Need for custom request handler

- Whenever we interact with the database,
- It will take time -> we need to use async-await
- We have to use try-catch Now, we have two options - try-catch or promises.
- Since we will be frequently interacting with the database in user controllers and video controllers, instead of writing this syntax every time, even though it's a common thing, why not create a utility file to assist me?
- I can create a generalized function for such wrappers.
- Whenever we need to execute a function in this way, we can pass the function to this handler method, and it will execute it and return the result.
- So basically, I will apply the wrapper around it.
-

> In Hindi
>
> - DB se jab bhi baat karenge
> - Time lagega hi -> async await use karna hi hoga
> - try-catch lagana hi padega
> - Ab humare paas 2 optionh ain - try-catch or promises
> - DB se to baat karne wale hi hain baar baar, user ke controller me karenge, video ke controller me karenge.
> - To har baar itna syntax likhna, jabki ye common cheez hai. To kyu na main ek utility file bana lun, jo meri help karde. Aur is tarah ke wrapper ka ek generalized function bana dun.
> - Aur jab bhi hume function is tarah se execute karna ho tab aap mere method me function pass kar dena, main execute kar ke wapas de dunga. Wrapper laga dunga uske aage.

utils/asyncHandler.js

```
// Higher order functions explained
// const asyncHandler = () =>{}
// const asyncHandler = (func) =>{}
// const asyncHandler = (func) => {() =>{}}
// const asyncHandler = (func) => async () =>{}

const asyncHandler = (requestHandler) => {
  return (req, res, next) => {
    Promise.resolve(requestHandler(req, res, next)).catch((err) => next(err));
  };
};
export { asyncHandler };


// using try catch

/*
const asyncHandler = (fn) => async (req, res, next) => {
  try {
    await fn(req, res, next);
```

```
  } catch (error) {
    res.status(err.code || 500).json({
      success: false,
      message: err.message,
    });
  }
};
*/
```

Explaination

asyncHandler Function:

- `const asyncHandler = (requestHandler) => {...}` : This function is a higher-order function that takes a requestHandler function as a parameter.
- `return (req, res, next) => {...}` : It returns an anonymous function that takes the standard Express.js middleware parameters (req, res, and next).

Promise and Error Handling:

- `Promise.resolve(requestHandler(req, res, next))` : This part of the code ensures that the requestHandler function is always wrapped in a resolved Promise, allowing consistent handling of asynchronous code.
- `.catch((err) => next(err))` : If there is any error during the execution of requestHandler, the error is caught, and next(err) is called to pass the error to the Express.js error-handling mechanism.

Usage of asyncHandler: You can use asyncHandler to wrap asynchronous route handlers.

Need for custom api error

```
class ApiError extends Error {
  constructor(
    statusCode,
    message = "Something went wrong",
    errors = [],
    stack = ""
  ) {
    super(message);
    this.statusCode = statusCode;
    this.data = null;
    this.message = message;
    this.success = false;
    this.errors = errors;
    if (stack) {
      this.stack = stack;
    } else {
      Error.captureStackTrace(this, this.constructor);
    }
  }
}
export { ApiError };
```

Code Explanation:

`ApiError` Class:

- Extends the built-in `Error` class to create a custom error class specifically for API-related errors.

Constructor Parameters:

- `statusCode` : HTTP status code to indicate the type of error (e.g., 404 for not found, 500 for internal server error).
- `message` : A human-readable error message. Defaults to "Something went wrong" if not provided.
- `errors` : An array containing additional error details or validation errors.
- `stack` : The stack trace for the error. If not provided, it captures the stack trace at the point of instantiation.

Additional Properties:

- `statusCode` : Represents the HTTP status code of the error.
- `data` : Can be used to attach additional data related to the error (initialized as `null`).
- `message` : The error message.
- `success` : Indicates whether the operation was successful (initialized as `false`).
- `errors` : An array to hold additional error details or validation errors.

Summary:

This `ApiError` class provides a standardized way to handle and represent API-related errors, including the ability to specify HTTP status codes, error messages, and additional error details. It enhances the error-handling capabilities in API development.