Web Applications [2025/2026]

Lab 1: Getting started with Node.js

In this lab, you will become familiar with JavaScript by putting into practice what you have seen in the last lectures.

0. Preparation and warm-up exercise

Before starting, make sure that Visual Studio Code and Node.js have been correctly installed on your computer. **Check the course slides for instructions, if necessary.**

Beware: at exam time, we will evaluate your assignments using Node.js on Linux. Therefore, if you develop on Windows, we suggest you to configure Node.js inside Windows Subsystem for Linux (WSL) 2. See instructions in the first set of slides of the course.

Create a function that, given an array of strings, for each string computes and prints a new one composed by the first two and last two characters. For instance, '`spring`' yields '`spng`'.

If the word is shorter than two characters, it prints an empty string. Otherwise, if the word is two or three characters long, the function prints the same character two times. For instance, '`it`' yields '`itit`' and '`cat`' yields '`caat`'.

Write some test instructions that call the function with a variety of strings and check the correctness of the results.

1. Create a Film Library

In this exercise, you will implement a simple application to track the films that a person wants to watch and the ones they have already watched. Each film is represented by the following fields:

- A unique numerical **id** (mandatory)
- A **title** (mandatory)
- A Boolean value to represent whether the film is among the person's **favorites** (default value: false)
- A date corresponding to the **date** when the person **watched the film** (optional)
- A numerical value between 1 and 5 to represent the **rating** that the person has given to the film after watching it (optional)

First, implement a constructor function to create **Film** objects.

Second, implement a constructor function to create a **FilmLibrary**, an object containing an array of Films.

Then, implement the **`addNewFilm`** method, which adds a new **Film** object, passed as parameter, to the **FilmLibrary**. For simplicity, the film id is decided by who created the **Film** object and no further checks are needed.

Finally, using the last method, **populate** the **FilmLibrary**. For instance, you can take inspiration from the following list:

```
Id: 1, Title: Pulp Fiction, Favorite: true, Watch date: March 10, 2023, Score: 5
Id: 2, Title: 21 Grams, Favorite: true, Watch date: March 17, 2023, Score: 4
Id: 3, Title: Star Wars, Favorite: false, Watch date: <not defined>, Score: <not assigned>
Id: 4, Title: Matrix, Favorite: false, Watch date: <not defined>, Score: <not assigned>
Id: 5, Title: Shrek, Favorite: false, Watch date: March 21, 2023, Score: 3
```

To verify that you correctly populated the FilmLibrary, implement a **print** method. This method prints in the console the whole list of Films stored by the FilmLibrary.

Hint: you may use the day.js library to create and handle the dates.

2. Add functionalities to the Film Library

In this exercise you will add a set of methods to the **FilmLibrary** object and write some test instructions to invoke them.

Hint: To implement the required functionalities described below you may use the functional programming paradigm to manipulate the array of films.

`sortByDate`: returns a new array containing the Films within the **FilmLibrary** instance sorted in ascending order of the watch date. The films that the user has not already watched should be put at the end. For example, after the sorting, the **FilmLibrary** shown in the previous exercise would look like:

```
***** List of films *****
Id: 1, Title: Pulp Fiction, Favorite: true, Watch date: March 10, 2023, Score: 5
Id: 2, Title: 21 Grams, Favorite: true, Watch date: March 17, 2023, Score: 4
Id: 5, Title: Shrek, Favorite: false, Watch date: March 21, 2023, Score: 3
Id: 3, Title: Star Wars, Favorite: false, Watch date: <not defined>, Score: <not defined>
Id: 4, Title: Matrix, Favorite: false, Watch date: <not defined>, Score: <not defined>
```

`deleteFilm`: deletes a **Film** from the **FilmLibrary** based on an Id received as a parameter.

`resetWatchedFilms`: deletes the Watch date of all the **Films** in the **FilmLibrary**.

`getRated`: selects the films that do have a defined score. Only films with an assigned score should be returned, ordered by decreasing score. After filtering the Films Library shown in exercise 1, the method should print:

```
***** Films filtered, only the rated ones *****
Id: 1, Title: Pulp Fiction, Favorite: true, Watch date: March 10, 2023, Score: 5
Id: 2, Title: 21 Grams, Favorite: true, Watch date: March 17, 2023, Score: 4
Id: 5, Title: Shrek, Favorite: false, Watch date: March 21, 2023, Score: 3
```

Finally, test the methods by invoking them over the **FilmLibrary** instance you created and populated in exercise 1.