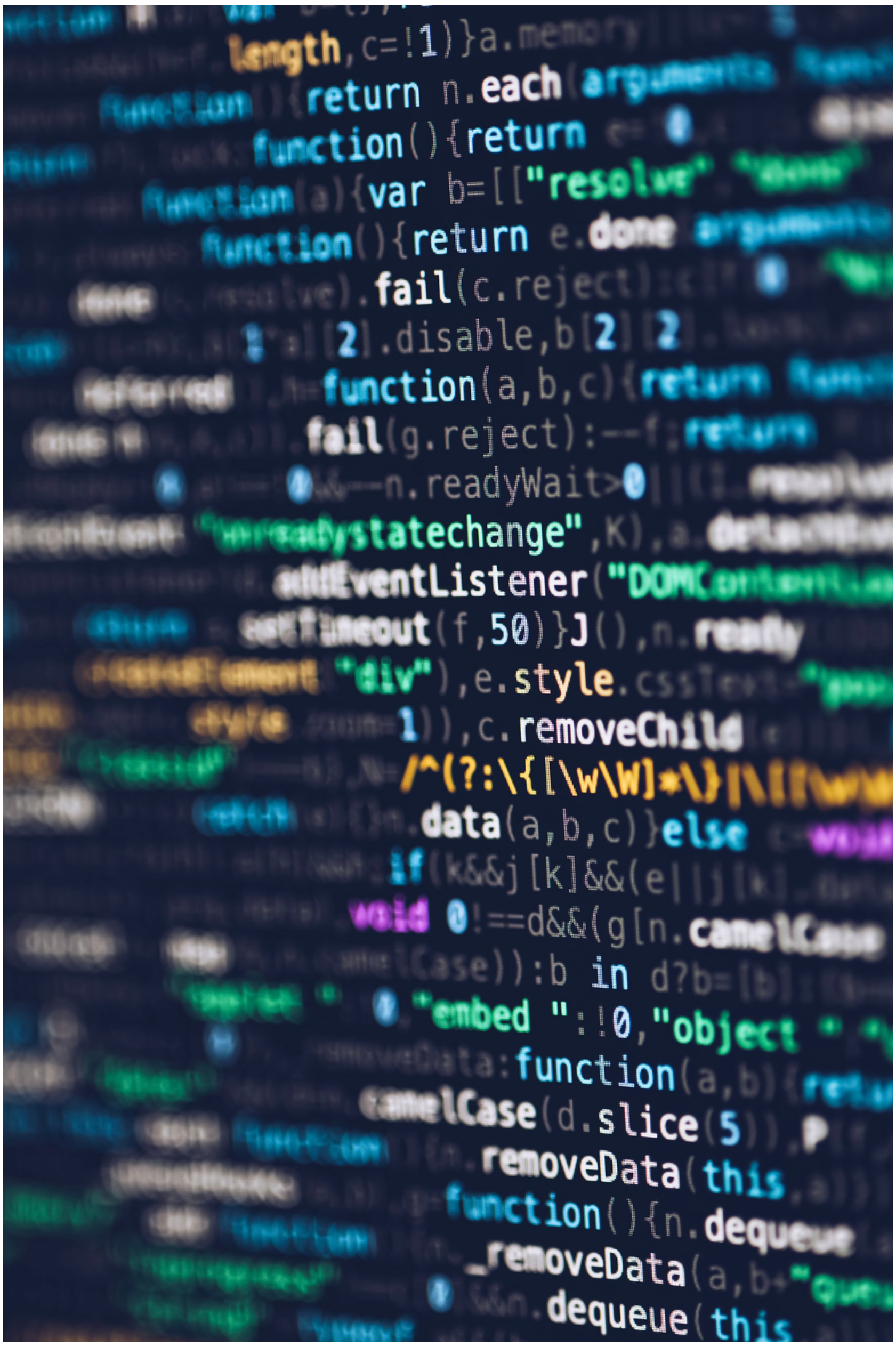
Lección 3:
Middlewares, next()



Tiempo de ejecución: 4 horas

Planteamiento de la sesión

Materiales

- Utilización del middleware de Express
- express-validator
- Middleware de terceros

Es hora de revisar entonces lo que es un middleware y cómo podemos implementarlo, aunque ya hemos trabajado con algunos a nivel de aplicación, revisaremos esos casos más adelante

¿Qué es un 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/respuesta de la aplicación. La siguiente función de middleware se denota normalmente con una variable denominada next.

Las funciones de 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.
- Invocar la siguiente función de middleware en la pila.

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.

Express divide las responsabilidades a la hora de atender un request en pequeñas funciones con objetivos y responsabilidades claras, que se encargan de organizar el código, a estas funciones las conocemos como middlewares.

Una aplicación Express puede utilizar diferentes tipos de middleware, sin embargo nos interesan solo algunos de ellos:

- Middlewares de aplicación
- De ruta
- De manejo de errores
- De terceros

Vamos a revisar un poco de cada uno de ellos, sin embargo, tenga en cuenta que los de terceros los utilizaremos un poco más cuando se trabaje con express validator express-validator

Middlewares de aplicación

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.

Este ejemplo muestra una función de middleware sin ninguna vía de acceso de montaje. La función se ejecuta cada vez que la aplicación recibe una solicitud.

```
app.use(function (req, res, next) {
  console.log('Time:', Date.now());
  next();
});
```

Si revisamos el entry point, vamos a encontrarnos con que ya tenemos varios métodos similar al del ejemplo, por nombrar uno de ellos tenemos:

app.use(morgan('dev'));

que registra en consola las solicitudes que llegan al servidor independiente de la ruta, ahora, si agregamos el código del ejemplo, en la consola junto con cada petición se imprimirá un número entero que se traduce en la fecha actual en milisegundos.

podríamos crear un Middleware que se encargue de guardar en los archivos de logs todas las peticiones que lleguen al servidor, algo similar a lo que se implementó en el log de la ruta raíz.

Middleware de ruta

El middleware de nivel de direccionador funciona de la misma manera que el middleware de nivel de aplicación, excepto que está enlazado a una instancia de express.Router().

Al implementar estos middleware vamos a encontrarnos que ya no hacen uso del app.use, sino del router, en nuestro caso, router.use o junto con el METHOD que corresponda, esto quiere decir, que el registro en logs que creamos anteriormente, podría perfectamente escribirse como un middleware de la ruta raíz.

Vamos a aplicar ese cambio, para ello vamos a crear un directorio llamado middlewares y dentro un archivo generalMiddleware para pasar el código que tenemos en el controlador a un middleware de ruta.

En la rama step4 puede encontrar los cambios aplicados

Básicamente lo que debemos hacer es tomar el código escrito en el controlador y llevarlo al archivo creado, recordando que debemos exportar el middleware para poder utilizarlo en nuestro router.

```
//a general middleware to log the request to the general_logs.txt file
const fs = require('fs');
const path = require('path');

const generalMiddleware = (req, res, next) => {
    const message = 'ingreso en la ruta ' + req.url;
    const dateTime = new Date().toLocaleString();
    fs.createWriteStream(path.join('logs', 'general_logs.txt'), { flags: 'a' });
       fs.appendFileSync(path.join('logs', 'general_logs.txt'), dateTime + ' - ' +
message + '\n');
    next();
}
module.exports = generalMiddleware;
```

Ahora es necesario importarlo en el router de libros y agregarlo en la ruta, entre el path y el controlador.

```
const generalMiddleware = require('../middlewares/generalMiddleware');

/* GET home page. */
router.get('/', generalMiddleware, BooksController.index);
```

Al ejecutar el programa debemos notar que el comportamiento es el mismo que con el caso anterior, con la ventaja que si tenemos más rutas, solo necesitamos invocar al middleware para hacer log de las que necesitemos.

Middleware de manejo de errores

El middleware de manejo de errores siempre utiliza cuatro argumentos. Debe proporcionar cuatro argumentos para identificarlo como una función de middleware de manejo de errores. Aunque no necesite utilizar el objeto next, debe especificarlo para mantener la firma. De lo contrario, el objeto next se interpretará como middleware normal y no podrá manejar errores.

El uso de los manejadores de errores como middleware es para los errores 500, es decir errores ocurridos en el servidor, la recomendación aquí es tomar el error, registrarlo en un log y darle una respuesta en una vista al usuario. El proceso es el mismo que para los middleware ya creados.

Si revisamos el entry point, encontramos que ya existe un middleware de manejo de error:

```
// error handler
app.use(function(err, req, res, next) {
  // set locals, only providing error in development
  res.locals.message = err.message;
  res.locals.error = req.app.get('env') === 'development' ? err : {};

  // render the error page
  res.status(err.status || 500);
  res.render('error');
});
```

La ejercitación sería entonces hacer que no solo se muestre la vista de error sino también que se cree un error log que almacene la info.

Finalmente en este apartado, los errores 404, aunque no son registrador como un middleware de error, Express generator crea un middleware para que cuando se detecte un error de este tipo, lo trabaje como un error del servidor.

Lo que nos interesa aquí es generar un página 404 personalizada, que primero es una buena práctica y segundo, podemos agregar información importante allí para que el usuario que no encuentre algo, no abandone el sitio.

Para ejemplo solo se pondrá una imagen con un 404 grande en una vista creada para tal fin. Necesitamos entonces crear una vista llamada 404, y modificar el middleware para renderizar esa vista.

```
// catch 404 and forward to 404 page
app.use((req, res, next) => {
  res.status(404).render('page404', { title: '404: Not Found' });
  next();
})
```

Puede también agregar un registro a un log si lo requiere. La vista se vería como la siguiente:

Middlewares de terceros

Utilice el middleware de terceros para añadir funcionalidad a las aplicaciones Express. Como se mencionó en un inicio, por ahora vamos a dejar en pendiente el middleware de terceros, ya que con express validator se trabajará con uno, que podemos implementar a nivel de ruta para verificar información que llega en la petición. Middleware de terceros.