JS Front-end: Exam Preparation 1

Link to contest: https://judge.softuni.org/Contests/3914

Problem 1. Sprint Review

It's the end of the **two-week** Sprint, and your job now is to **calculate** the **sum** of the **estimation points** for each of the **different columns** on the Sprint board and decide whether the **Sprint** was **successful** or **not**. The Sprint board contains **assignees** and their **individual tasks**, each task has a **task ID** (a unique string), **title**, **status** ('ToDo', 'In Progress', 'Code Review', 'Done'), and **estimation points**.

You will receive **an array as a parameter**. The **first element** in that array is an integer, **n**.

The **next n amount of elements** contain information about the **initial state** of the Sprint board, each element having the following format: **`"{assignee}:{taskId}:{title}:{status}:{estimatedPoints}"`**. An **assignee** can have **multiple tasks**!

The **other elements inside the array** contain commands that will **change** the **state** of the **tasks** inside the board, each command containing information separated with the symbol **`":"`**.

The commands will be the following:

- **`"Add New:{assignee}:{taskId}:{title}:{status}:{estimatedPoints}"`**:
  - You need to **add** the **new task to the end of** the tasks collection of the given **assignee**.
  - If the assignee **does not exist** on the board, print:
    - **`"Assignee {assignee} does not exist on the board!"`**
- **`"Change Status:{assignee}{taskId}:{newStatus}"`**:
  - Change the **status** of the task with the given **task ID** for the **assigned person**.
  - If the assignee **does not exist** on the board, print:
    - **`"Assignee {assignee} does not exist on the board!"`**
  - If the task ID **does not exist** inside the collection for the assignee, print:
    - **`"Task with ID {taskId} does not exist for {assignee}!"`**
- **`"Remove Task:{assignee}{index}"`**:
  - Remove the **task** at the **given index** from the assignee's **collection**.
  - If the assignee **does not exist** on the board, print:
    - **`"Assignee {assignee} does not exist on the board!"`**
  - If the index is **out of bounds**, print:
    - **`"Index is out of range!"`**

In the end, you have to print the **total points** of all **ToDo**, **In Progress**, **Code Review**, and **Done** tasks, each on a **separate line**, in the following format:

- **`"ToDo: {toDoTasksTotalPoints}pts"`**
- **`"In Progress: {inProgressTasksTotalPoints}pts"`**
- **`"Code Review: {codeReviewTasksTotalPoints}pts"`**
- **`"Done Points: {doneTasksTotalPoints}pts"`**

You also have to print **whether** or **not** the **Sprint** was **successful**. A two-week Sprint is successful if the **total points** for **all the tasks** with **status** "**Done**" is **more** or **equal than** the **sum** of **all the points** for the **other 3 types of tasks combined** ("**ToDo**", "**In Progress**" and "**Code Review**").

- If the Sprint was successful, print:
  - **`Sprint was successful!`**
- If the Sprint was **NOT** successful, print:
  - **`Sprint was unsuccessful...`**

Examples

Input	Output

`[`
`'5',`
`'Kiril:BOP-1209:Fix Minor Bug:ToDo:3',`
`'Mariya:BOP-1210:Fix Major Bug:In Progress:3',`
`'Peter:BOP-1211:POC:Code Review:5',`
`'Georgi:BOP-1212:Investigation Task:Done:2',`
`'Mariya:BOP-1213:New Account Page:In Progress:13',`
`'Add New:Kiril:BOP-1217:Add Info Page:In Progress:5',`
`'Change Status:Peter:BOP-1290:ToDo',`
`'Remove Task:Mariya:1',`
`'Remove Task:Joro:1',`
`]`
	`Task with ID BOP-1290 does not exist for Peter!`
	`Assignee Joro does not exist on the board!`
	`ToDo: 3pts`
	`In Progress: 8pts`
	`Code Review: 5pts`
	`Done Points: 2pts`
	`Sprint was unsuccessful...`

Comments

We first receive the **initial status** of the **Sprint board** and parse the information inside our **collection** structure. Then, we start receiving the commands. The **first** one **adds a new task** to the assignee that exists on the board. The **second command** tries to change the status of the task with ID **BOP-1290** that **does not exist** in the collection, so we print a message. The **third command** removes the task successfully. The **final command** tries to remove a task of a **non-existent person**, so we print the error message. In the end, we display a **Sprint review** – ToDo Tasks **{3}**, In Progress Tasks **{3 + 5 = 8}**, Code Review **{5}**, Done Tasks **{2}**. Unfortunately, this Sprint was **unsuccessful**.

Input	Output
`[` `'4',`	`Assignee Sam does not exist on the board!` `Assignee Will does not exist on the board!`

'Kiril:BOP-1213:Fix Typo:Done:1', 'Peter:BOP-1214:New Products Page:In Progress:2', 'Mariya:BOP-1215:Setup Routing:ToDo:8', 'Georgi:BOP-1216:Add Business Card:Code Review:3', 'Add New:Sam:BOP-1237:Testing Home Page:Done:3', 'Change Status:Georgi:BOP-1216:Done', 'Change Status:Will:BOP-1212:In Progress', 'Remove Task:Georgi:3', 'Change Status:Mariya:BOP-1215:Done',]	Index is out of range! ToDo: 0pts In Progress: 2pts Code Review: 0pts Done Points: 12pts Sprint was successful!

Problem 2 – Sprint Planning

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**.

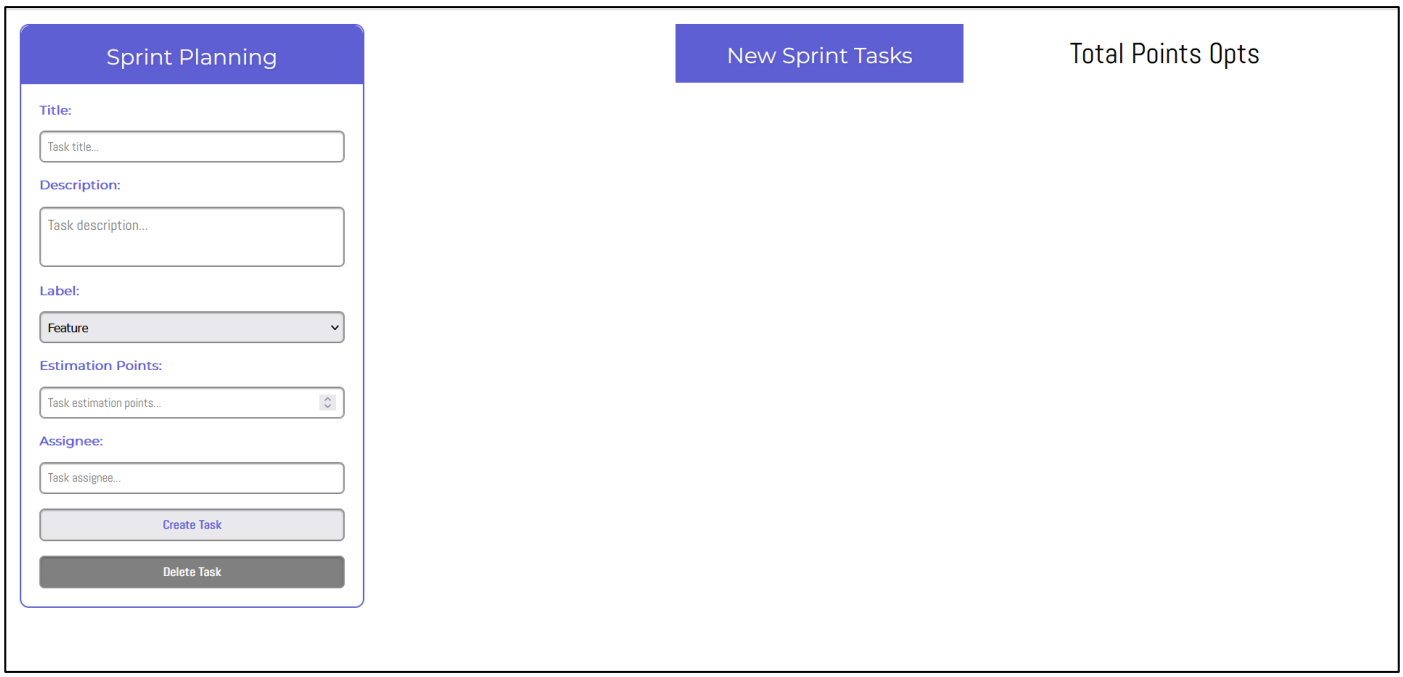
The following actions are **NOT** supported:

- `.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.

Write the missing JavaScript code to plan your next two-week **Sprint**:



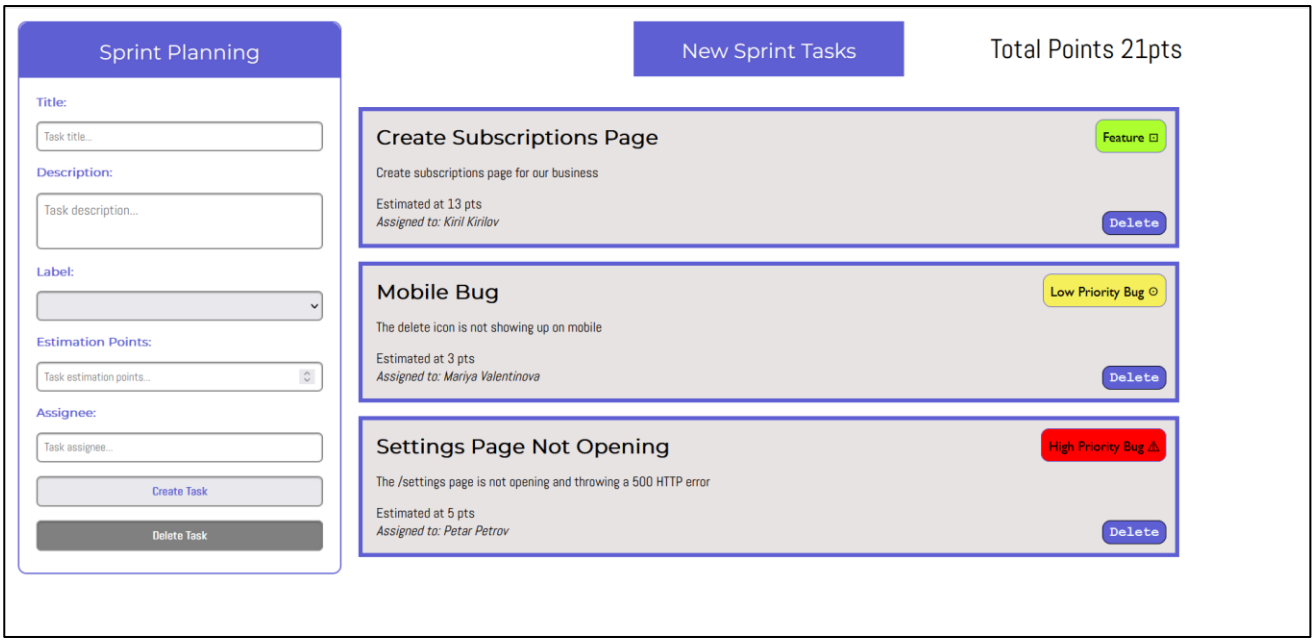
Create a Task

- You're provided with a form that contains the following fields: **Title**, **Description**, **Label** (a select field with 3 options), **Estimation Points**, **Assignee**, and **two buttons** – **[Create Task]** and **[Delete Task]**. After the page initialization, the **[Delete Task]** button is **disabled**.
- After the user **successfully fills out all of the fields** and **clicks** on **[Create Task] button**, it should create a new **article** inside the **`<section>`** with **`id "tasks-section"`**.
- That **article** has the **following HTML structure** *(be careful when you create it and add all of the necessary HTML elements and attributes)*:

```
<section id="tasks-section"> flex
  <h1>New Sprint Tasks</h1>
  <p id="total-sprint-points">Total Points 13pts</p>
<article id="task-1" class="task-card">
    <div class="task-card-label feature">Feature ⊡</div>
    <h3 class="task-card-title">Create Subscriptions Page</h3>
    <p class="task-card-description">
    Create subscriptions page for our business</p>
    <div class="task-card-points">Estimated at 13 pts</div>
    <div class="task-card-assignee">Assigned to: Kiril Kirilov</div>
  <div class="task-card-actions">
      <button>Delete</button> event
    </div>
  </article>
</section>
```

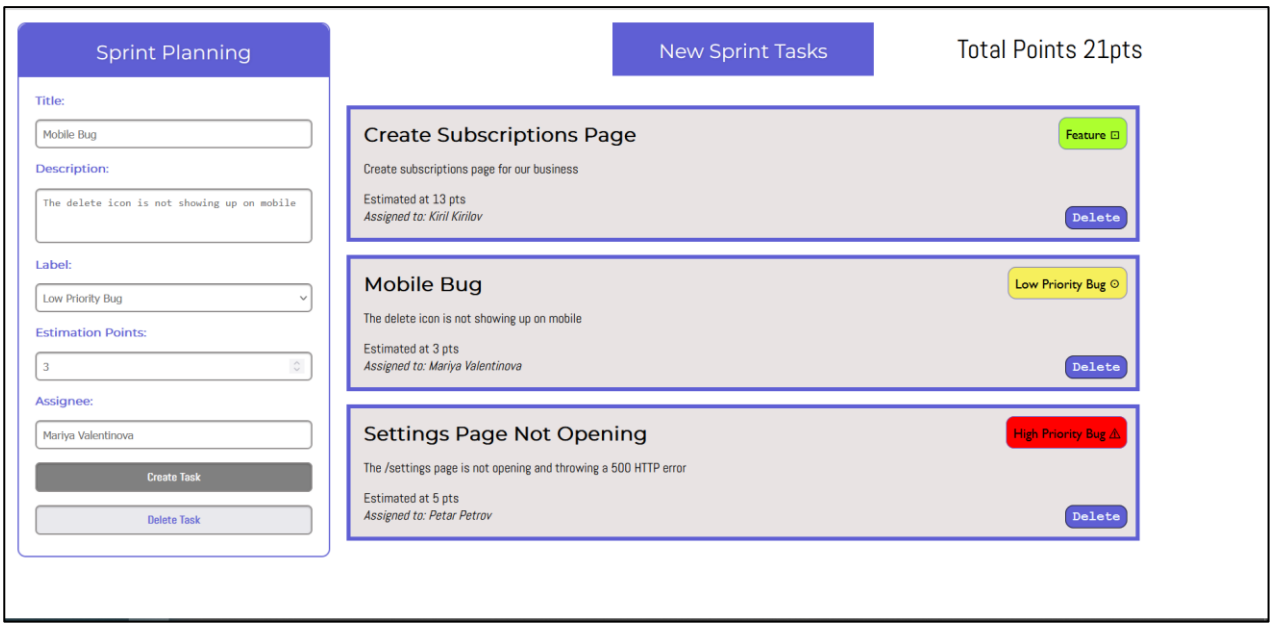
- You have to fill out **ALL of the input fields**, otherwise clicking on the **[Create Task]** button **shouldn't do anything**!
- The **`<div>`** with **`class "task-card-label"`** has different **HTML code icons** next to its name depending on whether it is a **feature**, **low priority bug**, or **high priority bug.** Here are the HTML code variations:
    - Feature: **⊡**
    - Low Priority Bug: **☉**

    - High Priority Bug: **⚠**
- The **label** **`<div>`** also has an **additional** **`class`** that styles it differently, so be sure to add it:
    - feature: **`"feature"`**
    - low priority bug: **`"low-priority"`**
    - high priority bug: **`"high-priority"`**
- Each task should also have an **`id`** attached to the **`<article>`**, which you have to generate in the following format: **`"task-1"`**, **`"task-2"`**, **`"task-3"`** etc. You will need that later!
- Successful creation of a new task should also **clear all of the input fields**!
- An example with all 3 types of tasks:



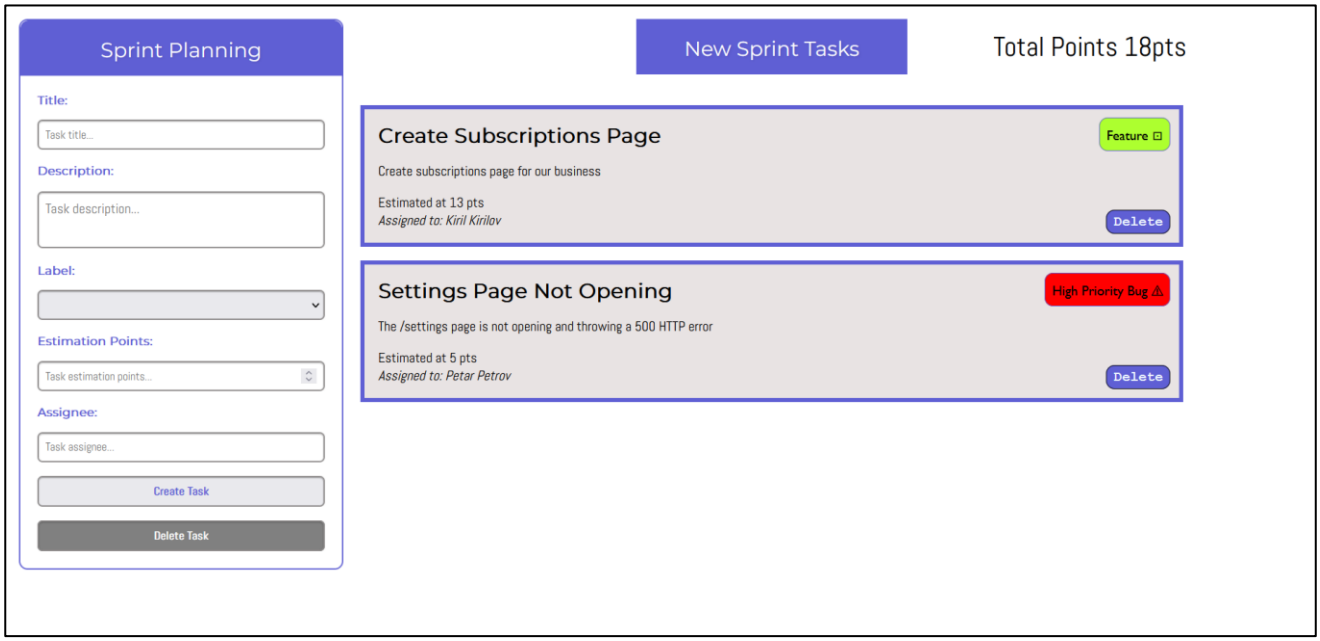
Load Confirm Delete

- Clicking on the **[Delete] button** on the **bottom right corner** of a **task** should **load all** of the information of the **current task** in the **form** on the **left**.
- This action should **enable** the **[Delete Task]** button on the form and also **disable** the **[Create Task]** button
- It should also **make all of the inputs disabled**!
- There is an **`<input>`** of type **`"hidden"`** in the form where you should store the **task ID** you generated when you created the task.



Delete a Task

- Clicking on the **[Delete Task] button** should remove the element from the DOM.
- **Clear out** all of the **fields** and **enable them again** after deleting.
- **Enable** the **[Create Task]** button and **disable** the **[Delete Task]** button.



Total Points

- On the **upper right corner** of the page, there is a **total points** paragraph that needs to be **updated**.
- When **creating a new task** add the new **estimation points** to that counter.
- After **successfully deleting a task** from the **form**, **subtract** the estimation points of the deleted task from the total points.

Problem 3 – Sprint Board

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 tasks. 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
  Sprint Board Tests
    √ Load Board (loads all in correct categories) (501ms)
    √ Create Task (successful create & clear inputs) (532ms)
    √ Move Task (from ToDo to In Progress) (551ms)
    √ Move Task (from In Progress to Code Review) (504ms)
    √ Move Task (from Code Review to Done) (508ms)
    √ Close Task (497ms)


6 passing (4s)
```

Endpoints

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

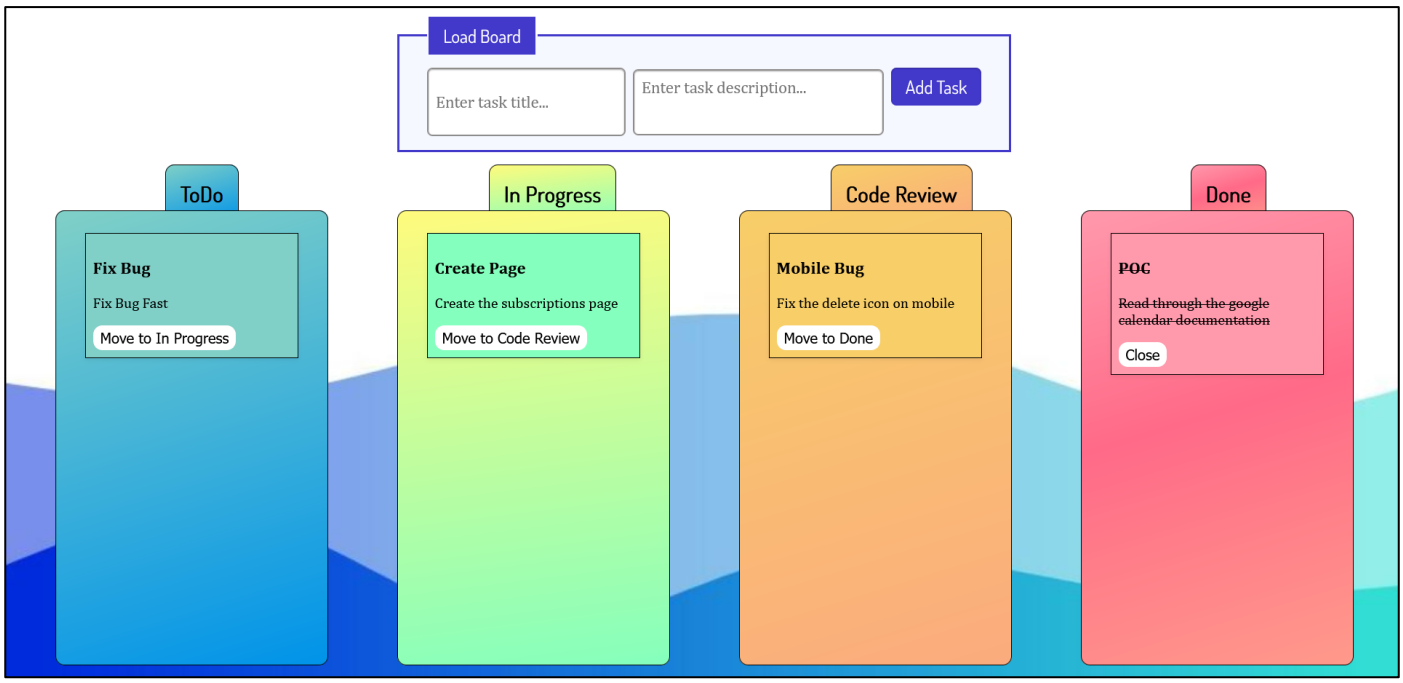
Load the Board

Clicking the **[Load Board]** button on the **top left** should send a **GET** request to the server to fetch **all sprint tasks** in your local database. You have to add each task to its **specified column list** (ToDo, In Progress, Code Review, or Done). The tasks should have **different text contents** inside their respective **buttons** depending on which column they are in:

- ToDo – **"Move to In Progress"**
- In Progress – **"Move to Code Review"**
- Code Review – **"Move to Done"**
- Done – **"Close"**

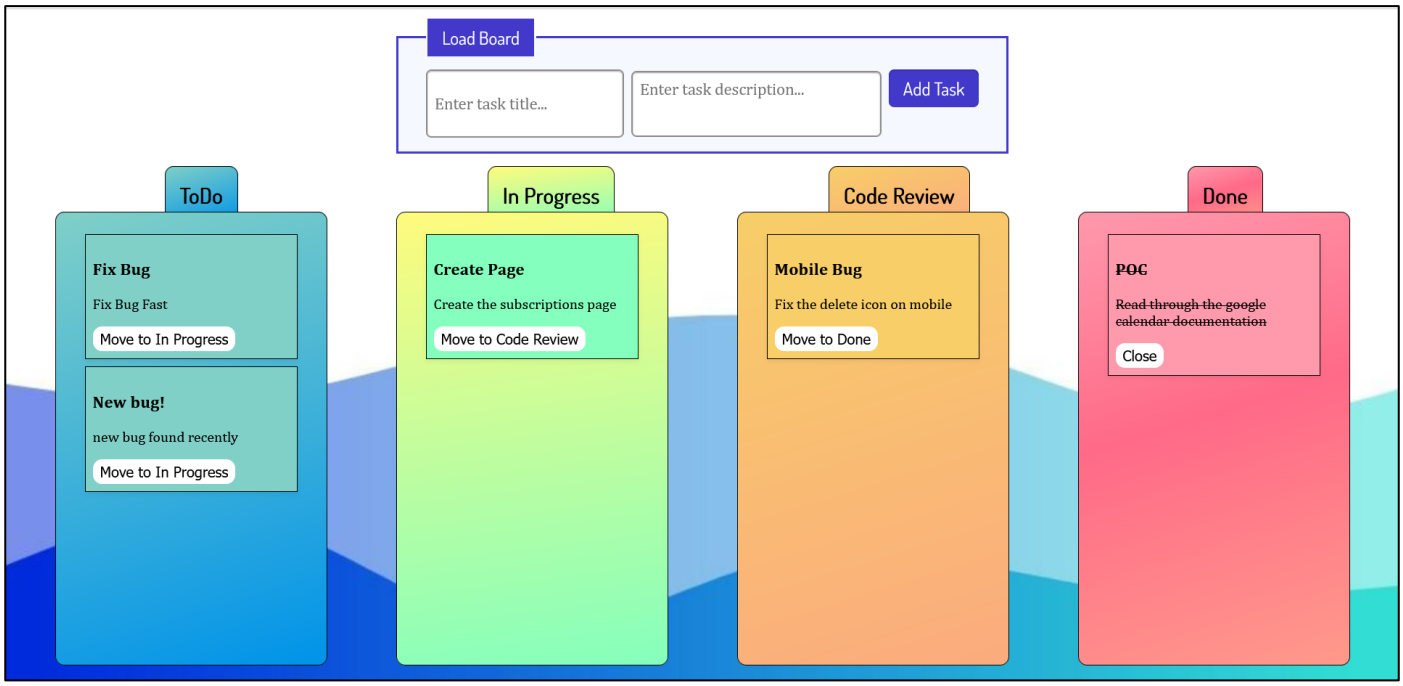
Each Sprint task has the following **HTML structure**:

```
<li class="task">
  <h3>Fix Bug</h3>
  <p>We have a new bug to fix</p>
  <button>Move to In Progress/Move to Code Review/Move to Done/Close</button>
</li>
```



Add a Task

Clicking the **[Add Task]** button should send a **POST** request to the server creating a new task with a **title** and **description** from the input values (the **status** should have an initial value of **'ToDo'**). After a successful creation, you should send another **GET** request to fetch all the tasks, including the **newly added one** into the **ToDo** column. You should also **clear all input fields** after the creation!



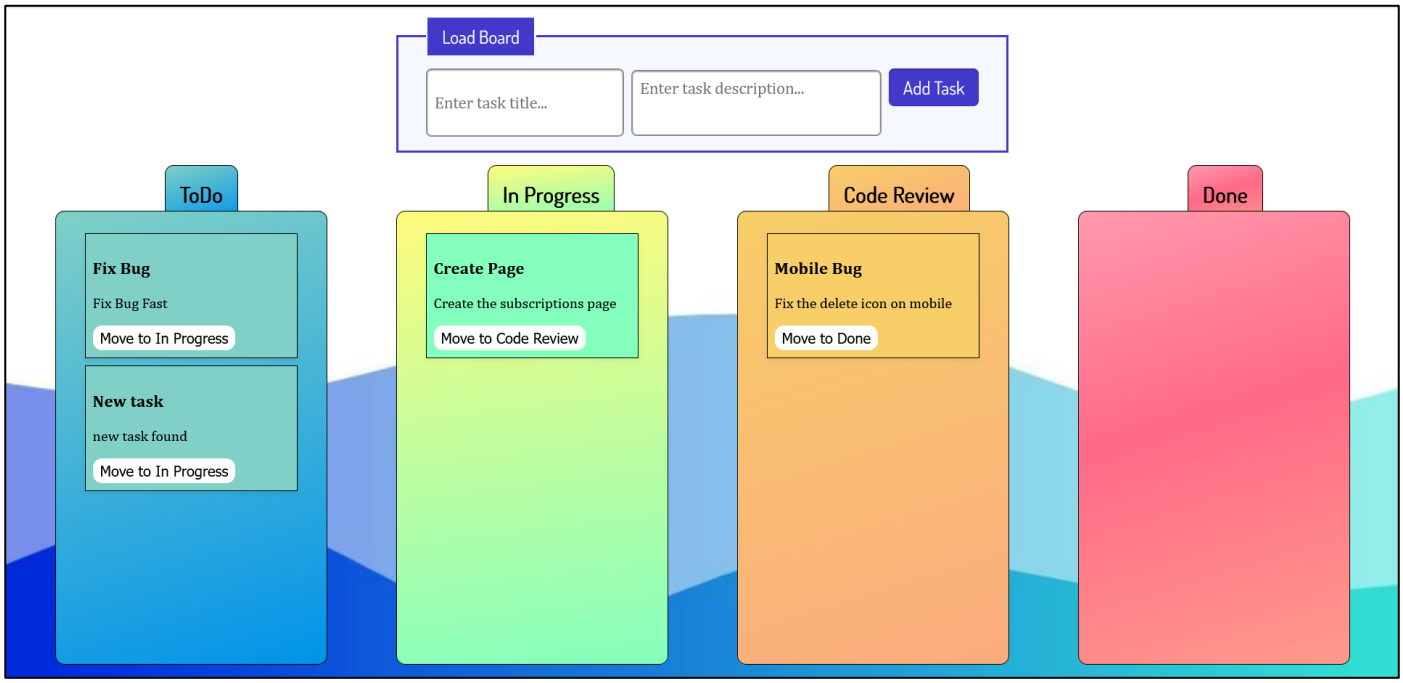
Move a Task

Clicking the **[Move]** button on the individual tasks from the **first 3 columns** should move the task from one column to the next – from **ToDo** to **In Progress** to **Code Review** and finally to **Done**.

After clicking the **[Move]** button, you should send a **PATCH** request to the server to **modify the status** of the changed item. After the successful request, you should **fetch the items again** and see that the change has been made.

Close a Task

Clicking the **[Close]** button on the tasks in the **Done section** 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.