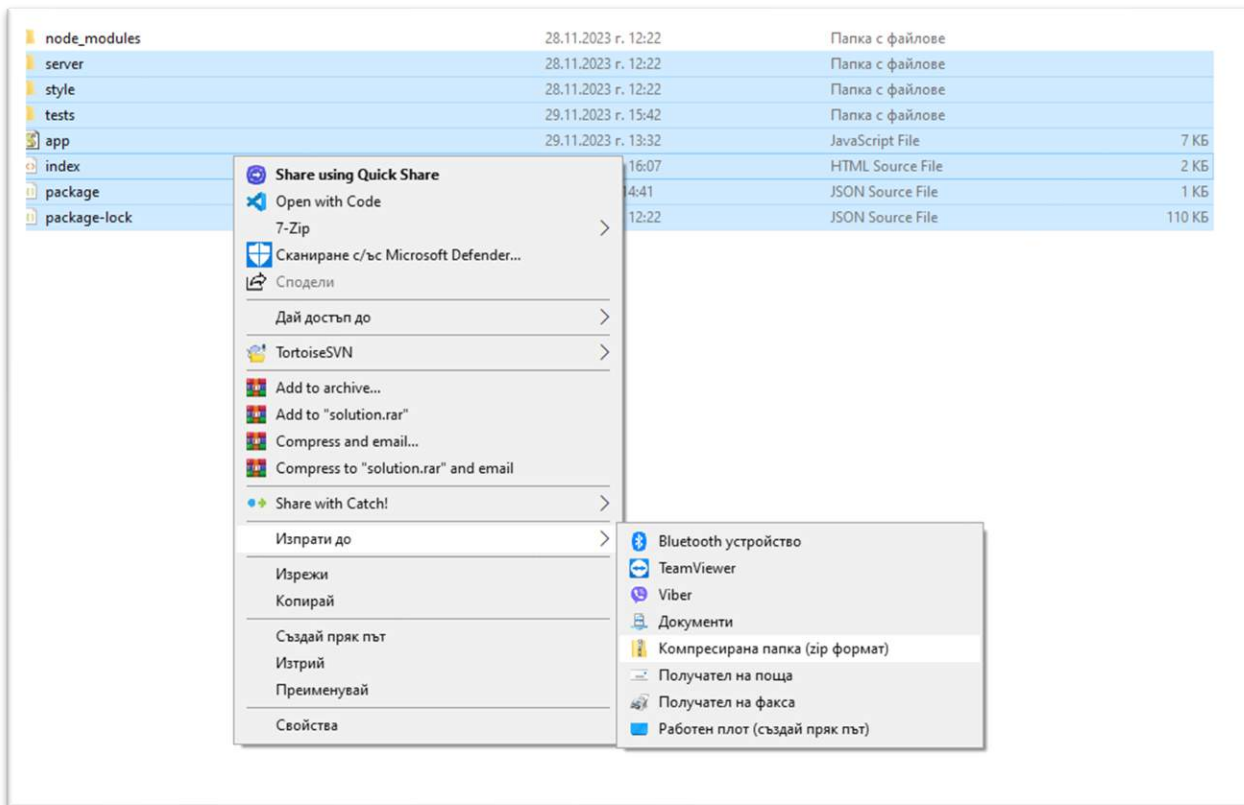


Delete

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 games 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.



[01.Wild West Adventure](#)[02.Contact List](#)[03.My Board Games Collection](#)[Add task](#)

03.My Board Games Collection


[Participants](#)[Tests](#)[Change](#)[Delete](#)[Administration](#)[Select files...](#) My Board Game Solution.zip

Allowed file extensions: zip

Allowed working time: 300.000 sec

Allowed memory: 16.00 MB

Size limit: 160000.00 KB

Checker: Trim 

JS Projects Mocha U... ▾

[Submit](#)

Submissions



Points

✓✓✓✓✓ 100 / 100

Time and memory used

Memory: 0.00 MB

Time: 0.000 s

