All you need to know about Authentication is here

Sep 10, 2019 | 11 minute read

Login and Logout is the heart of web application features. Developers will often feel like an application is incomplete without having the ability for users to login and logout, as it creates personalization within the application and makes the application more secure and robust. Therefore, understanding the authentication mechanism is vital for all developers to create a reliable application. Authentication is heavily implemented in nearly all modern applications. Based on the type of the application and the implementation choice of the developers, maintaining a session can create a secure application or increase the vulnerability of an application. As a developer, there has been a lot of discussion lately regarding where to store the user's authentication token. Does Cookie-based or Token-based authentication create a more secure application? Is using Cookies safer, how about local storage?

What is the preferred authentication mechanism and method for a session to be maintained in an application?

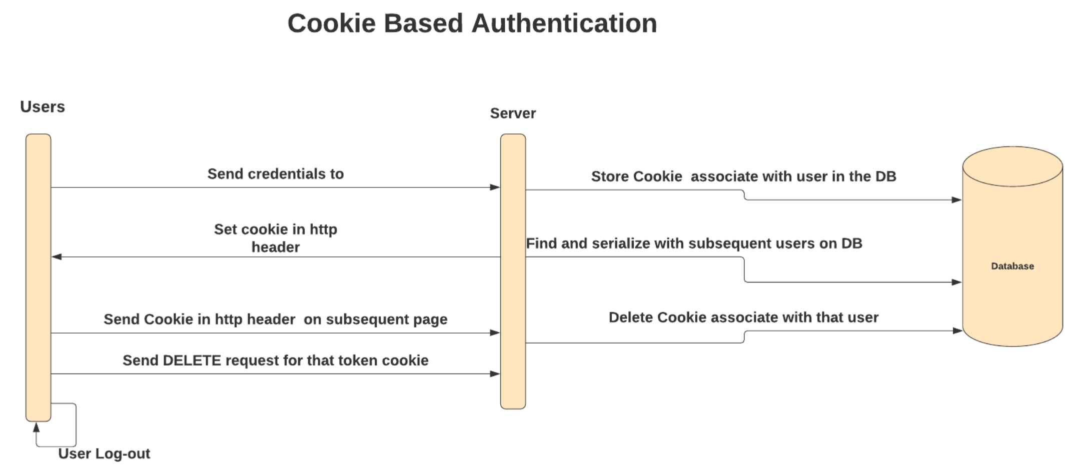
Before we deep dive into which authentication mechanism is preferred for an application, and discuss where we should store the authentication token, we need to understand the various types of authentication and how it works.

There are two types of authentication – cookie-based authentication and token-based authentication. I will quickly explain how these two authentication methods work. Then, I'll deep dive into the pros and cons of implementing either one of these authentications, so that you'll know how you can store authentication tokens for your application.

Cookie-based Authentication

Cookie-based authentication is stateful, meaning that the client and server will need to keep the token to manage a session between pages for a user.

A cookie is a name value pair of the user's unique identifier and generated token that has an expiry date. The cookie is typically stored on both the client and server. The server will store the cookie in the database, to keep track of each user session, and the client will hold the session identifier.

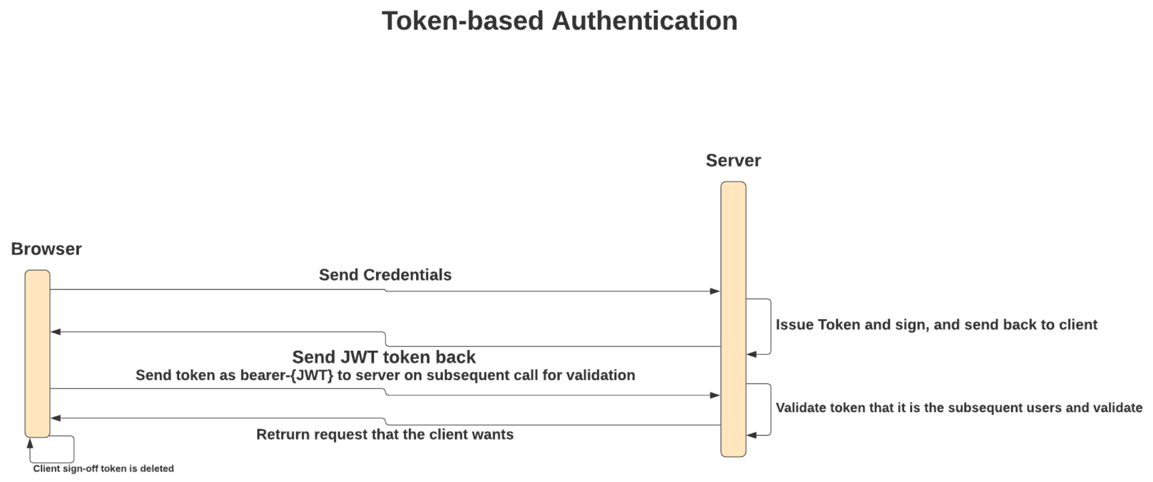

Cookie Based Authentication
Users
Server
Send credentials to
Store Cookie associate with user in the DB
Set cookie in http header
Find and serialize with subsequent users on DB
Database
Send Cookie in http header on subsequent page
Delete Cookie associate with that user
Send DELETE request for that token cookie
User Log-out

Cookie-based authentication works like this:

1. User logins by entering credentials
2. Server verifies that the user's credentials are correct, and creates a cookie with the session info which is stored in the database
3. The cookie, with the session ID and other information, is also stored in the browser
4. When the user navigates through various pages on the browser, the cookie is verified against the database to validate if the user credentials are valid
5. When the user logs out, the session is also deleted from the database

Token-based Authentication

When we talk about token-based authentication, we often refer to JWT (JSON Web Token), because it has been widely used in all industries and has become a de-facto standard for authentication. JWT is an open standard that defines a compact, secure, and self-contained way to transmit data between parties in JSON. JWT is a stateless type of authentication. It means that the server doesn't store any session information in the database. It doesn't need to keep a record of which user has logged in or which token is issued for which user. Instead, the client will send subsequent requests to the server with a header in the format of `bearer-{JWT-token}`, or more often, the client will send it in the body of a POST request or as a URL parameter.


Token-based Authentication
Server
Browser
Send Credentials
Issue Token and sign, and send back to client
Send JWT token back
Send token as bearer-{JWT} to server on subsequent call for validation
Validate token that it is the subsequent users and validate
Retrurn request that the client wants
Client sign-off token is deleted

Token-based Authentication works like this:

1. User logins with their credentials
2. Server verifies the user's credentials, creates a signed token, and sends the token back to the client.
3. The token is stored in either local-storage or session-storage on the client-side.
4. Subsequent requests to the server will include this token, usually embedded in the header in the format of bearer-{JWT-token}
5. Once the user logouts, the token is destroyed on the client-side; no interaction with the server is required because the server is stateless.

A JSON Web Token contains 3 parts:

1. Header
2. Payload (holds the user ID) + expiry date

3. Signature

Token-based authentication has gained a lot of popularity due to its widespread use in modern web applications, Single Page Applications (SPA), web APIs, and IoT. However, it can introduce new vulnerabilities to the application if developers didn't properly implement the authentication.

Cookie vs Token Based Authentication

Understanding how these authentication mechanisms work, helps you to develop user authentication the right way. You have more options to choose based on the type of application you are developing. Knowing the advantages and disadvantages of both authentication mechanisms can help you to wisely choose between the two, and to secure your application from malicious attacks.

Advantages of using Cookie-based Authentication

HttpOnly and Secure Flag

Implementing or storing your token in cookies makes the application stateful. Having HttpOnly Flag will secure the cookie from malicious JavaScript attacks (i.e. XSS-attack). XSS-attack is an attack where an entity injects a malicious script to the victims' website. The actual attack occurs when the victim tries to visit webpages that executes the malicious code. It is often hidden in the form submission, or an iframe embedded image. Further, session cookies can be created with a secure flag that prevents token transmission through unencrypted channels. Therefore, always transmit data through HTTPS so that the attacker is not able to eavesdrop between the communication channel of the browser and server, then steal the cookie to impersonate the user.

Less Work for the Client, More Work to the Server

By creating cookies in the server and inserting 'set-cookie' in the response header, the browser will automatically send the authentication information on every subsequent request to the server. Therefore, the client doesn't need to manually manage cookie states between pages.

Disadvantages of Using Cookie-based authentication

CSRF or XSRF Attack

CSRF attack is when the entity runs a malicious JavaScript script, targeting website without the knowledge of browser use. It is more of a target centric attack where the intruder wants to know what the user wants to perform. CSRF attack is not easily understood because you will need to know where the source attacker is coming from. In order to protect against CSRF, you can implement a synchronizer token. This link is from StackOverflow explaining how and what it is in more detail, if you want to learn more about synchronizer tokens. With synchronizer tokens, you will need to implement another JavaScript logic for the synchronized token in the UI. Therefore, it can add another layer of complexity in your system.

Performance and Scalability

Database storage is needed to store cookies associate with the user. It adds another complexity to state management and it's harder to maintain. Each time a user sends subsequent requests on each page to the server, you will need to read from the database to validate if the user is authorized. It creates a lot of turnover, each time a different user logs in, their credentials need to be stored in the database, every subsequent request requires validation and lookups from the database, and the cookies need to be deleted from the database when the user logs out.

Furthermore, modern applications are not only built in the web, but are also native. Implementing cookies for mobile applications is much more challenging and unreliable because it needs to operate in various native environments. Cookies on the web was created inside hyper-text language, or http, which creates a uniform environment for all browsers. Since most users will repeatedly use the same browser and computer, session cookies are able to track users' activity. Additionally, for mobile, not only will applications run on different operating systems, each native application also has its own rule. Browsers play in different "sandboxes" from the native applications when used on the same devices, which makes it very hard to implement cookies for native applications.

Advantages of using Token-based authentication

Stateless

The main benefit of being stateless is less complexity and logic in the application. Thus, it will be easier to manage, scale, decouple, and it's less prone to error. Each token is self-contained and doesn't require backend logic to verify each request. The backend only needs to sign the token in order to validate. Third party services, like OAuth, can handle the issuing of tokens, leaving the backend a single function of checking the validity of the token.

Performance

Instead of issuing another database for the backend to read, write and delete tokens, JWT tokens can decode the session token to validate the user's authenticity and authority. For instance, if you have an API of `/api/` users to create a user in the application, only users that have Admin permissions are able to do the operation. In the traditional cookie-based token design, you will need to perform a look-up to the database, then validate if the session is expired. Then you'll have to perform another look-up to check if the user has the right permissions. Lastly, submit a Post request to create the user. On the other hand with JWT, you are able to store the user's role permission in JWT on the browser, decode the JWT token to validate the user's permission, and submit a Post request to create users, all from the browser.

CSRF safe

Unlike cookie-based authentication, token-based authentication is not suspectable to CSRF attacks, since attacker site will first need to steal the token before it can make the AJAX call to the legitimate website.

Disadvantages of using Token-based authentication

XSS attack

Applications that implement token-based authentication will need to be aware of Cross-Site Scripting Attacks. Cross-Site Scripting Attacks occur when an intruder is able to execute malicious JavaScript from inside your application. This attack often occurs when form inputs are not sanitized, and data is properly validated before form submission. Therefore, JWT token is actually vulnerable. However, modern web framework has built-in functionalities to sanitize input to prevent arbitrary code during form submission. For instance, the React form input is automatically sanitized by default, unless you update the input setting to `dangerouslySetInnerHTML`. Other ways to prevent XSS attack while using token-based authentication is to set the token expiry time limit to one hour, so that even if the token is stolen, it will quickly become unusable.

Design Consideration - Where to store the token?

Common place to store JWT token is in local storage, and it works well for most cases and it is recommended. However, there are some issue with storing JWT token in local storage that you need to be aware of, such as Cross Site Scripting attacks. I will talk more about how to solve it later in the disadvantage section of token-based authentication below. Another place to store the token is in the cookie. However, since JWT token can be large in size, you need to be aware of the maximum size of storing it in the cookie (elaborate). Lastly, storing token in session storage acts the same as local storage. However, the token will be deleted once user close the browser.

Size of JWT

The size of JWT can be relatively large compared to a session cookie, since most cookies are smaller than the average size of a JWT. JWT tokens are at most 8KB in size, which is much larger than cookies that are 4KB. Depending on the way data is transfer, JWT token size might be problematic if you add many claims to it.

Which one should I use?

Use case for Cookie

If you have a full controlled on the server-side and it is a browser application, having the server to control the state of the application will ease the client side, and create a more robust application. Cookie-based authentication is great for browser based and it has been extensively studied, which create a much safer option. It can restrict or limit session to certain operation or certain time period and it can invalidate user if there are any security concerns.

Use case for Token

If you want to have flexibility in dealing with various domain and environment, Token based authentication token is recommended. You don’t need to deal with two different authentication schemes to support browser traffic.

Use case for Hybrid

Production system usually use both mechanisms to create flexibility and robust application. For instance, OAuth uses specific bearer-token and longer-lived refresh token to get bearer token.

Wrap Up

There isn't a one-method-fits-all best approach for authentication, but there is a preferred way depending on your application's use case. Every authentication mechanism has its own advantages and disadvantages. ***There is tradeoff between having a more robust cookie-based authentication, but not as performant, or a token-based authentication in the client side, which is more performant but not as robust.*** Depending on the scenario, using a hybrid of both can lead to a more secure system, than using one alone.

It all boils down to use-cases, analyzing trade-offs, and preference. However, in order to create a more secure application, you will need to understand how both authentication works, and its trade-offs so you are able to implement your authentication the right way.

Like this Article?

Sign up for my newsletter to get this content weekly!

Related Posts

LET ME TEACH YOU HOW TO IMPLEMENT MONAD WITH CATS

Create your own Custom Monad with Cats Library

HOW TO RANDOMLY SPLIT UP AN ELEMENTS FROM A STREAM OF DATA IN PERCENTAGE

How to randomly split up element from a stream of data without knowing the total amount of it?

DEMYSTIFY BUILDER PATTERN IN SCALA

It is surprisingly simple to create a Builder Pattern in Scala

WHAT IS REALLY SO SPECIAL ABOUT JAVASCRIPT CLOSURE?

It is easy to explain what it is but hard to understand why you need to use them.

PURE FUNCTION VS REFERENTIAL TRANSPARENCY

Referential Transparency might not equal to Pure Function

0 Comments Edward Huang Login

Recommend Tweet Share Sort by Best

Start the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

Be the first to comment.

Subscribe Add Disqus to your siteAdd DisqusAdd