🔐Authentication System — JWT + Express + MongoDB

A complete **authentication system** built using **Node.js, Express, MongoDB, JWT, and bcrypt**, following real-world backend security practices.

This project covers **user registration, login, JWT-based authentication, protected routes, error handling, and security testing**.

🗜️Features

- User Signup with password hashing
- User Login with JWT token generation
- JWT-based authentication (stateless)
- Protected routes using middleware
- Token expiration handling
- Invalid / missing token handling
- Secure error responses
- Real-world testing mindset

🛠️Tech Stack

- **Node.js**
- **Express.js**
- **MongoDB + Mongoose**
- **JWT (jsonwebtoken)**
- **bcryptjs**

🔦Authentication Flow

```
Register → Login → Receive Token → Access Protected Routes → Handle Errors
```

🔐JWT Token Overview

JWT (JSON Web Token) is used for **secure and stateless authentication**.

Token Contains:

- User ID
- User Email
- Expiration Time

The server does not store sessions, making the system scalable.

🧾 Token Creation Example

```
jwt.sign(
  { id: user._id, email: user.email },
  process.env.JWT_SECRET,
  { expiresIn: "1h" }
);
```

Token Expiration Options

- `5m` → Testing
- `15m` → Production access token
- `1h` → Standard usage
- `1d` → Long sessions

Shorter expiration improves security.

🛡️ Protected Routes (Middleware)

```
const authHeader = req.headers.authorization;

if (!authHeader || !authHeader.startsWith("Bearer ")) {
  return res.status(401).json({ message: "No token provided" });
}

const token = authHeader.split(" ")[1];
const decoded = jwt.verify(token, process.env.JWT_SECRET);
req.user = decoded;
next();
```

- Valid token → Access granted
- Invalid / expired token → Access denied

🧪 Testing the Auth Flow

✂️ Signup

```
POST /api/signup
```

✔️ User saved to database

✒️ Login

```
POST /api/login
```

✔️ JWT token returned

🛏️ Protected Route

```
GET /api/dashboard
Authorization: Bearer <TOKEN>
```

✔️ Access granted if token is valid

🔩 Error Handling Scenarios

Scenario	Response
No token	401 Unauthorized
Invalid token	Invalid token
Expired token	Token expired, login again

🔓 Weak Authentication (What NOT to Do)

```
if (token) next();
```

❌ Easily bypassed by fake tokens

🔐Security Best Practices Applied

- Password hashing using bcrypt
- JWT verification using secret key
- Token expiration handling
- Clear error messages
- No blind trust in user input

Backend Security Mindset

Never trust the user. Always validate everything.

Most vulnerable systems fail due to: - Weak authentication - No validation - Poor error handling

This project focuses on preventing those mistakes.

📁Project Structure

```
config/
  db.js

middleware/
  authMiddleware.js

models/
  User.js

routes/
  authRoute.js

server.js
```

🔗Final Outcome

- Complete JWT authentication flow implemented
- Secure and scalable backend design
- Real-world error handling and testing
- Interview-ready authentication project

🧰One-Line Interview Summary

> *Built a complete JWT-based authentication system with secure login, protected routes, token expiration handling, and robust error management.*

🚯Week 3 Completed — Authentication & Security Fundamentals Mastered