

PROYECTO 5:
SERVIDOR WEB
MEDIANTE EXPRESS
TIEMPO ESTIMADO: 60 MINUTOS

DESCRIPCIÓN

- *Montar un servidor web mediante paquetes propios de node*

OBJETIVOS

- Entender la arquitectura de Express
 - Middleware
 - Uso de paquetes para extender la funcionalidad de Express
- Creación de un servidor web que sirva:
 - Contenido estático
 - Contenido dinámico en base a templates
 - Content-type en JSON (para API posterior)
- Despliegue del servidor en un entorno serverless

EMPEZAMOS PROYECTO

DOCUMENTACIÓN

- <https://expressjs.com/>
 - Extensa pero consisa
 - Numerosos ejemplos

PRIMEROS PASOS

- No utilizaré ningún repo de GitHub.
- Creamos un directorio para la app y...
 - `npm init`
 - Instalar y configurar eslint (standard)

EXPRESS

- Instalar express mediante uno de los comandos siguientes:

```
npm install --save express@4.16.3  
npm i -S express@4.16.3
```

- Creamos el fichero server.js con el siguiente código:

```
const express = require('express')  
const app = express()  
app.get('/', (req, res) => {  
  res.send('Hola Mundo')  
})  
app.listen(3000)
```

EJECUCIÓN

- Debería ejecutarse de cualquiera de las siguientes maneras:

```
npm start  
node server.js
```

- Utiliza **nodemon** para evitar reinicios al cambiar código

PRUEBA FUNCIONAMIENTO

- Se echa de menos algún tipo de mensaje de arranque
- [Mira en la documentación](#) como añadir esa opción
 - Cuando el método listen haya terminado...

MÉTODO LISTEN

- Permite añadir más parámetros
- Un parámetro de tipo función (función de callback)
 - Es el único parámetro de este tipo, así los diferencia

```
app.listen([port[, host[, backlog]]][, callback])
```

```
app.listen(3000, () => {  
  console.log('Servidor web arrancado en el puerto 3000')  
})
```

AÑADIR OTRA RUTA

- Utiliza el plugin *ExpressSnippet* de Visual Code para completado
 - Comprueba su funcionamiento con el *app.listen*
- Datos nueva ruta:
 - URL: */contactar*
 - Muestre el mensaje *Página para contactar*

CÓDIGO RUTA CONTACTAR

```
app.get('/contactar', (req, res) => {  
  res.send('Página para contactar')  
})
```

USO DE JSON

- Vamos a devolver un JSON en vez de un string:

```
app.get('/contactar', (req, res) => {  
  res.send({  
    nombre: 'pepito'  
    email: 'pepito@gmail.com'  
  })  
})
```

COMPROBAR JSON

- Utiliza algún plugin de formateo de JSON dentro del navegador
 - El JSON puede ser más complejo que el anterior, por ej:

```
https://api.arasaac.org/api/pictograms/es/search/casa
```

- Comprueba como cambia el content type
 - Utiliza las herramientas de desarrollo
 - El cambio lo hace directamente Express

localhost:3000/contactar

Juan Daniel

localhost:3000/contactar

Aplicaciones Libros seo Programación Proyectos Servidores Raspberry Bookmarks Bolsa Otros marcadores

```
{
  "nombre": "pepito",
  "email": "pepito@gmail.com"
}
```

RawParsed

Elements Console Sources Network Performance Memory Application Security Audits Perf Redux AngularJS

View: Group by frame Preserve log Disable cache Offline Online

Filter Hide data URLs All XHR JS CSS Img Media Font Doc WS Manifest Other

20 ms 40 ms 60 ms 80 ms 100 ms 120 ms 140 ms 160 ms 180 ms 200 ms 220 ms 240 ms 260 ms 280 ms 300 ms 320 ms

Name

contactar

page.bundle.js

data:image/png;base...

Headers

Preview

Response

Cookies

Timing

Referrer Policy: no-referrer-when-downgrade

Response Headers

view source

Connection: keep-alive

Content-Length: 46

Content-Type: application/json; charset=utf-8

Date: Wed, 05 Sep 2018 08:25:28 GMT

ETag: W/"2e-5JzTjdpQHynnLkL5exRKCzUivEI"

X-Powered-By: Express

Request Headers

view source

3 requests | 258 B transferred | Finish...

- Los datos en JSON podrían estar:
 - En una variable
 - Incluso en otro fichero
 - Recibirse vía API
 -

```
const contacto = require('./contacto.json')
app.get('/contactar', (req, res) => {
  res.send(contacto)
})
```

MIDDLEWARE

- Son funciones que:
 - Se registran por express mediante *app.use* y se ejecutan en orden
 - Tienen acceso al objeto de solicitud (*req*), al objeto de respuesta (*res*) y a la siguiente función de middleware (*next*)

```
app.use((req, res, next)=>{  
  //operaciones del middleware  
  next() //para ir al siguiente middleware o a la ruta  
  // también podríamos hacer un send() y cortar  
  // la cola de middlewares, por ej en un control de permisos  
})
```

EJEMPLO DE MIDDLEWARE

- Podemos crear un middleware que guarde traza de las fechas de accesos
- Para ver las propiedades que nos hacen falta podemos usar la [API de Express](#)
- Es importante el orden
 - Entre los middlewares
 - Antes que las peticiones get, post....

```
var app = express()

app.use(function (req, res, next) {
  var now = new Date().toString()
  console.log(`Time: ${now} ${req.method} ${req.url}`)
  next()
})
```

LOGS A FICHERO

- Completa el middleware anterior para que guarde los cambios en el fichero *server.log*
- Piensa donde se debe poner el `next()` y si debes utilizar un método síncrono o asíncrono

SOLUCIÓN LOGS A FICHERO

```
app.use((req, res, next) => {  
  var now = new Date().toString()  
  var log = `${now}: ${req.method} ${req.url}`  
  console.log(log)  
  fs.appendFile('server.log', `${log}\n`, (err) => {  
    if (err) console.log(`No se ha podido usar el fichero de l  
  })  
  next()  
})
```

MÁS SOBRE LOGS

- Podemos querer utilizar distintos *transports* o medios para logs
 - Ficheros
 - Consola
- Distintos niveles (debug, err, warning...)
- Distintos formatos de visualización (colores, negrita...)
- Con posibilidad de ejecución de queries
-
- Lo mejor es **usar algún paquete que ya exista**

USO DE CONTENIDO ESTÁTICO

- Crea un fichero .html en la carpeta public
 - Ayúdate de emmet: *! + tab*
- Express ya tiene un middleware integrado para contenido estático
 - No deja de ser una función como las vistas anteriormente
 - No necesitamos importarla mediante un require
 - Una vez importado, hace un **send()** si existe el fichero, si no, un **next()**

CONFIGURACIÓN EXPRESS PARA CONTENIDO ESTÁTICO

```
const staticRoute = path.join(__dirname, 'public')
app.use(express.static(staticRoute))
```

- *__dirname* es la raíz del proyecto
- *path.join* para que sea multiplataforma
- Podríamos configurar un directorio virtual (static para public)

```
const staticRoute = path.join(__dirname, 'public')
app.use('/static', express.static(staticRoute))
```

TEMPLATE ENGINE

CONFIGURACIÓN DE HBS

- Instalación

```
npm i -S hbs
```

- Configuración

```
// const hbs = require('hbs')  
app.set('view engine', 'hbs'); // clave valor
```

- No hacemos un require porque no usamos ninguna función
- express lo llama internamente

- Indicamos a nodemon los tipos de ficheros a monitorizar (por defecto solo js):

```
"scripts": {  
  "start": "nodemon server.js -e js,hbs"  
},
```

USO DE HBS

- Definimos una carpeta views donde irán las templates
- Fichero *views/contactar.hbs*:

```
...  
<body>  
  <h1>{{pageTitle}}</h1>  
  <p>Aquí iría el formulario de contacto</p>  
  
  <footer>  
    <p>Copyright {{currentYear}}</p>  
  </footer>  
</body>  
....
```

- Ejecutamos el método *res.render()* en vez de *res.send()*
 - Admite un objeto como segundo parámetro para pasar variables

```
app.get('/contactar', (req, res) => {  
  res.render('contactar.hbs', {  
    pageTitle: 'Contactar',  
    currentYear: new Date().getFullYear()  
  })  
})
```

PARTIALS MEDIANTE HANDLEBARS

- Registramos el directorio donde se van a guardar:

```
const hbs = require('hbs')  
hbs.registerPartials(path.join(__dirname, 'views', 'partials'))  
app.set('view engine', 'hbs') // clave valor
```

- Creamos fichero *views/partials/footer.hbs*:

```
<footer>  
  <p>Copyright {{getCurrentYear}}</p>  
</footer>
```

- Lo incluimos dentro de nuestro fichero *views/contactar.hbs*:

```
...  
<body>  
  <h1>{{pageTitle}}</h1>  
  <p>Aquí iría el formulario de contacto</p>  
  {{> footer}}  
  <!-- con la linea anterior  
        tendríamos el footer mediante partials -->  
  
</body>  
...
```


EJERCICIO TEMPLATES

- Añade una página de inicio además de contactar
- Ambas deben cargar su correspondiente template que además cargará un partial para el header y otro para el footer

USO DE HELPERS

- Se registran funciones que devuelven un código dinámico
- Se pueden inyectar en cualquier template o partial.

```
hbs.registerHelper('getCurrentYear', () => new Date().getFullYear()  
// con paso de parámetro:  
hbs.registerHelper('toUpperCase', text => text.toUpperCase())
```

- Podríamos eliminar el paso de *currentYear* a las vistas y utilizar el helper:

```
<footer>  
  <p>Copyright {{getCurrentYear}}</p>  
  <p>{{toUpperCase "Licencia MIT"}}</p>  
  {{>header}}  
</footer>
```

DEPLOY

- Soluciones tradicionales como máquinas con Plesk, cPanel o similar no son aptas para node.js
 - Dan solución a CMS's, código en php y mySQL
- JavaScript es el estándar de facto para soluciones serverless
 - Google Cloud Functions por ejemplo solo funciona con JavaScript
 - Nosotros haremos el deploy usando Zeit

¿CONTINUAMOS?

- Ya estamos preparados para utilizar express para crear una API
 - Creación de una API