

# JWT-based register, login, and logout system using:-

- **Express.js** for backend (with MongoDB for user storage)
  - **React.js** for frontend
- 

## Backend (Express.js + MongoDB + JWT)

### 1. Setup

```
npm init -y
npm install express mongoose bcryptjs jsonwebtoken cors
```

### 2. Create `server.js`

```
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 = 'your_jwt_secret_here'; // 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'));
```

---

## Run your backend:-

```
D:\jwtlogin>node server.js
(node:6044) [MONGODB DRIVER] Warning: useNewUrlParser
(Use `node --trace-warnings ...` to show where the wa
(node:6044) [MONGODB DRIVER] Warning: useUnifiedTopo
Server running on http://localhost:4000
MongoDB connected
```

## Frontend (React.js)

### 1. Setup

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

```
C:\Users\Big Data>d:
D:\>npx create-react-app jwt-auth-client
```

```
cd jwt-auth-client
```

```
D:\>cd jwt-auth-client
D:\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 }}
    />
  </div>
);

```

```

    <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;

```

---

## How this works

- **Register:** Save a new user with hashed password.
- **Login:** Validate credentials, return JWT.
- **Logout:** Clear the token from client.
- **Protected route:** Backend verifies token before providing data.

Run your front end :-

Npm start

```

> react-scripts start
D:\jwt-auth-client>npm start

```



### JWT Auth Example





Logged out

---

If you want, I can help you with:

- Running MongoDB locally or using MongoDB Atlas
- Improving token storage/security
- Refresh tokens
- Full folder structure

Would you like to dive deeper?

**You said:**

give me example of `const JWT_SECRET = 'your_jwt_secret_here';`

**ChatGPT said:**

Sure! The `JWT_SECRET` is a string key used to **sign and verify** JWT tokens. It should be:

- **Long enough** (at least 32 characters)
- **Random and complex**
- **Kept secret** (never expose it publicly, especially in production)

Here's an example of a strong JWT secret string you can use during development:

```
const JWT_SECRET = 'S3cure!JWT_T0ken_Secr3t_#2025_ComplexKey';
```

Or, a more random-looking one:

```
const JWT_SECRET = 'm4X9z!b7R@yVqLp2#dF%hG$wZtCs8Jv0E';
```

---

## How to generate a strong secret?

You can generate one in Node.js console like this:

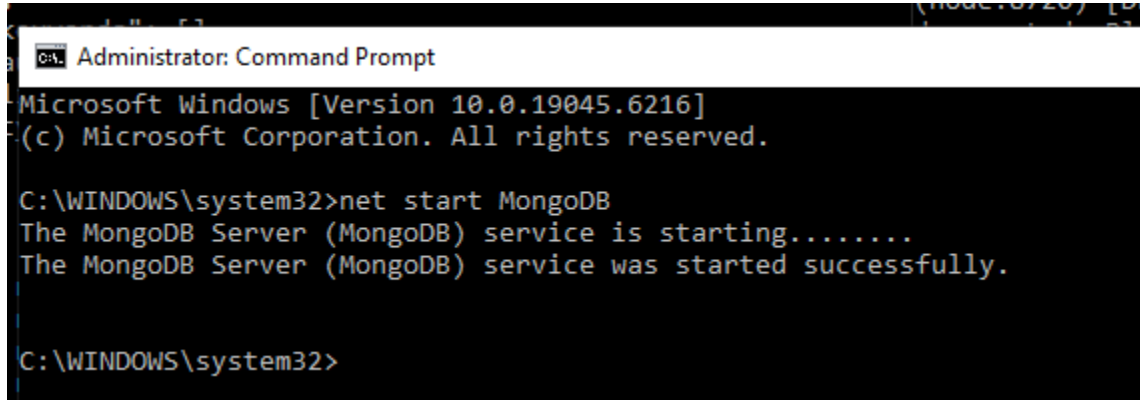
```
require('crypto').randomBytes(64).toString('hex');
```

Example output:

```
'9f2c8e3b4d6a1f9c8b7e0d12345a6789abcdef0123456789abcdef0123456789abcdef012345  
6789abcdef0123456789ab'
```

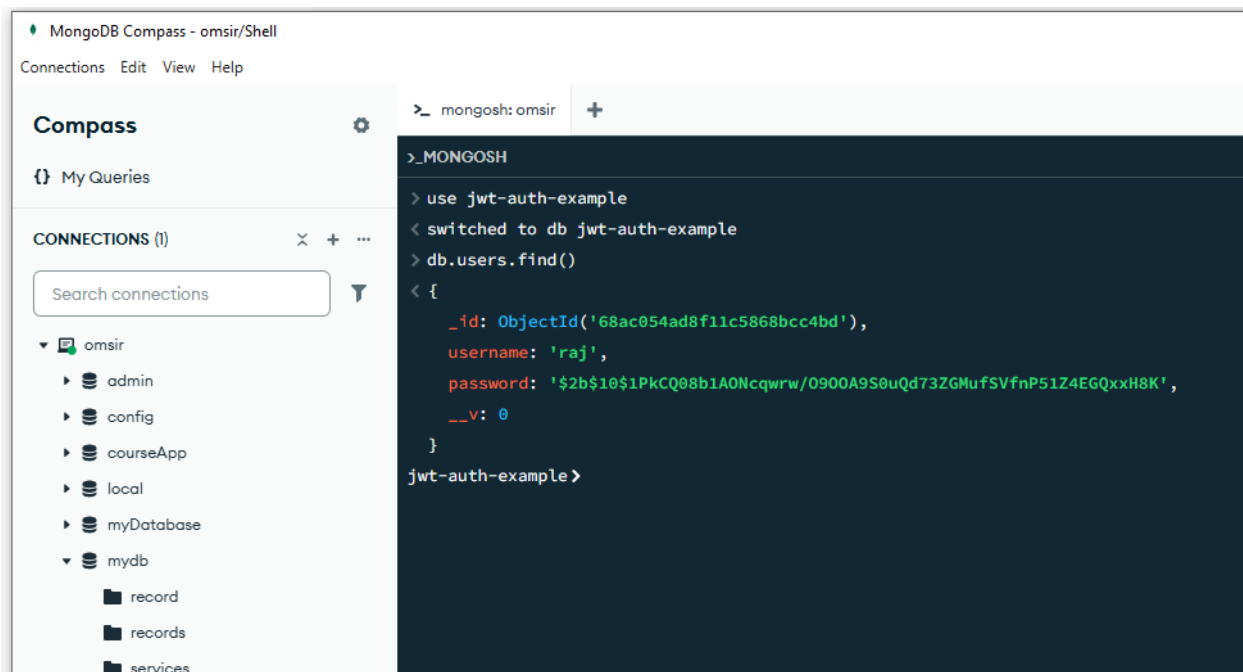
Note:-

First open your command prompt:-



```
C:\WINDOWS\system32>net start MongoDB
The MongoDB Server (MongoDB) service is starting.....
The MongoDB Server (MongoDB) service was started successfully.

C:\WINDOWS\system32>
```



```
MongoDB Compass - omsir/Shell
Connections Edit View Help

Compass
My Queries

CONNECTIONS (1)
Search connections
omsir
  admin
  config
  courseApp
  local
  myDatabase
  mydb
    record
    records
    services

>_ mongosh: omsir +
>_MONGOSH
> use jwt-auth-example
< switched to db jwt-auth-example
> db.users.find()
< {
  _id: ObjectId('68ac054ad8f11c5868bcc4bd'),
  username: 'raj',
  password: '$2b$10$1PkCQ08b1A0Ncqwrw/0900A9S0uQd73ZGMufSVfnP51Z4EGQxxH8K',
  __v: 0
}
jwt-auth-example >
```