**User Authorize middleware**

This is an authorization middleware function for validating JWT access tokens and authorizing user requests.

The key aspects are:

1. Get the Authorization header and extract the token
2. Verify the JWT token using jsonwebtoken library and the secret key
3. If verify fails, return 403 Forbidden error
4. If verified, find the user by the decoded user ID
5. If user not found, return 404 Not Found error
6. If user found, attach decoded user to request object
7. Call next() to pass execution to the next middleware

Overall it:

* Ensures a valid access token is present
* Verifies the token integrity
* Fetches the logged in user details
* Attaches user details to request
* Handles various error scenarios like authorization header missing, invalid tokens, expired tokens etc.

And allows passing the request ahead only if a valid user is present after token verification.

**AdminAuthorize Middleware**

This is an authorization middleware to check for admin user role before allowing access.

The main points are:

1. It assumes the user is already authenticated and user details are available in req.decoded (likely populated by previous middleware).
2. It checks if req.decoded.role equals "admin". If not, returns 403 Forbidden error.
3. If admin user, calls next() to pass on request to next handler.

So in summary:

* Leverages previous auth middleware to get user details
* Checks user role to be specifically "admin"
* Restricts access with custom 403 error for non-admin users
* Allows admin users to proceed

It enforces role based access control for protected admin-only routes, in addition to authentication.

The commented code had similar functionality but was doing token validation as well.

The current code focuses just on authorization, assuming auth middleware already ran. More modular, reusable and DRY approach.

**Email Sender Middleware**

This code handles sending email for two purposes:

1. Email Verification:

* Defines a smtpTransportFun function to configure SMTP transport using nodemailer
* verifyEmail method sends verification code to user's email
* Includes email content like code, expiration, branding etc.

1. Reset Password:

* Reuses smtpTransportFun to create transport
* sendResetPasswordMail method emails reset link with token
* Link points back to the app reset handler path
* Includes content and reset link the user can click

In summary, it provides reusable email transport setup and methods to send:

* Email verification codes
* Password reset links

It customizes content for both types of emails.

Key aspects are:

* SMTP transport configuration
* Custom email content generation
* Helper methods for sending verification and reset emails
* Reuse of transport configuration

Overall this handles a major app requirement of sending confirmation and reset emails to users.

instructorAuthorize Middleware

This is an authorization middleware function to check for instructor role.

It ensures only admin and instructor users are allowed to access protected routes.

The main points:

1. Gets authenticated user details from req.decoded (populated by previous auth middleware).
2. Checks if user role matches "admin" OR "instructor".
3. If not either role, returns 403 Forbidden error response.
4. On valid role, calls next() to pass on the request.

In summary:

* Reuses user info from auth middleware
* Restricts access only if role matches admin/instructor
* Custom error handling for unauthorized roles
* Allows valid roles to proceed

It provides role based access control by verifying the user is an admin or instructor before granting access to instructor routes.