* Ctrl + Shift + P (Shortcut for Palette)

* iifi (immediately invoked function)

(async () => { }) ()

to create function — to invoke the function

* we used async here because database is in another continent and we have to wait for data to be fetched

```
(async () => {
  try {
await mongoose.connect("mongodb://localhost:27017/ecom")
    console.log("connected");
    app.on('error', (err) => {
      console.error("Error:", err);
      throw err;
    });
    app.listen(5000, () => { console.log("Listening") });
  } catch (err) {
    console.log("Error:", err);
    throw err
  }
})()
```

(port, name; 'error' = event; catch error by express; to throw err so application stops here)

* app.on('name of event', callback), when app tries to connect to express

```
import mongoose from "mongoose";

const collectionSchema = new mongoose.Schema(
    {
        name: {
            type: String,
            required: ["true", "Please provide a collection name"],
            trim: true,
            maxLength : [
                120,
                "Collection name should not be more than 120 chars"
            ]
        }
    },{timestamps:true}
)
```

error message

error message

will add 2 fields ⇒ createdAt & updateAt to DB

export default mongoose.model("collection", collectionSchema);

```
password: {
    type: String,
    required: [true, "Password is required"],
    minLength: [8, "password must be at least 8 chars"],
    select: false
}
```

whenever I send a request in db to fetch details I do not want password to be fetched by default

imported

* enum : Object.values (AuthRoles)

limits the user within few options given by the admin for selection

`Object.values()` is a built-in method in JavaScript that returns an array containing the values of all enumerable properties in an object.

Here's an example of how to use `Object.values()`:

javascript — Copy code

```
const obj = {a: 1, b: 2, c: 3};
const values = Object.values(obj);
console.log(values); // [1, 2, 3]
```

AuthRoles.js

```
const AuthRoles = {
    ADMIN: "ADMIN",
    MODERATOR: "MODERATOR",
    USER: "USER"
}

export default AuthRoles
```

```
role: {
    type: String,
    enum: Object.values(AuthRoles),
    default: AuthRoles.USER
}
```

→ whenever a new Object will be created in db will be named as Admin if nobody passes anything extra

* forgotPasswordToken : String,
forgotPasswordExpiry: Date,

a unique token will be created and one copy will be sent to user and one in our db. So when the user clicks on the token which (with help of route) matches the copy of token which was sent to our db. We will allow the user to change password but for particular period of time

* userSchema.pre ("save", async function (next) {}
name of event — cannot use arrow functions here
// executed before saving a document

```
import mongoose from "mongoose";
import AuthRoles from "../utils/authRoles.js";
import bcrypt from "bcryptjs"

const userSchema = new mongoose.Schema({…
}, {timestamps: true})

//Encrypt the password before saving : HOOKS

userSchema.pre("save", async function(next){
    if (!this.isModified("password")) return next()
    this.password = await bcrypt.hash(this.password, 10)
    next() // pass on the flag that I am done
})

export default mongoose.model("User", userSchema)
```

* next() middleware is used to pass control to the next middleware function in the stack.

```
const express = require('express');
const app = express();

app.use((req, res, next) => {
  console.log('Middleware 1');
  next();
});

app.use((req, res, next) => {
  console.log('Middleware 2');
  next();
});

app.get('/', (req, res) => {
  console.log('Route handler');
  res.send('Hello, World!');
});

app.listen(3000, () => {
  console.log('Server listening on port 3000');
});
```

```
Middleware 1
Middleware 2
Route handler
```

* bcrypt.hash (what to bcrypt, how many time to rotate so that it becomes impossible to encrypt);

* hash : process of converting into long, unique string of characters

* Problem: Now password will be hashed so user will never be able to login

```
userSchema.pre("save", async function(next){
    if (!this.isModified("password")) return next()
    this.password = await bcrypt.hash(this.password, 10)
    next()
})

userSchema.methods = {
    //compare password
    comparePassword: async function(enteredPassword){
        return await bcrypt.compare(enteredPassword, this.password)
    }
}

export default mongoose.model("User", userSchema)
```

user input

* Example

```
schema.method({
    purr: function () {}
  , scratch: function () {}
});

// later
const fizz = new Kitty;
fizz.purr();
fizz.scratch();
```

* json web token

import JWT from "jsonwebtoken"

Whenever user logsin, their credentials are verified by the system. If credentials are valid, system generates JWT token and sends it back to user. This JWT token contains information about the user such as their ID, username and other details

* JWT_EXPIRY = 7d // Specifies expiration time of a token
JWT_SECRET = yoursecret // secret key used to sign the token and verify its authenticity

*

```
1  import dotenv from "dotenv"
2
3  dotenv.config() //method to load environment variables from .env file into process.env Object
4
5  const config = {
6      PORT: process.env.PORT || 5000,
7      MONGODB_URL: process.env.MONGODB_URL || "mongodb://localhost:27017/ecomm",
8      JWT_SECRET: process.env.JWT_SECRET || "yoursecret",
9      JWT_EXPIRY: process.env.JWT_EXPIRY || "30d"
10 }
11
12 export default config
```

* dotenv file is used to store configuration settings such as API keys, database credentials and other sensitive information that should not be hard coded in application code

* Create a token :

JWT.sign(payload, secretORprivatekey, [options, callback])

*

```
import mongoose from "mongoose";
import AuthRoles from "../utils/authRoles.js";
import bcrypt from "bcryptjs";
import JWT from "jsonwebtoken";
import config from "../config/index.js";

const userSchema = new mongoose.Schema({
}, {timestamps: true})

//Encrypt the password before saving: HOOKS

userSchema.pre("save", async function(next){
    if (!this.isModified("password")) return next()
    this.password = await bcrypt.hash(this.password, 10)
    next()
})

userSchema.methods = {
    //compare password
    comparePassword: async function(enteredPassword){
        return await bcrypt.compare(enteredPassword, this.password)
    },
    //generate JWT Token
    getJWTtoken: function(){
        JWT.sign({_id: this._id, role: this.role}, config.JWT_SECRET,
        {
            expiresIn: config.JWT_EXPIRY
        })
    }
}
```

default id given by mongoose

* JWT.sign() creates a long string which hides the information
* whoever know JWT.SECRET can decrypt the information
* payload : whichever information you want to encrypt in the token

```
const userSchema = new mongoose.Schema({
    name: {
        type: String,
        required: ["true", "Name is required"],
        maxLength: [50, "Name must be less than 50 chars"]
    },
    email: {
        type: String,
        required: ["true", "Email is required"],
    },
    password: {
        type: String,
        required: [true, "Password is required"],
        minLength: [8, "password must be at least 8 chars"],
        select: false
    },
    role: {
        type: String,
        enum: Object.values(AuthRoles),
        default: AuthRoles.USER
    },
    forgotPasswordToken: String,
    forgotPasswordExpiry: Date
}, {timestamps: true})
```

JWT.sign({data that will be stored in token}, will be used to sign the token, {expiry duration})

* Controllers communicate with client
* Models communicate with Database

* token = long string

* const forgotToken = crypto.randomByte(20).toString("hex")
// convert to hexadecimal string of 20 Bytes

*

```
//generate forgot password token
generateForgotPasswordToken: function (){
    const forgotToken = crypto.randomBytes(20).toString("hex")

    // just to encrypt the token generated by crypto
    this.forgotPasswordToken = crypto
    .createHash("sha256")
    .update(forgotToken)
    .digest("hex")

    //time for token to expire
    this.forgotPasswordExpiry = Date.now() + 20 * 60 * 1000

    return forgotToken
  }
}
```

convert string to hexa decimal
generate random string of length 20
algorithm
minutes seconds milliseconds = 1 second
20 minutes

* We store the link only which is provided by AWS after uploading the photos

* We cannot use this in arrow function