Technical University of Moldova

Faculty of Computers, Informatics and Microelectronics

Department of Software Engineering and Automation

Web Programming

Laboratory work #3

Learn JavaScript basics

Author:
Liviu Mocanu
std. gr. FAF-203

Supervisor:
Alexei Șerșun

Chișinău 2023

1 Task

Our task for this laboratory work was to:

1. Build an application for to-do list
2. The app has to cover basic needs:
    - to add to the list
    - to remove from the list
    - to mark as done
    - see ”done” and ”to-do” lists separately
3. The app has to look attractive

We were not allowed to use any third-party library for JS, except utility packages such as lodash or underscore. jquery was not allowed.

2 Results

Despite the fact that we were supposed to do only 3 additional tasks based on our judgement, I added a number of smaller features that could not count as a whole task per say. Here are the additions I made on my own behalf:

1. An alert **greeting** the user for the first time and asking for their **name**.
    - The name is displayed as `"#{Name}'s To Do List"` as the title of the page
    - Only the name is coloured in blue while the rest of the title maintains the black colour
    - The name is saved in the *localStorage*, therefore the user won't be prompted to insert their name again on page refresh
2. The ability to add a **date** to the task:
    - The date is optional and a user can have both tasks without and with a date in the same list
    - The ability to **toggle** the display of the date
    - The ability to **sort** the tasks by date (those without a date come last)
    - **Reformatting** the date from ”yyyy-mm-dd” as used internally to a more *European-friendly* ”dd/mm/yyyy” format
3. The ability to **drag and drop** tasks. If the tasked that was dropped on is different from the one that was dragged the following should happen:
    - The *text* of the tasks should be swapped
    - The *date* of the tasks should be swapped (while keeping in mind the date formatting differences between system and what is displayed”
    - The *style* of the tasks should swap (if one task is done while the other isn't)
4. A notification **congratulating** the user when checking a task as **done**
5. A **confirmation** from the user when **deleting** a task

All of these modifications use the *localStorage*, therefore even after a refresh, the **order** of the tasks, the **filter** applied (all/completed/ongoing) and the **display** of dates remains the same.

2.1 Greeting and User Name

As per the above mentioned functionality of the alert and user name, we will go over all the points and showcase their implementation:

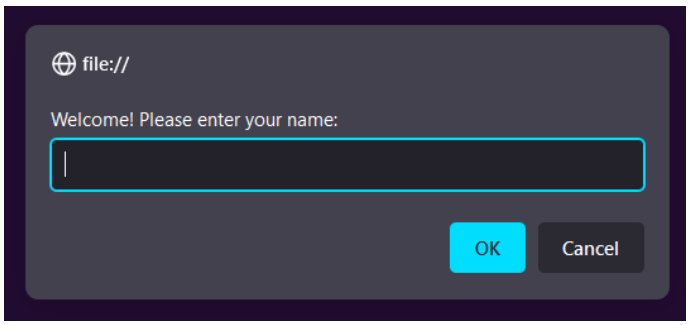
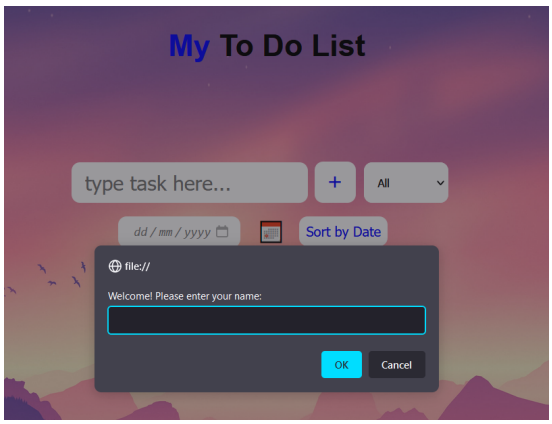


Figure 1: An alert greeting the user for the first time and asking for their name.

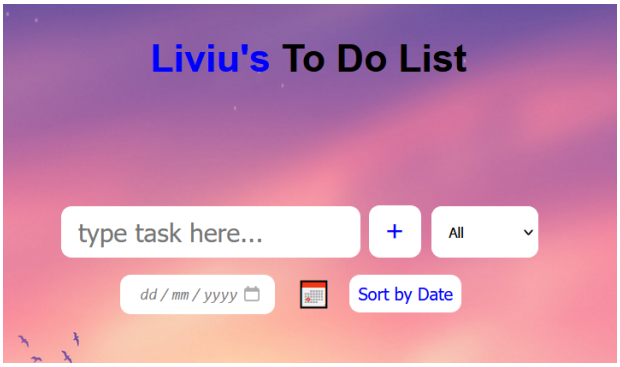
```
document.addEventListener("DOMContentLoaded", () => {
  const savedName = localStorage.getItem("name");
  if (savedName) {
    userName.textContent = `${savedName}'s`;
  } else {
    const inputName = prompt("Welcome! Please enter your name:");
    if (inputName) {
      userName.textContent = `${inputName}'s`;
      localStorage.setItem("name", inputName);
    }
  }
});
```

Listing 1: The greeting alert and user name interpolation js code snippet

We might observe that the code snippet in Listing 1 in fact provides more logic than the Figure 1. That is because we can also showcase the title's change after the name has been introduced:



(a) Title placeholder



(b) Title after name inserted

Figure 2: Name in the title changing to user's name

As previously mentioned, the name stays the same even after a refresh.

2.2 Adding a date to items

Mainly the functionality related to dates is accessed through the following components, located slightly lower than the main ones:

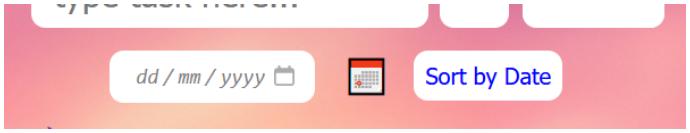
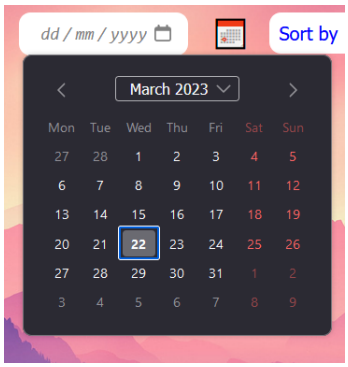


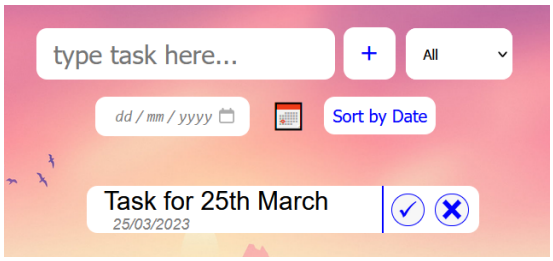
Figure 3: The main date related buttons.

The first button, that is `todo-date`, is responsible for providing a calendar and the ability to choose a date for which this task is scheduled:



(a) Calendar appears on click

(b) Date was set



(c) Task with a date succesfully created

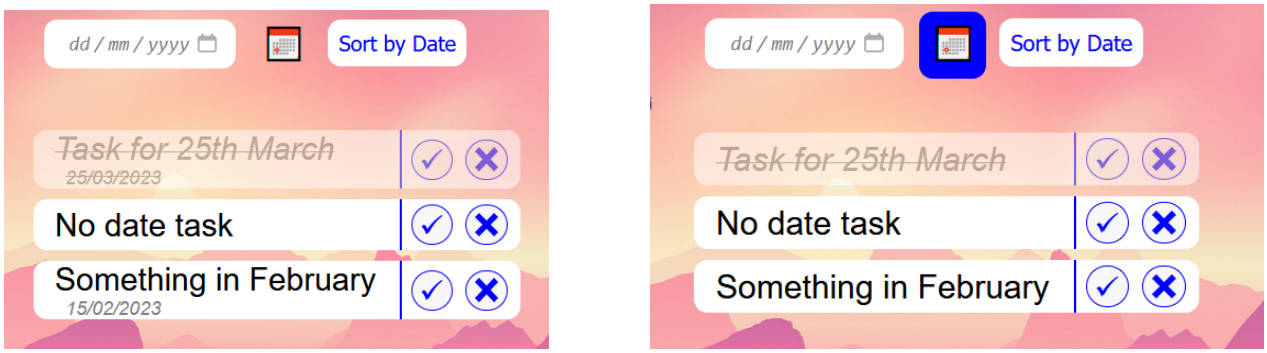
Figure 4: Creating a date for a task using the todo-date button

In fact, as we see in the Listing all the items have a date, it's just that if there is no input from the calendar when creating it, the date is actually an empty string:

```
function addTodoItem(event) {
    event.preventDefault(); // do not reload page

    // Check if the input is empty
    if (!todoInput.value.trim()) {
        alert("Please enter a task.");
        return;
    }

    createNewItem(todoInput.value, todoInputDate.value, false);

    //ADDING TO LOCAL STORAGE
    saveLocalTodos(todoInput.value, todoInputDate.value);
    todoInput.value = "";
    todoInputDate.value = "";
}
```

Listing 2: JavaScript code snippet for adding an item

In case the date in unnecessary and we just want an overview of our tasks, as previously mentioned we can toggle their display on and off using button `toggle-btn` as shown in Figure 5:



(a) Date toggled on (b) Date toggled off

Figure 5: Button `toggle-btn` in use

Basically the function as the js code, iterates through all the elements with class `.todo-date-span` using `dateElements.forEach()` and adds/removes `date-hidden` using `element.classList.toggle()`. This toggles the visibility of the date elements by hiding/showing them:

```
function toggleDateVisibility() {
    const dateElements = document.querySelectorAll(`.${TODO_DATE_SPAN}`);
    dateElements.forEach(function (element) {
        element.classList.toggle("date-hidden");
    });

    // Store current visibility state in localStorage
    localStorage.setItem("dateHidden", dateElements[0]?.classList.contains("date-
    hidden") || "false");
}
```

Listing 3: JavaScript code snippet for toggling the date

Aditionally, we have the button that would Sort our entries by Date, as seen in Figure 6:

(a) Unsorted list (b) List sorted by date

Figure 6: Ordering the list by date

If we take a look at the code in the script file, we'll see that the sorting functionality is implemented as an event listener for the "Sort by Date" button. When the user clicks this button, an anonymous function is called, which retrieves all tasks, sorts them by date, and re-appends them to the list in the sorted order. The sorted order is then saved in localStorage.

```
document.querySelector(".todo-sort-date").addEventListener("click", () => {
    let todos = Array.from(document.querySelectorAll(`.${TODO_ITEM}`));
    todos.sort((a, b) => {
```

```
        const dateA = a.querySelector(`.${TODO_DATE_SPAN}`).innerText;
        const dateB = b.querySelector(`.${TODO_DATE_SPAN}`).innerText;
        if (!dateA) return 1;
        if (!dateB) return -1;
        return new Date(dateA.split("/").reverse().join("-")) - new Date(dateB.split("/").reverse().join("-"));
    });

    todos.forEach((todo) => {
        todoList.appendChild(todo);
    });

    // Save sorted order to local storage
    let sortedTodos = todos.map(todo => getTodoData(todo));
    localStorage.setItem("todos", JSON.stringify(sortedTodos));
});
```

Listing 4: JavaScript code snippet for sorting the list by date

As per the above-mentioned notice, when sorting the list by dates, the tasks without a date go to the end of the list. Yet again, the order in which the tasks are displayed is saved in the localStorage.

2.3 Drag and drop

One of the most important features added was the ability to customize the order in which we want our tasks to be. Unfortunately there is not much potential to illustrate this feature, however we can go over the code and notice what happens behind the scenes.

Firstly, we should activate the drag and drop functionality from the HTML API add the according event listeners:

```
// Activate drag and drop functionality
todoItemDiv.setAttribute("draggable", "true");

// Add drag and drop event listeners
todoItemDiv.addEventListener("dragstart", dragStart);
todoItemDiv.addEventListener("dragover", dragOver);
todoItemDiv.addEventListener("drop", drop);
```

Listing 5: JavaScript code snippet for making te items draggable

`dragStart(e)` is called when the user starts dragging an item. It checks if the target element has the `TODO_ITEM` class, and if so, sets the `draggedItem` variable to the target element. The `effectAllowed` property of the event's `dataTransfer` object is set to "move", and the `setData()` method is called to store the dragged item's HTML content:

```
function dragStart(e) {
    if (e.target.classList.contains(TODO_ITEM)) {
        draggedItem = e.target;
        e.dataTransfer.effectAllowed = "move";
        e.dataTransfer.setData("text/html", e.target.innerHTML);
    }
}
```

Listing 6: JavaScript code snippet for the dragStart function

`dragOver(e)` is called when the dragged item is moved over another element.

By calling `e.preventDefault()`, it allows the dragged item to be dropped on the target element:

```
function dragOver(e) {
    e.preventDefault();
}
```

Listing 7: JavaScript code snippet for the dragOver function

`drop(e)` is called when the dragged item is dropped on a target element. It first stops the event propagation and prevents the default behavior. Then, it checks if the target element is a valid drop target (it has the `TODO_ITEM` class, and is not the same as the dragged item). If the conditions are met, the function swaps the data between the dragged item and the target element, and updates the local storage order:

```
function drop(e) {
    e.stopPropagation();
    e.preventDefault();

    const toDrop = e.target.parentElement.parentElement;

    if (toDrop.classList.contains(TODO_ITEM) && draggedItem !== null && toDrop !==
    draggedItem) {
        const tempData = getTodoData(draggedItem);
        updateTodoData(draggedItem, getTodoData(toDrop));
        updateTodoData(toDrop, tempData);

        // Update local storage order
        updateLocalStorageOrder(draggedItem, toDrop);
```

```
14
15          // Reset draggedItem
16          draggedItem = null;
17      }
18  }
```

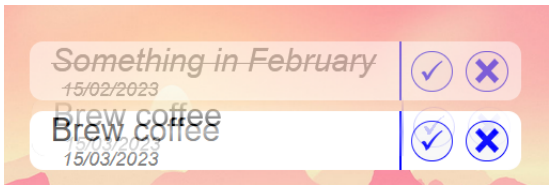
Listing 8: JavaScript code snippet for the drop function

`updateTodoData(todoItem, data)` updates the content and formatting of a todo item based on the given data object. It sets the text and date of the item and adds or removes the "completed" class depending on whether the `completed` property of the data object is true or false. It updates the item's data in local storage:

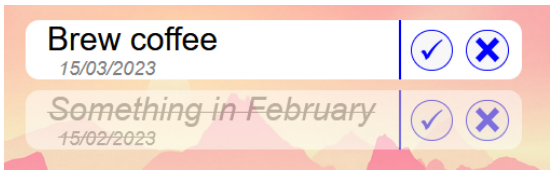
```
1  function updateTodoData(todoItem, data) {
2      let date = formatDateSlashed(data.date);
3      if(date.includes("NaN")) {
4          date = "";
5      }
6      todoItem.querySelector(`.${TODO_TASK}`).querySelector(`.todo-text`).innerText =
       data.text;
7      todoItem.querySelector(`.${TODO_TASK}`).querySelector(`.${TODO_DATE_SPAN}`).
       innerText = date;
8
9      if (data.completed) {
10         todoItem.classList.add("completed");
11     } else {
12         todoItem.classList.remove("completed");
13     }
14
15     // Update local storage
16     updateLocalTodos(todoItem);
17 }
```

Listing 9: JavaScript code snippet for the updateTodoData function

Of course, not only does the text and date of the item swap, but also the formatting in case we are interacting with a completed task:



(a) Dragging

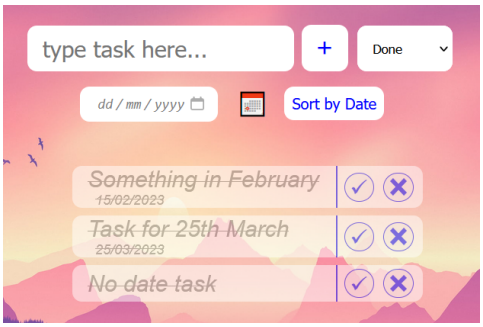


(b) Dropping the task

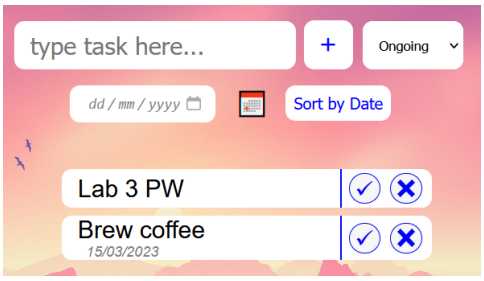
Figure 7: Example of dragging a task on top of another

2.4 Other minor features

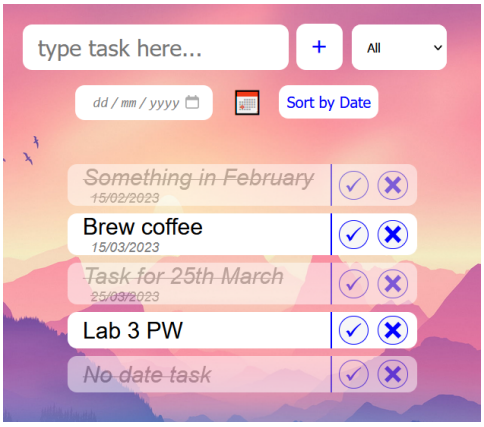
Other notable features that would be otherwise disregarded is the fact that even after applying a filter to the list (for example Done):



(a) Applying the filter Done on the list



(b) Applying the filter Ongoing on the list

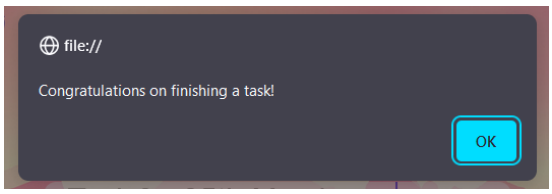


(c) Applying the filter All on the list

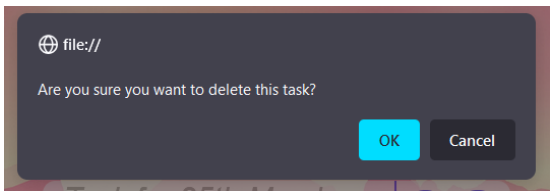
Figure 8: Applying a filter to the list

After a refresh, usually the list would return to showing all the elements. However, I'm using localStorage to prevent that:

Moreover, as I mentioned, I added a notification congratulating the user when checking a task as done as well as a confirmation from the user when deleting a task as we can see in Figure 9



(a) Congratulating alert when finishing a task



(b) Confirmation when deleting a task

Figure 9: Alerts when interacting with to do items

Besides that, I also added some other even more minor changes such as button hovering style change as can be seen in several screenshots so far.

3 Conclusion

This laboratory work for Web Programming involved implementing a to-do list application that meets the basic requirements of adding, removing, and marking tasks as done, as well as viewing them separately as "done" and "ongoing" lists. Additionally, I added several features that enhance the functionality of the application, including the ability to add dates to tasks, toggle the display of dates, sort tasks by date, drag and drop tasks, receive notifications when marking tasks as done, and confirmations when deleting tasks. The application also has an attractive design with a personalized touch by greeting the user by name and displaying it in the page title.

This laboratory work provided a valuable opportunity for me to improve my knowledge and skills in JavaScript. Through this project, I gained a deeper understanding of JavaScript concepts such as local storage, event handling, sorting, and drag and drop functionality.

References

[1] JavaScript tutorial, *https://www.w3schools.com/js/*

Accessed on March 13, 2023.

[2] EventTarget.addEventListener() tutorial, *https://developer.mozilla.org/en-US/docs/Web/API/EventTarget/addEventListener*

Accessed on March 14, 2023.

[3] Window.localStorage tutorial, *https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage*

Accessed on March 17, 2023 .

[4] Date in JS, *https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date*

Accessed on March 18, 2023 .

[5] HTML Drag and Drop API *https://www.w3schools.com/html/html5_draganddrop.asp*

Accessed on March 20, 2023.