

Desarrollo Backend

Bienvenid@s

Servidor Web con Express II
(*rutas - errores - códigos de estado*)

Clase 12



Pon a grabar la clase



Temario

- Automatizar el servidor web
 - definir comandos en package.json
 - utilizar node monitor
 - utilizar --watch
- Definición de múltiples rutas con Express
- El manejo de errores con Express
- Códigos de estado
 - 1xx - 2xx - 3xx - 4xx - 5xx



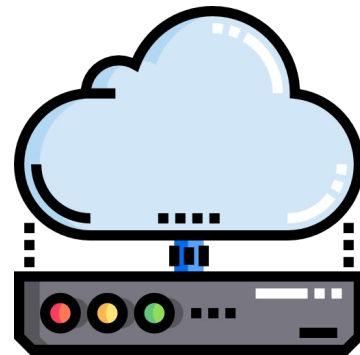
Automatizar el servidor web

Automatizar el servidor web

Hasta el momento, venimos ejecutando nuestros proyectos basados en servidores web, utilizando la Terminal mediante los comandos de Node.

Llegó el momento de liberarnos de este proceso repetitivo, automatizando algunos pasos de la mano de las diferentes herramientas que Node nos provee.

Veamos a continuación, tres maneras de automatizar.



con NPM y scripting

UNTREF

UNIVERSIDAD NACIONAL
DE TRES DE FEBRERO

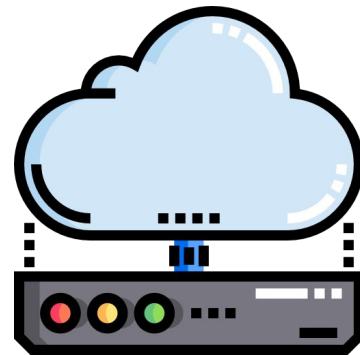
con NPM y scripting

Scripting en Package JSON

Tenemos la posibilidad de modificar un script generado por defecto en el archivo **package.json**. En el apartado “**scripts**”, encontraremos un parámetro similar al siguiente:

```
"scripts": {  
  "test": "echo \"Error: no test specified\" && exit 1"  
},
```

Este parámetro predeterminado puede ser modificado por el comando que escribimos de forma constante en la Terminal.



con NPM y scripting

Definimos un parámetro llamado **start**, y su valor será **node server.js**.
Luego, al momento de ejecutar el proyecto en la Terminal, tipeamos **npm start**, y se iniciará el proyecto.

A dark-themed terminal window with three window control buttons in the top-left corner. The title bar on the right says "package.json". The content is a JSON object for the "scripts" field.

```
package.json

"scripts": {
  "start": "node server.js",
  "end": "killall -9 node"
},
```


con NPM y scripting

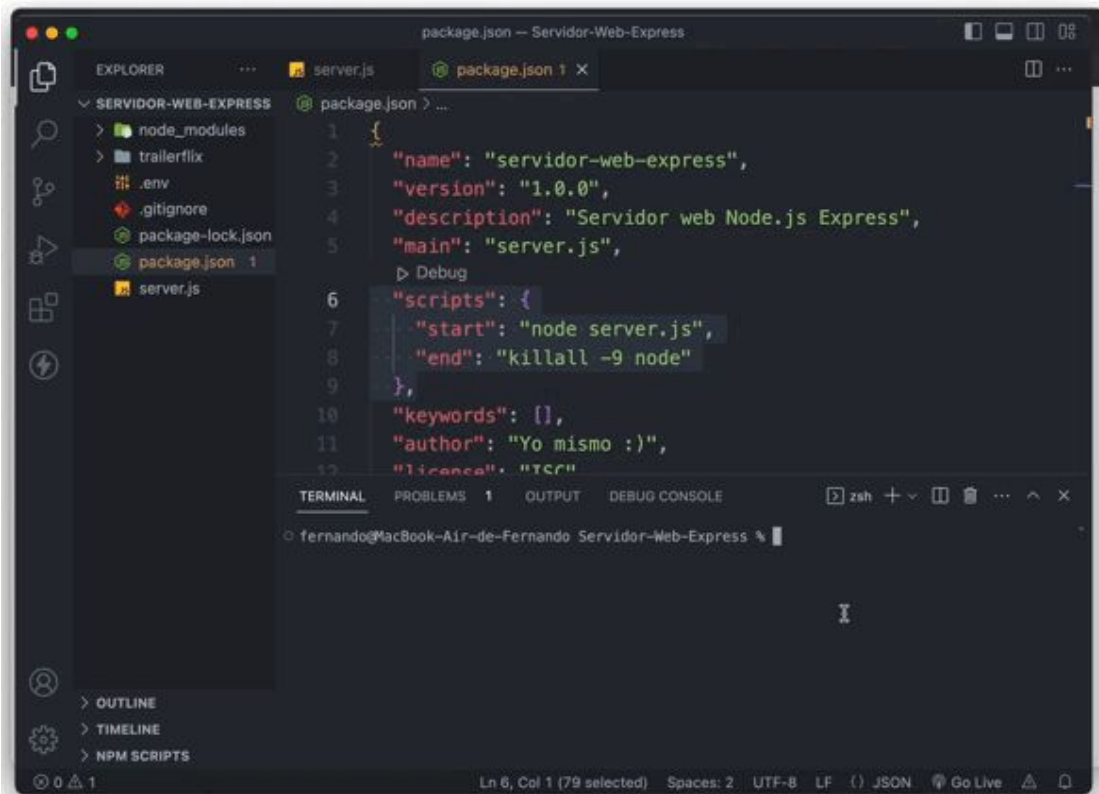
A continuación podemos definir otro parámetro llamado **end**, y que su valor sea **killall -9 node**. De esta manera, en los sistemas Unix, Linux, Mac, si se cuelga el proceso del servidor web, podremos interrumpirlo fácilmente.



```
package.json

"scripts": {
  "start": "node server.js",
  "end": "killall -9 node"
},
```

con NPM y scripting



The screenshot shows a VS Code editor window with a file explorer on the left and a code editor in the center. The file explorer shows a project named 'SERVIDOR-WEB-EXPRESS' with files like 'node_modules', 'trailerfix', '.env', '.gitignore', 'package-lock.json', 'package.json', and 'server.js'. The code editor shows the 'package.json' file with the following content:

```
{
  "name": "servidor-web-express",
  "version": "1.0.0",
  "description": "Servidor web Node.js Express",
  "main": "server.js",
  "scripts": {
    "start": "node server.js",
    "end": "killall -9 node"
  },
  "keywords": [],
  "author": "Yo mismo :)",
  "license": "ISC"
}
```

The 'scripts' section is highlighted, showing the 'start' and 'end' scripts. The terminal at the bottom shows the command prompt 'fernando@MacBook-Air-de-Fernando: Servidor-Web-Express %'.

El comando está listo para utilizarse en la Terminal.

Si bien podemos acortar más su nombre, debemos tener en cuenta siempre que el comando debe ser descriptivo.

con Nodemon

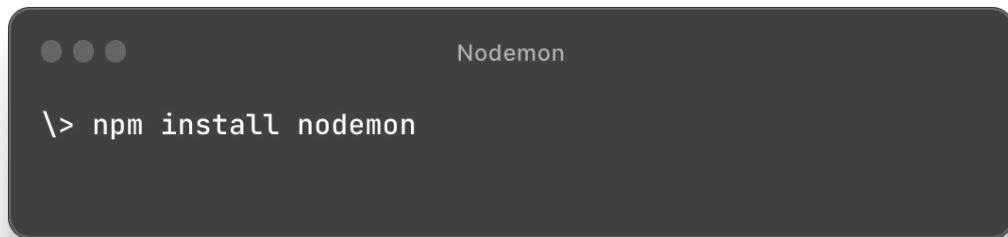
UNTREF

UNIVERSIDAD NACIONAL
DE TRES DE FEBRERO

con Nodemon

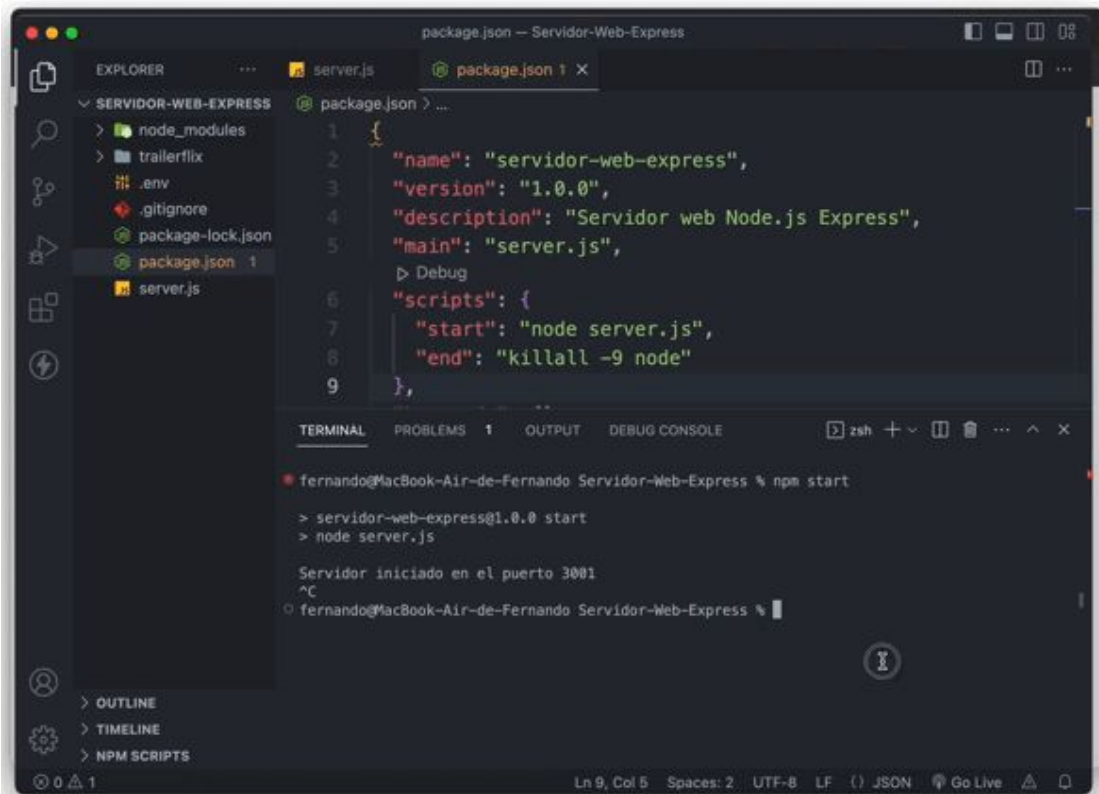
Uso de Nodemon

La otra alternativa para automatizar la ejecución de nuestra aplicación web, es instalando la dependencia nodemon. En principio, debemos instalar la dependencia desde la **Terminal**.

A dark-themed terminal window with a title bar containing three dots and the text "Nodemon". The terminal shows a command prompt "\>" followed by the command "npm install nodemon".

```
Nodemon  
  
\> npm install nodemon
```

con Nodemon



The screenshot shows a Visual Studio Code editor window with a dark theme. The Explorer sidebar on the left shows a project named 'SERVIDOR-WEB-EXPRESS' with files like 'node_modules', 'trailerflix', '.env', '.gitignore', 'package-lock.json', 'package.json', and 'server.js'. The 'package.json' file is open in the editor, showing the following content:

```
1 {
2   "name": "servidor-web-express",
3   "version": "1.0.0",
4   "description": "Servidor web Node.js Express",
5   "main": "server.js",
6   "scripts": {
7     "start": "node server.js",
8     "end": "killall -9 node"
9   },
10 }
```

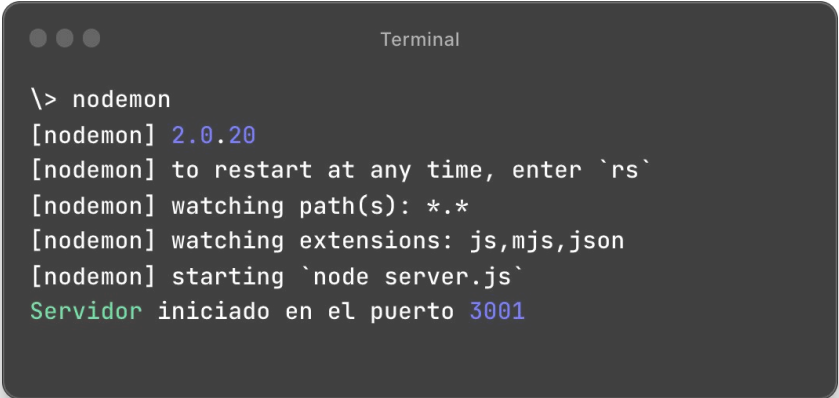
Below the editor, the TERMINAL panel is active, showing the command prompt for 'fernando@MacBook-Air-de-Fernando Servidor-Web-Express'. The user has entered 'npm start', which has executed the 'start' script defined in package.json, running 'node server.js'. The output shows 'Servidor iniciado en el puerto 3001' and a prompt to press ^C to stop the server. The user has then entered a new command in the terminal.

En la **Terminal** debemos escribir simplemente **nodemon**, y la dependencia se ocupará de ejecutar el **script** definido y, por cada cambio que realicemos en nuestro código, lo detiene y ejecuta de nuevo automáticamente.

con Nodemon

De esta forma, el proyecto está siempre ejecutándose, evitando que debamos reiniciarlo manualmente por cada cambio en el código que realicemos. Incluso, podemos reiniciarlo rápidamente, tipeando en la **Terminal**, el comando **rs**, seguido de **Enter**.

Y, para detenerlo, pulsamos la combinación **Ctrl + c** en la **Terminal**.


A terminal window with a dark background and light text. The title bar says "Terminal". The output shows the command "nodemon" being run, followed by version information, instructions to restart with "rs", watched paths and extensions, and a confirmation that the server is running on port 3001.

```
\> nodemon
[nodemon] 2.0.20
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node server.js`
Servidor iniciado en el puerto 3001
```

con NPM automatizado

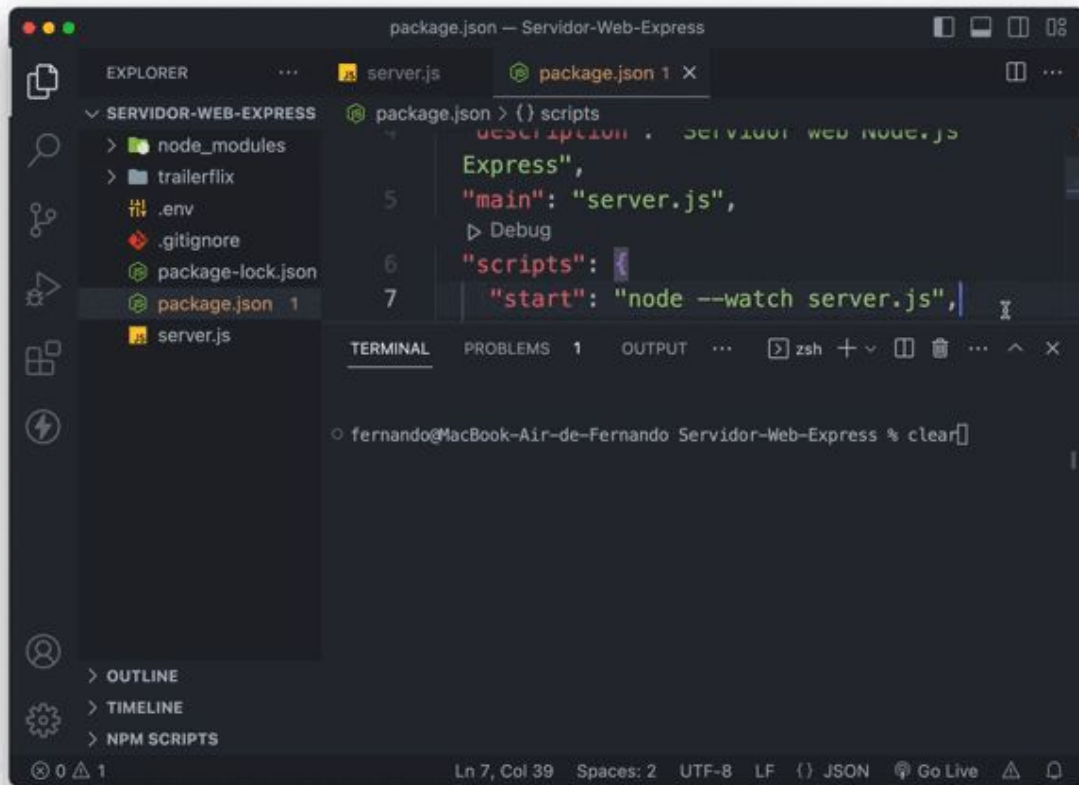
NPM automatizado

Hacia fines de 2022, Node incluyó un parámetro adicional en su línea de comandos, denominado **--watch**. Este permite ejecutar una aplicación Node.js en la **Terminal** con este parámetro adicional, y así que la aplicación en ejecución se detenga y vuelva a ejecutar cuando se detecta un cambio en el código.



```
"scripts": {  
  "start": "node --watch server.js",  
  "end": "killall -9 node"  
},
```


NPM automatizado



De esta forma, seguimos utilizando el clásico comando **npm start**, aunque ahora, con cada cambio en el código, se reiniciará el proyecto Node.js automáticamente.

Lo mismo que sucede con **Nodemon**, pero sin necesitar dicha dependencia.

Automatizar el servidor web

Como podemos apreciar, contamos con un sinfín de herramientas que nos permitirán automatizar pequeñas cuestiones durante la etapa de desarrollo de software. Cada una de ellas aporta sus pros y contras, y seremos nosotros quien elijamos cuál poner en uso en nuestro día a día como developers.

Manejo de rutas con Express JS

Manejo de rutas con Express JS

```
Rutas con Express JS

const server = http.createServer((req, res) => {
  if (req.url === '/') {
    // devolvemos datos sobre la ruta raíz o principal
  } else if (req.url === '/nosotros') {
    // devolvemos datos sobre 'Nuestra empresa'
  } else if (req.url === '/cursos') {
    // devolvemos datos sobre los cursos que ofrecemos
  }
  } else if (req.url === '/bootcamps') {
    // devolvemos datos sobre los bootcamps de código
  }
  } else if (req.url === '/contacto') {
    // devolvemos datos sobre cómo contactarnos
  } else {
    //si no hubo coincidencia...
    //¿manejo el error?
  }
});
```

El manejo de rutas que podríamos realizar con el módulo HTTP, seguramente lo encontraremos como algo complejo de poder controlar las posibles variantes que pueden darse con las rutas de un servidor web.

Manejo de rutas con Express JS



Por suerte, en Express JS, las rutas se definen como la forma en la cual se manejan las solicitudes HTTP. Cada solicitud a un servidor tiene una ruta asociada que indica qué acción tomará el servidor en respuesta a esa solicitud.

Manejo de rutas con Express JS

Para que podamos definir rutas en Express JS, debemos utilizar el objeto "*Router*" proporcionado por la librería.

Si miramos el código de ejemplo de la clase anterior, veremos que el control de peticiones que maneja **Express JS**, además de la función de retorno con **request** y **response**, antepone un parámetro el cual evalúa una ruta.



UNTREF

UNIVERSIDAD NACIONAL
DE TRES DE FEBRERO

Manejo de rutas con Express JS

Al utilizar el parámetro “/”, estamos definiendo que esta petición escuchará la ruta principal, o raíz, de nuestro servidor web. Ante la primera petición a dicha ruta, responderá con un mensaje de texto, tal como vemos en el código de ejemplo.



Rutas con Express JS

```
// Define una ruta básica
app.get('/', (req, res) => {
  res.send('¡Hola, mundo. Hola, Node.js!');
});
```

UNTREF

UNIVERSIDAD NACIONAL
DE TRES DE FEBRERO

Manejo de rutas con Express JS

En todo servidor web, la primera ruta siempre es definida con esta notación y corresponde a la página principal que responde a la **solicitud HTTP GET** en la raíz del servidor (*por ejemplo*: `http://localhost:3001/`).



Rutas con Express JS

```
// Define una ruta básica
app.get('/', (req, res) => {
  res.send('¡Hola, mundo. Hola, Node.js!');
});
```

UNTREF

UNIVERSIDAD NACIONAL
DE TRES DE FEBRERO

Manejo de rutas con Express JS

Por lo tanto, esta estructura de código simple para el manejo de rutas, deberá replicarse por cada una de las rutas que deseemos incluir en nuestro servidor web Express JS. Definimos entonces cuáles serán las rutas válidas que devolverán una respuesta y luego replicamos este código, una vez por cada una de estas rutas.

```

Rutas con Express JS

// Define una ruta básica
app.get('/', (req, res) => {
  res.send('¡Hola, mundo. Hola, Node.js!');
});
```

Manejo de rutas con Express JS



Rutas con Express JS

```
// La ruta raíz
app.get('/', (req, res) => {
  res.send('¡Hola, mundo. Hola, Node.js!');
});

app.get('/nosotros', (req, res) => {
  res.send('Aquí tienes información sobre nuestra empresa.');
```

```
});

app.get('/cursos', (req, res) => {
  res.send('Este es el listado de cursos que brindamos.');
```

```
});
```

Aquí tenemos un ejemplo similar al elaborado con el módulo HTTP, pero traducido íntegramente al framework Express JS.

Cada una de las rutas definidas, devolverá un mensaje de respuesta, de acuerdo a lo que dice el ejemplo de código.

UNTREF

UNIVERSIDAD NACIONAL
DE TRES DE FEBRERO

Manejo de rutas con Express JS

```

Rutas con Express JS

app.get('/bootcamps', (req, res) => {
  res.send('¿Te gusta la adrenalina? Puedes sumarte a
  nuestros bootcamps intensivos.');
```

```

});

app.get('/contacto', (req, res) => {
  res.send('¿Dudas, Consultas? No dejes de escribirnos a
  info@fakeEmail.dont');
```

```

});

// Inicia el servidor
app.listen(PORT, () => {
  console.log(`Servidor iniciado en el puerto ${PORT}`);
});
```

Por lo tanto, una vez planificada cada una de las rutas que contendrá nuestro servidor web, resolvemos copiando la estructura modelo, y agregándole el parámetro de **'/ruta'** que deseamos que se escuche a través de la petición HTTP GET.

Errores en rutas

Errores en rutas

En Express JS, es posible manejar solicitudes que no coinciden con ninguna de las rutas definidas. Esto se hace utilizando un middleware que se ejecuta después de todas las rutas definidas.

Para manejar rutas inexistentes, podemos agregar una ruta predeterminada (**catch-all**) utilizando **app.use()** y definir el controlador para esta ruta.



Errores en rutas

El método **.use()** es el que utilizamos para definir un Middleware que se ocupe de realizar tareas o procesos intermedios, por fuera de la lógica de nuestra aplicación.

Agregaremos entonces este método, contenido en el objeto **app**, para controlar toda aquella ruta que sea peticionada y que no exista en nuestra aplicación.



Errores en rutas

La estructura de este Middleware recibe como parámetro la misma función creada para el resto de las rutas definidas en nuestro servidor web, pero sin esperar un **path** específico.

El controlador para esta ruta devuelve una respuesta con un código de estado **404 - (recurso no encontrado)** y un mensaje para informar al usuario que el recurso solicitado no existe.

```

Rutas con Express JS

// Ruta predeterminada para manejar rutas inexistentes
app.use((req, res) => {
  res.status(404).send('Lo siento, la página que buscas no
  existe.');
```

Errores en rutas

También es muy común utilizar el método `.get()` seguido del path con el caracter comodín `"*"` para interceptar rutas inexistentes. La respuesta hacia al cliente será la misma que en el ejemplo de uso del Middleware y la ubicación en la estructura del código de este ejemplo, también debe respetarse.

```

Rutas con Express JS

// Ruta predeterminada para manejar rutas inexistentes
app.get('*', (req, res) => {
  res.status(404).send('Lo siento, la página que buscas no
  existe.');
```


Errores en rutas

Es importante destacar que, para definir correctamente el Middleware o el método get(“*”) que interceptan las peticiones a rutas o recursos inexistentes, sean alojados luego de todas las rutas definidas, para no interferir con las solicitudes que sí coinciden con rutas definidas.

Estados de peticiones

Estados de peticiones

Hasta ahora hemos aprendido que, ante determinadas peticiones, solemos retornar en algunos casos, un código de Estado, de acuerdo al tipo de petición.

Como vimos en el último ejemplo, ante la petición de un recurso inexistente, devolvimos un mensaje al usuario junto al código de Estado 404.

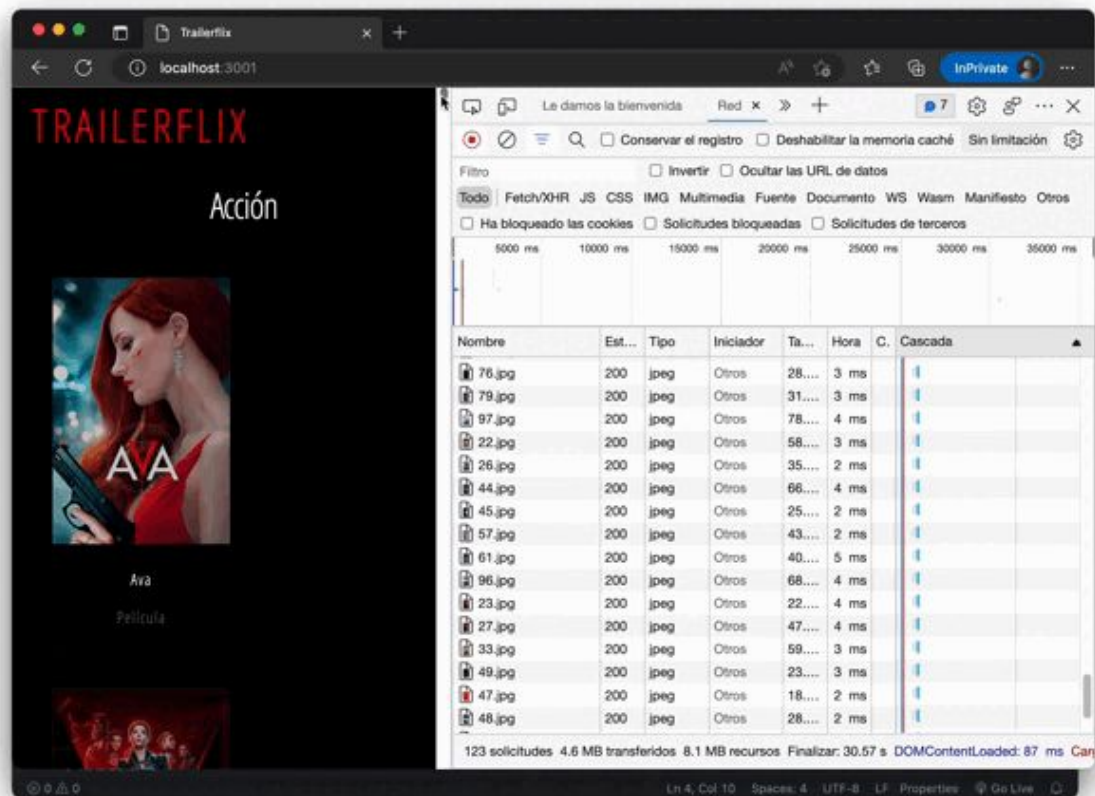


Estados de peticiones

Cuando Express JS responde a una solicitud de una ruta que sí existe no es necesario enviar un código de estado explícitamente, ya que este Framework asume internamente que la respuesta es exitosa y, junto a esta, envía un código de estado **200 - (OK)** de forma predeterminada.



Estados de peticiones



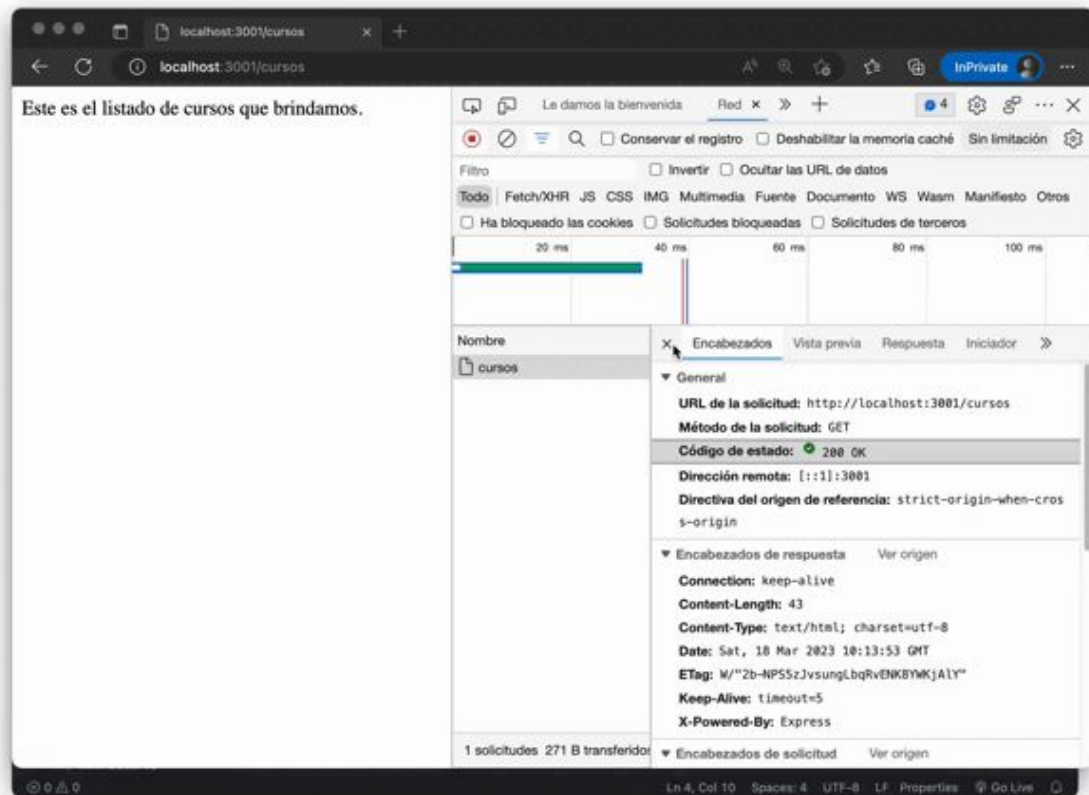
Aquí tenemos un ejemplo de acceso a una ruta existente.

El mismo framework se ocupa de responder con el mensaje correcto a dicha ruta, y si utilizamos **DevTools > Network** para analizar la respuesta, vemos efectivamente un **status 200 - OK**.

UNTREF

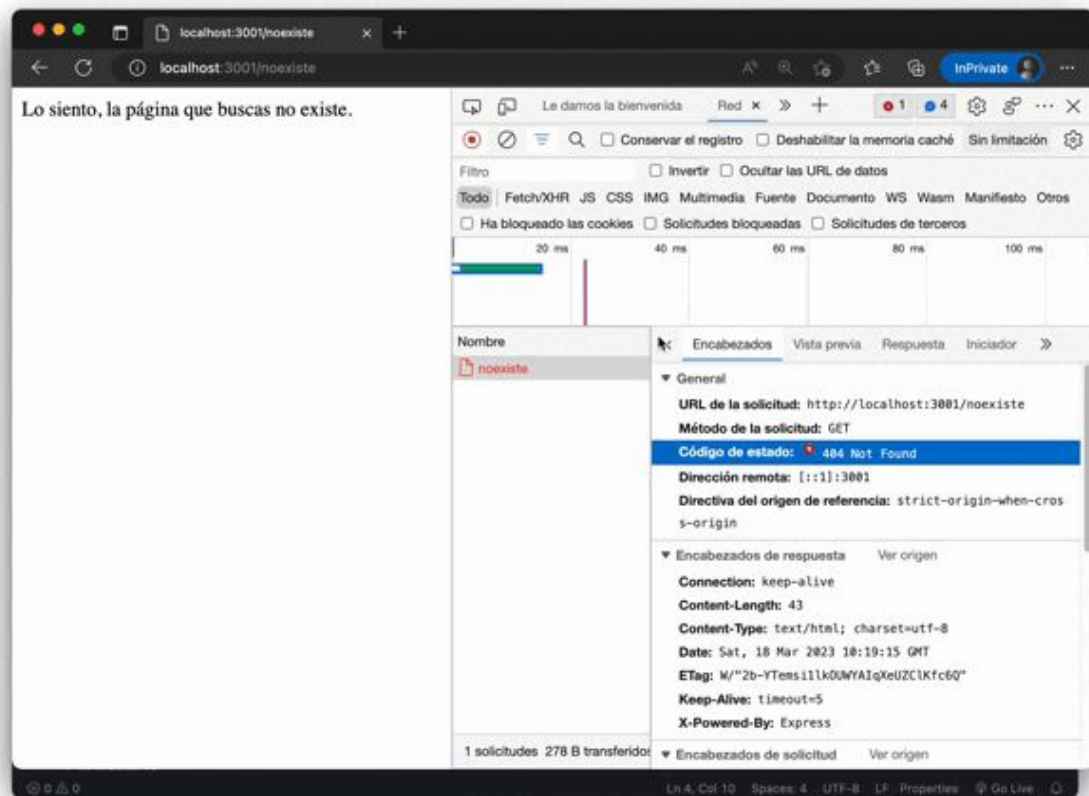
UNIVERSIDAD NACIONAL
DE TRES DE FEBRERO

Estados de peticiones



Luego, al peticionar a una ruta inexistente, el Middleware se ocupa de responder con el **status 404 - recurso inexistente**.

Estados de peticiones



El Framework también controla el estado de peticiones de un mismo cliente.

Cuando éste peticiona algo que ya había solicitado antes, y el recurso no cambió desde su última petición, el **status** de la misma será **304**, en lugar de **200**.

De esta forma, el cliente recuperará el recurso desde el caché en lugar de volver a descargarlo del servidor.

Códigos de estado

Códigos de estado

Cada petición y/o respuesta y/o rechazo, posee un código de estado y una descripción para el mismo. A su vez, se subdividen en categorías, para entender más rápido de dónde proviene el error.

Veamos a continuación información de cada categoría, sus códigos de error, y una descripción de los mismos.



Respuestas informativas - 1xx

Los **códigos de estado 1xx** en HTTP **proporcionan información adicional al cliente mientras la solicitud aún está en proceso** de ser procesada por el servidor.

Estas respuestas informativas pueden incluir detalles sobre el estado de la solicitud, la conexión o el protocolo.



Respuestas informativas - 1xx

Estas respuestas informativas pueden incluir detalles sobre el estado de la solicitud, la conexión o el protocolo.

Código	Descripción
100	Continuar: El servidor ha recibido los encabezados de la solicitud y el cliente debe continuar con la solicitud.
101	Cambio de protocolo: El servidor acepta cambiar el protocolo solicitado por el cliente.
102	Procesando: El servidor está procesando la solicitud y aún no se ha completado.

Respuestas informativas - 1xx

En general, estos códigos de estado no se utilizan con frecuencia, ya que la mayoría de las solicitudes se procesan rápidamente y no necesitan información adicional del servidor mientras se están procesando.

Sin embargo, en algunos casos, como con grandes cargas de datos, puede ser útil proporcionar información adicional sobre el estado de la solicitud para que el cliente pueda ajustar su comportamiento en consecuencia.



Respuestas informativas - 2xx

Los **códigos de estado 2xx** en HTTP indican que la solicitud del cliente fue procesada con éxito por el servidor.

A diferencia de los códigos de estado 1xx, que proporcionan información adicional mientras se procesa la solicitud, los códigos 2xx indican que la solicitud se completó correctamente.



Respuestas informativas - 2xx

Código	Descripción
200	OK: La solicitud ha sido procesada correctamente y el servidor devuelve una respuesta.
201	Creado: La solicitud ha sido procesada y el servidor ha creado un nuevo recurso.
202	Aceptado: La solicitud ha sido aceptada pero aún no ha sido procesada.
204	Sin contenido: La solicitud ha sido procesada correctamente, pero no hay contenido para devolver.

Respuestas informativas - 2xx

En general, los códigos de estado 2xx indican que la solicitud del cliente fue procesada con éxito y se devolvió una respuesta adecuada.

En el caso de los códigos 201 y 202, el servidor también puede proporcionar información adicional sobre el estado de la solicitud o el nuevo recurso creado.



Respuestas informativas - 3xx

Los **códigos de estado 3xx** en HTTP indican que la **solicitud del cliente se ha redirigido** a otro recurso.

Esto significa que el recurso solicitado no se encuentra en la ubicación esperada y se debe buscar en otra ubicación.



Respuestas informativas - 3xx

Código	Descripción
300	Múltiples opciones: La solicitud ha sido procesada correctamente y el servidor devuelve una respuesta.
301	Movido permanentemente: La solicitud ha sido procesada y el servidor ha creado un nuevo recurso.
302	Encontrado: La solicitud ha sido aceptada pero aún no ha sido procesada.
304	No modificado: La solicitud ha sido procesada correctamente, pero no hay contenido para devolver.
307	Redirección temporal: La solicitud se ha redirigido temporalmente a otra ubicación.

Respuestas informativas - 3xx

En general, los códigos de estado 3xx indican que la solicitud del cliente debe ser redirigida a otra ubicación para obtener el recurso solicitado.

En algunos casos, como con el código 304, la respuesta también puede indicar que el recurso solicitado no ha cambiado desde la última vez que se accedió a él, por lo que se puede utilizar una copia en caché en lugar de hacer una nueva solicitud.



Respuestas informativas - 4xx

Los **códigos de estado 4xx** en HTTP indican **que la solicitud del cliente no pudo ser procesada** debido a un error en la solicitud o en el cliente.



Respuestas informativas - 4xx

Código	Descripción
400	Solicitud incorrecta: La solicitud del cliente es incorrecta o no se puede procesar.
401	No autorizado: El cliente no está autorizado para acceder al recurso solicitado.
403	Prohibido: El servidor se niega a procesar la solicitud del cliente por motivos legales o de autorización.
404	No encontrado: El recurso solicitado no se encuentra en el servidor.
405	Método no permitido: El método de solicitud utilizado no está permitido para el recurso solicitado.

Respuestas informativas - 4xx

En general, los códigos de estado 4xx indican que se produjo un error en la solicitud del cliente, ya sea porque la solicitud era incorrecta, porque el cliente no tenía permiso para acceder al recurso o porque el recurso solicitado no se encontró en el servidor.

En algunos casos, como con el código 405, el servidor también puede proporcionar información adicional sobre los métodos de solicitud permitidos para el recurso solicitado.



Respuestas informativas - 5xx

Los **códigos de estado 5xx** en HTTP indican **que se produjo un error en el servidor** al procesar la solicitud del cliente.

Estos códigos indican que el servidor no pudo cumplir con la solicitud del cliente debido a un error interno en el servidor.



Respuestas informativas - 5xx

Código	Descripción
500	Error interno del servidor: El servidor encontró un error interno que le impidió procesar la solicitud del cliente.
501	No implementado: El servidor no es capaz de procesar la solicitud del cliente debido a que el método de solicitud utilizado no está implementado.
502	Puerta de enlace incorrecta: El servidor actuó como una puerta de enlace o proxy y recibió una respuesta no válida del servidor ascendente.
503	Servicio no disponible: El servidor no puede procesar la solicitud del cliente porque está sobrecargado o en mantenimiento.
504	Tiempo de espera agotado: El servidor actuó como una puerta de enlace o proxy y no recibió una respuesta oportuna del servidor ascendente.

Respuestas informativas - 5xx

En general, los códigos de estado 5xx indican que se produjo un error interno en el servidor al procesar la solicitud del cliente. Estos códigos pueden deberse a problemas de capacidad, problemas de conectividad de red, errores en el código del servidor, entre otros.

En algunos casos, como con el código 503, el servidor puede proporcionar información adicional sobre el problema, como el tiempo estimado de recuperación.



Códigos de estado

No es necesario aprenderlos a todos e, incluso, difícilmente utilices todas estas opciones durante el desarrollo de soluciones de software.

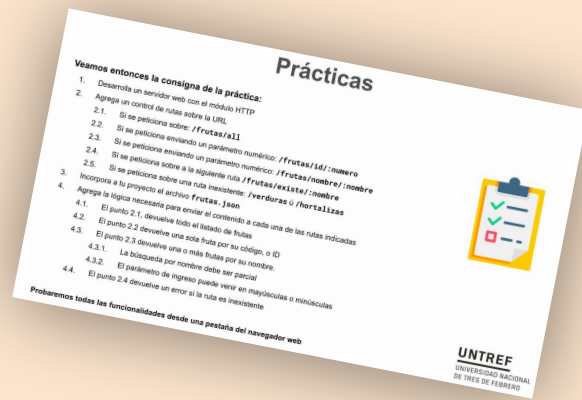
Con poder identificar el significado de cada grupo de códigos, te permitirá entender rápidamente si todo está OK o si hay algo erróneo en el camino.

Espacio de trabajo

Espacio de trabajo

Recrea el TP realizado en la clase 10, migrando su estructura de servidor web a Express JS. La funcionalidad deberá ser la misma que la solicitada en el trabajo práctico.

Actualiza el control de errores de este servidor web para las rutas inexistentes, de acuerdo a las opciones que vimos en esta unidad. Implementa la que más te parezca acorde.



Tiempo estimado: 30 minutos.



```
const questions = [ 'dudas', 'consultas', '🧐' ]
```



```
> node gracias.js
```

UNTREF

UNIVERSIDAD NACIONAL
DE TRES DE FEBRERO