

express

APRENDE
EXPRESS
DESDE CERO
HASTA AVANZADO

A C A D E M I A X

Contenido

1	Introducción	4
1.1	Bienvenida	4
1.1.1	Libro vivo	5
1.1.2	Alcance	5
1.2	Prerequisitos	6
1.3	¿Cómo evitar bloqueos?	6
2	Primeros pasos	7
2.1	Visión general	7
2.1.1	¿Qué es y porqué debes aprenderlo?	7
2.1.2	¿En dónde se utiliza?	8
2.1.3	¿Qué trabajos puedes conseguir?	9
2.1.4	¿Cuánto puedes ganar?	10
2.1.5	¿Cuales son las preguntas más comunes?	11
2.2	Historia, evolución, y versiones	12
2.3	Características, ventajas, y diferencias	13
2.4	Configuración	16
2.4.1	IDE	16
2.4.2	Entorno	16
2.5	Hola Mundo	19
3	Express	21
3.1	Instalación	21
3.2	Métodos	23
3.3	Códigos de Estado	27
3.4	Middleware	30
3.5	Ruteo	32
3.6	Contenido Estático	34
3.7	Plantillas	36

Contenido

3.8	Generador	39
4	Siguientes pasos	41
4.1	Herramientas	41
4.2	Recursos	42
4.3	¿Que viene después?	43
4.4	Preguntas de entrevista	45

1 Introducción

1.1 Bienvenida

Bienvenid@ a este libro de Academia X en donde aprenderás Express.

Hola, mi nombre es Xavier Reyes Ochoa y soy el autor de este libro. He trabajado como consultor en proyectos para Nintendo, Google, entre otros proyectos top-tier, trabajé como líder de un equipo de desarrollo por 3 años, y soy Ingeniero Ex-Amazon. En mis redes me conocen como Programador X y comparto videos semanalmente en YouTube desde diversas locaciones del mundo con el objetivo de guiar y motivar a mis estudiantes mientras trabajo haciendo lo que más me gusta: la programación.

Quiero expresar mi más sincero agradecimiento a Hugo Adrián Mejía Soteno por su valiosa contribución en la edición de este libro para Academia X como consultor. Su habilidad en el ámbito de la programación ha sido fundamental para dar vida a este recurso. Agradezco profundamente su dedicación y profesionalismo durante todo el proceso de creación. Espero que este libro sirva como un testimonio duradero de nuestro esfuerzo conjunto.

En este libro vas a aprender estos temas:

- Primeros pasos
- Instalación
- Métodos
- Códigos de Estado
- Middleware
- Ruteo
- Contenido Estático
- Plantillas
- Generador

La motivación de este libro es darte todo el conocimiento técnico que necesitas para elevar la calidad de tus proyectos y al mismo tiempo puedas perseguir metas más

1 INTRODUCCIÓN

grandes, ya sea utilizar esta tecnología para tus pasatiempos creativos, aumentar tus oportunidades laborales, o si tienes el espíritu emprendedor, incluso crear tu propio negocio en línea. Confío en que este libro te dará los recursos que necesitas para que tengas éxito en este campo.

Empecemos!

1.1.1 Libro vivo

Esta publicación fue planeada, editada, y revisada manualmente por Xavier Reyes Ochoa. La fundación del contenido fue generada parcialmente por inteligencia artificial usando ChatGPT (Feb 27, 2024 Version) de OpenAI. Puedes ver más detalles en <https://openai.com/>

Esto es a lo que llamo un trabajo **VIVO**, esto quiere decir que será actualizado en el tiempo a medida que existan cambios en la tecnología. La versión actual es 1.0.0 editada el 27 de marzo de 2024. Si tienes correcciones importantes, puedes escribirnos a nuestra sección de contacto en <https://www.academia-x.com>

1.1.2 Alcance

El objetivo de este libro es llenar el vacío que existe sobre esta tecnología en Español siguiendo el siguiente enfoque:

- Se revisan los temas con un enfoque práctico incluyendo ejemplos y retos.
- Se evita incluir material de relleno ya que no es eficiente.
- Se evita entrar en detalle en temas simples o avanzados no-prácticos.

Si deseas profundizar en algún tema, te dejo varios recursos oficiales, populares, y avanzados en la lección de recursos.

1.2 Prerequisitos

Antes de aprender Express, necesitas lo siguiente:

1. Un computador: cualquier computador moderno tiene las capacidades de correr este lenguaje. Te recomiendo un monitor de escritorio o laptop ya que dispositivos móviles o ipads no son cómodos para programar.
2. Sistema operativo: conocimiento básico de cómo utilizar el sistema operativo en el que se ejecutará Express (por ejemplo, Windows, MacOS, Linux). Te recomiendo tener el sistema operativo actualizado.
3. Conocimiento básico de la línea de comandos: se utiliza para ejecutar programas en Express.
4. Un editor de texto: lo necesitas para escribir código de Express. Los editores de texto más populares son Visual Studio Code, Sublime Text, Atom y Notepad ++.
5. Un navegador web y conexión al internet: es útil para investigar más sobre esta tecnología cuando tengas dudas. Los navegadores más populares son Google Chrome, Mozilla Firefox, Safari y Microsoft Edge. Se recomienda tener el navegador actualizado.

Si ya tienes estos requisitos, estarás en una buena posición para comenzar a aprender Express y profundizar en sus características y aplicaciones.

Si todavía no tienes conocimiento sobre algunos de estos temas, te recomiendo buscar tutoriales básicos en blogs a través de Google, ver videos en YouTube, o hacer preguntas a ChatGPT. Alternativamente, puedes empezar ya e investigar en línea a medida que encuentres bloqueos entendiendo los conceptos en este libro.

1.3 ¿Cómo evitar bloqueos?

Para sacarle el mayor provecho a este libro:

2 PRIMEROS PASOS

1. No solo leas este libro. La práctica es esencial en este campo. Practica todos los días y no pases de lección hasta que un concepto esté 100% claro.
2. No tienes que memorizarlo todo, solo tienes que saber donde están los temas para buscarlos rápidamente cuando tengas dudas.
3. Cuando tengas preguntas usa [Google](#), [StackOverFlow](#), y [ChatGPT](#)
4. Acepta que en esta carrera, mucho de tu tiempo lo vas utilizar investigando e innovando, no solo escribiendo código.
5. No tienes que aprender inglés ahora pero considera aprenderlo en un futuro porque los recursos más actualizados están en inglés y también te dará mejores oportunidades laborales.
6. Si pierdas la motivación, recuerda tus objetivos. Ninguna carrera es fácil pero ya tienes los recursos para llegar muy lejos. Te deseo lo mejor en este campo!

2 Primeros pasos

2.1 Visión general

2.1.1 ¿Qué es y por qué debes aprenderlo?

Express.js es un marco de aplicación web minimalista y flexible para Node.js. Proporciona un conjunto robusto de características que ayudan a desarrollar aplicaciones web y móviles. Es ampliamente utilizado debido a su facilidad de uso y su naturaleza ligera, lo que hace que el desarrollo en Node.js sea eficiente.

Aquí te dejo algunas razones por las que deberías considerar aprender Express.js:

1. **Eficacia:** Express.js está diseñado para la construcción de aplicaciones web y API rápido y fácil. Su configuración es simple pero altamente personalizable para construir el servidor web según tus necesidades.

2. **Middleware:** En Express.js, las funciones middleware tienen acceso al objeto de solicitud y respuesta. Esto ofrece la posibilidad de ejecutar cualquier código, hacer cambios en la solicitud y las respuestas, terminar el ciclo de solicitud-respuesta, e incluso llamar a la siguiente función middleware en la pila.
3. **Enrutamiento:** Express.js ofrece una forma directa de definir las rutas de tu aplicación. Puedes definir rutas para diferentes solicitudes HTTP como GET, POST, DELETE y PUT.
4. **Comunidad:** Express es muy popular entre los desarrolladores de Node.js. Por lo tanto, puedes encontrar muchas soluciones y recursos en línea en caso de que te encuentres con algún problema. Además, la comunidad siempre está lanzando nuevos paquetes y módulos que hacen que el desarrollo sea aún más rápido y eficiente.
5. **Base robusta:** Express.js ha sido desarrollado y mantenido por los mismos creadores de Node.js, lo que significa que está construido sobre una base sólida y estable.

Aprender Express.js te dará una gran ventaja en el desarrollo de aplicaciones JavaScript, ya que es una de las tecnologías clave en el desarrollo de aplicaciones web modernas.

2.1.2 ¿En dónde se utiliza?

Express, también conocido como Express.js, es un marco de aplicación web para Node.js. Se utiliza para construir aplicaciones web y API. También es bastante útil para configurar servidores intermedios capaces de responder a solicitudes HTTP, configurar rutas y gestionar la lógica de negocios, gestionar flujos de solicitudes y respuestas, e integrarse con bases de datos y bibliotecas para realizar operaciones CRUD.

En esencia, Express se puede usar en cualquier lugar donde se necesite una aplicación web o una API REST, como sitios web dinámicos, aplicaciones de una sola

página (SPA), aplicaciones de servidor isomórfico, y aplicaciones de tiempo real (RTA) como chats y juegos en línea.

Es un marco minimalista y flexible, que hace énfasis en la libertad para los desarrolladores a decidir cómo quieren estructurar su aplicación, en lugar de forzarlos a adherirse a patrones estrictos. Eso hace que sea una opción popular para una amplia gama de casos de uso.

2.1.3 ¿Qué trabajos puedes conseguir?

Estas son algunas de las posiciones profesionales que podrías considerar después de aprender Express.js:

1. **Desarrollador Backend Node.js:** Como desarrollador backend, tu trabajo principal sería manejar la lógica del servidor, la base de datos, la integración de todos los servidores y los sistemas de respaldo que garantizan que el usuario pueda navegar por el sitio de manera eficiente.
2. **Full Stack Developer:** En este rol, trabajarías tanto en el backend como en el frontend del desarrollo web. Como Full Stack Developer, necesitarás un conocimiento sólido de Express.js para manejar el aspecto del backend del desarrollo.
3. **Ingeniero de Software:** El uso de Express.js no está limitado a los desarrolladores web. Como ingeniero de software, puedes encontrar muchas oportunidades para usar Express.js, especialmente cuando se trabaja en aplicaciones de servidor en tiempo real.
4. **Desarrollador de APIs:** Algunos desarrolladores se especializan en la creación de API utilizando Node.js y Express.js. Estos desarrolladores crean las interfaces que permiten a las aplicaciones comunicarse entre sí.
5. **Desarrollador de Aplicaciones Móviles:** Node.js y Express.js se utilizan a menudo en el backend de las aplicaciones móviles. Como desarrollador de

aplicaciones móviles, puedes utilizar Express.js para gestionar las solicitudes del servidor, la autenticación del usuario y más.

Además de estas posiciones, el aprendizaje de Express.js podría abrirte oportunidades en muchas otras roles de desarrollo web. La demanda de desarrolladores con experiencia en Express.js sigue creciendo, así que sin importar a cuál de estos roles te sientas atraído, podrías encontrar muchas oportunidades de carrera emocionantes después de aprender Express.js.

2.1.4 ¿Cuánto puedes ganar?

Utilizar Express.js en tu trabajo puede tener un impacto positivo en tus ingresos potenciales, dependiendo de diversas circunstancias.

El salario de un desarrollador backend en Express en los Estados Unidos puede variar según diversos factores, como la ubicación geográfica, el nivel de experiencia, el tamaño de la empresa, entre otros. Sin embargo, en promedio, un desarrollador backend con experiencia en tecnologías como Express puede esperar ganar un salario anual que oscila entre los \$60,000 y los \$150,000 o más, dependiendo de los factores mencionados anteriormente. A continuación se detallan algunas de las formas en que puedes conseguirlo:

1. **Aumento del Valor de Mercado:** Como desarrollador, aprender y dominar una tecnología en demanda como Express.js puede aumentar tu valor en el mercado laboral. Esto podría llevar a aumentos de salario o a la posibilidad de negociar mejores condiciones de trabajo.
2. **Oportunidades de Carrera:** Express.js es una tecnología muy popular para el desarrollo de aplicaciones web. Empresas de diferentes tamaños, desde startups hasta corporaciones multinacionales, utilizan Express.js. Por lo tanto, dominarlo podría abrirte nuevas oportunidades de carrera.
3. **Freelancing:** Si prefieres trabajar de forma independiente, aprender Express.js podría ayudarte a acceder a una amplia gama de proyectos de

2 PRIMEROS PASOS

freelancing. La tarifa que puedes cobrar como freelance dependerá de varios factores, como tu experiencia y la complejidad del proyecto, pero el dominio de Express.js podría permitirte cobrar tarifas más altas.

4. **Desarrollo de producto:** Si tienes una idea para un producto o servicio web, Express.js puede ser una herramienta valiosa para desarrollar un prototipo o incluso un producto completo. A largo plazo, esto podría resultar en ingresos significativos, dependiendo del éxito del producto en el mercado.

Recuerda que aunque el dominio de una tecnología específica como Express.js puede contribuir a aumentar tus ingresos potenciales, también es fundamental tener un conjunto de habilidades sólido y diverso, así como la capacidad de aprender nuevas tecnologías a medida que evolucionan.

Es importante tener en cuenta que estos son solo promedios y que el salario real que puedes ganar puede ser mayor o menor, dependiendo de los factores mencionados anteriormente. Además, siempre es una buena idea investigar y hacer preguntas sobre los salarios y las condiciones laborales antes de aceptar un trabajo.

2.1.5 ¿Cuales son las preguntas más comunes?

1. ¿Qué es Express.js?
2. ¿Cómo se puede instalar Express.js?
3. ¿Cómo definir las rutas en una aplicación Express?
4. ¿Cuál es la mejor forma de manejar errores en Express?
5. ¿Cómo se puede servir archivos estáticos en Express?
6. ¿Cómo se pueden manejar consultas POST en Express?
7. ¿Cómo se pueden configurar las cookies con Express.js?
8. ¿Cómo se pueden implementar sesiones en Express?
9. ¿Cómo se puede conectar una base de datos como MongoDB o MySQL con Express?
10. ¿Cuán escalable es Express.js?

11. ¿Cómo se pueden usar middleware y para qué se usan en una aplicación Express.js?
12. ¿Cómo se pueden implementar procesos de autenticación en Express?
13. ¿Cómo integrar Express.js con un motor de plantillas como EJS o Pug?
14. ¿Cómo mejorar el rendimiento de una aplicación Express.js?
15. ¿Cómo se pueden gestionar las solicitudes CORS en Express.js?

Al finalizar este recurso, tendrás las habilidades necesarias para responder o encontrar las respuestas a estas preguntas fácilmente.

2.2 Historia, evolución, y versiones

Express.js, también conocido simplemente como Express, es un marco de aplicación web de back-end para Node.js, liberado como software libre bajo la licencia MIT. Es considerado el marco de servidor estándar para Node.js.

Historia y Evolución

Express fue desarrollado por TJ Holowaychuk, un miembro clave del equipo de Node.js, y lo describió como un servidor web sin servidor. La evolución de Express a lo largo de los años ha estado influenciada por una serie de factores como las necesidades del usuario, los cambios en la tecnología del servidor web y los avances en el ecosistema de Node.js.

La historia de Express comenzó en noviembre de 2010 con la versión 1.0.0. Esta versión inicial proporcionaba una interfaz de programación de aplicaciones (API) simplificada para construir servidores web y una serie de características útiles, como la capacidad para integrar “middlewares” en la tubería de manejo de solicitudes.

El marco web continuó evolucionando y en junio de 2012 se lanzó Express 3.0.0. Esta versión vino con varias mejoras, como soporte para rutas y “template engines”, haciendo que la creación de aplicaciones web fuese más eficiente.

La versión 4.0.0 se lanzó en abril de 2014 e introdujo cambios radicales. Se desacoplaron una serie de middleware en sus propios módulos, aportando así más flexibilidad y reduciendo la complejidad del núcleo de Express.

Con el tiempo, Express ha mantenido el ritmo de los cambios en el entorno de desarrollo web y ha incorporado nuevas características y mejoras. La versión actual en el momento de escribir esto es Express 4.17.1 que fue lanzada en 2019.

Versiones

A pesar de que Express se actualiza regularmente, la mayoría de las características entre las versiones son consistentes. Todas las versiones proporcionan una interfaz simple para crear aplicaciones web y APIs.

La versión 4.x es la más utilizada en la actualidad. Introdujo un enrutador modular, lo que significa que las rutas pueden ser divididas en archivos pequeños basados en un cierto conjunto de pautas.

Por otro lado, la versión 5.0 está en desarrollo y se espera que traiga una serie de mejoras significativas y características nuevas. Algunas de estas proposiciones son el manejo prometido de los errores, el soporte de los validadores y sanitizadores, etc.

A lo largo de las versiones, Express ha mantenido una sólida reputación por su robustez y facilidad de uso, convirtiéndose en el marco de referencia en el desarrollo de Node.js. Su minimalismo y flexibilidad, combinados con una gran comunidad de desarrolladores, aseguran que Express continúe siendo un pilar vital en el ecosistema de desarrollo web.

2.3 Características, ventajas, y diferencias

Características de Express.js

Express.js es un marco de aplicación web para Node.js, diseñado para la construcción de aplicaciones web y API. Aquí señalamos algunas de sus características clave:

1. **Middleware:** Express.js utiliza el concepto de funciones de middleware. Son funciones que tienen acceso al objeto de solicitud (request), al objeto de respuesta (response) y al siguiente middleware en el ciclo de solicitud/respuesta de la aplicación.
2. **Ruteo:** Express.js facilita la definición de rutas para diferentes solicitudes HTTP a URL específicas.
3. **Plantillas:** Express.js permite presentar los datos de la aplicación en una página web de forma muy eficiente mediante el uso de motores de plantillas.
4. **Gestión de Solicitud y Respuesta:** Express.js facilita la gestión de solicitudes y respuestas. Tiene manejadores de solicitudes y respuesta para las rutas definidas.

Ventajas de Express.js

Express.js lleva la simplicidad, la flexibilidad y lo ajustado de Node.js a un nuevo nivel, ya que permite al desarrollador manejar diferentes aspectos de una aplicación web de manera eficiente. Aquí puedes encontrar algunas ventajas:

1. **Fácil de aprender:** Si estás familiarizado con JavaScript, entonces Express.js es fácil de aprender y usar.
2. **Rápido y Eficiente:** Al ser una parte del entorno de Node.js, Express.js hereda su rendimiento. Es ligero, eficientemente asíncronico y de un simple pero robusto conjunto de funcionalidades.
3. **Soporte para Middleware:** Express.js tiene soporte completo para middleware, lo que permite a los desarrolladores escribir piezas muy flexibles y reutilizables de software.
4. **Gran Comunidad:** Express.js tiene una gran comunidad de desarrolladores que ofrecen soporte y contribuyen a mejorar su ecosistema.

Diferencias entre Express.js y otros lenguajes de programación

Es importante destacar que Express.js no es un lenguaje de programación, es un marco de aplicación web construido en Node.js, que es un entorno de ejecución de JavaScript. Aquí encontrarás algunas diferencias comparándolo con otros marcos de otros lenguajes:

1. **Express.js vs Django (Python):** Django es un marco de aplicación web de alto nivel que sigue el patrón MVT (Modelo-Vista-Plantilla), mientras que Express.js sigue el patrón MVC (Modelo-Vista-Controlador). Django tiene una estructura de directorios predefinida y requiere menos decisiones de diseño por parte del desarrollador. Express.js, por otro lado, es menos estricto y más flexible.
2. **Express.js vs Rails (Ruby):** Rails sigue una enfoque de convención sobre configuración, ofreciendo una estructura de directorio predefinida y requiriendo menos decisiones de diseño por parte del desarrollador. Express.js no impone una estructura de directorio específica y permite al desarrollador tomar más decisiones de diseño.
3. **Express.js vs Laravel (PHP):** Laravel es un marco PHP que proporciona muchas características robustas para el desarrollo web. Laravel tiene un ORM (Object-Relational Mapping) incorporado, mientras que en Express.js, el desarrollador tendría que integrar un ORM. Al igual que Rails, Laravel también sigue el enfoque de convención sobre configuración.

Express.js proporciona un marco de trabajo simplificado, altamente configurable y bastante eficiente para construir aplicaciones web y API, aprovechando el poder de JavaScript y Node.js.

2.4 Configuración

2.4.1 IDE

Los archivos de Express son archivos de texto. Puedes editarlos con **editores de texto** como Notepad en Windows o Notes en MacOS pero es recomendado utilizar un **IDE** (Integrated Development Environment) que es una aplicación de edición de código más avanzado que le da colores a tu código para que sea más fácil de leer y tengas funciones de autocompletado, entre otras. Algunos IDEs populares son [Brackets](#), [Atom](#), [Sublime Text](#), [Vim](#), y [Visual Studio Code](#).

El editor recomendado para practicar el código que vamos a ver es Visual Studio Code (o VSCode) que puedes bajar desde <https://code.visualstudio.com/>

2.4.2 Entorno

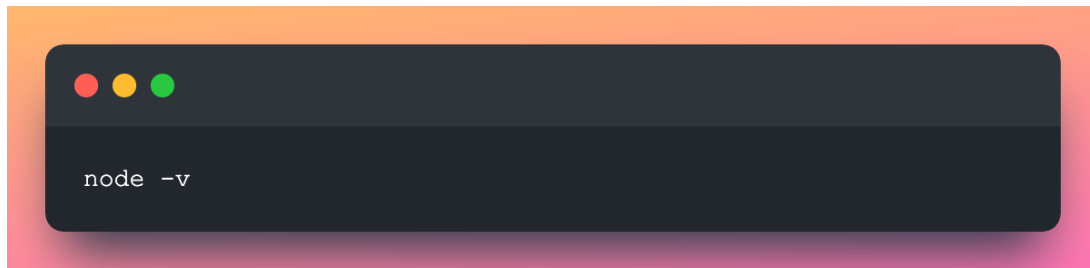
Para trabajar con Express en tu computadora, los pasos generalmente incluyen la instalación de Node.js, la instalación de Express con npm y finalmente la configuración de Express. Aquí detallo los pasos a seguir:

1. Instala Node.js

Express es un marco para Node.js, por lo que debes instalar Node.js en tu sistema. Descarga e instala Node.js desde su sitio oficial <https://nodejs.org/en/download/>.

Después de la instalación, puedes verificar que Node.js esté correctamente instalado ejecutando el siguiente comando en tu terminal:

2 PRIMEROS PASOS



Este comando debería mostrarte la versión de Node.js instalada.

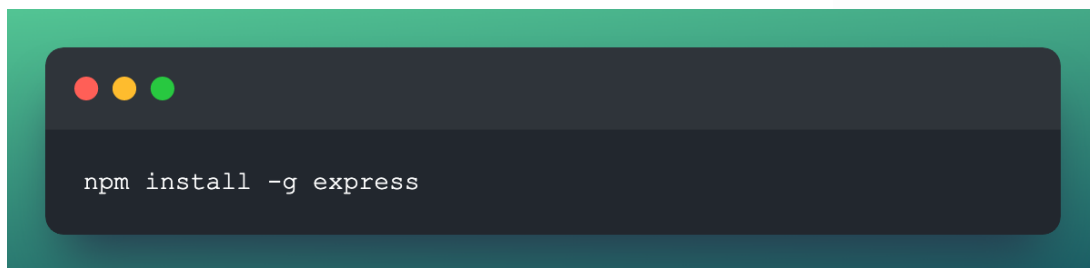
2. Instalar Express

Express se instala a través de npm (Node Package Manager), que se instala junto con Node.js. Para instalar Express, debes ejecutar el siguiente comando en tu terminal:



Esto instalará Express en tu directorio actual.

Si deseas instalar express de forma global en tu sistema, debes usar el siguiente comando:

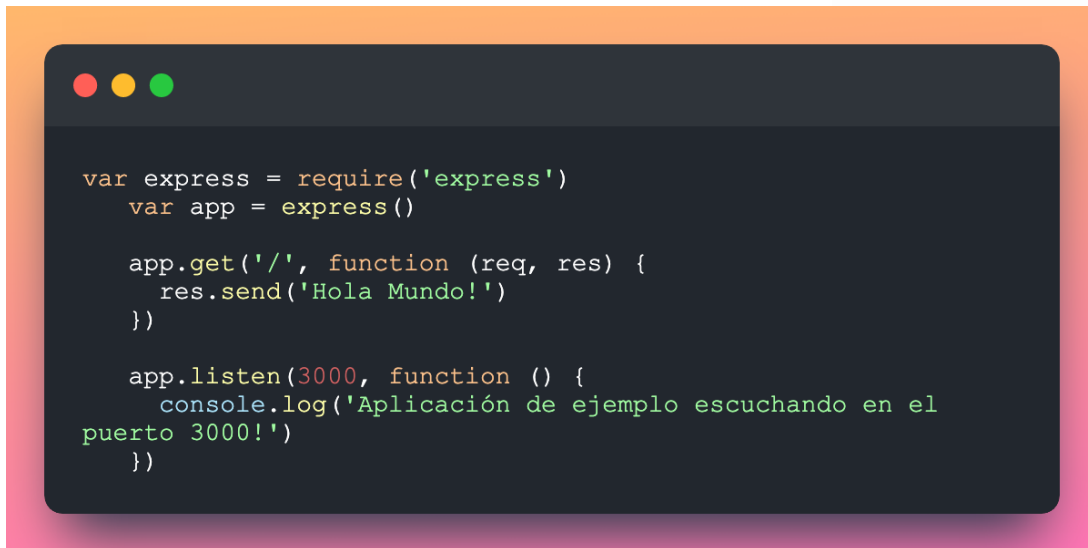


3. Configuración de Express

Para comenzar una aplicación Express, primero debes crear un nuevo archivo

2 PRIMEROS PASOS

.js, como app.js. En este archivo, debes requerir o solicitar Express mediante el uso de `require('express')`, lo cual permite acceder a todas las funcionalidades proporcionadas por este framework. Luego, en app.js, puedes escribir tu código de servidor Express para manejar rutas, solicitudes HTTP, respuestas y otras funcionalidades necesarias para tu aplicación web o API, como el siguiente ejemplo:

A code editor window with a dark background and a light orange header bar. The window contains the following JavaScript code:

```
var express = require('express')
var app = express()

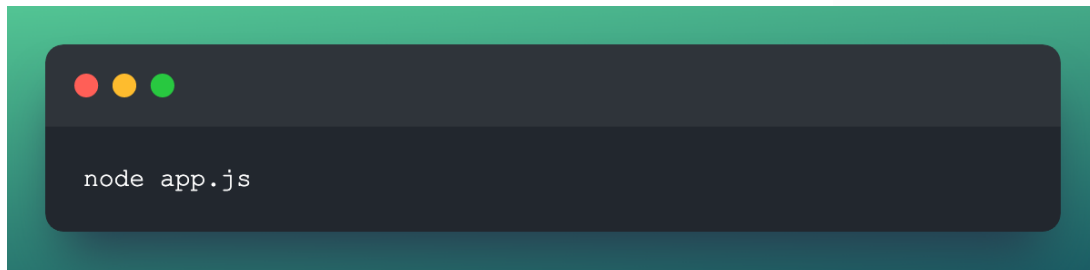
app.get('/', function (req, res) {
  res.send('Hola Mundo!')
})

app.listen(3000, function () {
  console.log('Aplicación de ejemplo escuchando en el puerto 3000!')
})
```

Este es un servidor Express muy básico que escucha en el puerto 3000 y responde con “Hola Mundo!” a las solicitudes HTTP GET al localhost en la ruta `'/'` .

4. Correr archivos de Express

Ahora, puedes ejecutar tu servidor Express con el siguiente comando en tu terminal:



En general, tu aplicación Express ahora se ejecuta en `http://localhost:3000`.

Por último, no olvides que la [documentación oficial de Express](#) es un gran recurso para aprender más sobre cómo configurar y utilizar Express. Está llena de ejemplos prácticos y explicaciones detalladas que deberían responder a la mayoría de tus preguntas.

¡Feliz codificación!

2.5 Hola Mundo

“Hola Mundo” es un ejemplo clásico que se utiliza para mostrar el funcionamiento básico de un lenguaje de programación.

En este ejemplo, se envía el texto “¡Hola mundo desde Express!” como respuesta en el navegador:

2 PRIMEROS PASOS



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

app.get('/', (req, res) => {
  res.send('¡Hola mundo desde Express!');
});

const PORT = process.env.PORT || 3000;

app.listen(PORT, () => {
  console.log(`Servidor Express escuchando en el puerto ${PORT}`);
});
```

1. Primero, importamos el módulo Express mediante `require('express')`, lo que nos permite acceder a todas las funcionalidades proporcionadas por Express.
2. Luego, creamos una instancia de la aplicación Express mediante `const app = express();`. Esto nos proporciona un objeto `app` que representa nuestra aplicación Express.
3. Definimos una ruta para la raíz de la aplicación utilizando `app.get('/' , ...)`. Esto significa que cuando se recibe una solicitud GET en la ruta raíz (`"/"`) del servidor, se ejecutará la función de devolución de llamada especificada. En este caso, la función de devolución de llamada simplemente envía la respuesta `"¡Hola mundo desde Express!"` utilizando `res.send()`.
4. Especificamos el puerto en el que queremos que la aplicación escuche las solicitudes. Utilizamos `process.env.PORT` para permitir que el puerto sea definido por una variable de entorno si está presente, lo que es útil si estamos desplegando la aplicación en un entorno como Heroku, por ejemplo. De lo contrario, utilizamos el puerto 3000.

3 EXPRESS

5. Finalmente, iniciamos el servidor Express llamando a `app.listen(PORT, ...)`. Esto hace que la aplicación Express comience a escuchar las solicitudes entrantes en el puerto especificado. También imprimimos un mensaje en la consola para indicar que el servidor se ha iniciado correctamente y en qué puerto está escuchando.

Este código crea un servidor Express que escucha en el puerto 3000 (o en el puerto definido por la variable de entorno `PORT`, si está definida). Cuando se accede a la ruta raíz (`/`) del servidor, se devuelve la cadena “¡Hola mundo desde Express!” como respuesta.

Reto:

Modifica el ejemplo anterior para imprimir “Hola Universo” en la consola.

3 Express

3.1 Instalación

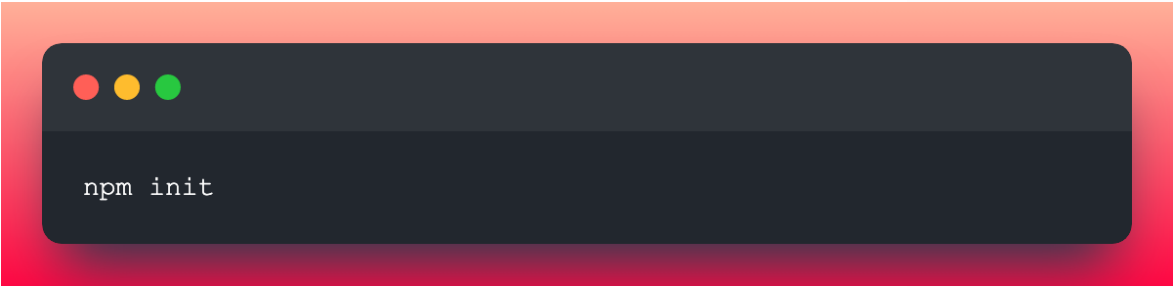
Express es un marco de aplicación web para Node.js que te permite manejar todo, desde rutas hasta manejo de errores y más. Es como una navaja suiza para el desarrollo de aplicaciones web. Sin iniciar un proyecto de Express, sería como intentar cortar una manzana con las manos en lugar de utilizar un cuchillo afilado.

Pasos para la Instalación de Express

Para instalar Express, necesitarás inicializar un nuevo proyecto de Node.js e instalar el módulo Express. Este proceso consta de dos etapas principales.

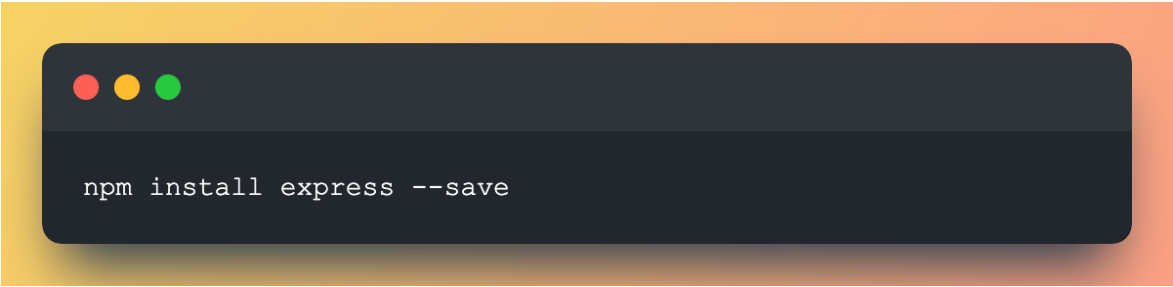
1. Inicia un nuevo proyecto de Node.js con `npm init`. Esto creará un nuevo archivo `package.json` en tu directorio de proyectos. Puedes pensar en `package.json` como la hoja de ruta para tu aplicación - es donde npm almacena las dependencias de tu proyecto.

3 EXPRESS

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. The text 'npm init' is displayed in a light gray monospace font.

```
npm init
```

2. Instala Express con `npm install express --save`. Esto añadirá Express a la lista de dependencias en tu archivo `package.json` y descargará el módulo Express en el directorio `node_modules`.

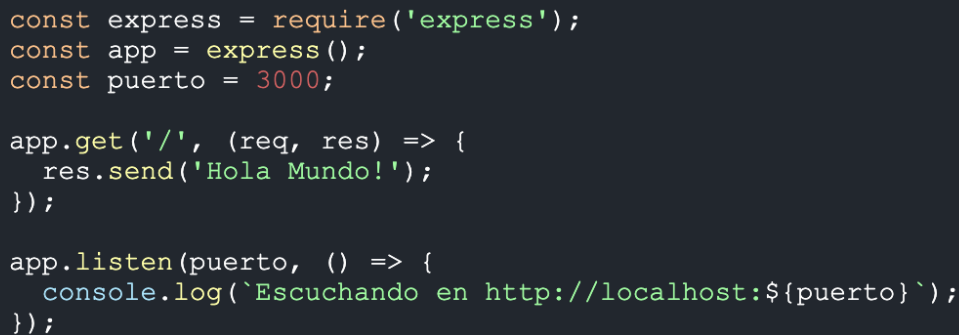
A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. The text 'npm install express --save' is displayed in a light gray monospace font.

```
npm install express --save
```

Ahora, estás listo para empezar a utilizar Express en tu aplicación.

Entendiendo el Código de Express

Aquí está el código Express básico que inicia un servidor de responde a las solicitudes HTTP GET al endpoint raíz (/) con “¡Hola Mundo!” .



```
const express = require('express');
const app = express();
const puerto = 3000;

app.get('/', (req, res) => {
  res.send('Hola Mundo!');
});

app.listen(puerto, () => {
  console.log(`Escuchando en http://localhost:${puerto}`);
});
```

1. Primero, importas Express usando require y lo asignas a una constante llamada express. Esta es la forma estándar de importar módulos en Node.js.
2. Luego, creas una nueva aplicación Express llamando a la función express(). Esto te devolverá un objeto que puedes usar para configurar tu aplicación.
3. A continuación, estableces el puerto en el que se ejecutará tu aplicación.
4. Después, defines una ruta. Cuando la aplicación reciba una solicitud GET a la ruta '/', se ejecutará la función de callback que proporcionas. Esta función recibe un objeto req (solicitud) y un objeto res (respuesta). Con res.send, puedes enviar una respuesta al cliente.
5. Finalmente, usa app.listen para iniciar tu aplicación. Le proporcionas el puerto y una función de callback que se ejecutará cuando la aplicación comience a escuchar en ese puerto.

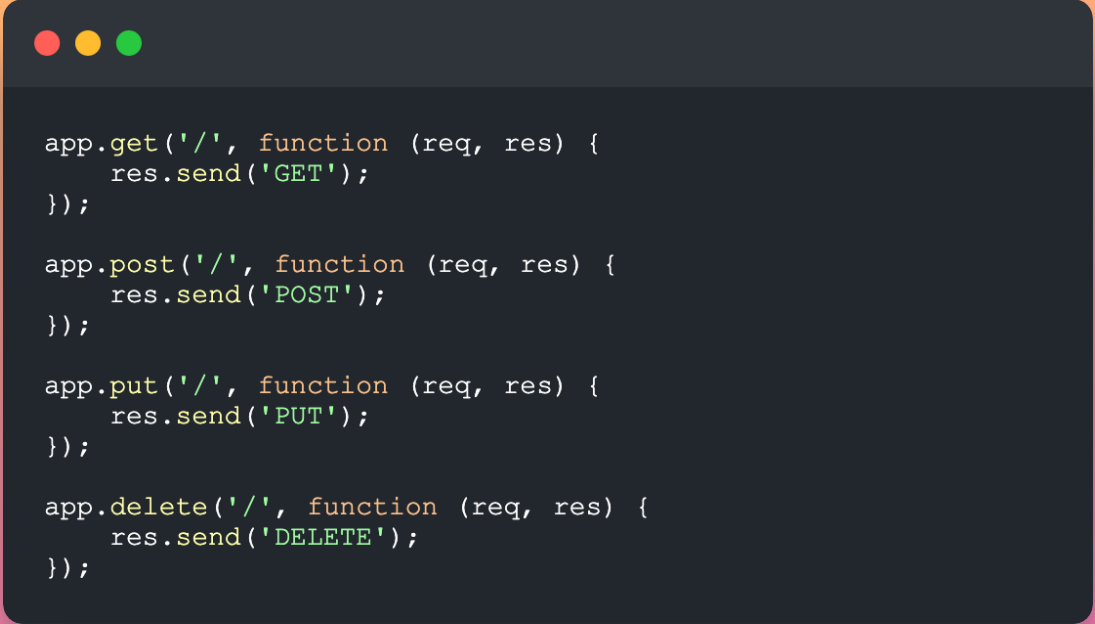
3.2 Métodos

En Express.js, un método se refiere a una forma en la que el servidor responde a una solicitud del cliente. Los métodos en Express.js representan los diferentes tipos

3 EXPRESS

de solicitudes HTTP que se pueden hacer desde un cliente a un servidor. Podemos usar métodos Express para solicitar, enviar o eliminar información del servidor.

Para que lo visualices de una manera más sencilla, imagina ir a tu tienda de comestibles local. Los métodos Express.js son como diferentes formas de pedirle al dependiente (el servidor) lo que necesitas. Podrías pedir algo (GET), traer algo para agregar (POST), cambiar algo que ya habías traído previamente (PUT), o pedir que se elimine algo (DELETE).



```
app.get('/', function (req, res) {  
  res.send('GET');  
});  
  
app.post('/', function (req, res) {  
  res.send('POST');  
});  
  
app.put('/', function (req, res) {  
  res.send('PUT');  
});  
  
app.delete('/', function (req, res) {  
  res.send('DELETE');  
});
```

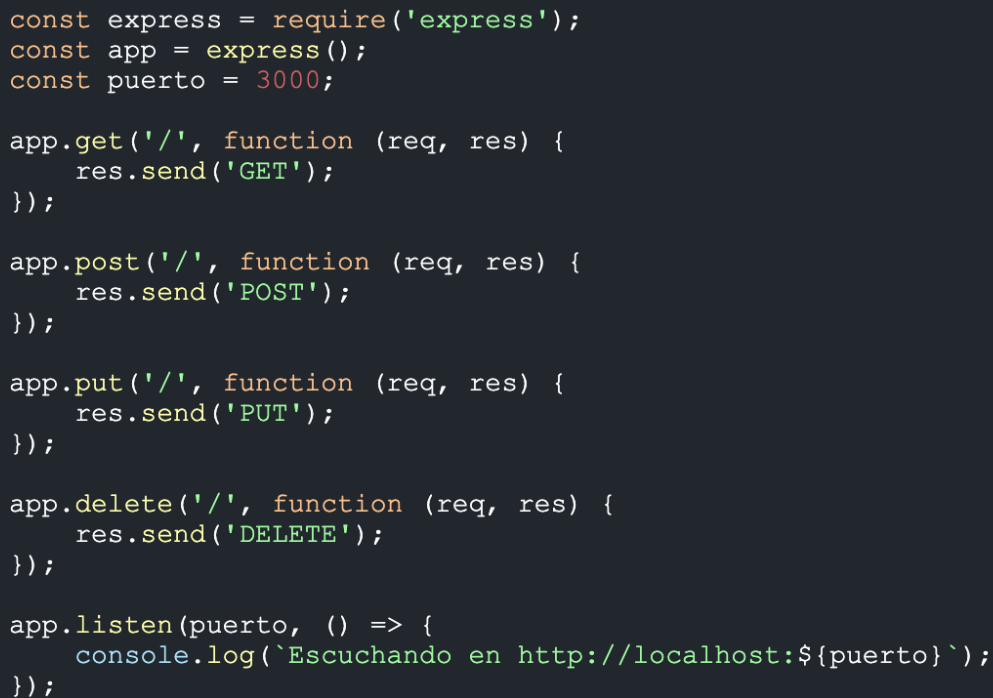
1. `app.get('/')`: Este método se utiliza para solicitar datos del servidor en la ruta especificada (`'/'`). Es una de las formas más comunes de interactuar con un servidor web cuando se desea obtener información. Imagina que estás visitando una página web, cuando introduces una URL en tu navegador y presionas Enter, tu navegador envía una solicitud GET al servidor para obtener la página web correspondiente. En este caso, cuando visitamos `http://localhost:3000/` en nuestro navegador, el servidor responderá con la cadena `"GET"` . (`'/'`).

3 EXPRESS

2. `app.post('/')`: El método POST se utiliza para enviar datos al servidor. Es útil cuando necesitas enviar información al servidor que puede ser procesada y almacenada, como enviar un formulario con datos de usuario. Cuando llenas un formulario en una página web y haces clic en el botón de enviar, tu navegador envía una solicitud POST al servidor con los datos del formulario. Por ejemplo, podrías enviar un formulario de registro con nombre de usuario y contraseña. En este caso, cuando se envía una solicitud POST a `http://localhost:3000/`, el servidor responderá con la cadena `"POST"` .
3. `app.put('/')`: El método PUT se utiliza para actualizar datos existentes en el servidor. Es útil cuando deseas realizar cambios en un recurso específico en el servidor. Supongamos que tienes un blog y deseas actualizar una publicación existente con nueva información. Puedes enviar una solicitud PUT al servidor con los datos actualizados de la publicación. En este caso, cuando se envía una solicitud PUT a `http://localhost:3000/`, el servidor responderá con la cadena `"PUT"` .
4. `app.delete('/')`: Como su nombre indica, el método DELETE se utiliza para eliminar datos del servidor. Es útil cuando necesitas eliminar un recurso específico almacenado en el servidor. Si deseas eliminar una publicación de tu blog, puedes enviar una solicitud DELETE al servidor con la identificación única de la publicación que deseas eliminar. En este caso, cuando se envía una solicitud DELETE a `http://localhost:3000/`, el servidor responderá con la cadena `"DELETE"` .

Estos son los métodos básicos de HTTP que Express.js provee para manejar solicitudes y respuestas en nuestro servidor, ejercitemos los conocimientos obtenidos.

3 EXPRESS



```
const express = require('express');
const app = express();
const puerto = 3000;

app.get('/', function (req, res) {
  res.send('GET');
});

app.post('/', function (req, res) {
  res.send('POST');
});

app.put('/', function (req, res) {
  res.send('PUT');
});

app.delete('/', function (req, res) {
  res.send('DELETE');
});

app.listen(puerto, () => {
  console.log(`Escuchando en http://localhost:${puerto}`);
});
```

En el código anterior, inicializamos Express y creamos un servidor que escucha en el puerto 3000. Después, definimos los métodos para las solicitudes HTTP “GET” , “POST” , “PUT” y “DELETE” en la ruta raíz de nuestro servidor (‘/’). Cada método simplemente envía una respuesta con su propio nombre cuando se solicita. Al ejecutar esta aplicación, el servidor comenzará a escuchar en `http://localhost:3000`, listo para responder a nuestras solicitudes utilizando los métodos que definimos.

3.3 Códigos de Estado

Primero que nada, los “Códigos de Estado” en Express son como las emociones digitales de tu aplicación web. Cada vez que el cliente (tu navegador, por ejemplo) le pide algo a tu servidor (tu aplicación Express), tu servidor responde con un mensaje. Y justamente, los Códigos de Estado son cómo tu servidor le dice al cliente cómo se sintió sobre su petición: contento, confundido, molesto, etc.

Objetivamente hablando, los códigos de estado HTTP son códigos numéricos estándar que son enviados en la respuesta de un servidor a una solicitud del cliente. Sirven para indicar la respuesta que se obtuvo a una petición determinada. Por ejemplo, si todo va bien y se puede realizar la operación pedida, se responde con un código 200, que significa “Ok” . Si el recurso solicitado no se encuentra, se responde con un código 404, que significa “Not Found” .

Ahora, la sintaxis para establecer códigos de estado en Express es `res.status(código)`. Aquí, `res` es el objeto de respuesta que obtienes como argumento en tus funciones de ruta. `status` es una función que define el código de estado HTTP para la respuesta y `código` es el número de status HTTP que quieres enviar.

Analiza el siguiente código y observa su explicación:

3 EXPRESS



```
const express = require('express');
const app = express();
const puerto = 3000;

app.get('/', function (req, res) {
  res.send('GET');
});

app.post('/', function (req, res) {
  res.status(201).send('POST');
});

app.put('/', function (req, res) {
  res.status(400).send('PUT');
});

app.delete('/', function (req, res) {
  res.status(401).send('DELETE');
});

app.listen(puerto, () => {
  console.log(`Escuchando en http://localhost:${puerto}`);
});
```

Se definen varias rutas con distintos tipos de petición HTTP (GET, POST, PUT, DELETE) en tu aplicación Express.

- `app.get('/')`: Cuando el cliente realiza una petición GET a la ruta raíz '/', el servidor simplemente responde con la cadena 'GET'.
- `app.post('/')`: Aquí, cuando el cliente realiza una petición POST a la ruta raíz, el servidor responde con un código de estado 201 (que indica que se ha creado algún recurso) y envía la cadena 'POST'.
- `app.put('/')` y `app.delete('/')`: Ambos responden respectivamente con los códigos de error 400 y 401, que indican algún tipo de error en la petición del cliente,

3 EXPRESS

y envían las cadenas 'PUT' y 'DELETE' .

Por último, escuchas el puerto que definiste (3000) y muestra un mensaje que indica que tu aplicación Express está activa y escuchando peticiones en ese puerto.

Categorías

Los códigos de estado HTTP se agrupan en categorías numéricas que proporcionan información sobre el resultado de una solicitud HTTP entre un cliente y un servidor.

En primer lugar, los códigos de estado 1xx son respuestas informativas que indican que la solicitud ha sido recibida y el servidor está procesando la petición. Por ejemplo, el código 100 (Continuar) indica que el servidor ha recibido los encabezados de la solicitud y el cliente debe continuar enviando el cuerpo de la misma.

Los códigos de estado 2xx son respuestas exitosas que indican que la solicitud del cliente ha sido recibida, entendida y aceptada correctamente por el servidor. Por ejemplo, el código 200 (OK) indica que la solicitud se ha completado correctamente.

En contraste, los códigos de estado 3xx son redirecciones que indican al cliente que debe realizar una acción adicional para completar la solicitud. Por ejemplo, el código 301 (Movido permanentemente) indica que la URL solicitada ha sido movida permanentemente a una nueva ubicación.

Los códigos de estado 4xx son errores del cliente que indican que la solicitud no se pudo completar debido a una acción incorrecta o faltante por parte del cliente, como el código 404 (No Encontrado) que indica que el recurso solicitado no se pudo encontrar en el servidor.

Finalmente, los códigos de estado 5xx son errores del servidor que indican que ha ocurrido un error en el servidor al intentar completar la solicitud del cliente, como el código 500 (Error Interno del Servidor) que indica un error interno en el servidor.

Cada grupo de códigos de estado proporciona información valiosa sobre el estado de la solicitud y es esencial para diagnosticar problemas y mejorar la comunicación

entre los componentes del sistema web. Lo más importante para recordar es que los códigos de estado son una forma de informar al cliente sobre la situación de su petición, y deben ser utilizados adecuadamente de acuerdo a lo que ocurra en el servidor.

3.4 Middleware

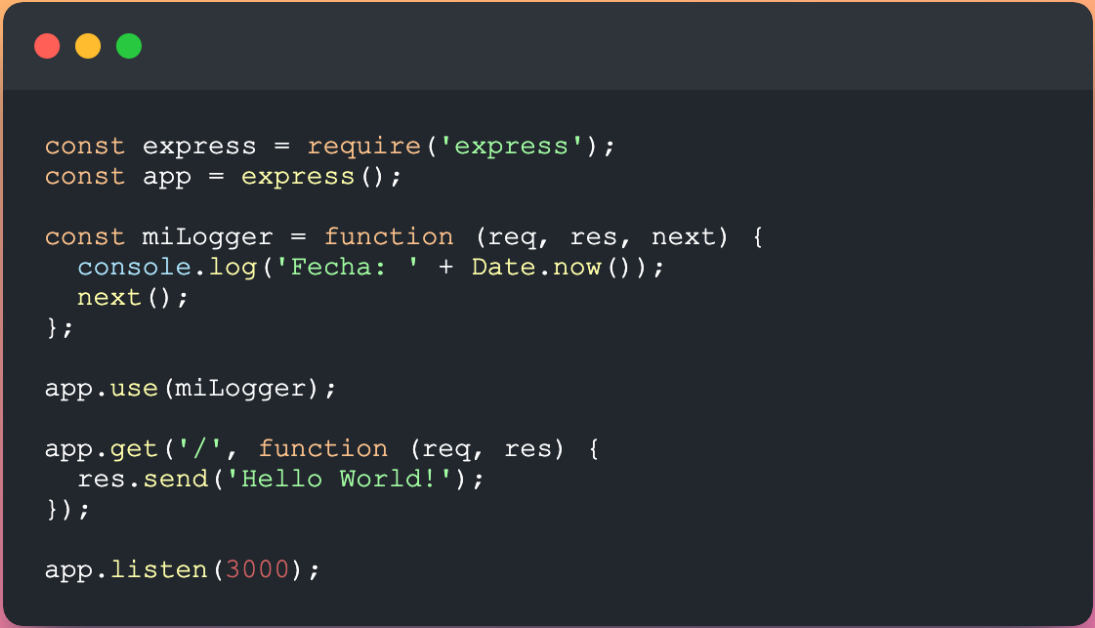
El término **Middleware** en Express.js se refiere a funciones que tienen acceso al objeto de solicitud (req), al objeto de respuesta (res), y a la siguiente función de middleware en el ciclo de solicitud/respuesta de la aplicación. En pocas palabras, los **middlewares** son como los filtros de una cafetera; el agua (datos) tiene que pasar por varios filtros (middleware) para convertirse en café (respuesta final).

Estas funciones pueden realizar las siguientes acciones:

- Ejecutar cualquier código.
- Hacer cambios en las solicitudes y respuestas.
- Finalizar el ciclo de solicitud-respuesta.
- Llamar a la siguiente función de middleware en la pila.

Si la función de middleware actual no termina el ciclo de solicitud-respuesta, debe llamar a `next()` para pasar el control a la siguiente función de middleware. De lo contrario, la solicitud se quedará colgada.

Ahora, echemos un vistazo a un ejemplo:



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

const miLogger = function (req, res, next) {
  console.log('Fecha: ' + Date.now());
  next();
};

app.use(miLogger);

app.get('/', function (req, res) {
  res.send('Hello World!');
});

app.listen(3000);
```

1. Primero, hemos definido `miLogger`, una función de middleware que simplemente registra la fecha y hora actual.
2. A continuación, usamos `app.use()` para indicar que nuestra aplicación debe usar este middleware.
3. Por último, dentro de `miLogger`, llamamos a `next()`, que le dice a Express que cuando esta función termine, debe ir a la siguiente función de middleware (si la hay) o a la ruta correspondiente.

Recuerda, los middlewares no sólo son un puente entre la solicitud y la respuesta, también pueden transformar los datos, ejecutar lógicas, y volver a dirigir a otro middleware o a la finalización de la respuesta.

Los middlewares se utilizan para realizar tareas como autenticación de usuarios, registro de solicitudes, manejo de errores, análisis de datos de entrada, etc. Los middlewares permiten modularizar el código y mejorar la seguridad, el rendimiento y la mantenibilidad de las aplicaciones web construidas con Express.js. Son una


herramienta versátil y poderosa para realizar diversas operaciones durante el procesamiento de solicitudes HTTP en una aplicación Express.js.

3.5 Ruteo

El ruteo es uno de los conceptos fundamentales en Express.js, un marco de aplicación web para Node.js. Puedes pensar en el ruteo como un sistema de señalización de tráfico. Cada vez que vas a una URL específica, el 'ruteo' en Express.js te dirige a la funcionalidad correcta en tu aplicación. Las rutas definen el punto de entrada a tu aplicación web y las diferentes acciones que deben tomar dependiendo de la URL que visita el usuario.

El objetivo de Express.js y su sistema de ruteo es proporcionar una forma simple, potente y eficiente para construir aplicaciones web y API's en Node.js. En un par de palabras, la ruteo en Express.js facilita toda la magia detrás de cómo los usuarios interactúan con tu aplicación web o API.

Veamos la sintaxis básica de la ruta raíz '/' :



```
app.get('/', function (req, res) {  
  res.send('Hogar');  
});
```

El sistema de ruteo en Express.js facilita la organización y estructuración del código de la aplicación, permitiendo definir múltiples rutas y funcionalidades asociadas a cada una de ellas.

Analiza el siguiente ejemplo:

3 EXPRESS

```
const express = require('express');
const app = express();
const puerto = 3000;
const usuarios = [
  'Robert Plant',
  'Jimmy Page',
  'John Paul Jones'
];

app.get('/', function (req, res) {
  res.send('Hogar');
});

app.get('/z/', function (req, res) {
  res.send(req.url);
});

app.get('/usuarios', function (req, res) {
  const html = usuarios.map(usuario => `<li>${usuario}</li>`).join('');
  res.send(html);
});

app.post('/usuarios', function (req, res) {
  const usuario = req.body.usuario;
  usuarios.push(usuario);
  res.status(201).json(usuario);
});

app.put('/usuarios/:id', function (req, res) {
  const usuario = req.body.usuario;
  const indice = req.params.id;
  usuarios[indice] = usuario;
  res.json(usuario);
});

app.delete('/usuarios/:id', function (req, res) {
  const indice = req.params.id;
  const borrado = usuarios.splice(indice, 1);
  res.json(borrado);
});

app.listen(puerto, () => {
  console.log(`Escuchando en http://localhost:${puerto}`);
});
```

3 EXPRESS

La sintaxis anterior corresponde a una ruta. Está asociada con la URL principal de tu aplicación (`'/'`) a través del método `get()` de Express. `app.get()` toma dos argumentos: la ruta como una cadena de texto y un callback. El callback acepta dos argumentos también, `req` y `res` que representan la solicitud y la respuesta, respectivamente.

Cada URL visitada por el usuario es dirigida por el sistema de ruteo a una función específica en la aplicación. En Express.js, las rutas se definen utilizando métodos como `get()`, `post()`, `put()`, y `delete()`, que corresponden a los diferentes tipos de solicitudes HTTP. Por ejemplo, `app.get('/' , ...)` define la ruta raíz de la aplicación, mientras que `app.post('/usuarios' , ...)` define una ruta para manejar solicitudes POST en la URL `'/usuarios'` .

Las líneas incluyen el conjunto de usuarios y la definición de diversas rutas para manejar las solicitudes GET, POST, PUT y DELETE. Cada una de estas rutas tiene su propia funcionalidad definida con un callback.

El fragmento `app.get(/z/,...)` captura cualquier ruta que contenga una `'z'` en su URL. `app.listen()` “enciende” la aplicación e inicia un servidor que escucha las solicitudes entrantes en el puerto definido.

El ruteo en Express.js es esencial para el desarrollo de aplicaciones web y API' s en Node.js, proporcionando una forma sencilla y eficiente de definir cómo los usuarios interactúan con la aplicación a través de las diferentes URLs.

3.6 Contenido Estático

El “Contenido Estático” en Express se refiere a los archivos que no son procesados ni modificados por tu servidor antes de ser enviados al navegador del usuario. Son archivos como CSS, Javascript e imágenes que viven en la carpeta “público” en tu aplicación Express.

Podrías pensar en tus archivos estáticos como en los muebles de tu casa. Los mue-

3 EXPRESS

bles no cambian a menudo, están siempre en el mismo lugar y son esenciales para la comodidad y la funcionalidad de tu hogar. De la misma manera, tus archivos estáticos proporcionan estilo, comportamiento y contenido visual a tu aplicación Express.

Observa la sintaxis básica para servir contenido estático:




```
app.use(express.static(path.join(__dirname, 'public')));
```

La línea `app.use(express.static(path.join(__dirname, 'public')));` en tu archivo de servidor declara que los archivos estáticos de tu aplicación están en una carpeta llamada `'public'`. Express busca esta carpeta en tu directorio actual (`__dirname`) y sirve estos archivos estáticos a tu aplicación.

La sintaxis más común para servir contenido estático en Express utiliza el middleware `express.static()`, que se configura para buscar los archivos estáticos en una carpeta específica, como `'public'`. En el código proporcionado, la línea `app.use(express.static(path.join(__dirname, 'public')));` declara que los archivos estáticos de la aplicación se encuentran en la carpeta `'public'` en el directorio actual (`__dirname`). Esto le indica a Express que sirva estos archivos estáticos a través de la ruta base de la aplicación.

Ahora, comprende el siguiente fragmento de código:



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

const app = express();
const puerto = 3000;

app.use(express.static(path.join(__dirname, 'publico')));

app.listen(puerto, () => {
  console.log(`Escuchando en http://localhost:${puerto}`);
});
```

Primero se importan los módulos necesarios (express y path). Luego, se inicializa una nueva aplicación Express y se configura un puerto para que escuche.

El código `app.use(express.static(path.join(__dirname, 'publico')));` le dice a Express que sirva los archivos estáticos desde la carpeta 'publico' en el directorio actual.

Por último, el servidor comienza a escuchar en el puerto configurado, listo para recibir y responder a las solicitudes.

Este enfoque permite a los desarrolladores proporcionar fácilmente recursos estáticos a sus aplicaciones Express, lo que mejora la experiencia del usuario al cargar estilos, scripts y otros activos visuales sin necesidad de procesamiento adicional por parte del servidor.

3.7 Plantillas

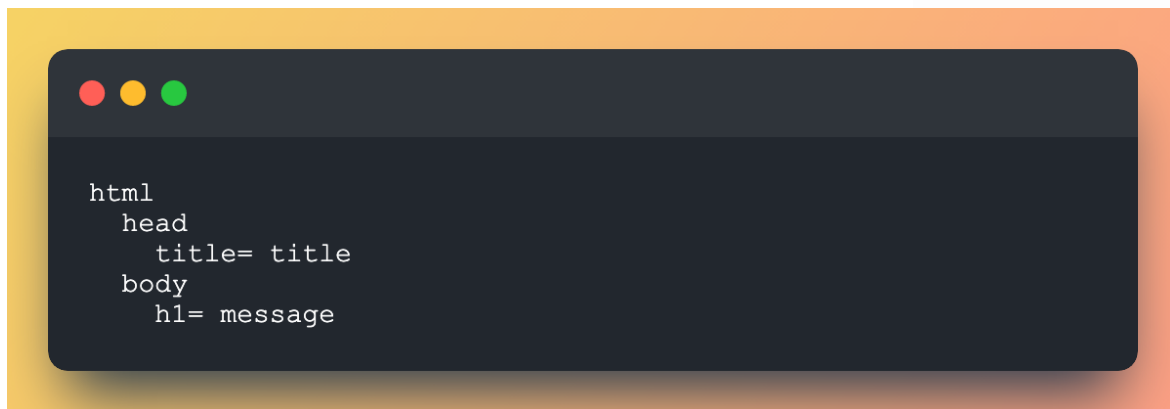
Las plantillas en Express son como los andamios que usan los constructores, pero en lugar de ladrillos y cemento, aquí construyes páginas web con datos y HTML. Su objetivo principal es incrustar datos en nuestras páginas web de forma dinámica

3 EXPRESS

y eficiente. No te preocupes, el trabajo pesado y repetitivo lo hace Express por ti, así podrás centrarte en lo que realmente importa: diseñar y construir tus páginas y aplicaciones web.

Las plantillas permiten generar HTML que varía según la información que se pasa desde el servidor. Por ejemplo, puedes mostrar el nombre de usuario en la página de inicio de sesión, y eso sería difícil (casi imposible) de realizar con HTML estático. Por lo tanto, las plantillas hacen que generar ese tipo de HTML sea mucho más fácil y eficiente.

Por ejemplo, en Express se utiliza la plantilla Pug (anteriormente conocida como Jade), que permite escribir en una sintaxis más fácil y limpiar que el HTML.



En el código anterior, Express se configura para usar un motor de plantillas llamado pug. A través de la ruta de acceso 'views' y 'view engine', se especifica dónde se pueden encontrar las plantillas de pug y cómo se procesarán.

A continuación, en la ruta '/', se presenta una función callback que utiliza el método `res.render` para enviar una respuesta HTML que es una función de la plantilla `index.pug` y datos (`{ title: 'Ejemplo', message: 'Hello there!' }`).

3 EXPRESS

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

const app = express();
const puerto = 3000;

app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'pug');

app.get('/', function (req, res) {
  res.render('index', { title: 'Ejemplo', message: 'HTML
generado con pug!' });
});

app.listen(puerto, () => {
  console.log(`Escuchando en http://localhost:${puerto}`);
});
```

Este script se inicia un servidor en localhost con el puerto 3000 y si visitas esta dirección en tu navegador, verás un mensaje que dice “HTML generado con pug”, que se genera dinámicamente a través de la plantilla ‘index.pug’.

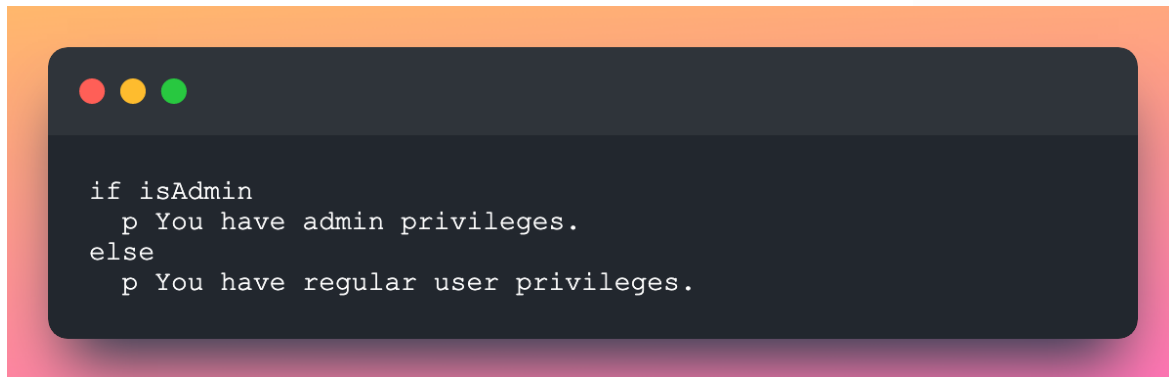
Las plantillas en Express permiten la interpolación de variables, lo que significa que puedes pasar datos dinámicos desde el servidor para ser mostrados en el HTML generado. Por ejemplo:

```
h1 Welcome, #{username}!
```

3 EXPRESS

Aquí, `username` es una variable que se pasará desde el servidor al renderizar la plantilla.

Además de la interpolación de variables, las plantillas también admiten estructuras de control, como condicionales y bucles, lo que permite una generación aún más dinámica de HTML. Por ejemplo:



Las plantillas en Express son una herramienta esencial para generar HTML dinámico de forma eficiente. Con la capacidad de interpolar variables, utilizar estructuras de control y aprovechar layouts y partials, las plantillas hacen que la creación de páginas web personalizadas sea fácil y efectiva. Al dominar el uso de plantillas en Express, los desarrolladores pueden construir aplicaciones web modernas y atractivas con facilidad, centrando su atención en el diseño y la funcionalidad.

3.8 Generador

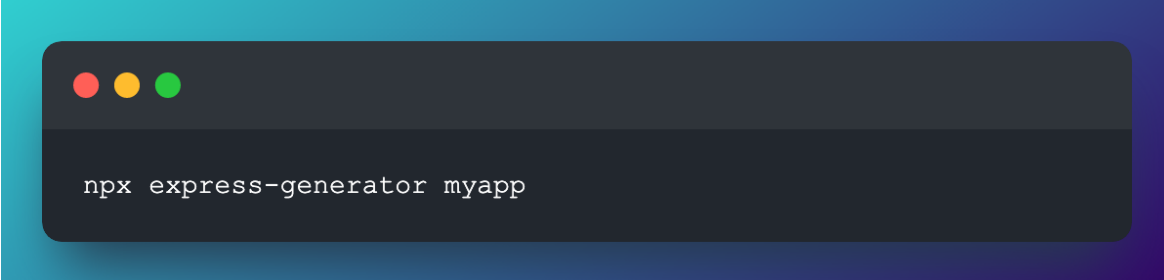
Los generadores son herramientas que nos permiten iniciar una aplicación Express.js de forma rápida y personalizada. Imagina que estás construyendo un coche, podrías hacerlo pieza por pieza desde cero OR, podrías usar una configuración preestablecida (como un coche deportivo, un camión, un todoterreno, etc.), que viene con todas las partes necesarias ya seleccionadas y organizadas para ayudarte a comenzar. De manera similar, un generador Express.js te proporciona

3 EXPRESS

una configuración básica de la aplicación con la que puedes comenzar, en lugar de tener que empezar desde cero.

Los generadores Express.js se utilizan principalmente para iniciar una nueva aplicación Express. Te proporciona un marco básico de aplicación, que incluye algunas rutas predefinidas, archivos de plantillas y configuraciones de servidores. En lugar de escribir todo esto manualmente, puedes utilizar el generador para hacerlo de manera rápida y eficiente, lo que te permite empezar a trabajar en la aplicación de inmediato.

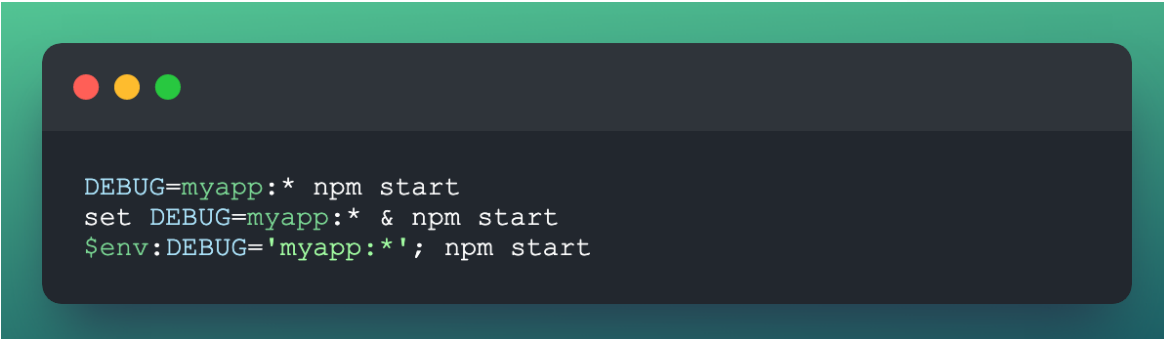
Aquí hay un ejemplo de cómo usar un generador en Express.js:



```
npx express-generator myapp
```

- 'npx' es una herramienta que te permite ejecutar paquetes de Node.js.
- 'express-generator' es el generador de aplicaciones Express.js que estamos utilizando.
- 'myapp' es el nombre de la nueva aplicación que estás creando.

El código a continuación es cómo se configura y se inicia la aplicación Express generada. Permíteme explicarte qué hace cada línea:



```
DEBUG=myapp:* npm start  
set DEBUG=myapp:* & npm start  
$env:DEBUG='myapp:*'; npm start
```


4 SIGUIENTES PASOS

- 'DEBUG=myapp:* npm start' es un comando que se utiliza en sistemas UNIX para iniciar la aplicación. Aquí, 'DEBUG=myapp:*' configura la aplicación para mostrar información de depuración en la consola, y 'npm start' inicia la aplicación.
- 'set DEBUG=myapp:* & npm start' hace lo mismo, pero es para sistemas Windows.
- '\$env:DEBUG='myapp:*'; npm start' es otra versión para sistemas Windows que utiliza PowerShell en lugar de la línea de comandos tradicional.

Y ahí lo tienes, eso es lo básico de los generadores Express.js y cómo se utilizan para iniciar una aplicación Express.js. Es una herramienta de ahorro de tiempo que te permite saltar directamente al desarrollo de tu aplicación.

4 Siguientes pasos

4.1 Herramientas

1. **Express.js:** Es un framework minimalista y flexible para desarrollar aplicaciones web en Node.js. Proporciona un gran conjunto de características para las aplicaciones móviles y web. <https://expressjs.com/>
2. **Morgan:** Es un middleware de registro de solicitudes HTTP para node.js. Hace el seguimiento de las solicitudes a la aplicación, lo que ayuda a identificar problemas. <https://www.npmjs.com/package/morgan>
3. **Body-Parser:** Body-parser extrae toda la parte del cuerpo de un flujo de solicitud entrante y la expone en req.body como algo más fácil de interactuar con. Esto es útil al crear servicios que procesen o almacenen la información enviada. <https://www.npmjs.com/package/body-parser>

4 SIGUIENTES PASOS

4. **CORS:** Este módulo proporciona un middleware que se puede usar para habilitar CORS con diversas opciones. <https://www.npmjs.com/package/cors>
5. **Mongoose:** Proporciona una solución directa y basada en esquemas para modelar los datos de su aplicación. Incluye validación de tipos incorporada, consultas, emulación de middleware de estilo de vida para Mongo, etc. <https://www.npmjs.com/package/mongoose>
6. **Nodemon:** Es una utilidad que monitorea cualquier cambio en el código fuente y automáticamente reinicia tu server. Es perfecto para el desarrollo ya que no tienes que detener y reiniciar el server cada vez que haces un cambio. <https://www.npmjs.com/package/nodemon>
7. **DotEnv:** Dotenv es un módulo de dependencia cero que carga variables de entorno desde un archivo .env a process.env. Esto es útil para esconder información confidencial. <https://www.npmjs.com/package/dotenv>
8. **Cookie-Parser:** Parsea las cookies y nos pone las cookies disponibles en req.cookies con la firma de la cookie que nos proporciona req.signedCookies si las cookies han sido firmadas. <https://www.npmjs.com/package/cookie-parser>
9. **Jsonwebtoken:** Este paquete nos permite autenticar usuarios con JWT (JSON Web Tokens) que son una forma abierta estándar de representar reclamos de forma segura entre dos partes. <https://www.npmjs.com/package/jsonwebtoken>
10. **Multer:** Multer es un middleware para el manejo de los multipart/form-data, que se utilizan para la carga de archivos. <https://www.npmjs.com/package/multer>

4.2 Recursos

1. **Sitio web oficial de Express.js:** Es el mejor lugar para empezar a aprender Express.js, ya que es el recurso establecido por los creadores del marco. Contiene documentación completa y actualizada, tutoriales y guías, ejemplos de

4 SIGUIENTES PASOS

códigos y más. [Express](#)

2. **Stack Overflow:** Es un recurso invaluable para todos los programadores, sin importar su nivel de habilidad. Contiene una gran cantidad de preguntas y respuestas relacionadas con Express.js y te permite hacer tus propias preguntas si no puedes encontrar una solución a tu problema. [Stack Overflow](#)
3. **GitHub:** Es un repositorio donde puedes encontrar y compartir código. Esto incluye el código fuente de Express.js y miles de proyectos que utilizan Express.js. Esto puede ser útil para ver cómo otros programadores están utilizando Express.js en el mundo real. [GitHub](#)
4. **MDN Web Docs:** Es un recurso establecido por Mozilla y es una de las mejores fuentes de información en línea sobre tecnología de la web, incluyendo Express.js. Los documentos son fáciles de entender y cubren una amplia gama de temas. [MDN Web Docs](#)
5. **NPM:** Es el administrador de paquetes predeterminado para Node.js y contiene una gran cantidad de módulos y paquetes que puedes usar en tus aplicaciones Express.js. [NPM](#)
6. **Node School:** Este sitio ofrece tutoriales interactivos en línea, incluyendo uno en Express.js llamado “expressworks” . Este tutorial te enseñará a usar Express.js a través de ejercicios de codificación que puedes hacer directamente en tu navegador. [Node School](#)
7. **W3Schools:** Este sitio es conocido por sus tutoriales en línea fáciles de seguir sobre una gran variedad de tecnologías web, incluyendo Express.js. [W3Schools](#)

4.3 ¿Que viene después?

1. **Poner en práctica lo aprendido:** El paso más inmediato después de aprender Express es poner en práctica lo aprendido. Crea pequeños proyectos, como

4 SIGUIENTES PASOS

una API sencilla, para familiarizarte con la sintaxis y las capacidades de Express. Así ganarás experiencia y confianza.

2. **Estudiar las funcionalidades avanzadas de Express:** Aprende sobre el middleware de Express, los manejadores de errores, la validación de peticiones y más. Existen muchos tutoriales y documentación útil para explorar.
3. **Aprender bases de datos:** Las aplicaciones web a menudo necesitan una forma de almacenar y recuperar datos. Aprende sobre bases de datos relacionales como PostgreSQL o MySQL, así como bases de datos NoSQL como MongoDB. Además, puedes aprender ORM (Object-Relational Mapping) como Sequelize o Mongoose para interactuar con la base de datos en un nivel más alto de abstracción.
4. **Autenticación y autorización:** Estos son conceptos importantes en el desarrollo web. Aprende sobre estrategias de autenticación, como JWT (JSON Web Tokens), y cómo implementar roles de usuario.
5. **Dominar las pruebas de software:** Las pruebas son cruciales para mantener la calidad del software y permitir actualizaciones y nuevas características. Aprende sobre las pruebas unitarias y de integración para tus aplicaciones Express.
6. **Implementar tu aplicación:** Aprende sobre las diferentes opciones de despliegue. Desde implementar tu aplicación en un servidor propio hasta utilizar plataformas como Heroku o AWS.
7. **Seguir aprendiendo y practicando:** La tecnología y las mejores prácticas cambian con frecuencia. Mantén actualizados tus conocimientos y siempre encuentra formas de mejorar y expandir tus habilidades. Cada aplicación que construyas te ayudará a seguir creciendo como desarrollador.

4.4 Preguntas de entrevista

1. ¿Qué es Express.js?

- Express.js es un marco de aplicación web para Node.js que se utiliza principalmente para desarrollar aplicaciones web y API. Fue diseñado para la construcción de aplicaciones web y API. Es la pieza E estándar de las aplicaciones MEAN y MERN.

2. ¿Cuáles son las principales características de Express.js?

- Las características más importantes de Express.js incluyen configuración sencilla, enrutamiento sólido, integración con vistas e integración con bases de datos.

3. ¿Cómo se define una ruta básica en Express.js?

- Una ruta básica en Express.js se define de la siguiente manera: `app.get('/', function(req, res){ res.send('Hola Mundo');});`

4. ¿Qué son los motores de vista en Express.js?

- Los motores de vista en Express.js son plantillas que permiten inyectar datos dinámicos en HTML. Ejemplos de plantillas son Pug, EJS y Mustache.

5. ¿Cómo se envía una respuesta JSON en Express.js?

- Para enviar una respuesta JSON en Express.js, usarías el método `res.json()`. Por ejemplo: `res.json({ usuario: 'John Doe' });`

4 SIGUIENTES PASOS

6. ¿Qué es 'app' en Express?

- 'app' es la instancia de la aplicación Express invocada por `express()`.

7. ¿Cómo se sirven archivos estáticos en Express.js?

- Los archivos estáticos se sirven en Express.js usando la función middleware `express.static()`. Por ejemplo: `app.use(express.static('public'))`.

8. ¿Qué es Middleware en Express.js?

- Middleware son funciones que tienen acceso al objeto de solicitud (`req`), al objeto de respuesta (`res`) y a la siguiente función middleware en el ciclo de solicitud/respuesta de la aplicación.

9. ¿Cómo captura y maneja errores Express.js?

- Express.js captura y maneja errores usando middleware de errores. Los errores se pasan usando `next()` y los captura un middleware de error al final de la cadena.

10. ¿Cómo rediriges a otra página en Express.js?

- En Express.js, puedes redirigir a otra página usando el método `res.redirect()`. Por ejemplo, `res.redirect('/login')`.

11. ¿Cómo usas una base de datos con Express.js?

- Puedes usar una base de datos con Express.js con ayuda de librerías adecuadas o drivers. Por ejemplo, para MongoDB, puedes usar Mongoose y para PostgreSQL puedes usar pg-promise.

4 SIGUIENTES PASOS

12. ¿Cómo estableces una cookie en Express.js?

- En Express.js, puedes establecer una cookie con el método `res.cookie()`. Por ejemplo: `res.cookie('nombre', 'valor')`.

13. ¿Cómo extraes parámetros de la URL en Express.js?

- Puedes extraer parámetros de la URL en Express.js usando `req.params`. Por ejemplo, para extraer un ID de usuario de una ruta como `/users/:id`, usarías `req.params.id`.

14. ¿Qué es la inyección de dependencias en Express.js?

- En Express.js, la inyección de dependencias permite sustituir módulos de la aplicación en tiempo de ejecución, facilitando el testing y la modularidad.

15. ¿Cómo manejas las sesiones en Express.js?

- Las sesiones en Express.js se pueden manejar con el middleware `express-session`.

16. ¿Cómo manejas el CORS en Express.js?

- El CORS (Cross-Origin Resource Sharing) se puede manejar en Express.js con la ayuda de un paquete como `cors` o manualmente agregando headers específicos a las respuestas.

17. ¿Cómo envías un archivo al cliente en Express.js?

- Puedes enviar un archivo al cliente en Express.js usando el método `res.sendFile()`. Por ejemplo: `res.sendFile(path.join(__dirname, 'index.html'))`.

18. ¿Cómo maneja la autenticación en Express.js?

- La autenticación en Express.js se puede manejar mediante el uso de middleware de autenticación como `passport.js`, que puede integrarse con diversas estrategias de autenticación.

4 SIGUIENTES PASOS

19. ¿Cómo se configura una plantilla de vista en Express.js?

- Puedes configurar una plantilla de vista en Express.js usando `app.set('view engine', 'nombre_del_motor')`, y después puedes usar `res.render('nombre_de_vista', { datos })` para renderizar la vista.

20. ¿Cómo haces pruebas unitarias en Express.js?

- Las pruebas unitarias en Express.js se pueden realizar con librerías como Mocha, Chai y Sinon. Se crean escenarios esperados y se verifica si el código produce los resultados esperados.