

Json Web Token (JWT) Made Easy

Additional Sources:

1. <https://medium.com/@shubhadeepchat/explaining-json-web-tokens-jwt-with-real-life-examples-62a183889022>
2. <https://jwt.io/introduction>
3. https://openid.net/specs/openid-connect-core-1_0.html#TokenEndpoint
4. https://developer.auth0.com/?_gl=1*kd27d8*_gcl_au*NjY0OTE5MzkxLjE3MzAwNDk0NDY.*_ga*MjlwNDU5MjEzLjE3MzAwNDk0NDY.*_ga_QKM5ODY5OC4yLjEuMTczMDEwMDY3Ny42MC4wLjA

What is JWT?

- JWT is a **compact, self-contained token** used to securely transmit information between two parties.

Components of JWT (Mind Map)

- Three main parts:
 1. **Header:**
 - **What it is:** Think of it as the **label** on your wristband that says how it's made.
 - **Contains:**
 - **Algorithm:** E.g., "HS256" (this tells how the token is signed).
 - **Type:** Always says "JWT."
 - **Why it matters:** This part defines the "rules" for how the token is created.
 2. **Payload:**
 - **What it is:** The **identity card** part of your wristband. It carries all the information the server needs to know about you.
 - **Contains:**
 - **Claims:** These are statements about the user, like:
 - **Standard Claims:** e.g., `sub` (subject, the user ID), `iat` (issued at time).
 - **Custom Claims:** Anything else the server wants to store (e.g., roles like admin or user).
 - **Analogy:** If you are in the VIP section of the theme park, your wristband shows that special permission in its payload.
 3. **Signature:**
 - **What it is:** The **security seal** on your wristband.
 - **Contains:**
 - It's made by combining the **Header** and **Payload** and then encoding it with a **secret key** known only to the server.
 - **Why it matters:** This ensures that if someone tries to modify the token, the signature will break, and the system will reject it.

Flow of JWT Authentication:

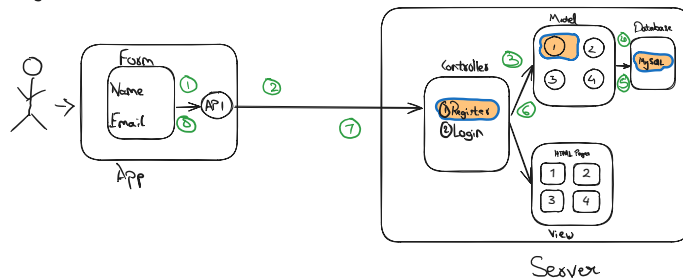
- **Step 1:** A user tries to log in to a website or app.
 - The system asks for credentials (username & password).
 - **Behind the scenes:** The server checks if the credentials are correct.
- **Step 2:** If correct, the server generates a **JWT**.
 - **What happens:** The server packages the user's data (ID, roles) into the **payload**, creates the **header**, and then creates a **signature** using its secret key.
 - **Comparison:** It's like issuing a special wristband with encrypted information.
- **Step 3:** The server sends the JWT to the user.
 - This token is now stored in the user's **local storage** or **browser cookies**.
- **Step 4:** Every time the user makes a request (e.g., accessing a secure page), the JWT is sent with the request.
 - **How it works:** The server doesn't need to ask for the user's password again—it just checks the token.
- **Step 5:** The server **verifies** the JWT's **signature**.
 - **If valid:** The user gets access.
 - **If invalid:** Access is denied

Important JWT Concepts (3 mins)

- **JWT is stateless:**

- Once the server gives you a token, it doesn't need to keep track of your session. The token itself holds all the information. This is useful for **scaling** applications.
- **JWT can expire:**
 - The server can set an **expiration time** on the token (e.g., 1 hour). After that, the token becomes invalid, and the user must log in again to get a new one.
- **JWT vs Session Cookies:**
 - **JWT:** All info is stored in the token itself (stateless).
 - **Session Cookies:** The server keeps track of your session in memory (stateful).

Registration:



Login:

