

Clase 2

Webservices con express



Temario

1. Qué es Express
2. Instalación
3. Creación de servidores
- 4. Middlewares**
5. Enrutamiento
6. Knex/Sequelize
7. Sesiones, Cookies y JWT

Middlewares

Tal y como se menciona en la [web oficial](#), Express es un framework web de enrutamiento y middleware. El manejo de middleware es una de las principales funcionalidades de Express.

Cualquier función que se ejecute entre el ciclo request-response de una solicitud HTTP se conoce como una función de middleware.

Las funciones de middleware son 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/respuestas de la aplicación. La siguiente función de middleware se denota normalmente con una variable denominada next.

Las funciones middleware pueden realizar las siguientes tareas:

- Ejecutar cualquier código.
- Realizar cambios en la solicitud y los objetos de respuesta.
- Finalizar el ciclo de solicitud/respuesta. (Si la función de middleware actual no finaliza el ciclo de solicitud/respuestas, debe invocar next() para pasar el control a la siguiente función de middleware. De lo contrario, la solicitud quedará colgada.)
- Invocar la siguiente función de middleware en la pila.

Middlewares de aplicación 1/3

Enlace el middleware de nivel de aplicación a una instancia del objeto de aplicación utilizando las funciones **app.use()** y **app.METHOD()**, donde METHOD es el método HTTP de la solicitud que maneja la función de middleware (por ejemplo, GET, PUT o POST) en minúsculas.

```
// La función se ejecuta cada vez que la aplicación recibe una
solicitud.
app.use((req, res, next) => {
  console.log("Time:", Date.now());
  next();
});
// La función se ejecuta para cualquier tipo de solicitud HTTP en ruta
/user/:id.
app.use("/user/:id", (req, res, next) => {
  console.log("Request Type:", req.method);
  next();
});
// La función maneja las solicitudes GET a la ruta /user/:id.
app.get("/user/:id", (req, res, next) => {
  res.send("USER");
});
```

```
// Subpila de middleware que imprime información
de solicitud para cualquier tipo de solicitud HTTP
en la ruta /user/:id.
app.use("/user/:id",
  (req, res, next) => {
    console.log("Request URL:", req.originalUrl);
    next();
  },
  (req, res, next) => {
    console.log("Request Type composed:",
req.method);
    next();
  }
);
```

Middlewares de aplicación 2/3

Los manejadores de rutas permiten definir varias rutas para una vía de acceso. El ejemplo siguiente define dos rutas para las solicitudes GET a la vía de acceso `/book/:id`. La segunda ruta no dará ningún problema, pero nunca se invocará, ya que la primera ruta finaliza el ciclo de solicitud/respuestas.

```
// Este ejemplo muestra una subpila de middleware que maneja
// solicitudes GET a la ruta /user/:id.
app.get("/book/:id",
  (req, res, next) => {
    console.log("BOOK ID:", req.params.id);
    next();
  },
  (req, res, next) => {
    res.send("Book info");
  }
);
// Manejador para la ruta / book/:id que imprime el book ID
// Este middleware nunca se ejecutará
app.get("/book/:id", (req, res, next) => {
  res.end(req.params.id);
});
```

Middleware de aplicación 3/3

Para omitir el resto de las funciones de middleware de una pila de middleware de enrutador, invoque `next('route')` para pasar el control a la siguiente ruta. NOTA: `next('route')` sólo funcionará en las funciones de middleware que se hayan cargado utilizando las funciones `app.METHOD()` o `router.METHOD()`.

```
// Subpila de middleware que maneja solicitudes GET a la ruta / student/:id.
app.get("/student/:id",
  (req, res, next) => {
    // Si el student ID es 0, salta a la siguiente ruta 'special'
    if (req.params.id == 0) next("route");
    // en otro caso pasa el control al siguiente middleware 'regular'
    else next(); //
  },
  (req, res, next) => {
    // 'renderiza' una página regular
    res.send("regular");
  }
);
// manejador para la ruta / student/:id que 'renderiza' una página especial
app.get("/student/:id", (req, res, next) => {
  res.send("special");
});
```

Middlewares incorporados

Desde que Express ya no depende de [Connect](#), muchas de las funciones middlewares que venían incluidas en Express se han separado en módulos diferentes. Consulta la [lista de funciones middleware](#).

Actualmente existen varios middlewares incorporados en Express, entre ellos tenemos:

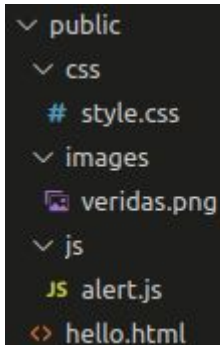
- [express.static\(\)](#): Sirve archivos estáticos como imágenes, archivos CSS y Javascript.
- [express.json\(\)](#): Convierte el cuerpo de una petición con este tipo de contenido ("Content-Type: application/json") en un objeto json javascript.
- [express.Router\(\)](#): Función para crear nuevas rutas.
- [express.text\(\)](#): Convierte el cuerpo de una petición con este tipo de contenido ("Content-Type: text/plain") en un string javascript.
- [express.urlencoded\(\)](#): Convierte el cuerpo de una petición con este tipo de contenido ("Content-Type: application/x-www-form-urlencoded") en un objeto javascript.

Middlewares incorporados:

express.static

Se puede tener más de un directorio de estáticos para cada aplicación. [Más info aquí.](#)

```
app.use('/static', express.static(__dirname + '/public'))
```



```
public
├── css
│   └── style.css
├── images
│   └── veridas.png
├── js
│   └── alert.js
└── hello.html
```

```
/* style.css */
body {
  background-color:
blue;
  color: white;
  font-size: 30px;
}
```

```
// alert.js
function clickMe () {
  alert ("Ejemplo de
estático javascript" );
}
```

```
<!-- public/hello.html -->
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width,
initial-scale=1.0">
  <link rel="stylesheet" href="/static/css/style.css" >
  <title>Sirviendo Estáticos</ title>
</head>
<body>
  <p>Hola, estoy sirviendo estáticos</ p>
  
  <br />
  <button onclick="clickMe()">Click Me!</ button>
  <script language="javascript"
src="/static/js/alert.js" ></script>
</body>
</html>
```


Middlewares incorporados: express.json, express.text, express.urlencoded

Convierten los datos de una petición de cada tipo de dato en un cuerpo ('body') manejable en javascript.

```
// Declaro los middlewares que quiero usar
app.use(express.json());
app.use(express.text());
app.use(express.urlencoded({extended: false}));
// Defino la ruta que se llamará cuando se reciba
una petición HTTP POST
// en la dirección '/user'
app.post('/user', (req, res) => {
  // Imprime por consola el tipo del body y el body
  parseado
  console.log(typeof(req.body), req.body);
  res.end();
});
```

```
# Ejecutar en la terminal

curl -v -POST http://localhost:3000/user -H "content-type:
application/json" -d '{"title": "Express json"}'

curl -v -POST http://localhost:3000/user -H "content-type:
text/plain" -d '{"title": "Express text"}'

curl -v -POST http://localhost:3000/user -H "content-type:
application/x-www-form-urlencoded" --data-urlencode 'title=Express
urlencode'
```

Middlewares incorporados: express.Router

Una instancia Router es un sistema middleware y enrutamiento completo, por este motivo, a menudo se conoce como una “miniaplicación”. En ella se puede declarar los mismos middlewares que a nivel de aplicación.

```
// routes/task.js
const express = require("express");
let router = express.Router();
// La función se ejecuta cada vez que la aplicación recibe una
solicitud.
router.use((req, res, next) => {
  console.log("Time:", Date.now());
  next();
});
// La función se ejecuta para cualquier tipo de solicitud HTTP
en ruta /:id.
router.use("/:id", (req, res, next) => {
  console.log("Request Type:", req.method);
  next();
});
// La función maneja las solicitudes GET a la ruta /:id.
router.get("/:id", (req, res, next) => {
  res.send("ID: " + req.params.id);
});

module.exports = router;
```

```
// app.js
const express = require("express");
// Cargo el módulo router task
const task_router = require("./routes/task");
const app = express();
const port = 3000;
// Cargo el router de task en mi aplicación
app.use('/task', task_router);
// Creo el servidor en el puerto ${port}
app.listen(port, () => {
  // Se escribe la URL para el acceso al servidor
  console.log(`Example server listening on
http://localhost: ${port}`);
});
```

Middlewares de terceros

Los middlewares de terceros son utilizados para añadir funcionalidad a las aplicaciones Express.

Consulta la [lista de las funciones de middleware de terceros](#) que más se utilizan con Express.

El siguiente ejemplo ilustra la instalación y carga del middleware de análisis de cookies.

```
npm install cookie-parser --save
```

```
// Se carga el módulo de Express
const express = require("express");
// Se carga el módulo de cookie-parser
const cookieParser = require("cookie-parser");
// Creo la aplicación Express
const app = express();
// Se carga la función de cookie-parsing
app.use(cookieParser());
// Ruta para asignar una cookie
app.get('/cookie', (req, res) => {
    res.cookie('customCookie' , 'cookie
value').send('Cookie is set' )
});
// Obtengo los valores de las cookies
app.get('/check-cookie' , (req, res) => {
    // Muestro las cookies por consola
    console.log('Cookies: ', req.cookies);
    res.end(req.cookies.customCookie )
});
```

Middlewares de manejo de errores

Las funciones de middleware de manejo de errores se definen de la misma forma que otras funciones de middleware, excepto que las funciones de manejo de errores tienen cuatro argumentos en lugar de tres: (err, req, res, next). Por ejemplo:

```
app.use((err, req, res, next) => {  
  console.error(err.stack);  
  res.status(500).send("Algo ha fallado!: " +  
err.message);  
});
```

Características

- El middleware de manejo de errores se define al final, después de otras llamadas de rutas y app.use().
- Las respuestas pueden estar en el formato que se prefiera, html, json, un texto simple.
- Se pueden definir varios manejadores de errores.
- Se invoca llamando a la función next() con cualquier valor como argumento, excepto el texto 'route'. En este caso Express considera que la solicitud actual tiene un error y omitirá las restantes funciones de middleware y direccionamiento que no son de manejo de errores.

```
// Subpila de middleware que maneja solicitudes GET a la  
ruta /student/:id.  
app.get("/student/:id",  
  (req, res, next) => {  
    if (isNaN(Number(req.params.id))) throw new  
Error('Student ID inválido, introduzca un número' );  
    // Si el student ID es 0, salta a la siguiente ruta  
'special'  
    if (req.params.id == 0) next("route");  
    // en otro caso pasa el control al siguiente  
middleware 'regular'  
    else next(); //  
  },  
  (req, res, next) => {  
    // 'renderiza' una página regular  
    res.send("regular");  
  }  
);
```

Ejercicios

1. En los ejemplos de middlewares `express.json`, `express.text` y `express.urlencoded`, en lugar de una respuesta vacía, devuelve el valor del campo `title`.
2. Añadir una validación en la ruta **`/student/:id`** para devolver un error usando **`next`**, en caso de que el `id` sea mayor de 99.
3. Refactorizar el archivo `app.js` y crear un router para las rutas **`book`**. Importar y usar el router en el archivo principal `app.js`



ALBAÑILES DIGITALES

Developing your new career



VeriDas