

# PEC 2

## Lenguajes de desarrollo front-end



Universitat Oberta  
de Catalunya

### Información relevante:

- Fecha límite de entrega: 25 de octubre.
- Peso en la nota de FC: 15%.



## Contenido

Información docente	3
Presentación	3
Objetivos	3
Enunciado	4
Ejercicio 1 – Arrays (3,5 puntos)	4
Ejercicio 2 – Callbacks/Promesas/async-await (2 puntos)	6
Ejercicio 3 – Arquitectura MVC usando VanillaJS (4 puntos)	8
Ejercicio 4 – Conociendo TypeScript (0,5 puntos)	12
Formato y fecha de entrega	13

# Información docente

## Presentación

Esta Práctica se centra en conocer las mejoras en el lenguaje de JavaScript a lo largo de los últimos años basados en el estándar ECMAScript, proporcionando fundamentos del lenguaje Web que es utilizado por todos los frameworks de hoy en día. Además, se comienza a introducir TypeScript, que es el superconjunto adoptado por la comunidad como la mejor opción para crear código JavaScript que escala.

## Objetivos

Los objetivos que se desean lograr con el desarrollo de esta PEC son:

- **Desarrollar** código JavaScript **usando las características de ES6-ES11**.
- **Utilizar el patrón de arquitectura MVC** en el desarrollo de una aplicación Web.
- Comprender la importancia de desarrollar código usando **TypeScript**.

# Enunciado

Esta PEC contiene 4 ejercicios evaluables. Debéis entregar vuestra solución de los 4 ejercicios evaluables (ver el último apartado).



Debido a que las actividades están encadenadas (i.e. para hacer una se debe haber comprendido la anterior), **es altamente recomendable hacer las tareas y ejercicios en el orden en que aparecen en este enunciado.**

Antes de continuar debéis:

Haber leído el recurso teórico `T01.PEC2_Teoria_2020.pdf` disponible en el apartado contenidos y recursos del aula de esta PEC.

## Ejercicio 1 – Arrays (3,5 puntos)

Utilizando los métodos nativos de `array` como son *filter*, *some*, *map*, *reduce*, .... debes completar los ficheros `core.js` que se suministran en el proyecto `Ejer1_Array_Methods.zip` incluido en la PEC.

Antes de continuar debes:

Descargar fichero `PEC2_Ej1_Array_Methods.zip` adjunto en la PEC

Para realizar el ejercicio solo debes modificar los ficheros `core.js`, **no se deben modificar los ficheros de tests `core.spec.js`.**

El objetivo es conseguir que la ejecución de los test de las pruebas definidas dé como resultado los tests `passed`.

Instrucciones:

- Instalar mocha con el comando `npm install -g mocha`.
- Completar el código del fichero `core.js` de cada subcarpeta
- Ejecutar cada uno de los test con la instrucción `mocha <nombre prueba>` por ejemplo `mocha a_map`

**En Windows** el paso anterior puede dar error la primera vez que se ejecuta.

Si obtenéis un error como:

```
mocha.ps1 cannot be loaded because running scripts is disabled on this system. For more information, see about_Execution_Policies at ...
```

Debéis realizar estos pasos en VS Code:

- File -> Preferences -> Settings y en la barra de búsqueda buscar "automation".
- Después de esto, pulsad en "edit in settings.json" del sistema operativo Windows
- Añadid esta línea:  
  

```
"terminal.integrated.shellArgs.windows": ["-ExecutionPolicy", "Bypass"]
```
- Guardad y reiniciad VSCode.
- Ya no debería dar ese error al ejecutar mocha.

Tenéis que completar el código del fichero `core.js` y conseguir que los test den como resultado `passed` para todas las carpetas suministradas en el zip.

- a. map – 0.5 puntos
- b. filter – 0.5 puntos
- c. reduce – 0.5 puntos
- d. every – 0.5 puntos
- e. some – 0.5 puntos
- f. zoo (ejercicio completo) – 1 punto

Si necesitas más información, el siguiente enlace dispone de abundante documentación sobre los métodos nativos de Array:

[https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\\_Objects/Array](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array)

## Ejercicio 2 – Callbacks/Promesas/async-await (2 puntos)

Antes de continuar debéis:

Descargar fichero `PEC2_Ej2_Callbacks_Promesas_Async-await.zip` adjunto en la PEC

```
const findOne = (list, { key, value }, { onSuccess, onError }) => {
  setTimeout(() => {
    const element = list.find(element => element[key] === value);
    element ? onSuccess(element) : onError({ msg: 'ERROR: Element Not Found' });
  }, 2000);
};

const onSuccess = ({ name }) => console.log(`user: ${name}`);
const onError = ({ msg }) => console.log(msg);

const users = [
  {
    name: 'Carlos',
    rol: 'Teacher',
  },
  {
    name: 'Ana',
    rol: 'Boss',
  },
];

console.log('findOne success');
findOne(users, { key: 'name', value: 'Carlos' }, { onSuccess, onError });

console.log('findOne error');
findOne(users, { key: 'name', value: 'Fermin' }, { onSuccess, onError });

/*
findOne success
findOne error
//wait 2 seconds
user: Carlos
ERROR: Element Not Found
*/
```

- El código anterior hace uso de *callbacks* para realizar una tarea después de hacer la búsqueda en un *array* (`find`). Crea un fichero `ejer2-a.js` y documenta en el código, que se está realizando en cada línea, haciendo hincapié en el callback y el valor mostrado por la pantalla. (0.5 puntos)
- Modifica el código anterior para eliminar los *callbacks* y, en su lugar, hacer uso de promesas (hay que crearlas en la función `findOne`) y consumirlas en el código principal (con las palabras reservadas `then` y `catch`). Documenta en el código, línea a línea, que cambios has realizado. (0.5 puntos)

- c. Modifica el código anterior para hacer uso de `async/await` (ten en cuenta que deberás separar en una función el uso de la función `async`. Es decir, podrás crear una función extra en caso de ser necesario). Explica línea a línea a línea, dentro del propio código, qué has modificado. (0.5 puntos)
- d. El código obtenido tras aplicar `async/await` se ha vuelto secuencial. Escribe y documenta en el propio código, como sería, usando promesas y `async/await` la versión paralela, es decir, que se ejecuten todas las llamadas a la función `findOne` en paralelo, sin necesidad de esperar a que concluya la primera para ejecutarse la segunda. (0.5 puntos)

## Ejercicio 3 – Arquitectura MVC usando VanillaJS (4 puntos)

En este ejercicio vamos a construir, usando todas las novedades de JavaScript, una aplicación SPA (Single Page Application) sin utilizar ningún framework, lo que se conoce como programar usando VanillaJS o JavaScript puro. Es decir, VanillaJS es la referencia para crear aplicaciones SPA basadas en JavaScript que no hacen uso de ningún framework (p.ej. Angular o Vue). Eso sí, seguiremos la arquitectura de MVC (Modelo-Vista-Controlador). Esta práctica te permitirá conocer la arquitectura que siguen frameworks como Angular, pero utilizando como lenguaje base JavaScript.

En primer lugar, debemos comprender cómo se construye una aplicación usando el patrón de arquitectura MVC utilizando VanillaJS. Para ello, os facilitamos un proyecto ya terminado llamado `Ejer3 - TODO` que consiste en una aplicación SPA que realiza las 4 operaciones elementales CRUD (Create, Read, Update y Delete) sobre una lista de tareas (TODO). Una vez comprendido el proyecto `Ejer3 - TODO`, tendréis que hacer una aplicación similar llamada `Ejer3 - MVC Users`.

Antes de continuar debéis:











- Descargar el fichero `PEC2_Ej3_Arquitectura MVC.zip` adjunto en la PEC, que contiene los dos proyectos `Ejer3 - TODO` y `Ejer3 - MVC Users`.
- Leer el recurso `P02.PEC2_Ejemplo_Arquitectura_MVC.pdf` que explica cómo está construido el proyecto `Ejer3 - TODO`.



Una vez visto el ejemplo de una aplicación completa usando el patrón MVC **programada en** VanillaJS, debéis crear una aplicación MVC de usuarios. Para ello debéis seguir los pasos del anterior código para crear un CRUD sobre un registro de usuarios. Es decir, se realizarán las operaciones de crear, borrar, editar y mostrar un registro de usuarios.

En el código facilitado para comprender el MVC se realizan todas las operaciones sobre objetos TODO, en este caso, el dominio del problema ha cambiado para ser registros de Usuarios.

El código fuente inicial de la práctica (Ejer 3 – MVC Users) genera la siguiente tabla:

Manage Employees					
				Delete	Add New Employee
<input type="checkbox"/>	Name	Email	Address	Phone	Actions
<input type="checkbox"/>	Thomas Hardy	thomashardy@mail.com	89 Chiaroscuro Rd, Portland, USA	(171) 555-2222	 
<input type="checkbox"/>	Dominique Perrier	dominiqueperrier@mail.com	Obere Str. 57, Berlin, Germany	(313) 555-5735	 
<input type="checkbox"/>	Maria Anders	mariaanders@mail.com	25, rue Lauriston, Paris, France	(503) 555-9931	 
<input type="checkbox"/>	Fran Wilson	franwilson@mail.com	C/ Araquil, 67, Madrid, Spain	(204) 619-5731	 
<input type="checkbox"/>	Martin Blank	martinblank@mail.com	Via Monte Bianco 34, Turin, Italy	(480) 631-2097	 
Showing 5 out of 25 entries				Previous 1 2 3 4 5 Next	

La información de los usuarios que queremos gestionar es la siguiente:

- Name
- Email
- Address
- Phone

Además, al pulsar el botón de editar o borrar se generará una ventana modal (lanzada por jQuery y que sólo habrá que mover a su fichero correspondiente y bindear/enlazar con el servicio a través del controlador) en la cual aparecerán formularios que permitirán introducir la información de los usuarios.

Antes de empezar con el desarrollo de MVC – Users, responded a la siguiente pregunta sobre Ejer 3 – TODO.

- a. (0.50 puntos) Observad que se han creado funciones *handle* en el fichero controlador (`todo.controller.js`), las cuales son pasadas como parámetro. Esto es debido al problema con el cambio de contexto (*this*) que existe en JavaScript. Ahora mismo si no tienes muy claro que está sucediendo, revisa qué hacen las “fat-arrow” de ES6 sobre el objeto *this*, y prueba a cambiar el código para comprender qué está sucediendo cuando se modifica la siguiente línea:

```
this.view.bindAddTodo(this.handleAddTodo);
```

Por esta:

```
this.view.bindAddTodo(this.service.addTodo);
```

Responded, en un documento texto en el directorio de entrega a la siguiente pregunta:

¿Por qué es el valor de *this* es *undefined*?

A continuación, lo que se pide es modificar el código del proyecto `Ejer 3- MVC Users` para convertir un simple `.html` en una estructura de MVC haciendo uso del potencial de ES6 y siguiendo el código de ejemplo (TODO) que se ha proporcionado.

- b. (0.50 puntos) Analizad el código que se ha facilitado para el desarrollo de este ejercicio (directorio `MVC-Users`). Observaréis que existe un único fichero `.html` donde hay pedazos de código relacionados con la apariencia (`html` y `css`) y otros con la interacción (`JS`) haciendo uso de la biblioteca `jQuery` para manipular el `DOM`. En primer lugar, debéis generar la estructura de directorios de nuestro proyecto (modelos-vistas-servicios-controladores) y separar el contenido del fichero `index.html` en diferentes ficheros, de modo que el fichero `index.html` solamente contenga la estructura relativa a la Tabla (alguien podría incluso pensar en llevarse el contenido de `html` de la tabla a otro fichero y daría lugar a su primer componente, pero esto es adelantarnos en el curso).

- c. (0.50 puntos) Construid las clases relativas a modelos, controladores, servicios, vistas y lanzador de la aplicación desde donde iréis desarrollando la aplicación. En este punto sólo debéis crear la estructura de ficheros que modelan nuestro problema. Es decir, organizar las clases relativas a modelos (`user.model.js`), controladores (`user.controller.js`), servicios (`user.service.js`) y lanzadora (`app.js`).
- d. (0.50 puntos) Construid la clase modelo (anémico) que sea necesaria para esta aplicación.
- e. (1 punto) Construid la clase servicio que es la encargada de realizar todas las operaciones sobre una estructura de datos (donde se almacenará la información de todos los usuarios).
- f. (0.50 puntos) Construid la clase vista que controlará todas las operaciones relativas a la vista. Ten en cuenta que la vista sólo debe repintar la tabla y lanzar los modales con la información pertinente y comunicar la acción al controlador.
- g. (0.50 puntos) Construid el controlador que es el encargado de poner en comunicación la vista con el servicio, en este proyecto.

## Ejercicio 4 – Conociendo TypeScript (0,5 puntos)

TypeScript es un superconjunto sobre JavaScript que permite generar código de JavaScript que escale, según la propia definición de TypeScript. Es decir, está pensado para ser usado cuando JavaScript comienza a tener una complejidad de aplicación, dejando de lado su faceta de script.

En el ejercicio anterior se ha realizado una pantalla en la cual se hacía la gestión de una lista de TODOs y de usuarios. Con ella habéis podido comprobar cómo la dificultad de gestionar JavaScript ha ido incrementando exponencialmente a medida que se introducen características/funcionalidades en la aplicación. No obstante, si analizáis con cuidado la aplicación que se ha construido, no es más que una pantalla de las decenas que puede tener una aplicación web.

En la siguiente PEC profundizaremos en TypeScript y las ventajas que nos aporta. No obstante, en este ejercicio os proponemos que vosotros mismos veáis algunas de las ventajas que TypeScript tiene sobre JavaScript. Para ello os pedimos que comparéis el “mismo” código escrito en JavaScript y en TypeScript.

Antes de continuar debéis:

- Descargar fichero **PEC2\_Ej4\_Conociendo\_TypeScript.zip**

- Consultar el siguiente video sobre Typescript:

[https://learning.oreilly.com/videos/understanding-typescript/9781789951905/9781789951905-video1\\_2](https://learning.oreilly.com/videos/understanding-typescript/9781789951905/9781789951905-video1_2)

Ahora analizad el código en JavaScript (**Ejer4.js**) y su versión en TypeScript (**Ejer4.ts**)

En un fichero de texto que se entregará como solución de este ejercicio, enumera y explica las principales ventajas que aporta el uso de TypeScript sobre JavaScript.

# Formato y fecha de entrega

Tienes que entregar un fichero \*.zip, cuyo nombre tiene que seguir este patrón: loginUOC\_PEC2.zip. Por ejemplo: dgarciaso\_PEC2.zip. Este fichero comprimido tiene que incluir los siguientes elementos:

- Una carpeta PEC2\_Ej1 con el código completado siguiendo las peticiones y especificaciones del Ejercicio 1.
- Una carpeta PEC2\_Ej2 con el código completado y comentado siguiendo las peticiones y especificaciones del Ejercicio 2. Como lo que se solicita es modificar el código ejer2.js, debéis entregar un ejer2.js modificado para cada apartado de los solicitados. Es decir, debéis entregar cuatro ficheros: ejer2-a.js, ejer2-b.js, ejer2-c.js y ejer2-d.js. No es necesario entregar ni el original ejer2.js, ni el index.html, pero si es muy importante que todo el código que se entregue este comentado con los cambios realizados y el motivo de ellos.
- Una carpeta PEC2\_Ej3
  - Con un fichero texto (sirve pdf, doc, txt, md,...) que tenga como nombre S03\_PEC2\_Solución\_ Ejercicio\_3a y responda a la pregunta formulada en el apartado A del Ejercicio 3.
  - Una subcarpeta Ej3 - MVC Users con el código completado siguiendo las peticiones y especificaciones del Ejercicio 3
  - No es necesario entregar la carpeta ejemplo Ej3 - TODO
- Un documento de texto S04\_PEC2\_Solución\_Ejercicio\_4 (sirve cualquier tipo de documento pdf, doc, txt, md,...) que contenga el enunciado y las respuestas a las preguntas del Ejercicio 4.

El último día para entregar esta PEC es el **25 de octubre de 2020** hasta las **23:59**. Cualquier PEC entregada más tarde será considerada como no presentada.