JSON Web Token (JWT) in Web Applications

Authentication

Authentication means **verifying that the user is really the account owner**.
The simplest form is **username + password**. If both are correct, the system assumes the person is the account owner.

But accounts can be hacked, so developers often add extra layers like:

- One-Time Password (OTP) via email or SMS
- Fingerprint or Face recognition
- Multi-Factor Authentication (MFA)

All of these ensure that the person logging in is the **actual account owner**.

Authorization

Authorization is about **defining user permissions**.
For example, if the user is a *student*, they cannot assign grades. But if the user is a *teacher*, they have permission to do so.

So, **authentication = who you are**, while **authorization = what you're allowed to do**.

Separate Frontend and Backend

Modern web apps usually have a **separate frontend and backend**, often deployed on different servers around the world.

- This raises a question:
  How can the backend ensure that the person sending a request from the frontend is really the account owner, not someone else?

- One of the most popular solutions to this problem is **JWT**

What is JWT?

JWT (JSON Web Token) is a method for **securely transmitting information** between the frontend and backend. It helps with both **authentication** and **authorization**.

A JWT consists of **three parts**:

1. **Header** → contains the algorithm used (e.g., HMAC SHA256).
2. **Payload** → contains the data (JSON).
   - It is **Base64 encoded** (not encrypted).
   - Anyone can decode it, so sensitive data should never be stored here.
   - Common claims include user ID, roles, or even an IP address for validation.
3. **Signature** → generated using the header, payload, secret key, and algorithm.
   - It ensures that the token hasn't been tampered with.

- 401 ➔ unauthorized
- Refresh Token

Registration (Sign Up)

Purpose: Create a new account in the system.

Flow:

1. The user submits their details (e.g., username, email, password).
2. The backend validates the input (unique email, strong password, etc.).
3. The password is hashed & stored in the database (never store raw passwords).
4. The backend usually returns a success response (e.g., “Account created”).
5. At this point, the user can either:
    - Be asked to log in with their new credentials (common approach), or
    - Automatically receive a JWT upon successful registration (so they’re signed in immediately).

Login (Sign In)

Purpose: Authenticate an existing account.

Flow (what you already wrote):

1. User sends username/email + password.
2. Backend checks the credentials against the database.
3. If valid → generate JWT and return it to the frontend.
4. From then on, every request must include the token in the Authorization header.

How JWT Works in a Website

1. When a user logs in successfully, the backend **creates a JWT** and returns it to the frontend.
2. The frontend stores the token (usually in local Storage or cookies).
3. For every request, the frontend sends the token in the **Authorization header**:

   Authorization: Bearer <JWT_TOKEN>

4. The backend verifies the token using its **secret key** (without going back to the database).
5. If the token is valid → the request is accepted.
6. If the token is invalid → return **401 Unauthorized**.

401 Unauthorized

This is the HTTP response code sent when:

- The user **didn't provide a token**.
- The token is **invalid** or **expired**.
- The token signature doesn't match (possibly tampered).

It tells the client: *"You are not authorized. Please log in again."*

Refresh Token

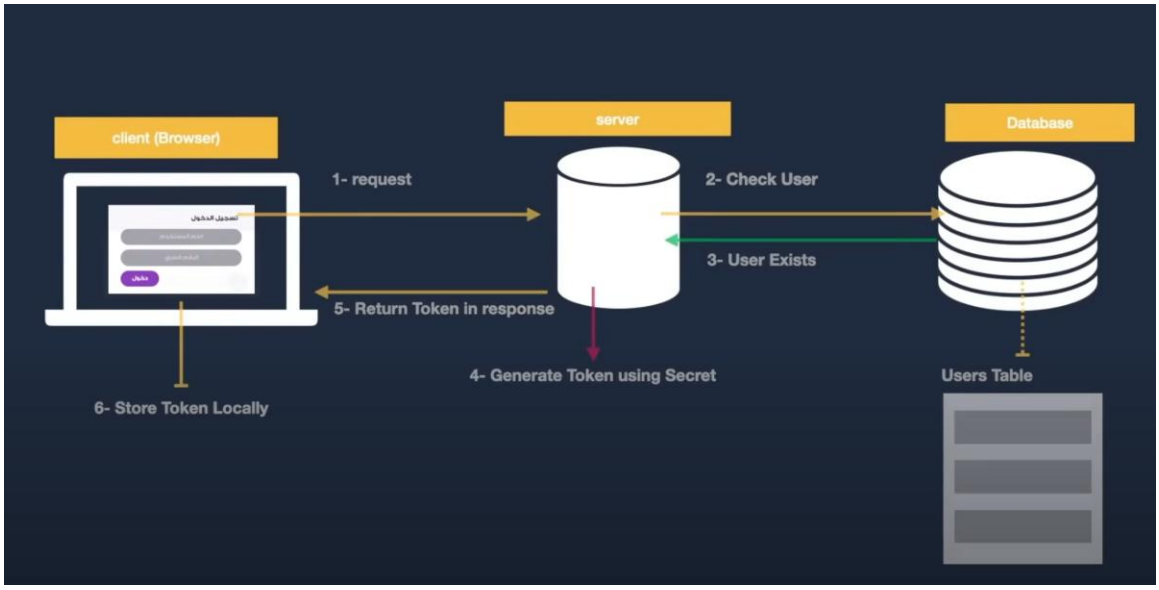
Since JWTs have an expiration time (e.g., 15 minutes or 1 hour), users would otherwise be logged out frequently.

To solve this, we use a **refresh token**:

- A long-lived token stored securely on the client.

- When the access JWT expires, the client uses the refresh token to request a **new access token** without asking the user to log in again.
- If the refresh token itself expires or is invalid, the user must log in again.

This system keeps the app **secure and user-friendly**.



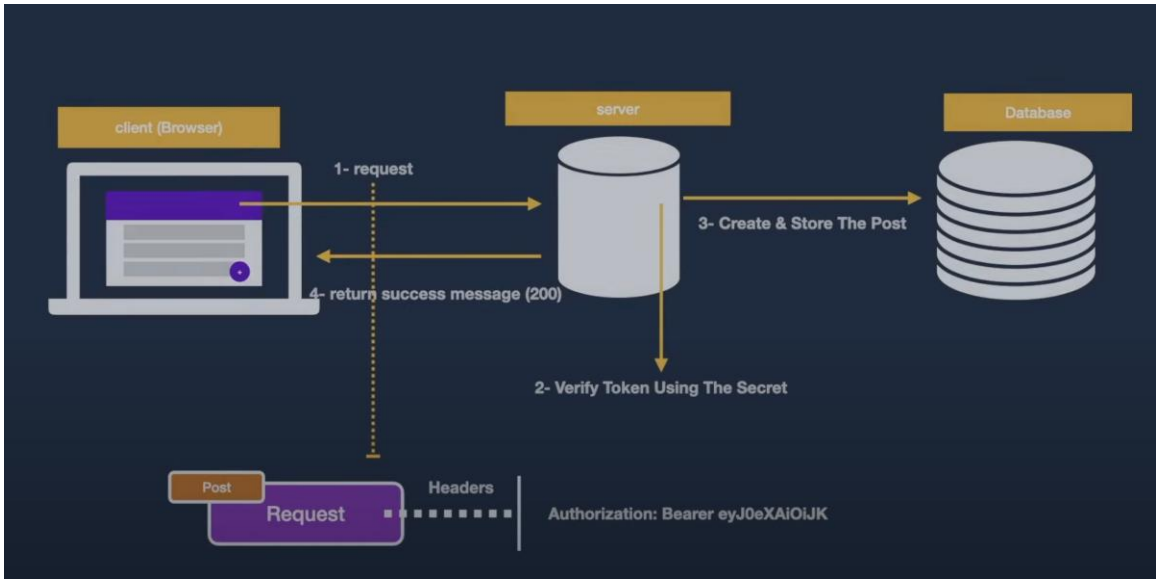
This diagram explains the **JWT login process**.
Here's the flow in **points**

JWT Login Flow (Step by Step)

1. **Request from Client (Browser)**
    - The user enters their credentials (username & password) in the login form.
    - The browser sends a **login request** to the server.
2. **Server Checks User**
    - The server receives the request and checks the credentials against the database.

3. **User Exists?**
    - If the user is found and the credentials are correct → continue.
    - If not → return an error (401 Unauthorized).
4. **Generate JWT**
    - The server generates a **JWT token** using its **secret key**.
    - The token includes encoded header, payload (user info, expiration, roles, etc.), and signature.
5. **Return Token in Response**
    - The server sends the generated token back to the client in the response body.
6. **Store Token Locally**
    - The client stores the token (commonly in **localStorage** or a **cookie**).
    - From now on, the client will attach this token in the **Authorization header** with every request:

Authorization: Bearer <JWT_TOKEN>



This diagram illustrates how **JWT is used when making a request to a protected API endpoint** (e.g., creating a new post).

Here's the process explained in points

JWT Authorization Flow (Creating a Post)

1. **Request from Client (Browser)**
    - The user tries to create a new post from the browser.
    - The client sends a **request** (POST request) to the server.
    - The request includes the **JWT token** in the Authorization header:
    - Authorization: Bearer <JWT_TOKEN>
2. **Verify Token using Secret**
    - The server receives the request.
    - It verifies the JWT using its **secret key** (or public key if using RS256).

- If the token is invalid or expired → the server rejects the request with **401 Unauthorized**.

3. **Create & Store the Post**
    - If the token is valid, the server proceeds with the request.
    - It creates the new post and stores it in the database.
4. **Return Success Message (200 OK)**
    - The server sends a success response (HTTP status **200**) back to the client.
    - The client now sees that the post was created successfully.

References:

Trmeez Academy [**What is JWT & How it works**]. YouTube.https://www.youtube.com/watch?v=1O3L1hzfRQc

Name: Hassan Yousef Al-Hussaini

Date: 26/08/2025