Security:

- You have absolutely no control on the client.
- Users can modify the frontend code.

Cookies:

- **Introduction to cookies:**
- **Cookies** are text files with small pieces of key-value pairs of data that are used to identify your computer as you use a computer network. Data stored in a cookie is created by the server upon your connection. This data is labeled with an ID unique to you and your computer. When the cookie is exchanged between your computer and the network server, the server reads the ID and knows what information to specifically serve to you.
- Cookies are embedded in the headers of HTTP requests and responses.
  I.e. Cookies are key/value pairs sent back and forth between the browser and the server in HTTP request and response.
- Cookies:
  - Contain text data (Up to 4kb)
  - May or may not have an expiration date
  - Are bound to a domain name and a path.
    I.e. Every website has cookies and your browser links/connects that website with those cookies.
  - May have security flags
  - Can be manipulated from the client and the server
- Cookies are good for:
  - Shopping cart
  - Browsing preferences
  - User authentication
  - Tracking and advertisement
- The purpose of the computer cookie is to help the website keep track of your visits and activity. For example, many online retailers use cookies to keep track of the items in a user's shopping cart as they explore the site. Without cookies, your shopping cart would reset to zero every time you clicked a new link on the site.
- A website might also use cookies to keep a record of your most recent visit or to record your login information. Many people find this useful so that they can store passwords on frequently used sites, or simply so they know what they have visited or downloaded in the past.
- There are 2 main types of cookies:
  1. **Magic Cookies:**
  - Magic cookies are an old computing term that refers to packets of information that are sent and received without changes. This concept predates the modern cookie we use today.
  2. **HTTP Cookies:**
  - HTTP cookies are a repurposed version of the magic cookie built for internet browsing. Web browser programmer Lou Montulli used the magic cookie as inspiration in 1994. He recreated this concept for browsers when he helped an online shopping store fix their overloaded servers.
  - HTTP cookies, or internet cookies, are built specifically for internet web browsers to track, personalize, and save information about each user's session. A session just refers to the time you spend on a site.
  - Cookies are created to identify you when you visit a new website. The web server sends a short stream of identifying info to your web browser.

    - Browser cookies are identified and read by name-value pairs. These tell cookies where to be sent and what data to recall.
- Websites use HTTP cookies to streamline your web experiences. Without cookies, you'd have to login again after you leave a site or rebuild your shopping cart if you accidentally close the page.
- **Here's how HTTP cookies are intended to be used:**
    1. **Session management:** For example, cookies let websites recognize users and recall their individual login information and preferences.
    2. **Personalization:** Customized advertising is the main way cookies are used to personalize your sessions. You may view certain items or parts of a site, and cookies use this data to help build targeted ads that you might enjoy.
    3. **Tracking:** Shopping sites use cookies to track items users previously viewed, allowing the sites to suggest other goods they might like and keep items in shopping carts while they continue shopping.
- **Different types of HTTP cookies:**
- There are 2 main types of cookies:
    1. **Session cookies** are used only while navigating a website. They are stored in random access memory and are never written to the hard drive.
       When the session ends, session cookies are automatically deleted. They also help the "back" button or third-party anonymizer plugins work. These plugins are designed for specific browsers to work and help maintain user privacy.
    2. **Persistent cookies** remain on a computer indefinitely, although many include an expiration date and are automatically removed when that date is reached.
       Persistent cookies are used primarily for authentication and tracking.
       For authentication, cookies track whether a user is logged in and under what name. They also streamline login information, so users don't have to remember site passwords.
       For tracking, cookies track multiple visits to the same site over time. For example, some online merchants use cookies to track visits from particular users, including the pages and products viewed. The information they gain allows them to suggest other items that might interest visitors. Gradually, a profile is built based on a user's browsing history on that site.
- **Why cookies can be dangerous:**
- While cookies can't infect computers with viruses or other malware, the danger lies in their ability to track individuals' browsing histories.
- **First-party cookies** are directly created by the website you are using. These are generally safer, as long as you are browsing reputable websites or ones that have not been compromised.
- **Third-party cookies** are more troubling. They are generated by websites that are different from the web pages users are currently surfing, usually because they're linked to ads on that page. Visiting a site with 10 ads may generate 10 cookies, even if users never click on those ads. Third-party cookies let advertisers or analytics companies track an individual's browsing history across the web on any sites that contain their ads. Consequently, the advertiser could determine that a user first searched for running apparel at a specific outdoor store before checking a particular sporting goods site and then a certain online sportswear boutique.
- **Zombie cookies** are from a third-party and permanently installed on users' computers, even when they opt not to install cookies. They also reappear after they've been deleted. Like other third-party cookies, zombie cookies can be used by web analytics companies to track unique individuals' browsing histories. Websites may also use zombies to ban specific users.

- **Manipulating cookies:**
- A cookie can be modified, as long as there is no cookie flag set.
- On the server side, we can use cookie in Express.
  E.g. **const cookie = require('cookie');** ← Used in Express.
- On the client side, we can use Document.cookie in Javascript.

Sessions:

- **Introduction to sessions:**
- A **web session** is a series of contiguous actions by a visitor on an individual website within a given time frame. This could include your search engine searches, filling out a form to receive content, scrolling on a website page, adding items to a shopping cart, researching airfare, or which pages you viewed on a single website. Any interaction that you have with a single website is recorded as a web session to that website property.
- To track sessions, a **web session ID** is stored in a visitor's browser. This session ID is passed along with any HTTP requests that the visitor makes while on the site.
- To avoid storing massive amounts of information in-browser, developers use session IDs to store information server-side while enabling user privacy. Every time a user takes an action or makes a request on a web application, the application sends the session ID and cookie ID back to the server, along with a description of the action itself.
- Once a web developer accrues enough information on how users traverse their site, they can start to create very personalized, engaging experiences.
- A session can be defined as a server-side storage of information that is desired to persist throughout the user's interaction with the web site or web application.
- Instead of storing large and constantly changing information via cookies in the user's browser, only a unique identifier is stored on the client side called a session id. This session id is passed to the web server every time the browser makes an HTTP request. The web application pairs this session id with it's internal database and retrieves the stored variables for use by the requested page.
- **General concepts of sessions:**
- There is a session id, a token, between the browser and the web application.
- The session id should be unique and unforgeable. It is usually a long random number or a hash.
- The session id is bound to key/value pairs data.
- The session id is stored in a cookie while session key/value pairs are stored on the server.
- The user can create, modify, delete the session ID in the cookie, but cannot access the key/value pairs stored on the server.

Web Authentication:

- There are several ways to do web authentication:
  1. **Local authentication:**
     - Manage username and password yourself.
       I.e. A user creates an account. You store the username and password in a database, and each time a user wants to log in, you compare the inputted username and password to the ones you have in the database.

- **How to store passwords:**

- **Salted hash** means that we're going to add a random string (the salt) to the password to make it strong. Each new salted password will be unique as the salt is always unique. Then, we'll hash the salted password. Salted hash is resistant to brute force attacks.
- **Basic/Stateless Authentication:**
- (Standard) RFC 2617
- Login and password are sent in clear (Base64 encoding) in the headers "authorization".
- Pros:
  - Since there aren't many operations going on, authentication can be faster with this method.
  - Easy to implement.
  - Supported by all major browsers.
- Cons:
  - Base64 is not the same as encryption. It's just another way to represent data. The base64 encoded string can easily be decoded since it's sent in plain text. This poor security feature calls for many types of attacks. Because of this, HTTPS/SSL is absolutely essential.
  - Credentials must be sent with every request.
  - Users can only be logged out by rewriting the credentials with an invalid one.
- **Session/Stateful Authentication:**
- (Standard) RFC 6265
- Uses cookies.
- The user enters a login and password and the frontend sends them to the backend (POST request).
  Then, the backend verifies the login/password based on information stored on the server (usually in the database).
  Then, the backend stores user information in a session.
  Then, the backend grants access to resources based on the information contained in the session.
- Pros:
  - Faster subsequent logins, as the credentials are not required.
  - Improved user experience.
  - Fairly easy to implement. Many frameworks provide this feature out-of-the-box.

- Cons:
    - Cookies are sent with every request, even if it does not require authentication

- **Do/Don't with passwords:**
- On the client side, either send passwords in the headers (automatic with basic authentication) or in the body (POST request with session authentication). Never send/show passwords in the URL.
- On the server, store passwords as salted hash passwords only. Never store passwords in clear or non-salted hash.

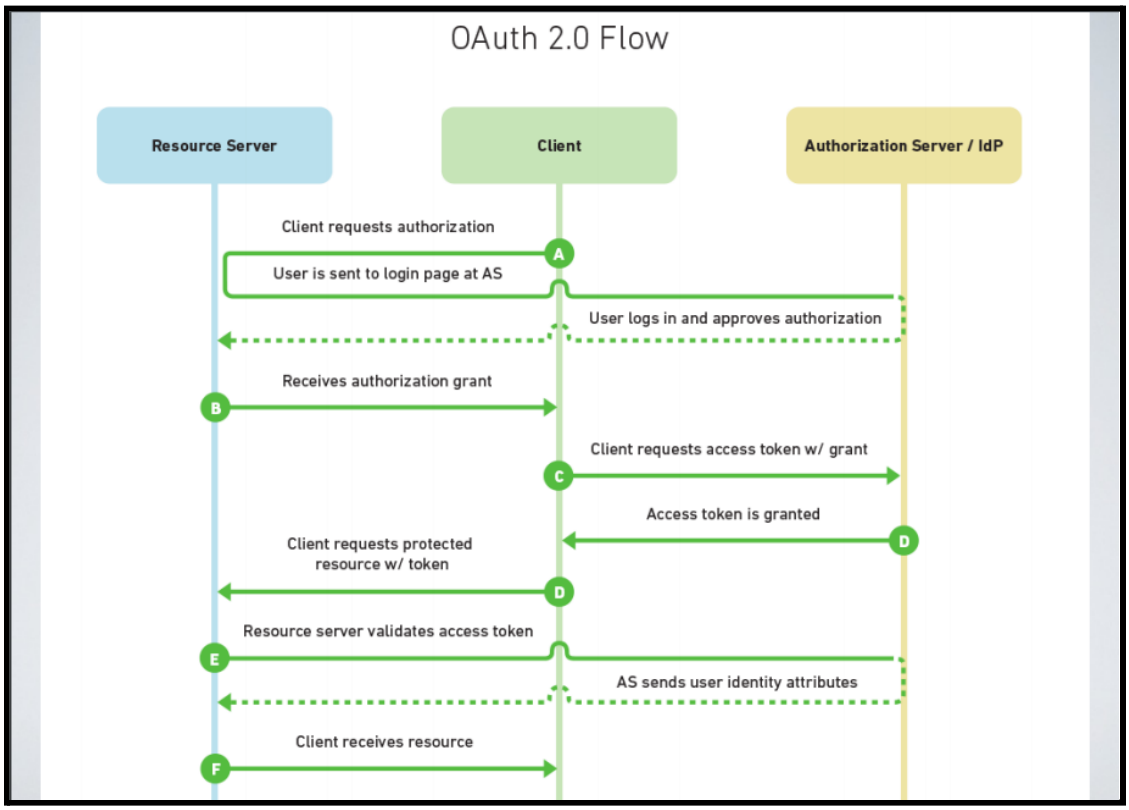
2. **Token-based authentication:**

- This method uses tokens to authenticate users instead of cookies. The user authenticates using valid credentials and the server returns a signed token. This token can be used for subsequent requests.
- **HMAC:**
- (Standard) RFC 2104
- For each authenticated HTTP request, the frontend computes and sends a message digest that combines the user's secret and some request arguments.
- The user's password never transits back and forth except perhaps for the first time it is exchanged.
- The digest can be sent in clear. Should not store sensitive information in the digest.
- **JSON Web Token:**
- (Standard) RFC 7519
- Encodes the user's information in a string (token) that is URL safe.
- The token is usually authenticated and sometimes encrypted.
- The web token can be used for stateful but yet session-less authentication.
- Stateless JSON Web Token is a self-contained token which does not need any representation on the backend.
- Stateful JSON Web Token is a token which contains only part of the required data such as session/user ID and the rest is stored on the server side.
- Revoking tokens can be complicated.

3. **Third party authentication:**

- You can sign into a website through signing into another website.
- **Single sign-on (SSO)** is an authentication scheme that allows a user to log in with a single ID and password to any of several related, yet independent, software systems.
- There are many types of SSO, such as Pubcookie, OpenID, SAML, OAuth, among others.
- **Social login** is a form of single sign-on using existing information from a social networking service such as Facebook, Twitter or Google, to sign into a third party website instead of creating a new login account specifically for that website. It is designed to simplify logins for end users as well as provide more and more reliable demographic information.
- **OpenID:**
- An HTTP based protocol that uses an identity provider to validate a user. The user's password is secured with one identity provider. This allows other service providers a way to achieve SSO without requiring a password from the user.
- **SAML:**
- Is XML based.
- Is used in many enterprise applications to enable enterprises to monitor who has access to corporate resources.

- **OAuth 2.0:**
- (Standard) RFC 6749
- Is JSON based.
- OAuth 2.0 is a security standard where you give one application permission to access your data in another application. You authorize one application to access your data, or use features in another application on your behalf, without giving them your password. OAuth 2.0 does this by allowing a token to be issued by the identity provider to these third party applications, with the approval of the user.
- OAuth doesn't share password data but instead uses authorization tokens to prove an identity between consumers and service providers. OAuth is an authentication protocol that allows you to approve one application interacting with another on your behalf without giving away your password.
- How it works:
    1. The backend redirects the user to the third-party login-page.
    2. The third-party asks and verifies the login/password based on the third party user information.
    3. The third party redirects the user back to the application with an OAuth token and verifier in the url.
    4. Backend verifies the token with the third party.
    5. Backend starts a session.



- The user's login/password never transit by the application's frontend or backend.