**Summary – Spring security**

Definition:

Spring security is an application framework that lets you do application level security. Which means it handles: Login and logout functionality, Allow/ block access to urls to logged-in users, Allow/ block access to urls to logged-in users with certain roles. Some of its functionalities are:

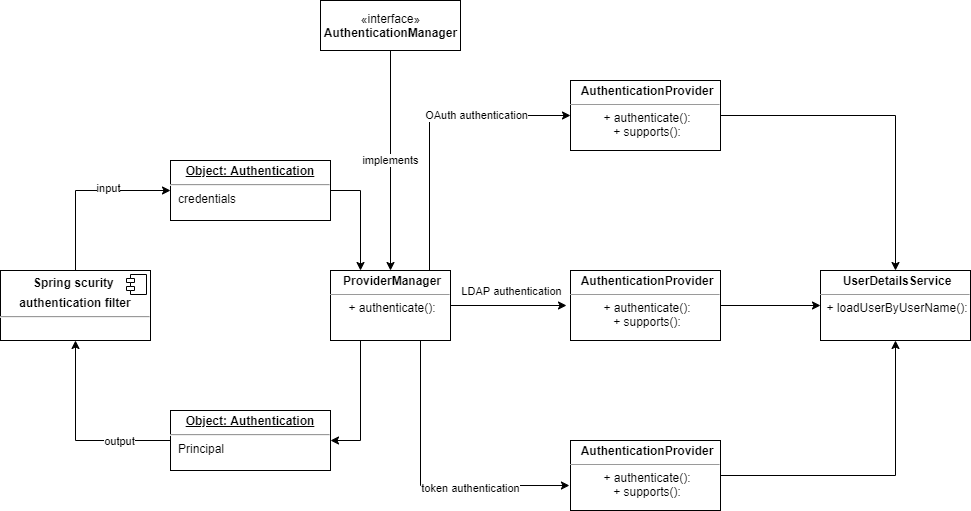
* Username and password authentication
* Adds login form
* Handles login error
* Creates a user and sets a default password
* SSO/ Okta/ LDAP
* App level authorization
* Intra app authorization like OAuth
* MS security (using token, JWT)
* Method level security

Some concepts:

* Authentication – answers the question: “who are you?” which is basically providing username and password
* Authorization – answers the question: “can the user perform the action it wants to do?” which is basically permits/ block actions based on user.
* Principal – the currently logged-in user. Once authenticating, the system returns the principal and that’s the reason you don’t need to provide username and password each time you are performing an action in the system.
* Granted authority – is the permission that the system grants the user for a specific action
* Role – group of granted authorities

Technological implementation of Spring security

Spring security is using some Filters that acts as interceptors between the user and the servlets. The authentication filter works in the following way:

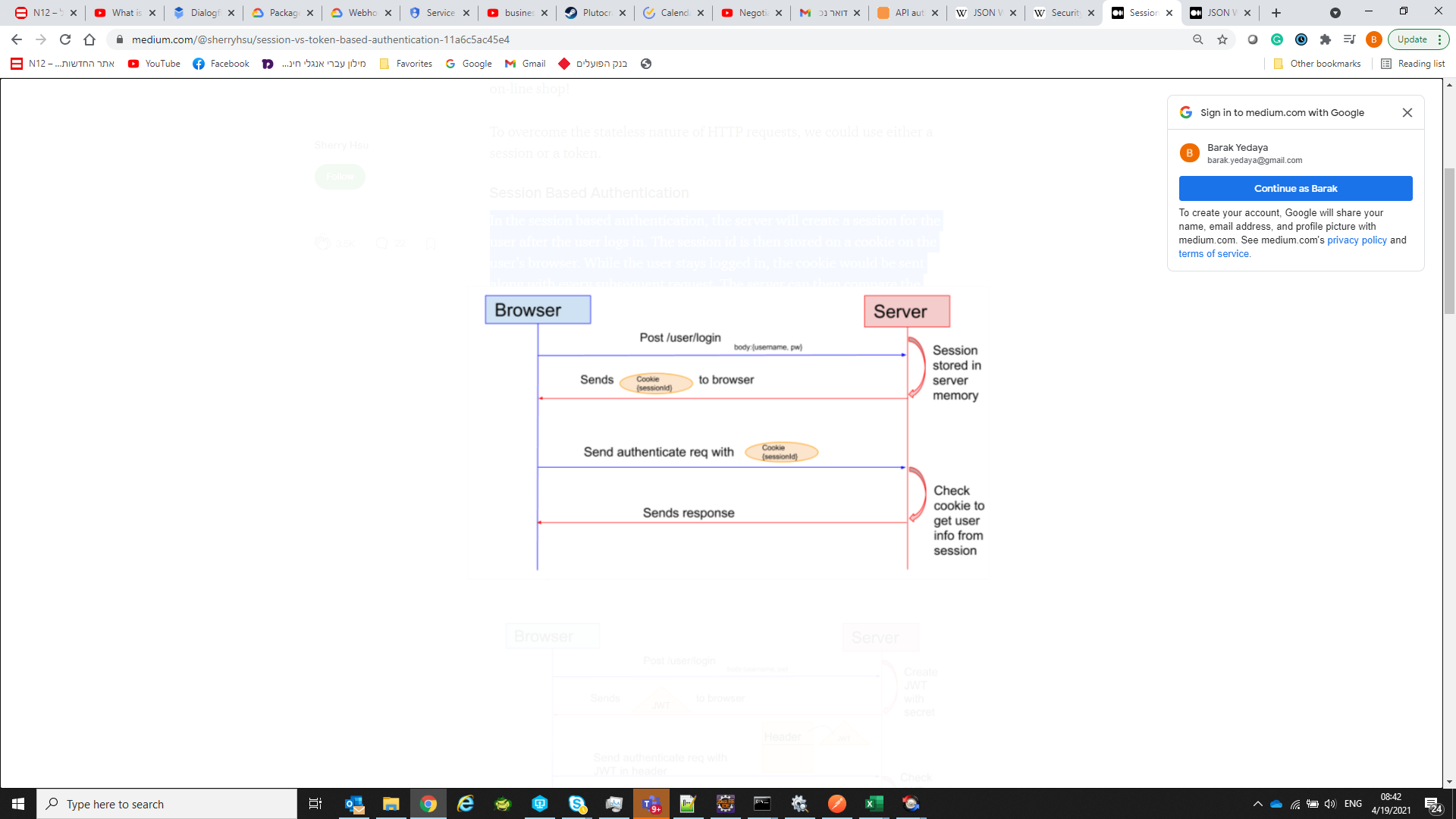


Types of authentications

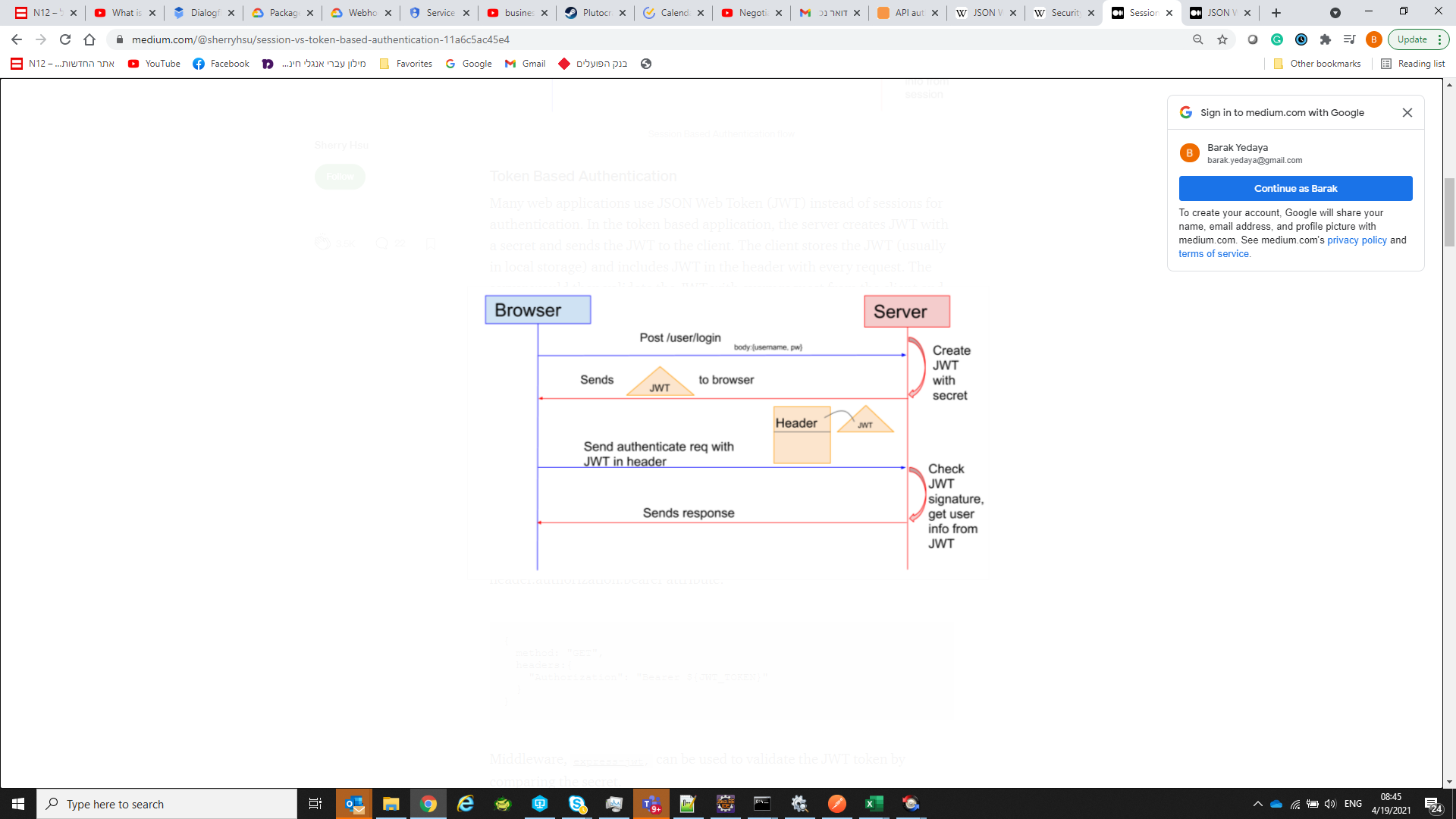
1. **Single-Factor/Primary Authentication** – Historically the most common form of authentication, Single-Factor Authentication, is also the least secure, as it only requires one factor to gain full system access. It could be a username and password, pin-number, or another simple code. While user-friendly, Single-Factor authenticated systems are relatively easy to infiltrate by phishing, key logging, or mere guessing. As there is no other authentication gate to get through, this approach is highly vulnerable to attack.
2. **Two-Factor Authentication (2FA)** – By adding a second factor for verification, two-factor authentication reinforces security efforts. It is an added layer that essentially double-checks that a user is, in reality, the user they’re attempting to log in as—making it much harder to break. With this method, users enter their primary authentication credentials (like the username/password mentioned above) and then must input a secondary piece of identifying information. The secondary factor is usually more difficult, as it often requires something the valid user would have access to, unrelated to the given system. Possible secondary factors are a one-time password from an authenticator app, a phone number, or device that can receive a push notification or SMS code, or a biometric like fingerprint (Touch ID) or facial (Face ID) or voice recognition. 2FA significantly minimizes the risk of system or resource compromise, as it’s unlikely an invalid user would know or have access to both authentication factors. While two-factor authentication is now more widely adopted for this reason, it does cause some user inconvenience, which is still something to consider in implementation.
3. **Multi-Factor Authentication (MFA)** – Multi-factor authentication is a high-assurance method, as it uses more system-irrelevant factors to legitimize users. Like 2FA, MFA uses factors like biometrics, device-based confirmation, additional passwords, and even location or behavior-based information (e.g., keystroke pattern or typing speed) to confirm user identity. However, the difference is that while 2FA always utilizes only two factors, MFA could use two or three, with the ability to vary between sessions, adding an elusive element for invalid users.
4. **SSO** – With SSO, users only have to log in to one application and, in doing so, gain access to many other applications. This method is more convenient for users, as it removes the obligation to retain multiple sets of credentials and creates a more seamless experience during operative sessions. Organizations can accomplish this by identifying a central domain (most ideally, an IAM system) and then creating secure SSO links between resources. This process allows domain-monitored user authentication and, with single sign-off, can ensure that when valid users end their session, they successfully log out of all linked resources and applications.

Types of technical post-authentication

1. **Session based authentication** – In the session based authentication, the server will create a session for the user after the user logs in. The session id is then stored on a cookie on the user’s browser. While the user stays logged in, the cookie would be sent along with every subsequent request. The server can then compare the session id stored on the cookie against the session information stored in the memory to verify user’s identity and sends response with the corresponding state!



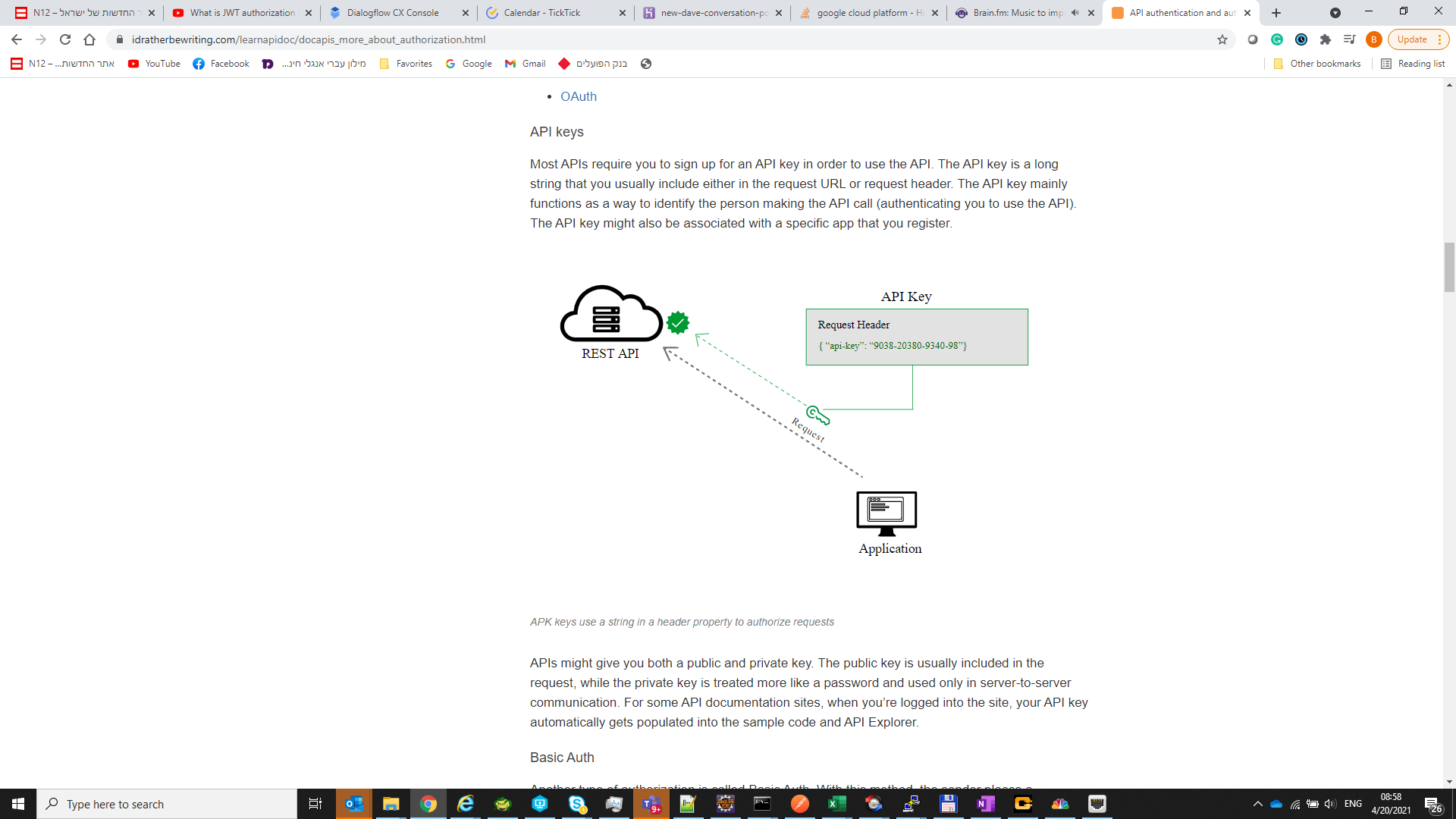
1. **Token based authentication** – In the token-based application, the server creates JWT with a secret and sends the JWT to the client. The client stores the JWT (usually in local storage) and includes JWT in the header with every request. The server would then validate the JWT with every request from the client and sends response.



Types of authorizations

1. **API keys** – Most APIs require you to sign up for an API key to use the API. The API key is a long string that you usually include either in the request URL or request header. The API key mainly functions as a way to identify the person making the API call (authenticating you to use the API). The API key might also be associated with a specific app that you register.

APIs might give you both a public and private key. The public key is usually included in the request, while the private key is treated more like a password and used only in server-to-server communication. For some API documentation sites, when you’re logged into the site, your API key automatically gets populated into the sample code and API Explorer.



1. **Basic Auth** –With this method, the sender places a “username:password” into the request header. The username and password are encoded with Base64, which is an encoding technique that converts the username and password into a set of 64 characters to ensure safe transmission.

Here’s an example of a Basic Auth in a request header:

Authorization: Basic bG9sOnNlY3VyZQ==

APIs that use Basic Auth will also use HTTPS, which means the message content will be encrypted within the HTTP transport protocol. (Without HTTPS, it would be easy for people to decode the username and password.)

When the API server receives the message, it decrypts the message and examines the header. After decoding the string and analyzing the username and password, it then decides whether to accept or reject the request.

In Postman, you can configure Basic Authorization by clicking the Authorization tab, selecting Basic Auth from the drop-down selector, and then typing the username and password on the right of the colon on each row. Postman handles the Base64 encoding for you automatically. when you enter a username and password with Basic Auth selected.

1. **HMAC (Hash-based message authorization code)** – HMAC is a stronger type of authentication, more common in financial APIs. With HMAC, both the sender and receiver know a secret key that no one else does. The sender creates a message based on some system properties (for example, the request timestamp plus account ID).

The message is then encoded by the secret key and passed through a secure hashing algorithm (SHA). The resulting value, referred to as a signature, is placed in the request header.

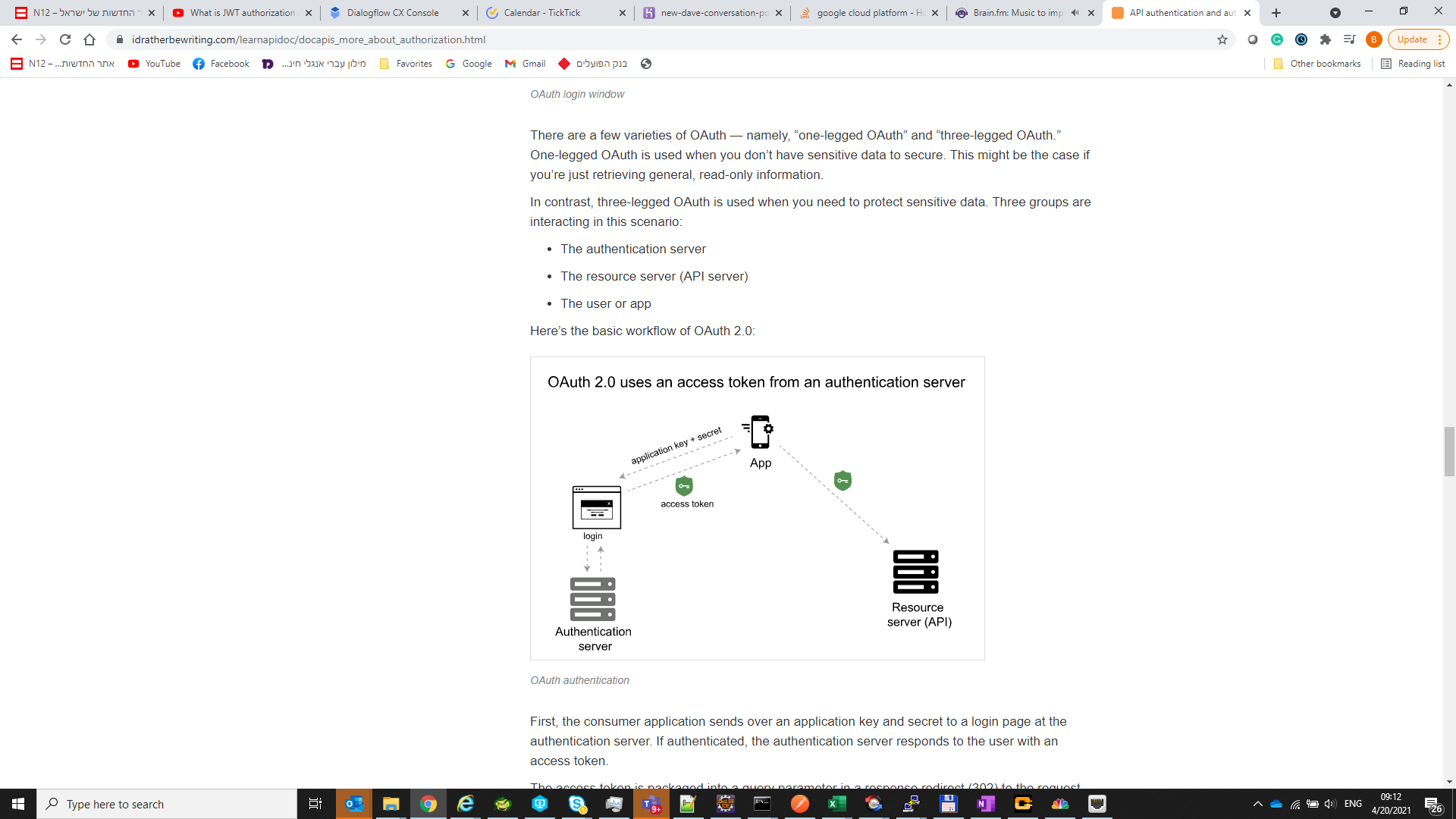
When the receiver (the API server) receives the request, it takes the same system properties (the request timestamp plus account ID) and uses the secret key (which only the requester and API server know) and SHA to generate the same string. If the string matches the signature in the request header, it accepts the request. If the strings don’t match, then the request is rejected. The important point is that the secret key (critical to reconstructing the hash) is known only to the sender and receiver. The secret key is not included in the request. HMAC security is used when you want to ensure the request is both authentic and hasn’t been tampered with.

1. **OAuth 2.0** – One popular method for authenticating and authorizing users is OAuth 2.0. Authorizing approach handles delegated authorization, meaning, when the user is delegating a certain service, authorizations for some of his data.

For example, let’s say that a user is entering a printing pictures site. The user must upload pictures to the site and the site prints them. Now, let’s assume that the user’s pictures are stored at Google cloud. The user doesn’t want to download the pictures to his PC and upload it to the printing site. So, instead, it delegates the printing site authority to login to his Google cloud pictures to take the pictures from there and upload it to the site.

There are 3 Three groups that are interacting using this mechanism:

* The authentication server – in this case Google authenticated server
* The resource server (API server) – in this case Google cloud server
* The user or app – in this case the printing site server



First, the consumer application sends over an application key and secret to a login page at the authentication server. If authenticated, the authentication server responds to the user with an access token. The access token is packaged into a query parameter in a response redirect (302) to the request. The redirect points the user’s request back to the resource server (the API server).

The user then makes a request to the resource server (API server). The access token gets added to the header of the API request with the word “Bearer” followed by the token string. The API server checks the access token in the user’s request and decides whether to authenticate the user.

Access tokens not only provide authentication for the requester but also define the permissions of how the user can use the API. Additionally, access tokens usually expire after a period of time and require the user to log in again.

Authenticating approach means that the client is using Google/ Facebook/ Others to do the authentication of the user for him because it trusts their authentication process. You often see OAuth 2.0 when you’re using a site and are prompted to log in using a service like Twitter, Google, or Facebook.

Types of exchanging secure data

1. **JWT** – JSON Web Token (JWT) is an open standard (RFC 7519) for securely transmitting information between endpoints as JSON objects. It is mainly used to prove that the sent data was created by an authentic source. JWT consists of three concatenated Base64url-encoded strings, separated by dots (.), which are:
   1. Header
   2. Payload
   3. Signature

A JWT typically looks like: xxxxx.yyyyy.zzzzz

Header

The first part typically consists of two parts; the type of the token, which is JWT, and the signing algorithm being used such as HMAC SHA256 or RSA. If you don’t define the algorithm, it uses HS256 by default. For example:

{  
“alg”: “HS256”,  
“typ”: “JWT”  
}

Payload

The second part consists of a set of claims that are basically verifiable security statements, such as the identity of the user and the permissions they are allowed.

There are three types of claims: registered, public, and private claims. Note that the claim names are short as JWT is meant to be compact for fast requests.

And wait! Be careful not to put sensitive data such as passwords in your payload as this can easily be decoded.

An example payload could be:

{  
“sub”: “123456789”,  
“name”: “Anamika Ahmed”,  
“admin”: true  
}

Signature

The last part is the signature which is the sum of the encoded header, the encoded payload, a secret, and lastly, the algorithm which is specified in the header.

For example, if you want to use the HS256 algorithm, the signature would be created in the following way:

HS256(  
base64UrlEncode(header) + “.” +  
base64UrlEncode(payload),  
secret)

The signature is used to verify the message wasn’t changed along the way. It is the most important part of the JWT structure as header and payload can easily be decoded, but not the signature.

The signature is not publicly readable because it is encrypted with a secret key. Unless someone has the secret key, they cannot decrypt this information.

1. **SAML** - Security Assertion Markup Language (SAML) is an open standard that allows identity providers (IdP) to pass authorization credentials to service providers (SP). That means that you can use one set of credentials to log into many different websites. It’s much simpler to manage one login per user than it is to manage separate logins to email, customer relationship management (CRM) software, Active Directory, etc.

SAML transactions use XML for standardized communications between the identity provider and service providers. SAML is the link between the authentication of a user’s identity and the authorization to use a service.

SAML simplifies federated authentication and authorization processes for users, Identity providers, and service providers. SAML provides a solution to allow your identity provider and service providers to exist separately from each other, which centralizes user management and provides access to SaaS solutions.

SAML implements a secure method of passing user authentications and authorizations between the identity provider and service providers. When a user logs into a SAML enabled application, the service provider requests authorization from the appropriate identity provider. The identity provider authenticates the user’s credentials and then returns the authorization for the user to the service provider, and the user is now able to use the application.

SAML authentication is the process of verifying the user’s identity and credentials (password, two-factor authentication, etc.). SAML authorization tells the service provider what access to grant the authenticated user.

A SAML provider is a system that helps a user access a service they need. There are two primary types of SAML providers, service provider, and identity provider.

A service provider needs the authentication from the identity provider to grant authorization to the user.

An identity provider performs the authentication that the end user is who they say they are and sends that data to the service provider along with the user’s access rights for the service.

Microsoft Active Directory or Azure are common identity providers. Salesforce and other CRM solutions are usually service providers, in that they depend on an identity provider for user authentication.

A SAML Assertion is the XML document that the identity provider sends to the service provider that contains the user authorization. There are three different types of SAML Assertions – authentication, attribute XViewAttachmentRes and authorization decision.

Authentication assertions prove identification of the user and provide the time the user logged in and what method of authentication they used (I.e., Kerberos, 2 factor, etc.)

The attribution assertion passes the SAML attributes to the service provider – SAML attributes are specific pieces of data that provide information about the user.

An authorization decision assertion says if the user is authorized to use the service or if the identify provider denied their request due to a password failure or lack of rights to the service.

