

db.js

const mongoose = require("mongoose");

const asynchandler = require("express-async-handler");

require("dotenv").config(); // .env 파일 사용하기 위해

const connectDb = asynchandler(async () => {

const connect = await mongoose.connect(process.env.DB\_CONNECT);

console.log(`DB connected: ${connect.connection.host}`);

});

module.exports = connectDb;

Images.js

const mongoose = require('mongoose');

// 스키마 정의

const ImageSchema = new mongoose.Schema({

// 게시물 제목

title: {

type: String,

required: true,

},

// 게시물 내용

body: {

type: String,

required: true,

},

filename: {

type: String,

required: true,

},

path: {

type: String,

require: true,

},

// 게시물 작성일

createdAt: {

type: Date,

default: Date.now(),

},

});

module.exports = mongoose.model('Image', ImageSchema);

Post.js  
  
const mongoose = require("mongoose");

// 스키마 정의

const PostSchema = new mongoose.Schema({

// 게시물 제목

title: {

type: String,

required: true,

},

// 게시물 내용

body: {

type: String,

required: true,

},

// 게시물 작성일

createdAt: {

type: Date,

default: Date.now,

},

});

// 모델 생성

module.exports = mongoose.models.Post || mongoose.model("Post", PostSchema);

User.js

const mongoose = require("mongoose");

const userSchema = new mongoose.Schema({

username: {

type: String, // 데이터 타입은 문자열

required: true, // 필수 항목

unique: true, // username은 중복할 수 없음

},

password: {

type: String, // 데이터 타입은 문자열

required: true, // 필수 항목

},

});

// 스키마를 모델로 변환한 후 내보내기

module.exports = mongoose.model("User", userSchema);

admin.js

const express = require("express");

const router = express.Router();

const adminLayout = "layouts/admin";

const adminLayout2 = "layouts/admin-nologout";

const asyncHandler = require("express-async-handler");

const bcrypt = require("bcrypt");

const User = require("../models/User");

const Post = require("../models/Post");

const jwt = require("jsonwebtoken");

const JWT\_SECRET = process.env.JWT\_SECRET;

// Check Login

const checkLogin = async (req, res, next) => {

const token = req.cookies.token; // 쿠키 정보 가져오기

// 토큰이 없다면

if (!token) {

return res.redirect("/admin");

}

// 토큰이 있다면 토큰을 확인하고 사용자 정보를 요청에 추가

try {

const decoded = jwt.verify(token, JWT\_SECRET);

req.userId = decoded.userId; // res.userId가 아닌 req.userId에 저장

next();

} catch (error) {

return res.redirect("/admin");

}

};

// GET /logout

// Admin Logout

router.get("/logout", (req, res) => {

res.clearCookie("token");

res.redirect("/"); // 루트 페이지로 이동

});

// GET /admin

// 관리자 페이지

router.get("/admin", (req, res) => {

const locals = {

title: "관리자 페이지",

};

res.render("admin/index", { locals, layout: adminLayout2 });

});

// GET /register

// 회원가입 페이지

router.get(

"/register",

asyncHandler(async (req, res) => {

res.render("admin/index", { layout: adminLayout2 });

})

);

// POST /admin

// 관리자 페이지 로그인

router.post(

"/admin",

asyncHandler(async (req, res) => {

const { username, password } = req.body; // req.body에서 username과 password를 가져옴

// 사용자 이름으로 사용자 찾기

const user = await User.findOne({ username });

// 일치하는 사용자가 없으면 에러 메시지 출력

if (!user) {

return res.status(401).json({ message: "일치하는 사용자가 없습니다." });

}

const isValidPassword = await bcrypt.compare(password, user.password);

// 비밀번호가 일치하지 않으면 에러 메시지 출력

if (!isValidPassword) {

return res.status(401).json({ message: "비밀번호가 일치하지 않습니다." });

}

// JWT 토큰 생성

const token = jwt.sign({ id: user.\_id }, JWT\_SECRET);

// JWT 토큰을 쿠키에 저장

res.cookie("token", token, {

httpOnly: true,

});

// 로그인에 성공하면 전체 게시물 목록 페이지로 이동

res.redirect("/allPosts");

})

);

// GET /allPosts

// 전체 게시물 목록 페이지

router.get(

"/allPosts",

checkLogin,

asyncHandler(async (req, res) => {

const locals = {

title: "전체 게시물",

};

const data = await Post.find().sort({ updatedAt: "desc", createdAt: "desc" });

res.render("admin/allPosts", {

locals,

data,

layout: adminLayout,

});

})

);

// GET /add

// Admin - Add Post

router.get(

"/add",

checkLogin,

asyncHandler(async (req, res) => {

const locals = {

title: "게시물 작성",

};

res.render("admin/add", {

locals,

layout: adminLayout,

});

})

);

// POST /add

// Admin - Add Post

router.post(

"/add",

checkLogin,

asyncHandler(async (req, res) => {

const { title, body } = req.body;

const newPost = new Post({

title: title,

body: body,

});

await Post.create(newPost);

res.redirect("/allPosts");

})

);

/\*\*

\* Get edit/:id

\* Admin - Edit Post

\*/

router.get(

"/edit/:id",

checkLogin,

asyncHandler(async (req, res) => {

const locals = {

title: "게시물 편집",

};

// id값을 사용해서 게시물 가져오기

const data = await Post.findOne({\_id: req.params.id});

res.render('admin/edit', {

locals,

data,

layout: adminLayout,

});

})

);

/\*\*

\* PUT /edit/:id

\* Admin - Edit Post

\*/

router.put(

"/edit/:id",

checkLogin,

asyncHandler(async (req, res) => {

await Post.findByIdAndUpdate(req.params.id, {

title: req.body.title,

body: req.body.body,

createdAt: Date.now(),

});

// 수정한 후 전체 목록 다시 표시

res.redirect('/allPosts');

})

)

/\*\*

\* DELETE /delete/:id

\* Admin - Delete Post

\*/

router.delete(

"/delete/:id",

checkLogin,

asyncHandler(async (req, res) => {

await Post.deleteOne({\_id: req.params.id});

res.redirect('/allPosts');

})

);

module.exports = router;

main.js

const express = require("express");

const router = express.Router();

const mainLayout = "layouts/main";

const Post = require("../models/Post");

// const Image = require("../models/Images");

const asynchandler = require("express-async-handler");

// const path = require('path');

// const fs = require('fs');

// const multer = require('multer');

// "/home" 라우트

router.get(

["/", "/home"],

asynchandler(async (req, res) => {

const locals = {

title: "Home",

};

const data = await Post.find().sort({ updatedAt: "desc", createdAt: "desc" });

res.render("index", { locals, data, layout: mainLayout });

})

);

router.get('/about', (req, res) => {

// about.ejs를 렌더링 하는데 mainLayout 레이아웃으로 감싸기

res.render('about', {layout: mainLayout});

});

// "/upload" 라우트

// router.get(

// "/upload",

// asynchandler(async (req, res) => {

// const locals = {

// title: "이미지 업로드",

// };

// res.render("addImage", { layout: mainLayout });

// })

// );

// GET /add

// Admin - Add Post

router.get(

"/upload",

asynchandler(async (req, res) => {

const locals = {

title: "이미지 업로드",

};

res.render("addImage", {

locals,

layout: mainLayout,

});

})

);

// router.get(

// "/upload/:id",

// asynchandler(async (req, res) => {

// const image = await Image.findOne({ \_id: req.params.id }); // 데이터를 'post'로 가져옴

// if (!image) {

// return res.status(404).send("게시물을 찾을 수 없습니다.");

// }

// res.render("addImage", { image, layout: mainLayout }); // post 객체로 전달

// })

// );

// router.get(

// "/uploaded",

// asynchandler(async (req, res) => {

// const locals = {

// title: "이미지 업로드",

// };

// res.render("addImage", {

// locals,

// layout: mainLayout,

// });

// })

// );

// GET /post/:id

// 게시물 상세 보기

router.get(

"/post/:id",

asynchandler(async (req, res) => {

const post = await Post.findOne({ \_id: req.params.id }); // 데이터를 'post'로 가져옴

if (!post) {

return res.status(404).send("게시물을 찾을 수 없습니다.");

}

res.render("post", { post, layout: mainLayout }); // post 객체로 전달

})

);

module.exports = router;

admin > add.ejs

<a href="/allPosts">&larr; 뒤로</a>

<div class="admin-title">

<h2><%= locals.title %></h2>

</div>

<form action="/add" method="POST">

<label for="title"><b>제목</b></label>

<input type="text" placeholder="게시물 제목" name="title" id="title">

<label for="body"><b>내용</b></label>

<textarea placeholder="게시물 내용" name="body" cols="50" rows="10"></textarea>

<input type="submit" value="등록" class="btn">

</form>

admin > allPosts.ejs

<div class="admin-title">

<h2>

<%= locals.title %>

</h2>

<a href="/add" class="button">+ 새 게시물</a>

</div>

<ul class="admin-posts">

<% data.forEach(post=> { %>

<li>

<div>

<a href="/post/<%= post.\_id %>">

<%= post.title %>

</a>

<p class="time-text">작성 시간: <%= new Date(post.createdAt).toLocaleString() %>

</p>

</div>

<div class="admin-post-controls">

<a href="/edit/<%= post.\_id %>" class="btn">편집</a>

<form action="/delete/<%= post.\_id %>?\_method=DELETE" method="POST">

<input type="submit" value="삭제" class="btn-delree btn">

</form>

</div>

</li>

<% }) %>

</ul>

admin > edit.ejs  
  
<a href="/allPosts">&larr; 뒤로</a>

<div class="admin-title">

<h2><%= locals.title %></h2>

<form action="/delete/<%= data.\_id %>?\_method=DELETE" method="POST">

<input type="submit" value="삭제" class="btn-delete btn">

</form>

</div>

<form action="/edit/<%= data.\_id %>?\_method=PUT" method="POST">

<label for="title"><b>제목</b></label>

<input type="text" name="title" id="title" value="<%= data.title %>">

<label for="body"><b>Content</b></label>

<textarea name="body" id="body" cols="50" rows="10"><%= data.body %></textarea>

<input type="submit" value="수정" class="btn">

</form>

admin > index.ejs

<h3>로그인</h3>

<form action="/admin" method="POST">

<label for="username"><b>사용자 이름</b></label>

<input type="text" name="username" id="username" />

<label for="password"><b>비밀번호</b></label>

<input type="password" name="password" id="password" />

<input type="submit" value="로그인" class="btn" />

</form>

<!-- 관리자 등록 -->

<!-- <h3>등록</h3>

<form action="/register" method="POST">

<label for="username"><b>사용자 이름</b></label>

<input type="text" name="username" id="username" />

<label for="password"><b>비밀번호</b></label>

<input type="password" name="password" id="password" />

<input type="submit" value="등록" class="btn" />

</form> -->

layouts > admin-nologout.ejs

<!DOCTYPE html>

<html lang="ko">

<head>

<meta charset="UTF-8" />

<meta http-equiv="X-UA-Compatible" content="IE=edge" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<title><%= locals.title %></title>

<meta name="description" content="My first application using Node.js, Express and MongoDB" />

<link rel="stylesheet" href="/css/style.css" />

</head>

<body>

<div class="container">

<!-- 헤더 : 로고, 상단 메뉴, 로그인 -->

<header class="header">

<!-- 로고 -->

<a href="/" class="header-logo">관리자 페이지</a>

<!-- 상단 메뉴 -->

<nav class="header-nav">

<ul>

<li>

<a href="/home">Home</a>

</li>

<li>

<a href="/about">About</a>

</li>

</ul>

</nav>

</header>

<!-- 메인 : 실제 내용이 들어갈 부분 -->

<main class="main"> <%- body %> </main>

</div>

</body>

</html>

layouts > admin.ejs

<!DOCTYPE html>

<html lang="ko">

<head>

<meta charset="UTF-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title><%= locals.title %></title>

<meta name="description" content="My first application using Node.js, Express and MongoDB">

<link rel="stylesheet" href="/css/style.css">

</head>

<body>

<div class="container">

<!-- 헤더 : 로고, 상단 메뉴, 로그인 -->

<header class="header">

<!-- 로고 -->

<a href="/" class="header-logo">Admin Page</a>

<!-- 상단 메뉴 -->

<nav class="header-nav">

<ul>

<li>

<a href="/home">Home</a>

</li>

<li>

<a href="/about">About</a>

</li>

</ul>

</nav>

<!-- 관리자 로그아웃 -->

<div class="header-button">

<a href="/logout">관리자 로그아웃</a>

</div>

</header>

<!-- 메인 : 실제 내용이 들어갈 부분 -->

<main class="main">

<%- body %>

</main>

</div>

</body>

</html>

layouts > main.ejs

<!DOCTYPE html>

<html lang="ko">

<head>

<meta charset="UTF-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title><%= locals.title %></title>

<meta name="description" content="My first application using Node.js, Express and MongoDB">

<link rel="stylesheet" href="/css/style.css">

</head>

<body>

<div class="container">

<!-- 헤더 : 로고, 상단 메뉴, 로그인 -->

<header class="header">

<!-- 로고 -->

<a href="/" class="header-logo">{ Geek Code }</a>

<!-- 상단 메뉴 -->

<nav class="header-nav">

<ul>

<li>

<a href="/home">{ Home }</a>

</li>

<li>

<a href="/about">{ About }</a>

</li>

<li>

<a href="/upload">{ Upload }</a>

</li>

</ul>

</nav>

<!-- 관리자 로그인 -->

<div class="header-button">

<a href="/admin">관리자 로그인</a>

</div>

</header>

<!-- 메인 : 실제 내용이 들어갈 부분 -->

<main class="main">

<%- body %>

</main>

</div>

</body>

</html>

addImage.ejs

<h1><%= locals.title %></h1>

<br>

<!-- 업로드 폼 -->

<form action="/upload" method="post" enctype="multipart/form-data">

<label for="file">파일선택:</label>

<input type="file" id="file" name="file" required />

<label for="title">제목:</label>

<input type="text" id="title" name="title" placeholder="제목">

<label for="body">내용:</label>

<textarea name="body" id="body" placeholder="내용"></textarea>

<button type="submit">업로드</button>

</form>

<br>

<!-- 업로드된 파일 정보가 있을 경우 표시 -->

<% if (typeof uploadedFile !== 'undefined') { %>

<h2>업로드된 파일:</h2>

<p>제목: <%= uploadedFile.title %></p>

<p>내용: <%= uploadedFile.body %></p>

<p>파일명: <%= uploadedFile.filename %></p>

<p>파일 경로: <%= uploadedFile.path %></p>

<!-- 이미지 미리보기 -->

<h3>업로드된 이미지:</h3>

<img src="/uploads/<%= uploadedFile.filename %>" alt="Uploaded Image" style="max-width: 500px;">

<% } %>

index.ejs

<!-- 상단 소개글, 히어로 이미지 -->

<div class="top">

<h1 class="top-heading">괴짜 코드</h1>

<p class="top-body">코드를 진행하며 손수 경험한 기괴한 코드를 공유합니다.</p>

</div>

<img src="/img/banner.jpg" alt="메인이미지" class="hero-image">

<!-- 최근 게시물 -->

<section class="articles">

<h2 class="articles-heading">최근 게시물</h2>

<ul class="article-ul">

<% data.forEach(post => { %> <!-- 반복문 시작 -->

<li>

<a href="/post/<%= post.\_id %>">

<span><%= post.title %></span>

<span class="article-list-date">

<%= post.createdAt.toDateString() %>

</span>

</a>

</li>

<% }) %> <!-- 반복문 종료 -->

</ul>

</section>

post.ejs

<h1><%= post.title %></h1>

<p>작성일: <%= post.createdAt.toDateString() %></p>

<article class="article">

<%= post.body %>

</article>

<a href="/home">← 뒤로가기</a>

app.js

require('dotenv').config({ path: './path/to/.env' });

const express = require("express");

const expressEjsLayouts = require("express-ejs-layouts");

const connectDb = require("./config/db"); // DB 연결 함수 가져오기

const cookieParser = require("cookie-parser"); // 쿠키 파서 가져오기

const methodOverride = require("method-override")

// const mainLayout = "../views/layouts/main.ejs";

// 이미지 업로드 관련

const multer = require('multer');

const Image = require('./models/Images');

const path = require('path');

const fs = require('fs');

const app = express();

const port = process.env.PORT || 3001 // .env에 포트가 없으면 3001번 포트로 설정

connectDb(); // DB 연결 함수 실행

app.use(express.json()); // JSON 요청 본문 파싱

app.use(express.urlencoded({extended: true})); // URL-encoded 요청 본문 파싱

app.use(methodOverride("\_method"));

/\*\*

\* /upload 라우트를 위한 layout 지정

\* 기본적으로 expressEjsLayouts는 layout을 찾으려 하므로

\* POST시 /views/layout.ejs를 찾으려 헀을 것이다.

\*

\* 하지만 layout.ejs는 존재하지 않기 때문에 기본 layout경로를 설정해줄 필요가 있다.

\* main.js에서 한 번에 관리하는 것은 지속적으로 실패 중

\*/

app.set('layout', 'layouts/main');

app.use(expressEjsLayouts); // ejs 레이아웃 사용

app.use(cookieParser()); // 쿠키 파서 사용

app.set("view engine", "ejs"); // ejs 사용

app.set("views", path.join(\_\_dirname, "/views")); // views 폴더 설정

app.use(express.static(\_\_dirname + "/public")); // public 폴더 설정

app.use("/", require("./routes/main")); // 루트 경로로 들어오면 main.js로 이동

app.use("/upload", require("./routes/main"));

app.use("/", require("./routes/admin")); // /admin 경로로 들어오면 admin.js로 이동

// 업로드 폴더 확인 후 생성

const uploadDir = path.join(\_\_dirname, 'uploads');

if (!fs.existsSync(uploadDir)) {

fs.mkdirSync(uploadDir);

}

// Multer 저장 설정 (디스크 저장 방식)

const storage = multer.diskStorage({

destination: function (req, file, cb) { // callback

cb(null, uploadDir);

},

filename: function(req, file, cb) {

cb(null, Date.now() + path.extname(file.originalname));

// 파일 이름을 현재 시간과 확장사로 지정

}

});

// Multer 미들웨어 설정

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

// 업로드된 파일을 정적 파일로 제공

app.use('/uploads', express.static(uploadDir));

// 파일 업로드 폼을 EJS 템플릿으로 렌더링

app.get('/upload', (req, res) => {

// views/index.ejs 파일을 렌더링

res.render('addImage');

});

// 파일 업로드를 처리하는 라우터

app.post('/upload', upload.single('file'), (req, res) => {

if (!req.file) {

return res.status(400).send('파일이 업로드 되지 않았습니다.');

}

const { title, body } = req.body;

console.log('파일 정보:', req.file);

console.log('제목:', title);

console.log('내용:', body);

// MongoDB

const newImage = new Image({

title: title,

body: body,

filename: req.file.filename,

path: path.join('uploads', req.file.filename),

});

newImage.save()

.then(() => {

console.log('파일 업로드 완료');

// EJS에 파일과 제목, 내용을 모두 전달

res.render('addImage', {

uploadedFile: {

title,

body,

filename: req.file.filename,

path: req.file.path

}

});

})

.catch((err) => {

console.error('오류:', err);

res.status(500).send('파일 업로드 오류');

});

});

app.listen(port, () => {

console.log(`Server is running on http://localhost:${port}`);

});