Node.js Express Authentication with React: Class Note

Learning Objectives

- Understand user authentication (register, login, profile, protected routes).
- Set up a Node.js Express server with MongoDB, Mongoose, and React as the view engine.
- Implement secure password hashing with `bcrypt`.
- Generate and verify JSON Web Tokens (JWT) for authentication.
- Use environment variables with `.env` for configuration.
- Organize backend code with separate controllers and routes.
- Create a React frontend with pages for Home, Login, Signup, and Profile.
- Handle asynchronous errors using `express-async-handler`.

Prerequisites

- Basic knowledge of JavaScript, Node.js, Express, and React.
- MongoDB installed locally or a MongoDB Atlas account.
- Node.js and npm installed.
- Code editor (e.g., VS Code).

Step 1: Project Setup

1. **Initialize a Node.js Project**:
   - Create a project folder and initialize:

```
mkdir auth-app          # Creates a new directory named auth-app
cd auth-app             # Changes the current directory to auth-app
npm init -y             # Initializes a new Node.js project with
default package.json
```

2. **Install Dependencies**:
   - Install backend and frontend dependencies:

```
npm install express mongoose bcryptjs jsonwebtoken dotenv express-
async-handler express-react-views react react-dom axios  # Installs
core dependencies
npm install --save-dev nodemon @babel/core @babel/preset-env
@babel/preset-react  # Installs dev dependencies for auto-restart and
React JSX
```

3. **Set Up Project Structure**:
   - Create the following folder structure:

```
auth-app/
├── controllers/          # Backend controllers
│   └── userController.js # User-related controller functions
├── routes/               # Backend routes
│   └── userRoutes.js     # User-related routes
├── models/               # Mongoose models
│   └── userModel.js      # User schema and model
├── middleware/           # Middleware functions
│   └── authMiddleware.js # Authentication middleware
├── config/               # Configuration files
│   └── db.js             # Database connection logic
├── views/                # React components
│   ├── Home.jsx          # Home page component
│   ├── Login.jsx         # Login page component
│   ├── Signup.jsx        # Signup page component
│   └── Profile.jsx       # Profile page component
├── public/               # Static assets
│   └── styles.css        # CSS for frontend
├── .env                  # Environment variables
├── .babelrc              # Babel configuration
├── server.js             # Main server file
└── package.json          # Project configuration
```

4. **Configure** `package.json`:
   - Add scripts to `package.json`:

```
"scripts": {
  "start": "node server.js",  // Runs the server in production
  "dev": "nodemon server.js"  // Runs the server with nodemon for
development
}
```

5. **Configure Babel**:
   - Create `.babelrc` in the root:

```
{
  "presets": ["@babel/preset-env", "@babel/preset-react"]  // Enables
ES modules and React JSX
}
```

Step 2: Environment Variables Setup

1. **Create `.env` File**:
   - In the root, create a `.env` file:

```
PORT=5000                          # Port number for the server
MONGO_URI=mongodb://localhost:27017/authdb  # MongoDB connection string
JWT_SECRET=your_jwt_secret_key     # Secret key for JWT (use a strong,
random string)
```

- Replace `MONGO_URI` with your MongoDB connection string.
- Generate `JWT_SECRET` (e.g., `node -e "console.log(require('crypto').randomBytes(64).toString('hex'))"`).

2. **Load Environment Variables**:
   - Loaded in `server.js` (shown in Step 8).

Step 3: Connect to MongoDB

1. **Create Database Connection**:
   - In `config/db.js`:

```
const mongoose = require('mongoose'); // Imports Mongoose for MongoDB

const connectDB = async () => { // Defines async function to connect to
MongoDB
  try { // Starts try block for error handling
    const conn = await mongoose.connect(process.env.MONGO_URI, { //
Connects to MongoDB
      useNewUrlParser: true, // Uses new URL parser
      useUnifiedTopology: true, // Uses new topology engine
    });
    console.log(`MongoDB Connected: ${conn.connection.host}`); // Logs
successful connection
  } catch (error) { // Catches connection errors
    console.error(`Error: ${error.message}`); // Logs error message
    process.exit(1); // Exits process with failure
  }
};

module.exports = connectDB; // Exports connectDB function
```

Step 4: Create User Model

1. **Define User Schema**:
   - In `models/userModel.js`:

```
const mongoose = require('mongoose'); // Imports Mongoose
const bcrypt = require('bcryptjs'); // Imports bcrypt for hashing

const userSchema = mongoose.Schema( // Defines user schema
```

```
  {
    name: { // Name field
      type: String, // String type
      required: [true, 'Please add a name'], // Required with error
message
    },
    email: { // Email field
      type: String, // String type
      required: [true, 'Please add an email'], // Required
      unique: true, // Ensures unique email
      match: [/.+\@.+\..+/, 'Please add a valid email'], // Validates
email format
    },
    password: { // Password field
      type: String, // String type
      required: [true, 'Please add a password'], // Required
      minlength: 6, // Minimum length
    },
  },
  { timestamps: true } // Adds timestamps
);

userSchema.pre('save', async function (next) { // Pre-save hook for
password hashing
  if (!this.isModified('password')) { // Skips if password not modified
    next(); // Proceeds to next middleware
  }
  const salt = await bcrypt.genSalt(10); // Generates salt
  this.password = await bcrypt.hash(this.password, salt); // Hashes
password
  next(); // Proceeds
});

userSchema.methods.matchPassword = async function (enteredPassword) {
// Method to compare passwords
  return await bcrypt.compare(enteredPassword, this.password); //
Compares passwords
};

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

Step 5: Create Controllers

1. **Create User Controller**:
   - In `controllers/userController.js`:

```
const asyncHandler = require('express-async-handler'); // Imports
asyncHandler
const jwt = require('jsonwebtoken'); // Imports JWT
```

```
const User = require('../models/userModel'); // Imports User model

const registerUser = asyncHandler(async (req, res) => { // Handles user
registration
  const { name, email, password } = req.body; // Destructures request
body
  if (!name || !email || !password) { // Validates input
    res.status(400); // Sets 400 status
    throw new Error('Please include all fields'); // Throws error
  }
  const userExists = await User.findOne({ email }); // Checks for
existing user
  if (userExists) { // If user exists
    res.status(400); // Sets 400 status
    throw new Error('User already exists'); // Throws error
  }
  const user = await User.create({ name, email, password }); // Creates
user
  if (user) { // If creation successful
    const token = generateToken(user._id); // Generates token
    res.status(201).json({ _id: user._id, name, email, token }); //
Sends response
  } else { // If creation failed
    res.status(400); // Sets 400 status
    throw new Error('Invalid user data'); // Throws error
  }
});

const loginUser = asyncHandler(async (req, res) => { // Handles user
login
  const { email, password } = req.body; // Destructures request body
  const user = await User.findOne({ email }); // Finds user by email
  if (user && (await user.matchPassword(password))) { // Validates
credentials
    const token = generateToken(user._id); // Generates token
    res.json({ _id: user._id, name: user.name, email, token }); //
Sends response
  } else { // If invalid
    res.status(401); // Sets 401 status
    throw new Error('Invalid email or password'); // Throws error
  }
});

const getMe = asyncHandler(async (req, res) => { // Gets current user
  res.status(200).json(req.user); // Sends user data
});

const generateToken = (id) => { // Generates JWT
  return jwt.sign({ id }, process.env.JWT_SECRET, { // Signs token
    expiresIn: '30d', // Sets expiration
  });
};
```

```
module.exports = { registerUser, loginUser, getMe }; // Exports
functions
```

Step 6: Create Authentication Middleware

1. **Protect Routes**:
   - In `middleware/authMiddleware.js`:

```
const jwt = require('jsonwebtoken'); // Imports JWT
const asyncHandler = require('express-async-handler'); // Imports
asyncHandler
const User = require('../models/userModel'); // Imports User model

const protect = asyncHandler(async (req, res, next) => { // Protects
routes
  let token; // Declares token variable
  if (req.headers.authorization &&
req.headers.authorization.startsWith('Bearer')) { // Checks for Bearer
token
    try { // Starts try block
      token = req.headers.authorization.split(' ')[1]; // Extracts
token
      const decoded = jwt.verify(token, process.env.JWT_SECRET); //
Verifies token
      req.user = await User.findById(decoded.id).select('-password');
// Finds user
      if (!req.user) { // If user not found
        res.status(401); // Sets 401 status
        throw new Error('Not authorized, user not found'); // Throws
error
      }
      next(); // Proceeds
    } catch (error) { // Catches errors
      res.status(401); // Sets 401 status
      throw new Error('Not authorized, token failed'); // Throws error
    }
  } else { // If no token
    res.status(401); // Sets 401 status
    throw new Error('Not authorized, no token'); // Throws error
  }
});

module.exports = { protect }; // Exports middleware
```

Step 7: Create Routes

1. **Define User Routes**:
   - In `routes/userRoutes.js`:

```
const express = require('express'); // Imports Express
const router = express.Router(); // Creates router
const { registerUser, loginUser, getMe } =
require('../controllers/userController'); // Imports controllers
const { protect } = require('../middleware/authMiddleware'); // Imports
middleware

router.post('/register', registerUser); // Registration route
router.post('/login', loginUser); // Login route
router.get('/me', protect, getMe); // Protected user data route

module.exports = router; // Exports router
```

Step 8: Set Up Express Server with React Views

1. **Create Server File**:
   - In `server.js`:

```
const express = require('express'); // Imports Express
const dotenv = require('dotenv').config(); // Loads .env
const connectDB = require('./config/db'); // Imports DB connection
const path = require('path'); // Imports path for file handling
const erv = require('express-react-views'); // Imports express-react-
views

connectDB(); // Connects to MongoDB

const app = express(); // Creates Express app

app.set('views', path.join(__dirname, 'views')); // Sets views
directory
app.set('view engine', 'jsx'); // Sets JSX as view engine
app.engine('jsx', erv.createEngine()); // Configures express-react-
views

app.use(express.json()); // Parses JSON
app.use(express.urlencoded({ extended: false })); // Parses URL-encoded
data
app.use(express.static(path.join(__dirname, 'public'))); // Serves
static files

app.use('/api/users', require('./routes/userRoutes')); // Mounts API
routes

app.get('/', (req, res) => res.render('Home')); // Renders Home page
app.get('/login', (req, res) => res.render('Login')); // Renders Login
page
app.get('/signup', (req, res) => res.render('Signup')); // Renders
Signup page
```

```
app.get('/profile', (req, res) => res.render('Profile')); // Renders
Profile page

const PORT = process.env.PORT || 5000; // Sets port
app.listen(PORT, () => console.log(`Server running on port ${PORT}`));
// Starts server
```

Step 9: Create React Components

1. **Home Page**:
   - In `views/Home.jsx`:

```
import React from 'react'; // Imports React

const Home = () => { // Defines Home component
  return ( // Returns JSX
    <div className="container">
      <h1>Welcome to the Auth App</h1> {/* Page title */}
      <p>Navigate to <a href="/login">Login</a> or <a
href="/signup">Signup</a>.</p> {/* Navigation links */}
    </div>
  );
};

export default Home; // Exports component
```

2. **Login Page**:
   - In `views/Login.jsx`:

```
import React, { useState } from 'react'; // Imports React and useState
import axios from 'axios'; // Imports axios for API calls

const Login = () => { // Defines Login component
  const [email, setEmail] = useState(''); // State for email
  const [password, setPassword] = useState(''); // State for password
  const [error, setError] = useState(''); // State for error message

  const handleSubmit = async (e) => { // Handles form submission
    e.preventDefault(); // Prevents default form behavior
    try { // Starts try block
      const res = await axios.post('/api/users/login', { email,
password }); // Sends login request
      localStorage.setItem('token', res.data.token); // Stores token
      window.location.href = '/profile'; // Redirects to profile
    } catch (err) { // Catches errors
      setError(err.response?.data?.message || 'Login failed'); // Sets
error message
    }
```

```
  };

  return ( // Returns JSX
    <div className="container">
      <h2>Login</h2> {/* Page title */}
      {error && <p className="error">{error}</p>} {/* Displays error
*/}
      <form onSubmit={handleSubmit}> {/* Form with submit handler */}
        <div>
          <label>Email:</label> {/* Email label */}
          <input
            type="email"
            value={email}
            onChange={(e) => setEmail(e.target.value)} // Updates email
state
            required
          />
        </div>
        <div>
          <label>Password:</label> {/* Password label */}
          <input
            type="password"
            value={password}
            onChange={(e) => setPassword(e.target.value)} // Updates
password state
            required
          />
        </div>
        <button type="submit">Login</button> {/* Submit button */}
      </form>
      <p>Don't have an account? <a href="/signup">Signup</a></p> {/*
Signup link */}
    </div>
  );
};

export default Login; // Exports component
```

3. **Signup Page**:
   - In `views/Signup.jsx`:

```
import React, { useState } from 'react'; // Imports React and useState
import axios from 'axios'; // Imports axios

const Signup = () => { // Defines Signup component
  const [name, setName] = useState(''); // State for name
  const [email, setEmail] = useState(''); // State for email
  const [password, setPassword] = useState(''); // State for password
  const [error, setError] = useState(''); // State for error

  const handleSubmit = async (e) => { // Handles form submission
```

```
    e.preventDefault(); // Prevents default behavior
    try { // Starts try block
      const res = await axios.post('/api/users/register', { name,
email, password }); // Sends signup request
      localStorage.setItem('token', res.data.token); // Stores token
      window.location.href = '/profile'; // Redirects to profile
    } catch (err) { // Catches errors
      setError(err.response?.data?.message || 'Signup failed'); // Sets
error
    }
  };

  return ( // Returns JSX
    <div className="container">
      <h2>Signup</h2> {/* Page title */}
      {error && <p className="error">{error}</p>} {/* Displays error
*/}
      <form onSubmit={handleSubmit}> {/* Form with submit handler */}
        <div>
          <label>Name:</label> {/* Name label */}
          <input
            type="text"
            value={name}
            onChange={(e) => setName(e.target.value)} // Updates name
state
            required
          />
        </div>
        <div>
          <label>Email:</label> {/* Email label */}
          <input
            type="email"
            value={email}
            onChange={(e) => setEmail(e.target.value)} // Updates email
state
            required
          />
        </div>
        <div>
          <label>Password:</label> {/* Password label */}
          <input
            type="password"
            value={password}
            onChange={(e) => setPassword(e.target.value)} // Updates
password state
            required
          />
        </div>
        <button type="submit">Signup</button> {/* Submit button */}
      </form>
      <p>Already have an account? <a href="/login">Login</a></p> {/*
Login link */}
    </div>
  );
```

```
};

export default Signup; // Exports component
```

4. **Profile Page**:
   - In `views/Profile.jsx`:

```
import React, { useState, useEffect } from 'react'; // Imports React,
useState, useEffect
import axios from 'axios'; // Imports axios

const Profile = () => { // Defines Profile component
  const [user, setUser] = useState(null); // State for user data
  const [error, setError] = useState(''); // State for error

  useEffect(() => { // Runs on component mount
    const fetchUser = async () => { // Fetches user data
      try { // Starts try block
        const token = localStorage.getItem('token'); // Gets token
        if (!token) throw new Error('No token found'); // Checks for
token
        const res = await axios.get('/api/users/me', { // Sends request
          headers: { Authorization: `Bearer ${token}` }, // Includes
token
        });
        setUser(res.data); // Sets user data
      } catch (err) { // Catches errors
        setError(err.response?.data?.message || 'Failed to load
profile'); // Sets error
      }
    };
    fetchUser(); // Calls fetch function
  }, []); // Empty dependency array

  const handleLogout = () => { // Handles logout
    localStorage.removeItem('token'); // Removes token
    window.location.href = '/login'; // Redirects to login
  };

  if (error) return <div className="container"><p className="error">
{error}</p></div>; // Shows error
  if (!user) return <div className="container">Loading...</div>; //
Shows loading

  return ( // Returns JSX
    <div className="container">
      <h2>Profile</h2> {/* Page title */}
      <p>Name: {user.name}</p> {/* Displays name */}
      <p>Email: {user.email}</p> {/* Displays email */}
      <button onClick={handleLogout}>Logout</button> {/* Logout button
*/}
```

```
      </div>
    );
};

export default Profile; // Exports component
```

Step 10: Add Basic Styling

1. **Create CSS File**:
   - In `public/styles.css`:

```
body { /* Styles body */
  font-family: Arial, sans-serif; /* Sets font */
  margin: 0; /* Removes default margin */
  padding: 0; /* Removes default padding */
}
.container { /* Styles container */
  max-width: 600px; /* Sets max width */
  margin: 50px auto; /* Centers container */
  padding: 20px; /* Adds padding */
}
h1, h2 { /* Styles headings */
  text-align: center; /* Centers text */
}
form { /* Styles form */
  display: flex; /* Uses flexbox */
  flex-direction: column; /* Stacks elements */
  gap: 15px; /* Adds spacing */
}
label { /* Styles labels */
  font-weight: bold; /* Makes text bold */
}
input { /* Styles inputs */
  padding: 8px; /* Adds padding */
  font-size: 16px; /* Sets font size */
}
button { /* Styles buttons */
  padding: 10px; /* Adds padding */
  background-color: #007bff; /* Sets background */
  color: white; /* Sets text color */
  border: none; /* Removes border */
  cursor: pointer; /* Adds pointer cursor */
}
button:hover { /* Styles button hover */
  background-color: #0056b3; /* Darkens background */
}
.error { /* Styles error messages */
  color: red; /* Sets color */
  text-align: center; /* Centers text */
}
```

```
a { /* Styles links */
  color: #007bff; /* Sets color */
  text-decoration: none; /* Removes underline */
}
a:hover { /* Styles link hover */
  text-decoration: underline; /* Adds underline */
}
```

Step 11: Test the Application

1. **Start the Server**:
   - Run:

     ```
     npm run dev  # Starts server with nodemon
     ```

2. **Access Pages**:
   - Home: `http://localhost:5000/` # Displays welcome page
   - Login: `http://localhost:5000/login` # Displays login form
   - Signup: `http://localhost:5000/signup` # Displays signup form
   - Profile: `http://localhost:5000/profile` # Displays user profile (requires login)
3. **Test Functionality**:
   - Signup with a new user. # Creates user and redirects to profile
   - Login with the same user. # Authenticates and redirects to profile
   - View profile data. # Displays user info
   - Logout and verify redirection to login. # Clears token
   - Test error cases (e.g., invalid credentials). # Shows error messages
4. **Verify**:
   - Ensure MongoDB is running. # Confirms database connection
   - Check console for errors. # Validates server and client behavior

---

Best Practices Covered

- **Environment Variables**: Store secrets in `.env`. # Enhances security
- **Error Handling**: Use `express-async-handler`. # Simplifies async errors
- **Password Security**: Hash passwords with `bcrypt`. # Protects credentials
- **JWT Authentication**: Secure routes with JWT. # Ensures authorization
- **Code Organization**: Separate backend and frontend concerns. # Improves maintainability
- **Input Validation**: Basic validation in schema and controllers. # Prevents invalid data
- **React Integration**: Use `express-react-views` for server-side rendering. # Combines frontend and backend

---

Assignment

1. Add password confirmation in Signup. # Enhances security

2. Implement a forgot password feature. # Improves usability
3. Add client-side form validation. # Strengthens input checking
4. Allow profile updates (e.g., name, email). # Extends functionality

---

Additional Resources

- Express Documentation # Official guide
- Mongoose Documentation # Mongoose guide
- React Documentation # React guide
- JWT Documentation # JWT resource
- MongoDB Atlas # Cloud MongoDB