

Examen

## Final julio 2018

**Aplicaciones Web**  
Año 2017/2018  
Grado en Ing. Software  
Fdi - UCM

### Instrucciones generales:

- La duración del examen es de **tres horas**. Su peso en la calificación de la asignatura es del 60 %. Para obtener la calificación de APTO es necesaria la nota mínima de 4 sobre 10.
- No se permite ningún tipo de material sobre la mesa, salvo un bolígrafo y este enunciado.
- El examen consta de dos ejercicios que se realizan en el ordenador. Para su realización se proporciona un proyecto plantilla. Tras el enunciado de cada ejercicio se indica qué ficheros hay que modificar o añadir.
- **Para entregar:** sigue las instrucciones que están al final del enunciado.

### ▷ 1. Aplicaciones web con *Express.js* y *EJS*

Queremos realizar una aplicación web que gestione las reservas de mesas en un restaurante. La base de datos dispone de una única tabla (llamada mesas) que contiene la información sobre las mesas del restaurante y sus reservas (Figura 1).

Nombre columna	Tipo	Atributos
<u>id</u>	INT	NOT NULL, PRIMARY_KEY, AUTO_INCREMENT
posicion	VARCHAR(100)	NOT NULL
num_sillas	INT	NOT NULL
nombre_reserva	VARCHAR(100)	
comensales	INT	NOT NULL, DEFAULT 0

Figura 1: Descripción de la tabla mesas.

La columna `id` es el número de mesa, que toma el rol de clave primaria. La columna `posicion` es una cadena de texto que indica dónde se encuentra la mesa dentro del restaurante. El atributo `num_sillas` indica el número máximo de personas que caben en la mesa. En el caso en que una mesa esté reservada, los atributos `nombre_reserva` y `comensales` contienen el nombre de la persona que ha realizado la reserva, y el número de personas que ocuparán la mesa reservada. Este número siempre es inferior a `num_sillas`. Si una mesa no tiene reserva, los atributos `nombre_reserva` y `comensales` tomarán el valor `NULL` y `0`, respectivamente. La Figura 2 contiene algunos ejemplos.

Suponemos, también, que existe una clase DAO con un único atributo (`pool`), que es una referencia a un *pool* de conexiones a la base de datos.

- (a). [1 pt] Implementa dentro de la clase DAO un método asíncrono `obtenerTodasMesas(callback)` que obtenga, para cada una de las mesas de la base de datos: su número, su posición, el número

id	posicion	num_sillas	nombre_reserva	comensales
1	Entrada izqda.	4	NULL	0
2	Entrada dcha.	3	NULL	0
3	Pasillo 1	6	Adrián	5
4	Pasillo 1	4	NULL	0
⋮	⋮	⋮	⋮	⋮

Figura 2: Datos de ejemplo para la tabla mesa.

de asientos ocupados, y el número de asientos libres. Para ello debe construir un array de objetos, cada uno de ellos con cuatro atributos: `id`, `posicion`, `numOcupadas` y `numLibres`. En aquellos casos en los que la mesa no esté reservada, el atributo `numOcupadas` tomará el valor 0, y el atributo `numLibres` será igual al número de sillas total de la mesa. Por ejemplo, si la tabla `mesas` contiene los datos de la Figura 2, debe construirse el siguiente array:

```
[
  { id: 1, posicion: "Entrada izqda.", numOcupadas: 0, numLibres: 4 },
  { id: 2, posicion: "Entrada dcha.", numOcupadas: 0, numLibres: 3 },
  { id: 3, posicion: "Pasillo 1", numOcupadas: 5, numLibres: 1 },
  ...
]
```

El resultado debe ser pasado a la función `callback` recibida como parámetro, utilizando el convenio habitual de las funciones asíncronas de *Node.js*, esto es, un argumento para indicar el posible error (o `null` en caso de éxito), y otro argumento para el resultado de la operación.

- (b). [2 pt] Dentro de una aplicación *Express.js*, implementa un manejador de ruta `app.get('/', ...)` que devuelva un documento HTML con una tabla como la de la Figura 3. En esta tabla se muestra, para cada mesa: su identificador, su posición, una cadena SÍ o NO, en función de si la mesa está reservada o no, y una representación gráfica de las sillas de la mesa. Esta última contiene tantos cuadrados verdes como asientos libres hay en la mesa, y tantos cuadrados rojos como asientos ocupados hay en la mesa.
- (c). [1.5 pt] Implementa, dentro de DAO, un método asíncrono `obtenerMesaLibre(numComensales, callback)` que devuelva el identificador de la mesa no reservada más pequeña que pueda acoger al número de comensales pasado como parámetro. Si existen varias mesas iguales que cumplen este criterio, puede devolverse cualquiera de ellas. Si no existen mesas libres, o todas las mesas libres tienen menos asientos que el número de comensales pedido, la función devolverá `null`.

**Nota:** Este apartado debe realizarse utilizando una única consulta SQL paramétrica que devuelva una única fila con el identificador de la mesa a devolver, o ninguna fila en el caso en que se deba devolver `null`. No está permitido obtener todas las mesas de la BD y recorrer posteriormente el array de filas buscando la mesa pedida. El resultado obtenido debe ser pasado a la función `callback`, utilizando el convenio habitual en *Node.js*.

Núm. mesa	Posición	Reservada	Ocupación
1	Entrada izq.	NO	4 green squares
2	Entrada dch.	NO	3 green squares
3	Pasillo 1	SÍ	6 red squares, 1 green square
4	Pasillo 1	NO	4 green squares
5	Pasillo 1	NO	4 green squares
6	Pasillo 2	SÍ	4 red squares
7	Pasillo 2	NO	6 green squares
8	Reservado	NO	10 green squares

Figura 3: Vista de las mesas ocupadas y disponibles.

- (d). [1 pt] Supongamos que hemos añadido a la aplicación web del apartado (b) un formulario para hacer reservas, como el mostrado en la Figura 4. Este formulario contiene dos cuadros de texto en los que introducir un nombre y un número de comensales para la reserva. Escribe el manejador de ruta de este formulario para que se busque una mesa disponible y se actualice la BD con la información proporcionada. Puedes suponer que existe en la clase DAO el siguiente método asíncrono,

```
buscarYReservarMesa(nombre, numComensales, callback)
```

que realiza, dentro de la misma transacción, la búsqueda de la mejor mesa disponible (según el apartado (c)) y la actualización de la fila correspondiente a esa mesa con los datos introducidos. Después llama a la función `callback` pasando, además del objeto error (en caso de producirse) el número de mesa reservada o con `null` si no pudo hacerse una reserva.

Si alguno de los campos del formulario está vacío, o el campo *Número de comensales* no contiene un número estrictamente mayor que cero, deberá redirigirse a la ruta `/ej1_error.html`. Si, aún siendo la información del formulario correcta, no existe ninguna mesa libre que pueda acoger al número de comensales indicado, se deberá redirigir a la ruta `/ej1_no_place.html`.

- (e). [1.5 pt] Modifica la aplicación para que “recuerde” a aquellos usuarios que han realizado una reserva. En particular, si un usuario ya ha hecho una reserva, al acceder a la ruta `’/’` se deberá mostrar el mensaje mostrado en la Figura 5 (indicando el número de mesa reservada), en lugar del formulario de reservas. Utiliza, para ello, atributos de sesión.

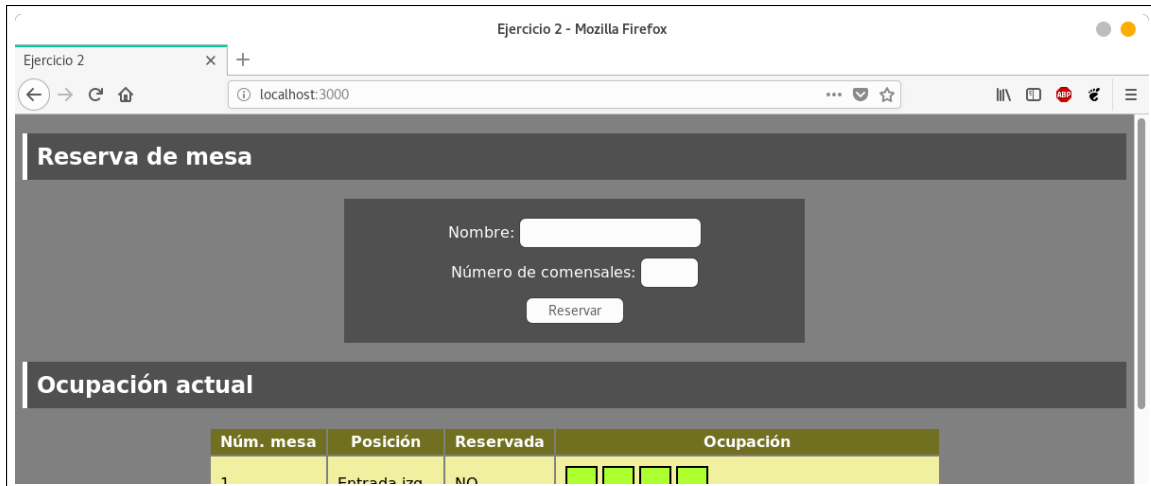


Figura 4: Formulario de reserva de mesas.

#### Instrucciones para el ejercicio 1

- Realiza los apartados (a) y (c) modificando el fichero `ej1_dao.js`. Allí se explican las acciones que tienes que realizar en caso de no haber realizado el ejercicio, para que puedas continuar realizando el resto de apartados.
- Realiza los apartados (b), (d) y (e) modificando el fichero `ej1.js`. Puedes utilizar como base el fichero `public/ej1_plantilla.html` para la elaboración de la plantilla EJS.
- Puedes comprobar la validez del apartado (c) ejecutando el fichero `ej1_test.js`, que contiene una batería de tests.

#### ▷ 2. AJAX, jQuery y servicios web

Supongamos que queremos hacer una encuesta sobre los lenguajes de programación más populares. Creamos una aplicación web en la que los usuarios podrán votar sus lenguajes de programación favoritos, y el servidor mantendrá una variable global `lenguajes` con los resultados actuales de la encuesta. Por ejemplo:

```
let lenguajes = {
  "C++": 25,
  "Java": 20,
  "Javascript": 21
};
```

En este caso se indica que el lenguaje C++ ha obtenido 25 votos, el lenguaje Java ha obtenido 20, y Javascript ha obtenido 21.

Esta aplicación proporciona un servicio web de tipo REST que devuelve los resultados actuales de la encuesta:

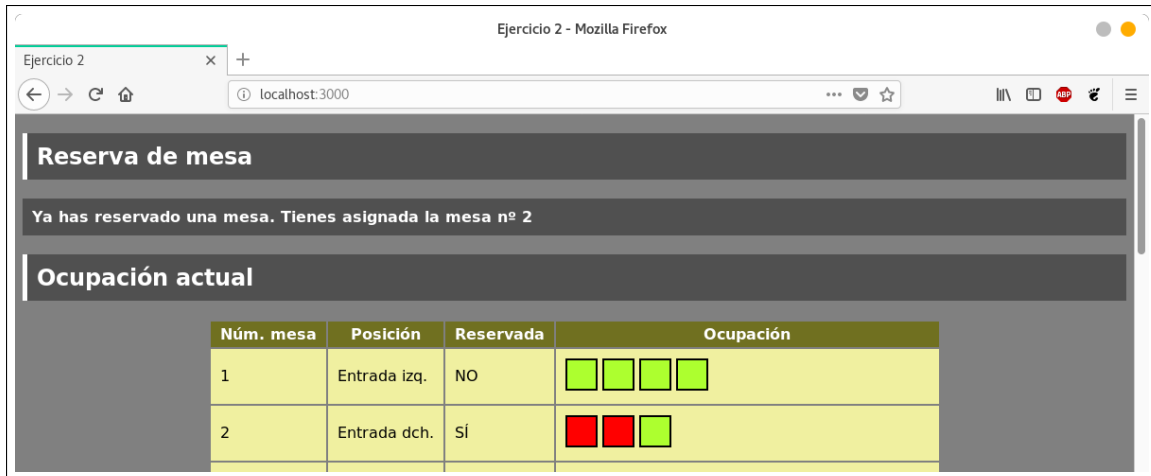


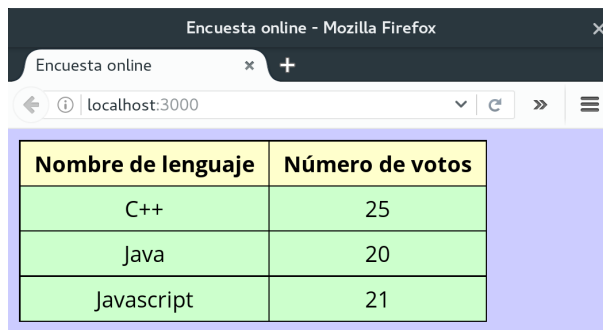
Figura 5: Mensaje mostrado a los usuarios que ya habían reservado una mesa.

```
app.get("/ranking", (request, response) => {
    response.json(lenguajes);
});
```

Por otro lado, supongamos una página web `index.html` que contiene una tabla como la siguiente:

```
<script src="rankingLenguajes.js"></script>
...
<table id="ranking">
  <thead>
    <tr>
      <th>Nombre de lenguaje</th>
      <th>Número de votos</th>
    </tr>
  </thead>
  <tbody>
    <!-- Aquí irán los resultados de la votación -->
  </tbody>
</table>
```

- [1.5 pt]* Implementa una función `actualizarRanking()` dentro de `rankingLenguajes.js` para que realice una llamada AJAX al servicio `/ranking` y añada los resultados a la tabla HTML mostrada anteriormente. Cada resultado es una fila con dos celdas: una con el nombre del lenguaje y otra con el número de votos obtenidos por el mismo (Figura 6).
- [1.5 pt]* Implementa un servicio web para añadir un voto a la encuesta. El servicio recibirá el nombre de un lenguaje de programación y le sumará un voto al atributo correspondiente dentro del objeto `lenguajes`. Si el atributo no existiese dentro de este objeto, se añadirá un nuevo atributo para ese lenguaje, asociado a la cantidad de 1 voto.



The screenshot shows a web browser window titled 'Encuesta online - Mozilla Firefox'. The address bar displays 'localhost:3000'. The main content area contains a table with two columns: 'Nombre de lenguaje' and 'Número de votos'. The table has three rows of data: C++ with 25 votes, Java with 20 votes, and Javascript with 21 votes.

Nombre de lenguaje	Número de votos
C++	25
Java	20
Javascript	21

Figura 6: Tabla con los resultados de la encuesta.

**Método:** PUT

**URL:** /ranking/:nombre

**Parámetros de entrada:** El nombre del lenguaje al que se desea añadir un voto. Se encuentra dentro de la variable nombre de la URL paramétrica.

**Códigos de respuesta:** 200 (OK) o 500 si hubo error.

**Tipos resultado:** Ninguno.

**Resultado:** Ninguno.

### Instrucciones para el ejercicio 2

- Realiza el apartado (a) dentro de public/rankingLenguajes.js. Para comprobar si es correcto puedes arrancar el servidor contenido en ej2.js y acceder a <http://localhost:3000>.
- Para realizar el apartado (b) modifica el fichero ej2.js. El formulario contenido en la página anterior hace uso de este servicio Web para añadir un voto a la encuesta. Puedes utilizarlo para comprobar si tu solución es correcta.

### Instrucciones de entrega

- ☐ Comprueba que has rellenado el fichero Alumno.txt.
- ☐ Crea un fichero comprimido (.zip) con el proyecto Node (incluyendo la carpeta node\_modules)
- ☐ El nombre del fichero tiene que ser de la forma *DNI\_Apellido.zip*. Incluye la letra del DNI y solamente el primer apellido.
- ☐ Entrega el fichero .zip utilizando el enlace del escritorio *Exámenes en LABs - Entregas*. En la carpeta que aparece seleccionar *ALUMNOS entrega de prácticas y exámenes* y subir el .zip.
- ☐ **Antes de cerrar sesión**, dirígete al puesto del profesor para comprobar que el fichero se ha subido correctamente.