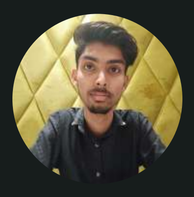
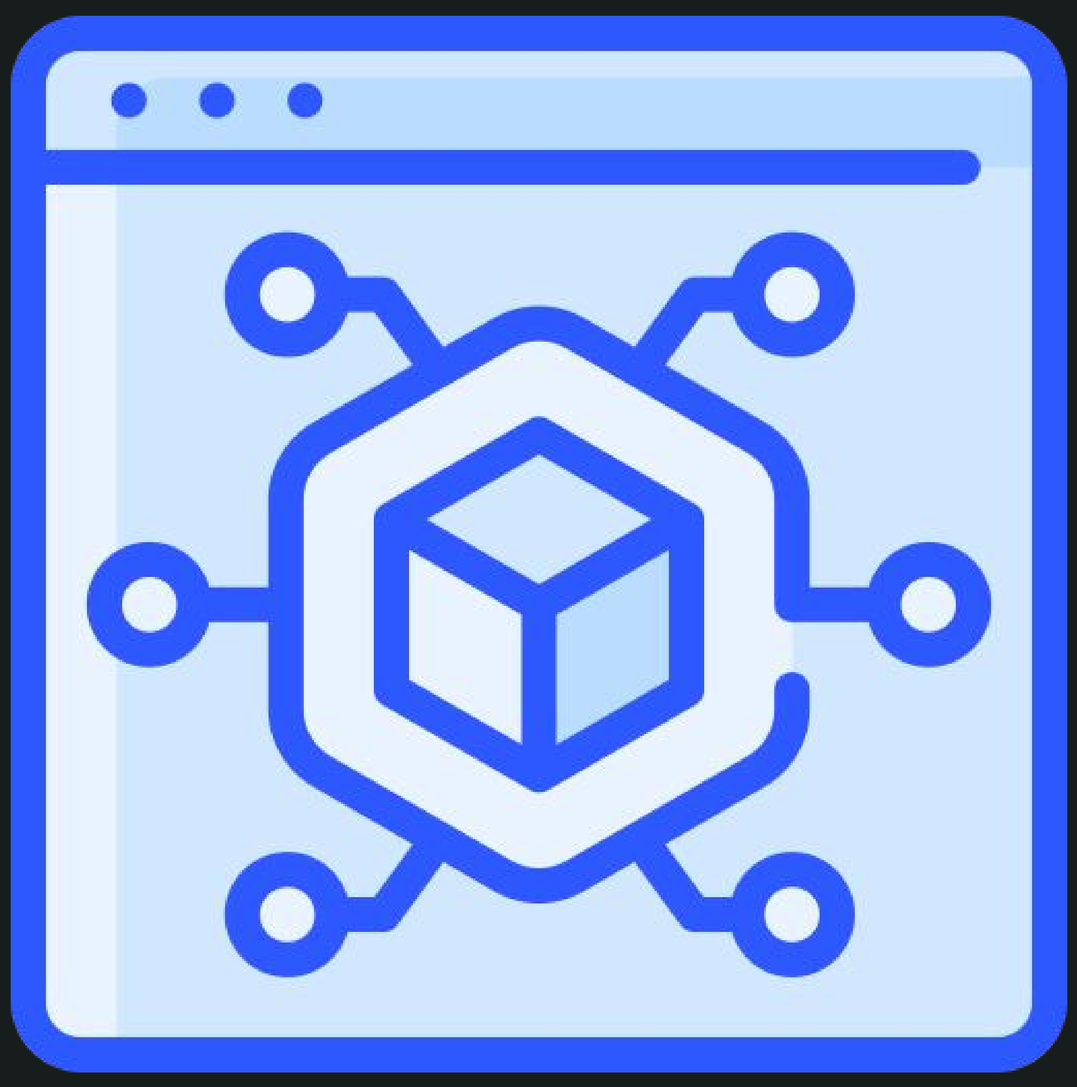


Gagan Saini
@gagan-saini-gs

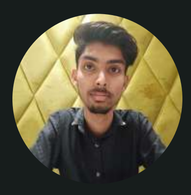
Github: Gagan-Saini-GS



Become Authorize Using

JSON WEB TOKEN

Created By
Gagan Saini



Gagan Saini
@gagan-saini-gs

Github: Gagan-Saini-GS

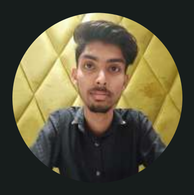
JWT

→ JWT is a compact, URL-safe way to represent claims between two parties JWT is used for securely transmitting information for authentication purposes in web apps.

A token can look like

jwt

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJfaWQiOiI2Njhk
MTU1YTRiZjMyZGE3Y2VjMDZlZTUiLCJuYW1lIjoiR3Vlc3QgVXNlc
iIsImVtYWlsIjoiZ3Vlc3R1c2VyQGNsb3NldGZhc2hpb24uY29tIi
wiaWF0IjoxNzI0NzY3MzYyfQ.-nZTrRvsIdXR9StnV0edN8Xv79xK
lKs7lb2fz6riRx9
```



Gagan Saini
@gagan-saini-gs

Github: Gagan-Saini-GS

Create Token

First install jsonwebtoken library to create jwt.

jwt

```
npm install jsonwebtoken
```

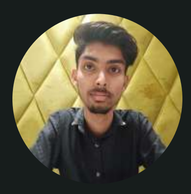
jwt

```
// Payload can be anything valid.
const payload = {
  userId: user._id,
  email: user.email,
  role: user.role
};

// Create token using sign method
const token = jwt.sign(
  payload,
  JWT_SECRET,
  { expiresIn: '1h' }
);
```

You can use any data in payload to create your json web tokens.

The signing algorithm (like HMAC SHA-256) uses the **JWT_SECRET** to generate a hash, which becomes the signature part of the token. And this **same JWT_SECRET** is used to **verify** tokens also.


Gagan Saini
@gagan-saini-gs

Github: Gagan-Saini-GS

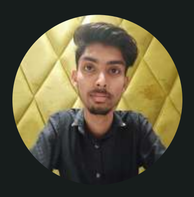
How to use JWT?

Client Side → The JWT is stored on the client in local storage, session storage, or cookies.

Request with JWT → For every subsequent request to a protected route, the client includes the JWT in the authorization header.

```jwt
Authorization: Bearer <your_jwt_token>
```

Server Side → The server validates the token's signature and checks expiration. If valid, the user is granted access.


Gagan Saini
@gagan-saini-gs

Github: Gagan-Saini-GS

How to verify JWT?

When a client sends the JWT back to the server, the server verifies it by using the same JWT_SECRET that was used to sign it.

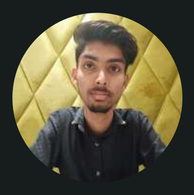
This ensures that the token is valid and hasn’t been modified.

jwt

```
const decoded = jwt.verify(
    token,
    process.env.JWT_SECRET
);
```

If the secret used during verification matches the one used to sign the token, the server knows the token is legitimate.

And allows user to continue using services.


Gagan Saini
@gagan-saini-gs

Github: Gagan-Saini-GS

Store JWT

Cookies is a safe choice for storage because cookies can be made HTTPOnly to prevent access via JavaScript.

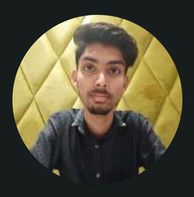
You can store JWT in local storage also it is easier to implement but vulnerable to XSS (Cross-Site Scripting) attacks.

Pros

Stateless → No need to store sessions on the server.

Scalable → Easily scalable across distributed systems.

Portable → Works across different platforms and services.


Gagan Saini
@gagan-saini-gs

Github: Gagan-Saini-GS

Cons ❌

Security Risks → JWTs are vulnerable to XSS if stored in local Storage or session storage.

Token Size → Larger payloads increase the size of the token.

No Revocation → Once a JWT is issued, it can’t be revoked unless handled via expiration or blacklist mechanisms.

Alternatives

OAuth 2.0
↳ Widely used for authorization, allowing third-party services to access resources without exposing credentials.

Session-based Authentication
↳ Traditional method where user session info is stored on the server, but it’s harder to scale.

Did you find it Useful?

Leave a comment!

Alamin CodePapa

@CodePapa360

FOLLOW FOR MORE

Like Comment Repost