Web app

Web app: a piece of s/w that can be accessed from browser. Multiple users, same version for all. No device storage is needed. Needs a **web server and db**.

Web server: network application that listens on a port for a request. (No request? Sits idle.) via some transport protocol (http, https) and send response with the required resources (html, text, pdf, img). Physical device with network connectivity. A web server can host multiple web app.

Web client: The client side (user side) of the Web. A Web client typically refers to the Web browser in the user's machine or mobile device. It may also refer to extensions and helper applications that enhance the browser to support special services from the site. Example: chrome, safari. Daraz server (not traditional)

User agent: something that can talk to a web server. a user agent is any software, acting on behalf of a user, which "retrieves, renders and facilitates end-user interaction with Web content."

Web client <--------a-------c------b----------→ web server

a. Request send from web client
b. Response send from server
c. Transport protocol (http, https,smtp)

Hhtp://	fb.com/	about	-> request
Protocol	web server Address	address of resources	

Port address: A port is identified for each transport protocol and address combination by a 16-bit unsigned number, known as the **port number**. A port number is always associated with an IP address of a host and the type of transport protocol used for communication. Specific port numbers are reserved to identify specific services so that an arriving packet can be easily forwarded to a running application.

Static vs dynamic web app: depends on which resources to return based on a particular request

Static website

The codes are fixed for each page so the information contained in the page does not change and it looks like a printed page. Its web pages are coded in HTML. Example: Github.io

Dynamic website

Dynamic website is a collection of dynamic web pages whose content changes dynamically. It accesses content from a database or Content Management System (CMS). Therefore, when you alter or update the content of the database, the content of the website is also altered or updated.

Static Website	Dynamic Website
Prebuilt content is same every time the page is loaded.	Content is generated quickly and changes regularly.
It uses the **HTML** code for developing a website.	It uses the server side languages such as **PHP,SERVLET, JSP, and ASP.NET** etc. for developing a website.
It sends exactly the same response for every request.	It may generate different HTML for each of the request.
The content is only changed when someone publishes and updates the file (sends it to the web server).	The page contains "server-side" code which allows the server to generate the unique content when the page is loaded.
Flexibility is the main advantage of static website.	Content Management System (CMS) is the main advantage of dynamic website.

HTTP

The Hypertext Transfer Protocol (HTTP) is an **application-layer protocol** for distributed, collaborative, hypermedia information systems. HTTP clients generally use **Transmission Control Protocol (TCP)** connections to communicate with servers

Stateless protocol: The server and client are aware of each other only during a current request. Afterwards, both of them forget about each other.

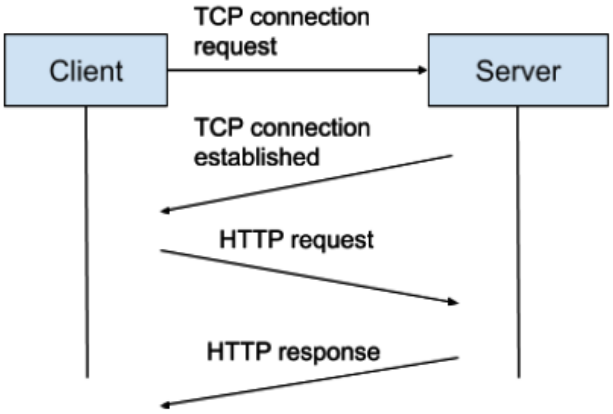
In HTTP protocol, neither the client nor the browser can retain information between different requests across the web pages.

Connectionless: The HTTP client, i.e., a browser initiates an HTTP request and after a request is made, the client waits for the response. The server processes the request and sends a response back after which client disconnect the connection. So client and server knows about each other during current request and response only. Further requests are made on new connection like client and server are new to each other.

Media independent: any type of data can be sent by HTTP as long as both the client and the server know how to handle the data content. It is required for the client as well as the server to specify the content type using appropriate MIME-type. Example: text/plain,/html,/csv; image/jpg,png; application/pdf,zip; video/mp4

Message format: request and response

How does req/res work



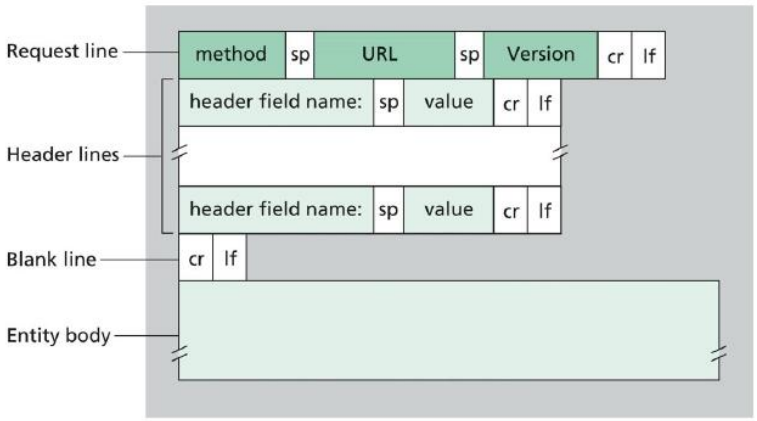
----------connection demolished----------

To request the server in HTTP protocol, first a TCP connection is sent from the client to the server. After the connection has been established, then

HTTP request is made from the client to the server. From the connection request to the connection establishment is
called one **round trip time**. Similarly, from the HTTP request to the HTTP response is **one round trip**. after that the connection is close/demolished. Usually server side close the connection by sending a connection flag. Else client side can manually close the connection.

http request:

format



Method: get,post,delete,put,head,connect,options,trace

Version: http 1.0-can send only 1 res/req between the duration of the one connection

http 1.1-can send only multiple res/req between the duration of the one connection

Blank line: if there's no body still we need it to **encode** the req/res

** there is no fixed size for http req. so need to follow the format strictly

Cr: carriage return(/r)

Lf: /n

User agent: Client information that is sending the req. example: mozila/4.0 windowsnt.

**server side a connection: keep alive rakhle browser er circle ghurtei thakbe.

Example

Get / http/1.1

Method/relative (base) url/http/1.1

Host: dev.fb.com

Header field name: value

Accept-language: fr(French) ; ---- res return er time if there is a French version of that doc, then it will return that.else it'll return the default lang

```
GET / HTTP/1.1
Host: developer.mozilla.org
Accept-Language: fr
```

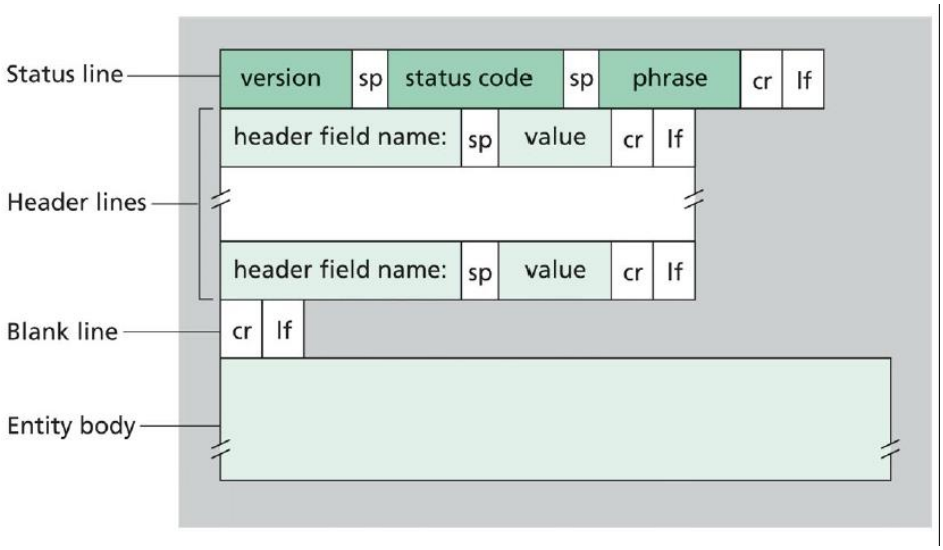
```
GET /hello.htm HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: www.tutorialspoint.com
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: Keep-Alive
```

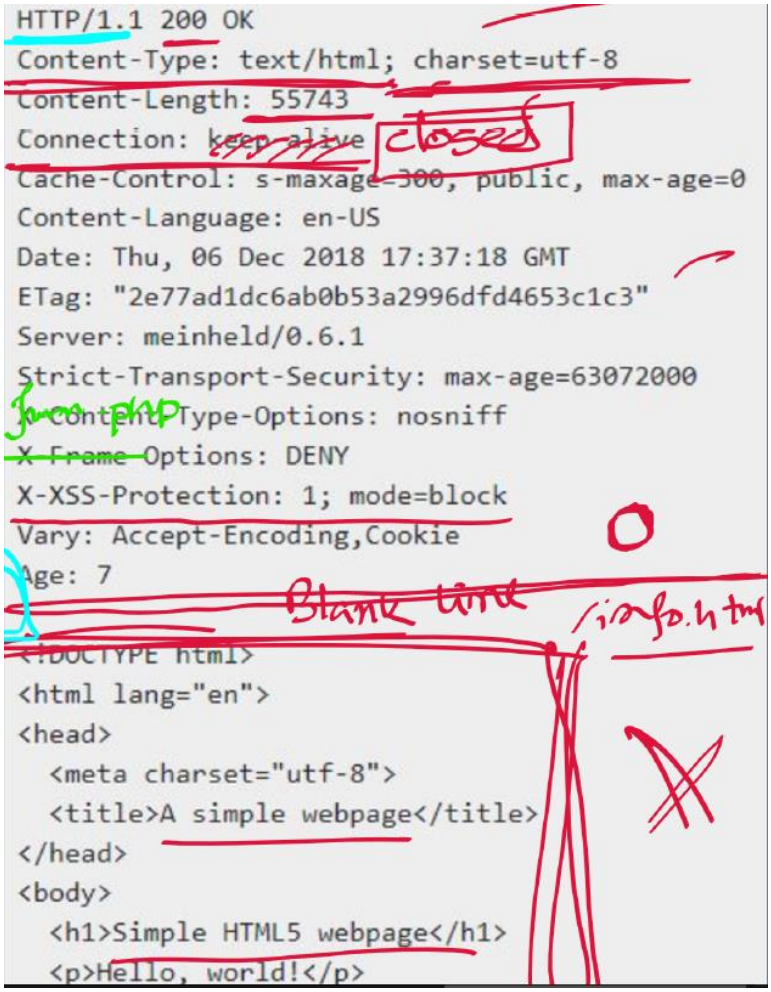
```
POST /cgi-bin/process.cgi HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: www.tutorialspoint.com
Content-Type: application/x-www-form-urlencoded
Content-Length: length
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: Keep-Alive

licenseID=string&content=string&/paramsXML=string
```

http response

format





x-xss-protection: cross site scripting attack protection. When server is sending a html doc, there can be possibility that someone may inject malicious script so that when the doc will be running in the browser it can cause harm. This protection means that the server is ensuring that the response data are safe from malicious/unwanted scrpit injection

Status Code

The Status-Code element is a 3-digit integer where first digit of the Status-Code defines the class of response and the last two digits do not have any categorization role. There are 5 values for the first digit:

S.N.	Code and Description
1	**1xx: Informational** It means the request was received and the process is continuing.
2	**2xx: Success** It means the action was successfully received, understood, and accepted.
3	**3xx: Redirection** It means further action must be taken in order to complete the request.
4	**4xx: Client Error** It means the request contains incorrect syntax or cannot be fulfilled.
5	**5xx: Server Error** It means the server failed to fulfill an apparently valid request.

HTTP METHODS

The set of common methods for HTTP/1.1 is defined below and this set can be expanded based on requirements. These method names are case sensitive and they must be used in uppercase.

S.N.	Method and Description
1	**GET**

	The GET method is used to retrieve information from the given server using a given URI. Requests using GET should only retrieve data and should have no other effect on the data.
2	**HEAD** Same as GET, but transfers the status line and header section only.
3	**POST** A POST request is used to send data to the server, for example, customer information, file upload, etc. using HTML forms.
4	**PUT** Replaces all current representations of the target resource with the uploaded content.
5	**DELETE** Removes all current representations of the target resource given by a URI.
6	**CONNECT** Establishes a tunnel to the server identified by a given URI.
7	**OPTIONS** Describes the communication options for the target resource.
8	**TRACE** Performs a message loop-back test along the path to the target resource.

GET Method

A GET request retrieves data from a web server by specifying parameters in the URL portion of the request. This is the main method used for document retrieval. The following example makes use of GET method to fetch hello.htm:

```
GET /hello.htm HTTP/1.1
```

```
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: www.tutorialspoint.com
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: Keep-Alive
```

The server response against the above HEAD request will be as follows:

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache/2.2.14 (Win32)
Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT
ETag: "34aa387-d-1568eb00"
Vary: Authorization,Accept
Accept-Ranges: bytes
Content-Length: 88
Content-Type: text/html
Connection: Closed
```

```
<html>
<body>
<h1>Hello, World!</h1>
</body>
</html>
```

HEAD Method

The HEAD method is functionally similar to GET, except that the server replies with a response line and headers, but no entity-body. The following example makes use of HEAD method to fetch header information about hello.htm:

```
HEAD /hello.htm HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: www.tutorialspoint.com
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: Keep-Alive
```

The server response against the above HEAD request will be as follows:

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache/2.2.14 (Win32)
Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT
ETag: "34aa387-d-1568eb00"
Vary: Authorization,Accept
Accept-Ranges: bytes
Content-Length: 88
```

```
Content-Type: text/html
Connection: Closed
```

You can notice that here server the does not send any data after header.

What is the HTTP HEAD request method used for?

The HTTP HEAD request may be executed before loading a large resource, to check resource size, validity, accessibility, and recent modification. (would be indicated by Content-Length/Last-Modified headers).

POST Method

The POST method is used when you want to send some data to the server, for example, file update, form data, etc. The following example makes use of POST method to send a form data to the server, which will be processed by a process.cgi and finally a response will be returned:

```
POST /cgi-bin/process.cgi HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: www.tutorialspoint.com
Content-Type: text/xml; charset=utf-8
Content-Length: 88
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: Keep-Alive
```

```
<?xml version="1.0" encoding="utf-8"?>
<string xmlns="http://clearforest.com/">string</string>
```

The server side script process.cgi processes the passed data and sends the following response:

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache/2.2.14 (Win32)
Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT
ETag: "34aa387-d-1568eb00"
Vary: Authorization,Accept
Accept-Ranges: bytes
Content-Length: 88
Content-Type: text/html
Connection: Closed
```

```
<html>
<body>
<h1>Request Processed Successfully</h1>
</body>
```

```
</html>
```

PUT Method

The PUT method is used to request the server to store the included entity-body at a location specified by the given URL. The following example requests the server to save the given entity-body in **hello.htm** at the root of the server:

```
PUT /hello.htm HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: www.tutorialspoint.com
Accept-Language: en-us
Connection: Keep-Alive
Content-type: text/html
Content-Length: 182
```

```
<html>
<body>
<h1>Hello, World!</h1>
</body>
</html>
```

The server will store the given entity-body in **hello.htm** file and will send the following response back to the client:

```
HTTP/1.1 201 Created
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache/2.2.14 (Win32)
Content-type: text/html
Content-length: 30
Connection: Closed
```

```
<html>
<body>
<h1>The file was created.</h1>
</body>
</html>
```

DELETE Method

The DELETE method is used to request the server to delete a file at a location specified by the given URL. The following example requests the server to delete the given file **hello.htm** at the root of the server:

```
DELETE /hello.htm HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
```

```
Host: www.tutorialspoint.com
Accept-Language: en-us
Connection: Keep-Alive
```

The server will delete the mentioned file **hello.htm** and will send the following response back to the client:

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache/2.2.14 (Win32)
Content-type: text/html
Content-length: 30
Connection: Closed
<html>
<body>
<h1>URL deleted.</h1>
</body>
</html>
```

CONNECT Method

The CONNECT method is used by the client to establish a network connection to a web server over HTTP. The following example requests a connection with a web server running on the host tutorialspoint.com:

```
CONNECT www.tutorialspoint.com HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
```

The connection is established with the server and the following response is sent back to the client:

```
HTTP/1.1 200 Connection established
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache/2.2.14 (Win32)
```

OPTIONS Method

The OPTIONS method is used by the client to find out the HTTP methods and other options supported by a web server. The client can specify a URL for the OPTIONS method, or an asterisk (*) to refer to the entire server. The following example requests a list of methods supported by a web server running on tutorialspoint.com:

```
OPTIONS * HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
```

The server will send an information based on the current configuration of the server, for example:

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache/2.2.14 (Win32)
Allow: GET,HEAD,POST,OPTIONS,TRACE
Content-Type: httpd/unix-directory
```

TRACE Method

The TRACE method is used to echo the contents of an HTTP Request back to the requester which can be used for debugging purpose at the time of development. The following example shows the usage of TRACE method:

```
TRACE / HTTP/1.1
Host: www.tutorialspoint.com
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
```

The server will send the following message in response to the above request:

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache/2.2.14 (Win32)
Connection: close
Content-Type: message/http
Content-Length: 39

TRACE / HTTP/1.1
Host: www.tutorialspoint.com
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
```

PATCH METHOD

The HTTP PATCH method is used to make partial changes to an existing resource. Typically, the PATCH method applies partial modifications to a resource, while the PUT method performs a complete replacement of the resource.
Unlike GET and HEAD requests, the PATCH requests may change the server state.

The PATCH method is not idempotent, which means that sending an identical PATCH request multiple times may additionally affect the state or cause further side effects.

********* https://www.tutorialspoint.com/http/index.htm *********

CONNECT METHOD: A CONNECT request urges your proxy to establish an HTTP tunnel to the remote end-point. **Usually** is it used for SSL connections, though it can be used with HTTP as well (used for the purposes of proxy-chaining and tunneling)

CONNECT www.google.com:443
The above line opens a connection from proxy to www.google.com on port 443. After this, content that is sent by the client is forwarded by the proxy to www.google.com:443.

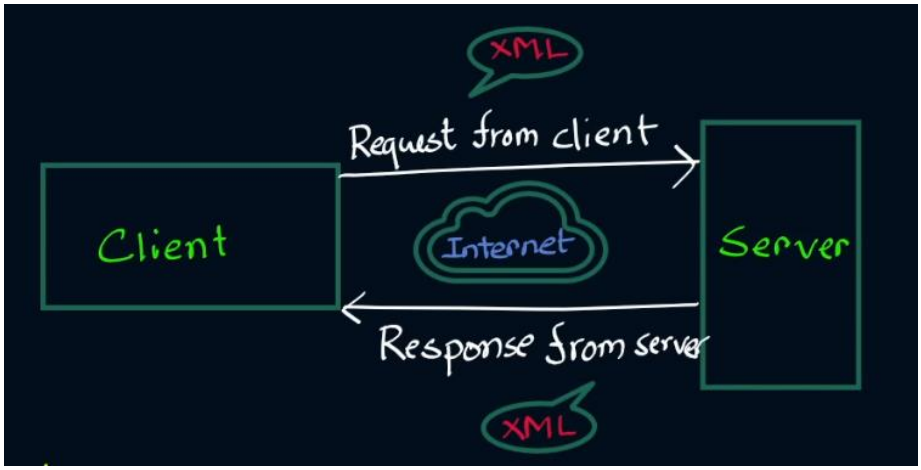
If a user tries to retrieve a page http://www.google.com, the proxy can send the exact same request and retrieve response for him, on his behalf.

With SSL(HTTPS), only the two remote end-points understand the requests, and the proxy cannot decipher them. Hence, all it does is open that tunnel using CONNECT, and lets the two end-points (webserver and client) talk to each other directly.

Web services

Def: a standard medium to propagate communication between client and server over a network. It Uses a standardized XML messaging system and not tied to any one operating system or programming language.

A s/w module to perform certain set of tasks.



Web service takes all the request from different systems in Xml(req,res)

Example

Consider a simple account-management and order processing system. The accounting personnel use a client application built with Visual Basic or JSP to create new accounts and enter new customer orders.

The processing logic for this system is written in Java and resides on a Solaris machine, which also interacts with a database to store information.

The steps to perform this operation are as follows −

- The client program bundles the account registration information into a SOAP message.
- This SOAP message is sent to the web service as the body of an HTTP POST request.
- The web service unpacks the SOAP request and converts it into a command that the application can understand.
- The application processes the information as required and responds with a new unique account number for that customer.

- Next, the web service packages the response into another SOAP message, which it sends back to the client program in response to its HTTP request.
- The client program unpacks the SOAP message to obtain the results of the account registration process.

How Does a Web Service Work?

A web service enables communication among various applications by using open standards such as HTML, XML, WSDL, and SOAP. A web service takes the help of −

- XML to tag the data
- SOAP to transfer a message
- WSDL to describe the availability of service.

Web services uses 'SOAP over HTTP' protocol to pass msg.

2 types web service: SOAP (protocol based WS)

RESTful (architectural WS)

Web service Advantage:

1. **Exposing the Existing Function on the network**
   A web service can be remotely invoked using HTTP. Web services allow us to expose the functionality of existing code over the network. Once it is exposed on the network, other applications can use the functionality of the program.
2. **Interoperability**
   Web services allow various applications to talk to each other and share data and services among themselves. Other applications can also use the web services. For example, a VB or .NET application can talk to Java web services and vice versa. Web services are used to make the application platform and technology independent.
3. **Standardized Protocol**
   Web services use standardized industry standard protocol for the communication. All the four layers (Service Transport, XML Messaging, Service Description, and Service Discovery layers) use well-defined protocols in the web services protocol stack. This standardization of

protocol stack gives the business many advantages such as a wide range of choices, reduction in the cost due to competition, and increase in the quality.

4. **Reduction in cost of Communication**
   Web services use SOAP over HTTP protocol, so you can use your existing low-cost internet for implementing web services. This solution is much less costly compared to proprietary solutions like EDI/B2B. Besides SOAP over HTTP, web services can also be implemented on other reliable transport mechanisms like FTP.

Web services have the following special behavioral characteristics −

1. **XML-Based**: Web services use XML at data representation and data transportation layers. Using XML eliminates any networking, operating system, or platform binding. Web services based applications are highly interoperable at their core level.
2. **Loosely Coupled:** A consumer of a web service is not tied to that web service directly. The web service interface can change over time without compromising the client's ability to interact with the service. Adopting a loosely coupled architecture tends to make software systems more manageable and allows simpler integration between different systems.
3. A**bility to be Synchronous or Asynchronous**: In synchronous invocations, the client blocks and waits for the service to complete its operation before continuing. Asynchronous operations allow a client to invoke a service and then execute other functions. Asynchronous clients retrieve their result at a later point in time, while synchronous clients receive their result when the service has completed. Asynchronous capability is a key factor in enabling loosely coupled systems.
4. **Supports Remote Procedure Calls**(RPCs): Web services allow clients to invoke procedures, functions, and methods on remote objects using an XML-based protocol. Remote procedures expose input and output parameters that a web service must support. A web service supports RPC by providing services of its own, equivalent to those of a traditional

component, or by translating incoming invocations into an invocation of an EJB or a .NET component.

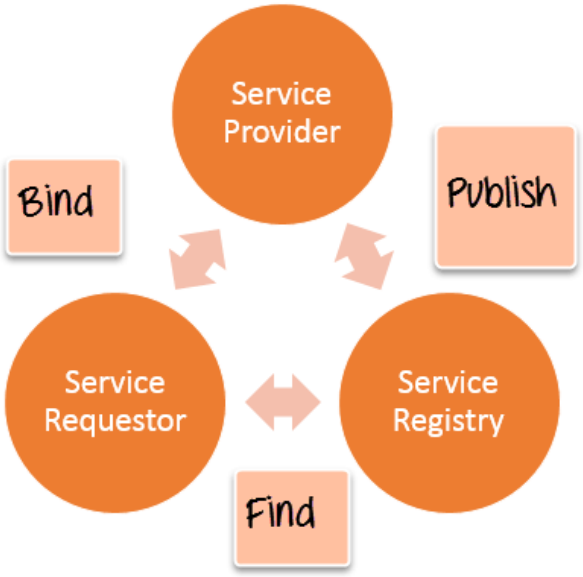
5. **Supports Document Exchange:** One of the key advantages of XML is its generic way of representing not only data, but also complex documents (books). Web services support the transparent exchange of documents to facilitate business integration.

Web architecture

Every framework needs some sort of architecture to make sure the entire framework works as desired, similarly, in web services. The **Web Services Architecture** consists of three distinct roles as given below :

1. **Provider** - The provider creates the web service and makes it available to client application who want to use it.
2. **Requestor** - A requestor is nothing but the client application that needs to contact a web service. The client application can be a .Net, Java, or any other language based application which looks for some sort of functionality via a web service.
3. **Broker** - The broker is nothing but the application which provides access to the UDDI. The UDDI, as discussed in the earlier topic enables the client application to locate the web service.

The diagram below showcases how the Service provider, the Service requestor and Service registry interact with each other.



Web Services Architecture

1. **Publish** - A provider informs the broker (service registry) about the existence of the web service by using the broker's publish interface to make the service accessible to clients
2. **Find** - The requestor consults the broker to locate a published web service
3. **Bind** - With the information it gained from the broker(service registry) about the web service, the requestor is able to bind, or invoke, the web service.

Web Service Protocol Stack

A second option for viewing the web service architecture is to examine the emerging web service protocol stack. The stack is still evolving, but currently has four main layers.

Service Transport

This layer is responsible for transporting messages between applications. Currently, this layer includes Hyper Text Transport Protocol (HTTP), Simple

Mail Transfer Protocol (SMTP), File Transfer Protocol (FTP), and newer protocols such as Blocks Extensible Exchange Protocol (BEEP).

XML Messaging

This layer is responsible for encoding messages in a common XML format so that messages can be understood at either end. Currently, this layer includes XML-RPC and SOAP.

Service Description

This layer is responsible for describing the public interface to a specific web service. Currently, service description is handled via the Web Service Description Language (WSDL).

Service Discovery

This layer is responsible for centralizing services into a common registry and providing easy publish/find functionality. Currently, service discovery is handled via Universal Description, Discovery, and Integration (UDDI).

Blocks Extensible Exchange Protocol (BEEP)

This is a promising alternative to HTTP. BEEP is a new Internet Engineering Task Force (IETF) framework for building new protocols. BEEP is layered directly on TCP and includes a number of built-in features, including an initial handshake protocol, authentication, security, and error handling. Using BEEP, one can create new protocols for a variety of applications, including instant messaging, file transfer, content syndication, and network management.

SOAP is not tied to any specific transport protocol.

Web service components:

Remote procedure call (rpc)

SOAP

WSDL

UDDI

What is XML-RPC

Sir: there can be multiple methods (with diff functionality) in a website. As a client we can call these methods through RPC. Methods are hosted by web services

XML-RPC is among the simplest and most foolproof web service approaches that makes it easy for **computers to call procedures on other computers**.

- XML-RPC permits programs to make **function or procedure calls across a network**.
- XML-RPC uses the HTTP protocol to pass information from a client computer to a server computer.
- XML-RPC uses a small XML vocabulary to describe the nature of requests and responses.
- XML-RPC client specifies a procedure name and parameters in the XML request, and the server returns either a fault or a response in the XML response.
- XML-RPC parameters are a simple list of types and content - structs and arrays are the most complex types available.
- XML-RPC has no notion of objects and no mechanism for including information that uses other XML vocabulary.
- With XML-RPC and web services, however, the Web becomes a collection of procedural connections where computers exchange information along tightly bound paths.

Why XML-RPC?

If you need to **integrate multiple computing environments**, but don't need to share complex data structures directly, you will find that XML-RPC lets you establish communications quickly and easily.

Even if you work within a single environment, you may find that the RPC approach makes it easy to connect programs that have different data models or processing expectations and that it can provide easy access to reusable logic.

Soap

Def: SOAP (Simple Object Access Protocol). It is a transport independent XML-based messaging protocol for exchanging information among computers. **the initial focus of SOAP is remote procedure calls transported via HTTP.** Only the structure of the XML document follows a specific pattern, but not the content.

Points to Note

- SOAP is a communication protocol designed to communicate via Internet.
- SOAP can extend HTTP for XML messaging.
- SOAP provides data transport for Web services.
- SOAP can exchange complete documents or call a remote procedure.
- SOAP can be used for broadcasting a message.
- SOAP is written in XML, so it is **platform- and language-independent**.
- SOAP is the XML way of defining what information is sent and how.
- SOAP enables client applications to easily connect to remote services and invoke remote methods.

Although SOAP can be used in a variety of messaging systems and can be delivered via a variety of transport protocols, **the initial focus of SOAP is remote procedure calls transported via HTTP.**

A SOAP message is an ordinary XML document containing the following elements −

- **Envelope** − Defines the start and the end of the message. It is a mandatory element. Root element.
- **Header** − The header contains the routing data which is basically the information which tells the XML document to which client it needs to be sent to.

- **Body** − Contains the XML data comprising the message being sent (actual data). It is a mandatory element.
- **Fault** − An Optional Fault element that provides information about errors that occur while processing the message.

```
<?xml version = "1.0"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV = "http://www.w3.org/2001/12/soap-envelope"
   SOAP-ENV:encodingStyle = "http://www.w3.org/2001/12/soap-encoding">

   <SOAP-ENV:Header>
      ...
      ...
   </SOAP-ENV:Header>
   <SOAP-ENV:Body>
      ...
      ...
      <SOAP-ENV:Fault>
         ...
         ...
      </SOAP-ENV:Fault>
      ...
   </SOAP-ENV:Body>
</SOAP_ENV:Envelope>
```

SOAP via HTTP

SOAP requests are sent via an HTTP request and SOAP responses are returned within the content of the HTTP response. Additionally, both HTTP requests and responses are required to set their content type to text/xml. SOAP responses delivered via HTTP are required to follow the same HTTP status codes. For example, a status code of 200 OK indicates a successful response. A status code of 500 Internal Server Error indicates that there is a server error and that the SOAP response includes a Fault element.

SOAP includes a built-in set of rules for encoding data types. It enables the SOAP message to indicate specific data types, such as integers, floats, doubles, or arrays.

WSDL

A web service cannot be used if it cannot be found. The client invoking the web service should know where the web service actually resides.

Secondly, the client application needs to know what the web service actually does, so that it can invoke the right web service. This is done with the help of the WSDL, known as the Web services description language. The WSDL file is an XML-based file which basically tells the client application what the web service does. By using the WSDL document, the client application would be able to understand where the web service is located and how it can be utilized. **WSDL is often used in combination with SOAP and XML Schema to provide web services over the Internet**. WSDL is the language that UDDI uses.

Given below is a WSDL file that is provided to demonstrate a simple WSDL program.

Let us assume the service provides a single publicly available function, called *sayHello*. This function expects a single string parameter and returns a single string greeting. For example, if you pass the parameter *world* then service function *sayHello* returns the greeting, "Hello, world!".

Example

Contents of HelloService.wsdl file −

```
<definitions name = "HelloService"
   targetNamespace = "http://www.examples.com/wsdl/HelloService.wsdl"
   xmlns = "http://schemas.xmlsoap.org/wsdl/"
   xmlns:soap = "http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:tns = "http://www.examples.com/wsdl/HelloService.wsdl"
   xmlns:xsd = "http://www.w3.org/2001/XMLSchema">

   <message name = "SayHelloRequest">
      <part name = "firstName" type = "xsd:string"/>
   </message>

   <message name = "SayHelloResponse">
      <part name = "greeting" type = "xsd:string"/>
   </message>

   <portType name = "Hello_PortType">
      <operation name = "sayHello">
         <input message = "tns:SayHelloRequest"/>
```

Commented [ft1]: Data element for each operation, it is an abstract definition of the data, in the form of a message presented either as an entire document or as arguments to be mapped to a method invocation.

```
      <output message = "tns:SayHelloResponse"/>
    </operation>
  </portType>

  <binding name = "Hello_Binding" type = "tns:Hello_PortType">
    <soap:binding style = "rpc"
      transport = "http://schemas.xmlsoap.org/soap/http"/>
    <operation name = "sayHello">
      <soap:operation soapAction = "sayHello"/>
      <input>
        <soap:body
          encodingStyle = "http://schemas.xmlsoap.org/soap/encoding/"
          namespace = "urn:examples:helloservice"
          use = "encoded"/>
      </input>

      <output>
        <soap:body
          encodingStyle = "http://schemas.xmlsoap.org/soap/encoding/"
          namespace = "urn:examples:helloservice"
          use = "encoded"/>
      </output>
    </operation>
  </binding>

  <service name = "Hello_Service">
    <documentation>WSDL File for HelloService</documentation>
    <port binding = "tns:Hello_Binding" name = "Hello_Port">
      <soap:address
        location = "http://www.examples.com/SayHello/" />
    </port>
  </service>
</definitions>
```

Commented [ft2]: consists of a request and a response service.

Example Analysis

- **Definitions** − HelloService
- **Type** − Using built-in data types and they are defined in XMLSchema.
- **Message** −

    - sayHelloRequest − firstName parameter
    - sayHelloresponse − greeting return value
- **Port Type** − sayHello operation that consists of a request and a response service.
- **Binding** − Direction to use the SOAP HTTP transport protocol.
- **Service** − Service available at http://www.examples.com/SayHello/
- **Port** − Associates the binding with the URI http://www.examples.com/SayHello/ where the running service can be accessed.

UDDI

UDDI (**Universal Description, Discovery, and Integration**) is an XML-based standard for describing, publishing, and finding web services. It is a database which contains all WSDL Files.

- UDDI is a platform-independent, open framework.
- UDDI can communicate via SOAP, CORBA, Java RMI Protocol.
- UDDI uses Web Service Definition Language(WSDL) to describe interfaces to web services.
- **UDDI is seen with SOAP and WSDL as one of the three foundation standards of web services.**
- UDDI is an open industry initiative, enabling businesses to discover each other and define how they interact over the Internet.

UDDI has two sections −

- A registry of all web service's metadata, including a pointer to the WSDL description of a service.
- A set of WSDL port type definitions for manipulating and searching that registry.

REST (Representational State Transfer)

What is : RESTful Web Services are basically REST Architecture based Web Services. In REST Architecture everything is a resource. It's more of an approach **that's centered around resources and things you can do to resources**. The HTTP verbs GET, POST, PUT and DELETE are typical actions that you can apply against any resource.
RESTful web services use the **HTTP protocol** to perform requests from a web service. The requests themselves are to URLs that represent resources (may not need body). Sometimes the requests will contain data in the body that could by HTML, JSON, binary data or other.
A purely RESTful web service only requires the URL and the HTTP verb to describe the requested action. The body data is usually a payload to be involved in the requested action. It should not dictate the requested action.

***Service Exposure* :** REST uses Uniform Service locators/URL to access to the components on the hardware device. For example, if there is an object which represents the data of an employee hosted on a URL as http://demo.guru99 , the
below are some of URI that can exist to access them
http://demo.guru99.com/Employee
http://demo.guru99.com/Employee/1

Bandwidth: REST does not need much bandwidth when requests are sent to the server. REST messages mostly just consist of JSON messages.
Data format: REST permits different data formats such as Plain text, HTML, XML, JSON, etc. But the most preferred format for transferring data is JSON.

Rules of REST API:
● REST is based on the resource or noun instead of action or verb based. It means that a URI of a REST API should always end with a noun. Example:
/api/users is a good example, but /api?type=users is a bad example of creating a REST API.
● HTTP verbs are used to identify the action. Some of the HTTP verbs are – GET, PUT, POST, DELETE, UPDATE, PATCH.
● A web application should be organized into resources like users and then

uses HTTP verbs like – GET, PUT, POST, DELETE to modify those resources.
And as a developer it should be clear that what needs to be done just by looking at the endpoint and HTTP method used.
URI HTTP VERB DESCRIPTION
api/users GET Get all users
api/users POST Add a user
api/users/1 PUT Update a user with id = 1
api/users/1 DELETE Delete a user with id = 1
api/users/1 GET Get a user with id = 1
● Always use plurals in URL to keep an API URI consistent throughout the application.

● Send a proper HTTP code to indicate a success or error status.

When to use REST?

One of the most highly debatable topics is when REST should be used or when to use SOAP while designing web services. Below are some of the key factors that determine when each technology should be used for web services **REST services should be used in the following instances**

- **Limited resources and bandwidth** – Since SOAP messages are heavier in content and consume a far greater bandwidth, REST should be used in instances where network bandwidth is a constraint.
- **Statelessness** – If there is no need to maintain a state of information from one request to another then REST should be used. If you need a proper information flow wherein some information from one request needs to flow into another then SOAP is more suited for that purpose. We can take the example of any online purchasing site. These sites normally need the user first to add items which need to be purchased to a cart. All of the cart items are then transferred to the payment page in order to complete the purchase. This is an example of an application which needs the state feature. The state of the cart items needs to be transferred to the payment page for further processing.
- **Caching** – If there is a need to cache a lot of requests then REST is the perfect solution. At times, clients could request for the same resource multiple times. This can increase the number of requests which are sent to the server. By implementing a cache, the most frequent queries results can be stored in an intermediate location. So whenever the client requests for a resource, it will first check the

cache. If the resources exist then, it will not proceed to the server. So caching can help in minimizing the amount of trips which are made to the web server.

- **Ease of coding** – Coding REST Services and subsequent implementation is far easier than SOAP. So if a quick win solution is required for web services, then REST is the way to go.

When to use SOAP?

SOAP should be used in the following instances

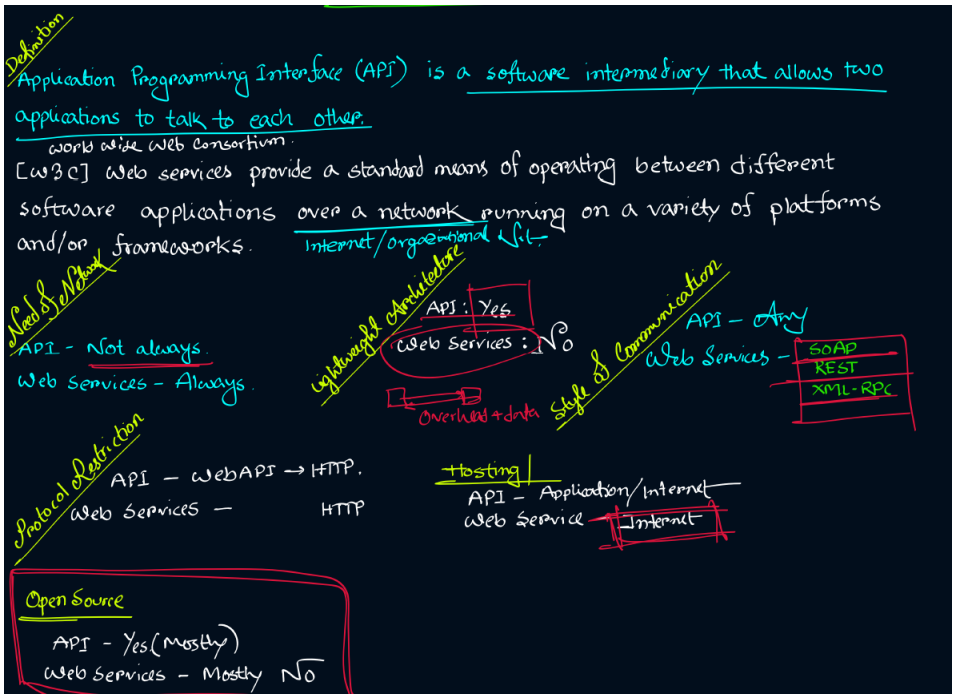
1. **Asynchronous processing and subsequent invocation** – if there is a requirement that the client needs a guaranteed level of reliability and security then the new SOAP standard of SOAP 1.2 provides a lot of additional features, especially when it comes to security.
2. **A Formal means of communication** – if both the client and server have an agreement on the exchange format then SOAP 1.2 gives the rigid specifications for this type of interaction. An example is an online purchasing site in which users add items to a cart before the payment is made. Let's assume we have a web service that does the final payment. There can be a firm agreement that the web service will only accept the cart item name, unit price, and quantity. If such a scenario exists then, it's always better to use the SOAP protocol.
3. **Stateful operations –** if the application has a requirement that state needs to be maintained from one request to another, then the SOAP 1.2 standard provides the WS* structure to support such requirements.

Difference between REST API and SOAP API

- SOAP follows a strict guideline to allow communication between the client and the server whereas REST is an architectural style that doesn't follow any strict standard
- SOAP uses only XML in its message format whereas REST use XML, JSON, Plain-text.
- REST can use SOAP protocol but SOAP cannot use REST.
- SOAP -> to call a method needs RPC, but for REST, URL is enough.

- SOAP is difficult to implement and it requires more bandwidth whereas REST is easy to implement and requires less bandwidth such as smartphones.
- Benefits of SOAP over REST as SOAP has ACID complaints transaction. Some of the applications require transaction ability which is accepted by SOAP whereas REST lacks in it.
- On the basis of Security, SOAP has SSL(**S**ecure **S**ocket **L**ayer) and WS-security whereas REST has SSL and HTTPS. In the case of Bank Account Password, Card Number, etc. SOAP is preferred over REST. The security issue is all about your application requirement, you have to build security on your own.

Web service vs API

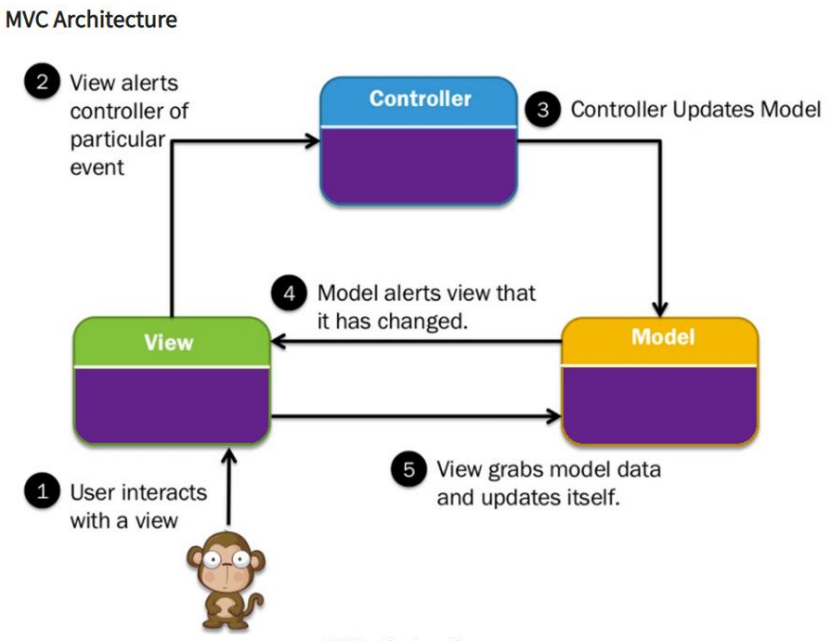


MVC

The **Model-View-Controller (MVC)** framework is an architectural pattern that separates an application into three main logical components Model, View, and Controller. Hence the abbreviation MVC. Each architecture component is built to handle specific development aspect of an application. MVC separates the business logic and presentation layer from each other. First developed for desktop, now web +app too.

Features of MVC

- Supports for Test Driven Development (TDD). Highly testable, extensible and pluggable framework
- Offers full control over your HTML & URLs
- Code reusability
- Clear separation of logic: Model, View, Controller. Separation of business logic, Ul logic, and input logic
- Easy debugging



MVC Architecture Diagram

- **Model**: It includes all the data and its related logic
- **View**: Present data to the user or handles user interaction
- **Controller**: An interface between Model and View components

View: responsible for visual elements

A View is that part of the application that represents the presentation of data.

Views are created by the data collected from the model data.

The view also represents the data from charts, diagrams, and table. For example, any customer view will include all the UI components like text boxes, drop downs, etc.

Controller: connect model & view, set of methods, only controller knows the biz logic

The Controller handles the user interaction. The controller interprets the mouse and keyboard inputs from the user, informing model and the view to change as appropriate.

A Controller send's commands to the model to update its state(E.g., Saving a specific document). The controller also sends commands to its associated view to change the view's presentation.

Model: DB/storage

The model component stores data and its related logic. It represents data that is being transferred between controller components or any other related business logic. For example, a Controller object will retrieve the customer info from the database. It manipulates data and send back to the database or use it to render the same data.

It responds to the request from the views and also responds to instructions from the controller to update itself. It is also the lowest level of the pattern which is responsible for maintaining data.

Advantages of MVC: Key Benefits

Here, are major benefits of using MVC architecture.

- Easy code maintenance
- Model component can be tested separately from the user
- Easy debugging
- Easy for multiple developers to collaborate and work together.
- It helps you to avoid complexity by dividing an application into the three units. Model, view, and controller
- Offers the best support for test-driven development
- Provides clean separation of concerns (SoC).
- Search Engine Optimization (SEO) Friendly.
- MVC allows logical grouping of related actions on a controller together.

Disadvantages of using MVC

- It is hard to understand the MVC architecture.
- Must have strict rules on methods.

N tier

Key terms:

- **Distributed Network:** It is a network architecture, where the components located in different computers and communicate through message passing. It is a collection of multiple systems situated at **different nodes** but appears to the user as a single system.
    - It provides a single data communication network which can be managed separately by different networks.
    - Example: LAN
- **Client-Server Architecture:** It is an architecture model where the client (one program) requests a service from a server (another program) **i.e.** It is a request-response service provided over the internet or through an intranet. Example: ATM machine. Bank-server, ATM-client

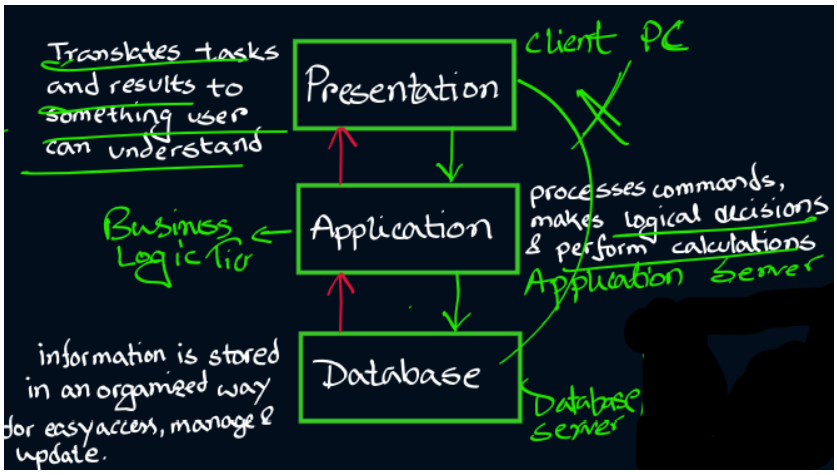
- **Platform:** a platform is a system on which applications program can run. It is a combination of hardware and software to perform specific operations. Example: Windows 2000 or Mac OS X
- **Database:** It is a collection of information in an organized way so that it can be easily accessed, managed and updated. Example: mysql,oracle

N-Tier Application program is one that is distributed among **n** separate computers in a distributed network. It is suitable to support **enterprise level** client-server applications by providing solutions to **scalability, security, fault tolerance, reusability, and maintainability.** It helps developers to create flexible and reusable applications.

3 tier:

3-tier architecture has three different layers.

- Presentation layer
- Business Logic layer
- Database layer



Business Access Layer -

This is the function of the business layer which accepts the data from the application layer and passes it to the data layer.

- Business logic acts as an interface between Client layer and Data Access Layer
- All business logic – like validation of data, calculations, data insertion/modification are written under business logic layer.
- It makes communication faster and easier between the client and data layer
- Defines a proper workflow activity that is necessary to complete a task.
- It sends request to the database server to store/retrieve data

** presentation tier can't communicate with database directly

2-Tier Architecture:

It is like Client-Server architecture, where communication takes place between client and server.

In this type of software architecture, the presentation layer or user interface layer runs on the client side while dataset layer gets executed and stored on server side.

There is no Business logic layer or immediate layer in between client and server.

Single Tier or 1-Tier Architecture:

It is the simplest one as it is equivalent to running the application on the personal computer. All of the required components for an application to run are on a single application or server.

Presentation layer, Business logic layer, and data layer are all located on a single machine.

No network activity, since there is no other tier, there is no need for msg passing. s/w that follows 1 tier architecture, usually are STAND ALONE s/w

3+ tier:

We can divide the 3 tiers into multiple parts and make multi-tier architecture. For example:

1. Presentation tier
2. Biz logic
3. Data access
4. Database

Here the application tier/biz logic tier is divided into 2 more tiers (2,3). tier 2 is used for biz logic and tier 3 is used for sending request to DB to store/retrieve data. We can divide any other tier also.

Advantage / disadvantages

1 / 2 TIER:

ADVANTAGES

1. Since there is no multiple device msg passing, development is easy.
2. Client and database can communicate directly, fewer process calls. Hence fast for a lower number of users.
3. Low cost for h/w, maintenance, deployment (need less bandwidth)

DISADVANTAGES

1. No scalability properties. If there are large number of users, we need to create new connections. It will put more load in db.
2. Since client can directly access the database without any validation, there is no security.
3. Less fault tolerance properties. If the database server crashes, then the whole app will be stopped.
4. No reusability properties
5. Tough maintenance
6. Less data integrity. Since client can access the db directly without validation, we can't ensure that there will be data consistency.

3 TIER:

ADVANTAGES

1) **Fault tolerance**: since there are multiple tiers, even if any one of the server crashes, we can retrieve the data from different servers.
2) **Security**: Business logic, db tiers can use extra firewall separately for ensuring the security. We can also add extra tier for security (authentication, DOS prevention)
3) **Easy maintenance**: We can modify each tier without affecting other tiers
4) **More reusable**: we can use same module multiple times
5) **Data integrity:** since clients cannot access the db directly, data integrity remains intact
6) **Scalability**: Each tier is grouped with similar properties. App can be scaled up by db clustering. Multiple client can use the app server & from there they can access the db. Hence reducing the load on db. We can scale up by using load balancer without affecting other tiers. A separate back-end tier, for example, allows you to deploy to a variety of databases instead of being locked into one particular technology. It also allows you to scale up by adding multiple web servers.
7) **Reliability**: It adds reliability and more independence of the underlying servers or services

Commented [ft3]: sir

DISADVANTAGES

1. Needs more bandwidth, else the app will become slow
2. Higher cost in comparison to 1 / 2 tiers
3. Increase in complexity

SQL VS NOSQL

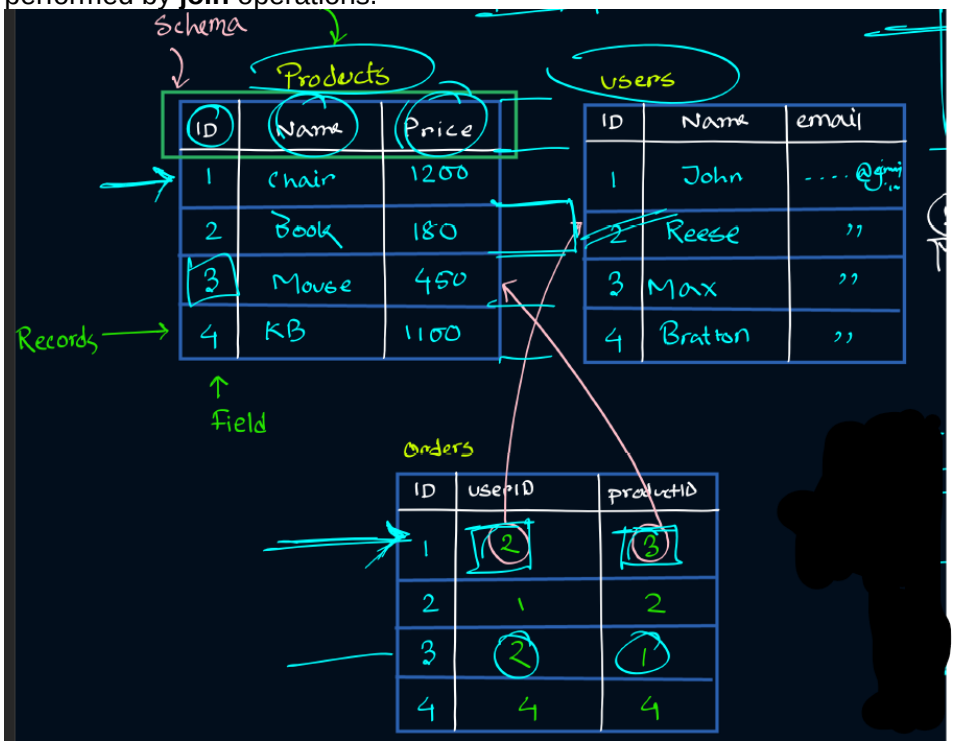
Sql

Follows a **strict schema**: defined tables, the sequence of fields must be maintained. Can't add new field for a particular record. Example:

Id,name, price
Id, price, name -- not allowed

Id, name,price ,qty -- not allowed

There are relations btn multiple tables. 1-1,1-M,M-M . the queries are performed by **join** operations.



NoSQL

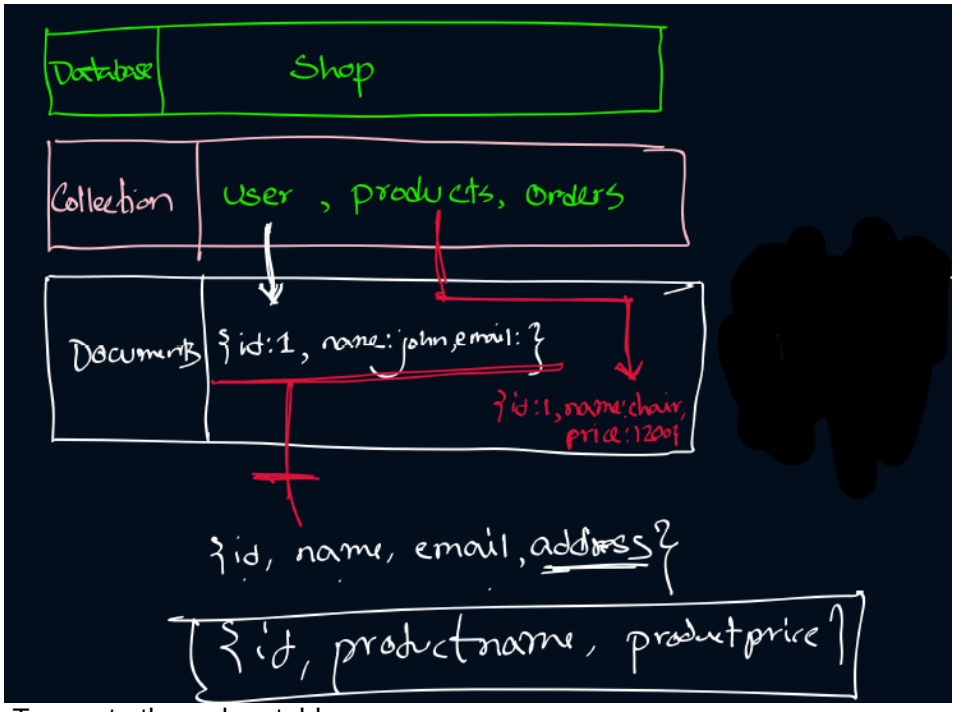
**

sql	nosql
table	collections
record	documents

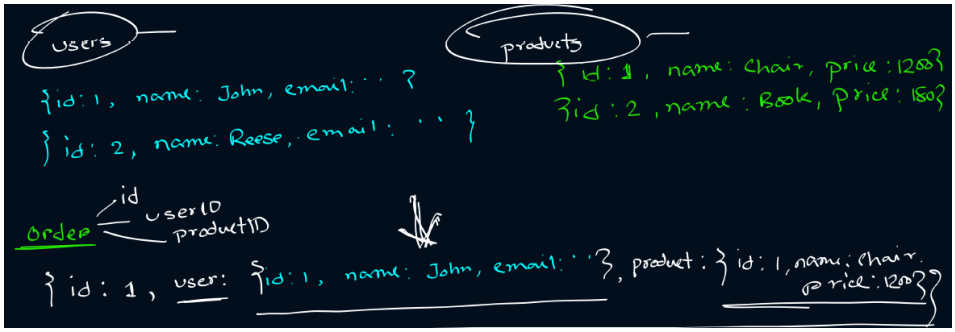
Non relational: there are no relations between multiple collections. No join operations.
Schemaless: don't follow strict schema.

Id,name, price
Id, price, name -- allowed
Id, name,price ,qty -- allowed

Flexible: For future change. NoSQL is preferable for those types of applications that are prone to frequent change. Example: a health related app used to take data for bp,pulse. Now they take O2 level too. If we use SQL then we had to modify all the previous records of the table (which is a hassle). But with Nosql we can add the new O2 data without modifying the previous records. Also not all the users need to add O2 level in Nosql.



To create the orders table:
It is not necessary to store userId,productId in the collection for Orders. There may be some data duplicacy



Why NoSQL?:
If the application performs a lot of read operations and less number of writing operations, then NoSQL is preferable.

Data read:
In SQL data retrieval is slower since data is scattered into multiple tables. Join operation takes some time. But in NoSQL there are no relations between collections. All the data is stored in one table. So data retrieval is faster (no need to access multiple collections). Also the 'users,products' table doesn't need to be in the same server as 'orders'. Bcz user,product info are already stored in the 'orders' table. To make NoSQL we need to design the db efficiently, else it will create problems (become slow, data inconsistency, manually validate data)

Data write:
In NoSQL it is slow. Because we need to store data in multiple tables (users,products). For orders-> need to fetch data from users and products and then store them. But since the app is handling write requests a lot less than read requests,it's better to use NoSQL. Example: Facebook-> we read posts from others more than we post ourselves.

The smaller number of collections the faster is NoSQL. How?
Design the database efficiently. Follow the hierarchical db. Example: Blog post=>

Sql: users,posts,comments.
NoSQL: users-----> id
> name
> address
> posts------> id
> desc
> comments-------> time
> id

Here we used 1 collection and add new **points** for a table. Can do it in reverse order too.
If the number of collections is the same as SQL then there might be some problems; data inconsistency (order table a user email). Need to manually code for data validation.

Sql vs NoSQL

SQL	**NoSQL**
Follows strict **schema**. (order of the fields, data format must be defined and same for all the records)	Schemaless. Changing the structure or schema will not impact development cycles or create any downtime for the application.
If Application is Less likely to **change**, use it	If Application is prone to change, use it
Relational db. Performs join operations	Non-relational db.
Data is distributed across **multiple tables**. Cons: data retrieval slow.	Data is Merged: order table. To get the advantages of NoSQL. Cons: data is not organized like SQL : may have duplicate data
Scalability: Vertical scaling is possible, horizontal is either very difficult/impossible. Here can't distribute data across	**Scalability:** Both vertical and horizontal scaling is possible. Since there are no relation between multiple tables, we can add multiple servers to store data. Then

multiple servers easily bcz of the relations between multiple tables.	again since there are multiple servers available, we can reduce load on the servers by distributing data across server. (it makes read request faster)
Easy to implement/design bcz of defined schema	Need to carefully/efficiently design the db
Defined schema helps to make a smaller number of mistakes	If not designed efficiently, we can make mistakes: data inconsistency, manual data validation

Scaling:

Vertical scaling: scaling vertically is adding extra hardware, RAM, processing power, etc. in order to increase capacity. With SQL we're limited because we will inevitably max out on capacity and scaling up is expensive

Horizontal scaling: When scaling out or horizontally we are adding resources to a single node (a computer or server). We can have one database working on multiple nodes. Scaling out (or back in) means we can easily add and remove nodes. Distributed servers

When to use SQL instead of NoSQL

1. Working with complex queries and reports. NoSQL doesn't support relations between data types. Running queries in NoSQL is doable, but much slower.
2. need to ensure ACID compliance or defining exactly how transactions interact with a database.
3. have a high transaction application. SQL databases are a better fit for heavy duty or complex transactions because it's more stable and ensure data integrity.
4. No need for a lot of changes or growth. If you're not working with a large volume of data or many data types, NoSQL would be overkill.

When to use NoSQL instead of SQL

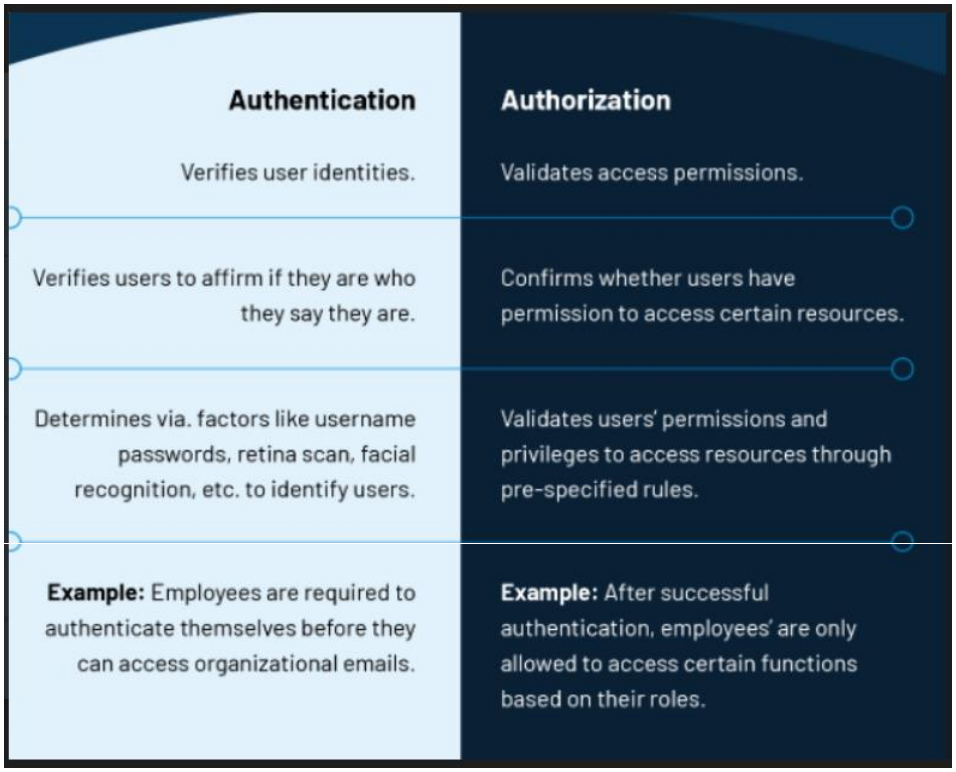
1. You are constantly adding new features, functions, data types, and it's difficult to predict how the application will grow over time.
2. Changing a data model in SQL is clunky and requires code changes. A lot of time is invested designing the data model because changes will impact all or most of the layers in the application.
3. In NoSQL, we are working with a highly flexible schema design or no predefined schema. The data modeling process is iterative and adaptive. Changing the structure or schema will not impact development cycles or create any downtime for the application.
4. You are not concerned about data consistency and 100% data integrity is not your top goal.
5. You have a lot of data, many different data types, and your data needs will only grow over time. NoSQL makes it easy to store all different types of data together and without having to invest time into defining what type of data you're storing in advance.
6. Your data needs scale up, out, and down. NoSQL provides much greater flexibility and the ability to control costs as your data needs change.

Authentication and authorization

Authentication: The process of verifying and confirming a user. It means to prove who I am. Usually, authentication by a server entails the use of a username and password. Other ways to authenticate can be through cards, retina scans, voice recognition, and fingerprints.

Authorization: The process of determining which user has access to what role/ what info he has access to. Authorization determines exactly what information the students are authorized to access on the university website after successful authentication

Authentication vs Authorization



Authentication	Authorization
Verifies user identities.	Validates access permissions.
Verifies users to affirm if they are who they say they are.	Confirms whether users have permission to access certain resources.
Determines via. factors like username passwords, retina scan, facial recognition, etc. to identify users.	Validates users' permissions and privileges to access resources through pre-specified rules.
Example: Employees are required to authenticate themselves before they can access organizational emails.	**Example:** After successful authentication, employees' are only allowed to access certain functions based on their roles.

If authentication is who you are, authorization is what you can access and modify. In simple terms, authentication is determining whether someone is who he claims to be. Authorization, on the other hand, is determining his rights to access resources.

Authentication factor: 3- something that you 1. Know

2. have

3. are

We can take any combination of these 3 to authenticate.

Single factor: only pass

2 factor auth: OTP+pass(have OTP,know pass) ; are + know (name/biometric + pass)

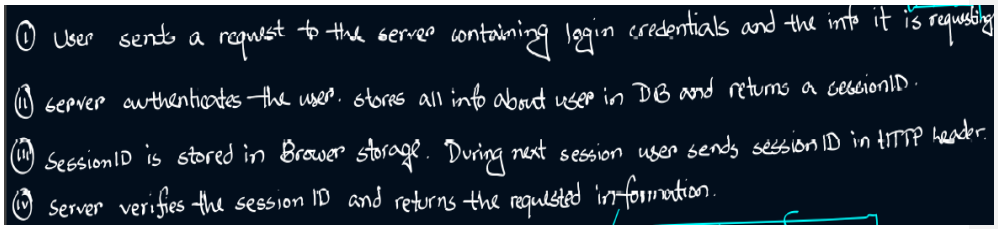
: atm (know pass+have card)

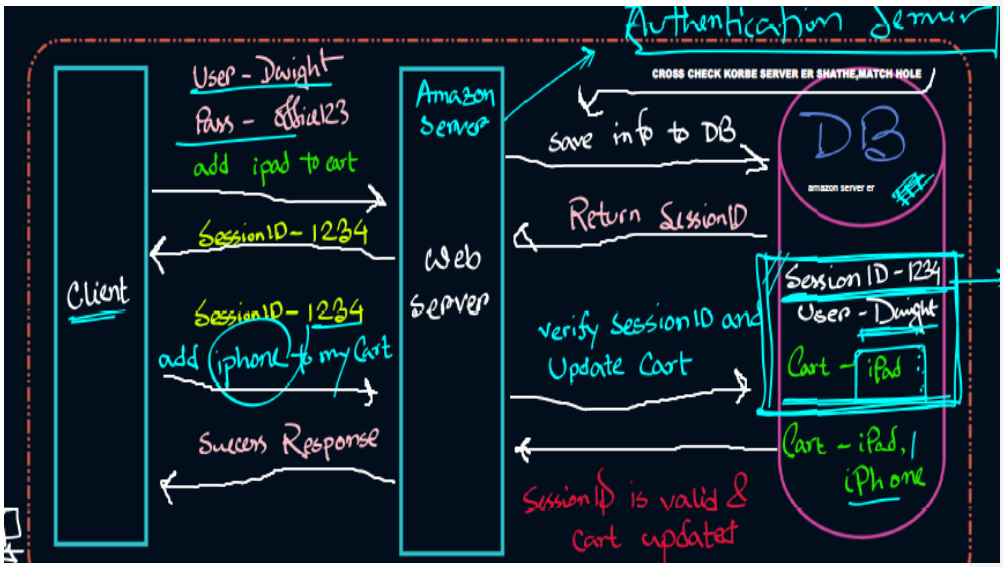
3 factor: know + have + are (name +card + pass)

#since http stateless,every time we send req we need to be authenticated. To prevent this hassle and security risk session / token base auth.

Session based authentication:

Steps to create a session btn client and web server:





Usually session time is until the tab/browser is closed. But we can change it according to the app needs.

What happens if we make the session ID availability infinite?

Session id works only between client and server. Clients send the session id in http header through a shared line, someone else might steal the session id & use it to send different http request impersonating as the user. Then the server thinks that the client sent that req and send appropriate res. This is also known as **CSRF** attack. That is why session id is temporary, so that even if someone does get the session id, he can't use it. Since validity would be expired.

Browser storage:

a) Local
b) Session
c) Cookie – 1. Session cookie
2. permanent cookie
3. 3rd party cookie

	Local storage	Session storage	Cookie
Storage limit	5mb	5-10mb	4kb
Accessible	Client+server	Client+server	Server Client- not always (if HTTPonly flag is true then client side code/js can't access)
Validity	Lifetime (unless browser is uninstalled/ manually delete storage)	Browser/tab close	**Session cookie**: till the session is valid. **Permanent**: lifetime **3rd party cookie:** used for refined search/browsing habit.

Cross-site request forgery (or CSRF) is a web security vulnerability that allows an attacker to induce a victim user to perform actions that they do not intend to. CSRF can be described as a "one-way" vulnerability, in that while an attacker can induce the victim to issue an HTTP request, they cannot retrieve the response from that request

CSRF Attack : Someone acts as client and send requests to server. Basically impersonating a trusted user. This is happen if attacker steals the Session ID. Also known as Cookie Fraud.

Cookie flag

HTTP only: to prevent XSS attack. If it is true then only the **app server can access** the cookie, no 3rd party can access it.

Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted websites: Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into trusted websites. When app server sends res to the client. In the mid way attacker can inject some JS code and send it to the client. After that the attacker can get the cookie from users browsers and send

req to the server impersonating as the client. Again they can create dummy server & req the app server to get cookie. That's why HTTP-only is used

Secure flag: A **secure flag** is set by the application server while sending a new cookie to the user using an HTTP Response. The **secure flag** is used to prevent cookies from being observed and manipulated by an unauthorized party or parties. It needs SSL certificate (means cookie will be encrypted) to send cookie. The req will be sent as HTTPS.

The cookie is sent automatically. In the http res, there is a attribute=> set-cookie: {key=value}. when browser receives that res, it stores the cookie. After that every time the browser sends req it attaches the cookie in the http header.

Drawbacks

Distributed system: there are multiple servers. In session based auth, we store data in the server database. As there are multiple servers in a distributed system, it is difficult to find out which server the user info is stored in. we can store the user data in all the servers but it is not a good idea.

Performance issue: it is a costly process because it needs storage in server. cookies is for web browser. This makes it less scalable and increases overhead when the site is having many users. we need to do extra work for mobile native platform which is hard to implement.

Cookie fraud: It is vulnerable to CSRF attack. Cannot set cookie for cross domain: It poses issues when APIs are served from a different domain to mobile and web devices

Token based authentication

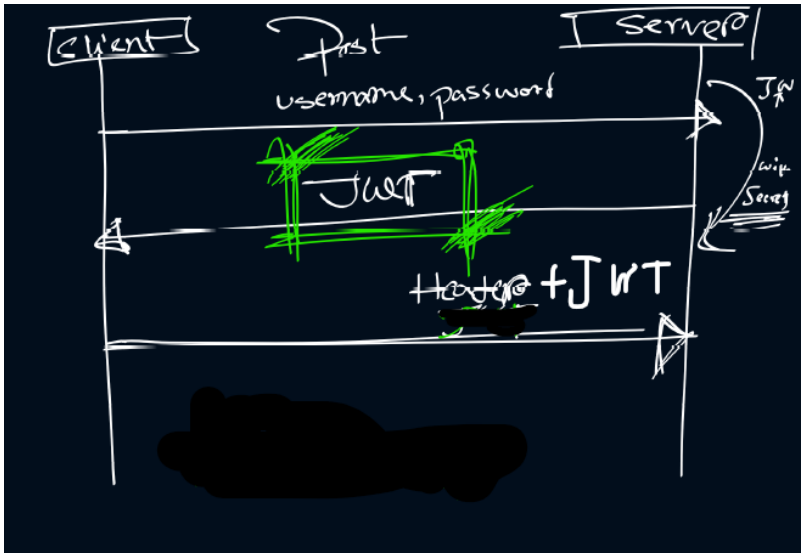
We encrypt JWT with RSA or HMAC-SHA256 algorithm.

Handshaking Protocol used to Generate Secret. That means to generate a private key using public keys. We then Encrypt the JWT with the Secret(private key).

We couldn't access Third Party Authentication to Login in Session Based if we used certain Flags. We can do it with Token Based Auth. For example, Gmail can generate A JWT with minimal info required which we can use to log in to Zoom.

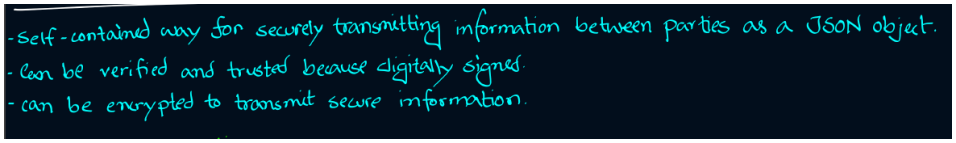
Steps to create a token based authentication between client and web server:

1. The user submits login credentials to the backend server through post request.
2. Upon the request, the server verifies the credentials before generating an encrypted JWT with a secret key and sends it back to the client.
3. On the client-side, the browser stores the token in the local storage.
4. On future requests, the JWT is added to the authorization header and the server will validate its signature by decoding the token before sending a response.
5. On the logout operation, the token on the client-side is destroyed without server interaction.



Here distributed system won't be a problem. Because all the user info, user role is stored in the jwt. If a server can decrypt it then it can accept the request.

JWT def:



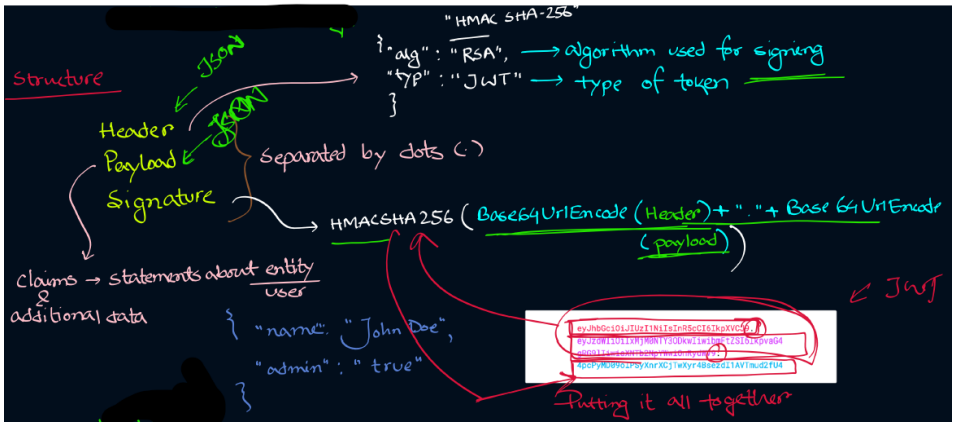
JWT with secret:

JWT encrypts the data & send it to client.

Both client and server have some public key, they are used to create a secret key. This secret key encrypts the JWT and send it to client. After that the client will send req (header+JWT) to server. If server has the key to decrypt then it can accept the request and proceed to send the response.

JWT is **stored in local storage** because of its size. If we can ensure that it won't cross 4kb then we can store it in cookie.

JWT structure:



Pros:

1) Scalability & statelessness: Since tokens are stateless and only need to be stored on the user's side, they're a more scalable solution. All the server needs to do is create and verify the tokens and the information itself never needs to be stored. That means that you can maintain more users on the website at once.
2) More secure: since the token itself stores no sensitive information. Instead, the token acts as a placeholder for the user's credentials. As the token travels between the server, web application, and the user's browser, the user's credentials are never at risk of being compromised.
3) Can be generated anywhere
4) Helpful in authorization: while JWT is being generated, we can define the user role. It helps to authorize people. Easy 3rd party authentication. 3rd party apps will just send the minimum amount of info to the server for user authentication. It will help to keep the user info confidential.

Cons:

Needs more bandwidth because it stores a lot of data.

https://www.section.io/engineering-education/cookie-vs-token-authentication/

Middleware

function (req,res,next)
middleware can- execute any code
- can change req/res or end its cycle
- can call any other middleware
- can redirect a user to a particular page based on the req

5 types:

1. application level middleware => app.use()[page not found] signature:req/res/next
2. router level middleware => router.use() based on the req on that route it redirects to somewhere else signature:req/res/next
3. error handling middleware => sends a particular response code or file. it has [req,res,next,err]
4. built in middleware => styling
   => bodyparser.json/ URL encoded it checks if it has any default json data. we can then use that data because we use built in middleware

5. 3rd party middleware => to track the response time, we can use 3rd party middleware(Morgan)

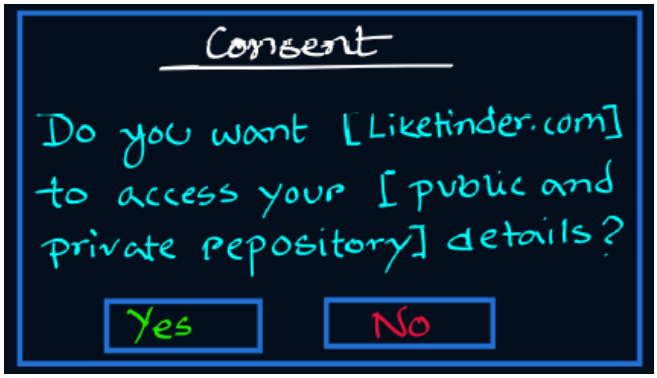
req->run middleware->res
static object (doesn't change)=> styling,image

oAuth v2.0

Not for authentication but for authorization.

Doesn’t check who i really am, just provides the data

Suppose there’s an app called MURGI. We can login to it using github. MURGI uses users git repo info. Here **resource owner** is the **user**. **Client** is **MURGI**. Github won’t provide the info without any authorization. the **authorization server** **(Github)** checks if MURGI has right to get the info. After checking, the auth server authorized MURGI. Then the **resource server** **(Github)** provides the resources of the resource owner(user). Not necessarily both servers will be separate entities. They can be one server providing both functionalities. Or Auth server maybe a 3rd party that resource server trusts.



Scope: Scope is a mechanism in OAuth 2.0 to limit an application's access to a user's account. An application can request one or more scopes, this information is then presented to the user in the consent screen, and the access token issued to the application will be limited to the scopes granted.
Consent: User **consent**. When you use **OAuth** 2.0 for authentication, your users are authenticated after they agree to terms that are presented to them on a user **consent** screen. Google verifies public applications that use **OAuth** 2.0 and meet one or more of the verification criteria.
Response type: auth code. What type of flow we are using. 4 types of flow:

1. auth code flow,
2. implicit flow,

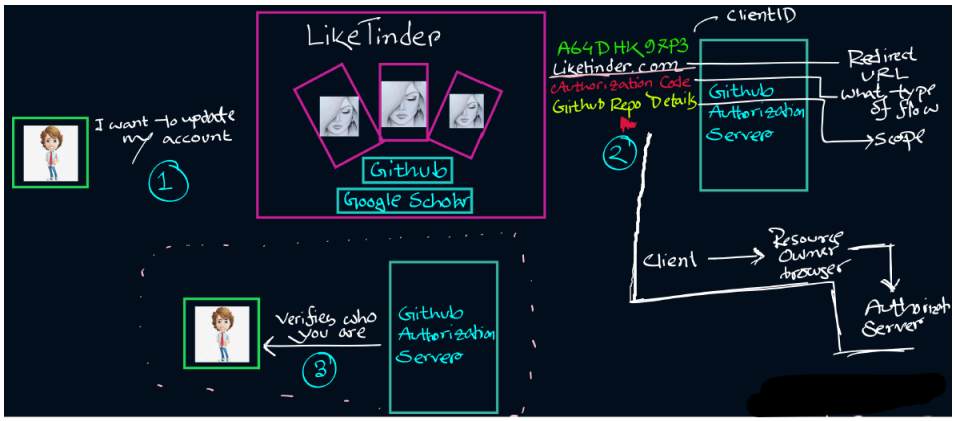
3. resource owner password credential flow,
4. client credential flow.

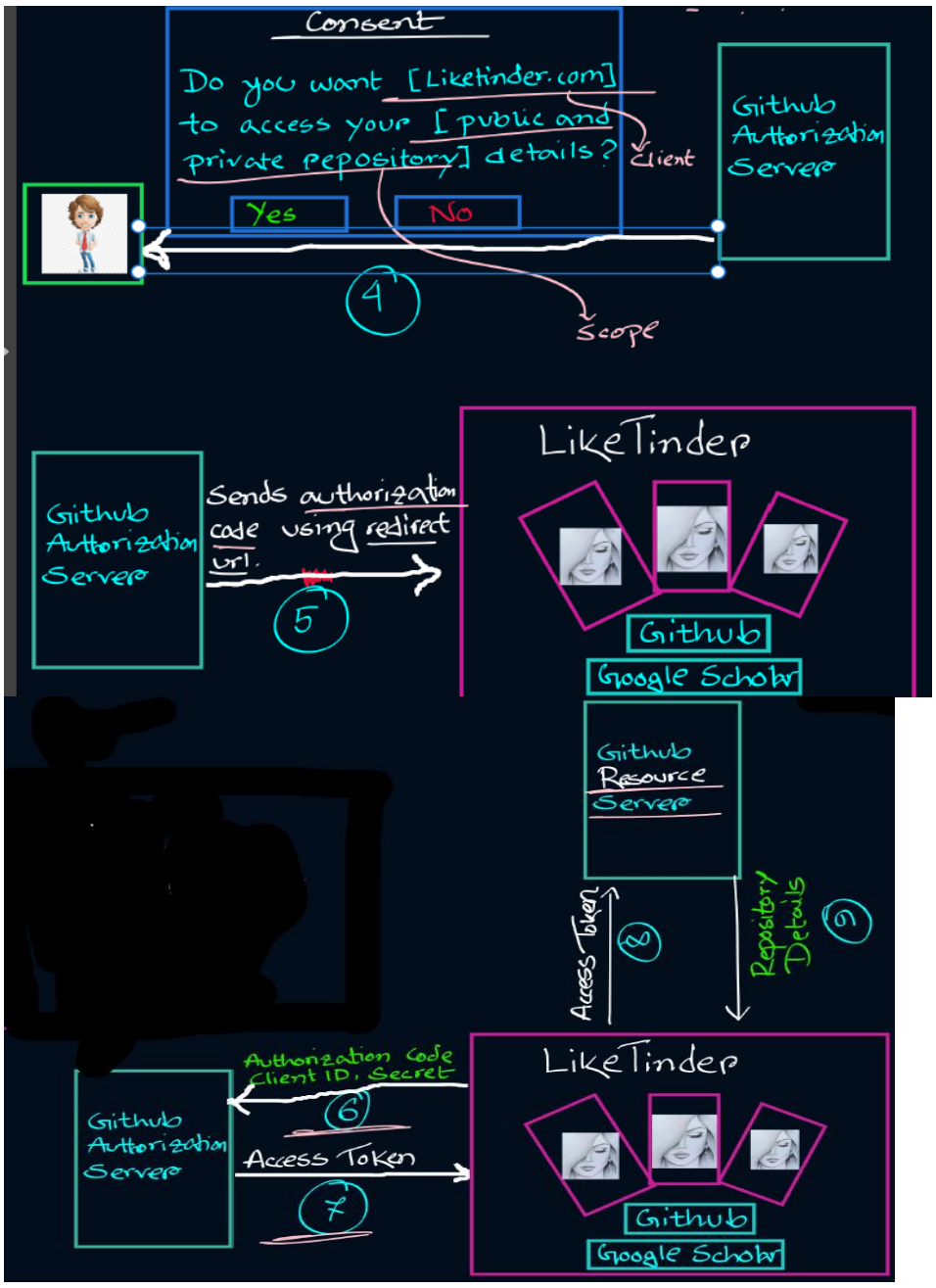
Redirect url: server decides who to send the auth code. After a user successfully authorizes an application, the authorization server will redirect the user back to the application with either an authorization code or access token in the URL. Because the redirect URL will contain sensitive information, it is critical that the service doesn't redirect the user to arbitrary locations.

Phishing attack: Phishing is a type of attack that is used to steal user data, including login credentials and credit card numbers. Eavesdropper may be impersonated as a trusted entity, dupes a victim into opening an email, instant message, or text message. The recipient is then tricked into clicking a malicious link, which can lead to the leak of personal information. Client secret: only client and auth server knows. It allows to securely share info between them. Prevents from phishing attack.

Auth code flow:

Most common and used flow. Others have some serious vulnerabilities.




Consent
Do you want [Liketinder.com] to access your [public and private repository] details?
Client
Github Authorization Server
Yes
No
4
Scope
Github Authorization Server
Sends authorization code using redirect url.
5
LikeTinder
Github
Google Scholar
Github Resource Server
Access Token
8
Repository Details
9
Authorization Code Client ID, Secret
6
Github Authorization Server
Access Token
7
LikeTinder
Github
Google Scholar

Without client secret, no other 3rd party(eavesdropper) can access to github. Here browser won't be involved. Only in step 2,5 browser will be involved. In step 6,7 browser won't be involved. Github will directly communicate with liketinder.

implicit flow

It sends the secret key in step 2 through browser. Then in step 5, auth server sends the auth code. No need for step 6,7. Here the browser knows the secret key.

Cons: needs to have trusted browser. Else someone may inject malicious scripts into the browser. For large scale app it is not preferable,as there are a lot of browsers. So it is not possible to secure the app.

resource owner password credential flow,

Users provide github credentials to the client (app). Then client directly communicate with the auth server and send the auth request along with the users git credentials to the auth server. Auth server verifies it & then give auth code.

Cons: vulnerable- since we give git credentials to the app. So there's a possibility that the app will sell the info or if the app server is compromised, then all of the users info will be leaked. Use this flow only if the app is trusted.

client credential flow

No need for resource owner, consent. Client communicates with the auth server directly and get the public info (no private/sensitive info is shared). Example: github public repo. Anyone can access that.

Why need oAuth v2.0?

- It prevents phishing attack by using secret key that only the client and auth server knows.
- When there's no oAuth, the app used to provide a login page for GitHub. Users would login there. Then client will get the info and

communicate with auth server and perform the authentication. Here client knows the git credentials of the user, So there's a possibility that the app will sell the info or if the app server is compromised, then all of the users info will be leaked.

- Resource owner has complete power over what info client has access to. Before client could access all the info, now resource owner can control it
- Without sharing the credentials, we can perform authorization.

Differences Between OAuth 1 and 2

1. Where OAuth 2.0 defines four roles, (client, authorization server, resource server, and resource owner,) OAuth 1 uses a different set of terms for these roles. The OAuth 2.0 "client" is known as the "consumer," the "resource owner" is known simply as the "user," and the "resource server" is known as the "service provider".
2. OAuth 1 also does not explicitly separate the roles of resource server and authorization server. OAuth 2.0 does.
3. OAuth 2.0 is a complete rewrite of OAuth 1.0 from the ground up, sharing only overall goals and general user experience. OAuth 2.0 is not backwards compatible with OAuth 1.0 or 1.1, and should be thought of as a completely new protocol. If an app used oAuth OAuth 2.0, it can't be replace with v1. And vice versa.
4. V1 and OAuth 2.0 are different in security point of view. V1 is more secure. The security is provided by encoding everything (all the messages) with a signature or key. It hinders the performance of the app. OAuth 2.0 doesn't provide this sort of security. It doesn't encrypt the messages, it simply sends them as plain text. But OAuth 2.0 needs TLS, SSL certificate. Without it, OAuth 2.0 can't be used. We can do all the msg passing only if we use HTTPS, because https already encrypts the messages.