Desarrollo web
Backend

Programación y administración de redes - Práctica 1

Grado en *Ingeniería Informática*

Departamento de Informática. Universidad de Jaén

La WWW como plataforma software

Definición

La World Wide Web, en origen una red de documentos de texto entrelazados mediante hipervínculos, es en la actualidad una de las plataformas de ejecución de aplicaciones más importantes.

Conceptos

- La arquitectura de las aplicaciones web se ajusta al modelo **cliente/servidor**
- En el servidor, el software que actúa como ***backend*** queda a la escucha de recibir peticiones procedentes del cliente
- El cliente opera como interfaz de usuario o ***frontend***, traduciendo las acciones del usuario en operaciones sobre el *backend*
- Los verbos del protocolo HTTP (`GET`, `POST`, `PUT`, etc.) determinan las operaciones a efectuar, dando lugar a lo que se conoce como API REST (*Representational State Transfer*)
- El intercambio de información suele hacerse en formato JSON (*JavaScript Object Notation*)

Desarrollo *backend* de una aplicación web

El desarrollo del *backend* es el primer paso en la creación de una aplicación web.

- En el *backend* se establecen los *endpoints* de servicio: verbo HTTP y **URI de acceso** a los que accederá el *frontend*
- Por cada *endpoint*, que sería el equivalente a una función en un lenguaje como C++ o Java, hay que determinar los **parámetros de entrada y de salida**
- El desarrollo *backend* implica elegir una solución concreta:
  - El **lenguaje de programación** en que se implementará
  - El **servidor web** (en caso necesario) que interactuará con el protocolo HTTP
  - La solución de **almacenamiento de los datos** en el servidor
- Las soluciones disponibles son numerosas:
  - Java con **JSP** y un servidor de aplicaciones de la plataforma Java
  - **PHP** como lenguaje y Apache/IIS Server/Nginx como servidor web
  - Ruby on Rails, Python con Django, ASP.NET MVC …
- Una de las alternativas más populares es **Node.js con Express**

Node.js

¿Qué es y cómo funciona Node.js?

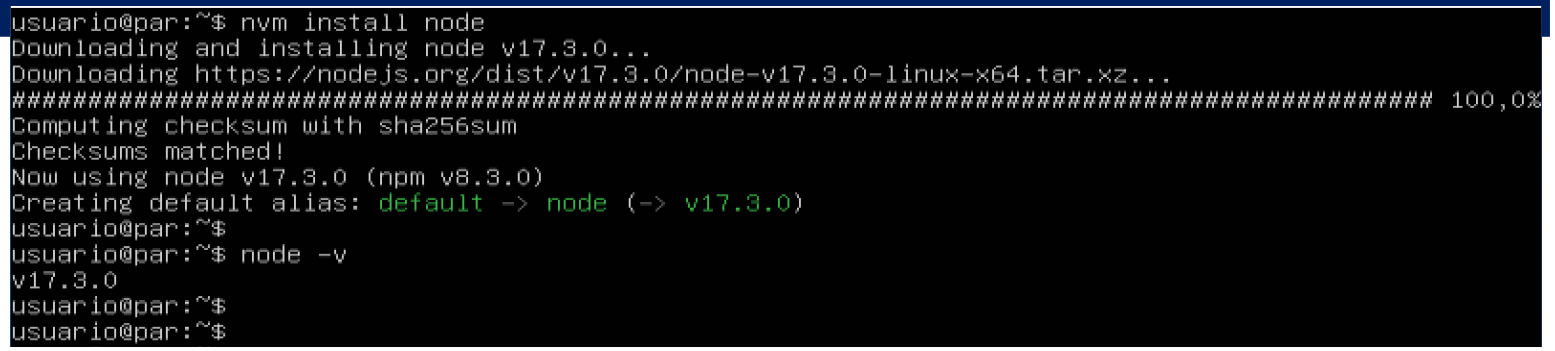
- Node.js es un entorno de ejecución para el lenguaje **JavaScript** basado en el motor V8 de Chrome
- Con Node.js es posible escribir aplicaciones de servidor (*backend*) con el mismo lenguaje que se emplea en el *frontend*, lo que **ahorra esfuerzo** de desarrollo
- Principales características de Node.js:
  - Solución de código abierto y **multiplataforma** (para varios sistemas operativos)
  - Modelo de ejecución **asíncrona** diseñado para ofrecer un alto rendimiento
  - No precisa de un servidor web externo
- Cómo funciona Node.js:
  - Opera como un **entorno** capaz de ejecutar aplicaciones escritas en JavaScript
  - Aporta un **modelo de objetos** propio que facilita la creación de programas de servidor
  - Extensa colección de **paquetes** disponibles para multitud de tareas
- Usaremos Node.js en GNU/Linux, pero su funcionamiento sería análogo en Windows, macOS y otros sistemas operativos

Instalación de Node.js

Proceso de instalación

Hay diferentes alternativas para instalar Node.js. En nuestro caso usaremos la utilidad NVM (*Node Version Manager*). Abre la consola de Ubuntu y reproduce los pasos indicados a continuación:

Acción	Comando a ejecutar en la consola
Descarga e instalación de nvm	`$ curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.0/install.sh | bash`
Instalación de Node.js	`$ nvm install node`
Verifica la instalación	`$ node -v`



Completado el primer paso **deberás cerrar (^d) y volver a abrir** la consola para completar la configuración de `nvm`, tras lo que podrá instalarse Node.js y verificar su versión

Primer ejercicio con Node.js

Desde la consola de Ubuntu ejecuta el comando `nano hola.js` para crear y editar el que será nuestro primer ejercicio con Node.js. A continuación, introduce estas líneas en él:

Servidor web que escucha devuelve una página con un título

```
const http = require('http');  // Cargar el paquete http para aceptar solicitudes HTTP

const puerto = 1234;           // Puerto TCP en el que el servidor esperará peticiones
const servidor = 'localhost';  // Nombre o IP del servidor

http.createServer( (request, response) => {                    // Creamos un servidor con una función
   response.writeHead(200, { 'Content-Type': 'text/html'} );    // que responde enviando una cabecera
   response.end('<h1>Hola Node.js</h1>');                       // con código 200 y el tipo MIME
}).listen(puerto, servidor, () => {                            // La aplicación queda a la espera de
   console.log(`Aplicación a la escucha en el puerto ${puerto}`);// peticiones
});
```

1. Guarda el archivo y vuelve a la consola
2. Ejecuta el comando `node hola.js` para poner en marcha la aplicación
3. Desde otra consola usa `lynx localhost:1234` para enviar la solicitud y ver el resultado en pantalla

Endpoints

Esta aplicación tiene un único *endpoint* que responde al verbo `GET` de HTTP sin ningún parámetro como entrada

Primer ejercicio con Node.js - Análisis

Examinemos la parte central del código de este ejercicio:

```
http.createServer( (request, response) => {
   response.writeHead(200, { 'Content-Type': 'text/html'} );
   response.end('<h1>Hola Node.js</h1>');
}).listen(puerto, servidor, () => {
   console.log(`Aplicación a la escucha en el puerto ${puerto}`);
});
```

- El método `createServer()` de `http` crea un componente servidor y lo devuelve como resultado. Podríamos almacenarlo en una variable: `var aplicacion = http.createServer(frespuesta);`
- Dicho método toma como argumento una función que recibirá dos parámetros:
  - Un objeto con la información de la **solicitud** recibida (parámetro `request` en el ejercicio)
  - Un objeto que permite elaborar la **respuesta** a devolver (parámetro `response`)
- En este ejercicio dicha función no se ha definido antes, sino que se define en la propia llamada a `createServer()` en forma de función *lambda*, pero podría usarse la **sintaxis de la derecha** y obtener el mismo resultado
- El método `listen()` del objeto devuelto por `createServer()` pone el servidor **a la escucha**, en el puerto recibido como primer argumento e IP indicada por el segundo
- El objeto `console` representa la consola del sistema y el método `log()` facilita el envío de mensajes para mostrar en ella

```
 GNU nano 5.4                                    hola.js *
const http = require('http'); // Cargar el paquete http

const puerto = 1234;
const servidor = 'localhost';

function frespuesta(request, response) {
  response.writeHead(200, { 'Content-Type' : 'text/html' });
  response.end('<h1>Hola Node.js</h1>');
}

var aplicacion = http.createServer(frespuesta);
aplicacion.listen(puerto, servidor);

console.log(`Aplicación a la escucha en en puerto ${puerto}`);
```

El lenguaje JavaScript

JavaScript es un lenguaje con sintaxis similar a C++/Java:

- Se diferencia entre **mayúsculas y minúsculas** en los identificadores
- El final de cada sentencia se marca con el símbolo `;`
- Los bloques de código (p.e. cuerpo de una función) se delimitan entre llaves `{ ... }`
- La sintaxis `objeto.miembro` accede a variables de objetos e invoca métodos

Características avanzadas de JavaScript:

- Permite crear fácilmente colecciones de pares **clave-valor** con la sintaxis
  `{ clave1 : valor1, clave2 : valor2, ... }`
- Los **vectores** se delimitan entre corchetes con la sintaxis
  `[ elemento1, elemento2, ... ]`
- Es posible definir **funciones anónimas** como expresiones *lambda*:
  `(parámetros) => { cuerpo de la función }`
- Según el entorno de ejecución (Node.js o el navegador) JavaScript tendrá a su disposición unos objetos u otros:
  - Por ejemplo: en el navegador existe el objeto `window` pero en Node.js no

Tutorial

En https://www.w3schools.com/js tienes un tutorial de iniciación a JavaScript

El paquete `express`

Procesamiento de las solicitudes

- El objeto `http`, ya incluido en Node.js, permite procesar cualquier solicitud HTTP. Para ello es necesario **examinar el contenido** del parámetro `request`, a fin de identificar el verbo HTTP empleado, la ruta y sus parámetros
- En una aplicación real, con múltiples *endpoints*, esa tarea llega a ser tediosa
- El paquete `express` es el más usado para crear aplicaciones web con Node.js y nos ofrece:
  - Métodos específicos para procesar cada verbo de HTTP
  - Un procedimiento sencillo para asociar cada ruta con el código a ejecutar
  - Mecanismos de obtención de los parámetros asociados a la petición

Acción	Método de `express` a usar
Obtener el objeto que representa a la aplicación	`const app = express(); // Guardamos en variable app`
Responder a petición `GET`, `POST`, `PUT` o `DELETE`	`app.método(ruta, func); // get, post, put, delete`
Gestionar respuesta a una ruta	`app.route(ruta).método(func).método(func)...`
Poner la aplicación a la espera de peticiones	`app.listen(puerto, [host]);`
Cargar funciones intermediarias para procesar las peticiones	`express.use(función);`

Instalación de `express` y otros paquetes

Proceso de instalación

Tras la instalación de Node.js (véase en un paso anterior), en el sistema también disponemos de la utilidad `npm`, instalador de paquetes para Node.js.

Vamos a instalar los paquetes que necesitaremos con el comando:

```
npm install expres uuid db
```

Paquete db

Lo instalamos en previsión de que usásemos una base de datos externa para almacenar la información, pero en esta práctica **no lo necesitaremos** porque usaremos archivos JSON

Paquete	Utilidad
express	Gestión de peticiones HTTP
uuid	Generación identificadores únicos
db	Almacenamiento de datos en archivo (MySQL, MongoDB, FireBase, etc.)

```
usuario@par:~$
usuario@par:~$ npm install express uuid db
npm WARN deprecated har-validator@5.1.5: this library is no longer supported
npm WARN deprecated querystring@0.2.0: The querystring API is considered Legacy. new code should use
 the URLSearchParams API instead.
npm WARN deprecated uuid@3.3.2: Please upgrade  to version 7 or higher.  Older versions may use Math
.random() in certain circumstances, which is known to be problematic.  See https://v8.dev/blog/math-
random for details.
npm WARN deprecated uuid@3.4.0: Please upgrade  to version 7 or higher.  Older versions may use Math
.random() in certain circumstances, which is known to be problematic.  See https://v8.dev/blog/math-
random for details.
npm WARN deprecated request@2.88.2: request has been deprecated, see https://github.com/request/requ
est/issues/3142

added 80 packages, and audited 132 packages in 10s

5 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
usuario@par:~$
```

Primer ejercicio con Express

holaexpress.js

```
const express = require('express');   // Cargamos el paquete express
const app = express();  // Creamos el objeto que actuará como aplicación

app.get('/', (req, res) => {    // Respuesta a una solicitud GET con la ruta /
  res.send(`
  <ul>
    <li><a href="tareas/">Lista de tareas</a>
    <li><a href="tarea/1234">Tarea 1234</a>
  <ul>`);
});

app.get('/tareas', (req, res) => {  // Respuesta a una solicitud GET con la ruta /tareas
  res.send('Lista de tareas');
});

app.get('/tarea/:id', (req, res) => {  // Respuesta a una solicitud GET con la ruta /tarea/identificador
  const { id } = req.params; // Obtenemos el parámetro enviado en la solicitud
  res.send({id,});  // Enviamos no una cadena, sino un JSON con el identificador
});

app.listen(3000);   // Quedamos a la espera de peticiones
```

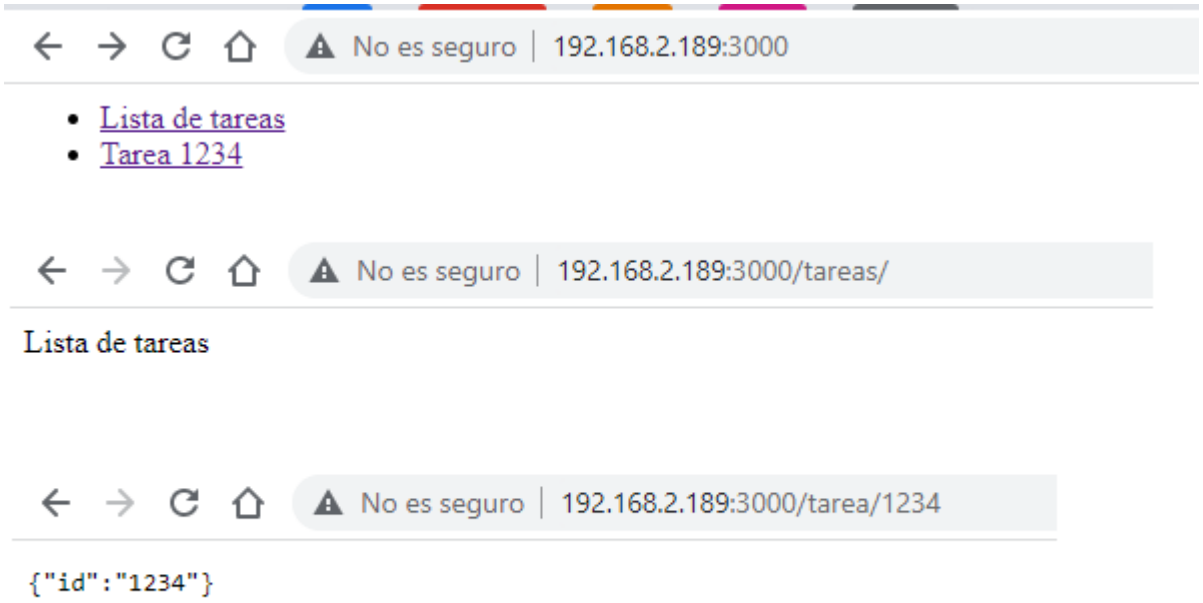
Primer ejercicio con Express

Acceso a la aplicación

Puedes usar el navegador Lynx desde la consola de texto o bien acceder **desde otra máquina** en la que tengas instalado un navegador estándar:

- Usa la orden `ip addr` para obtener la IP asignada a tu máquina virtual
- Abre el navegador en la otra máquina e introduce esa IP en la barra de direcciones

Según la ruta de entrada la respuesta obtenida será una u otra, según se aprecia en la imagen de la derecha



Respuesta a peticiones distintas de una misma ruta

expressCRUD.js

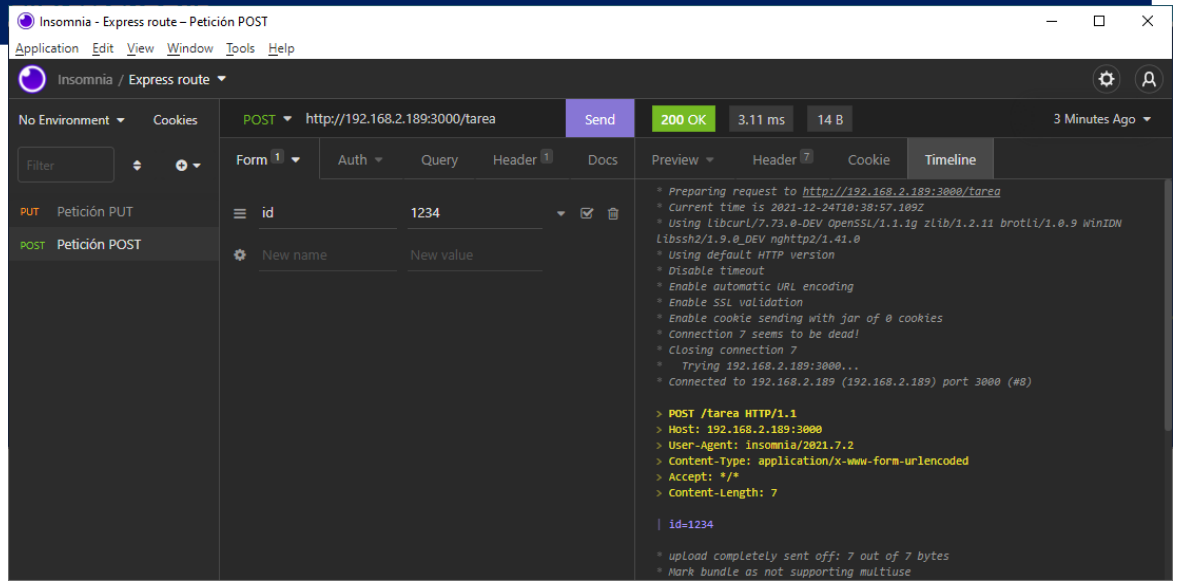
```
const express = require('express');
const app = express();

app.route('/tarea') // Ruta común a todas las peticiones
  .get((req, res) => { res.send("Petición GET"); } )      // Cada tipo de petición
  .post((req, res) => {  res.send("Petición POST"); } )   // tiene su respuesta
  .put((req, res) => {  res.send("Petición PUT"); } )     // a medida
  .delete((req, res) => { res.send("Petición DELETE"); } )

app.listen(3000);
```

Pruebas con Insomnia

- Desde el navegador web solo se pueden enviar peticiones `GET`
- Descarga e instala Insomnia para hacer otros tipos de peticiones.

Con esta herramienta podrás:

- Crear un proyecto de prueba y, dentro de él, definir múltiples peticiones
- Para cada petición establecer su tipo y parámetros
- Examinar la respuesta obtenida del *backend*, incluyendo las cabeceras de petición y de respuesta



Procesamiento de formularios

En Node.js

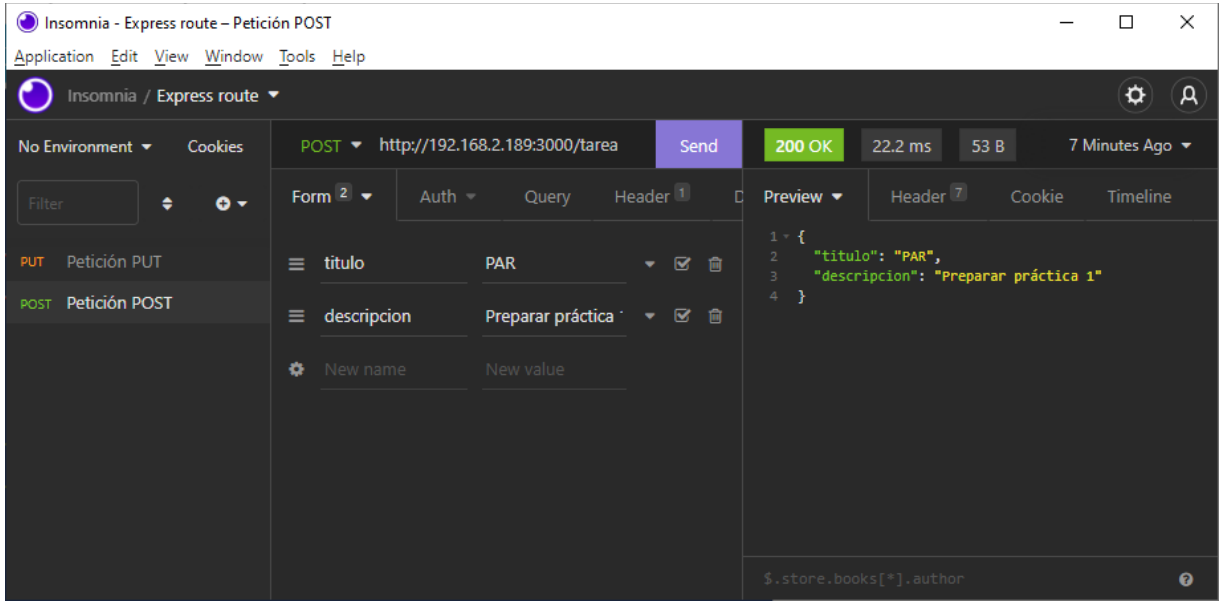
- Hay que activar la función `urlencoded()` para poder extraer los datos de un formulario enviado por HTTP
- Activada dicha función, basta con asignar el contenido del campo `body` de la petición a variables que tendrán el mismo nombre que los campos del formulario
- Los datos se usarían para realizar algún procesamiento en el *backend.* En el ejemplo inferior simplemente los devolvemos de nuevo al *frontend*

Cambios sobre expressCRUD.js

```
...
// Funcionalidad que hay que activar para tratar
// con datos de formularios y en formato JSON
app.use(express.json());
app.use(express.urlencoded({extended: true}));
...
 .post((req, res) => {
    // Extraemos los datos del formulario
    const { titulo, descripcion } = req.body;
    res.send({titulo, descripcion});
  })
...
```

En Insomnia

- Crea una petición de tipo `POST`
- Establece el URL al que debe enviarse la petición
- Detalla el nombre de los campos del formulario y su contenido
- Envía la petición al servidor Node.js para obtener la respuesta



Almacenamiento de datos

Lectura y escritura de archivos JSON

- Un archivo en formato JSON puede abrirse desde cualquier editor y tiene un formato fácilmente legible
- La función `require(ruta)` permite cargar el contenido de una archivo JSON y almacenarlo en una variable JavaScript
- Dicha variable será un vector de estructuras, usándose la notación `variable.colección[elemento].campo` para acceder a su contenido
- El almacenamiento de la variable, si sufre cambios, se completa con la función `fs.writeFile()`

expressDB.js

```
...
const fs = require('fs')  // Para poder usar writeFile()
const uuid = require('uuid'); // Para generar identificador
const db = require('./db.json'); // Cargar lista de tareas

app.post('/tarea', (req, res) => {
   const { titulo, descripcion } = req.body;
   const otratarea = { // Construcción de la nueva tarea
    id: uuid.v4(), titulo, descripcion, terminada: false,
   };
   db.tareas.push(otratarea); // Añadirla a la lista
   fs.writeFile('./db.json', JSON.stringify(db,null,2),() => {} );
   res.send(otratarea); // Devolverla al frontend
});
...
```

Archivo db.json

```
{
  "tareas": [
    {
      "id": "c621231b-b173-4d8d-a5c8-1f70d3f0c825",
      "titulo": "PAR",
      "descripcion": "Prepararar práctica 1",
      "terminada": true
    },
    {
      "id": "2f38a28f-1767-48e8-acb7-ca90dbc5fac2",
      "titulo": "Compra",
      "descripcion": "Macarrones y tomate",
      "terminada": false
    }
  ]
}
```

Uso interactivo de Node.js

La consola de Node

- Introduce en la línea de comandos la orden `node` y pulsa **Intro**

- Se abrirá la **consola interactiva** de Node.js, desde la que puedes realizar pruebas sin necesidad de crear archivos con todas las órdenes

- Usa la consola para **probar una funcionalidad** antes de usarla en un programa. En el ejemplo de la derecha se muestra cómo obtener un identificador único con `uuid.v4()`

- La consola también te servirá para **familiarizarte con la sintaxis** del lenguaje JavaScript. A la derecha, en la parte inferior, se muestra cómo acceder a un dato concreto de los cargados del archivo JSON

- Usa `^d` para **salir** de la consola

```
Welcome to Node.js v17.3.0.
Type ".help" for more information.
> const uuid = require('uuid')
undefined
> uuid.v4()
'756fc086-9767-407d-8dd9-4ae5a49bb8d9'
> datos = require('./db.json');
{
  tareas: [
    {
      id: 'c621231b-b173-4d8d-a5c8-1f70d3f0c825',
      titulo: 'PAR',
      descripcion: 'Prepararar práctica 1',
      terminada: true
    },
    {
      id: '2f38a28f-1767-48e8-acb7-ca90dbc5fac2',
      titulo: 'Compra',
      descripcion: 'Macarrones y tomate',
      terminada: false
    },
    {
      id: 'd1dda557-132c-4748-bd28-779fb2238796',
      titulo: 'Semana 2 PAR',
      descripcion: 'Estudiar ejercicios',
      terminada: false
    }
  ]
}
> datos.tareas[2].descripcion
'Estudiar ejercicios'
>
>
```

Para saber más …

- En https://www.w3schools.com/js hay un tutorial paso a paso de JavaScript con ejemplos que pueden ir modificándose y probándose directamente en el navegador o consola de Node.js.

- En http://expressjs.com/es/ se encuentra la documentación y guía de iniciación a Express, con ejemplos breves sobre cómo realizar cada tarea.

- En https://www.npmjs.com/ se ofrece un buscador de paquetes para Node.js. Localizado el paquete que interesa, se instalaría como hemos hecho en la diapositiva 10.

- A través de la **Biblioteca UJA** (tras el inicio de sesión) tienes acceso a libros como Web Development with Node and Express, en los que encontrarás información más detallada y ejercicios más extensos.