Backend (Express.js + MongoDB + JWT):-

1. Setup

npm init -y

npm install express mongoose bcryptjs jsonwebtoken cors

2. Create server.js

Server.js file code:-

```
const express = require('express');

const mongoose = require('mongoose');

const bcrypt = require('bcryptjs');

const jwt = require('jsonwebtoken');

const cors = require('cors');


const app = express();
```

```
app.use(express.json());

app.use(cors());


const JWT_SECRET = 'qP9s#V@lT8z!wK1xR3gDf2eL0uNmYcXa'; // Change this to a strong secret


// Connect to MongoDB

mongoose.connect('mongodb://localhost:27017/jwt-auth-example', {

  useNewUrlParser: true,

  useUnifiedTopology: true,

}).then(() => console.log('MongoDB connected'));


// User schema

const userSchema = new mongoose.Schema({

  username: { type: String, unique: true },

  password: String,

});


const User = mongoose.model('User', userSchema);


// Register

app.post('/api/register', async (req, res) => {

  const { username, password } = req.body;


  if (!username || !password)

    return res.status(400).json({ message: 'Username and password required' });
```

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

  const hashedPassword = await bcrypt.hash(password, 10);

  const user = new User({ username, password: hashedPassword });
  await user.save();

  res.json({ message: 'User registered successfully' });
});

// Login
app.post('/api/login', async (req, res) => {
  const { username, password } = req.body;

  const user = await User.findOne({ username });
  if (!user) return res.status(400).json({ message: 'Invalid credentials' });

  const isPasswordValid = await bcrypt.compare(password, user.password);
  if (!isPasswordValid)
    return res.status(400).json({ message: 'Invalid credentials' });

  const token = jwt.sign({ userId: user._id, username: user.username }, JWT_SECRET, { expiresIn: '1h' });
```

```
  res.json({ token });
});

// Middleware to verify token
const authMiddleware = (req, res, next) => {
  const token = req.headers['authorization']?.split(' ')[1]; // Bearer token

  if (!token) return res.status(401).json({ message: 'No token provided' });

  try {
    const decoded = jwt.verify(token, JWT_SECRET);
    req.user = decoded;
    next();
  } catch (err) {
    return res.status(401).json({ message: 'Invalid token' });
  }
};

// Protected route example
app.get('/api/protected', authMiddleware, (req, res) => {
  res.json({ message: `Welcome ${req.user.username}, this is protected data.` });
});

app.listen(4000, () => console.log('Server running on http://localhost:4000'))
```

Frontend (React.js):-

1. Setup

```
npx create-react-app jwt-auth-client

cd jwt-auth-client

npm install axios

npm start
```

2. Example React app (src/App.js):-

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

import axios from 'axios';


const API_URL = 'http://localhost:4000/api';


function App() {

  const [username, setUsername] = useState('');

  const [password, setPassword] = useState('');

  const [token, setToken] = useState(localStorage.getItem('token') || '');

  const [message, setMessage] = useState('');


  const register = async () => {

    try {

      const res = await axios.post(`${API_URL}/register`, { username, password });

      setMessage(res.data.message);

    } catch (err) {

      setMessage(err.response?.data?.message || 'Registration failed');
```

```
  }
};

const login = async () => {
  try {
    const res = await axios.post(`${API_URL}/login`, { username, password });
    setToken(res.data.token);
    localStorage.setItem('token', res.data.token);
    setMessage('Logged in successfully');
  } catch (err) {
    setMessage(err.response?.data?.message || 'Login failed');
  }
};

const logout = () => {
  setToken('');
  localStorage.removeItem('token');
  setMessage('Logged out');
};

const getProtectedData = async () => {
  if (!token) {
    setMessage('No token, please login');
    return;
  }
```

```
  try {
    const res = await axios.get(`${API_URL}/protected`, {
      headers: { Authorization: `Bearer ${token}` },
    });
    setMessage(res.data.message);
  } catch (err) {
    setMessage('Failed to fetch protected data');
  }
};

return (
  <div style={{ maxWidth: 400, margin: 'auto', padding: 20 }}>
    <h2>JWT Auth Example</h2>

    <input
      placeholder="Username"
      value={username}
      onChange={(e) => setUsername(e.target.value)}
      style={{ width: '100%', marginBottom: 10 }}
    />

    <input
      placeholder="Password"
      type="password"
```

```
        value={password}
        onChange={(e) => setPassword(e.target.value)}
        style={{ width: '100%', marginBottom: 10 }}
      />

      <button onClick={register} style={{ marginRight: 10 }}>
        Register
      </button>

      <button onClick={login} style={{ marginRight: 10 }}>
        Login
      </button>

      <button onClick={logout} style={{ marginRight: 10 }}>
        Logout
      </button>

      <button onClick={getProtectedData}>Get Protected Data</button>

      <p>{message}</p>

      {token && <p>Token: <code>{token}</code></p>}
    </div>
  );
}
```

```
export default App;
```

Here's your code organized into a **modular and scalable project structure**, which is best practice for maintainability and clean architecture in a Node.js + Express application.

□ Suggested Folder Structure:

```
jwt-auth-example/
├── config/
│   └── db.js
├── middleware/
│   └── auth.js
├── models/
│   └── User.js
├── routes/
│   └── auth.js
│   └── protected.js
├── .env
├── server.js
├── package.json
```

□ Step-by-Step Breakdown:

---

1. `config/db.js` – MongoDB connection

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

const connectDB = async () => {
  try {
    await mongoose.connect('mongodb://localhost:27017/jwt-auth-example', {
      useNewUrlParser: true,
      useUnifiedTopology: true,
    });
    console.log('MongoDB connected');
  } catch (err) {
    console.error('MongoDB connection error:', err.message);
    process.exit(1);
  }
};

module.exports = connectDB;
```

---

2. `models/User.js` – Mongoose schema

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

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

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

---

3. `middleware/auth.js` – JWT middleware

```
const jwt = require('jsonwebtoken');
const JWT_SECRET = process.env.JWT_SECRET;

const authMiddleware = (req, res, next) => {
  const token = req.headers['authorization']?.split(' ')[1]; // Bearer token

  if (!token) return res.status(401).json({ message: 'No token provided' });

  try {
    const decoded = jwt.verify(token, JWT_SECRET);
    req.user = decoded;
    next();
  } catch (err) {
    return res.status(401).json({ message: 'Invalid token' });
  }
};

module.exports = authMiddleware;
```

---

4. `routes/auth.js` – Auth routes

```
const express = require('express');
const bcrypt = require('bcryptjs');
const jwt = require('jsonwebtoken');
const User = require('../models/User');

const router = express.Router();
const JWT_SECRET = process.env.JWT_SECRET;

// Register
router.post('/register', async (req, res) => {
  const { username, password } = req.body;

  if (!username || !password)
    return res.status(400).json({ message: 'Username and password required'
});

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

  const hashedPassword = await bcrypt.hash(password, 10);
  const user = new User({ username, password: hashedPassword });
  await user.save();

  res.json({ message: 'User registered successfully' });
});

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

  const user = await User.findOne({ username });
  if (!user)
    return res.status(400).json({ message: 'Invalid credentials' });

  const isPasswordValid = await bcrypt.compare(password, user.password);
  if (!isPasswordValid)
    return res.status(400).json({ message: 'Invalid credentials' });

  const token = jwt.sign(
    { userId: user._id, username: user.username },
    JWT_SECRET,
    { expiresIn: '1h' }
  );

  res.json({ token });
});

module.exports = router;
```

---

5. `routes/protected.js` – Example protected route

```
const express = require('express');
const authMiddleware = require('../middleware/auth');

const router = express.Router();

router.get('/', authMiddleware, (req, res) => {
  res.json({ message: `Welcome ${req.user.username}, this is protected data.`
});
});

module.exports = router;
```

---

6. `.env` – Environment variables

```
JWT_SECRET=qP9s#V@lT8z!wK1xR3gDf2eL0uNmYcXa
PORT=4000
```

Don't forget to add `.env` to your `.gitignore` file to keep secrets private.

---

7. `server.js` – App entry point

```
const express = require('express');
const cors = require('cors');
const dotenv = require('dotenv');
const connectDB = require('./config/db');

dotenv.config();

const app = express();
app.use(express.json());
app.use(cors());

// DB connection
connectDB();

// Routes
app.use('/api', require('./routes/auth'));
app.use('/api/protected', require('./routes/protected'));

// Server start
const PORT = process.env.PORT || 4000;
app.listen(PORT, () => console.log(`Server running on
http://localhost:${PORT}`));
```

□ Benefits of This Structure:

- Separation of concerns (routes, models, middleware, config).
- Easier to maintain and scale.
- Easier testing and debugging.
- Secure secrets using `.env`.

To simplify token handling in your React app and avoid manually attaching the `Authorization` header every time, you can use an **Axios interceptor**.

□ What You'll Do:

1. Create a centralized Axios instance.
2. Add a request interceptor to attach the JWT token from `localStorage`.
3. Replace direct `axios` calls with the custom Axios instance.

□ Step-by-Step Implementation

---

1. Create `axiosInstance.js`

Create a file `axiosInstance.js` in your project :

```
// axiosInstance.js
import axios from 'axios';

const API_URL = 'http://localhost:4000/api';

const axiosInstance = axios.create({
  baseURL: API_URL,
});

// Request interceptor to add token
```

```
axiosInstance.interceptors.request.use(
  (config) => {
    const token = localStorage.getItem('token');
    if (token) {
      config.headers['Authorization'] = `Bearer ${token}`;
    }
    return config;
  },
  (error) => Promise.reject(error)
);

export default axiosInstance;
```

---

2. Update Your `App.js` to Use `axiosInstance`

Replace all `axios` imports and usage with `axiosInstance`.

```
import React, { useState } from 'react';
import axiosInstance from './axiosInstance'; // Adjust path as needed

function App() {
  const [username, setUsername] = useState('');
  const [password, setPassword] = useState('');
  const [token, setToken] = useState(localStorage.getItem('token') || '');
  const [message, setMessage] = useState('');

  const register = async () => {
    try {
      const res = await axiosInstance.post('/register', { username, password
});
      setMessage(res.data.message);
    } catch (err) {
      setMessage(err.response?.data?.message || 'Registration failed');
    }
  };

  const login = async () => {
    try {
      const res = await axiosInstance.post('/login', { username, password });
      setToken(res.data.token);
      localStorage.setItem('token', res.data.token);
      setMessage('Logged in successfully');
    } catch (err) {
      setMessage(err.response?.data?.message || 'Login failed');
    }
  };

  const logout = () => {
    setToken('');
    localStorage.removeItem('token');
    setMessage('Logged out');
  };

  const getProtectedData = async () => {
```

```
    if (!token) {
      setMessage('No token, please login');
      return;
    }

    try {
      const res = await axiosInstance.get('/protected');
      setMessage(res.data.message);
    } catch (err) {
      setMessage('Failed to fetch protected data');
    }
  };

  return (
    <div style={{ maxWidth: 400, margin: 'auto', padding: 20 }}>
      <h2>JWT Auth Example</h2>

      <input
        placeholder="Username"
        value={username}
        onChange={(e) => setUsername(e.target.value)}
        style={{ width: '100%', marginBottom: 10 }}
      />

      <input
        placeholder="Password"
        type="password"
        value={password}
        onChange={(e) => setPassword(e.target.value)}
        style={{ width: '100%', marginBottom: 10 }}
      />

      <button onClick={register} style={{ marginRight: 10 }}>
        Register
      </button>

      <button onClick={login} style={{ marginRight: 10 }}>
        Login
      </button>

      <button onClick={logout} style={{ marginRight: 10 }}>
        Logout
      </button>

      <button onClick={getProtectedData}>Get Protected Data</button>

      <p>{message}</p>

      {token && <p>Token: <code>{token}</code></p>}
    </div>
  );
}

export default App;
```

□ Benefits of Using Axios Interceptors:

- Centralized token management.
- Cleaner and DRY code.
- Easy to add **global error handling** (e.g., for 401 responses or token expiration).