UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II
BACHELOR OF SCIENCE IN COMPUTER SCIENCE
WEB TECHNOLOGIES COURSE – SPRING 2024
TEACHER: Luigi Libero Lucio Starace, Ph.D.

DATE: April 5, 2024
BEST DONE BEFORE: April 12, 2024

ASSIGNMENT: #07 TOPIC: JAVASCRIPT – ASYNCHRONISM

EXERCISE 1. IMPROVING OUR FIRST TODO APP

In **EXERCISE 1** of **ASSIGNMENT 06**, you implemented a To-do list application in a web page using nothing but JavaScript. The application you developed is simple, ergonomic and effective. However, it is affected by a little flaw: upon reloading the web page, all our to-do items are gone! That is a little inconvenient for an app whose goal is to help people keep track of things!

In this exercise, you will address this limitation, using the Web Storage API. You should start from the code you developed in Exercise 1 of Assignment 06 and change it so that the to-do items are properly saved and can survive a full page reload, and a full browser restart.

Beware: this exercise is not as easy as it may seem! Here's a few hints, in case you need some help:

- `localStorage` can only store strings. In this exercise, you might need to store an array of structured data. You can use the JSON.stringify(object) to serialize an object into a string and JSON.parse(string) to de-serialize a string into the corresponding object.
- In Assignment 06, you might have stored the to-do items directly and only in the DOM of the web page. In that exercise, you only needed to append a new item to the list, to toggle a class on *click*, or to delete a `<li>` DOM element on *dblclick*. In other words, the full state of the application was encoded in the DOM. In this exercise, there need to be two different representations of the current list of to-do items: one is in the DOM of the web page (the one users can see and interact with), and the other is stored in the `localStorage`. These two representations need to be aligned and consistent at every time. The suggested way to tackle this issue for this exercise is to start by focusing only on the `localStorage` array of to-do items. Then, implement a function that updates the list in the DOM so that it reflects the contents of the `localStorage`, and make sure to call that function everytime the to-do list in the `localStorage` changes. For this exercise, you can simply delete all the list items from the list, and re-create the list items that are currently stored in `localStorage`. This is not the most efficient way to proceed (e.g.: we're re-creating the entire list from scratch even though only one element changed!), but we will settle with this solution for now. Tackling this issue more efficiently requires a little more efforts, and is not required for this exercise. We will get back to this challenge when we discuss *state management* in single page applications, later during the Web Technologies course.
- If you experience issues with the *dblclick* event (which may not fire, as two separate *click* events are fired instead), you can switch to a right click to delete to-dos ("*contextmenu*" event).

ASSIGNMENT: #07 TOPIC: JAVASCRIPT – ASYNCHRONISM

EXERCISE 2. WEB TECH'S FIRST VIRTUAL ASYNCHRONOUS MUSEUM

The Art Institute of Chicago (AIC) is one of the oldest and largest art museums in the United States, with a permanent collection of more than 300 thousand artworks, including masterpieces such as *Nightawks* by Edward Hopper, *American Gothic* by Grant Wood, and *A Sunday on la Grande Jatte* by Georges Seurat.

The AIC also manages a free API to access all of its artworks virtually. In this assignment, you'll build a web page that uses the API to dynamically display artworks.

- Start from the provided **museum.html** file. Create the *script.js* and a *style.css* files.
- When the user clicks on the button, get the input from the input field, and then use the fetch() API to send a GET request to

```
"https://api.artic.edu/api/v1/artworks/search?limit=100&q=QUERY"
```

- In the above URL, replace **QUERY** with the keywords provided by the user.
- For the above request, the API will return a JSON containing (in its `data` property) a list of at most 100 artworks that matched with the provided keywords.
- Randomly select one of the artworks (if any) returned by the previous API call. Each artwork object contains an `api_link` property. Such property is the link to which a GET request should be sent to get more details about the artworks.
- Send a new HTTP request to the `api_link` of the selected artwork. The call will return a new JSON object with a `data` property containing data about the artwork. This data object contains a `title` property, which indicated the title of the artwork, an `artist_title` property representing the name of the author, and an `image_id` property. The URL of the image representing the selected artwork can then be computed as

```
`https://www.artic.edu/iiif/2/${image_id}/full/843,/0/default.jpg`
```

- Update the web page to properly display the image and the correct title and author of the artwork.
- Make sure to properly handle the scenario in which no artwork matching the current keywords is found.
- When you are done with acquiring data using the APIs, make your virtual museum look presentable on both mobile and desktop devices. Feel free to style the web page as you like. The final result should be somewhat similar to the images provided hereafter.

UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II
BACHELOR OF SCIENCE IN COMPUTER SCIENCE
WEB TECHNOLOGIES COURSE – SPRING 2024
TEACHER: Luigi Libero Lucio Starace, Ph.D.

DATE: April 5, 2024
BEST DONE BEFORE: April 12, 2024

ASSIGNMENT: #07 TOPIC: JAVASCRIPT – ASYNCHRONISM