Day-05: Authentication and Authorization (Node.js + Express)

1. Introduction

Day-05 focuses on implementing **Authentication** and **Authorization** in a Node.js application. These concepts are essential for securing web applications and controlling user access.

This day builds on previous CRUD and Express concepts and introduces **user identity verification** and **access control**.

2. What is Authentication?

Authentication is the process of verifying **who a user is**.

Examples:

- Login using email & password
- JWT-based login
- Session-based login

If authentication succeeds → user is logged in.
If it fails → access is denied.

3. What is Authorization?

Authorization determines **what an authenticated user is allowed to do**.

Examples:

- Only logged-in users can create data
- Only admin users can delete records
- Users can edit only their own data

Authentication comes **first**, authorization comes **after**.

4. Tech Stack Used

- **Node.js**
- **Express.js**
- **MongoDB**
- **Mongoose**
- **JWT (JSON Web Token)**
- **bcrypt** (for password hashing)
- **EJS** (for UI rendering)

5. Authentication Flow (JWT Based)

1. User registers with email & password
2. Password is hashed using bcrypt
3. User data is stored in MongoDB
4. User logs in with credentials
5. Server verifies credentials
6. JWT token is generated
7. Token is sent to browser (cookie/localStorage)
8. Token is verified on protected routes

6. Password Hashing (Security)

Passwords are **never stored as plain text**.

Why hashing?

- Prevents password leaks
- Increases application security

bcrypt converts passwords into hashed format before saving them in MongoDB.

7. JWT (JSON Web Token)

JWT is used for **stateless authentication**.

JWT contains:

- Header
- Payload (user info)
- Signature

JWT is verified on every protected request.

8. Middleware for Authentication

Middleware checks whether the user is logged in.

Purpose:

- Verify JWT token
- Allow or deny access

If token is valid → request continues
If token is invalid → error or redirect

9. Authorization Middleware

Authorization ensures role-based or permission-based access.

Examples:

- Admin-only routes
- Owner-only access

Authorization middleware runs **after authentication**.

10. Protected Routes

Protected routes are accessible only to authenticated users.

Examples:

- Dashboard
- Profile page
- Create / Edit data

Unauthenticated users are redirected to the login page.

11. Error Handling

- Invalid credentials
- Token expired
- Unauthorized access

Custom error messages improve user experience and debugging.

12. Folder Structure (Recommended)

```
project-root/
│
├── models/
│   └── user.js
├── routes/
│   ├── auth.js
│   └── protected.js
├── middleware/
│   └── authMiddleware.js
├── views/
│   ├── login.ejs
│   ├── register.ejs
│   └── dashboard.ejs
├── public/
│   └── style.css
├── app.js
```

```
└── package.json
```

13. Learning Outcomes

- Understood authentication vs authorization
- Implemented JWT-based login
- Used bcrypt for password security
- Protected routes using middleware
- Learned real-world backend security concepts

14. Future Enhancements

- Refresh tokens
- Role-based access control (RBAC)
- OAuth (Google, GitHub login)
- Rate limiting
- Email verification

15. Conclusion

Day-05 covers one of the most important backend topics: **Authentication and Authorization**. Mastering these concepts is essential for building secure and scalable web applications.

This implementation provides a strong foundation for real-world projects and advanced backend development.