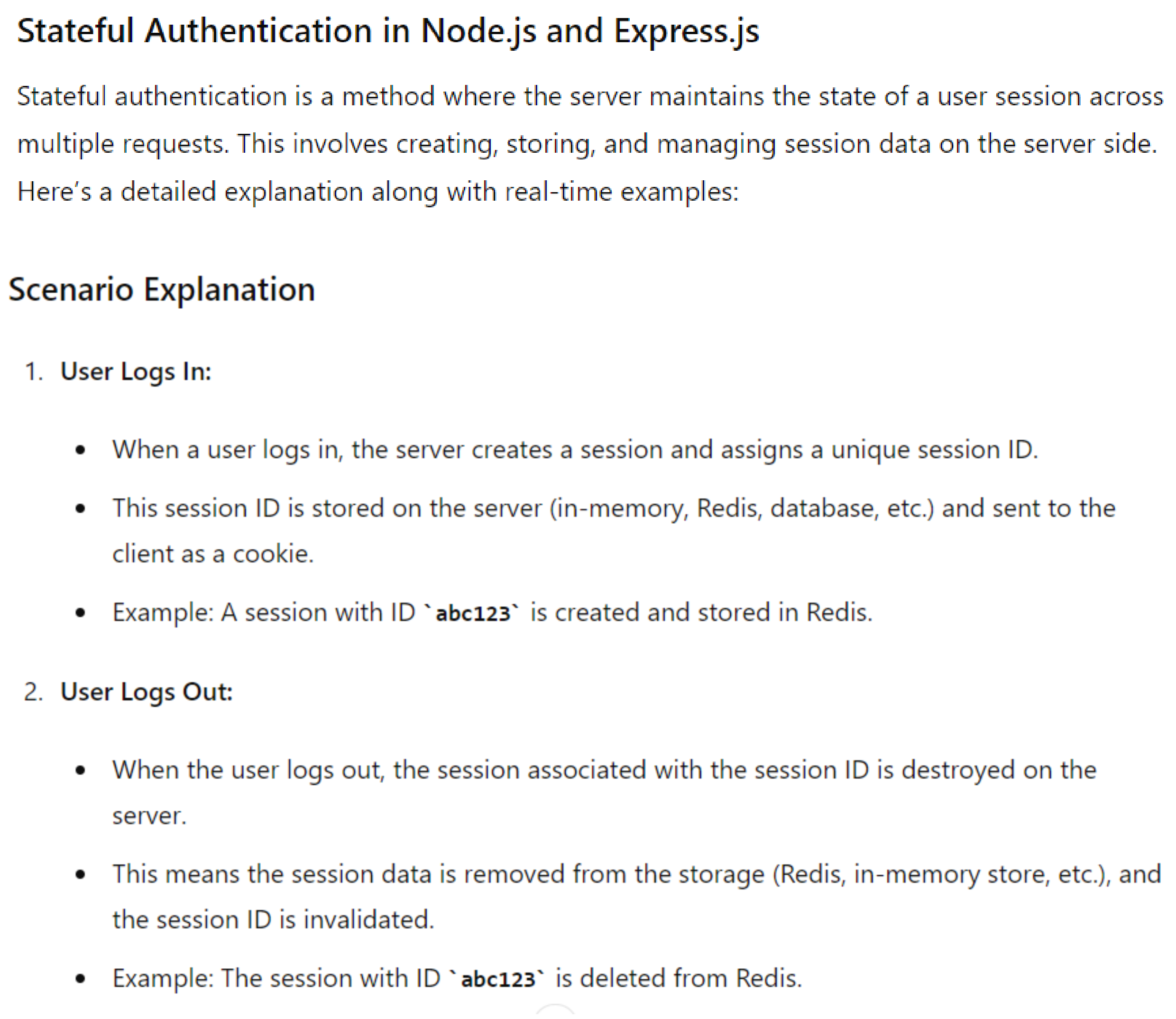
AUTHECATION CONTAINS TWO TYPES →

STATEFULL

STATELTSS → JWT

STATEFUL AUTHECATION



Stateful Authentication in Node.js and Express.js

Stateful authentication is a method where the server maintains the state of a user session across multiple requests. This involves creating, storing, and managing session data on the server side. Here's a detailed explanation along with real-time examples:

Scenario Explanation

1. **User Logs In:**
   - When a user logs in, the server creates a session and assigns a unique session ID.
   - This session ID is stored on the server (in-memory, Redis, database, etc.) and sent to the client as a cookie.
   - Example: A session with ID **`abc123`** is created and stored in Redis.
2. **User Logs Out:**
   - When the user logs out, the session associated with the session ID is destroyed on the server.
   - This means the session data is removed from the storage (Redis, in-memory store, etc.), and the session ID is invalidated.
   - Example: The session with ID **`abc123`** is deleted from Redis.

3. **User Logs In Again After Some Time:**
    - When the user logs in again after logging out, a new session is created with a new unique session ID.
    - The previous session ID (`` `abc123` ``) is no longer valid and is not reused.
    - Example: A new session with ID `` `def456` `` is created and stored in Redis.

Key Points:

- **Session ID Uniqueness:** Each login creates a unique session ID. Even if the same user logs in multiple times, each session is distinct.
- **Session Destruction:** Logging out removes the session data from the server, ensuring that the session ID cannot be reused.
- **Security:** Reusing session IDs can pose security risks, so generating a new session ID upon each login is a good practice.

STATELESS AUTHECATION → JWT

JWT Tokens in Stateless Authentication: Simple Explanation

JSON Web Tokens (JWT) are a way to securely transmit information between parties as a JSON object. They are used for stateless authentication, meaning the server does not store any session information about the user. Instead, the information is stored in the token itself, which the client sends with each request.

Steps and Scenarios

1. User Login

Scenario: User logs into a web application (e.g., an online bookstore).

1. **User sends login credentials** (username and password) to the server.
2. **Server verifies credentials**:
    - If valid, the server creates a JWT.
3. **Server sends the JWT** back to the client.

2. Storing the JWT

Scenario: The client receives the JWT and stores it for future requests.

1. **Client stores the JWT** in local storage or a cookie.

3. Making Authenticated Requests

Scenario: User wants to view their profile.

1. **Client sends a request** to the server with the JWT in the Authorization header.
2. **Server verifies the JWT**:
   - If valid, the server processes the request.
   - If invalid or expired, the server rejects the request.

4. User Logout

Scenario: User logs out from the application.

1. **Client deletes the JWT** from local storage or cookie.
2. **Server doesn't need to do anything** because it does not store session data.

- **Use Stateful Authentication** when:
    - The application requires complex state management.
    - You need to handle sensitive data securely.
    - User interactions are complex and involve multiple steps or persistent data.
- **Use Stateless Authentication** when:
    - Scalability is a priority.
    - The architecture is distributed or involves microservices.
    - The application involves APIs or mobile clients where maintaining server-side sessions is impractical.

Choosing between stateful and stateless authentication depends on the specific needs of your application, including scalability, security, and the complexity of user interactions.

For security, **stateful authentication** is generally preferred.

- **Reason:** Stateful authentication allows for more centralized control over session data and easier management of session invalidation. This is crucial in scenarios where immediate revocation of access (logging out) or monitoring of active sessions is important for security reasons, such as in banking applications or systems dealing with sensitive data. Stateful authentication simplifies auditing and ensures that session data is securely stored and managed on the server side, reducing risks associated with client-side token management.