LAB 3

❖ CONTENT

- Authorization (role-based access)
- Install new package: multer
- Image upload
- Create and test Restful APIs for CRUD features

❖ INTRODUCTION

- API stands for Application Programming Interface. It serves as a bridge between different application, allowing them to communicate and interact with each other
- Common data format of API: JSON, XML
- Common types of API: RESTful, SOAP, GraphQL
- Common methods of API: GET (READ), POST (CREATE), PUT (UPDATE), DELETE (DELETE)
- The most popular usage of API nowadays: RESTful with JSON

❖ INSTRUCTION

1. Authorization

```
router.post('/register', async (req, res) => {
    try {
        var userRegistration = req.body;
        var hashPassword = bcrypt.hashSync(userRegistration.password, salt);
        var user = {
            username: userRegistration.username,
            password: hashPassword,
            role: 'user'
```

*Figure 1 - Set custom role for new user (**routes/auth.js**)*

```
router.post('/login', async (req, res) => {
   try {
      var userLogin = req.body;
      var user = await UserModel.findOne({ username: userLogin.username })
      if (user) {
         var hash = bcrypt.compareSync(userLogin.password, user.password)
         if (hash) {
            //initialize session after login success
            req.session.username = user.username;
            req.session.role = user.role;
            if (user.role == 'admin') {
               //redirect to admin page
            }
            else {
               //redirect to user page
            }
```

*Figure 2 – Add role to session and redirect page by role after login success (**routes/auth.js**)*

```
//check login only
const checkLoginSession = (req, res, next) => {
   if (req.session.username) {
      next();
   } else {
      res.redirect('/auth/login');
   }
}

//check single role
const checkSingleSession = (req, res, next) => {
   if (req.session.username && req.session.role == 'admin') {
      next();
   }
   else {
      res.redirect('/auth/login');
      return;
   }
}

//check multiple roles
const checkMultipleSession = (allowedRoles) => (req, res, next) => {
   if (req.session.username && allowedRoles.includes(req.session.role)) {
      next();
   } else {
      res.redirect('/auth/login');
   }
}

module.exports = {
   checkLoginSession,
   checkSingleSession,
   checkMultipleSession
}
```

*Figure 3 - update **auth** middleware to validate authorization by roles **(middlewares/auth.js)***

```
const { checkSingleSession, checkMultipleSession } = require('../middlewares/auth');

router.get('/', checkMultipleSession(['user', 'admin']) async (req, res) => {
   var productList = await ProductModel.find({}).populate('category');
   res.render('product/index', { productList });
});

router.get('/add', checkSingleSession, async (req, res) => {
   var categoryList = await CategoryModel.find({});
   res.render('product/add', { categoryList });
})
```

*Figure 4 – import and add middleware to routes for validation (**routes** folder)*

2. Install new package

```
npm install multer
```

Figure 5 - Install new package

3. Image upload

```
<form action="" method="post" enctype="multipart/form-data">
Image: <input type="file" name="image" id="" required>
```

*Figure 6 - update form add (**views/product/add.hbs**)*

```
//import and config "multer" package
var multer = require('multer');

var prefix = Math.floor(Math.random() * 100000000) + 1;

const storage = multer.diskStorage(
   {
      destination: (req, file, cb) => {
         cb(null, './public/images');
      },
      filename: (req, file, cb) => {
         let fileName = prefix + "_" + file.originalname;
         cb(null, fileName);
      }
   }
);

const upload = multer({ storage: storage })
```

*Figure 7 - import **multer** package, set **filename** and **file upload location** (**routes/product.js**)*

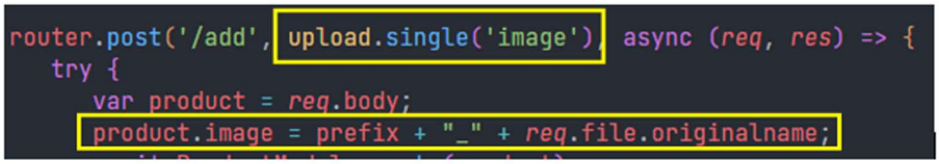

*Figure 8 - update route function to upload image and save it to db (**routes/product.js**)*

*Figure 9 - update image link on view to display (**views/product/index.hbs**)*

⇨ *You should create a middleware for image upload to reuse codes in other places*

4. Create Restful APIs

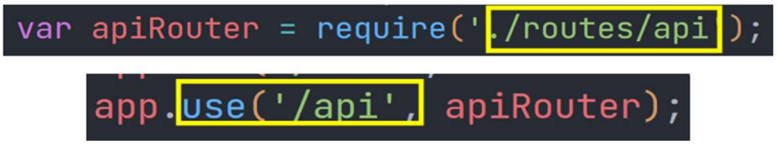

*Figure 10 - declare **apiRouter** to store APIs **(routes/api.js)***

Figure 11 - GET method (READ feature)

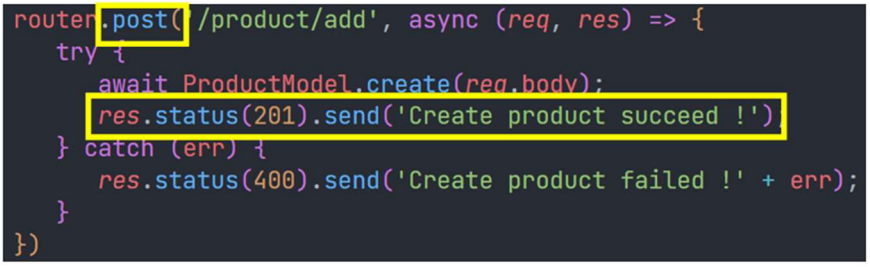

Figure 12 - POST method (CREATE feature)

```
router.put('/product/edit/:id', async (req, res) => {
    try {
        await ProductModel.findByIdAndUpdate(req.params.id, req.body);
        res.status(200).send('Edit product succeed !');
    } catch (err) {
        res.status(400).send('Edit product failed !' + err);
    }
})
```

Figure 13 - PUT method (UPDATE feature)

```
router.delete('/product/delete/:id', async (req, res) => {
    try {
        await ProductModel.findByIdAndDelete(req.params.id);
        res.status(200).send('Delete product succeed !');
    } catch (err) {
        res.status(400).send('Delete product failed !' + err);
    }
})
```

Figure 14 - DELETE method (DELETE feature)

5. Test Restful APIs

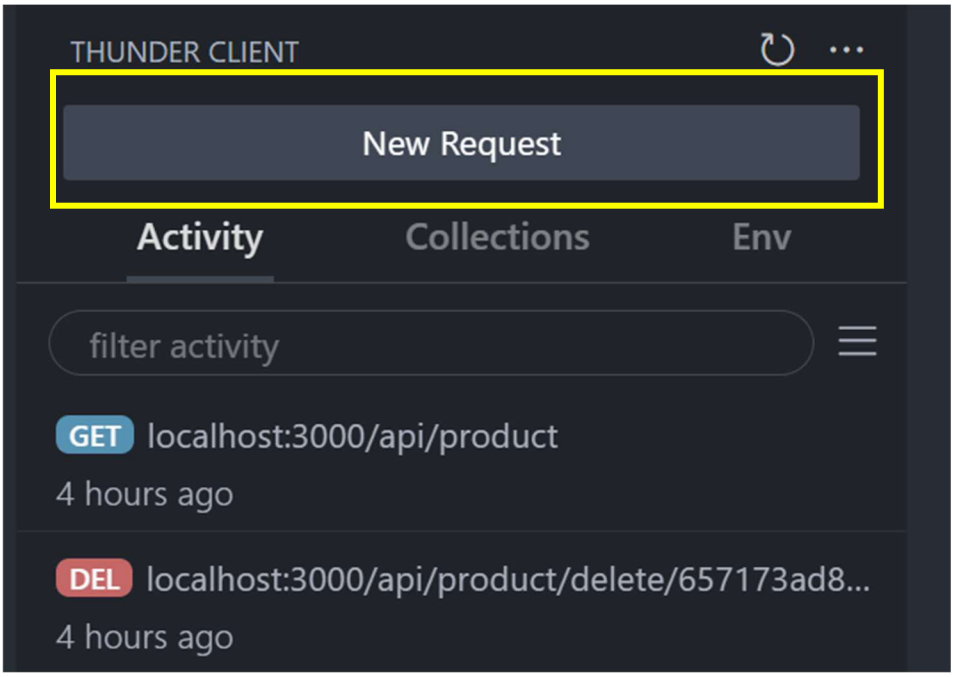

Figure 15 - Test Resful APIs with Thunder Client extension (or other similar tools)

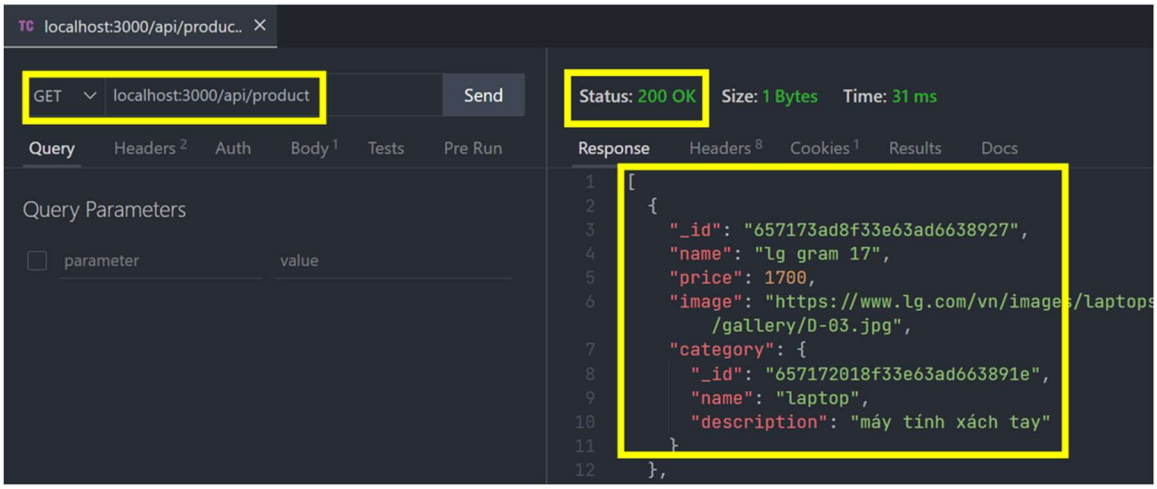

Figure 16 - Test GET method

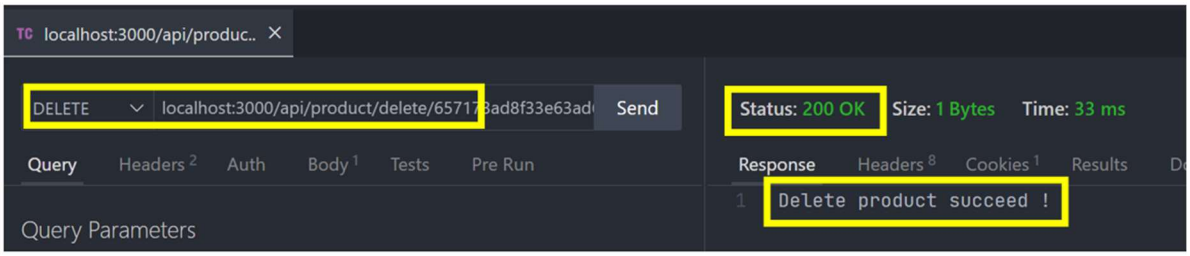

Figure 17 - Test DELETE method

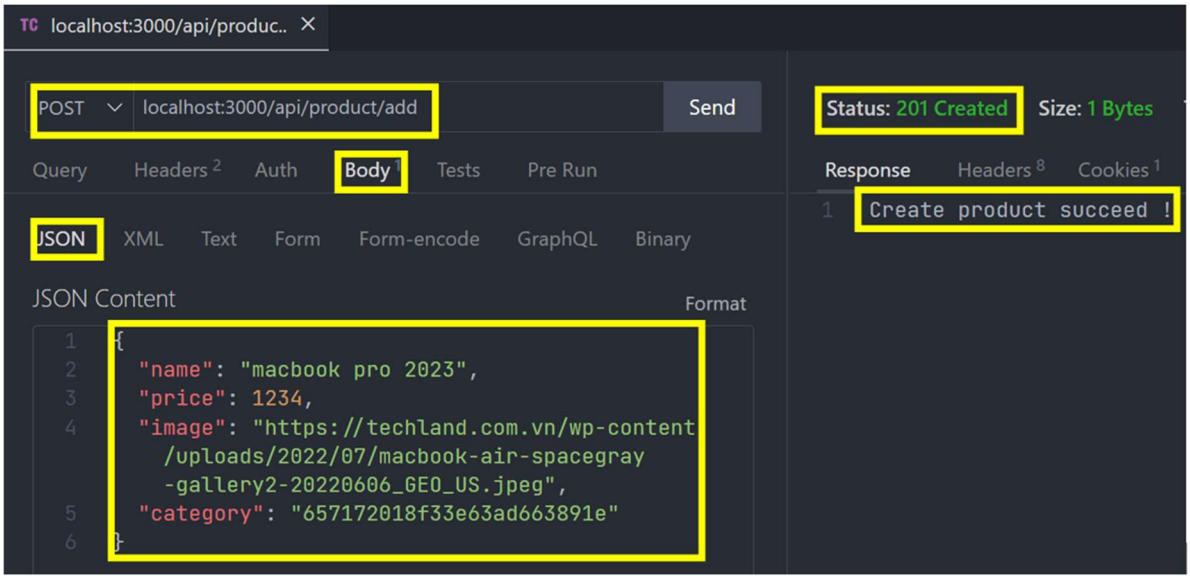

Figure 17 - Test CREATE method

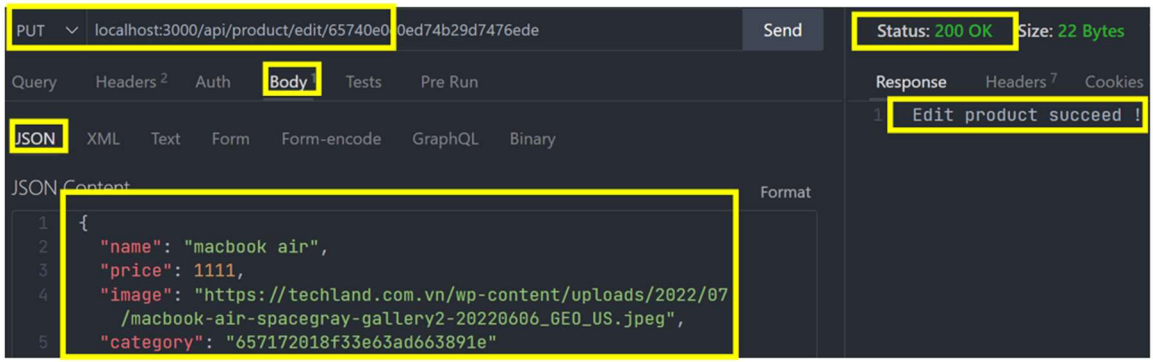

Figure 18 - Test PUT method