UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Centro de e-Learning

NodeJS Nivel Intermedio

Centro de e-Learning SCEU UTN - BA.
Medrano 951 2do piso (1179) // Tel. +54 11 4867 7589 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

Módulo 1

Unidad 2: Introducción a Express

Presentación:

Actualmente NodeJS es una de las principales tecnologías de desarrollo del lado servidor. Es utilizado por empresas como Netflix, PayPal, LinkedIn, Uber, Walmart, EBay entre otras. Una de las características más atractivas de este runtime es que se basa en JavaScript por lo que los desarrolladores front-end pueden desarrollar back-end sin necesidad de aprender un nuevo lenguaje de programación.

En esta unidad revisaremos los conceptos básicos de Internet, su funcionamiento, y el protocolo HTTP. Además definiremos que es un Framework para el desarrollo web, introduciremos que es Express, como instalarlo y cómo realizar nuestra primer aplicación con dicha herramienta. Además introduciremos los conceptos de manejo de errores y su importancia, y la modularización de nuestra aplicación.

Objetivos:

Que los participantes*:

- Repasen conocimientos de Internet necesarios para el curso.
- Incorporen los conocimientos básicos de Express.
- Aprendan a instalar Express en sus equipos.
- Comprendan la estructura básica de una aplicación en Express

Bloques temáticos*:

1. Introducción a Express
    - Conceptos básicos de Internet
    - ¿Cómo funcionan las aplicaciones Web?
    - ¿Qué es Express?
    - Instalación
    - Primer ejemplo
2. Configuración
3. Middleware
    - Concepto de middleware
    - Middleware en Express
4. Modularización
    - ¿Qué son los módulos?
    - ¿Por qué dividir en módulos?
5. Manejo de errores
    - Importancia del manejo de errores
    - Manejo de errores en Express

Consignas para el aprendizaje colaborativo

En esta Unidad los participantes se encontrarán con diferentes tipos de actividades que, en el marco de los fundamentos del MEC*, los referenciarán a tres comunidades de aprendizaje, que pondremos en funcionamiento en esta instancia de formación, a los efectos de aprovecharlas pedagógicamente:

- Los foros proactivos asociados a cada una de las unidades.
- La Web 2.0.
- Los contextos de desempeño de los participantes.

Es importante que todos los participantes realicen algunas de las actividades sugeridas y compartan en los foros los resultados obtenidos.

Además, también se propondrán reflexiones, notas especiales y vinculaciones a bibliografía y sitios web.

El carácter constructivista y colaborativo del MEC nos exige que todas las actividades realizadas por los participantes sean compartidas en los foros.

** El MEC es el modelo de E-learning colaborativo de nuestro Centro.*

Tomen nota*

Las actividades son opcionales y pueden realizarse en forma individual, pero siempre es deseable que se las realice en equipo, con la finalidad de estimular y favorecer el trabajo colaborativo y el aprendizaje entre pares. Tenga en cuenta que, si bien las actividades son opcionales, su realización es de vital importancia para el logro de los objetivos de aprendizaje de esta instancia de formación. Si su tiempo no le permite realizar todas las actividades, por lo menos realice alguna, es fundamental que lo haga. Si cada uno de los participantes realiza alguna, el foro, que es una instancia clave en este tipo de cursos, tendrá una actividad muy enriquecedora.

Asimismo, también tengan en cuenta cuando trabajen en la Web, que en ella hay de todo, cosas excelentes, muy buenas, buenas, regulares, malas y muy malas. Por eso, es necesario aplicar filtros críticos para que las investigaciones y búsquedas se encaminen a la excelencia. Si tienen dudas con alguno de los datos recolectados, no dejen de consultar al profesor-tutor. También aprovechen en el foro proactivo las opiniones de sus compañeros de curso y colegas.

**** Está página queda como está. El contenidista no le quita ni le agrega nada.***

1. Introducción a Express

Conceptos básicos de Internet

Internet es la columna vertebral de la web, la infraestructura técnica que hace la web posible. Simplificando su estructura, podríamos decir que es un conjunto de computadoras interconectadas entre sí.

Los inicios de Internet son por los años 60, cuando el departamento de defensa de los Estados Unidos de América decidió crear una red de computadoras que enfatizara la robustez y supervivencia. Incluyendo la capacidad de resistir a la pérdida de grandes partes de su red. Luego, en los años 80, evolucionó a una infraestructura pública (con el apoyo de Universidades y empresas privadas). Las tecnologías que dan soporte a Internet han evolucionado a lo largo de estos años, pero su forma de trabajo no ha cambiado demasiado. Internet es una forma de interconectar las computadoras y asegurarse que, no importa lo que pase, van a encontrar una forma de mantenerse conectadas.

Composición

Como mencionamos anteriormente, Internet está compuesto por computadoras conectadas entre sí, ahora veremos cómo es dicha conexión entre las computadoras y cómo viaja la información de un origen a un destino.

La conexión entre computadoras se realiza a través de un dispositivo llamado router, el cual se encarga que un mensaje de la computadora A a la computadora C llegue a destino. En el diagrama A1 podemos identificar a estos dispositivos con la letra R (R1, R2, R3, R4).

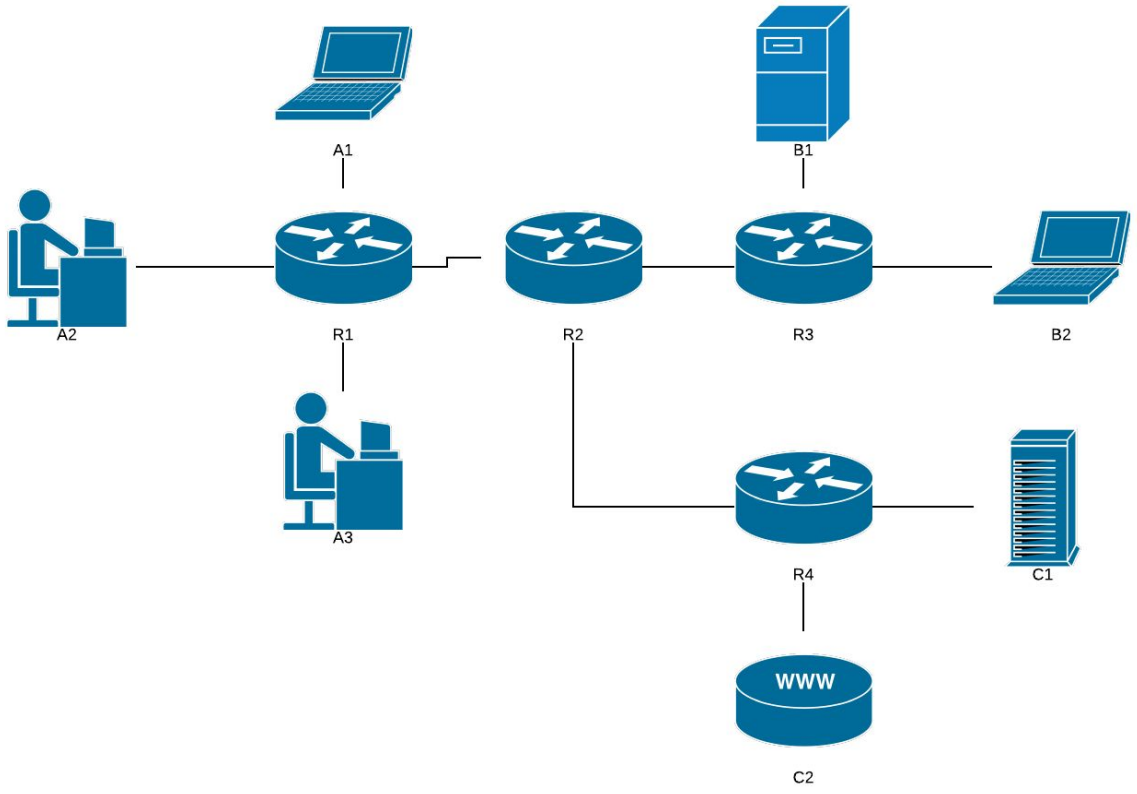


Diagrama A1

Si el equipo A2 quiere conectarse con el equipo B1:

1. El equipo A2 le indica a R1 que quiere enviar un mensaje a B1
2. R1 redirige el mensaje a R2
3. R2 redirige el mensaje a R3
4. R3 redirige el mensaje a B1
5. B1 recibe el mensaje enviado por A2

Encontrando el destino

Para que el equipo A2 pueda enviar un mensaje al equipo B1, el equipo B1 tiene que tener un identificador único (el cual permite diferenciarse de todos los demás equipos). A este identificador único, lo llamamos dirección IP, y está compuesto por 4 números separados por punto (.). Ej: 149.32.11.92

Todo equipo conectado a Internet tiene una dirección IP.

Para las computadoras es fácil manejar números, pero para nosotros es mucho más difícil poder memorizar 4 números que un nombre descriptivo. Es por eso que se crearon “alias” (otra forma de llamar a esa dirección IP) y es por un nombre (nombre de dominio). Ej:

La dirección IP 200.42.136.212 tiene el “alias” www.clarin.com (mucho más fácil de recordar por nosotros)

En Internet existen unos servicios especiales que son ejecutados en servidores los cuales se encargan de realizar la traducción de los nombres entendibles por nosotros (www.clarin.com) a su correspondiente IP (200.42.136.212) que es la que entienden las computadoras, estos servicios se llaman DNS (Domain Name Service)

Recomendamos mirar el video “How the Internet Works in 5 minutes” (Cómo funciona Internet en 5 minutos) https://youtu.be/7_LPdttKXPc

(El video se encuentra en Inglés, pero se pueden seleccionar subtítulos traducidos al Español).

Clientes y Servidores

Podemos simplificar los tipos de computadoras conectadas a Internet llamando a algunas de ellas clientes y a otras servidores.

Los clientes son aquellos dispositivos comunes que se conectan a internet y principalmente consumen información de Internet (nuestras computadoras, nuestros celulares).

Los servidores son equipos que almacenan páginas web, archivos, aplicaciones. Principalmente ofrecen información a Internet.

¿Cómo funcionan las aplicaciones web?

Cuando un dispositivo cliente solicita una página web, el servidor entrega una copia de dicha página web, la cual es descargada en el dispositivo cliente y es mostrada en el navegador web del cliente.

La comunicación entre el cliente y servidor de páginas web, se hace a través de un protocolo (convención) llamado HTTP (Hypertext Transfer Protocol). Este protocolo define un “lenguaje” de comunicación que conoce tanto el cliente como el servidor, lo cual les permite comunicarse uno con el otro. HTTP utiliza la infraestructura de Internet para realizar su comunicación.

Solicitud de una página web

1. El usuario ingresa el nombre de página web a la cual desea acceder
2. La computadora cliente se conecta con el servidor de DNS para consultar cual es la dirección IP para el nombre ingresado
3. La computadora cliente hace una petición HTTP a la dirección IP del servidor, solicitando que le responda con una copia del sitio web.
4. El servidor recibe la petición del cliente (la cual viajó a través de los routers), y en caso de poder responder a la solicitud, responde con un mensaje 200 - OK y la copia de la página web solicitada. La página web es retornada en fragmentos pequeños, llamados paquetes de datos.

5. El cliente recibe y ensambla los paquetes de datos recibidos, en una página web, y la muestra al usuario.

Servidor Web

Cuando se habla de Servidor Web, se puede hacer referencia tanto al Hardware (equipo físico), como al software (programas que corren), o ambos en su conjunto.

Un servidor Web está compuesto por:

- **Hardware** que es la computadora en la cual se alojan los archivos que componen el sitio web, así como también las aplicaciones necesarias.
- **Software** son los programas que son utilizados que permiten entregar la información al cliente. Como mínimos se necesita de una aplicación servidora de HTTP (que es la encargada de entender el protocolo y comunicarse con el cliente)

El software del Servidor Web, la aplicación que se comunica con el cliente, puede ser del tipo estático o dinámico.

- **Servidor web estático**, es el hardware y el software de servidor web HTTP. Se llama estático porque el mismo retorna los archivos tal cual están en el servidor.
- **Servidor web dinámico**, consiste en un servidor web estático más cierto software que permite modificar y generar respuestas dinámicamente al cliente. Pudiendo tomar archivos que están en el servidor, modificarlos y luego retornarlos al cliente. Generalmente, el software, está compuesto por un servidor de aplicación y base de datos.

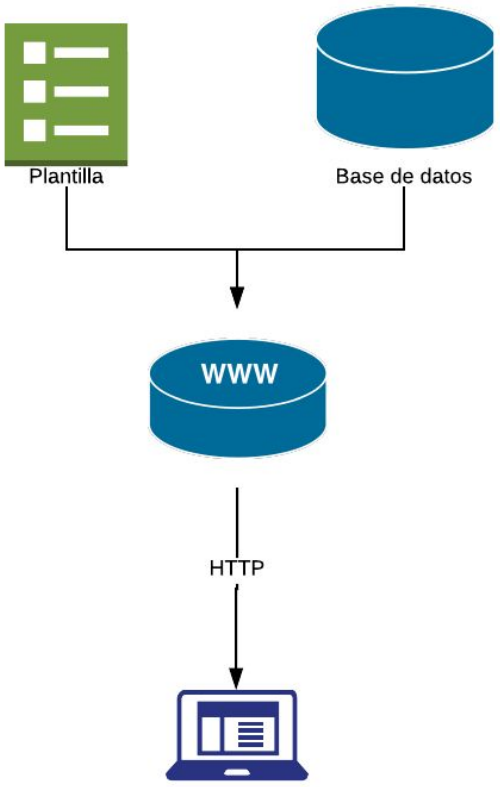


Diagrama A2

El Diagrama A2 es un ejemplo de Servidor Web Dinámico, en el cual tiene una plantilla con la estructura del contenido, y la información es completada con los datos obtenidos de una base de datos. El resultado de "unir" la plantilla con los datos, es una página web lista para retornar al cliente.

Consideraciones de HTTP

- Es el lenguaje con el cual se comunica el cliente y servidor web
- Todos los comandos HTTP son textos legibles por el humano
- Los comandos son a una URL, nombre del recurso al cual se le realiza la petición (Ej: www.pagina12.com/noticia1.html para obtener la noticia1.html del servidor www.pagina12.com)
- El protocolo HTTP no mantiene estado, ni el cliente, ni el servidor recuerdan las comunicaciones previas que tuvieron entre sí. Cada comunicación que se realiza

es siempre como una nueva (más adelante veremos como se puede mantener información).

- Para mantener el estado entre peticiones, el servidor le entrega al cliente un identificador único, el cual envía el cliente en cada petición que le realice al servidor. El servidor, cuando recibe un identificador, verifica si dicho identificador existe y aún es válido (no caducado) y en caso que así sea puede identificar quien es el cliente y recuperar información de peticiones anteriores.

Métodos HTTP más comunes

Todas las peticiones que hace un cliente a una URL del servidor, son acompañados por un verbo (GET, POST, PUT, DELETE) que le indica al servidor que operación el cliente desea realizar sobre ese recurso. Por ejemplo GET http://www.google.com le indica al servidor de Google, que el cliente desea solicitar (GET) dicha página web.

En cambio cuando completamos un formulario, por ejemplo de contacto, en un sitio web (Ej: http://prueba.com/mi_formulario.html) y enviamos dicho formulario (presionamos el botón enviar). El cliente (nuestra computadora) le indica al servidor que va a enviar información, para ello realiza un POST http://prueba.com/mi_formulario.html.

Los métodos (verbos) más utilizados son los siguientes.

Método	Uso y Descripción
GET	Utilizado por el cliente para pedir información al servidor
POST	Utilizado por el cliente para enviar información al servidor
PUT	Utilizado por el cliente para enviar información al servidor (generalmente utilizado para reemplazar un recurso)
DELETE	Utilizado por el cliente para solicitar borrar un recurso en el servidor

¿Qué es Express?

Infraestructura de Aplicaciones Web

Es un software que hacen más fácil escribir, mantener y escalar aplicaciones web. Proveen de herramientas y bibliotecas que simplifican tareas habituales de desarrollo web. Generalmente incluyen:

- Interacción con base de datos
- Soporte de sesiones (poder mantener información de un cliente a través de las peticiones HTTP)
- Autenticación y autorización de usuarios
- Formateo de la respuesta
- Uso de plantillas
- Redirección de una petición a una URL específica al componente que maneja esa petición.

Ventajas de usar una Infraestructura de Aplicaciones Web

- Permiten trabajar con las peticiones HTTP realizadas al servidor, así como también las respuestas que el mismo retorna al cliente, de manera sencilla. Las peticiones al servidor son llamadas “requests” y las respuestas del servidor hacia el cliente “responses”. Una Infraestructura de Aplicaciones Web, abstrae la complejidad del método HTTP, lo cual simplifica el desarrollo, permitiendo trabajar de manera rápida y fácil con las peticiones del cliente.
- Direccionar la petición al manejador correspondiente. Cada petición de una URL al servidor, es recibido por la Infraestructura de Aplicaciones Web, y esta deriva la petición al código que el desarrollador le haya indicado (generalmente cada URL tiene un manejador asociado). El manejador, desde el punto de vista del desarrollador, es una función que él realiza, a la cual le asigna una URL que cuando es llamada se ejecuta la función indicada.
- Simplifican el acceso a la información enviada por el cliente. HTTP implementa diferentes formas de envío de información, algunas más sencillas y otras más complejas. Una Infraestructura de Aplicaciones Web permite manejar esos datos recibidos de una manera homogénea, fácil y rápida.
- Sistemas de plantillas fáciles de usar. Se genera una plantilla en la cual se le dejan “espacios” que luego se completarán con datos al momento de generación de la página web.

Express

Es una infraestructura de aplicaciones web Node.js mínima y flexible que proporciona un conjunto sólido de características para las aplicaciones web y móviles.
La primer versión de Express fue lanzada al mercado en Noviembre de 2010.

Es un paquete de NodeJS, así como los miles de paquetes disponibles por medio de Node Package Manager (NPM).

Express es muy popular, principalmente porque facilita el traspaso de desarrolladores JavaScript del lado cliente, al lado servidor. Además de ser una infraestructura muy eficiente para el manejo de recursos (ya que corre por encima de NodeJS)

Al ser una Infraestructura minimalista no incorpora todos los componentes que podemos necesitar (por ejemplo, de acceso a datos, soporte de sesiones). Pero existen muchos componentes que pueden ser integrados con Express para proveer dicha funcionalidad, permitiendo al desarrollador tener un abanico más amplio de componentes a utilizar (no estando atado a una única forma de hacer las cosas).

A su vez, se la conoce como una infraestructura no dogmática, que no impone una forma “correcta” de hacer las cosas. No obliga a dividir la aplicación en diferentes archivos, o una estructura específica, lo cual implicará que nosotros debamos tomar dicha decisión.

Ventajas de Express

- Aplicaciones Web: dispone de características robustas para el desarrollo de aplicaciones Web
- API: dispone de varias utilidades HTTP que permiten crear API robustas de manera fácil y rápida
- Performance: es una pequeña capa con mucha funcionalidad que permite una respuesta rápida

Algunas empresas que utilizan Express

- IBM
- Accenture
- Fox Sports
- Mulesoft
- Uber

Instalación

La instalación de Express, como cualquier otro paquete de NodeJS, se realiza por linea de comandos.

Tomaremos como ejemplo la aplicación hola curso que mostrará un mensaje de bienvenida al curso en Express.

Ingresamos a la línea de comando de nuestro sistema operativo (para Windows command line, y terminal para Linux o MAC)

Nos dirigimos al directorio en el cual deseamos crear nuestra aplicación (utilizar el comando cd para cambiar de directorio)

```
mkdir hola-curso
cd hola-curso
```

mkdir crea el directorio con el nombre “hola-curso”

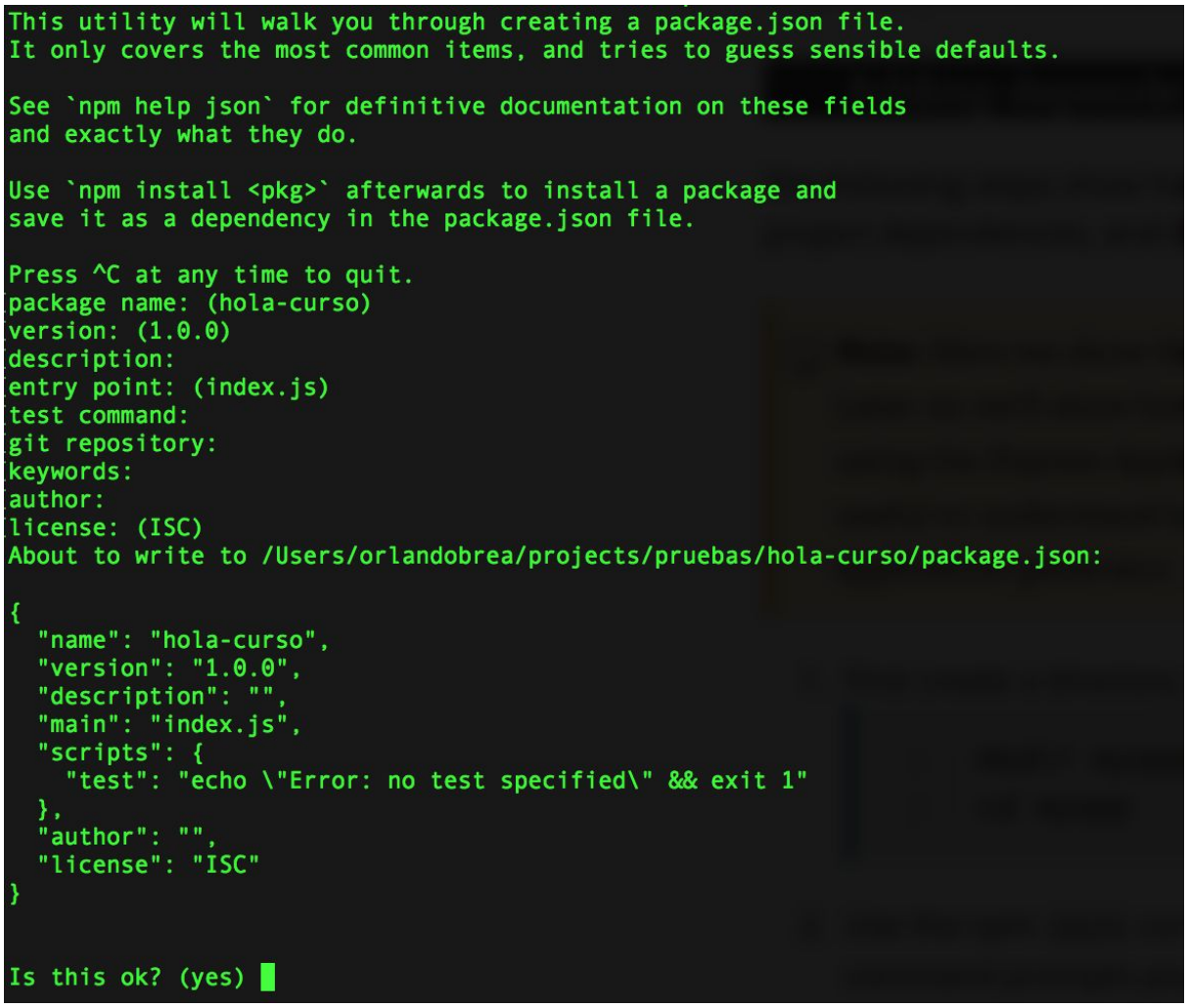
cd (change directory / cambiar directorio) nos posiciona dentro del directorio que acabamos de crear.

```
npm init
```

Crear un archivo *package.json* dentro del directorio de nuestra primer aplicación.

El archivo *package.json* es el encargado de mantener información de nuestra aplicación y tambíen las dependencias de la misma (paquetes externos que necesita nuestra aplicación para operar).

Al ejecutar el comando npm init, nos realizará una serie de preguntas destinadas a crear el archivo previamente mencionado.



Las preguntas que nos realiza el comando son las siguientes (pueden llegar a variar dependiendo de la versión).

package-name: el nombre que queremos asignarle a nuestra aplicación. De manera predeterminada nos sugiere el nombre de la carpeta en la cual ejecutamos el comando. Dejaremos la opción que nos indica.

version: la versión de nuestra aplicación. Indicaremos cual es la versión inicial que estamos realizando de esta aplicación. Dejaremos la opción que nos indica.

description: un resúmen de las características de nuestra aplicación. En aplicaciones reales, debemos intentar ser concisos y precisos para que cualquier otro desarrollador pueda fácilmente entender que es lo que hace nuestra aplicación.

entry point: punto de entrada en caso de que otra aplicación haga uso de la nuestra. Como la aplicación que estamos desarrollando no será incluída, ni utilizada por otras aplicaciones, será un campo que no tendrá relevancia.

test command: comando que se ejecutará cuando se quieran realizar tests a nuestra aplicación. Cuando se ejecuta el comando '*npm test*' se ejecutará el comando que aquí especifiquemos. Podemos dejar este campo con el código sugerido.

git repository: dirección del repositorio GIT en el cual se encuentra alojada nuestra aplicación. En caso de utilizar la herramienta de versionado GIT, podemos indicar la dirección del repositorio aquí. Para este ejemplo no utilizaremos ningún repositorio GIT.

En caso de nunca haber utilizado una herramienta de control de versiones recomendamos acceder a la documentación de GIT (https://git-scm.com/doc) en la cual existen diferentes recursos para poder aprender sobre esta herramienta.

keywords: palabras claves por las cuales deseamos que encuentren nuestra aplicación. En este ejemplo, no estaremos publicando nuestra aplicación para que pueda ser utilizada por otros desarrolladores, por lo que este campo puede quedar en blanco.

author: ingresamos nuestro nombre.

license: en caso de publicar nuestra aplicación, qué tipo de licencia deseamos que se aplique.

```
{
  "name": "hola-curso",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
```

```
  "author": "",
  "license": "ISC"
}
```

Archivo package.json generado

Hasta ahora solo hemos creado el archivo package.json con el contenido básico para cualquier aplicación de NodeJS. Para indicar que vamos a utilizar Express en nuestra aplicación como infraestructura de aplicaciones web, debemos ejecutar el siguiente comando

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

Le indicamos al administrador de paquetes npm, que deseamos utilizar Express en nuestra aplicación. La opción --save modificará nuestro archivo package.json incluyendo la dependencia de Express.

Al ejecutar el comando npm realiza las siguientes acciones:

1. Descarga el paquete express de Internet
2. Si el paquete tiene dependencias, también descarga las dependencias
3. Modifica el archivo package.json, incluyendo el paquete express como dependencia de nuestro proyecto.

```
{
  "name": "hola-curso",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^4.16.3"
  }
}
```

Archivo package.json modificado por el comando npm install express --save

Ya tenemos toda la estructura necesaria para comenzar a desarrollar nuestra aplicación (archivo package.json con información de nuestra aplicación y dependencias). A continuación realizaremos nuestra primer aplicación web dinámica.

Primer ejemplo

Para realizar nuestra primer aplicación web con express, creamos el archivo index.js con el código que se encuentra a continuación

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

app.get('/', function(request, response) {
  response.send('Bienvenido al curso de NodeJS nivel Intermedio!');
});

app.listen(3000, function() {
  console.log('Iniciando la aplicación en http://localhost:3000 <-
copia esta URL en tu navegador');
});
```

Archivo index.js

Ahora veremos en mayor detalle cada línea del código

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

Se incluye el módulo de Express en la variable express. Cuando trabajamos con módulos, debemos incluir los mismos en los archivos en los cuales utilizaremos el mismo.

```
var app = express();
```

Se crea la variable app que es un objeto de la aplicación de Express, por medio de la cual accederemos a la aplicación para definir configuración, rutas, etc. El nombre de la variable app no es obligatorio, pero por convención se utiliza este nombre de variable.

```
app.get('/', function(request, response) {
..
});
```

El código lo podemos descomponer en dos partes, la llamada a un método del objeto express (app) con 2 parámetros (ruta, función callback)

```
app.get('/', callback1);
```

Y la función callback (asignándole el nombre callback1)

```
function callback1(request, response) {
..
}
```

Ahora veremos las dos partes por separado para poder analizar mejor su comportamiento.

```
app.get('/', callback1);
```

El objeto app (Express) tiene varios métodos que podemos acceder, los cuales vimos anteriormente (GET, POST, PUT, DELETE). En este caso le estamos indicando a Express que cuando el cliente realice una petición GET a / llame a la callback que le indicamos.

Express traduce el código que nosotros le indicamos en una tabla similar a

Condición	Código a ejecutar si se cumple
Si el cliente pide (GET) la ruta / (http://localhost)	Ejecutar el código especificado en callback1

```
function callback1(request, response) {
..
}
```

callback1 es la función que se llamará en caso que se cumpla la condición mencionada anteriormente (que el cliente realice un GET /) a nuestra aplicación.

La función recibe 2 parámetros

- **request:** es el objeto petición, el cual contiene la solicitud que realizó el cliente a nuestra aplicación
- **response:** es el objeto respuesta, el cual contendrá la respuesta que enviaremos al cliente.

Reemplazando callback1 por el contenido de la función, volvemos al código orignal del ejemplo.

```
app.get('/', function(request, response) {
..
});
```

Analizando el contenido de la función

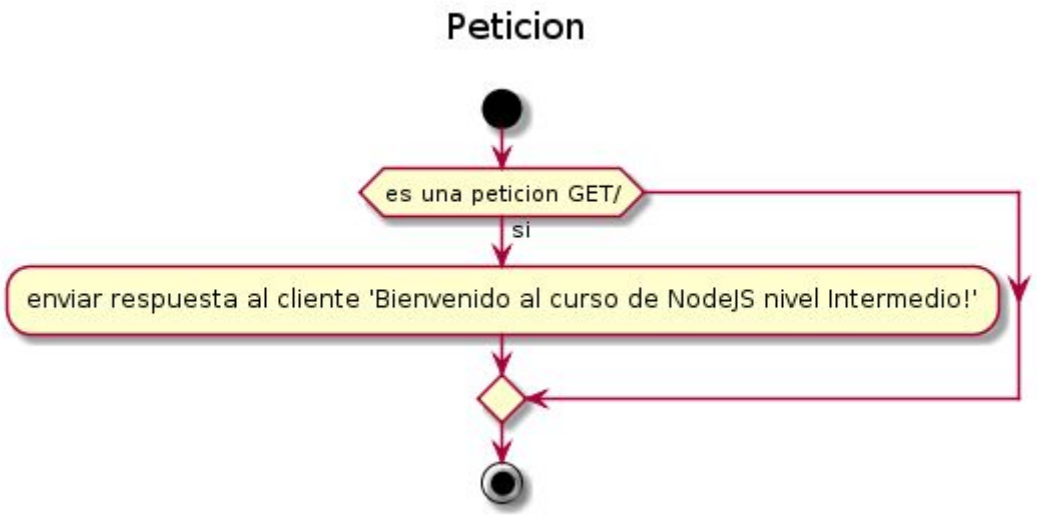
```
  response.send('Bienvenido al curso de NodeJS nivel Intermedio!');
```

Se llama al método send, del objeto response. Esto provoca que el texto que pasamos como parámetro a la función, sea enviado al cliente.

Repasemos el bloque completo de código para agrupar los conceptos vertidos previamente.

```
app.get('/', function(request, response) {
  response.send('Bienvenido al curso de NodeJS nivel Intermedio!');
});
```

El flujo de ejecución del código lo podemos graficar con el siguiente diagrama.



Si el cliente hace una petición GET a / en nuestra aplicación, se ejecutará el código de la función. En caso que no se realice una petición, o que la misma no sea GET o sea a otra ruta, NO se ejecutará el código de la función.

```
app.listen(3000, function() {
  console.log('Iniciando la aplicación en http://localhost:3000 <-
copia esta URL en tu navegador');
});
```

El objeto Express, tiene un método listen, que es el encargado de “iniciar” la aplicación. Recibe dos parámetros:

- **puerto** en el cual “escucha” nuestra aplicación (en el ejemplo es el puerto 3000)
- **función** que se ejecutará al iniciar la aplicación. En este caso es una función que solo mostrará por la consola del servidor (cuando se inicie) el mensaje especificado (‘Iniciando la aplicación en http://localhost:3000 <- copia esta URL en tu navegador’).

Ejecutar la aplicación

Para ejecutar la aplicación solo debemos, por línea de comando, ejecutar

```
node index.js
```

Y en la misma línea de comando veremos el siguiente mensaje

```
Iniciando la aplicación en http://localhost:3000 <- copia esta URL en
tu navegador
```

Ingresamos en un navegador (Chrome, Firefox, Safari, Opera, Internet Explorer) la URL http://localhost:3000

Y veremos

2. Configuración

Express permite configurar algunas de sus características, a continuación mostraremos las más relevantes.

A través del comando `app.set(<propiedad>, <valor>)` podemos asignar los parámetros de configuración a la infraestructura Express.

Configuración	Tipo de valor	Detalle
app.set('case sensitive routing', <valor>);	Booleano	Indica a Express si las rutas a las cuales responde son sensibles de mayúsculas/minúsculas. Ej: si debe tratar /ejemplo diferente que /Ejemplo
app.set('env', <valor>);	Cadena	Indica si la aplicación debe correr en modo desarrollo (development) o en modo productivo (production). Mientras estemos desarrollando la aplicación utilizaremos modo desarrollo, y utilizaremos modo productivo cuando publiquemos nuestra aplicación en Internet.
app.set('strict routing', <valor>);	Booleano	Indica si las rutas deben ser estrictas en su definición. Ej: si /ejemplo es una ruta diferente a /ejemplo/
app.set('views', <valor>);	Cadena	Indica el directorio en donde se alojan las plantillas de la aplicación
app.set('view engine', <valor>');	Cadena	El motor de templates que se utilizará para transformar las plantillas en archivos finales. Para el curso utilizaremos Handlebars (sobre el cual profundizaremos en la próxima unidad)
app.set('x-powered-by', <valor>);	Boolean	Indica si debe responder con el encabezado HTTP '*X-Powered-By:*

		Express'. Es un atributo que recomendamos asignarlo a false, para no indicar al cliente que tipo de Infraestructura estamos utilizando.

3. Middleware

Concepto de middleware

El término middleware es ampliamente utilizado en diferentes tecnologías, aunque teniendo diferentes definiciones para cada una de ellas. Cuando hablamos de middleware estamos hablando de una capa intermedia (algo entre 2 cosas, que generalmente hace de nexo entre ellas).

Cuando hablemos de middleware en express, estaremos hablando de un componente que está diseñado para modificar una petición o respuesta HTTP, pero que generalmente no da la respuesta. Está diseñado para ser encadenado (ejecución uno después del otro), para formar una tubería de cambios de comportamiento durante el procesamiento de una petición.

Middleware en Express

Son funciones que tienen acceso al objeto de petición (request), el de respuesta (response), y a la función que le sucede (next) en el ciclo de ejecución de la aplicación. La función siguiente (next), es una función que, cuando es llamada, ejecuta el siguiente middleware.

Estas funciones son ejecutadas previo a las funciones específicas para una ruta.

Las funciones middleware pueden realizar las siguientes tareas:

- Ejecutar cualquier código
- Realizar cambios en el objeto petición (request) y en objeto respuesta (response)
- Finalizar el ciclo de una petición
- Llamar a la siguiente función middleware

Al finalizar cada función middleware se debe llamar a next() para que continúe la ejecución de la siguiente función middleware o ruta. En caso de no llamarse a next(), no se ejecutará ninguna otra función.

Veamos un ejemplo

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

var myLogger = function (req, res, next) {
  console.log('-Ejecutando Logger-');
  next();
}

app.use(myLogger);

app.get('/', function (req, res) {
  res.send('Bienvenido al Curso!');
})

app.listen(3000);
```

La salida del código es la siguiente:

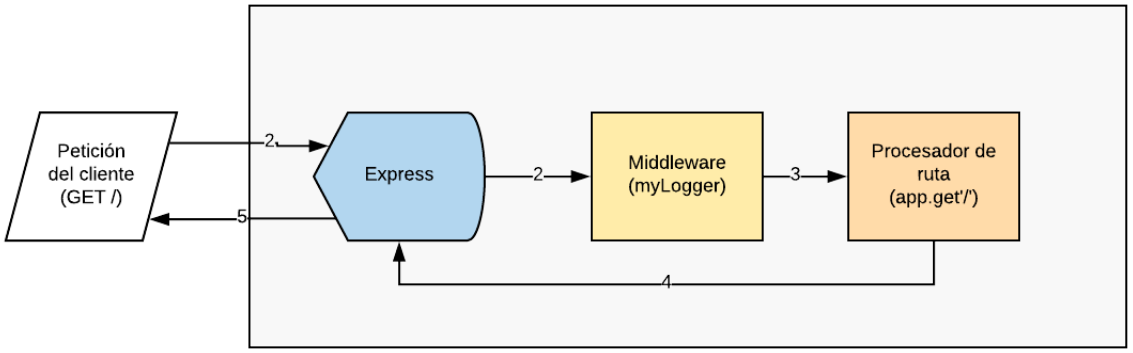
Salida por consola	Respuesta al cliente (HTTP)
01 -Ejecutando Logger- 02	01 02 Bienvenido al Curso!

Primero mostrando por consola el mensaje -Ejecutando Logger- y luego respondiendo al cliente con “Bienvenido al Curso!”

Como podemos ver en el ejemplo, la única ruta que concuerda con la petición de un GET / es la de la línea 11 (app.get….), pero igualmente se ejecuta el código de la función guardada en la variable myLogger. Esto es porque myLogger es un middleware, un

código que se ejecuta antes de ejecutarse el código de la ruta, y luego se ejecuta el código de la ruta específica.

Secuencia de ejecución



Analicemos el código del middleware

```
var myLogger = function (request, response, next) {
  console.log('-Ejecutando Logger-');
  next();
}

app.use(myLogger);
```

Las funciones middleware reciben 3 parámetros

- request: petición que realiza el cliente
- response: respuesta que se le entrega al cliente
- next: próxima función de la cadena de ejecución (en este caso, como hay un único middleware, la próxima función es la que procesa la ruta app.get('/', …

```
var myLogger = function (request, response, next) {
...
}
```

La función middleware, es guardada en una variable (myLogger), para luego poder utilizarla más adelante.

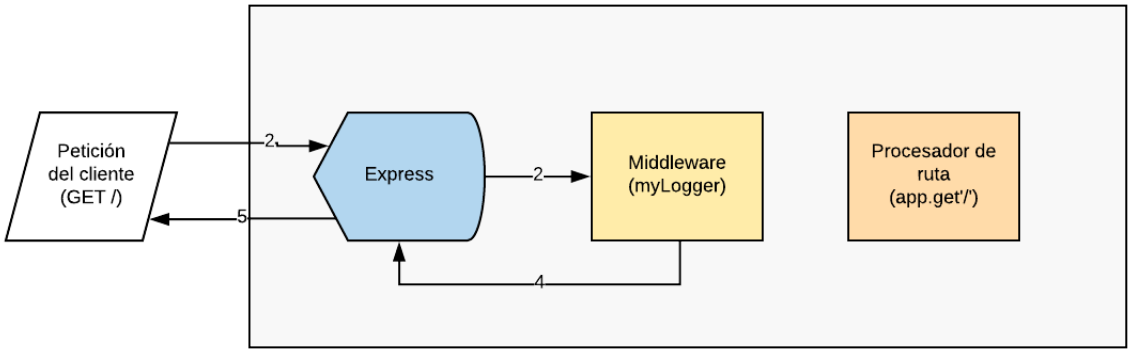
```
console.log('-Ejecutando Logger-');
next();
```

En la primer línea solo muestra por consola el mensaje -Ejecutando Logger-

En la línea siguiente, le indica a Express, que luego de ejecutar esta función middleware, ejecute la siguiente función. En caso de no incluir el next(), no se ejecutará ninguna otra función de la cadena de ejecución, siendo esta la última función en ejecutarse. La próxima función en la cadena, puede ser otro middleware, o una ruta.

Es de suma importancia incluir el next() en caso que deseemos que siga la cadena de ejecución.

En caso de no ejecutar el next() en el ejemplo anterior, la secuencia de ejecución sería la del diagrama siguiente.



En el cual podemos apreciar que la cadena de ejecución se termina en el middleware y nunca es llamado el código que procesa la petición a la ruta específica.

Un ejemplo de caso en el cual NO deseamos que continúe la ejecución, puede ser cuando realizamos una validación si el usuario ha ingresado en el sistema, y en caso que no haya ingresado, lo podemos (desde el mismo middleware) redireccionar a la página de

login. En este caso, no deseamos pasar la ejecución a la próxima función (que puede ser la ruta que responda

```
app.use(myLogger);
```

El middleware no será ejecutado, a menos que le indiquemos a Express que incluya dicho middleware en la cadena de ejecución. Para incluirlo en la cadena de ejecución, utilizamos el método use del objeto Express.

4. Modularización

¿Que son lo módulos?

Un módulo es una biblioteca o código JavaScript que puede ser importado (incorporado) en otro código JavaScript, por medio de la función require() de NodeJS.

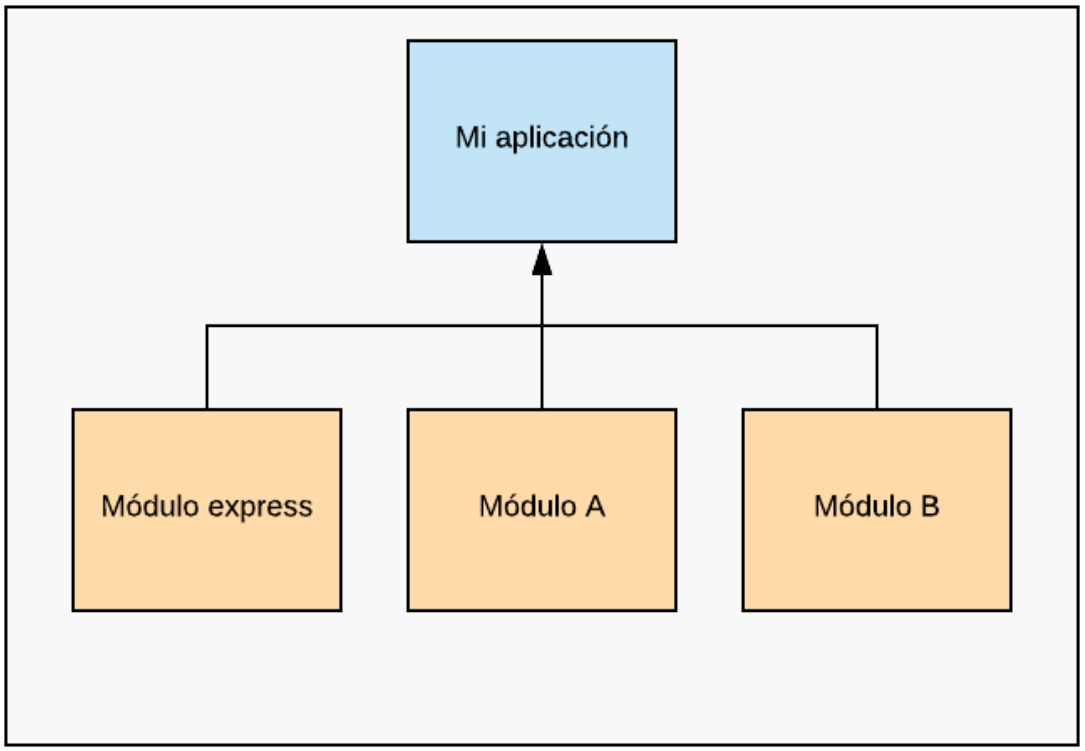
Los módulos permiten dividir la complejidad de la aplicación en porciones más fáciles de administrar, y a su vez poder reutilizar el código que hemos dividido.

En los ejemplos que hemos planteado en esta unidad, hemos estado trabajando con el módulo de Express.

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

En el código podemos ver que importamos (incluímos) el módulo de Express en nuestra aplicación por medio de la función `require()`, y el módulo incluído es guardado en la variable `var express`.

De esta forma, podemos dividir nuestra aplicación en diferentes módulos, por ejemplo



En este ejemplo, nuestra aplicación hace uso de 3 módulos Express, A, y B.

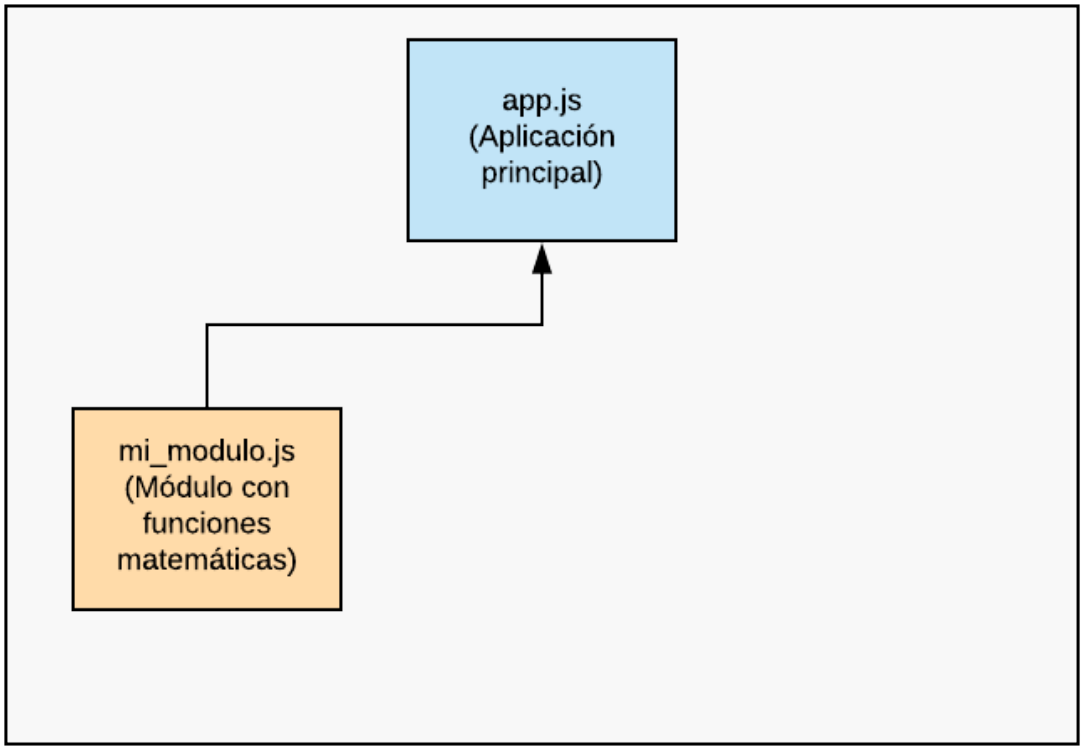
¿Por qué dividir en módulos?

La división de nuestro código en módulos permite:

- Poder dividir la complejidad de la aplicación en partes más fáciles de manejar
- Poder reutilizar código (cualquier módulo que generemos, luego lo podemos usar en otros proyectos).
- El mantenimiento de la aplicación es más fácil
- Podemos seguir la división propuesta en muchas infraestructuras web, de dividir la aplicación en lógica de comunicación con el usuario, lógica de negocio, lógica de comunicación con la base de datos (utilizando un módulo para cada una de ellas)

Ejemplo de uso de módulos

Vamos a realizar una aplicación principal (app.js) y un módulo que tendrá algunas funciones matemáticas que utiliza la aplicación principal.




```
exports.sumar = function(a,b) {
    return a+b;
}

exports.multiplicar = function(a,b) {
    return a*b;
}
```

Archivo mi_modulo.js

Por medio de exports, podemos exportar las funciones del módulo. En este caso estamos exportando 2 funciones: sumar y multiplicar. Ambas funciones reciben 2 parámetros (a y b).

```
var funciones = require('./mi_modulo');

console.log(funciones.sumar(2,8));

console.log(funciones.multiplicar(2,8));
```

Archivo app.js

En el archivo app.js importamos el módulo creado anteriormente (mi_modulo.js) el cual se encuentra en el mismo directorio que este archivo. Asignamos el módulo importado a la variable funciones (la cual referencia a exports de mi_modulo.js). Luego podemos llamar a las funciones que hemos exportado en mi_modulo.js (sumar y multiplicar)

5. Manejo de errores

Importancia del manejo de errores

Toda aplicación que desarrollemos es propensa a presentar un error en algún momento. Los errores se pueden producir por escribir un código que contenga errores, un ingreso del usuario erróneo o no esperado, un problema con la conexión con la base de datos, un problema con conexión con un servicio externo, etc.

Express, de manera standard, cuando encuentra un error en la aplicación que no es manejado por el código que nosotros desarrollamos, aborta la ejecución del programa y termina el mismo. En caso de no realizar un manejo de error, nuestra aplicación podría "cerrarse", por lo que nadie podría seguir utilizando la aplicación hasta que volvamos a iniciar la aplicación.

Un manejo apropiado de errores implica:

- Manejar los errores que se sucedan en nuestra aplicación
- Brindar un mensaje entendible al usuario en caso de producirse un error
- Que la aplicación pueda continuar funcionando a pesar de un error
- Guardar en un log el error, para luego poder detectar el origen y corregir el código

Manejo de errores en Express

Express ya viene con la tecnología necesaria para manejar errores que se produzcan en nuestra aplicación.

Solo debemos incorporar el siguiente código (adaptándolo a nuestros requerimientos particulares) dentro de nuestro archivo principal de express (en el ejemplo que estamos trabajando durante esta unidad, sería el archivo index.js)

```
app.use(function(err, req, res, next) {
  res.status(err.status || 500);
  res.send(err.message);
});
```

Parte del archivo index.js

Como podemos ver, el manejo de errores se realiza por medio de un middleware que recibe 4 parámetros:

- err que contiene el error producido
- req que contiene la petición del cliente
- res que es donde se da respuesta al cliente
- next qué es el próximo manejador de la cadena

En el código solamente estamos retornan una respuesta de error al cliente (HTTP Status 500, que es un estado de error), y el mensaje del error que se ha producido.

A los fines de centrarnos solamente en la función de manejo de errores, hemos simplificado el middleware a la expresión mínima necesaria.

En las aplicaciones que desarrollemos, el manejo de error deberá ser más complejo, guardando el mensaje de error en un archivo de log, y brindando una respuesta al usuario que pueda ser entendida por el.

Bibliografía utilizada y sugerida

Introducción a Express/Node. (n.d.). Recuperado de https://developer.mozilla.org/es/docs/Learn/Server-side/Express_Nodejs/Introduction

Express - Node.js web application framework. (n.d.). Recuperado de http://expressjs.com

Lo que vimos:

Conceptos básicos de Internet

Introducción a Express y nuestra primer aplicación

Modularización

Manejo de errores en Express

Lo que viene:

Motores de templates (View Engines)

Handlebars

Ruteo