
SENPAI
ACADEMY

try - catch

Node.js - Manejo de errores

La keyword ***try*** permite controlar sentencias para poder hacer un correcto manejo de errores en nuestra aplicación.

Un bloque ***try*** es seguido de un bloque ***catch*** donde especificamos que hacer en caso de que ocurra una excepción.

Además podemos sumar un bloque ***finally*** que se ejecutará siempre, sin importar si ocurre un error o no.

Las excepciones que controlamos son errores en tiempo de ejecución, que debemos estar preparados para resolver.

```
try {
  console.log('Inicio del Try');
  objNoDefinido.x = 100;
  console.log('Fin del Try');
} catch (err) {
  console.log('Ocurrió un error!');
  console.error(err);
} finally {
  console.log('finally!');
}
```

SENPAI ACADEMY

Códigos de error HTTP

SENPAI
ACADEMY

Node.js - Códigos de error HTTP

Los códigos de error se utilizan para expresar que error ocurrió en la petición HTTP, pero es nuestra responsabilidad retornar el código correcto.

400 Bad Request
El servidor no va a procesar la solicitud debido a algo que es percibido como un error del cliente. Ej: Solicitud mal formada, sintaxis errónea, etc.

403 Forbidden
La solicitud fue legal, pero el servidor rehúsa responderla dado que el cliente no tiene los privilegios para realizarla.

Node.js - Códigos de error HTTP

401 Unauthorized
Similar a 403, pero específicamente para cuando la autentificación es posible pero ha fallado o no fue enviada.

404 Not Found
Recurso no encontrado. Se utiliza cuando el servidor web no encuentra la página o recurso solicitado.

500 Internal Server Error
Este código lo emitimos frente a situaciones de error ajenas al funcionamiento normal de la aplicación.

Error handler de Express

SENPAI
ACADEMY

Node.js - Manejo de errores

Express implementa por defecto un manejador de errores.

Si un error no es capturado y manejado, entonces Express responderá automáticamente con un Status Code 500, renderizando el Stacktrace del error en el navegador del usuario.

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

app.get('/', function(request, response) {
  objNoDefinido.x = 100;
  response.send('Llego a ' + request.originalUrl);
});

app.listen(3000, function () {
  console.log('App corriendo en el puerto 3000');
});
```

SENPAI ACADEMY

Node.js - Manejo de errores

Sin embargo, cuando el error ocurre en código asincrónico, si no lo controlamos, nuestra aplicación terminará.

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

app.get('/', function(request, response) {
  setTimeout(() => {
    objNoDefinido.x = 100;
    response.send('Llego a ' + request.originalUrl);
  }, 100);
});

app.listen(3000, function () {
  console.log('App corriendo en el puerto 3000');
});
```

SENPAI ACADEMY

Node.js - Manejo de errores

Para poder redirigir al manejador de errores de Express manualmente, debemos invocar a la función next pasando el error que capturamos como parámetro. Lo mismo se aplica en Callbacks o Promesas.

```
app.get('/', function(request, response, next) {
  setTimeout(() => {
    try {
      objNoDefinido.x = 100;
      response.send('Llego a ' + request.originalUrl);
    } catch(err) {
      next(err);
    }
  }, 100);
});
```

SENPAI ACADEMY

Node.js - Manejo de errores

Express también nos permite cambiar al manejador de errores, definiendo un middleware. De esta forma, podríamos devolver una pantalla de error personalizada.
Como los demás middlewares, esto también se puede manejar por cada ruta.

```
app.use(function(err, req, res, next) {
  res.status(500);
  res.sendFile(__dirname + "/public/error.html");
})

app.use('/api', function(err, req, res, next) {
  res.status(500);
  res.sendFile(__dirname + "/public/error.html");
})
```

SENPAI ACADEMY

SENPAI
ACADEMY

gustavguez

gustavguez

GUSTAVO RODRIGUEZ

FULL STACK DEVELOPER
SOLCRE