PEC 1. Desarrollo front-end con framew. JavaScript

Ignacio Casares Ruiz

Marzo de 2023

Información

- **Login**: icasaresr
- **Nombre y apellidos**: Ignacio Casares Ruiz

Breve descripción de lo realizado en la PEC

Para esta PEC, se nos pedía instalar npm y Visual Studio Code. En ambos casos se trata de herramientas que yo ya utilizaba, por lo que no he tenido que realizar la instalación original de ninguna de ellas. Sin embargo, he instalado la extensión Live Server de VSCode, según lo sugerido en la tarea 1 de la PEC. También tenía ya instalado git, por lo que no he tenido que realizar ese paso tampoco.

Posteriormente, he creado, según la tarea 2, el repositorio local en git a través del comando `git init`, y he modificado mi nombre y mi e-mail a través de `git config`. He comprobado que la información era la correcta a través de `git config --list`.

A continuación, se ha creado este fichero en formato `.md`, se ha añadido al repositorio y realizado un commit simultáneamente mediante el comando `git add . && git commit -m "Added README.md file"`.

A lo largo de la elaboraciño de la práctica, se ha utilizado git para añadir cambios y actualizarlos mediante `add` y `commit`. En alguna ocasión he tenido que cancelar *staged changes* a través de la utilización del comando `git reset HEAD`. También se utilizó `git status` para confirmar los cambios antes de realizar el `commit` en varias ocasiones.

Ejercicio 2.1: descripción y mejoras

- Se ha añadido el atributo `async` al elemento `script` de forma que este no bloquee la carga de los elementos HTML y «haga esperar» al usuario que carga la página. Esto se ha hecho teniendo en cuenta que los scripts utilizados no son de gran envergadura y que no va a afectar a la funcionalidad global de la página.
- En la función `isValidEmail(email)` que analiza si el e-mail es válido, se ha preferido utilizar el método `match()` para comprobar si las condiciones impuestas por la expresión regular se cumplen. Además, se ha sustituido la expresión regular propuesta por una menos restrictiva.
- Al crear la función que devuelve una palabra con la primera letra en mayúscula, se ha preferido utilizar una función de uso global, en lugar de una específica para las strings de tipo `input.id` y pasar este tipo de strings con el método como argumento.

La principal dificultad en la realización de este ejercicio pasó por tener que constuir las expresiones regulares que nos permitiesen abordar el filtro de contraseña exigido en el enunciado. En este caso, tuve que recurrir a regex101 para poder testear y conseguir una guía que me permitiese resolver esta problemática adecuadamente. A partir de este ejercicio, me he puesto como objetivo aprender a utilizar `regex` mediante el uso de un sistema de repetición espaciada como Anki.

Ejercicio 2.2: descripción y mejoras

- Se ha utilizado la versión 6 de la API en lugar de la 4 como se especifica en el vídeo.
- Se ha utilizado, en lugar de la imagen propuesta, una imagen libre almacenada en Wikimedia Commons. At.: Jon Opprecht, CC0 1.0 (UPDD). Enlace.
- Se ha añadido un botón de carga durante las peticiones a la API. Este se ha confeccionado puramente en CSS y aparecerá al lado del botón «swap». Se ha realizado mediante la adición de un elemento `div` con una clase adicional que actualiza la propiedad CSS de `visibility: hidden;` a `visibility: visible;` mediante modificaciones a la variable `loader.className` de la función `calculate()`:

```
function calculate() {
    loader.className = 'loadersmall loading'
    let currency_one = currencyEl_one.value;
    let currency_two = currencyEl_two.value;


    ↪ fetch(`https://v6.exchangerate-api.com/v6/370d55c233982cc0f62a011c/latest/${curre
        .then(res => res.json())
        .then(data => {
            let rate = data.conversion_rates[currency_two]
            rateEl.innerText = `1 ${currency_one} = ${rate}
↪ ${currency_two}`
            amountEl_two.value = (amountEl_one.value * rate).toFixed(2)
            loader.className = 'loadersmall'
        })
}
```

- Se ha añadido un mensaje de error. Se puede comprobar cambiando la moneda a «ERROR» (esta moneda se ha añadido a propósito para poder comprobar su correcto funcionamiento).

Esta práctica no dio demasiados problemas. En la resolución de la parte del enunciado que pedía crear un mensaje de error tuve que averiguar cómo devolvía la promesa de la API en casos de error, pero esta tiene un mecanismo sencillo con el que no resulta complejo lidiar.

Ejercicio 2.3: descripción y mejoras

La realización la primera parte del ejercicio se realizó sin mayor problema. Siguiendo el vídeo instructivo y utilizando mdn ocasionalmente pudo resolverse de forma adecuada.

Respecto a la parte que no estaba incluida en el vídeo, sí existieron algunos problemas:

- En este caso, hubo especial dificultad a la hora de abordar la modificación del texto de las opciones de las diferentes películas para que se adecúe a la conversión de moneda según el menú select construido en la parte superior de la pagína. En un principio se intentó abordar esta problemática a través de la creación de elementos `span` que serían modificados a través del código `JS`. Sin embargo, al comprobar que HTML lee los `span` dentro de los elementos `option` como parte de un mergeado del `innerText` del elemento y no como un elemento individual, se descartó este abordaje y se decidió construir un nuevo `string` a través del uso de `template literals`.
- Además, tuve problemas a la hora de modificar los strings de la página en los usuarios que accediesen por primera vez. Esto lo solucioné creando una función `firstCurrencyApplier()` específica para este fin.

Se han utilizado comentarios para describir exhaustivamente las funciones de las distintas variables y funciones declaradas a lo largo del archivo `script.js`.