Error Handler

Язык оригинала: en

# Оригинал

What is  
asyncHandler.js  
?  
In a professional  
Express.js  
project,  
asyncHandler.js  
(or  
catchAsync.js  
) is a  
utility function  
used to  
wrap your async route handlers  
(like  
signup  
,  
login  
, etc.) so that you don't need to write  
try...catch  
blocks in every controller.  
Problem It Solves:  
Normally in an async controller:  
export const signup = async (req, res, next) => {  
 try {  
 // your logic  
 } catch (error) {  
 next(error); // manually send error to global error handler  
 }  
};  
Enter fullscreen mode  
Exit fullscreen mode  
That repetition gets messy.  
asyncHandler  
makes it cleaner.  
The  
asyncHandler  
Utility:  
You can create a file called  
utils/asyncHandler.js  
and write this:  
const asyncHandler = (fn) => (req, res, next) => {  
 Promise.resolve(fn(req, res, next)).catch(next);  
};  
  
export default asyncHandler;  
Enter fullscreen mode  
Exit fullscreen mode  
How to Use It:  
Instead of:  
export const signup = async (req, res, next) => {  
 try {  
 // your logic  
 } catch (err) {  
 next(err);  
 }  
};  
Enter fullscreen mode  
Exit fullscreen mode  
You write:  
import asyncHandler from '../utils/asyncHandler.js';  
  
export const signup = asyncHandler(async (req, res, next) => {  
 // just write your logic — no need for try-catch  
});  
Enter fullscreen mode  
Exit fullscreen mode  
Why Using  
asyncHandler  
?  
Cleaner controllers  
Centralized error handling  
Less boilerplate  
Best practice in real-world Express apps  
Directory example:  
project-root/  
├── controllers/  
│ └── auth.controller.js  
├── middlewares/  
│ └── errorHandlers.js  
├── utils/  
│ └── asyncHandler.js ✅  
├── routes/  
├── models/  
├── app.js  
Enter fullscreen mode  
Exit fullscreen mode  
When to use  
try/catch  
after using  
asyncHandler  
You still need  
try/catch  
inside an  
asyncHandler  
only if you are doing something like  
database transactions  
or  
manual rollback  
— where cleanup is needed even if an error occurs or you need custom logic during failure.  
Example: Database transaction rollback  
const signup = asyncHandler(async (req, res) => {  
 const transaction = await sequelize.transaction();  
  
 try {  
 const user = await User.create({...}, { transaction });  
  
 // simulate a bug (e.g., missing env var)  
 const token = jwt.sign({ id: user.id }, process.env.JWT\_SECRET);  
  
 await transaction.commit();  
  
 res.status(201).json({ token });  
  
 } catch (error) {  
 await transaction.rollback(); // 🛑 must rollback manually  
 throw error; // let asyncHandler handle the error  
 }  
});  
Enter fullscreen mode  
Exit fullscreen mode  
If you didn’t use  
try/catch  
here, the transaction would not rollback, and you'd leave your DB in a broken state (e.g., partial insert).  
So:  
asyncHandler  
= automatic error forwarding to Express  
try/catch  
= necessary for manual cleanup logic before the error is forwarded  
@adefam

# Перевод на русский

Что такое  
Asynchandler.js  
?  
В профессионале  
Express.js  
проект,  
Asynchandler.js  
(или  
Catchasync.js  
) есть  
функция утилиты  
привыкший  
Оберните свои асинхронные обработки маршрутов  
(нравиться  
зарегистрироваться  
В  
авторизоваться  
и т. д.), чтобы вам не нужно писать  
Попробуй ... поймать  
блоки в каждом контроллере.  
Проблема это решает:  
Обычно в асинхронном контроллере:  
Экспорт const invignup = async (req, res, next) => {  
 пытаться {  
 // ваша логика  
 } catch (error) {  
 Далее (ошибка); // вручную отправлять ошибку в Global Error Ganserler  
 }  
};  
Введите полноэкранный режим  
Выйдите из полноэкранного режима  
Это повторение становится грязным.  
Асинхендлер  
делает его чище.  
А  
Асинхендлер  
Утилита:  
Вы можете создать файл с названием  
utils/asynchandler.js  
и напишите это:  
const asynchandler = (fn) => (req, res, next) => {  
 Spears.Resolve (fn (req, res, next)). Catch (Next);  
};  
  
Экспорт дефолта Асинхендлер;  
Введите полноэкранный режим  
Выйдите из полноэкранного режима  
Как его использовать:  
Вместо:  
Экспорт const invignup = async (req, res, next) => {  
 пытаться {  
 // ваша логика  
 } catch (err) {  
 Next (err);  
 }  
};  
Введите полноэкранный режим  
Выйдите из полноэкранного режима  
Вы пишете:  
импортировать асинхендлер из '../utils/asynchandler.js';  
  
Export const invignup = asynchandler (async (req, res, next) => {  
 // просто напишите свою логику-нет необходимости в промежутке  
});  
Введите полноэкранный режим  
Выйдите из полноэкранного режима  
Почему использование  
Асинхендлер  
?  
Более чистые контроллеры  
Централизованная обработка ошибок  
Меньше шаблон  
Передовая практика в приложениях для экспресс-реальных  
Пример каталога:  
проект-корень/  
Controllers/ Controllers/  
│ └ └í auth.controller.js  
├ack Middlewares/  
│ └ack righthandlers.js  
Utils/  
Asynchandler.js ✅  
├-- -маршруты/  
Models/ Модели/  
App.js  
Введите полноэкранный режим  
Выйдите из полноэкранного режима  
Когда использовать  
попробуйте/поймать  
После использования  
Асинхендлер  
Тебе все еще нужно  
попробуйте/поймать  
внутри  
Асинхендлер  
Только если вы делаете что -то вроде  
Транзакции базы данных  
или  
ручной откат  
- где необходима очистка, даже если возникает ошибка или вам нужна пользовательская логика во время сбоя.  
Пример: откат транзакции базы данных  
const inficup = asynchandler (async (req, res) => {  
 const transaction = wait sedize.transaction ();  
  
 пытаться {  
 const user = await user.create ({...}, {transaction});  
  
 // имитировать ошибку (например, отсутствующий env var)  
 const token = jwt.sign ({id: user.id}, process.env.jwt\_secret);  
  
 ждать transaction.commit ();  
  
 res.status (201) .json ({token});  
  
 } catch (error) {  
 ждать транзакции. rollback (); // 🛑 Должен откатываться вручную  
 бросить ошибку; // пусть асинхендлер обрабатывает ошибку  
 }  
});  
Введите полноэкранный режим  
Выйдите из полноэкранного режима  
Если вы не использовали  
попробуйте/поймать  
Здесь транзакция не будет отказываться, и вы оставите свой БД в сломанном состоянии (например, частичная вставка).  
Так:  
Асинхендлер  
= автоматическая пересылка ошибок для выражения  
попробуйте/поймать  
= Необходимо для ручной логики очистки до пересечения ошибки  
@adefam