



ChatGPT ▾

[Share](#) ...[New chat](#)[Library](#)[Sora](#)[GPTs](#)

Chats

Switch Statement Example

[Optimize Code with useMemo](#)[TanStack Table Sorting](#)[React style conversion](#)[React Icons vs SVGs](#)[Camera SVG Code](#)[Close SVG Icon](#)[Swiper Next Slide Button](#)[Custom Date Picker Tailwind](#)

switch statement with example in javascript

A `switch` statement in JavaScript is used to perform different actions based on different conditions. It's a cleaner alternative to multiple `if-else` statements when checking a variable against many values.

Syntax:

javascript

[Copy](#) [Edit](#)

```
switch (expression) {  
  case value1:  
    // code block  
    break;  
  case value2:  
    // code block  
    break;  
  default:  
    // code block  
}
```

- `expression` : value to compare
- `case` : checks if expression matches value
- `break` : exits the switch block (without it, it "falls through" to the next case)
- `default` : runs if no case matches

✓ Example:

javascript

[Copy](#) [Edit](#)

```
let day = 3;  
let dayName =
```

```
switch (day) {
```

```

switch (day) {
  case 1:
    dayName = "Monday";
    break;
  case 2:
    dayName = "Tuesday";
    break;
  case 3:
    dayName = "Wednesday";
    break;
  case 4:
    dayName = "Thursday";
    break;
  case 5:
    dayName = "Friday";
    break;
  case 6:
    break;
  case 7:
    dayName = "Sunday";
    break;
  default:
    dayName = "Invalid day";
}

console.log(dayName); // Output: Wednesday

```

Let me know if you want an example with strings, functions, or without using `break` to show fall-through behavior.



```

import { Request, Response } from "express";

// models
import User from "../models/users.model";

// utils
import { errorHandler } from "../utils/users.utils";

// post email
async function postEmail(req: Request, res: Response) {

```

```

async function postEmail(req: Request, res: Response) {
  try {
    const { email } = req.body;
    const otp = Math.floor(10000 + Math.random() * 90000).toString();
    const otpExp = new Date(Date.now() + 21 * 60 * 1000);
    await User.create({ email, otp, otpExp });
    res.status(201).json({ emailOTP: { email, otp, otpExp } });
  } catch (err) {
    errorHandler(↓);
    res.status(400).json({ error: "post email error" });
  }
}

```

```

// exports
export { postEmail };

function errorHandler(err: any) {
  console.log(err);
}

// exports
export { errorHandler };

```

I want to define the type of err instead of any how can I achieve



To type the `err` parameter in your `errorHandler` function more precisely instead of using `any`, you can use `unknown`, or a more specific type like `Error` or `Error & { code?: string }` depending on what you're handling.

✅ Best practice for general use:

ts

Copy Edit

```

function errorHandler(err: unknown) {
  if (err instanceof Error) {
    console.log("Error name:", err.name);
    console.log("Error message:", err.message);
    console.log("Stack trace:", err.stack);
  }
}

```

```

    } else {
      console.log("Unknown error:", err);
    }
  }
}

```

This is **safe** and future-proof — `unknown` forces you to **narrow** the type before using it.

🔧 If you're only working with standard JS errors:

```

ts                                                                    Copy Edit

function errorHandler(err: Error) {
  console.log(err.message);
}

```

But this assumes the error is always an `Error` object, which may not be true if something else is thrown (e.g., a string or object).

🌿 If you're handling database errors (like from Mongoose):

You may want to use more specific types:

```

ts                                                                    Copy Edit

import mongoose from "mongoose";

function errorHandler(err: unknown) {
  if (err instanceof mongoose.Error) {
    console.log("Mongoose error:", err.message);
  } else if (err instanceof Error) {
    console.log("General error:", err.message);
  } else {
    console.log("Unknown error:", err);
  }
}

```

✅ Final Suggestion (safe and flexible):

```
ts                                                                    Copy Edit

function errorHandler(err: unknown): void {
  if (err instanceof Error) {
    console.error("Error:", err.message);
  } else {
    console.error("Unknown error:", err);
  }
}
```

Let me know if you'd like a specific library (like Mongoose or Zod) and I can help you write more specific error types.

📄 👍 🗨 🔊 🔗 ↺

yes it is mongoose error to create a new user with the following schema

```
import { Document, Schema, Model, model } from "mongoose";
import { isEmail } from "validator";
```

```
// user interface
export interface IUser extends Document {
  email: string;
  otp: string;
  otpExp: Date;
  role: "super" | "super-admin" | "normal";
```

```
  createdAt?: Date;
  updatedAt?: Date;
}
```

```
// schema
const user: Schema<IUser> = new Schema<IUser>({
  {
    email: {
      type: String,
      required: [true, "Email address required."],
      unique: true,
      validate: {
```

```
    validator: (val: string) => isEmail(val),
    message: "Invalid email address.",
  },
},
otp: {
```

```
  required: [true, "OTP required."],
},
otpExp: {
  type: Date,
  required: [true, "OTP EXP required."],
},
role: {
  type: String,
  required: [true, "User role required."],
  default: "normal",
  enum: ["normal", "super", "sub"],
},
status: {
  type: String,
  required: [true, "Account status must be specified"],
  default: "pending",
  enum: ["penc", "active", "blocked"],
},
},
```

```
{
  timestamps: true,
}
);

// exports
export default model<IUser>("User", user);
```

so I want to define the err type correctly

Since you're using **Mongoose**, and errors from `.create()` or other model operations typically throw **Mongoose-specific error types**, you can handle them by importing and checking for specific error classes provided by Mongoose.

✔ Recommended Way to Type `err` (for Mongoose)

ts

Copy Edit

```
import mongoose from "mongoose";

function errorHandler(err: unknown): void {
  if (err instanceof mongoose.Error.ValidationError) {
    console.error("Validation Error:", err.message);
    for (const field in err.errors) {
      console.error(`Field: ${field}, Message: ${err.errors[field].message}`);
    }
  } else if (err instanceof mongoose.Error.CastError) {
    console.error("Cast Error:", err.message);
  } else if (err instanceof mongoose.Error) {
    console.error("Mongoose Error:", err.message);
  } else if (err instanceof Error) {
    console.error("Generic Error:", err.message);
  } else {
    console.error("Unknown Error:", err);
  }
}
```

💡 Why not use `any`?

- `any` gives up all type safety.
- `unknown` forces you to **narrow** the type before using it, which is safer.

🔗 Examples of Common Mongoose Error Types.

Error Class	When It Occurs
<code>mongoose.Error.ValidationError</code>	Schema validation fails
<code>mongoose.Error.CastError</code>	Invalid <code>_id</code> or field cast
<code>mongoose.Error.DocumentNotFoundError</code>	<code>.findByIdOrFail()</code> fails

🔧 Bonus: If you're handling duplicate email errors

You can also check for MongoDB duplicate key error ↓

```
if (
  err instanceof mongoose.Error &&
  "code" in err &&
  (err as any).code === 11000
) {
  console.error("Duplicate key error:", (err as any).message);
}
```

Would you like me to help you extract meaningful error messages and return them in

`res.status(400).json()`?

📋 👍 🗨 🔊 🔗 ↺ ✓



```
import mongoose from "mongoose";

function errorHandler(err: unknown): void {
  if (err instanceof mongoose.Error.ValidationError) {
    console.error("Validation Error:", err.message);
    for (const field in err.errors) {
      console.error(Field: ${field}, Message:
${err.errors[field].message});
    }
  } else if (err instanceof mongoose.Error.CastError) {
    console.error("Cast Error:", err.message);
  } else if (err instanceof mongoose.Error) {
    console.error("Mongoose General Error:", err.message);
  } else if (err instanceof Error) {
    console.error("Generic Error:", err.message);
  }
}
```



```
    console.error("Generic Error:", err.message);  
  } else {  
    console.error("Unknown Error:", err);  
  }  
}
```

can you explain it in detail

Absolutely! Let's go through your `errorHandler` function **line-by-line**, and I'll explain what it does and **why** it's useful for a Mongoose app like yours.

Full Function Recap

ts

 Copy  Edit

```
function errorHandler(err: unknown): void {  
  if (err instanceof mongoose.Error.ValidationError) {  
    console.error("Validation Error:", err.message);  
    for (const field in err.errors) {  
      console.error(`Field: ${field}, Message: ${err.errors[field].message}`);  
    }  
  } else if (err instanceof mongoose.Error.CastError) {  
    console.error("Cast Error:", err.message);  
  } else if (err instanceof mongoose.Error) {  
    console.error("Mongoose General Error:", err.message);  
  } else if (err instanceof Error) {  
    console.error("Generic Error:", err.message);  
  } else {  
    console.error("Unknown Error:", err);  
  }  
}
```

Purpose of the Function

This function is a **centralized error logger** that handles various types of errors — especially from **Mongoose** — in a safe and informative way. It lets you:

- Avoid `any` typing
- Understand what went wrong
- Customize what gets shown to the developer (or user, if needed)

Detailed Breakdown



1. `err: unknown`

You typed the `err` as `unknown`, which is **better than** `any` because:

- It forces you to narrow (type check) before using the value
- Helps avoid bugs when dealing with unexpected error types

2. `if (err instanceof mongoose.Error.ValidationError)`

This checks if the error was due to a **validation failure** in your schema (like invalid email or missing `otp` field).

- `ValidationError` contains a `.errors` object with details about each failing field.
- Example cause: `await User.create({ email: "b", otp: "", otpExp: new Date() })`

ts

 Copy  Edit

```
console.error("Validation Error:", err.message);
for (const field in err.errors) {
  console.error(`Field: ${field}, Message: ${err.errors[field].message}`);
}
```

This logs each specific field that failed validation (e.g., `email: Invalid email address.`)

3. `else if (err instanceof mongoose.Error.CastError)`

This checks if Mongoose failed to **cast** a value — most often happens with `ObjectId`.

- Example cause: `await User.findById("invalid-")`
- `CastError` is thrown when a value can't be converted to the expected type

```
ts                                                                    Copy Edit
console.error("Cast Error:", err.message);
```

4. `else if (err instanceof mongoose.Error)`

This catches **other general Mongoose errors** that don't fall into the above categories.

- Acts as a catch-all for all errors under `mongoose.Error`.
- Example: `MissingSchemaError`, `DiscriminatorError`, etc.

```
ts                                                                    Copy Edit
console.error("Mongoose General Error:", err.message);
```

5. `else if (err instanceof Error)`

This catches **general JavaScript errors** not specific to Mongoose.

- Example: `throw new Error("Something broke")`
- It's a fallback for any `Error` object not related to Mongoose.

```
ts                                                                    Copy Edit
console.error("Generic Error:", err.message);
```

6. `else`

This catches anything else — like if someone throws a **string**, **number**, or **custom object** (which is bad)

practice but still possible in JS).

ts

CopyEdit

```
console.error("Unknown Error:", err);
```

💡 What Else You Could Add

✅ Handle Duplicate Key Errors

Mongoose won't throw a `ValidationError` for unique violations — MongoDB throws a regular error with `code: 11000`.

You can add:



```
else if (typeof err === "object" && err && "code" in err && (err as any).code === 11000) {
  console.error("Duplicate Key Error:", (err as any).message);
}
```

✅ Summary Table

Check	What It Catches	Example
<code>mongoose.Error.ValidationError</code>	Schema validation failure	Missing/invalid field
<code>mongoose.Error.CastError</code>	Type casting failure	Invalid <code>_id</code>
<code>mongoose.Error</code>	Other MongoDB errors	Schema issues
<code>Error</code>	Regular JS errors	<code>throw new Error()</code>
<code>else</code>	Non-error types	<code>throw "string"</code>

Would you like me to help you return a **structured JSON response** for users (e.g., `res.status(400).json({`

message })) based on these error types:



No Data Message



Upgrade plan

More access to the best models

Ask anything



Tools



ChatGPT can make mistakes. Check important info.