🔐 Authentication Basics (README)

This README explains **why simple yes/no authentication is insecure**, what **server-side sessions** are, their problems, and how **authentication tokens (JWT)** work and why they are better for modern applications.

🕯️ Why a YES or NO reply from the server is not good enough?

When a user logs in, the server checks the email and password and may respond with **YES (authenticated)** or **NO (not authenticated)**.

🗄️ Why this is insecure

- A simple **YES** does not prove identity in future requests
- Anyone can fake a request and send `loggedIn=true`
- The server has no way to verify if the request is really from an authenticated user

🔑 Key Problem

> There is **no proof** attached to future requests.

So, authentication must provide **verifiable proof**, not just a one-time answer.

🖥️ What are Server-Side Sessions?

Server-side sessions are a traditional authentication method where:

🔦 How it works

1. User logs in with email & password
2. Server verifies credentials
3. Server creates a **session** and stores it
4. A **session ID** is sent to the client (usually via cookies)
5. Client sends the session ID with every request
6. Server checks session ID to verify the user

Example

```
Session ID: abc123
User: Parvez
```

The session data is stored **on the server**.

🔍 What is the problem with Server-Side Sessions?

🔩 Drawbacks

- Server must store session data (memory/database)
- Difficult to scale for large applications
- Not ideal for mobile apps and APIs
- Requires session cleanup and management
- Slower in distributed systems

📉 Scalability Issue

When users increase, server load increases because **sessions are stored and checked on every request**.

🔑 What are Authentication Tokens?

Authentication tokens (commonly **JWT – JSON Web Tokens**) are **self-contained proof of authentication**.

🔦 How tokens work

1. User logs in
2. Server verifies credentials
3. Server creates a **token** containing user info and expiry
4. Token is sent to the client
5. Client stores the token (memory / localStorage / cookie)
6. Token is sent with every request
7. Server verifies the token

Example Token Usage

```
Authorization: Bearer <JWT_TOKEN>
```

The server **does not store session data**.

🕐 How are Authentication Tokens better than Server-Side Sessions?

Feature	Server-Side Sessions	Authentication Tokens
Storage	Stored on server	Stored on client
Scalability	Poor	Excellent

Feature	Server-Side Sessions	Authentication Tokens
Mobile/API Friendly	❌ No	✅ Yes
Server Load	High	Low
Stateless	❌ No	✅ Yes
Modern Usage	Less	Widely used

📀 Why tokens are preferred today

- No server-side storage
- Easy to scale
- Secure (signed & expiring)
- Perfect for React, mobile apps, microservices

One-Line Summary

Authentication tokens provide **secure, scalable, and verifiable proof** of identity, unlike simple yes/no responses or server-side sessions.

🗜️ Recommended for Modern Apps

✅ Use **JWT-based authentication** for: - React applications - REST APIs - Mobile apps - SaaS products

---

End of README