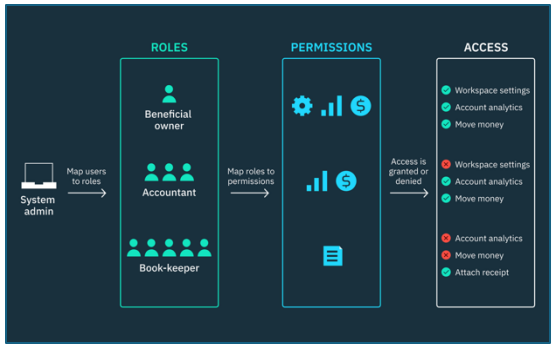
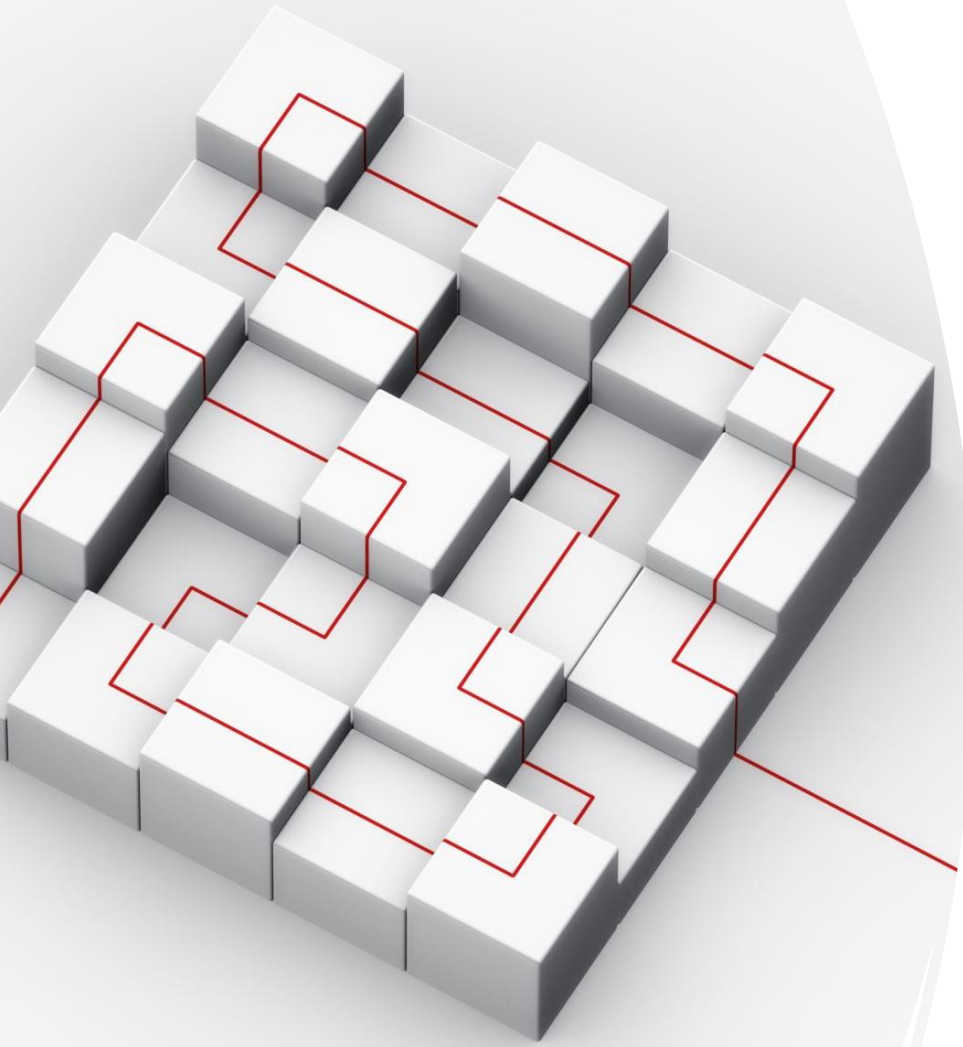
INT 161

Basic Backend Development

JWT-Based Authorization with Role-Based Access Control (RBAC)





Unit Objectives

- After completing this unit, you should be able to:
  - Differentiate between Authentication and Authorization
  - Explain the core principles of Authorization
  - Integrate JWT-based Role and Permission Checking
  - Implement Authorization Middleware in Express.js

What is Authorization?

- Determines what actions a user is allowed to perform
- Comes after authentication
- Example:
  - Authentication → “Who are you?”
  - Authorization → “What are you allowed to do?”

Common Authorization Models

- Role-Based Access Control (RBAC)
  - Users are assigned roles (e.g., admin, user, editor)
  - Users are granted permissions based **on predefined** roles.
  - User → Assigned Role
    - Role → Has specific permissions
    - System access → Checked by the user's role
- Permission-Based Access Control (PBAC)
  - Users have specific permissions (fine-grained)
  - Example: book:create, book:delete, order:view, user:update
- Attribute-Based Access Control (ABAC)
  - Based on attributes (user, resource, environment)
  - Use Policy Rule for grant permissions:

```
ALLOW if
 user.department == "Finance"
 AND resource.type == "Invoice"
 AND request.time < 6pm
```

Why Use JWT with Role-Based Authorization?

- ✅ Stateless & Scalable - No server-side sessions required.
- ✅ Fine-Grained Access Control - Different roles (e.g., USER, ADMIN).
- ✅ Secure - Signed & expired tokens prevent misuse.
- ✅ Microservices Ready - Can be shared across services.

JWT Authentication + Role-Based Access Flow

1. User logs in → Sends username & password.
2. Backend validates credentials → Generates a JWT token with roles.
3. Client stores JWT (e.g., HttpOnly cookie, localStorage).
4. Client sends JWT → API uses role-based authorization.
5. Backend verifies JWT → Allows or denies access based on roles.

Implementation Flow in Express

Client → [JWT Token in Header] → Middleware → Controller → Resource

1. Verify JWT
2. Decode user info (id, role, permissions)
3. Check role/permission before accessing route
4. If unauthorized → return 403 Forbidden

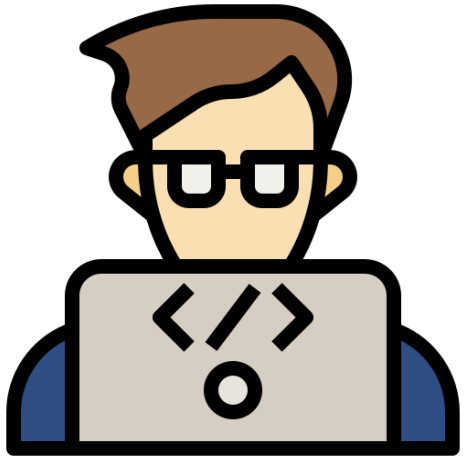
Sample Role/Authorizations

Role	Resource	Permission
ADMIN	films	ALL
	auth	LOGIN,LOGOUT
MANAGER	customers	ALL
	films	UPDATE
	auth	LOGIN,LOGOUT
USER	films	GET
	auth	LOGIN,LOGOUT
EVERY ROLES	countries	ALL
	auth	REGISTER,VERIFY-EMAIL

Sample Role/Authorizations

URI	Method	Permit For
/auth/logout	POST	Authenticated User
/auth/*	POST	All
/customers/*	*	MANAGER
/countries/*	*	All
/films/*	GET	Authenticated user
/films	POST	ADMIN
	PUT	ADMIN, MANAGER
	DELETE	ADMIN

Practices



Refresh Tokens

- Extend authentication without requiring login.
  - User logs in → Receives access token & refresh token.
  - When the access token expires, the client requests a new one using the refresh token.
  - Backend validates the refresh token and issues a new access token.
  - Refresh tokens expire after a longer period (e.g., 7 days).

Modify Auth Service (1/2) : auth-service.js

```
export const login = async (data) => {
   const { email, password } = data;
   const user = await repo.findByEmail(email);
   if (!user) throw errResp.unauthorizedError("Invalid email or password");
   if (!user.active) throw errResp.unauthorizedError("User is not active");
   const valid = await argon2.verify(user.password, password);
   if (!valid) throw errResp.unauthorizedError("Invalid email or password");

   user.password = undefined;
   user.tokenType = "REFRESH_TOKEN";
   const refreshToken = await generateToken(user,"7d");

   user.tokenType = "ACCESS_TOKEN";
   const accessToken = await generateToken(user);

   return { access_token: accessToken, refresh_token: refreshToken };
}
```

Modify Auth Service (2/2) : auth-service.js

```
export async function validUserToken(userFromToken) {
   const existingUser = await repo.findById(userFromToken.id);

   if (! existingUser) throw errResp.unauthorizedError("Invalid Token, user not found");
   if(! existingUser.active && userFromToken.tokenType != 'VERIFY_EMAIL_TOKEN’)
       throw errResp.unauthorizedError(`Invalid Token, user is not active`);
   if(existingUser.email !== userFromToken.email)
       throw errResp.unauthorizedError(`Invalid Token, email mismatch`);
   return existingUser;
}
```

```
export async function refreshToken(userFromToken) {
   const user = await this.validUserToken(userFromToken);
   user.tokenType = "ACCESS_TOKEN";
   user.password = undefined;
   const accessToken = await generateToken(user);
   return accessToken;
}
```

Modify Auth Controller (1/2) : auth-controller.js

```
export async function login(req, res) {
    const user = req.body;
    // console.log(user);
    const {access_token, refresh_token} = await authService.login(user);
    await addCookie(res,refresh_token);
    res.status(200).json({access_token: access_token});
};

async function addCookie(res, refreshToken) {
    res.cookie("refresh_token", refreshToken,{
        httpOnly: true, // This makes the cookie inaccessible to client-side JavaScript
        secure: process.env.NODE_ENV === 'production', // Recommended: set to true in production for HTTPS
        maxAge: 7*24*60*60*1000, // Cookie expires in 7 days (in milliseconds)
        sameSite: 'Strict' // Or 'Lax' for enhanced security against CSRF
    });
}
```

Modify Auth Controller (2/2) : auth-controller.js

```
export async function refreshToken(req, res) {
    const token = req.cookies.refresh_token;
    // console.log('refresh_token:',token);
    if (!token) throw errResp.unauthorizedError("Refresh Token is required");
    const user = await verifyToken(token);
    const accessToken = await authService.refreshToken(user);
    res.status(200).json({access_token: accessToken});
}

export async function logout(req, res) {
    res.clearCookie("refresh_token");
    res.status(204).end();
}
```

Add Auth middleware (1/2) : ./middlewares/auth.js

```
import 'dotenv/config';
import * as jwtUtil from "../utils/jwt.js";
import errResp from "../errors/error-response.js";

export const authorize = (... roles) => {
   return (req, res, next) => {
      if (!roles.includes(req.user.role)) {
         const error = errResp.forbiddenError('Access denied for role: ' + req.user.role);
         next(error);
      }
      next();
   };
};
```

error-response.js

```
forbiddenError: function (message) {
   const  err = new Error(message);
   err.code = 'FORBIDDEN_ERROR';
   err.status = 403;
   return err;
}
```

Add Auth middleware (2/2) : ./middlewares/auth.js

```
export const authenticate = async (req, res, next) => {
  const authHeader = req.headers.authorization;

  if (!authHeader || !authHeader.startsWith("Bearer ")) {
    const err =
      errResp.unauthorizedError('No token provided');
    return next(err);
  }
```

```
try {
    const token = authHeader.split(" ")[1];
    const claims = await jwtUtil.verifyToken(token);
    if (!claims) {
      const error =
          errResp.unauthorizedError("Invalid token");
      return next(error)
    } else {
      req.user = claims;
      return next();
    }
  } catch (err) {
    err.status = 401;
    return next(err);
  }
};
```

Modify Auth Router : auth-route.js

```
var express = require('express');
var router = express.Router();
const controller = require('../controllers/auth-controller');
const {authenticate} = require("../middlewares/auth");

router.post('/register', controller.register);
router.post('/verify-email', controller.verify);
router.post('/login', controller.login);
router.post('/refresh', controller.refreshToken);
router.post('/logout', authenticate, controller.logout);

module.exports = router;
```

Modify App : app.js

```
app.use('/auth', authRouter);
app.use('/customers', authenticate, authorize('MANAGER'),customerRouter);
app.use('/countries', countryRouter);
app.use('/films', authenticate,filmRouter);
```

Modify Auth Router : film-route.js

```
const express = require('express');
const router = express.Router();
const controller = require('../controllers/film-controller');
const {validate} = require('../validators/validate');
const {filmSchema, filmQuerySchema} =require('../validators/film-validator');
const {authorize} = require("../middlewares/auth");

router.get('/', validate(filmQuerySchema, 'query'),controller.list);
router.get('/:id', controller.get);
router.put('/:id', authorize('ADMIN','MANAGER'), controller.update);
router.delete('/:id', authorize('ADMIN'), controller.delete);
router.post('/', authorize('ADMIN'),validate(filmSchema, 'body'),
controller.create);

module.exports = router;
```