

teach me jwt and the flow of working in it using diagram
 teach me the theory of it
 tell me how it works ?
 give examples as well

JWT (JSON Web Token) Overview

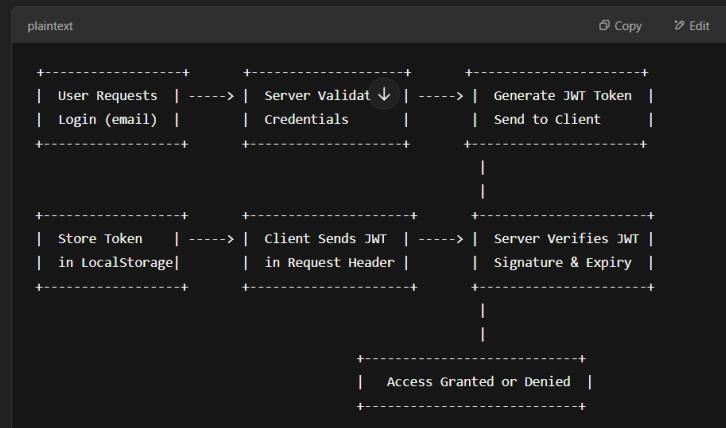
JWT (JSON Web Token) is a compact, URL-safe token format used for securely transmitting information between parties as a JSON object. It is commonly used for authentication and authorization in web applications.

JWT Flow (Working) with Diagram

Steps of JWT Authentication Flow

1. User Login:
 - The user provides their credentials (username & password).
2. Server Validates Credentials:
 - The backend verifies the credentials using a database.
3. JWT Generation:
 - If authentication is successful, the server generates a JWT containing user data and signs it using a secret key or private key.
4. Token Sent to Client:
 - The JWT is sent to the client (usually in the response body or as an HTTP-only cookie).
5. Client Stores the Token:
 - The client (frontend) stores the JWT in localStorage, sessionStorage, or cookies.
6. Client Requests with Token:
 - For future requests, the client sends the JWT in the `Authorization` header as a Bearer token.
7. Server Verifies Token:
 - The server extracts the token, verifies its signature, and checks if it's valid.
8. Access Granted:
 - If the token is valid, the user is granted access to protected resources. Otherwise, an error (e.g., 401 Unauthorized) is returned.

JWT Authentication Flow Diagram



How JWT Works?

A JWT consists of three parts:

1. **Header:** Contains metadata about the token (e.g., type & algorithm).
2. **Payload:** Contains claims (data about the user or permissions).
3. **Signature:** Used to verify the token's authenticity and integrity.

The JWT structure is:

```
CSS  
Header.Payload.Signature
```

Example JWT (Encoded):

```
Copy Edit  
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VycWlzc3VtZWlkIjoxNjkwMjk1NTMwfQ.JwbYo_MZT3kcv.
```

Breaking Down JWT Example

1. Header (Base64 Encoded)

```
json  
{  
  "alg": "HS256",  
  "typ": "JWT"  
}
```

2. Payload (Base64 Encoded)

```
json  
{  
  "userId": 123,  
  "exp": 1690295530  
}
```

3. Signature (Generated using a secret key)

```
SCSS  
HMACSHA256(base64UrlEncode(header) + "." + base64UrlEncode(payload), secret)
```

Example Code (JWT Implementation in Node.js + Express)

1. Install Dependencies

```
sh  
npm install express jsonwebtoken dotenv
```

2. Generate JWT Token (Login Route)

```
javascript  
const express = require("express");  
const jwt = require("jsonwebtoken");  
const dotenv = require("dotenv");  
  
dotenv.config();  
const app = express();  
app.use(express.json());  
  
const SECRET_KEY = process.env.JWT_SECRET || "your_secret_key";  
  
// Login Route  
app.post("/login", (req, res) => {  
  const { username, password } = req.body;  
  // Dummy user authentication  
  if (username === "admin" && password === "password") {  
    const token = jwt.sign({ username }, SECRET_KEY, { expiresIn: "1h" });  
    res.json({ token });  
  } else {  
    res.status(401).json({ message: "Invalid credentials" });  
  }  
});
```

3. Middleware to Verify JWT

```
javascript
const verifyToken = (req, res, next) => {
  const token = req.headers["authorization"];
  if (!token) return res.status(403).json({ message: "Token required" });

  try {
    const decoded = jwt.verify(token.split(" ")[1], SECRET_KEY);
    req.user = decoded;
    next();
  } catch (err) {
    res.status(401).json({ message: "Invalid token" });
  }
};
```

4. Protected Route

```
javascript
app.get("/protected", verifyToken, (req, res) => {
  res.json({ message: "Protected Data Accessed", user: req.user });
});

app.listen(5000, () => console.log("Server running on port 5000"));
```

Conclusion

- JWT is used for stateless authentication.
- It includes three parts: Header, Payload, and Signature.
- Used for securely transferring user data between client and server.
- Tokens should be stored securely (prefer HTTP-only cookies over localStorage).

Would you like a frontend example with React as well? ↗

question 1.
what is the most common way to send the jwt token after validating a user while logging and show me the code

question 2.
once the user is ↓ dated token is being sent to client and stored now how the token is coming while the user is again requesting ... what is the most common way ?

Answer 1: How to Send JWT Token After Validating a User?

The most common way to send a JWT token after validating a user is through the **response body** or **HTTP-only cookies** (more secure).

Method 1: Sending Token in Response Body (Common but Less Secure)

```
javascript
app.post("/login", (req, res) => {
  const { username, password } = req.body;
  if (username === "admin" && password === "password") {
    const token = jwt.sign({ username }, "your_secret_key", { expiresIn: "1h" });
    res.json({ token });
  } else {
    res.status(401).json({ message: "Invalid credentials" });
  }
});
```

Frontend stores this token in `localStorage` or `sessionStorage` and sends it with every request.

Method 2: Sending Token in HTTP-Only Cookie (More Secure)

```
javascript
const cookie = require("cookie-parser");
app.use(cookie());

app.post("/login", (req, res) => {
```

```

const { username, password } = req.body;
if (username === "admin" && password === "password") {
  const token = jwt.sign({ username }, "your_secret_key", { expiresIn: "1h" });
  res.cookie("token", token, { httpOnly: true, secure: true, sameSite: "Strict" });
  res.json({ message: "Logged in successfully" });
} else {
  res.status(401).json({ message: "Invalid credentials" });
}
);

```

Frontend doesn't need to store the token manually. The browser handles cookies automatically.

Answer 2: How the Token is Sent When the User Makes a Request Again?

The most common way is to send the token in the `Authorization` header as a Bearer Token in every request.

Client Sends Token in Header (Using Bearer Token)

```

javascript
fetch("/protected", {
  method: "GET",
  headers: {
    "Authorization": `Bearer ${localStorage.getItem("token")}`
  }
})
.then(response => response.json())
.then(data => console.log(data));

```

Server Verifies Token in Middleware

```

javascript
const verifyToken = (req, res, next) => {
  const token = req.headers["authorization"];
  if (!token) return res.status(403).json({ message: "Token required" });

  try {
    const decoded = jwt.verify(token.split(" ")[1], "your_secret_key");
    req.user = decoded;
    next();
  } catch (err) {
    res.status(401).json({ message: "Invalid token" });
  }
};

```

Alternative: Sending Token via HTTP-Only Cookies (More Secure)

- Client-side: No need to manually send the token. The browser automatically includes the cookie in each request.
- Server-side: Extract token from `req.cookies` instead of `Authorization` header.

```

javascript
const verifyToken = (req, res, next) => {
  const token = req.cookies.token;
  if (!token) return res.status(403).json({ message: "Token required" });

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

```

Most Common Approach?

- Bearer token in `Authorization` header (Widely used in REST APIs).
- HTTP-only cookies (More secure, prevents XSS attacks).

```
const cookie = require("cookie-parser");
app.use(cookie());
```

why this ?

Why `cookie-parser` and `app.use(cookie())`?

`cookie-parser` is a middleware in Express that parses cookies attached to incoming HTTP requests. It allows the server to read and manage cookies sent by the client.

Use Case in JWT Authentication

When using HTTP-only cookies to store JWT tokens, the token is automatically sent with each request. The server needs a way to extract this token from cookies, which is why we use `cookie-parser`.

How It Works?

1. Install `cookie-parser`

```
sh
npm install cookie-parser
```

2. Import and Use `cookie-parser`

```
javascript
const express = require("express");
const cookieParser = require("cookie-parser");

const app = express();
app.use(cookieParser()); // Middleware to parse cookies
```

3. Storing JWT in HTTP-Only Cookie During Login

```
javascript
app.post("/login", (req, res) => {
  const { username, password } = req.body;
  if (username === "admin" && password === "password") {
    const token = jwt.sign({ username }, "your_secret_key", { expiresIn: "1h" });

    // Store JWT in HTTP-Only cookie
    res.cookie("token", token, { httpOnly: true, secure: true, sameSite: "Strict" });
    res.json({ message: "Logged in successfully" });
  } else {
    res.status(401).json({ message: "Invalid credentials" });
  }
});
```

Why `httpOnly: true`? It prevents JavaScript access, reducing XSS attacks.

4. Extracting JWT Token from Cookies in Protected Routes

```
javascript
const verifyToken = (req, res, next) => {
  const token = req.cookies.token; // Extract token from cookies
  if (!token) return res.status(403).json({ message: "Token required" });

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

app.get("/protected", verifyToken, (req, res) => {
  res.json({ message: "Protected data accessed", user: req.user });
});
```

Benefit: The token is sent automatically with requests, no need to store it in `localStorage`.

Conclusion

- `cookie-parser` extracts cookies from incoming requests.

- Used to store JWT tokens securely in HTTP-only cookies.
 - Helps in automatic authentication without manually sending tokens in headers.

Would you like a frontend example as well?

