CTIS 256 Web Technologies II

Notes # 11

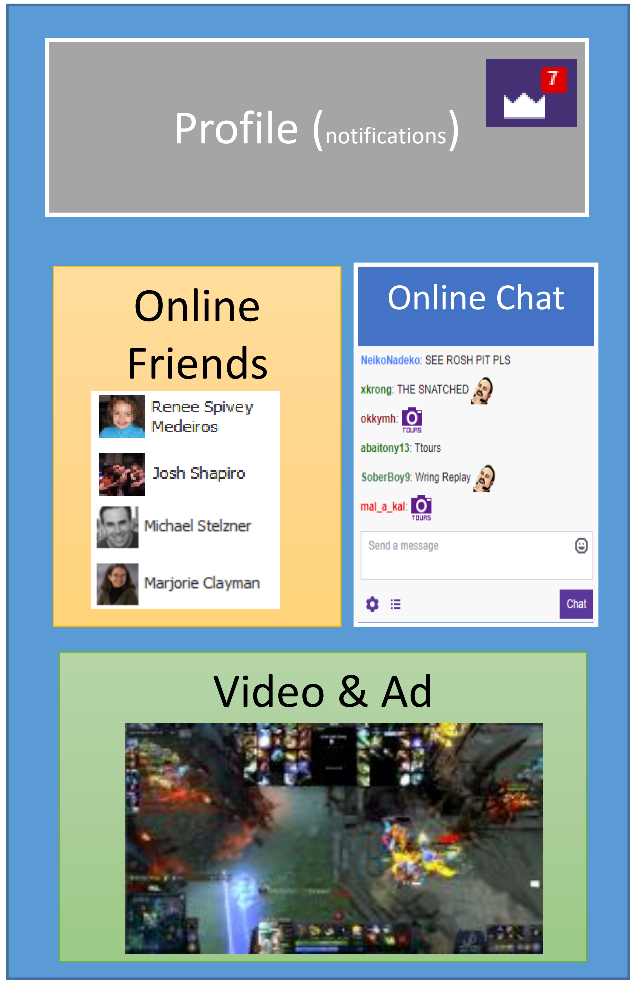
AJAX (Asynchronous JavaScript And XML)

Serkan GENÇ

Why do we need AJAX?

1. A page can be considered as a container for many independent applications.
   - If you start a page or refresh a page, you restart all applications from the beginning. You lose the states of the applications.
   - Clicking a link, or posting a form start a new page.
2. The content of the page is fixed and can not change after the page is displayed on the browser.
   - To retrieve NEW data from the server, we need to refresh the page, and all states of the applications are lost.

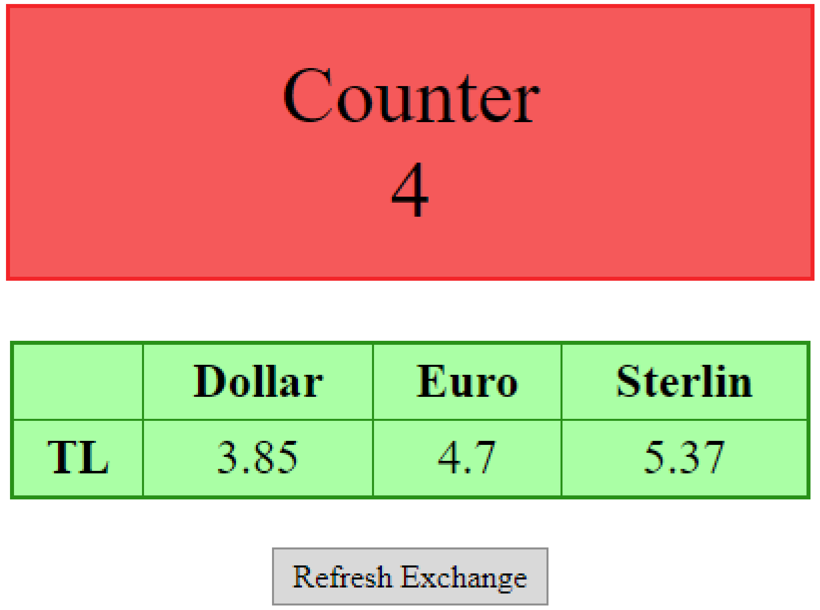
Solution : AJAX



NeikoNadeko: SEE ROSH PIT PLS
xkrong: THE SNATCHED
okkymh: TOURS
abaitony13: Ttours
SoberBoy9: Wring Replay
mal_a_kal: TOURS

Example without AJAX

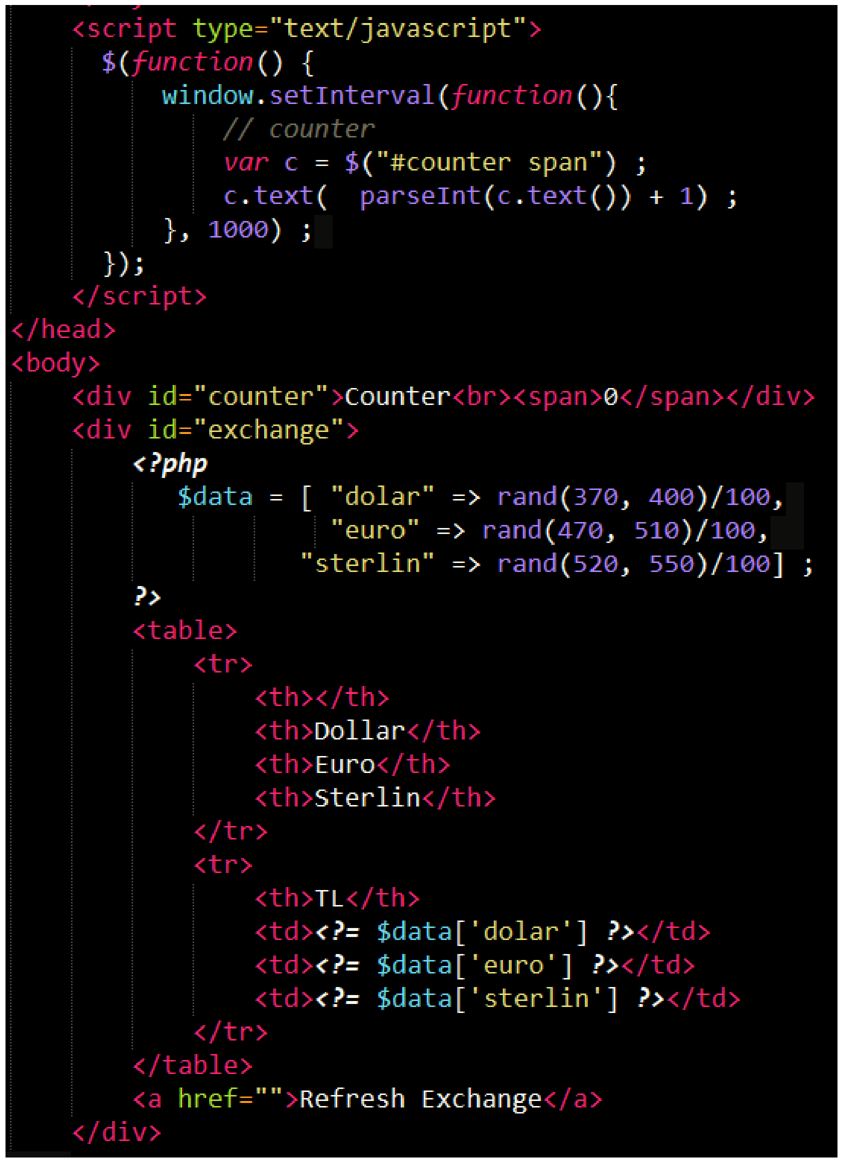
Two applications in the same page; counter and exchange rate



Counter
4

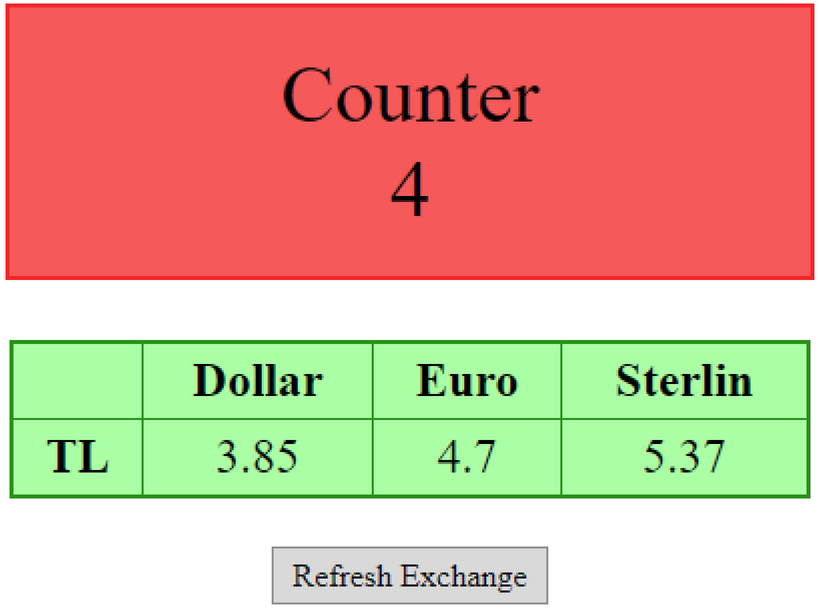
	Dollar	Euro	Sterlin
TL	3.85	4.7	5.37

Clicking button refreshes the whole page to retrieve new data.



```
    <script type="text/javascript">
     $(function() {
         window.setInterval(function(){
             // counter
             var c = $("#counter span") ;
             c.text(  parseInt(c.text()) + 1) ;
         }, 1000) ;
     });
    </script>
</head>
<body>
    <div id="counter">Counter<br><span>0</span></div>
    <div id="exchange">
        <?php
           $data = [ "dolar" => rand(370, 400)/100,
                     "euro" => rand(470, 510)/100,
                   "sterlin" => rand(520, 550)/100] ;
        ?>
        <table>
            <tr>
                <th></th>
                <th>Dollar</th>
                <th>Euro</th>
                <th>Sterlin</th>
            </tr>
            <tr>
                <th>TL</th>
                <td><?= $data['dolar'] ?></td>
                <td><?= $data['euro'] ?></td>
                <td><?= $data['sterlin'] ?></td>
            </tr>
        </table>
        <a href="">Refresh Exchange</a>
    </div>
```

Example with AJAX



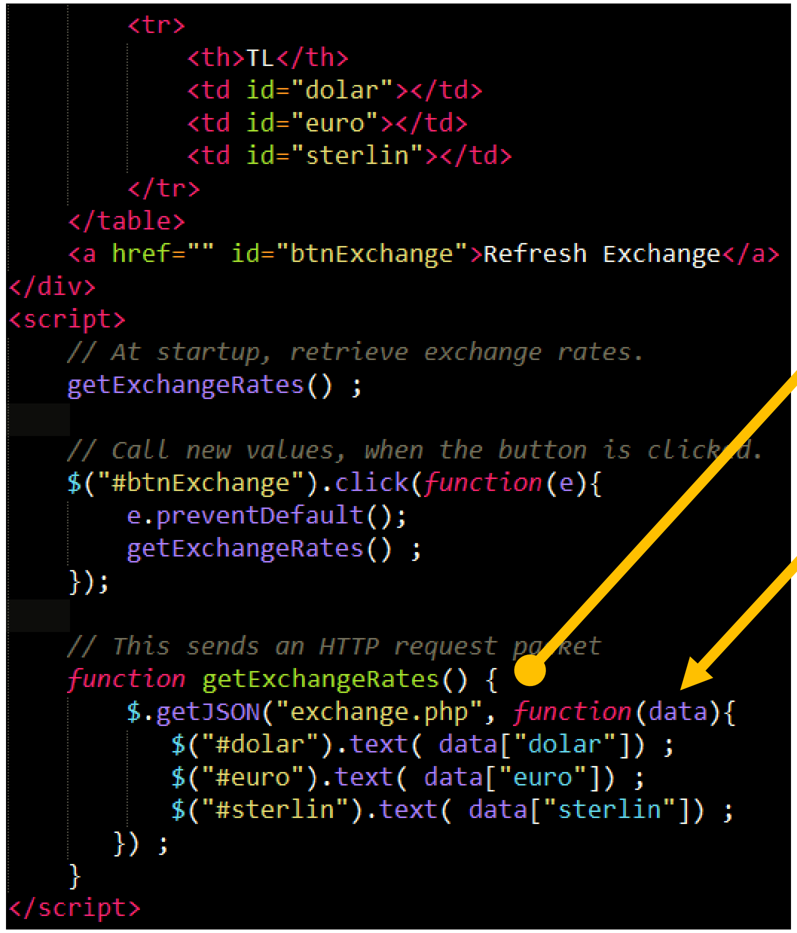
Counter
4

	Dollar	Euro	Sterlin
TL	3.85	4.7	5.37

Refresh Exchange

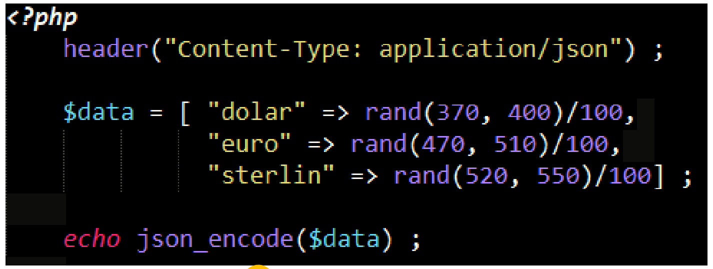
Clicking button refreshes the whole page

Main.php



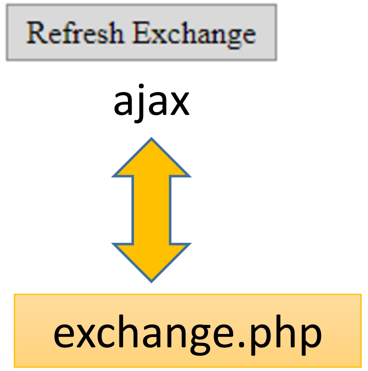
```
        <tr>
            <th>TL</th>
            <td id="dolar"></td>
            <td id="euro"></td>
            <td id="sterlin"></td>
        </tr>
    </table>
    <a href="" id="btnExchange">Refresh Exchange</a>
</div>
<script>
    // At startup, retrieve exchange rates.
    getExchangeRates() ;

    // Call new values, when the button is clicked.
    $("#btnExchange").click(function(e){
        e.preventDefault();
        getExchangeRates() ;
    });

    // This sends an HTTP request packet
    function getExchangeRates() {
        $.getJSON("exchange.php", function(data){
            $("#dolar").text( data["dolar"]) ;
            $("#euro").text( data["euro"]) ;
            $("#sterlin").text( data["sterlin"]) ;
        }) ;
    }
</script>
```

Exchange.php



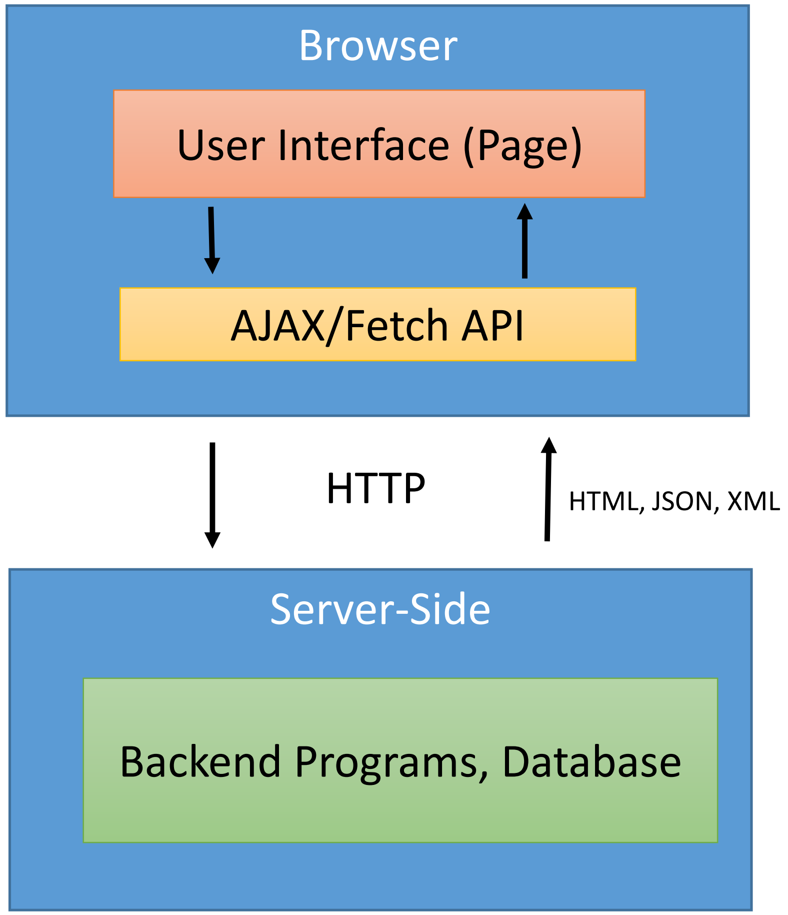
```
<?php
    header("Content-Type: application/json") ;

    $data = [ "dolar" => rand(370, 400)/100,
              "euro" => rand(470, 510)/100,
              "sterlin" => rand(520, 550)/100] ;

    echo json_encode($data) ;
```

```
{dolar: 3.79, euro: 5.06, sterlin: 5.45}
```



What is AJAX?

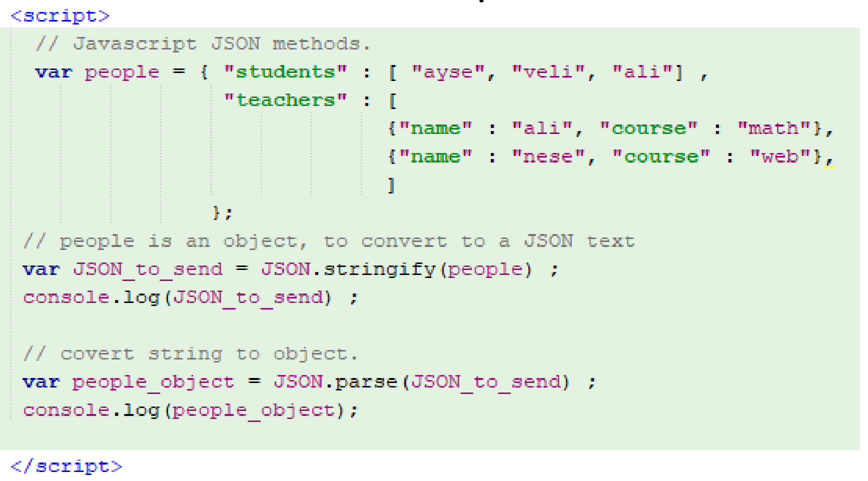
- AJAX is a JavaScript technology to request a URL (server-side resource) *without refreshing/reloading* the current page.
- After the page loaded, we can *send to and receive data* from server side programs in the background.
- It is a JavaScript object (**XMLHttpRequest**, XHR) that sends a HTML request (asynchronously), and gets the results (data), and then modifies the page accordingly. **Fetch API** is an alternative to XHR, and can be used for http request.
- Browsers have their own way to implement AJAX object, this is why jQuery is an appropriate library for AJAX tasks. It supports browser compatibility.
- With AJAX, we can simulate or turn Web-based application to Desktop-based application. This allows us to write single page application.



JSON

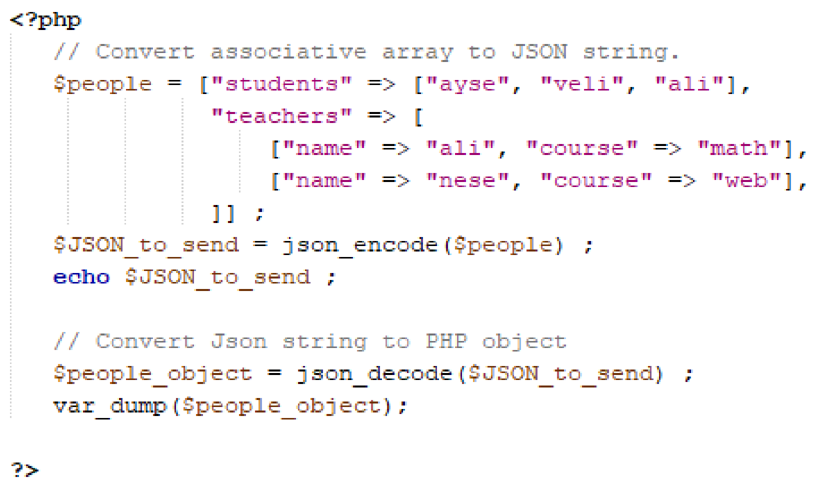
- JSON is an acronym for **J**ava**S**cript **O**bject **N**otation.
- JSON is a data exchange format between client (browser) and server (php program).
- It is a text format, readable, simple, and language neutral.
- **JSON.stringify()** builtin javascript method to convert from object to string format.
- **JSON.parse()** converts Json text to javascript object.
- In PHP, **json_decode()** to parse json string to PHP object, and **json_encode()** to convert from associative array to json string.

Javascript



```
<script>
  // Javascript JSON methods.
  var people = { "students" : [ "ayse", "veli", "ali"] ,
                 "teachers" : [
                              {"name" : "ali", "course" : "math"},
                              {"name" : "nese", "course" : "web"},
                              ]
               };
 // people is an object, to convert to a JSON text
 var JSON_to_send = JSON.stringify(people) ;
 console.log(JSON_to_send) ;

 // covert string to object.
 var people_object = JSON.parse(JSON_to_send) ;
 console.log(people_object);

</script>
```

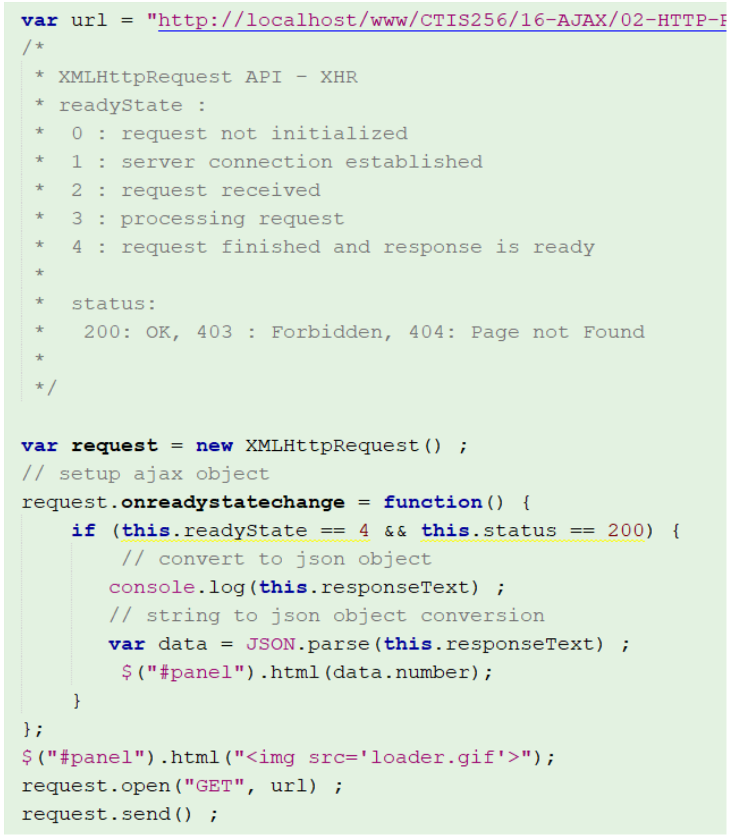
PHP



```
<?php
   // Convert associative array to JSON string.
   $people = ["students" => ["ayse", "veli", "ali"],
              "teachers" => [
                  ["name" => "ali", "course" => "math"],
                  ["name" => "nese", "course" => "web"],
              ]] ;
   $JSON_to_send = json_encode($people) ;
   echo $JSON_to_send ;

   // Convert Json string to PHP object
   $people_object = json_decode($JSON_to_send) ;
   var_dump($people_object);

?>
```

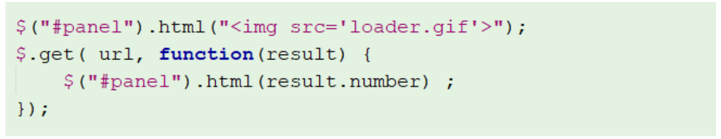
How to use AJAX?

XMLHttpRequest API



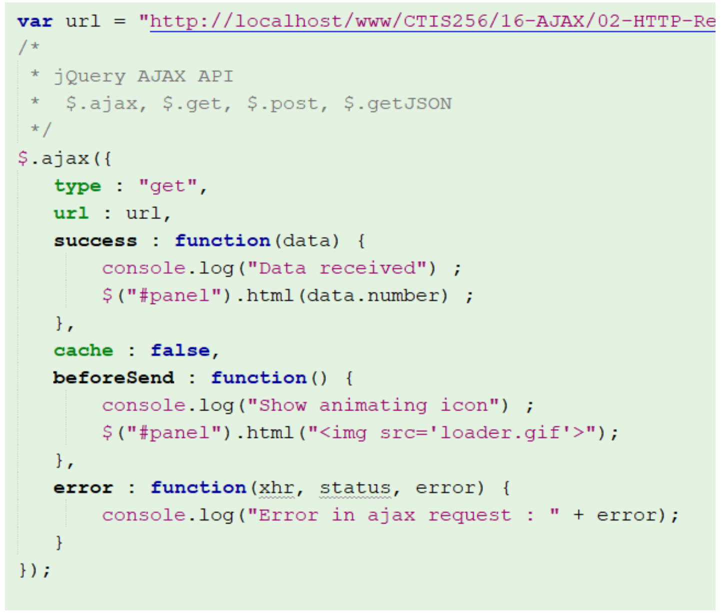
```
var url = "http://localhost/www/CTIS256/16-AJAX/02-HTTP-F
/*
 * XMLHttpRequest API - XHR
 * readyState :
 *  0 : request not initialized
 *  1 : server connection established
 *  2 : request received
 *  3 : processing request
 *  4 : request finished and response is ready
 *
 *  status:
 *   200: OK, 403 : Forbidden, 404: Page not Found
 *
 */

var request = new XMLHttpRequest() ;
// setup ajax object
request.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
        // convert to json object
       console.log(this.responseText) ;
       // string to json object conversion
       var data = JSON.parse(this.responseText) ;
        $("#panel").html(data.number);
    }
};
$("#panel").html("<img src='loader.gif'>");
request.open("GET", url) ;
request.send() ;
```

jQuery AJAX API



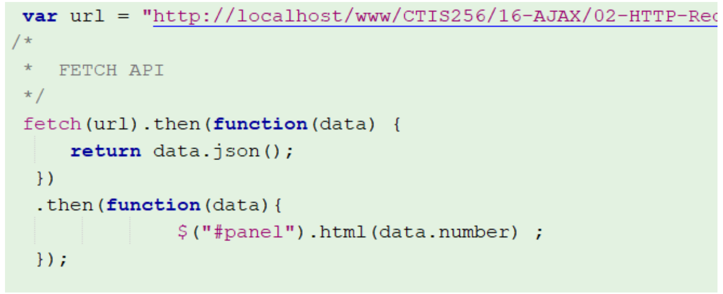
```
$("#panel").html("<img src='loader.gif'>");
$.get( url, function(result) {
    $("#panel").html(result.number) ;
});
```

Verbose Mode



```
var url = "http://localhost/www/CTIS256/16-AJAX/02-HTTP-Re
/*
 * jQuery AJAX API
 *  $.ajax, $.get, $.post, $.getJSON
 */
$.ajax({
   type : "get",
   url : url,
   success : function(data) {
       console.log("Data received") ;
       $("#panel").html(data.number) ;
   },
   cache : false,
   beforeSend : function() {
       console.log("Show animating icon") ;
       $("#panel").html("<img src='loader.gif'>");
   },
   error : function(xhr, status, error) {
       console.log("Error in ajax request : " + error);
   }
});
```

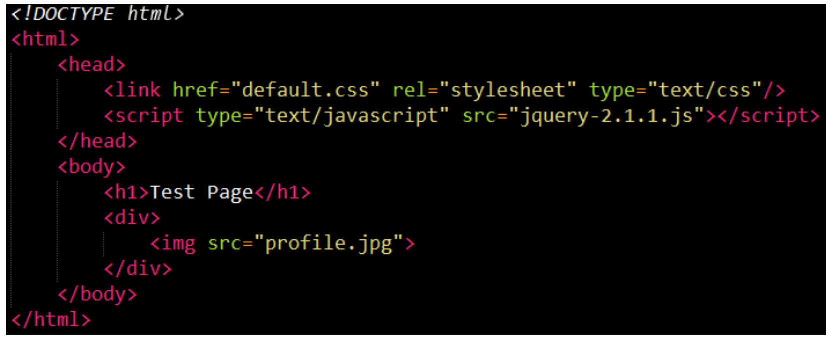
Fetch API



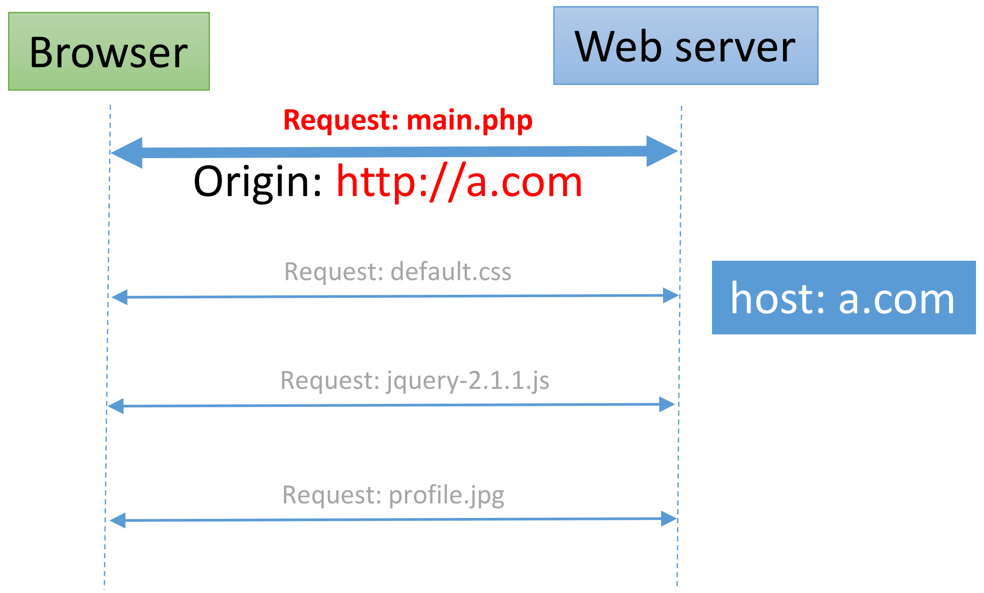
```
var url = "http://localhost/www/CTIS256/16-AJAX/02-HTTP-Rec
/*
 *  FETCH API
 */
fetch(url).then(function(data) {
    return data.json();
 })
 .then(function(data){
           $("#panel").html(data.number) ;
 });
```

Same/Cross Origin

main.php

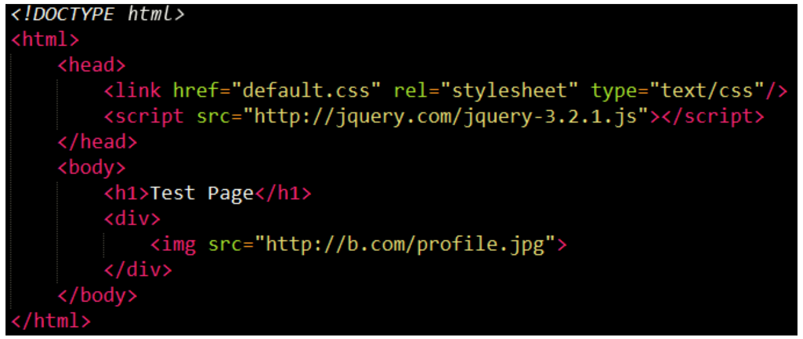


```
<!DOCTYPE html>
<html>
    <head>
        <link href="default.css" rel="stylesheet" type="text/css"/>
        <script type="text/javascript" src="jquery-2.1.1.js"></script>
    </head>
    <body>
        <h1>Test Page</h1>
        <div>
            <img src="profile.jpg">
        </div>
    </body>
</html>
```

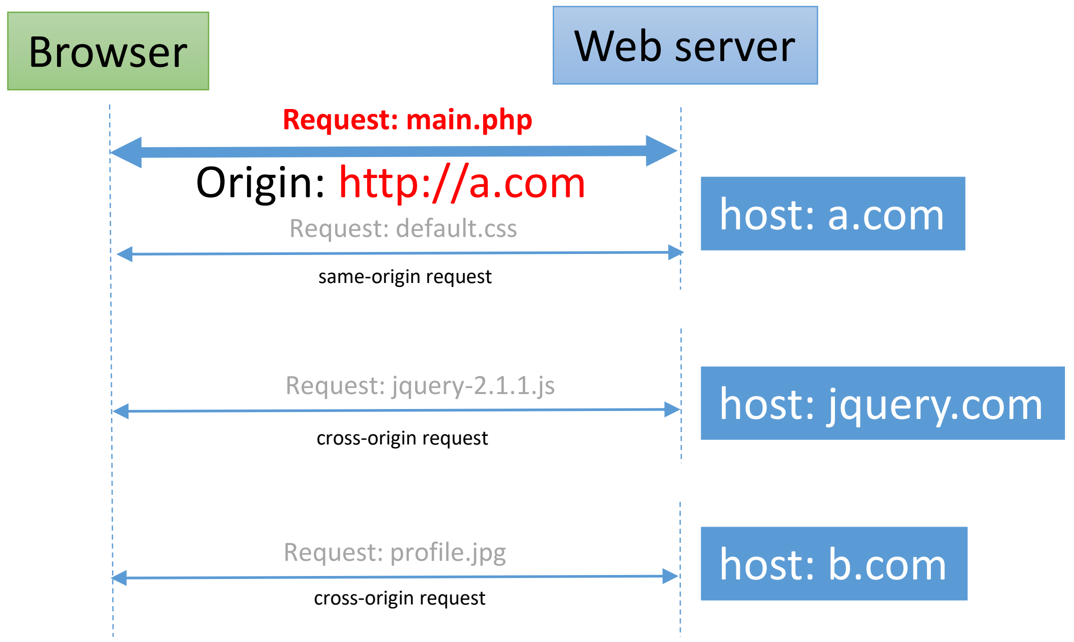


All files are in the same **origin** (host, protocol, port #)

main.php



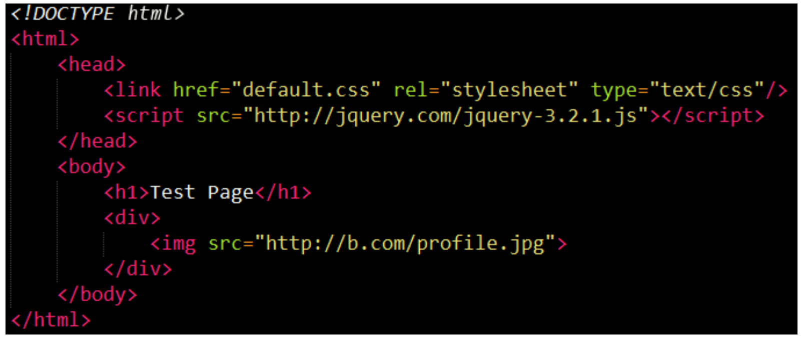
```
<!DOCTYPE html>
<html>
    <head>
        <link href="default.css" rel="stylesheet" type="text/css"/>
        <script src="http://jquery.com/jquery-3.2.1.js"></script>
    </head>
    <body>
        <h1>Test Page</h1>
        <div>
            <img src="http://b.com/profile.jpg">
        </div>
    </body>
</html>
```



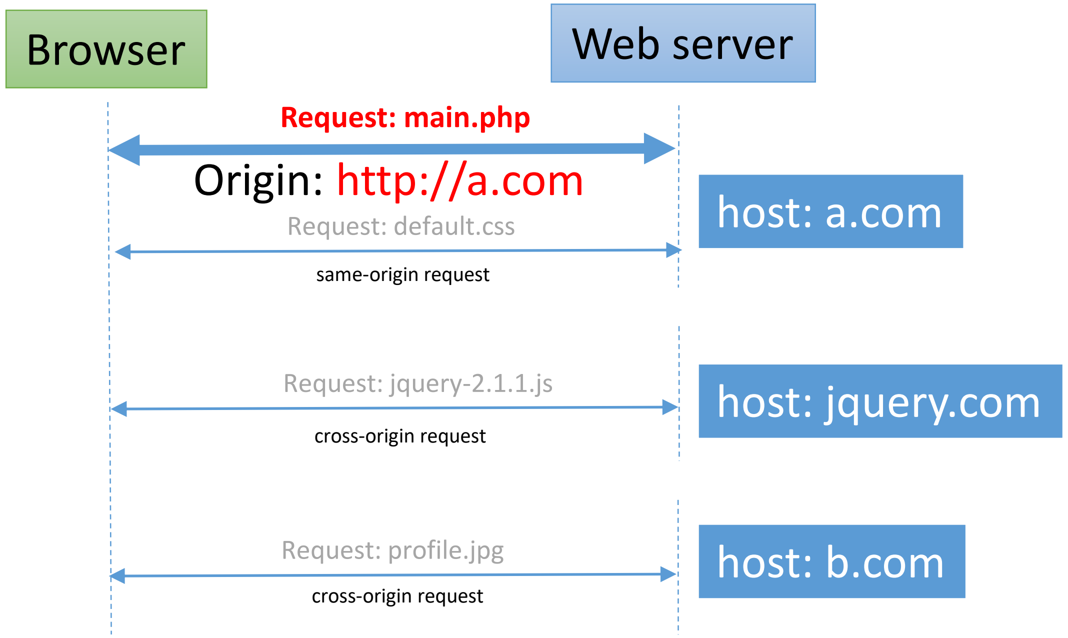
Origin is the tuple of **protocol/host/port** of the first transaction. In the example above, the request to main.php is the first transaction. The browser sends a request to "a.com" with "http" protocol at port "80". Therefore, origin is "http://a.com"

Same/Cross Origin

main.php



```html
<!DOCTYPE html>
<html>
    <head>
        <link href="default.css" rel="stylesheet" type="text/css"/>
        <script src="http://jquery.com/jquery-3.2.1.js"></script>
    </head>
    <body>
        <h1>Test Page</h1>
        <div>
            <img src="http://b.com/profile.jpg">
        </div>
    </body>
</html>
```



In html tags' attributes, one can safely use any url addresses whether they result in same-origin or cross-origin requests.

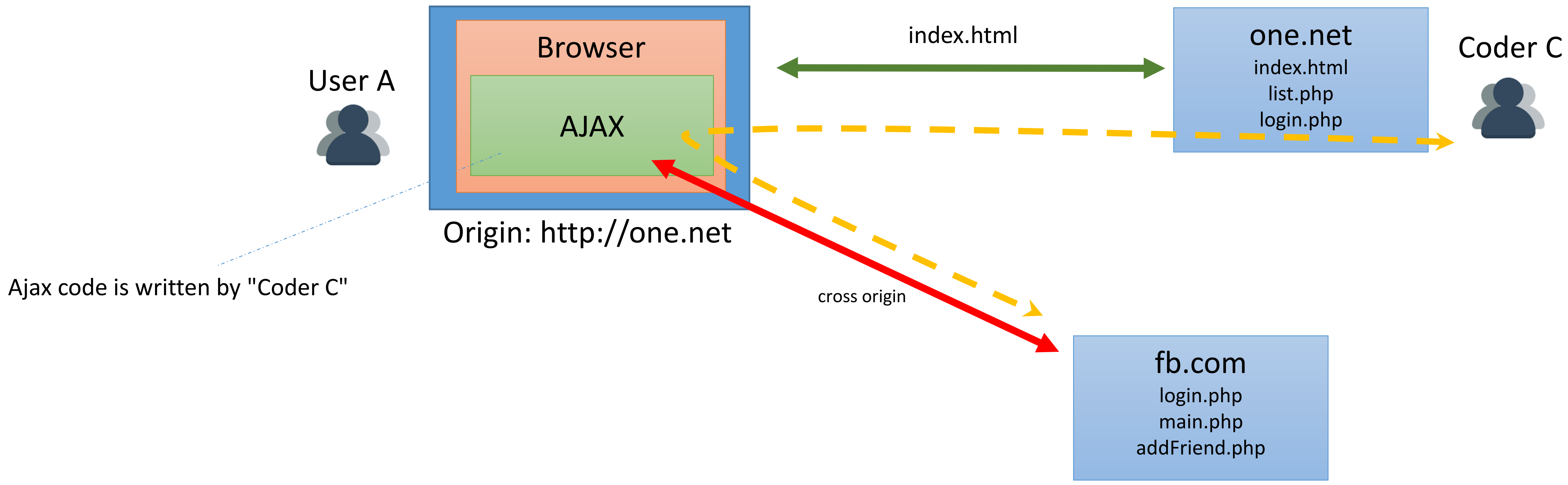
cross-origin requests are allowed

<a href="http://otherdomain.com">

<img src="http://flowers.com/daisy.jpg" />

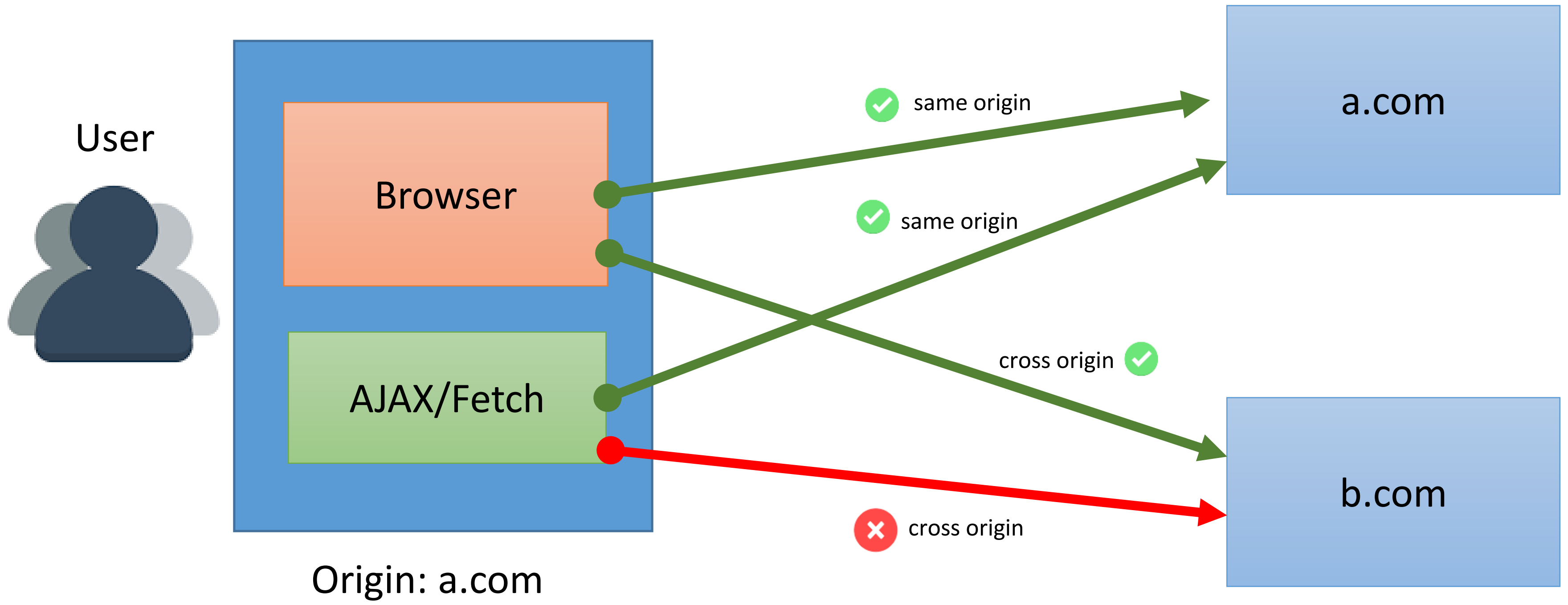
<script src="http://jslibs.com/faces.js">

Same Origin Policy (SOP)



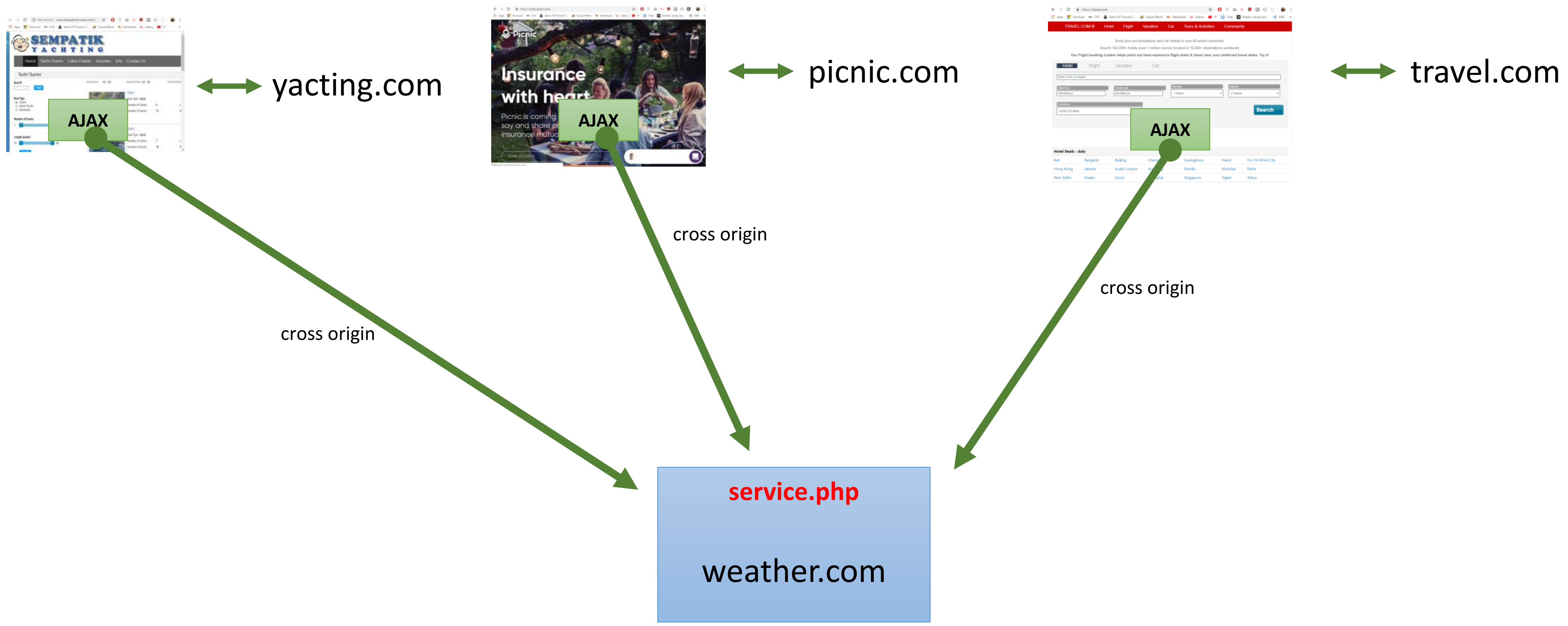
If Ajax script in "index.html" written by "Coder C" sends cross origin requests to "fb.com", it can use "User A's" account without the knowledge of "User A". In other words, "Coder C" can send HTTP requests on behalf of "User A" through AJAX. This is a security problem and therefore, cross origin requests from Ajax are limited by browsers.
Normally, an Ajax object can send HTTP requests only to the same origin. This is called "**Same Origin Policy**".

AJAX and SOP



Since browsers (user agent) are trusted, they are allowed to send cross-origin http requests.
(<img>, <script>, <link> are managed by the browsers)
However, JavaScript (AJAX) is not trusted and can not access cross sites. AJAX cannot load any resources such as image, html, json, script, css file, xml from cross site domain. Otherwise, it can log in a cross site, in the name of the user and access private user data. However, some domains may let AJAX accesses with CORS.

Cross Origin Resource Sharing (CORS)



Applications may want to access other applications in different domain. Assume "weather.com" wants to sell weather information to others. But How? Do not forget "Same Origin Policy".
If the server-side tells the browser that "it is ok, I accept cross-origin requests", then browser allows AJAX to send cross-origin request. This is called "Cross Origin Resource Sharing or CORS".

Cross-Origin Resource Sharing (CORS)

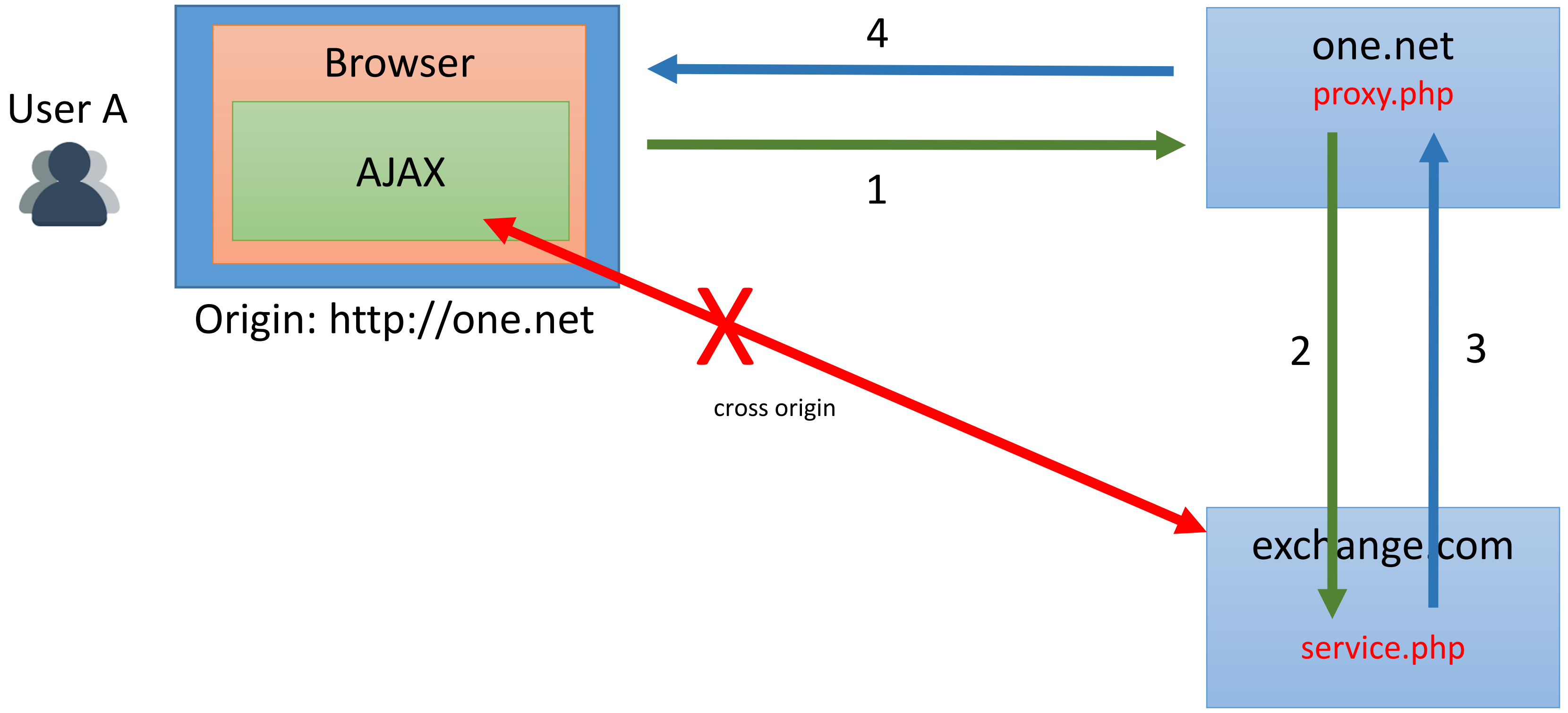
- Cross-Origin Resource Sharing (CORS) is a mechanism that uses additional HTTP headers to let a user agent gain permission to access selected resources from a server on a different origin (domain) than the site currently in use. A user agent makes a **cross-origin HTTP request** when it requests a resource from a different domain, protocol, or port than the one from which the current document originated.
- For security reasons, browsers restrict cross-origin HTTP requests initiated from scripts. XMLHttpRequest and Fetch API follow the same-origin policy. This means that a web application using AJAX API can only request HTTP resources from the same domain unless CORS headers are used by the application in the server-side.

```
<?php
header("Access-Control-Allow-Origin: *");
```

or you can set Web Server config file httpd.conf for Apache Web Server.

- HTTP Response Header:
  - Access-Control-Allow-Origin: <origin> | *
    - Access-Control-Allow-Origin: http://mozilla.org
    - Access-Control-Allow-Origin: *
  - Access-Control-Allow-Methods: <method>[, <method>]*
    - Access-Control-Allow-Methods: POST, GET, OPTIONS
  - Access-Control-Allow-Headers: <field-name>[, <field-name>]*
    - Access-Control-Allow-Headers: X-PINGOTHER, Content-Type
- The HTTP request headers
  - Origin: <origin>
    - Origin: http://foo.example
  - Access-Control-Request-Method: <method>
    - Access-Control-Request-Method: POST
  - Access-Control-Allow-Headers: <field-name>[, <field-name>]*
    - Access-Control-Allow-Headers: X-PINGOTHER, Content-Type

Proxy



if the server (exchange.com) does not allow CORS, you need to use a proxy. "proxy.php" can access "service.php" because it belongs to "one.net". The request is coming from "one.net" server. Therefore, the request is safe.

jQuery AJAX API

```
$("div:first").load("text.html");
```

```
 $.getJSON("exchange.php", function(data){
    $("#dolar").text( data["dolar"]) ;
    $("#euro").text( data["euro"]) ;
    $("#sterlin").text( data["sterlin"]) ;
}) ;
```

```
$.post( "add.php",
        { num1 : n1 , num2 : n2 },
        function ( data, status ) {
            $("div#sumResult").html(data.sum)   ;
        },
        "json") ;
```

```
$.get( "random.php", function(data){
    $("#rnd").append( ' ' + data);
}, 'html');
```

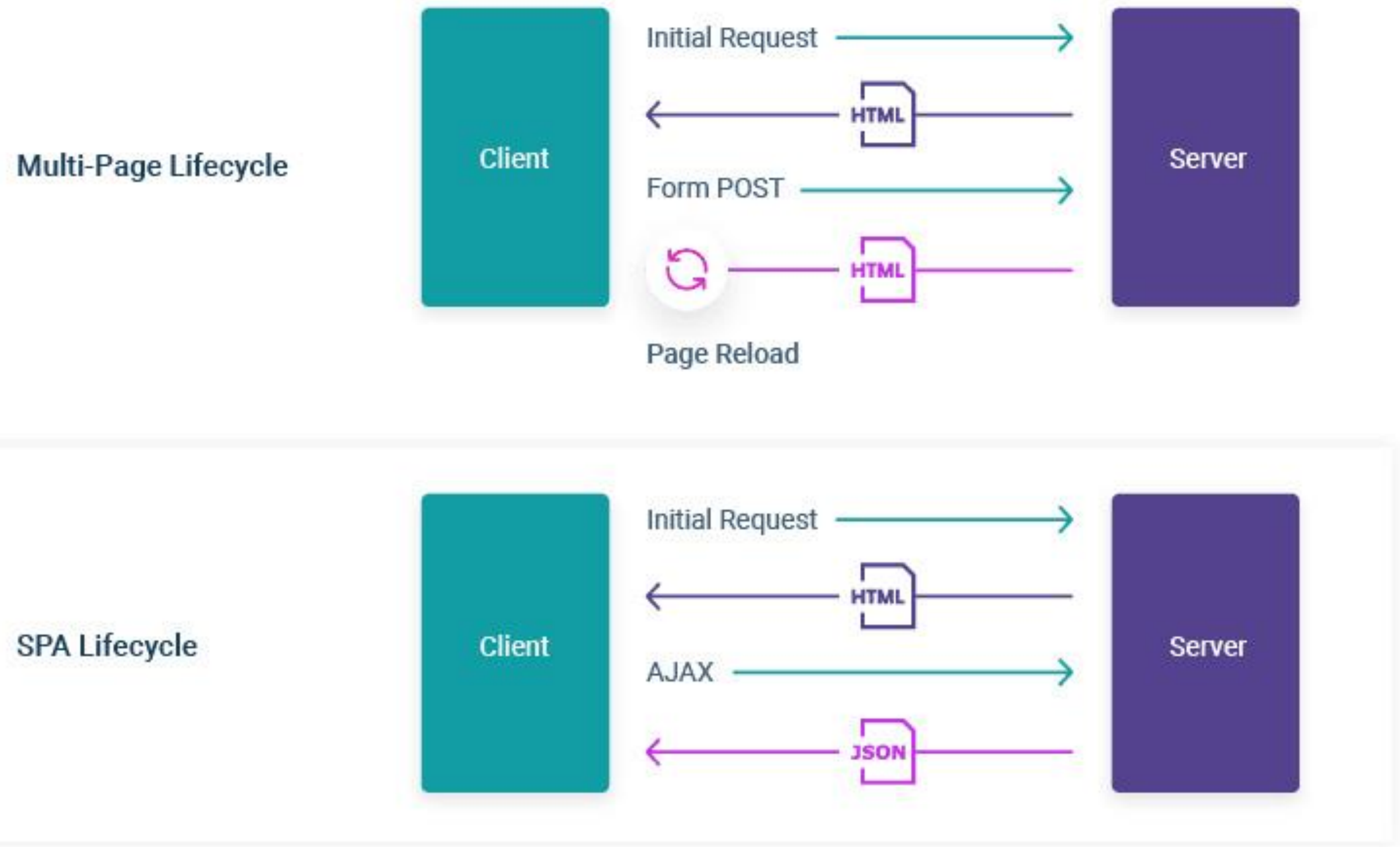
General AJAX Request

```
$.ajax({
  "type" : "post" or "get",
  "url" : "script.php" in the same domain,
  "data" : { name : "serkan" , "surname" : "genç" },
  "success" : function( data ) { } ,
  "dataType" : json/xml/html/jsonp/text
  "cache" : true/false,
  "beforeSend" : function() {}
  "error" : function(xhr, status, error) {}
  "async" : true/false
  });
```

Single Page Application (SPA)

- **MPA** (Multi Page Application) loads the entire page when you click on a link or a submit button. Server-side application (php) prepares all html codes.
- **SPA** is a web application architecture that interacts with the user by dynamically rewriting the current web page.
- Usually, there is one html page that interacts with the server-side.
- It prevents reloading the entire page.
- It retrieves new data from the server-side through AJAX and updates (renders) the page accordingly.
- Rendering the page is in the client-side. (Javascript prepares html codes)

MPA vs SPA


Multi-Page Lifecycle
Client
Initial Request
HTML
Form POST
HTML
Page Reload
Server
SPA Lifecycle
Client
Initial Request
HTML
AJAX
JSON
Server

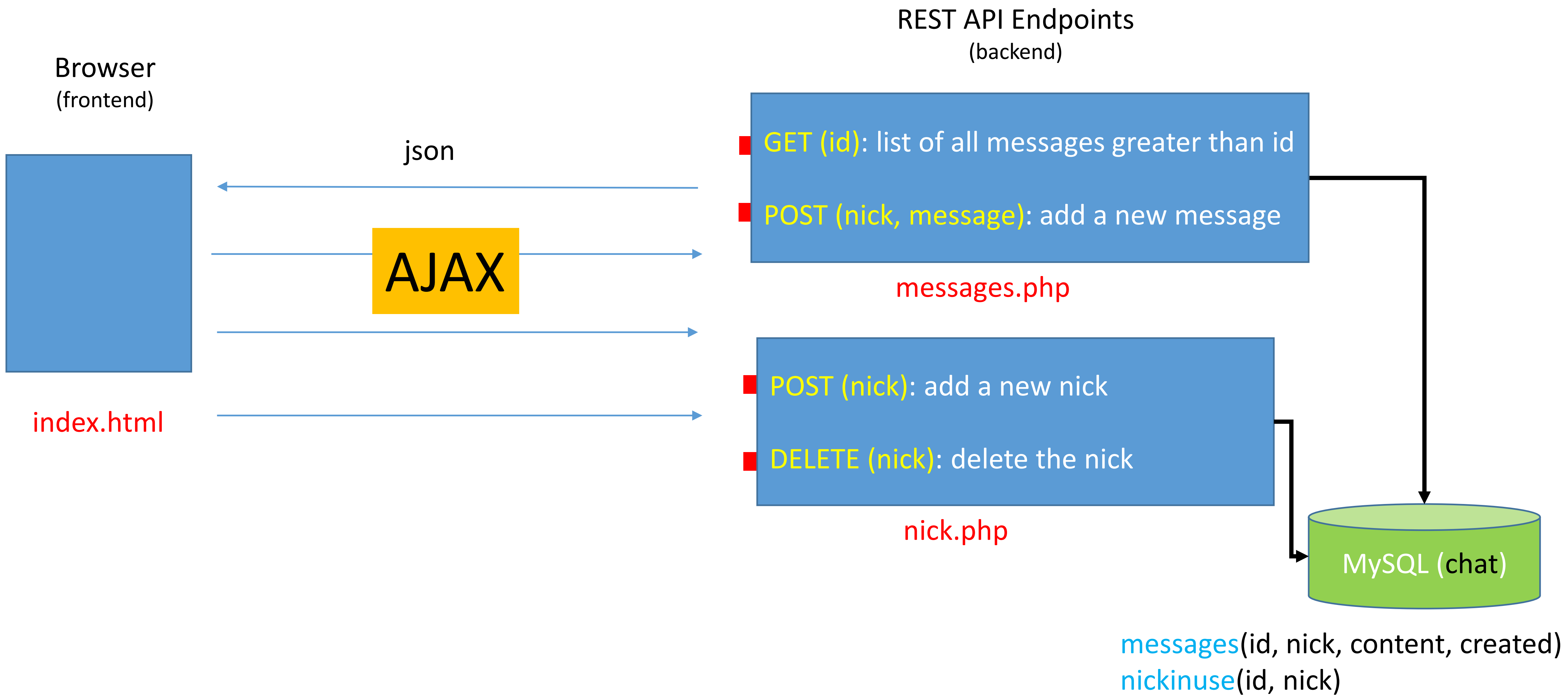
Single Page Application (SPA)

- **Advantages**:
  - Better UX (user experience), and faster response
  - Reduce server loads
  - Reuse the same backend code for web and mobile applications
  - Efficient Caching (no need to retrieve the data already requested before)
  - Coupling (Frontend and Backend separated)
- **Disadvantages**:
  - Significant load on the browser
  - Harder to code (memory leaks)
  - Javascript must be enabled.
  - Security (privacy of sensitive data, and prone to hack attacks esp. XSS)
  - Not SEO friendly (Search Engine Optimization)
    - one URL for the entire web application
    - not good for e-commerce apps, marketplaces, business catalogs
    - good for social network apps
- There are frontend frameworks for SPA such as *React*, *Angular* and *Vue* to make development easier (without any framework, it is possible to develop SPA)

REST Web Service

- **REST:** HTTP-based, lightweight, cacheable, scalable, stateless communication architecture btw client and server to exchange data.
- **Resource**: a URL specifies a resource (object)
  - http://www.one.com/api/messages (pretty URL – implementation independent)
  - http://www.one.com/api/nick.php
- **Request Verbs**:
  - CRUD = HTTP Verbs : (**C**:POST, **R**:GET, **U**:PUT/PATCH, **D**:DELETE)
- **Request Body**: Data sent with the request
- **Response Body**: body of the response, usually in **json** (xml, html)
- **Response Status Codes**: state of the request. (200: OK, 404: Not found)
- SOAP is a protocol for data exchange, based on XML, which is a complex protocol.

SPA – Chat System


REST API Endpoints
(backend)
Browser
(frontend)
json
AJAX
index.html
GET (id): list of all messages greater than id
POST (nick, message): add a new message
messages.php
POST (nick): add a new nick
DELETE (nick): delete the nick
nick.php
MySQL (chat)
messages(id, nick, content, created)
nickinuse(id, nick)