**Q. How does HTTPS work?**

HTTPS uses an encryption protocol to encrypt communications. The protocol is called Transport Layer Security (TLS), although formerly it was known as Secure Sockets Layer (SSL). This protocol secures communications by using what’s known as an asymmetric public key infrastructure. This type of security system uses two different keys to encrypt communications between two parties:

The private key - this key is controlled by the owner of a website and it’s kept, as the reader may have speculated, private. This key lives on a web server and is used to decrypt information encrypted by the public key.

The public key - this key is available to everyone who wants to interact with the server in a way that’s secure. Information that’s encrypted by the public key can only be decrypted by the private key.

The SSL layer has 2 main purposes:

Verifying that you are talking directly to the server that you think you are talking to

Ensuring that only the server can read what you send it and only you can read what it sends back

The really, really clever part is that anyone can intercept every single one of the messages you exchange with a server, including the ones where you are agreeing on the key and encryption strategy to use, and still not be able to read any of the actual data you send.

2. How an SSL connection is established

An SSL connection between a client and server is set up by a handshake, the goals of which are:

To satisfy the client that it is talking to the right server (and optionally visa versa)

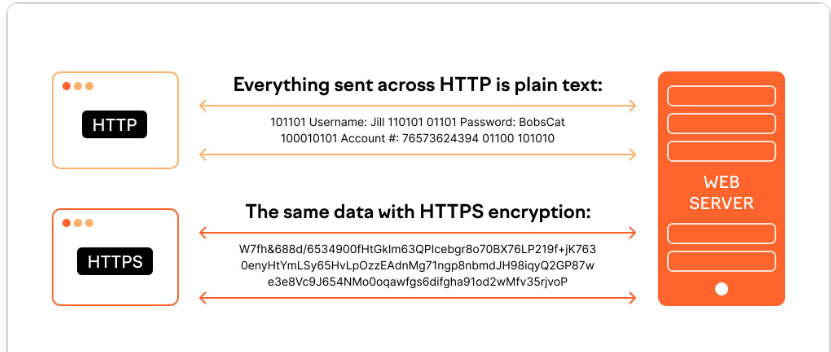
For the parties to have agreed on a “cipher suite”, which includes which encryption algorithm they will use to exchange data

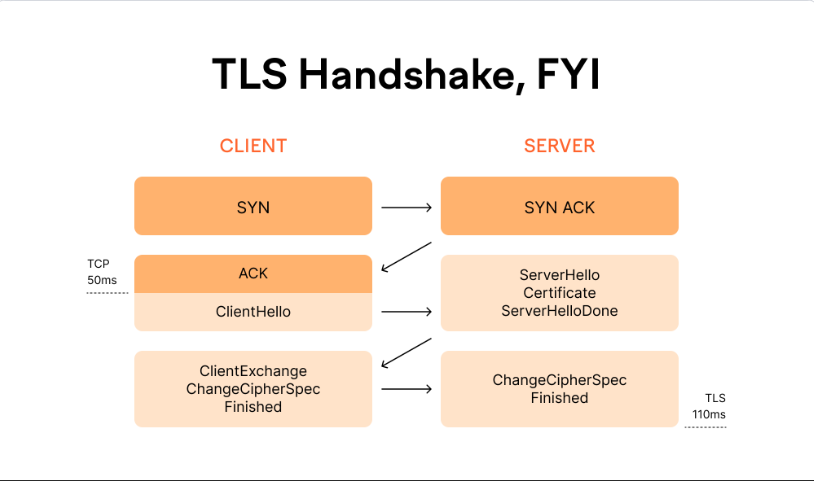
For the parties to have agreed on any necessary keys for this algorithm

Once the connection is established, both parties can use the agreed algorithm and keys to securely send messages to each other. We will break the handshake up into 3 main phases - Hello, Certificate Exchange and Key Exchange.

Hello - The handshake begins with the client sending a ClientHello message. This contains all the information the server needs in order to connect to the client via SSL, including the various cipher suites and maximum SSL version that it supports. The server responds with a ServerHello, which contains similar information required by the client, including a decision based on the client’s preferences about which cipher suite and version of SSL will be used.

Certificate Exchange - Now that contact has been established, the server has to prove its identity to the client. This is achieved using its SSL certificate, which is a very tiny bit like its passport. An SSL certificate contains various pieces of data, including the name of the owner, the property (eg. domain) it is attached to, the certificate’s public key, the digital signature and information about the certificate’s validity dates. The client checks that it either implicitly trusts the certificate, or that it is verified and trusted by one of several Certificate Authorities (CAs) that it also implicitly trusts. Much more about this shortly. Note that the server is also allowed to require a certificate to prove the client’s identity, but this typically only happens in very sensitive applications.

Key Exchange - The encryption of the actual message data exchanged by the client and server will be done using a symmetric algorithm, the exact nature of which was already agreed during the Hello phase. A symmetric algorithm uses a single key for both encryption and decryption, in contrast to asymmetric algorithms that require a public/private key pair. Both parties need to agree on this single, symmetric key, a process that is accomplished securely using asymmetric encryption and the server’s public/private keys. 



**Q.** **What are different http methods available and what they exactly do.**

**Available HTTP Methods**

The primary or most commonly-used HTTP methods are POST, GET, PUT, PATCH, and DELETE. These methods correspond to create, read, update, and delete (or CRUD) operations, respectively. There are a number of other methods, too, but they are utilized less frequently.

Below is a table summarizing HTTP methods available in Oro API and their return values in combination with the resource URIs:

| **HTTP Method** | **CRUD operation** | **Entire Collection (e.g. /users)** | **Specific Item (e.g. /users/{id})** |
| --- | --- | --- | --- |
| GET | Read | 200 (OK), list of entities. Use pagination, sorting and filtering to navigate big lists. | 200 (OK), single entity.  404 (Not Found), if ID not found or invalid. |
| POST | Create | 201 (Created), Response contains response similar to **GET** /user/{id} containing new ID. | **not applicable** |
| PATCH | Update | [Batch API](https://doc.oroinc.com/api/batch-api/#web-services-api-batch-api) | 200 (OK) or 204 (No Content).  404 (Not Found), if ID not found or invalid. |
| DELETE | Delete | 204 (No Content).  400(Bad Request) if no filter is specified. | 204 (No Content).  404 (Not Found), if ID not found or invalid. |
| PUT | Update/Replace | **not implemented** | **not implemented** |

**Q. Understand and explain the use of various http response codes.**

**HTTP response status codes**

HTTP response status codes indicate whether a specific HTTP request has been successfully completed. Responses are grouped in five classes:

Informational responses (100 – 199)

Successful responses (200 – 299)

Redirection messages (300 – 399)

Client error responses (400 – 499)

Server error responses (500 – 599)

Information responses

100 Continue

This interim response indicates that the client should continue the request or ignore the response if the request is already finished.

101 Switching Protocols

This code is sent in response to an Upgrade request header from the client and indicates the protocol the server is switching to.

102 Processing (WebDAV)

This code indicates that the server has received and is processing the request, but no response is available yet.

103 Early Hints Experimental

This status code is primarily intended to be used with the Link header, letting the user agent start preloading resources while the server prepares a response.

Successful responses

200 OK

The request succeeded. The result meaning of "success" depends on the HTTP method:

GET: The resource has been fetched and transmitted in the message body.

HEAD: The representation headers are included in the response without any message body.

PUT or POST: The resource describing the result of the action is transmitted in the message body.

TRACE: The message body contains the request message as received by the server.

201 Created

The request succeeded, and a new resource was created as a result. This is typically the response sent after POST requests, or some PUT requests.

202 Accepted

The request has been received but not yet acted upon. It is noncommittal, since there is no way in HTTP to later send an asynchronous response indicating the outcome of the request. It is intended for cases where another process or server handles the request, or for batch processing.

203 Non-Authoritative Information

This response code means the returned metadata is not exactly the same as is available from the origin server, but is collected from a local or a third-party copy. This is mostly used for mirrors or backups of another resource. Except for that specific case, the 200 OK response is preferred to this status.

204 No Content

There is no content to send for this request, but the headers may be useful. The user agent may update its cached headers for this resource with the new ones.

205 Reset Content

Tells the user agent to reset the document which sent this request.

206 Partial Content

This response code is used when the Range header is sent from the client to request only part of a resource.

207 Multi-Status (WebDAV)

Conveys information about multiple resources, for situations where multiple status codes might be appropriate.

208 Already Reported (WebDAV)

Used inside a <dav:propstat> response element to avoid repeatedly enumerating the internal members of multiple bindings to the same collection.

226 IM Used (HTTP Delta encoding)

The server has fulfilled a GET request for the resource, and the response is a representation of the result of one or more instance-manipulations applied to the current instance.

Redirection messages

300 Multiple Choices

The request has more than one possible response. The user agent or user should choose one of them. (There is no standardized way of choosing one of the responses, but HTML links to the possibilities are recommended so the user can pick.)

301 Moved Permanently

The URL of the requested resource has been changed permanently. The new URL is given in the response.

302 Found

This response code means that the URI of requested resource has been changed temporarily. Further changes in the URI might be made in the future. Therefore, this same URI should be used by the client in future requests.

303 See Other

The server sent this response to direct the client to get the requested resource at another URI with a GET request.

304 Not Modified

This is used for caching purposes. It tells the client that the response has not been modified, so the client can continue to use the same cached version of the response.

305 Use Proxy Deprecated

Defined in a previous version of the HTTP specification to indicate that a requested response must be accessed by a proxy. It has been deprecated due to security concerns regarding in-band configuration of a proxy.

306 unused

This response code is no longer used; it is just reserved. It was used in a previous version of the HTTP/1.1 specification.

307 Temporary Redirect

The server sends this response to direct the client to get the requested resource at another URI with the same method that was used in the prior request. This has the same semantics as the 302 Found HTTP response code, with the exception that the user agent must not change the HTTP method used: if a POST was used in the first request, a POST must be used in the second request.

308 Permanent Redirect

This means that the resource is now permanently located at another URI, specified by the Location: HTTP Response header. This has the same semantics as the 301 Moved Permanently HTTP response code, with the exception that the user agent must not change the HTTP method used: if a POST was used in the first request, a POST must be used in the second request.

Client error responses

400 Bad Request

The server cannot or will not process the request due to something that is perceived to be a client error (e.g., malformed request syntax, invalid request message framing, or deceptive request routing).

401 Unauthorized

Although the HTTP standard specifies "unauthorized", semantically this response means "unauthenticated". That is, the client must authenticate itself to get the requested response.

402 Payment Required Experimental

This response code is reserved for future use. The initial aim for creating this code was using it for digital payment systems, however this status code is used very rarely and no standard convention exists.

403 Forbidden

The client does not have access rights to the content; that is, it is unauthorized, so the server is refusing to give the requested resource. Unlike 401 Unauthorized, the client's identity is known to the server.

404 Not Found

The server cannot find the requested resource. In the browser, this means the URL is not recognized. In an API, this can also mean that the endpoint is valid but the resource itself does not exist. Servers may also send this response instead of 403 Forbidden to hide the existence of a resource from an unauthorized client. This response code is probably the most well known due to its frequent occurrence on the web.

405 Method Not Allowed

The request method is known by the server but is not supported by the target resource. For example, an API may not allow calling DELETE to remove a resource.

406 Not Acceptable

This response is sent when the web server, after performing server-driven content negotiation, doesn't find any content that conforms to the criteria given by the user agent.

407 Proxy Authentication Required

This is similar to 401 Unauthorized but authentication is needed to be done by a proxy.

408 Request Timeout

This response is sent on an idle connection by some servers, even without any previous request by the client. It means that the server would like to shut down this unused connection. This response is used much more since some browsers, like Chrome, Firefox 27+, or IE9, use HTTP pre-connection mechanisms to speed up surfing. Also note that some servers merely shut down the connection without sending this message.

409 Conflict

This response is sent when a request conflicts with the current state of the server.

410 Gone

This response is sent when the requested content has been permanently deleted from server, with no forwarding address. Clients are expected to remove their caches and links to the resource. The HTTP specification intends this status code to be used for "limited-time, promotional services". APIs should not feel compelled to indicate resources that have been deleted with this status code.

411 Length Required

Server rejected the request because the Content-Length header field is not defined and the server requires it.

412 Precondition Failed

The client has indicated preconditions in its headers which the server does not meet.

413 Payload Too Large

Request entity is larger than limits defined by server. The server might close the connection or return an Retry-After header field.

414 URI Too Long

The URI requested by the client is longer than the server is willing to interpret.

415 Unsupported Media Type

The media format of the requested data is not supported by the server, so the server is rejecting the request.

416 Range Not Satisfiable

The range specified by the Range header field in the request cannot be fulfilled. It's possible that the range is outside the size of the target URI's data.

417 Expectation Failed

This response code means the expectation indicated by the Expect request header field cannot be met by the server.

418 I'm a teapot

The server refuses the attempt to brew coffee with a teapot.

421 Misdirected Request

The request was directed at a server that is not able to produce a response. This can be sent by a server that is not configured to produce responses for the combination of scheme and authority that are included in the request URI.

422 Unprocessable Content (WebDAV)

The request was well-formed but was unable to be followed due to semantic errors.

423 Locked (WebDAV)

The resource that is being accessed is locked.

424 Failed Dependency (WebDAV)

The request failed due to failure of a previous request.

425 Too Early Experimental

Indicates that the server is unwilling to risk processing a request that might be replayed.

426 Upgrade Required

The server refuses to perform the request using the current protocol but might be willing to do so after the client upgrades to a different protocol. The server sends an Upgrade header in a 426 response to indicate the required protocol(s).

428 Precondition Required

The origin server requires the request to be conditional. This response is intended to prevent the 'lost update' problem, where a client GETs a resource's state, modifies it and PUTs it back to the server, when meanwhile a third party has modified the state on the server, leading to a conflict.

429 Too Many Requests

The user has sent too many requests in a given amount of time ("rate limiting").

431 Request Header Fields Too Large

The server is unwilling to process the request because its header fields are too large. The request may be resubmitted after reducing the size of the request header fields.

451 Unavailable For Legal Reasons

The user agent requested a resource that cannot legally be provided, such as a web page censored by a government.

Server error responses

500 Internal Server Error

The server has encountered a situation it does not know how to handle.

501 Not Implemented

The request method is not supported by the server and cannot be handled. The only methods that servers are required to support (and therefore that must not return this code) are GET and HEAD.

502 Bad Gateway

This error response means that the server, while working as a gateway to get a response needed to handle the request, got an invalid response.

503 Service Unavailable

The server is not ready to handle the request. Common causes are a server that is down for maintenance or that is overloaded. Note that together with this response, a user-friendly page explaining the problem should be sent. This response should be used for temporary conditions and the Retry-After HTTP header should, if possible, contain the estimated time before the recovery of the service. The webmaster must also take care about the caching-related headers that are sent along with this response, as these temporary condition responses should usually not be cached.

504 Gateway Timeout

This error response is given when the server is acting as a gateway and cannot get a response in time.

505 HTTP Version Not Supported

The HTTP version used in the request is not supported by the server.

506 Variant Also Negotiates

The server has an internal configuration error: the chosen variant resource is configured to engage in transparent content negotiation itself, and is therefore not a proper end point in the negotiation process.

507 Insufficient Storage (WebDAV)

The method could not be performed on the resource because the server is unable to store the representation needed to successfully complete the request.

508 Loop Detected (WebDAV)

The server detected an infinite loop while processing the request.

510 Not Extended

Further extensions to the request are required for the server to fulfill it.

511 Network Authentication Required

Indicates that the client needs to authenticate to gain network access.

**Q. What are the different web communication protocols and their use cases?**

There are various types of protocols that support a major and compassionate role in communicating with different devices across the network. These are:

Transmission Control Protocol (TCP)

Internet Protocol (IP)

User Datagram Protocol (UDP)

Post office Protocol (POP)

Simple mail transport Protocol (SMTP)

File Transfer Protocol (FTP)

Hyper Text Transfer Protocol (HTTP)

Hyper Text Transfer Protocol Secure (HTTPS)

Telnet

Gopher

Let's discuss each of them briefly:

Transmission Control Protocol (TCP): TCP is a popular communication protocol which is used for communicating over a network. It divides any message into series of packets that are sent from source to destination and there it gets reassembled at the destination.

Internet Protocol (IP): IP is designed explicitly as addressing protocol. It is mostly used with TCP. The IP addresses in packets help in routing them through different nodes in a network until it reaches the destination system. TCP/IP is the most popular protocol connecting the networks.

User Datagram Protocol (UDP): UDP is a substitute communication protocol to Transmission Control Protocol implemented primarily for creating loss-tolerating and low-latency linking between different applications.

Post office Protocol (POP): POP3 is designed for receiving incoming E-mails.

Simple mail transport Protocol (SMTP): SMTP is designed to send and distribute outgoing E-Mail.

File Transfer Protocol (FTP): FTP allows users to transfer files from one machine to another. Types of files may include program files, multimedia files, text files, and documents, etc.

Hyper Text Transfer Protocol (HTTP): HTTP is designed for transferring a hypertext among two or more systems. HTML tags are used for creating links. These links may be in any form like text or images. HTTP is designed on Client-server principles which allow a client system for establishing a connection with the server machine for making a request. The server acknowledges the request initiated by the client and responds accordingly.

Hyper Text Transfer Protocol Secure (HTTPS): HTTPS is abbreviated as Hyper Text Transfer Protocol Secure is a standard protocol to secure the communication among two computers one using the browser and other fetching data from web server. HTTP is used for transferring data between the client browser (request) and the web server (response) in the hypertext format, same in case of HTTPS except that the transferring of data is done in an encrypted format. So it can be said that https thwart hackers from interpretation or modification of data throughout the transfer of packets.

Telnet: Telnet is a set of rules designed for connecting one system with another. The connecting process here is termed as remote login. The system which requests for connection is the local computer, and the system which accepts the connection is the remote computer.

Gopher: Gopher is a collection of rules implemented for searching, retrieving as well as displaying documents from isolated sites. Gopher also works on the client/server principle.

**Q. Pros and cons of Single page and multi page applications.**

**Single Page Application**

A SPA application is a single page continuously interacting with the user by dynamically rewriting the current page rather than loading entire new pages from a server. Trello, Facebook, Gmail, and Twitter are a few single-page app examples.

**Single Page Application Pros**

Single-page applications are fast as most of the resources, including HTML, CSS, and Scripts, are loaded once, and only data is transmitted back and forth. Here are some of the business benefits of building single-page applications:

1. Quick Loading Time

A page taking over 200 milliseconds to load can significantly affect your online business and, eventually, sales.

With the SPA approach, your full page loads quicker than traditional web applications, as it only has to load a page at the first request. On the other hand, traditional web apps have to load pages at every request, taking more time.

2. Seamless User Experience

SPAs deliver an experience like a desktop or mobile app. Users do not have to watch a new page load, as only the content changes, not the page, making the experience enjoyable.

3. Ease in Building Feature-rich Apps

SPA application makes adding advanced features to a web application easy. For example, it is easier to build a content-editing web app with real-time analysis using SPA development. Doing this with a traditional web app requires a total page reload to perform content analysis.

4. Uses Less Bandwidth

It is no surprise that SPAs consume less bandwidth since they only load web pages once. Besides that, they can also do well in areas with a slow internet connection. Hence, it is convenient for everyone, regardless of internet speed.

**Single Page Application Cons**

Single page application architecture is best for developing high-performing SAAS platforms and social networks. However, this approach has some disadvantages that make it unsuitable for developing highly secure and SEO-optimized websites.

1. Doesn’t Perform Well With SEO

One of the metrics that search engines use is the number of pages a website has. However, since SPAs only load a single page, it is a disadvantage when ranking on search engines

2. Uses a Lot of Browser Resources

SPAs require many resources from the web browser since the browser does most of the tasks for the SPAs. Creating SPAs often need users to use the latest browsers with support for some modern features.

3. Security Issues

Compared to multi-page apps, SPAs are more prone to cross-site scripting attacks. Using XSS, it becomes easy for hackers to introduce client-side scripts into a web app. Also, SPAs are more likely to expose sensitive data to all users.

**Multi-Page Application**

A multi-page Application is a traditional web application where a new page is requested from the server to display when data is exchanged back and forth. The amount of content they carry is enormous. Thus, they are generally multi-level deep with many links and intricate UIs.

**Advantages of Multi-Page Applications**

Better SEO

SEO is better on MPAs since the architecture is native to search engine crawlers. Such apps provide better control over SEO thanks to multiple pages and changing content. Moreover, developers can add meta tags to every page. An MPA gives a better chance of ranking for different keywords since an app can be optimized for one keyword per page.

Unlimited scalability

MPAs allow you to create new content and place it on new pages. Multi-page apps can include as much information about products or services as required, with no page limitations. Single-page applications don’t allow a lot of features on one page, which can lead to longer loading times. Therefore, when company needs more features, they decide to use multi-page applications.

Better Insights from Analytics

MPAs can provide lots of analytics with valuable information on how a website is performing: which features are working and which aren’t. With a single-page app, the only useful information you can collect is who visitors are and for how long they stay on the site.

**Disadvantages of Multi-Page Applications**

Along with the advantages there are some considerable disadvantages of building a multi-page app. Let’s consider them.

Slower speed and overall performance

With multi-page apps, a server needs to reload most resources such as HTML, CSS, and scripts with every interaction. When loading another page, the browser completely reloads page data and downloads all resources again, even components that are repeated throughout all pages such as the header and footer. This negatively affects speed and performance.

More time for development

Compared to SPAs, MPAs take longer to develop. In most cases, developers have to code the backend from scratch. There are also difficulties in frontend and backend separation since they interact very closely with each other. Developers need to use frameworks for both the frontend or backend. This results in longer app development.

Harder maintenance and updates

Maintaining and updating multiple pages can be a pain, and things only get worse the bigger a website becomes. In addition, maintaining security may be problematic because developers need to secure each separate page. Single-page apps allow developers to just secure data endpoints.