**Ureesh Surajballi**

**ST10075191**

**INSY7314**

**ICE 2**

JWT, or JSON Web Token, is a compact and self-contained way to securely transmit information between parties as a JSON object. It’s widely used in modern web applications for authentication and authorization. A JWT consists of three parts: the header, which specifies the signing algorithm; the payload, which contains claims about the user or system; and the signature, which ensures the token hasn’t been tampered with. Because it’s digitally signed, a JWT can be trusted by the server without needing to store session data, making it ideal for stateless applications.

JWTs are essential for secure web applications because they enable scalable, decentralized authentication. Instead of relying on server-side sessions, which can be cumbersome and difficult to manage across distributed systems, JWTs allow each request to carry its own proof of identity. This is particularly useful in microservices architectures and mobile-first environments, where performance and security are critical. JWTs also support granular access control by embedding user roles and permissions directly into the token, and they can be set to expire, reducing the risk of long-lived credentials.

In practice, JWTs are transmitted via HTTP headers, typically using the Authorization header with the Bearer schema. For example, a client might send a request like Authorization: Bearer <token>. The server then extracts the token, verifies its signature and claims, and grants or denies access accordingly. This approach is clean, efficient, and compatible with RESTful APIs, making it a go-to solution for secure communication between clients and servers.

However, JWTs must be implemented carefully. A real-world breach occurred when developers misconfigured their JWT validation logic to accept tokens with the algorithm set to "none". This effectively disabled signature verification, allowing attackers to forge tokens and gain unauthorized access. One such vulnerability was discovered in Auth0’s implementation, where researchers demonstrated how they could escalate privileges by crafting unsigned tokens. The lesson here is clear: always enforce strong signing algorithms like HS256 or RS256 and never allow "none" as a valid option.

If you're working on an Android prototype with SSO and REST API integration, securing your token flow is critical. I can help you implement best practices for token storage, expiry, and refresh strategies to ensure your civic tech app remains trustworthy and resilient. Let me know if you'd like to explore that next.