Lec-5 Cookie, Login & Authentication

Agenda

- Cookies and it's usecase
- JWT token
- Signup
- Login
- Protect route
- Logout

Cookies and it's usecase

Cookies are small text files that are stored on your computer by websites you visit. They serve as a way for websites to remember information about you and your preferences, enhancing your browsing experience.
1994 - Cookie

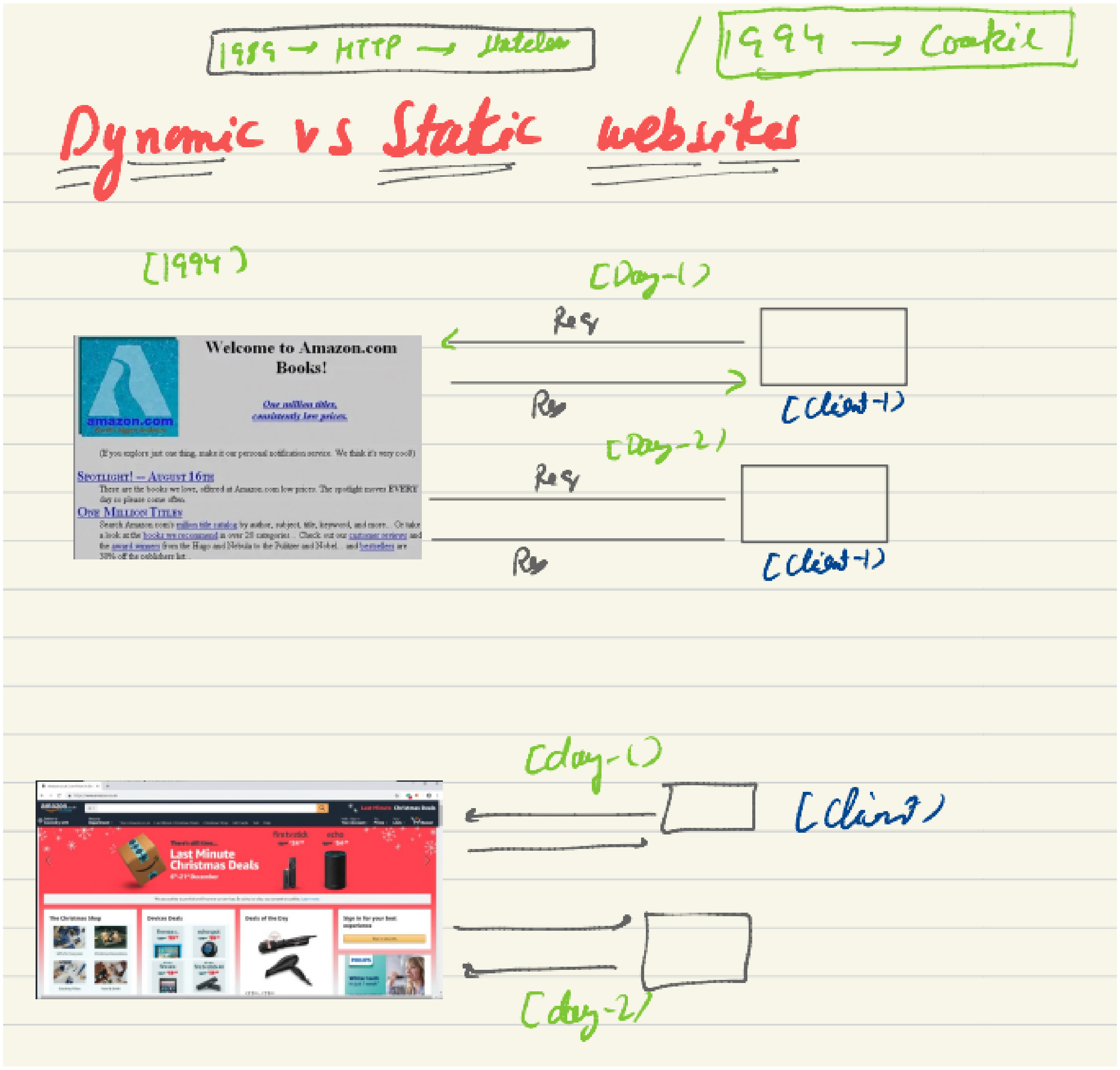
Dynamic vs Static website

Static Websites

The content of a static website remains the same for all visitors unless manually updated.
Examples: Basic landing pages, simple portfolios, and brochure websites.

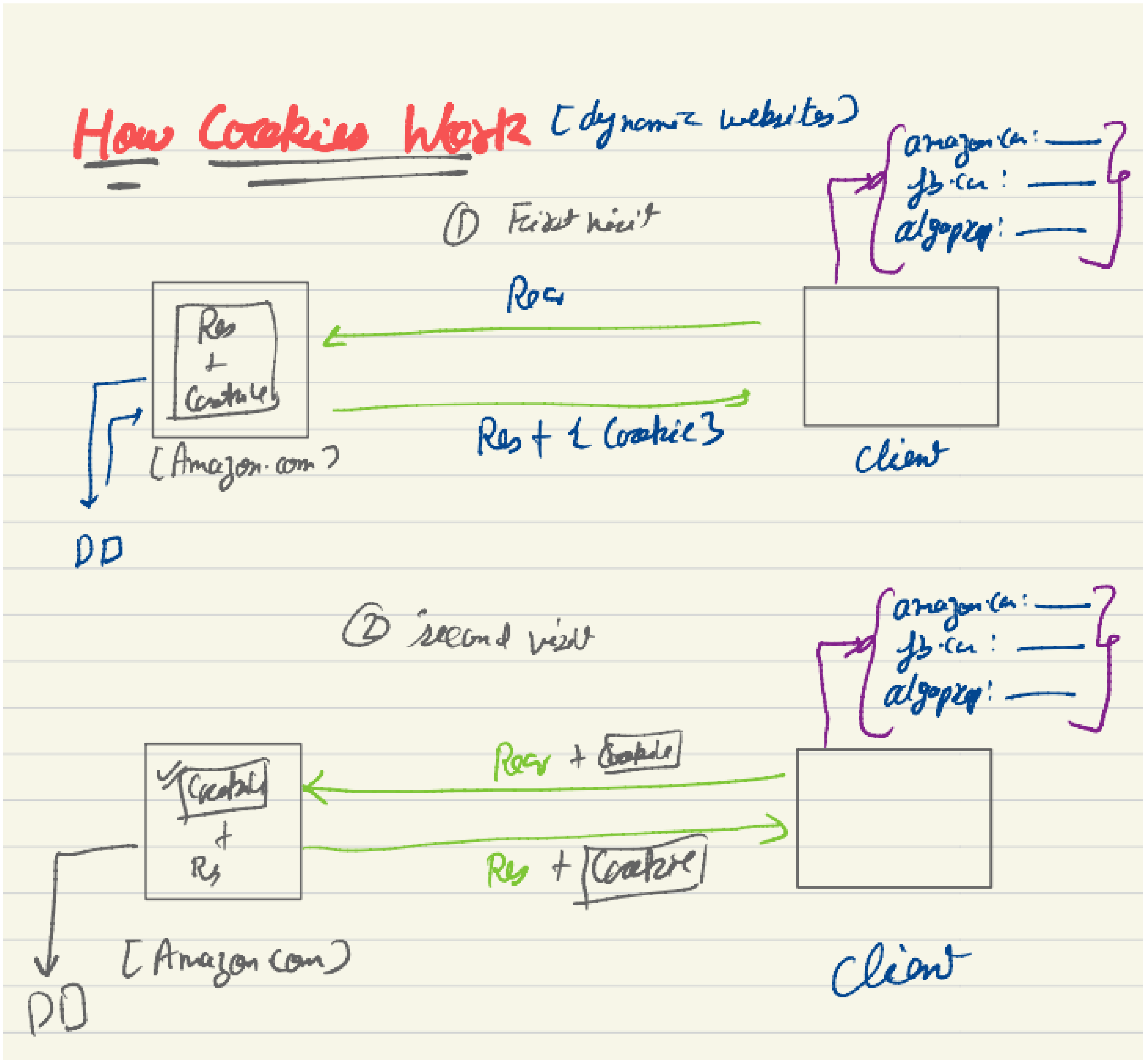
Dynamic Websites

The content of a dynamic website can change based on various factors, such as user interactions, database queries, or server-side scripts.
Examples: E-commerce websites, blogs, social media platforms, and web applications.



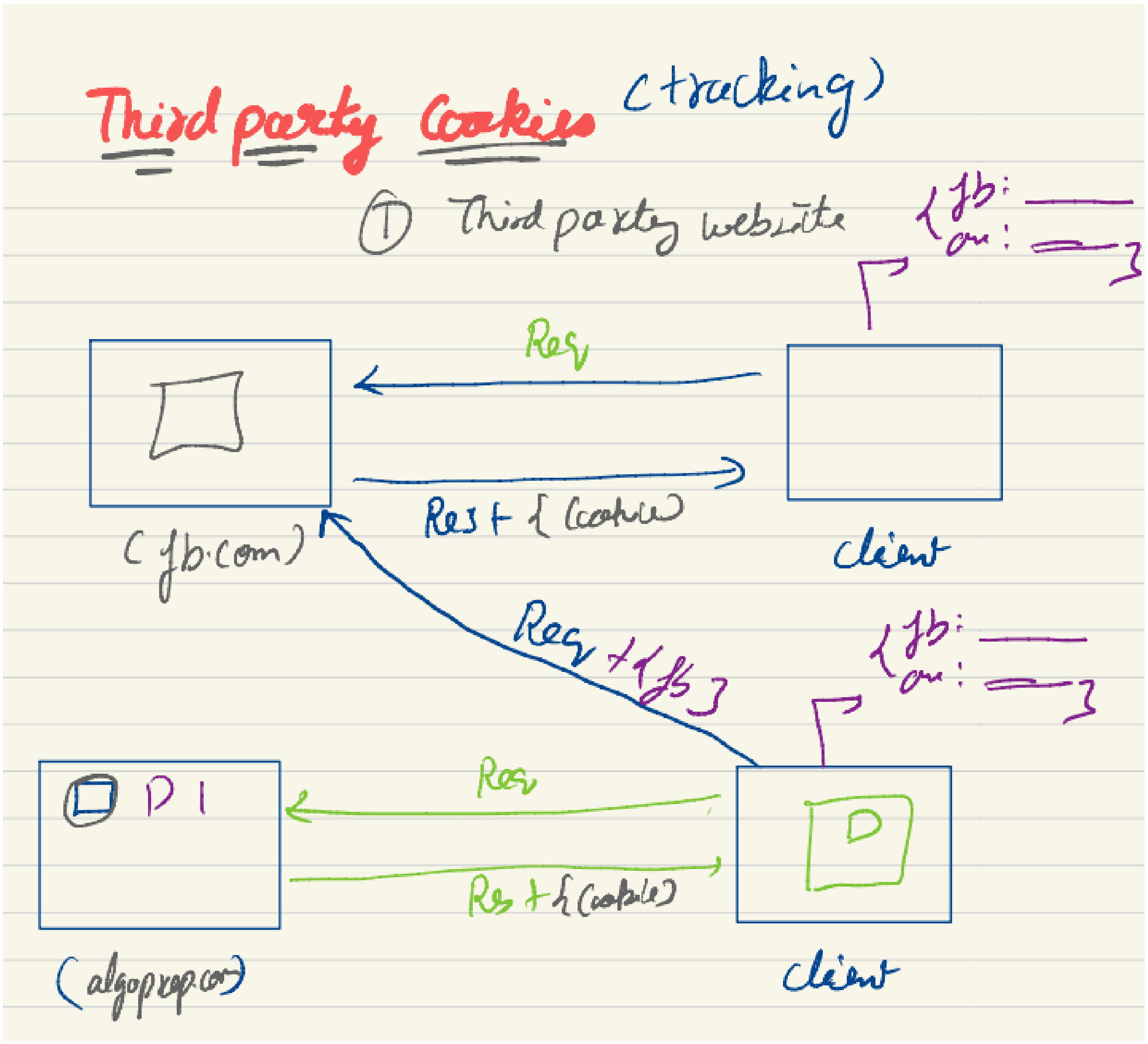
How cookies work

In essence, cookies provide a way for dynamic websites to remember information about users, personalize their experiences, and track their behaviour, leading to a more engaging and tailored interaction.



Third party cookies

Third-party cookies are cookies that are set by a domain different from the one you are currently visiting. These cookies are often used for tracking purposes, advertising, and analytics.



Cookies with express

Cookie-parser

`cookie-parser` is a middleware for Express.js that parses cookies attached to the client request object (`req`). It makes cookies available on `req.cookies` and can also sign cookies if desired, providing additional security.

Installation:

```
npm install cookie-parser
```

cookie.js

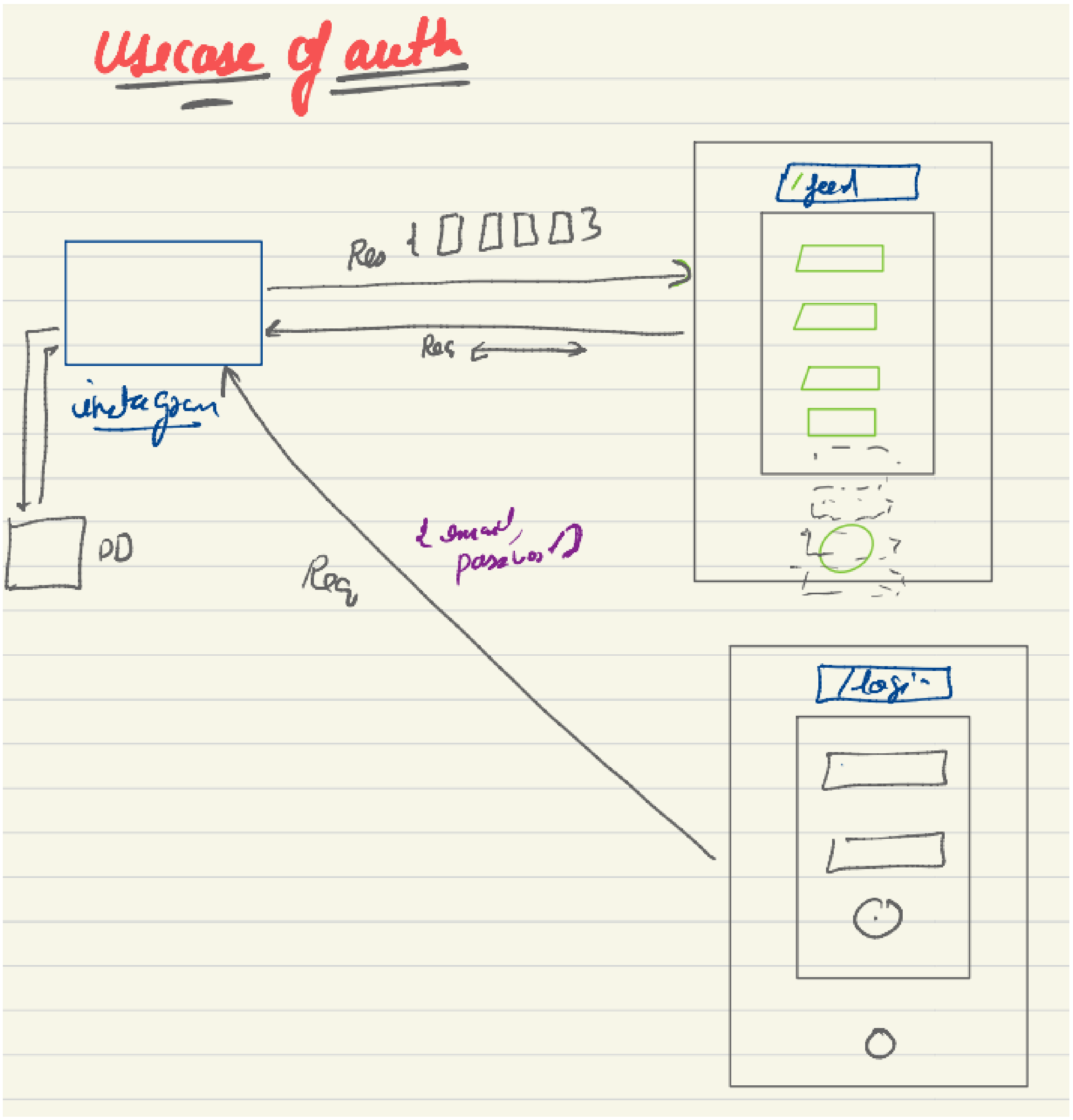
```
const express = require("express");
const app = express();
const cookieParser = require("cookie-parser");

app.use(cookieParser());
app.get("/", function(req,res){
    console.log("get request received");
    // res cookie
    res.cookie("prevpage","home",{
        maxAge: 1000 * 60 * 60 * 24,
    })

    res.status(200).json({
        message:"received request at home page"
    })
});

app.get("/product",function(req,res){
    let messageStr = "";
    if(req.cookies && req.cookies.prevpage){
        messageStr = `You visited ${req.cookies.prevpage} page before`
    }else{
        messageStr = "No previous page found."
    }
    res.status(200).json({
        message:messageStr
    })
})

app.listen(3000,function(){
    console.log("server is running at 3000 port");
})
```

For clearing the cookies

```
app.get("/clearCookies",function(req,res){
   res.clearCookie('prevpage',{ path: "/"});
    res.status(200).json({
        message:"I have cleared your cookie"
    })
})
```

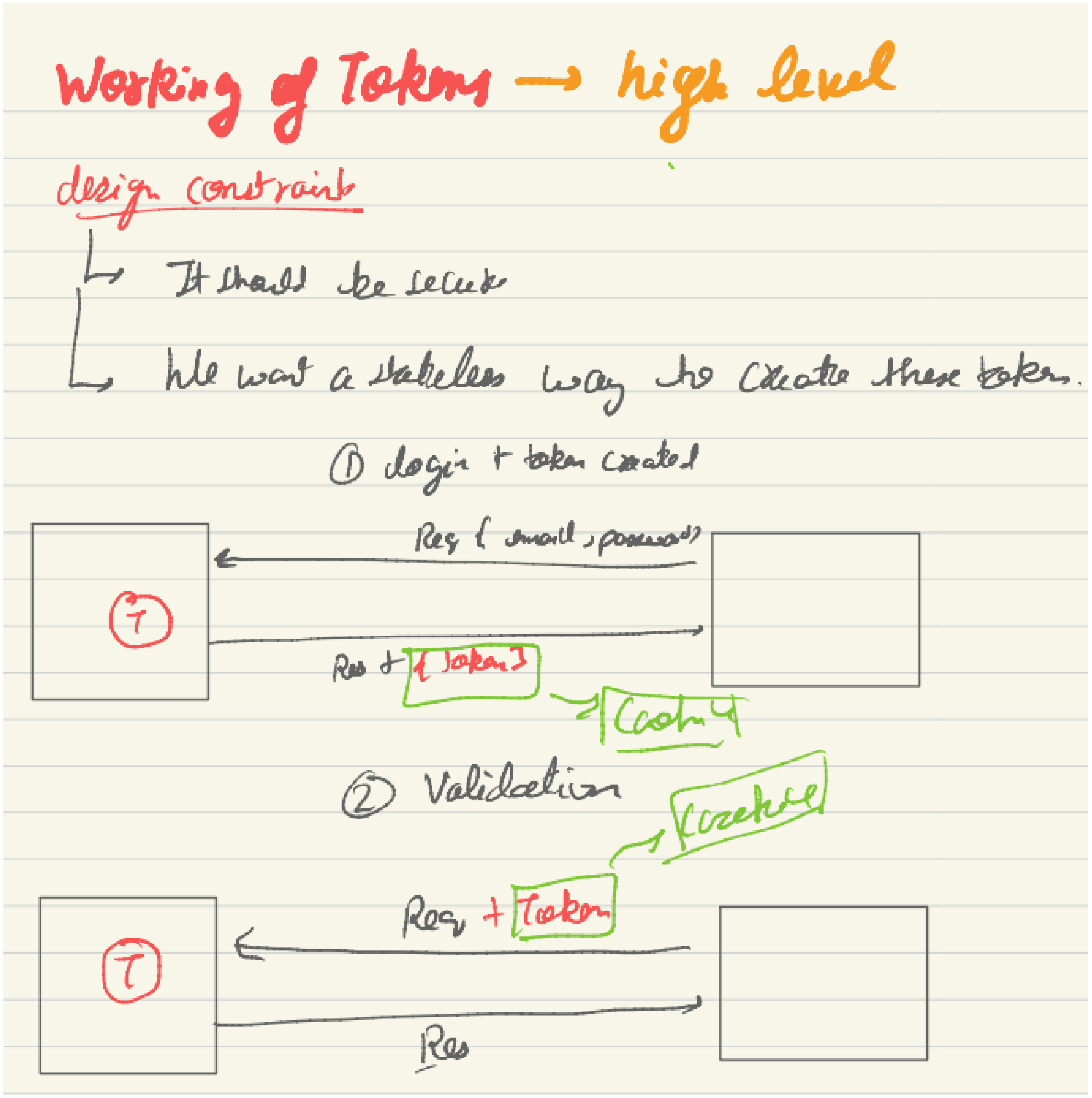
Authentication

Cookies are commonly used for authentication in web applications due to their ability to store session information on the user's device.



Working of Token

It should be secure.
We want a stateless way to create these token.


Working of Tokens → high level

design constraint

It shall be secure

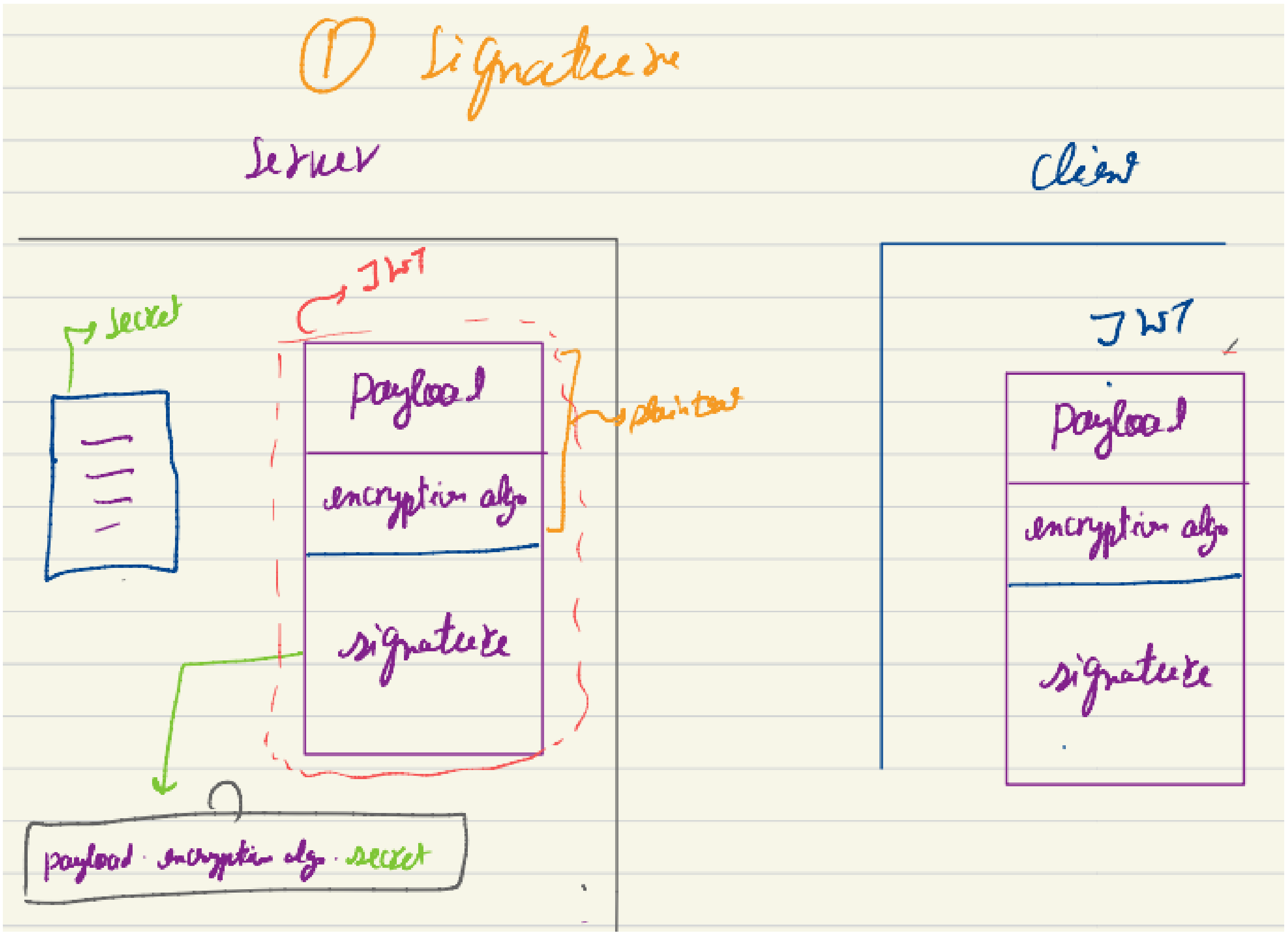
We want a stateless way to create these token.

Token Creation

```
const express = require("express");
const app = express();
const cookieParser = require("cookie-parser");
const jwt = require("jsonwebtoken");

const util = require("util");
const promisify = util.promisify;
const promisifiedJWTsign = promisify(jwt.sign);
app.use(cookieParser());

const SECRET_KEY = "abrakadabra"
```

```
// token creation
app.get("/sign",async function(req, res){
    // token create
    const authToken = await promisifiedJWTsign({"payload":
"aadfdkjfh"},SECRET_KEY)
    // token -> cookies
    res.cookie("jwt", authToken, {
        maxAge: 1000 * 60 * 60 *24,
        httpOnly:true, // it can only be accessed by the server
    })
    // res send
    res.status(200).json({
        message:"signed the jwt and sending it in the cookie"
    })

})
app.listen(3000,function(){
    console.log("server is running at 3000 port");
})
```

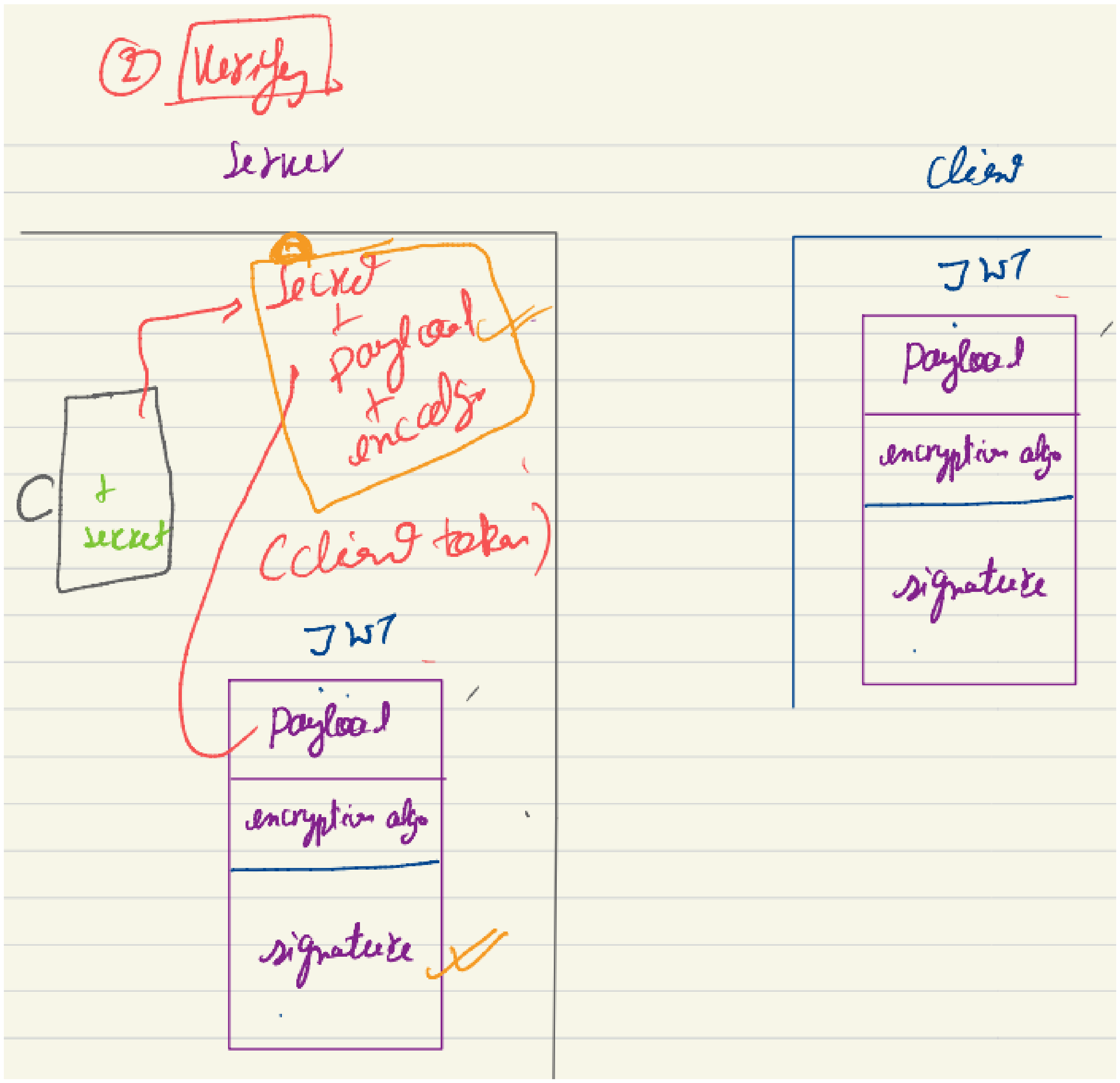
Let's understand with diagram.



Token Verification

```
const promisifiedJWTverify = promisify(jwt.verify);
app.get("/verify", async function(req, res){
    if(req.cookies && req.cookies.jwt){
        const authToken = req.cookies.jwt;
        const unlockedToken = await promisifiedJWTverify(authToken,SECRET_KEY);
        res.status(200).json({
            message:"jwt token is verified",
            "unlockedToken":unlockedToken
        })
    }else{
        res.status(400).json({
            message:"no jwt token found"
        })
    }
})
```

Let's understand with diagram



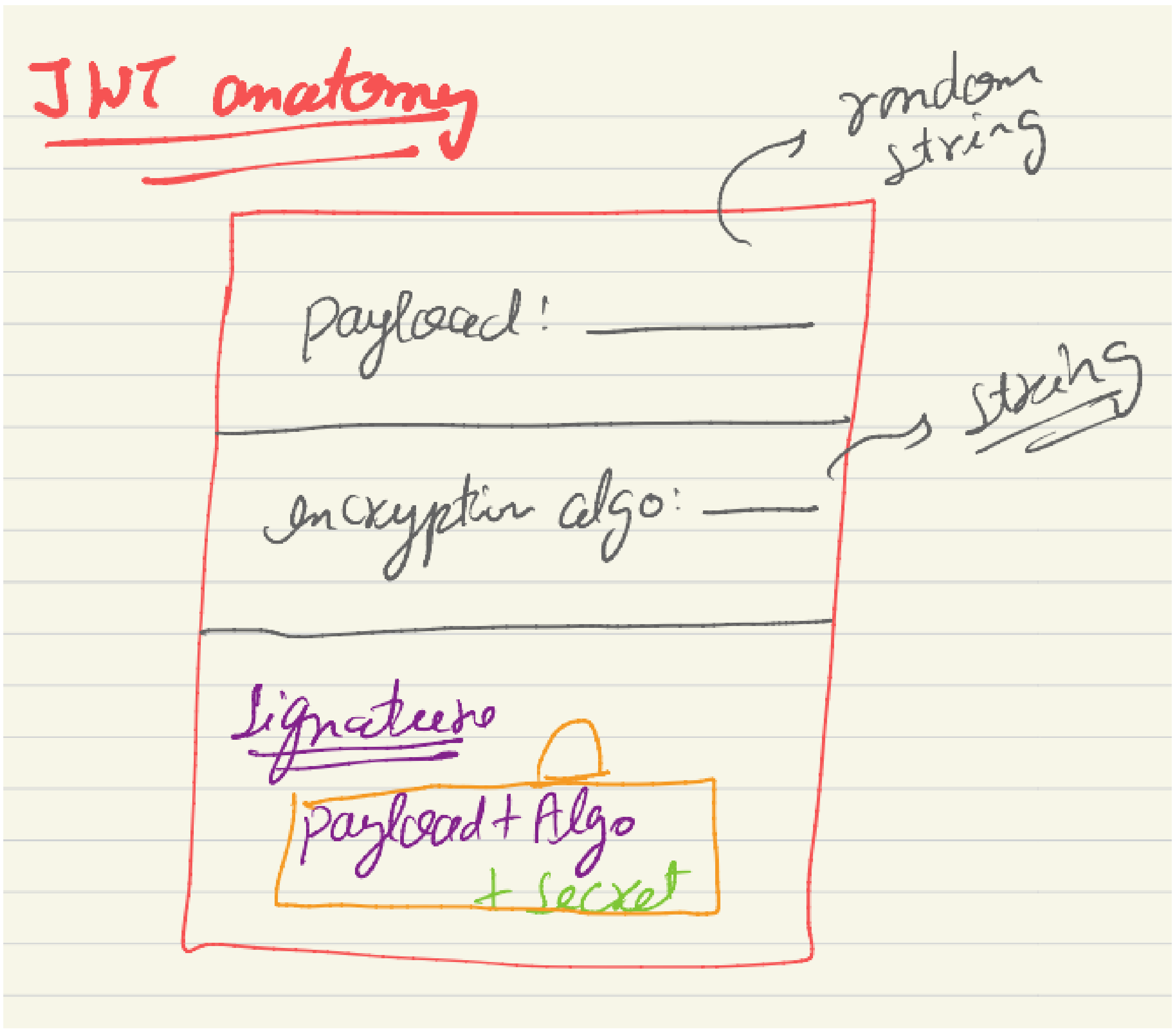
JWT Anatomy

JSON Web Token (JWT) is a standard for securely transmitting information between parties as a JSON object. It's commonly used for authentication and authorization in web app.

A JWT consists of three parts.

- **Header:** Contains metadata about the token, including the algorithm used to sign the token and the token type.
- **Payload:** Contains the claims or data to be transmitted. These can include user information, permissions, or expiration times.

- **Signature:** A cryptographic signature that verifies the integrity of the token and ensures it hasn't been tampered with.



Signup/Login

- **User Signup:**
    - When a user signs up, their information (e.g., email, password) is stored in a database.
    - A unique user identifier (e.g., user ID) is generated for the user.
- **Token Generation:**
    - Upon successful login, the server generates a JWT containing information about the user, such as their user ID and any relevant permissions.
    - This token is typically signed using a secret key to ensure its authenticity and integrity.
- **Token Storage and Transmission:**

    - The JWT is sent to the client (usually stored in a session cookie or local storage).
    - Subsequent requests from the client include the JWT in an authorization header.
- **Token Verification:**
    - On the server-side, the JWT is verified by validating its signature and ensuring it hasn't expired.
    - If the token is valid, the server extracts the user information from the payload and grants the user access to the requested resources.

Profile

- Add middleware function to verify the token .
- If you logged in then allow access otherwise return back.

Time stamp

- Cookies and it's usecase (0:09:38 - 0:51:00)
- Authentication (0:52:00 - 1:00:00)
- Working of Token (1:00:00 - 1:26:20)
- JWT Anatomy (1:26:20 - 1:36:40)
- Login/Signup (1:38:00 - 1:40:40)
- Profile (1:40:40 - end)