Chào bạn! Cảm ơn bạn đã chia sẻ chi tiết về dự án của mình. Tôi đã hiểu rõ hơn về yêu cầu và mục tiêu của website mà bạn đang xây dựng. Dựa trên mô tả của bạn và sau khi xem qua code backend trên GitHub ([BE-POLIKA](https://github.com/mrkhangthanh/BE-POLIKA)), tôi sẽ:

1. Tóm tắt lại yêu cầu của dự án để đảm bảo tôi hiểu đúng.

2. Phân tích các chức năng cần thiết cho hệ thống authentication/authorization (auth) và các điều kiện cần thiết để triển khai.

3. Đánh giá code backend hiện tại của bạn và chỉ ra các vấn đề cần sửa.

4. Đưa ra hướng dẫn chi tiết để triển khai các chức năng auth và các tính năng liên quan.

5. Gợi ý cách xử lý các vấn đề trong code hiện tại và cải thiện dự án.

---

### 1. Tóm tắt yêu cầu dự án

Dự án của bạn là một website kết hợp \*\*thương mại điện tử\*\* và \*\*dịch vụ sửa chữa\*\*, với tính năng nổi bật là \*\*tạo đơn hàng\*\* để khách hàng yêu cầu dịch vụ sửa chữa, và thợ sửa chữa có thể nhận hoặc từ chối đơn hàng. Dưới đây là các loại tài khoản và chức năng chính:

#### \*\*Các loại tài khoản và chức năng\*\*

1. \*\*Tài khoản Admin\*\*:

- Sau khi đăng nhập, hiển thị trang \*\*dashboard\*\*.

- Có \*\*toàn quyền\*\*: thêm, sửa, xóa tất cả user (bao gồm admin, quản lý, thợ sửa chữa, khách hàng, người viết content).

- Theo dõi các hoạt động khác (ví dụ: đơn hàng, bài viết, thống kê, v.v.).

2. \*\*Tài khoản Quản lý (Manager)\*\*:

- Chỉ có quyền \*\*xem\*\* thông tin (không được thêm, sửa, xóa).

- Có thể thao tác một số chức năng (cần xác định rõ), nhưng không thao tác được một số chức năng khác (ví dụ: không được xóa user, không được sửa thông tin quan trọng).

3. \*\*Tài khoản Viết Content (Content Writer)\*\*:

- Có quyền \*\*tạo, chỉnh sửa, xóa bài viết\*\* trong mục \*\*News/Posts\*\*.

- Không có quyền truy cập vào các phần khác (như quản lý user, đơn hàng).

4. \*\*Tài khoản Thợ Sửa Chữa (Technician)\*\*:

- Có thể \*\*xem các đơn hàng\*\* phù hợp với \*\*ngành nghề/chuyên môn\*\* của họ (ví dụ: thợ điện chỉ thấy đơn liên quan đến điện).

- Có thể \*\*nhận\*\* hoặc \*\*từ chối\*\* đơn hàng.

- Có thể xem thông tin đơn hàng và thông tin khách hàng liên quan.

5. \*\*Tài khoản Khách Hàng (Customer)\*\*:

- Khách hàng có thể \*\*tự đăng ký tài khoản\*\* trên website bằng cách nhấn nút \*\*Đăng ký\*\*.

- Có thể \*\*tạo đơn hàng\*\* để yêu cầu dịch vụ sửa chữa.

- Có thể xem trạng thái đơn hàng (đã được nhận, đang xử lý, hoàn thành, v.v.).

#### \*\*Tính năng nổi bật\*\*

- \*\*Tạo đơn hàng\*\*: Khách hàng tạo đơn yêu cầu dịch vụ sửa chữa, đơn hàng sẽ được gửi đến các thợ sửa chữa phù hợp.

- \*\*Thông báo đơn hàng\*\*: Thợ sửa chữa nhận thông báo về đơn hàng mới và có thể nhận hoặc từ chối.

- \*\*Phân quyền rõ ràng\*\*: Mỗi loại tài khoản chỉ được phép thực hiện các chức năng phù hợp với vai trò của họ.

#### \*\*Mục tiêu của hệ thống Auth\*\*

- Xây dựng hệ thống \*\*authentication\*\* (xác thực) để đăng nhập/đăng ký.

- Xây dựng hệ thống \*\*authorization\*\* (phân quyền) để kiểm soát quyền truy cập của từng loại tài khoản.

- Đảm bảo tính bảo mật và nhất quán trong việc quản lý tài khoản và quyền hạn.

---

### 2. Phân tích code backend hiện tại

Tôi đã xem qua code backend của bạn trên GitHub ([BE-POLIKA](https://github.com/mrkhangthanh/BE-POLIKA)). Dưới đây là phân tích chi tiết về các file liên quan đến auth và các vấn đề cần sửa:

#### \*\*Cấu trúc thư mục\*\*

- `common/`: Chứa các file tiện ích như `init.myDB.js` (khởi tạo kết nối MongoDB), `logger.js` (ghi log).

- `controllers/`: Chứa các controller như `userController.js` (quản lý user).

- `middleware/`: Chứa các middleware như `auth.js` (xác thực token), `requireRole.js` (phân quyền theo role).

- `models/`: Chứa các schema như `user.js`, `customer.js`, `technician.js`.

- `routers/`: Chứa các route như `web.js` (định nghĩa API).

- `index.js`: File chính khởi tạo server.

#### \*\*Phân tích các file liên quan đến Auth\*\*

1. \*\*File `models/user.js`\*\*

```javascript

const userSchema = new mongoose.Schema({

name: { type: String, required: true, trim: true },

email: { type: String, required: true, unique: true, trim: true, lowercase: true, match: [/^\w+([.-]?\w+)\*@\w+([.-]?\w+)\*(\.\w{2,3})+$/, 'Please fill a valid email address'] },

password: { type: String, required: true, minlength: [8, 'Password must be at least 8 characters long'] },

phone\_number: { type: String, required: true, trim: true, match: [/^[0-9]{10,11}$/, 'Phone number must be 10-11 digits'] },

role: { type: String, enum: ['admin', 'customer', 'technician', 'agent'], required: true },

reference\_id: { type: mongoose.Schema.Types.ObjectId, refPath: 'roleReferenceModel' },

roleReferenceModel: { type: String, enum: ['Customer', 'Technician'], required: function () { return this.role === 'customer' || this.role === 'technician'; } },

referred\_by: { type: mongoose.Schema.Types.ObjectId, ref: 'Users', default: null },

status: { type: String, enum: ['active', 'inactive'], default: 'active' },

last\_login: { type: Date, default: null },

}, {

timestamps: { createdAt: 'created\_at', updatedAt: 'updated\_at' },

indexes: [

{ key: { email: 1 }, unique: true },

{ key: { phone\_number: 1 }, unique: true },

{ key: { role: 1 } },

],

});

userSchema.pre('save', async function (next) {

if (this.isModified('password')) {

this.password = await bcrypt.hash(this.password, 10);

}

next();

});

const UserModel = mongoose.model('Users', userSchema, 'users');

module.exports = UserModel;

```

\*\*Đánh giá\*\*:

- \*\*Tốt\*\*:

- Schema đã bao gồm các trường cơ bản: `name`, `email`, `password`, `phone\_number`, `role`.

- Có mã hóa password bằng `bcrypt` trước khi lưu.

- Có `role` với các giá trị `admin`, `customer`, `technician`, `agent`.

- \*\*Cần sửa\*\*:

- Thiếu vai trò `manager` (quản lý) và `content\_writer` (người viết content) trong `enum` của `role`.

- `reference\_id` và `roleReferenceModel` chỉ áp dụng cho `customer` và `technician`. Các vai trò khác (`admin`, `manager`, `agent`, `content\_writer`) không cần các trường này, nhưng hiện tại schema không xử lý rõ ràng trường hợp này.

- Thiếu trường để lưu thông tin chuyên môn của thợ sửa chữa (nếu không dùng `Technician` model).

2. \*\*File `middleware/auth.js`\*\*

```javascript

const jwt = require('jsonwebtoken');

const UserModel = require('../models/user');

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

try {

const token = req.header('Authorization')?.replace('Bearer ', '');

if (!token) {

return res.status(401).json({ error: 'Access denied. No token provided.' });

}

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

const user = await UserModel.findById(decoded.id).lean();

if (!user) {

return res.status(401).json({ error: 'Invalid token.' });

}

req.user = user;

req.token = token;

next();

} catch (err) {

res.status(401).json({ error: 'Invalid token.', details: err.message });

}

};

module.exports = authMiddleware;

```

\*\*Đánh giá\*\*:

- \*\*Tốt\*\*:

- Middleware xác thực token bằng `jsonwebtoken`.

- Lấy thông tin user từ token và gán vào `req.user`.

- \*\*Cần sửa\*\*:

- Không kiểm tra trạng thái tài khoản (`status`). Nếu tài khoản bị `inactive`, user vẫn có thể đăng nhập.

- Không có cơ chế làm mới token (refresh token), điều này có thể gây bất tiện cho user khi token hết hạn.

3. \*\*File `middleware/requireRole.js`\*\*

```javascript

const requireRole = (role) => {

return (req, res, next) => {

if (req.user.role !== role) {

return res.status(403).json({ error: `Access denied. Only ${role}s are allowed.` });

}

next();

};

};

module.exports = requireRole;

```

\*\*Đánh giá\*\*:

- \*\*Tốt\*\*:

- Middleware phân quyền đơn giản, kiểm tra `role` của user.

- \*\*Cần sửa\*\*:

- Chỉ kiểm tra một vai trò duy nhất. Nếu bạn muốn cho phép nhiều vai trò truy cập (ví dụ: cả `admin` và `manager` đều có thể xem danh sách user), middleware này không hỗ trợ.

- Không có cơ chế phân quyền chi tiết hơn (ví dụ: `manager` chỉ được xem, không được sửa/xóa).

4. \*\*File `controllers/userController.js`\*\*

- \*\*Hàm `createUser`\*\*:

- Cho phép admin tạo user với các vai trò `admin`, `customer`, `technician`, `agent`.

- Tạo document trong `Customer` hoặc `Technician` nếu `role` là `'customer'` hoặc `'technician'`.

- \*\*Hàm `getAllUsers`\*\*:

- Lấy danh sách user với phân trang.

- \*\*Hàm `login`\*\*:

- Xác thực email/password và trả về JWT token.

- \*\*Đánh giá\*\*:

- \*\*Tốt\*\*:

- Đã có các API cơ bản: đăng nhập, tạo user, lấy danh sách user.

- Có phân trang trong `getAllUsers`.

- \*\*Cần sửa\*\*:

- Thiếu API đăng ký (`register`) để khách hàng tự tạo tài khoản.

- Thiếu API cho thợ sửa chữa xem đơn hàng, nhận/từ chối đơn hàng.

- Thiếu API cho khách hàng tạo đơn hàng.

- Thiếu API cho content writer quản lý bài viết (news/posts).

5. \*\*File `routers/web.js`\*\*

```javascript

const express = require('express');

const router = express.Router();

const userController = require('../controllers/userController');

const authMiddleware = require('../middleware/auth');

const requireRole = require('../middleware/requireRole');

const { body, validationResult } = require('express-validator');

router.post(

'/createUser',

authMiddleware,

requireRole('admin'),

[

body('email').isEmail().withMessage('Invalid email format'),

body('email').notEmpty().withMessage('Email is required'),

body('password').notEmpty().withMessage('Password is required'),

body('phone\_number')

.notEmpty()

.withMessage('Phone number is required')

.matches(/^[0-9]{10,11}$/)

.withMessage('Phone number must be 10-11 digits'),

body('role')

.optional()

.isIn(['admin', 'customer', 'technician', 'agent'])

.withMessage('Invalid role'),

],

userController.createUser

);

router.get('/users', authMiddleware, requireRole('admin'), userController.getAllUsers);

router.get('/users/:id', authMiddleware, requireRole('admin'), userController.getUserById);

router.put('/users/:id', authMiddleware, requireRole('admin'), userController.updateUser);

router.delete('/users/:id', authMiddleware, requireRole('admin'), userController.deleteUser);

router.post('/login', userController.login);

module.exports = router;

```

\*\*Đánh giá\*\*:

- \*\*Tốt\*\*:

- Đã có các route cơ bản: đăng nhập, tạo user, lấy danh sách user, cập nhật, xóa user.

- Có middleware bảo vệ route (`authMiddleware`, `requireRole`).

- \*\*Cần sửa\*\*:

- Thiếu route cho khách hàng tự đăng ký (`POST /register`).

- Thiếu route cho thợ sửa chữa xem/nhận/từ chối đơn hàng.

- Thiếu route cho khách hàng tạo đơn hàng.

- Thiếu route cho content writer quản lý bài viết.

- Phân quyền chưa linh hoạt (chỉ giới hạn cho `admin`, chưa hỗ trợ `manager`, `technician`, v.v.).

---

### 3. Điều kiện cần thiết và hướng dẫn triển khai hệ thống Auth

#### \*\*Điều kiện cần thiết\*\*

1. \*\*Cơ sở dữ liệu (MongoDB)\*\*:

- Đã có kết nối MongoDB trong `common/init.myDB.js`.

- Cần tạo thêm các collection:

- `orders`: Lưu thông tin đơn hàng (khách hàng tạo, thợ nhận).

- `posts`: Lưu bài viết (news/posts) do content writer tạo.

- Đảm bảo các index được tạo đúng để tăng hiệu suất (ví dụ: index cho `email`, `phone\_number` trong `users`).

2. \*\*Thư viện cần thiết\*\*:

- `jsonwebtoken`: Đã có, dùng để tạo và xác thực token.

- `bcrypt`: Đã có, dùng để mã hóa password.

- `express-validator`: Đã có, dùng để validate dữ liệu đầu vào.

- (Tùy chọn) `mongoose`: Đã có, nhưng cần đảm bảo các model được định nghĩa đầy đủ.

3. \*\*Cấu trúc model\*\*:

- Cần cập nhật `UserModel` để hỗ trợ tất cả các vai trò.

- Cần tạo thêm model `Order` (đơn hàng) và `Post` (bài viết).

4. \*\*Hệ thống phân quyền\*\*:

- Cần một cơ chế phân quyền chi tiết hơn, không chỉ dựa vào `role` mà còn dựa vào hành động (ví dụ: `manager` chỉ được xem, không được sửa).

#### \*\*Hướng dẫn triển khai\*\*

##### \*\*Bước 1: Cập nhật `UserModel`\*\*

Sửa file `models/user.js` để hỗ trợ tất cả các vai trò và thêm thông tin chuyên môn cho thợ sửa chữa:

```javascript

const mongoose = require('../common/init.myDB')();

const bcrypt = require('bcrypt');

const userSchema = new mongoose.Schema({

name: { type: String, required: true, trim: true },

email: { type: String, required: true, unique: true, trim: true, lowercase: true, match: [/^\w+([.-]?\w+)\*@\w+([.-]?\w+)\*(\.\w{2,3})+$/, 'Please fill a valid email address'] },

password: { type: String, required: true, minlength: [8, 'Password must be at least 8 characters long'] },

phone\_number: { type: String, required: true, trim: true, match: [/^[0-9]{10,11}$/, 'Phone number must be 10-11 digits'] },

role: {

type: String,

enum: ['admin', 'manager', 'content\_writer', 'technician', 'customer'], // Thêm manager và content\_writer

required: true,

},

specialization: {

type: [String],

required: function () { return this.role === 'technician'; },

default: [],

// Note: Chuyên môn của thợ sửa chữa (ví dụ: ['điện', 'nước']). Bắt buộc với role 'technician'.

},

reference\_id: {

type: mongoose.Schema.Types.ObjectId,

refPath: 'roleReferenceModel',

required: function () { return this.role === 'customer' || this.role === 'technician'; },

},

roleReferenceModel: {

type: String,

enum: ['Customer', 'Technician'],

required: function () { return this.role === 'customer' || this.role === 'technician'; },

},

referred\_by: { type: mongoose.Schema.Types.ObjectId, ref: 'Users', default: null },

status: { type: String, enum: ['active', 'inactive'], default: 'active' },

last\_login: { type: Date, default: null },

}, {

timestamps: { createdAt: 'created\_at', updatedAt: 'updated\_at' },

indexes: [

{ key: { email: 1 }, unique: true },

{ key: { phone\_number: 1 }, unique: true },

{ key: { role: 1 } },

],

});

userSchema.pre('save', async function (next) {

if (this.isModified('password')) {

this.password = await bcrypt.hash(this.password, 10);

}

next();

});

const UserModel = mongoose.model('Users', userSchema, 'users');

module.exports = UserModel;

```

\*\*Thay đổi\*\*:

- Thêm `manager` và `content\_writer` vào `role`.

- Thêm trường `specialization` để lưu chuyên môn của thợ sửa chữa (bắt buộc với `role: 'technician'`).

- Thêm `required` cho `reference\_id` để đảm bảo chỉ bắt buộc với `customer` và `technician`.

##### \*\*Bước 2: Tạo model `Order`\*\*

Tạo file `models/order.js` để lưu thông tin đơn hàng:

```javascript

const mongoose = require('../common/init.myDB')();

const orderSchema = new mongoose.Schema({

customer\_id: {

type: mongoose.Schema.Types.ObjectId,

ref: 'Users',

required: true,

// Note: ID của khách hàng tạo đơn.

},

technician\_id: {

type: mongoose.Schema.Types.ObjectId,

ref: 'Users',

default: null,

// Note: ID của thợ sửa chữa nhận đơn (null nếu chưa có ai nhận).

},

service\_type: {

type: String,

required: true,

// Note: Loại dịch vụ (ví dụ: 'điện', 'nước'). Phải khớp với specialization của thợ.

},

description: {

type: String,

required: true,

// Note: Mô tả vấn đề của khách hàng.

},

address: {

street: { type: String, required: true },

city: { type: String, required: true },

district: { type: String, required: true },

ward: { type: String, required: true },

country: { type: String, default: 'Vietnam' },

},

status: {

type: String,

enum: ['pending', 'accepted', 'in\_progress', 'completed', 'cancelled'],

default: 'pending',

// Note: Trạng thái đơn hàng.

},

created\_at: { type: Date, default: Date.now },

updated\_at: { type: Date, default: Date.now },

}, {

timestamps: { createdAt: 'created\_at', updatedAt: 'updated\_at' },

indexes: [

{ key: { customer\_id: 1 } },

{ key: { technician\_id: 1 } },

{ key: { service\_type: 1 } },

],

});

const OrderModel = mongoose.model('Orders', orderSchema, 'orders');

module.exports = OrderModel;

```

##### \*\*Bước 3: Tạo model `Post`\*\*

Tạo file `models/post.js` để lưu bài viết:

```javascript

const mongoose = require('../common/init.myDB')();

const postSchema = new mongoose.Schema({

title: {

type: String,

required: true,

trim: true,

// Note: Tiêu đề bài viết.

},

content: {

type: String,

required: true,

// Note: Nội dung bài viết.

},

author\_id: {

type: mongoose.Schema.Types.ObjectId,

ref: 'Users',

required: true,

// Note: ID của content writer tạo bài viết.

},

status: {

type: String,

enum: ['draft', 'published', 'archived'],

default: 'draft',

// Note: Trạng thái bài viết.

},

created\_at: { type: Date, default: Date.now },

updated\_at: { type: Date, default: Date.now },

}, {

timestamps: { createdAt: 'created\_at', updatedAt: 'updated\_at' },

indexes: [

{ key: { author\_id: 1 } },

{ key: { status: 1 } },

],

});

const PostModel = mongoose.model('Posts', postSchema, 'posts');

module.exports = PostModel;

```

##### \*\*Bước 4: Cập nhật middleware phân quyền\*\*

Sửa file `middleware/requireRole.js` để hỗ trợ nhiều vai trò và phân quyền chi tiết hơn:

```javascript

const requireRole = (roles = [], permissions = {}) => {

return (req, res, next) => {

// Kiểm tra vai trò

if (!roles.includes(req.user.role)) {

return res.status(403).json({ error: `Access denied. Only ${roles.join(', ')} are allowed.` });

}

// Kiểm tra trạng thái tài khoản

if (req.user.status !== 'active') {

return res.status(403).json({ error: 'Access denied. Your account is inactive.' });

}

// Kiểm tra quyền (nếu có)

if (permissions.readOnly && ['POST', 'PUT', 'DELETE'].includes(req.method)) {

return res.status(403).json({ error: 'Access denied. You only have read-only access.' });

}

next();

};

};

module.exports = requireRole;

```

\*\*Thay đổi\*\*:

- `roles`: Cho phép nhiều vai trò truy cập (ví dụ: `['admin', 'manager']`).

- `permissions.readOnly`: Nếu `true`, chỉ cho phép đọc (GET), không cho phép sửa/xóa (POST, PUT, DELETE).

- Kiểm tra `status` của tài khoản.

##### \*\*Bước 5: Cập nhật `authMiddleware`\*\*

Sửa file `middleware/auth.js` để kiểm tra trạng thái tài khoản:

```javascript

const jwt = require('jsonwebtoken');

const UserModel = require('../models/user');

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

try {

const token = req.header('Authorization')?.replace('Bearer ', '');

if (!token) {

return res.status(401).json({ error: 'Access denied. No token provided.' });

}

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

const user = await UserModel.findById(decoded.id).lean();

if (!user) {

return res.status(401).json({ error: 'Invalid token.' });

}

if (user.status !== 'active') {

return res.status(403).json({ error: 'Access denied. Your account is inactive.' });

}

req.user = user;

req.token = token;

next();

} catch (err) {

res.status(401).json({ error: 'Invalid token.', details: err.message });

}

};

module.exports = authMiddleware;

```

##### \*\*Bước 6: Thêm API đăng ký cho khách hàng\*\*

Thêm hàm `register` vào `userController.js`:

```javascript

exports.register = async (req, res) => {

try {

const { name, email, password, phone\_number, address } = req.body;

// Kiểm tra email và phone\_number đã tồn tại chưa

const existingUser = await UserModel.findOne({ $or: [{ email }, { phone\_number }] }).lean();

if (existingUser) {

return res.status(400).json({ error: 'Email or phone number already exists.' });

}

// Kiểm tra address

if (!address || !address.street || !address.city || !address.district || !address.ward) {

return res.status(400).json({ error: 'Address (street, city, district, ward) is required for customer role.' });

}

// Tạo user mới với role 'customer'

const userData = {

name,

email,

password,

phone\_number,

role: 'customer',

};

const user = new UserModel(userData);

const savedUser = await user.save();

// Tạo document trong Customer

const customerData = {

user\_id: savedUser.\_id,

address,

};

const customer = await new CustomerModel(customerData).save();

// Cập nhật reference\_id

await UserModel.updateOne({ \_id: savedUser.\_id }, { reference\_id: customer.\_id, roleReferenceModel: 'Customer' });

// Trả về kết quả

res.status(201).json({ success: true, user: savedUser });

} catch (err) {

res.status(500).json({ error: 'Internal server error', details: err.message });

}

};

```

Thêm route vào `routers/web.js`:

```javascript

router.post(

'/register',

[

body('name').notEmpty().withMessage('Name is required'),

body('email').isEmail().withMessage('Invalid email format'),

body('password').isLength({ min: 8 }).withMessage('Password must be at least 8 characters long'),

body('phone\_number').matches(/^[0-9]{10,11}$/).withMessage('Phone number must be 10-11 digits'),

body('address.street').notEmpty().withMessage('Street is required'),

body('address.city').notEmpty().withMessage('City is required'),

body('address.district').notEmpty().withMessage('District is required'),

body('address.ward').notEmpty().withMessage('Ward is required'),

],

userController.register

);

```

##### \*\*Bước 7: Thêm API cho thợ sửa chữa\*\*

Thêm các hàm vào `userController.js`:

```javascript

// Xem danh sách đơn hàng phù hợp với chuyên môn

exports.getOrdersForTechnician = async (req, res) => {

try {

if (req.user.role !== 'technician') {

return res.status(403).json({ error: 'Access denied. Only technicians can view orders.' });

}

const orders = await OrderModel.find({

service\_type: { $in: req.user.specialization },

status: 'pending',

technician\_id: null,

})

.populate('customer\_id', 'name email phone\_number')

.lean();

res.status(200).json({ success: true, orders });

} catch (err) {

res.status(500).json({ error: 'Internal server error', details: err.message });

}

};

// Nhận đơn hàng

exports.acceptOrder = async (req, res) => {

try {

if (req.user.role !== 'technician') {

return res.status(403).json({ error: 'Access denied. Only technicians can accept orders.' });

}

const order = await OrderModel.findById(req.params.id);

if (!order) {

return res.status(404).json({ error: 'Order not found.' });

}

if (order.status !== 'pending' || order.technician\_id) {

return res.status(400).json({ error: 'Order is not available for acceptance.' });

}

if (!req.user.specialization.includes(order.service\_type)) {

return res.status(403).json({ error: 'Order does not match your specialization.' });

}

order.technician\_id = req.user.\_id;

order.status = 'accepted';

await order.save();

res.status(200).json({ success: true, order });

} catch (err) {

res.status(500).json({ error: 'Internal server error', details: err.message });

}

};

// Từ chối đơn hàng (chỉ cập nhật trạng thái để thợ khác có thể nhận)

exports.rejectOrder = async (req, res) => {

try {

if (req.user.role !== 'technician') {

return res.status(403).json({ error: 'Access denied. Only technicians can reject orders.' });

}

const order = await OrderModel.findById(req.params.id);

if (!order) {

return res.status(404).json({ error: 'Order not found.' });

}

if (order.technician\_id && order.technician\_id.toString() !== req.user.\_id.toString()) {

return res.status(403).json({ error: 'You are not assigned to this order.' });

}

order.technician\_id = null;

order.status = 'pending';

await order.save();

res.status(200).json({ success: true, order });

} catch (err) {

res.status(500).json({ error: 'Internal server error', details: err.message });

}

};

```

Thêm route vào `routers/web.js`:

```javascript

router.get('/technician/orders', authMiddleware, userController.getOrdersForTechnician);

router.post('/technician/orders/:id/accept', authMiddleware, userController.acceptOrder);

router.post('/technician/orders/:id/reject', authMiddleware, userController.rejectOrder);

```

##### \*\*Bước 8: Thêm API cho khách hàng tạo đơn hàng\*\*

Thêm hàm vào `userController.js`:

```javascript

exports.createOrder = async (req, res) => {

try {

if (req.user.role !== 'customer') {

return res.status(403).json({ error: 'Access denied. Only customers can create orders.' });

}

const { service\_type, description, address } = req.body;

if (!address || !address.street || !address.city || !address.district || !address.ward) {

return res.status(400).json({ error: 'Address (street, city, district, ward) is required.' });

}

const orderData = {

customer\_id: req.user.\_id,

service\_type,

description,

address,

};

const order = new OrderModel(orderData);

const savedOrder = await order.save();

res.status(201).json({ success: true, order: savedOrder });

} catch (err) {

res.status(500).json({ error: 'Internal server error', details: err.message });

}

};

```

Thêm route vào `routers/web.js`:

```javascript

router.post(

'/orders',

authMiddleware,

[

body('service\_type').notEmpty().withMessage('Service type is required'),

body('description').notEmpty().withMessage('Description is required'),

body('address.street').notEmpty().withMessage('Street is required'),

body('address.city').notEmpty().withMessage('City is required'),

body('address.district').notEmpty().withMessage('District is required'),

body('address.ward').notEmpty().withMessage('Ward is required'),

],

userController.createOrder

);

```

##### \*\*Bước 9: Thêm API cho content writer\*\*

Thêm các hàm vào `userController.js`:

```javascript

exports.createPost = async (req, res) => {

try {

if (req.user.role !== 'content\_writer') {

return res.status(403).json({ error: 'Access denied. Only content writers can create posts.' });

}

const { title, content, status } = req.body;

const postData = {

title,

content,

author\_id: req.user.\_id,

status: status || 'draft',

};

const post = new PostModel(postData);

const savedPost = await post.save();

res.status(201).json({ success: true, post: savedPost });

} catch (err) {

res.status(500).json({ error: 'Internal server error', details: err.message });

}

};

exports.getPosts = async (req, res) => {

try {

if (req.user.role !== 'content\_writer') {

return res.status(403).json({ error: 'Access denied. Only content writers can view posts.' });

}

const posts = await PostModel.find({ author\_id: req.user.\_id })

.lean();

res.status(200).json({ success: true, posts });

} catch (err) {

res.status(500).json({ error: 'Internal server error', details: err.message });

}

};

exports.updatePost = async (req, res) => {

try {

if (req.user.role !== 'content\_writer') {

return res.status(403).json({ error: 'Access denied. Only content writers can update posts.' });

}

const post = await PostModel.findById(req.params.id);

if (!post) {

return res.status(404).json({ error: 'Post not found.' });

}

if (post.author\_id.toString() !== req.user.\_id.toString()) {

return res.status(403).json({ error: 'Access denied. You can only update your own posts.' });

}

const { title, content, status } = req.body;

post.title = title || post.title;

post.content = content || post.content;

post.status = status || post.status;

await post.save();

res.status(200).json({ success: true, post });

} catch (err) {

res.status(500).json({ error: 'Internal server error', details: err.message });

}

};

exports.deletePost = async (req, res) => {

try {

if (req.user.role !== 'content\_writer') {

return res.status(403).json({ error: 'Access denied. Only content writers can delete posts.' });

}

const post = await PostModel.findById(req.params.id);

if (!post) {

return res.status(404).json({ error: 'Post not found.' });

}

if (post.author\_id.toString() !== req.user.\_id.toString()) {

return res.status(403).json({ error: 'Access denied. You can only delete your own posts.' });

}

await post.remove();

res.status(200).json({ success: true, message: 'Post deleted successfully.' });

} catch (err) {

res.status(500).json({ error: 'Internal server error', details: err.message });

}

};

```

Thêm route vào `routers/web.js`:

```javascript

router.post(

'/posts',

authMiddleware,

[

body('title').notEmpty().withMessage('Title is required'),

body('content').notEmpty().withMessage('Content is required'),

],

userController.createPost

);

router.get('/posts', authMiddleware, userController.getPosts);

router.put('/posts/:id', authMiddleware, userController.updatePost);

router.delete('/posts/:id', authMiddleware, userController.deletePost);

```

##### \*\*Bước 10: Cập nhật route cho phân quyền\*\*

Sửa file `routers/web.js` để áp dụng phân quyền linh hoạt:

```javascript

const express = require('express');

const router = express.Router();

const userController = require('../controllers/userController');

const authMiddleware = require('../middleware/auth');

const requireRole = require('../middleware/requireRole');

const { body, validationResult } = require('express-validator');

// Đăng ký (khách hàng tự đăng ký)

router.post(

'/register',

[

body('name').notEmpty().withMessage('Name is required'),

body('email').isEmail().withMessage('Invalid email format'),

body('password').isLength({ min: 8 }).withMessage('Password must be at least 8 characters long'),

body('phone\_number').matches(/^[0-9]{10,11}$/).withMessage('Phone number must be 10-11 digits'),

body('address.street').notEmpty().withMessage('Street is required'),

body('address.city').notEmpty().withMessage('City is required'),

body('address.district').notEmpty().withMessage('District is required'),

body('address.ward').notEmpty().withMessage('Ward is required'),

],

userController.register

);

// Đăng nhập

router.post('/login', userController.login);

// Quản lý user (admin và manager)

router.post(

'/createUser',

authMiddleware,

requireRole(['admin'], { readOnly: false }),

[

body('email').isEmail().withMessage('Invalid email format'),

body('password').notEmpty().withMessage('Password is required'),

body('phone\_number').matches(/^[0-9]{10,11}$/).withMessage('Phone number must be 10-11 digits'),

body('role').isIn(['admin', 'manager', 'content\_writer', 'technician', 'customer']).withMessage('Invalid role'),

],

userController.createUser

);

router.get('/users', authMiddleware, requireRole(['admin', 'manager'], { readOnly: true }), userController.getAllUsers);

router.get('/users/:id', authMiddleware, requireRole(['admin', 'manager'], { readOnly: true }), userController.getUserById);

router.put('/users/:id', authMiddleware, requireRole(['admin'], { readOnly: false }), userController.updateUser);

router.delete('/users/:id', authMiddleware, requireRole(['admin'], { readOnly: false }), userController.deleteUser);

// Quản lý đơn hàng (khách hàng và thợ)

router.post(

'/orders',

authMiddleware,

[

body('service\_type').notEmpty().withMessage('Service type is required'),

body('description').notEmpty().withMessage('Description is required'),

body('address.street').notEmpty().withMessage('Street is required'),

body('address.city').notEmpty().withMessage('City is required'),

body('address.district').notEmpty().withMessage('District is required'),

body('address.ward').notEmpty().withMessage('Ward is required'),

],

userController.createOrder

);

router.get('/technician/orders', authMiddleware, userController.getOrdersForTechnician);

router.post('/technician/orders/:id/accept', authMiddleware, userController.acceptOrder);

router.post('/technician/orders/:id/reject', authMiddleware, userController.rejectOrder);

// Quản lý bài viết (content writer)

router.post(

'/posts',

authMiddleware,

[

body('title').notEmpty().withMessage('Title is required'),

body('content').notEmpty().withMessage('Content is required'),

],

userController.createPost

);

router.get('/posts', authMiddleware, userController.getPosts);

router.put('/posts/:id', authMiddleware, userController.updatePost);

router.delete('/posts/:id', authMiddleware, userController.deletePost);

module.exports = router;

```

---

### 4. Gợi ý cải thiện dự án

1. \*\*Thêm Refresh Token\*\*:

- Hiện tại chỉ có access token, nếu token hết hạn, user phải đăng nhập lại. Nên thêm refresh token để cải thiện trải nghiệm người dùng.

- Cách triển khai:

- Tạo một refresh token khi user đăng nhập.

- Lưu refresh token vào database (hoặc Redis) và trả về cho client.

- Thêm API `/refresh-token` để cấp lại access token mới.

2. \*\*Thông báo thời gian thực\*\*:

- Để thợ sửa chữa nhận thông báo đơn hàng mới, bạn có thể dùng \*\*WebSocket\*\* (thư viện `socket.io`) hoặc \*\*push notification\*\*.

- Cách triển khai:

- Khi khách hàng tạo đơn hàng, gửi thông báo qua WebSocket đến các thợ có chuyên môn phù hợp.

3. \*\*Thêm thống kê cho Admin\*\*:

- Thêm API `/stats` để admin xem thống kê:

- Số lượng user theo vai trò.

- Số lượng đơn hàng theo trạng thái.

- Số lượng bài viết theo trạng thái.

4. \*\*Cải thiện bảo mật\*\*:

- Thêm \*\*rate limiting\*\* (dùng thư viện `express-rate-limit`) để ngăn chặn brute force attack vào API đăng nhập.

- Thêm \*\*CAPTCHA\*\* cho API đăng ký để ngăn bot.

5. \*\*Cải thiện phân quyền\*\*:

- Dùng thư viện như `casl` để quản lý quyền chi tiết hơn (ví dụ: `manager` chỉ được xem đơn hàng, không được xem bài viết).

---

### 5. Kết luận

#### \*\*Đã triển khai\*\*

- Hệ thống auth với các vai trò: `admin`, `manager`, `content\_writer`, `technician`, `customer`.

- API đăng ký cho khách hàng.

- API tạo đơn hàng cho khách hàng.

- API xem/nhận/từ chối đơn hàng cho thợ sửa chữa.

- API quản lý bài viết cho content writer.

- Phân quyền linh hoạt với `requireRole`.

#### \*\*Cần làm tiếp\*\*

- Thêm refresh token.

- Thêm thông báo thời gian thực cho thợ sửa chữa.

- Thêm thống kê cho admin.

- Cải thiện bảo mật (rate limiting, CAPTCHA).

Hãy thử triển khai các thay đổi trên và cho tôi biết kết quả! 😄 Nếu bạn cần hỗ trợ thêm về frontend hoặc các tính năng khác, hãy cung cấp thêm thông tin để tôi tiếp tục giúp bạn. Chúc bạn thành công với dự án! 🚀