* Product.schema.js

```
import mongoose from "mongoose";

const productSchema = new mongoose.Schema({
    name: {
        type: String,
        required: ["true", "please provide a product name"],
        trim: true,
        maxLength: [120, "product name should not be max than 120 chars"]
    },
price: {
    type: Number,
    required: ["true", "please provide a product price"],
    maxLength: [5, "product name should not be max than 5 chars"]
},
description: {
    type: String
},
photos: [
    {
        secure_url: {
            type: String,
            required: true
        }
    }
    stock: {
        type: Number,
        default: 0
    },
    sold: {
        type: Number,
        default: 0
    },
    collectionId: {
        type: mongoose.Schema.Types.ObjectId,
        ref: "Collection"
    }
}, {timestamps: true})

export default mongoose.model("Product", productSchema)
```

* Order.schema

```
import mongoose from "mongoose"

const orderSchema = new mongoose.Schema({
    product: {
        type: [
            {
                productId: {
                    type: mongoose.Schema.Types.ObjectId,
                    ref: "Product"
                },
                count: Number,
                price: Number
            }
        ]
    }
}, {timestamps: true})

export default mongoose.model("Order", orderSchema)
```

Refering to Product from Product.schema.js

* Moon Modeler // Create Database Website

* Coupon.Schema.js

```
import mongoose from "mongoose"

const couponSchema = new mongoose.Schema({
    code: {
        type: String,
        required: [true, "Please provide a coupon code"]
    },
    discount: {
        type: Number,
        default: 0
    },
    active: {
        type: Boolean,
        default: true
    }
}, {timestamps: true})


export default mongoose.model("Coupon", couponSchema)
```

* A function that returns a function or takes other functions as arguments is called higher Order function

* asyncHandler.js

```
const asyncHandler = (fn) => async (req, res, next) => {
    try {
        await fn(req, res, next)
    } catch (error) {
        res.status(error.code || 500).json({
            success: false,
            message: error.message
        })
    }
}

export default asyncHandler;
```

Controllers

```
import asyncHandler from "../service/asyncHandler";
import CustomError from "../utils/customError";

export const signUp = asyncHandler(async(req, res) => {
    //get data from user
    const {name, email, password } = req.body

    //validation
    if (!name || !email || !password) {
        res.status(400).json({
            success: false,
            message
        })
    }
})
```

using default error class
throw new Error ('Require Fil')

but now our custom class
throw new Error ("Please add all fields", 400);
↑ error code

Custom Error.js

```
class CustomError extends Error {
    constructor(message, code){
        super(message);
        this.code = code
    }
}

export default CustomError
```

① creating class CustomError
② Extending Error which is provided by default.

```
//check if user already exists
const existingUser = await User.findOne({email})
if (existingUser) {
    throw new CustomError("User already exists", 400)
}

const user = await User.create({
    name,
    email,
    password
})

const token = user.getJWTtoken()
//safety
user.password = undefined
```

here role will be USER as we set it as default

when we create, the select : false does not work

```
//store this token in user's cookie
res.cookie("token", token, cookieOption)

// send back a response to user
res.status(200).json({
    success: true,
    token,
    user,
})
```

Cross Origin Resource Sharing

* app.js

```
import express from "express"
import cors from "cors"
import cookieParser from "cookie-parser"

const app = express()

app.use(express.json()) //accept JSON data
app.use(express.urlencoded({extended: true})) // accept URL encoded Data
app.use(cors())
app.use(cookieParser()) // Server can access user's cookie

export default app;
```

*

```
export const cookieOptions = {
    expires: new Date(Date.now() + 3 * 24 * 60 * 60 * 1000),
    httpOnly: true   // user can only read cookies
}
```

3 days — milli sec

Cookie cannot be accessed by client side but only by server side.

* auth.controller.js

```
export const login = asyncHandler(async (req, res) => {
    const {email, password} = req.body

    //validation
    if (!email || !password) {
        throw new CustomError("PLease fill all details", 400)
    }

    //validation
    if (!email || !password) {
        throw new CustomError("PLease fill all details", 400)
    }

    const user = User.findOne({email}).select("+password")

    if (!user) {
        throw new CustomError("Invalid credentials", 400)
    }

const isPasswordMatched = await user.comparePassword(password)

if (isPasswordMatched) {
    const token = user.getJWTtoken()
    user.password = undefined
    res.cookie("token", token, cookieOptions)
    return res.status(200).json({
        success: true,
        token,
        user
    })
}

throw new CustomError("Password incorrect", 400);
}
```

"to match password as it won't be returned by default"

email : email

name — what you want to give

```
export const logout = asyncHandler(async (req, res) => {
    res.cookie("token", null, {
        expires: new Date(Date.now()),
        httpOnly: true
    })

    res.status(200).json({
        success: true,
        message: 'Logged Out'
    })
})
```