* Whenever we want to connect Schema and have a linkage between them

① collectionID : {
    type : mongoose.Schema.Types.ObjectID,
    ref : "collection" //name of the collection you want to link
}

* Whenever we are using middleware next, at the end of the function we have to call it

*

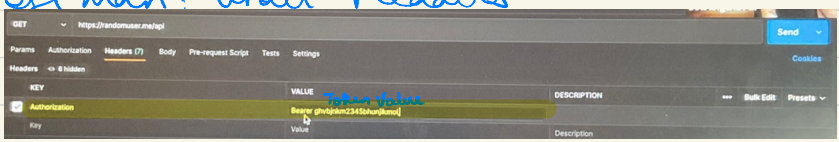
```
import User from "../models/user.schema.js";
import JWT from "jsonwebtoken"
import asyncHandler from "../service/asyncHandler.js";
import config from "../config.js";
import CustomError from "../utils/CustomError.js";



export const isLoggedIn = asyncHandler(async (req, res, next) => {
    let token;

    if (req.cookies.token) {
        token = req.cookies.token
    }
})
```

* Whenever sending a token which is authorisation in Postman: under Headers



```
export const isLoggedIn = asyncHandler(async (req, res, next) => {
    let token;

    if (req.cookies.token || (req.headers.authorization && req.headers.authorization.
    startsWith("Bearer")) ) {
        token = req.cookies.token ||
    }

    if (!token) {
        throw new CustomError("Not authorized to access this resource", 401)
    }

})
```

* StartsWith: Checks if the String starts with a specific String

```
token = req.cookies.token || req.headers.authorization.split(" ")[1];
```

split (condition) [1] → which value you want from the array
↳ will give array

* The Authorization token is always present in the form of "Bearer {token number}"

Seperate after space
["Bearer", "123456..."]
access the 1 Value from Array which is the number of the token

```
if (!token) {
    throw new CustomError("Not authorized to access this resource", 401)
}

try {
    const decodedJwtPayload = JWT.verify(token, config.JWT_SECRET)
} catch (error) {
    throw new CustomError("Not authorized to access this resource", 401)
}
```

will be a Object and all the Values of Payload are available
what to verify — with which to verify

```
try {
    const decodedJwtPayload = JWT.verify(token, config.JWT_SECRET);
    req.user = await User.findById(decodedJwtPayload._id) // we are saving every thing in our req
} catch (error) {
    throw new CustomError("Not authorized to access this resource", 401)
}
next();
```

imported

```
try {
    const decodedJwtPayload = JWT.verify(token, config.JWT_SECRET);

    req.user = await User.findById(decodedJwtPayload._id, "name email role")
    next()
} catch (error) {
    throw new CustomError("Not authorized to access this resource", 401)
}
```

*Note: No comma in between

Fields which we want to bring in after finding
here we are bringing name, email & role

* To check who is logged in, user, owner or moderator

```
export const authorize = (...requiredRoles) => asyncHandler( async (req, res, next) =>
{
    if (!requiredRoles.includes(req.user.role)) { //we can use this only after we have stored the
        throw new CustomError("You are not authorized to access this resource", 403)
    }
    next()
})
```

anyone user submits

information in request in the previous middleware

```
src > controllers > auth.controller.js > getProfile > asyncHandler() callback
 62 export const login = asyncHandler(async (req, res) => {…
 90 })
 91
 92 export const logout = asyncHandler(async (req, res) => {…
102 })
103
104 export const getProfile = asyncHandler(async (req, res) => {
105
106     const {user} = req    //destructured req user
107
108     if (!user) {
109         throw new CustomError("User not found", 401)
110     }
111
112     res.status(200).json({
113         success: true,
114         user
115     })
116 })
```

* Collection.controller.js

```
import asyncHandler from "../service/asyncHandler.js"
import CustomError from "../utils/CustomError.js"


export const createCollection = asyncHandler(async(req, res) => {
    const {name} = req.body

    if (!name) {
        throw new CustomError("Colection name is required", 400)
    }

    const collection = await Collection.create({
        name
    })

    res.status(200).json({
        success: true,
        message: "Collection was created successfully",
        collection
    })
})
```

```
let updatedCollection = await Collection.findByIdAndUpdate(collectionId, {
    name
}, {
    new: true
})
```

condition
name or name: name // what to Update
// pass the updated version back
runValidators: true // if mode Validators to be enabled like max length etc

```
if (!name) {
    throw new CustomError("Colection name is required", 400)
}

let updatedCollection = await Collection.findByIdAndUpdate(collectionId, {
    name
}, {
    new: true,
    runValidators: true
})

if (!updatedCollection) {
    throw new CustomError("Colection not found", 400)
}

res.status(200).json({
    success: true,
    message: "Collection was created successfully",
    collection
})
```

```
export const deleteCollection = asyncHandler(async(req, res) => {

    const {id: collectionId} = req.params

    const collectionToDelete = await Collection.findById(collectionId)

    if (!collectionToDelete) {
        throw new CustomError("Colection to be deleted not found", 400)
    }

    await collectionToDelete.remove()

    res.status(200).json({
        success: true,
        message: "Collection deleted successfully",

    })
})
```

```
export const getAllCollection = asyncHandler(async(req, res) => {



    const collections = await Collection.find()

    if (!collections) {
        throw new CustomError("No collection found", 400)
    }

    res.status(200).json({
        success: true,
        collections

    })
})
```

*To store images, videos in database using AWS

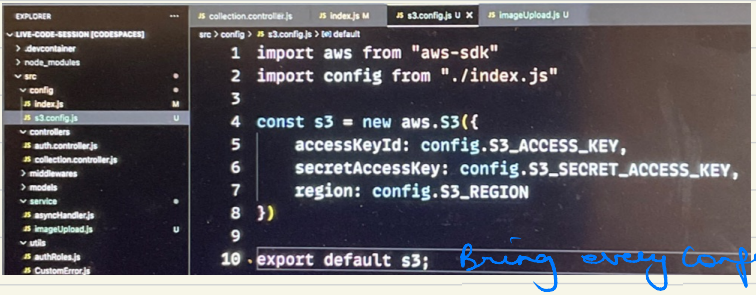
```
import dotenv from "dotenv"

dotenv.config()

const config = {
    PORT: process.env.PORT || 5000,
    MONGODB_URL: process.env.MONGODB_URL || "mongodb://localhost:27017/ecomm",
    JWT_SECRET: process.env.JWT_SECRET || "yoursecret",
    JWT_EXPIRY: process.env.JWT_EXPIRY || "30d",
    S3_ACCESS_KEY: process.env.S3_ACCESS_KEY,
    S3_SECRET_ACCESS_KEY: process.env.S3_SECRET_ACCESS_KEY,
    S3_BUCKET_NAME: process.env.S3_BUCKET_NAME,
    S3_REGION: process.env.S3_REGION,
}

export default config
```

*



Bring every configuration in one place

*

```
import s3 from "../config/s3.config.js"

export const s3FileUpload = async ({bucketName, key, body, contentType}) => {
    return await s3.upload({
        Bucket: bucketName,
        Key: key, //name of the file which should be unique for every pic or video
        Body: body, // where to upload
        ContentType: contentType // eg .png or .jpg etc
    })
    .promise()
}

export const s3deleteFile = async ({bucketName, key}) => {
    return await s3.deleteObject({
        Bucket: bucketName,
        Key: key,
    })
    .promise()
}
```

* npm i formidable // Upload files

```
import Product from "../models/product.schema.js"
import formidable from "formidable"
import { s3FileUpload, s3deleteFile} from "../service/imageUpload.js"
import Mongoose from "mongoose"
import asyncHandler from "../service/asyncHandler.js"
import CustomError from "../utils/CustomError.js"
import config from "../config/index.js"
import fs from "fs"


/******************************************************
 * @ADD_PRODUCT
 * @route https://localhost:5000/api/product
 * @description Controller used for creating a new product
 * @description Only admin can create the coupon
 * @descriptio Uses AWS S3 Bucket for image upload
 * @returns Product Object
 ******************************************************/


export const addProduct = asyncHandler(async (req, res) => {
    const form = formidable({ multiples: true, keepExtensions: true });

    form.parse(req, async function (err, fields, files){
        if (err) {
            throw new CustomError(err.message || "Something went wrong", 500)
        }

        let productId = new Mongoose.Types.ObjectId().toHexString()

        console.log(fields, files);
```

Upload multiple files

type of file

* Fields : contain any non file form submitted in the form (object)
* Files contains information about the uploaded file in the form (object)
* When uploading insta photo with caption - caption stored in fields and information about the photo stored in Files

```
yaml                                                    Copy code

Fields: { text_field: 'hello' }
Files: { file_field:
   File {
     _events: [Object: null prototype] {},
     _eventsCount: 0,
     _maxListeners: undefined,
     size: 123456,
     path: '/tmp/upload_123456789abcdefg.jpg',
     name: 'file.jpg',
     type: 'image/jpeg',
     hash: null,
     lastModifiedDate: 2021-11-05T00:00:00.000Z,
     _writeStream:
      WriteStream {
        _writableState: [WritableState],
        writable: false,
        _events: [Object: null prototype] {},
        _eventsCount: 0,
        _maxListeners: undefined,
        path: '/tmp/upload_123456789abcdefg.jpg',
        fd: null,
        flags: 'w',
        mode: 438,
        start: undefined,
        autoClose: true,
        pos: undefined,
        bytesWritten: 123456,
        closed: true
      },
     [Symbol(kCapture)]: false
   } }
```

```
lua                                                     Copy code

const formidable = require('formidable');
const http = require('http');

http.createServer(function(req, res) {
  if (req.url === '/upload' && req.method.toLowerCase() === 'post') {
    const form = formidable({ multiples: true });

    form.parse(req, function(err, fields, files) {
      if (err) {
        console.error(err);
        res.statusCode = 500;
        res.end('Error uploading file');
        return;
      }

      console.log('Fields:', fields);
      console.log('Files:', files);

      res.statusCode = 200;
      res.end('File upload successful');
    });

    return;
  }

  // If the request is not for /upload, serve a form that allows file upload
  res.writeHead(200, { 'Content-Type': 'text/html' });
  res.end(`
    <form action="/upload" enctype="multipart/form-data" method="post">
      <div>Text field: <input type="text" name="text_field"></div>
      <div>File field: <input type="file" name="file_field" multiple></div>
      <input type="submit" value="Upload">
    </form>
  `);
}).listen(8080);
```