Training Day 13 Report

Date: 10 July 2025

Project: MERN Notes App

Topic Covered: User Authentication with JWT (Signup & Login)

On the thirteenth day, we introduced user authentication so each user can manage their own notes securely. We implemented **Signup** and **Login** routes using password hashing (bcrypt) and JSON Web Tokens (JWT).

Concepts Learned

- **bcrypt.js** for securely hashing and comparing passwords.
- **jsonwebtoken (JWT)** for generating tokens that authenticate users.
- Tokens are returned on login/signup and stored on the client for authenticated requests.

User Model

In `backend/models/userModel.js`:

Listing 1: User Schema with bcrypt password hashing

```
const mongoose = require('mongoose');
const bcrypt = require('bcryptjs');

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

// Hash password before saving
userSchema.pre('save', async function(next) {
  if (!this.isModified('password')) return next();
  const salt = await bcrypt.genSalt(10);
  this.password = await bcrypt.hash(this.password, salt);
  next();
});

module.exports = mongoose.model('User', userSchema);
```

Signup Route

In `backend/routes/userRoutes.js`:

Listing 2: POST /api/users/signup – Register new user

```
router.post('/signup', async (req, res) => {
  const { name, email, password } = req.body;

  try {
    const existingUser = await User.findOne({ email });
    if (existingUser) {
      return res.status(400).json({ message: 'User already exists' });
    }

    const user = await User.create({ name, email, password });

    res.status(201).json({
      _id: user._id,
      name: user.name,
      email: user.email,
      token: jwt.sign({ id: user._id }, process.env.JWT_SECRET, {
          expiresIn: '30d' })
    });
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});
```

Login Route

Listing 3: POST /api/users/login – Authenticate user

```
router.post('/login', async (req, res) => {
  const { email, password } = req.body;

  try {
    const user = await User.findOne({ email });
    if (user && await bcrypt.compare(password, user.password)) {
      res.json({
        _id: user._id,
        name: user.name,
        email: user.email,
        token: jwt.sign({ id: user._id }, process.env.JWT_SECRET, {
            expiresIn: '30d' })
      });
    } else {
      res.status(401).json({ message: 'Invalid email or password' });
    }
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});
```

Testing the Endpoints

- Used Postman to test `POST /api/users/signup` with name, email, and password.

- Verified that a new user was created in MongoDB and a token was returned.
- Tested `POST /api/users/login` with the same credentials to confirm authentication.
- Confirmed that incorrect credentials return a `401 Unauthorized`.

Outcome of the Day

- Learned how to implement secure password storage with bcrypt.
- Successfully created Signup and Login endpoints with JWT-based authentication.
- Application is now capable of identifying users and issuing tokens for secure access.