SECTION 9 — Identity Layer Implementation

Register, Login, JWT, Access Token, Refresh Token, Logout, Login Attempts & Account Lock

1. Purpose of the Identity Layer

The Identity Layer is responsible for:

- User registration
- User authentication
- Token generation
- Access control foundation
- Session security

It answers two core questions:

1. Who is the user?
2. Is the user authenticated?

Identity handles authentication. Authorization will be handled later using RBAC.

2. Identity Module Structure

```
modules/identity/
 ├── models/
 │     └── user.model.js
 ├── auth.routes.js
 ├── auth.controller.js
 ├── auth.service.js
```

Each file has a defined responsibility.

3. User Model Design

The user model defines how user data is stored in MongoDB.

Example:

```
import mongoose from "mongoose";

const userSchema = new mongoose.Schema({
  email: {
    type: String,
    required: true,
    unique: true
  },
  password: {
    type: String,
    required: true
  },
```

```
  phone: {
    type: String
  },
  roles: {
    type: [String],
    default: ["CUSTOMER"]
  },
  loginAttempts: {
    type: Number,
    default: 0
  },
  lockUntil: {
    type: Date
  }
}, { timestamps: true });

export default mongoose.model("User", userSchema);
```

Important Fields

- roles: Determines user permissions later.
- loginAttempts: Tracks failed login attempts.
- lockUntil: Temporarily locks account after too many failures.

This prepares the system for security enforcement.

4. Register Implementation

Endpoint

POST /api/auth/register

Workflow

```
Route
  ↓
Controller
  ↓
Service
  ↓
Hash password
  ↓
Save user
  ↓
Send response
```

Service Logic (Conceptual)

1. Check if email already exists.
2. Hash password using password utility.
3. Create user in database.
4. Return success response.

Why Hash Password?

Passwords must never be stored in plain text.

Hashing ensures:

- Database compromise does not expose real passwords.
- Secure authentication process.

5. Login Implementation

Endpoint

POST /api/auth/login

Workflow

```
Controller
  ↓
Find user by email
  ↓
Check if account locked
  ↓
Compare password
  ↓
Generate tokens
  ↓
Reset loginAttempts
  ↓
Send response
```

6. Access Token vs Refresh Token

This is a critical concept.

Access Token

- Short-lived (e.g., 15 minutes)
- Sent in Authorization header

- Used to access protected routes
- Contains user identity

Example header:

Authorization: Bearer <accessToken>

Refresh Token

- Long-lived (e.g., 7 days)
- Stored in HTTP-only cookie
- Used to generate new access token
- Not exposed to JavaScript

Why Two Tokens?

If access token is compromised:

- It expires quickly.

Refresh token:

- Allows secure token renewal
- Reduces need for repeated login
- Improves user experience

This approach balances:

- Security
- Scalability
- User convenience

7. How JWT Authentication Works

Login:

1. User submits credentials.
2. Backend verifies credentials.
3. Backend generates access token.
4. Backend generates refresh token.
5. Tokens sent to client.

Accessing protected route:

1. Client sends access token.
2. protect middleware verifies token.
3. If valid → request continues.
4. If invalid → 401 Unauthorized.

8. Protect Middleware (Authentication Middleware)

Purpose:

- Verify JWT
- Attach user to request

Example flow:

```
Request
  ↓
Extract Authorization header
  ↓
Verify token
  ↓
Decode payload
  ↓
Attach user to req.user
  ↓
Continue to next()
```

Without valid token, access is denied.

9. Login Attempts and Account Lock

Enterprise applications must prevent brute-force attacks.

Why Track Login Attempts?

If someone repeatedly tries wrong passwords:

- It may be an attack.
- System should temporarily lock account.

Implementation Logic

1. Each failed login increments loginAttempts.
2. If loginAttempts reaches threshold (e.g., 5):
    - Set lockUntil = current time + 15 minutes.
3. If lockUntil is in future:
    - Reject login.
4. On successful login:
    - Reset loginAttempts to 0.
    - Clear lockUntil.

Example Logic

```
if (user.lockUntil && user.lockUntil > Date.now()) {
  throw new AppError("Account locked. Try later.", 403);
}

if (!passwordMatch) {
  user.loginAttempts += 1;

  if (user.loginAttempts >= 5) {
    user.lockUntil = Date.now() + 15 * 60 * 1000;
  }

  await user.save();
  throw new AppError("Invalid credentials", 401);
}
```

This improves security significantly.

10. Logout Implementation

Endpoint

POST /api/auth/logout

Logout process:

- Clear refresh token cookie.
- Optionally invalidate token in database (if tracking).

Access tokens automatically expire.

Logout ensures refresh token cannot be reused.

11. Identity Flow Summary

Registration Flow

User → Register → Hash Password → Save → Success Response

Login Flow

User → Login

→ Check Lock

→ Compare Password

→ Generate Tokens

→ Reset Attempts

→ Send Response

Protected Route Flow

Request → protect middleware → Verify Token → Continue

12. Security Principles Applied

Identity layer enforces:

- Password hashing
- Short-lived access tokens
- Refresh token rotation
- Account lock mechanism
- Role-based identity storage
- No sensitive data exposure

These are enterprise-grade security practices.

13. What Students Must Understand

1. Authentication verifies identity.
2. Authorization verifies permissions.
3. Passwords must be hashed.
4. JWT enables stateless authentication.
5. Access tokens and refresh tokens serve different purposes.
6. Login attempts prevent brute-force attacks.
7. Middleware enforces security.

14. Summary

By the end of this section, students should understand:

- How registration works internally
- How login works internally
- How JWT tokens are generated and verified
- Why access and refresh tokens both exist
- How protect middleware secures routes
- How login attempt lock increases security

This completes the Identity Layer foundation.