Flexible Communication Client-Server with Ajax

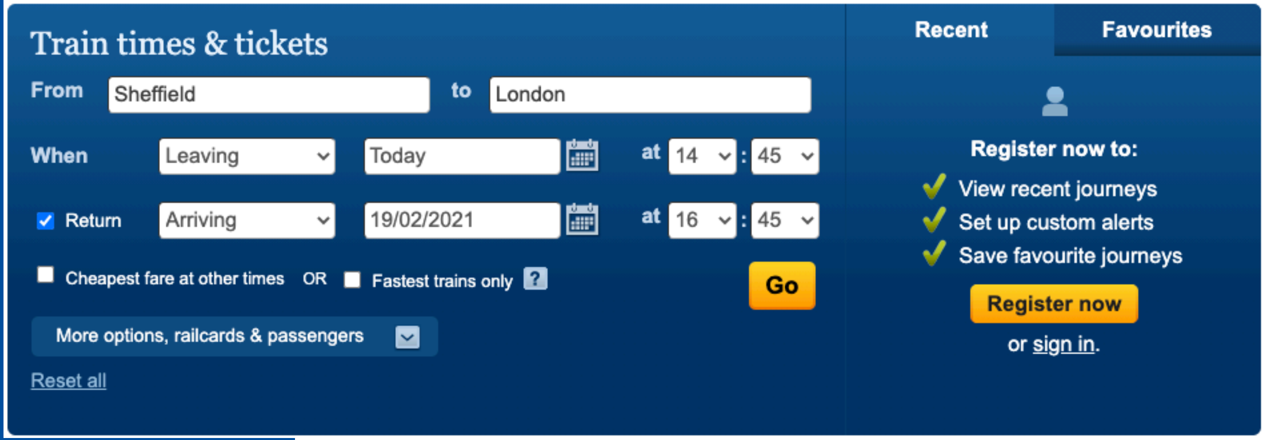
Prof. Fabio Ciravegna
Dipartimento di Informatica
Università di Torino
fabio.ciravegna@unito.it

Client Server Communication

- In node JS we have seen the concept of event based computing
  - Long operations are executed in the background and raise an event when the operation is finished
    - Callbacks
    - Promises
- How about browsers?
  - For a long time we have event based communication via Javascript, e.g.
    - *document.getElementById("button1").onclick = function() {myFunction()};*
  - But how about communication with the server?
    - can it be event based?

Classical communication

- generally based on the browser connecting to the server (e.g via a form) and waiting for an answer, e.g. a new page
  - While waiting for the response, the client is blocked



Train times & tickets

From	Sheffield	to	London		
When	Leaving	Today	at	14	45
✓ Return	Arriving	19/02/2021	at	16	45

☐ Cheapest fare at other times OR ☐ Fastest trains only ?

Go

More options, railcards & passengers

Reset all

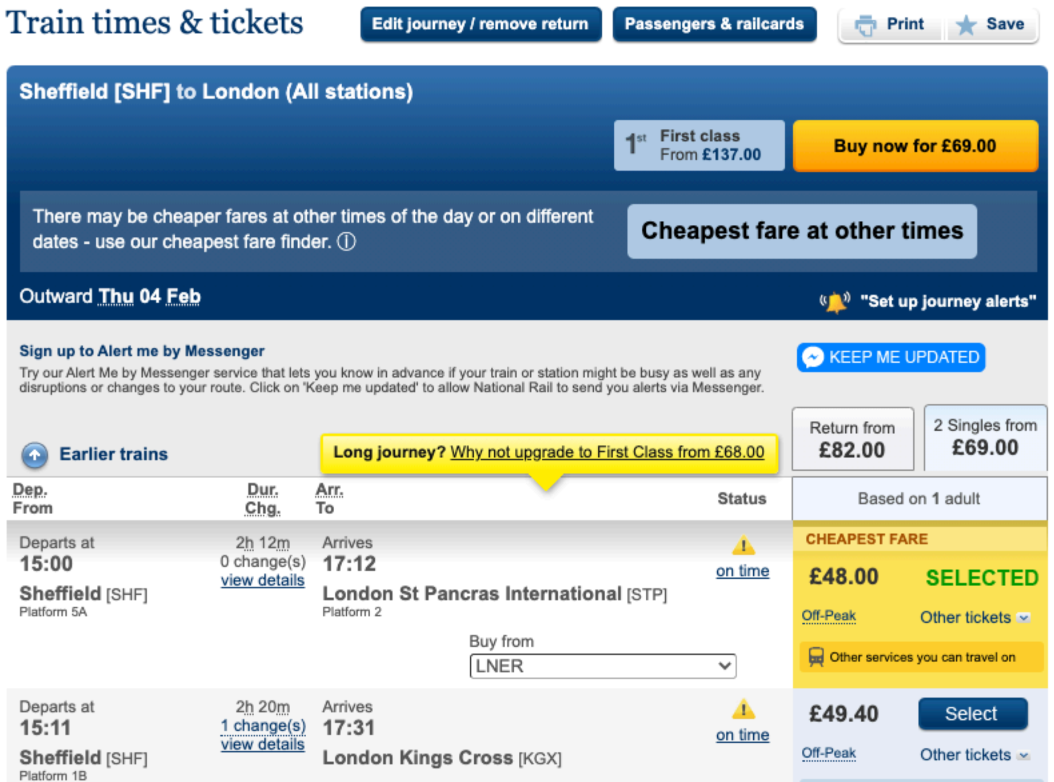
Recent | Favourites

Register now to:

- ✓ View recent journeys
- ✓ Set up custom alerts
- ✓ Save favourite journeys

Register now

or sign in.



Train times & tickets

Edit journey / remove return | Passengers & railcards | Print | Save

Sheffield [SHF] to London (All stations)

1st First class From £137.00 | Buy now for £69.00

There may be cheaper fares at other times of the day or on different dates - use our cheapest fare finder. ⓘ

Cheapest fare at other times

Outward Thu 04 Feb

"Set up journey alerts"

Sign up to Alert me by Messenger

Try our Alert Me by Messenger service that lets you know in advance if your train or station might be busy as well as any disruptions or changes to your route. Click on 'Keep me updated' to allow National Rail to send you alerts via Messenger.

KEEP ME UPDATED

Earlier trains

Long journey? Why not upgrade to First Class from £68.00

Return from £82.00 | 2 Singles from £69.00

Dep. From	Dur. Chg.	Arr. To	Status	Based on 1 adult
Departs at 15:00 Sheffield [SHF] Platform 5A	2h 12m 0 change(s) view details	Arrives 17:12 London St Pancras International [STP] Platform 2 Buy from LNER	on time	CHEAPEST FARE £48.00 SELECTED Off-Peak Other tickets Other services you can travel on
Departs at 15:11 Sheffield [SHF] Platform 1B	2h 20m 1 change(s) view details	Arrives 17:31 London Kings Cross [KGX]	on time	£49.40 Select Off-Peak Other tickets

UNIVERSITÀ DI TORINO

Towards a different Client-Server Architecture

Ajax: A New Approach to Web Applications
by Jesse James Garrett
http://adaptivepath.com/publications/essays/archives/000385.php

- Classic web application model
  - Most user actions in the interface trigger an HTTP request back to a web server.
  - The server does some processing — retrieving data, crunching numbers, talking to various legacy systems — and then returns an HTML page to the client.
- This approach makes a lot of technical sense, but it doesn't make for a great user experience.
  - While the server is doing its thing, what's the user doing?
    - That's right, waiting.
  - And at every step in a task, the user waits some more.
- Obviously, if we were designing the Web from scratch for applications, we wouldn't make users wait around. Once an interface is loaded, why should the user interaction come to a halt every time the application needs something from the server? In fact, why should the user see the application go to the server at all?

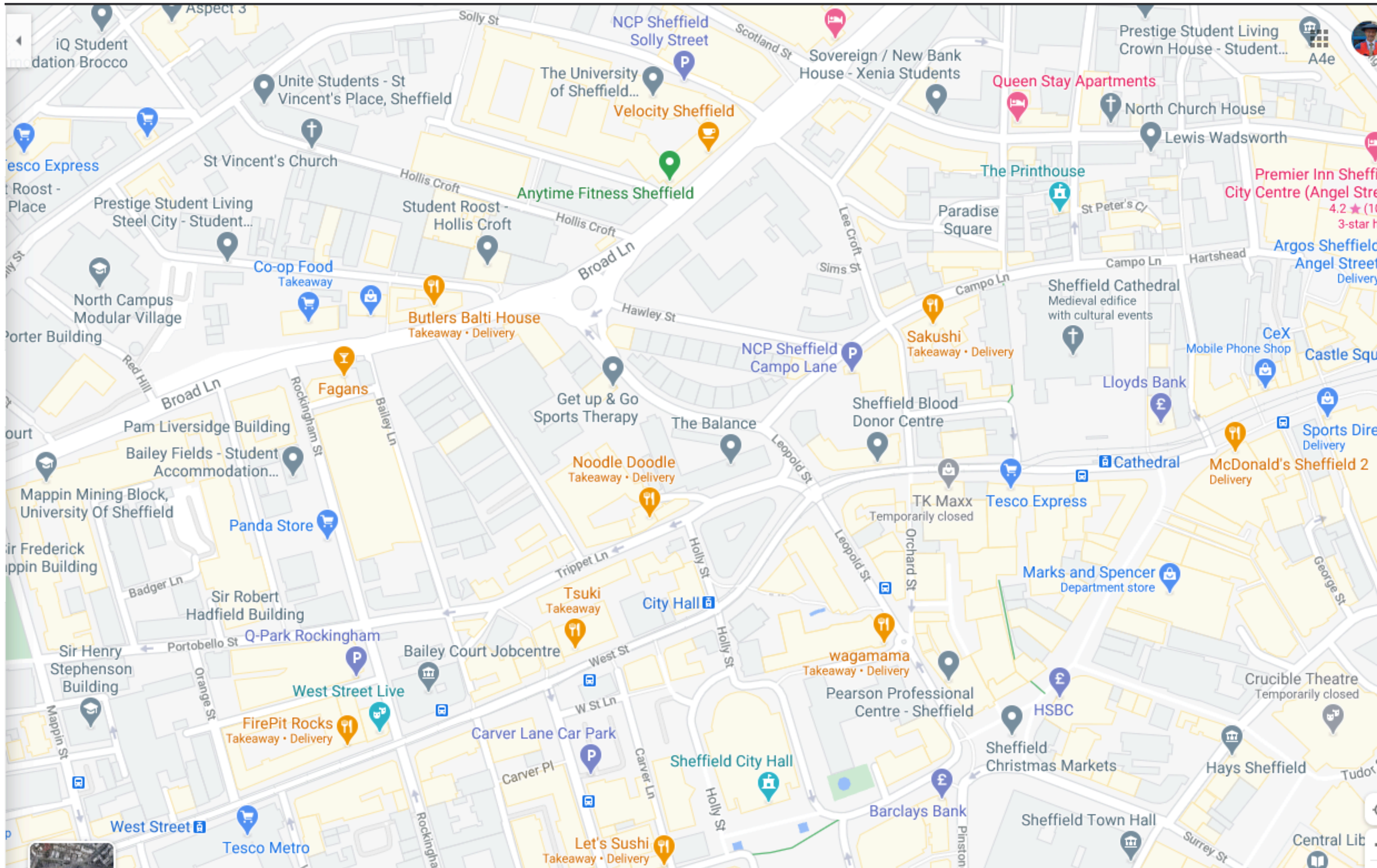
HTTP based = stateless+directional

- The communication implies sending a request (GET/POST) and receiving a full document to Display (e.g. HTML+CSS)
- Communication is stateless and directional (REST)
  - the next request will have to contain all the information because the server will have forgotten everything about my request
  - The client requests the information and the server responds
- For security reasons it is not possible to establish a communication where client and servers collaborate to achieve a task
  - e.g. the client asks for some data
    - and then asks for some more but without reissuing the request for the entire data

Example: Mapping

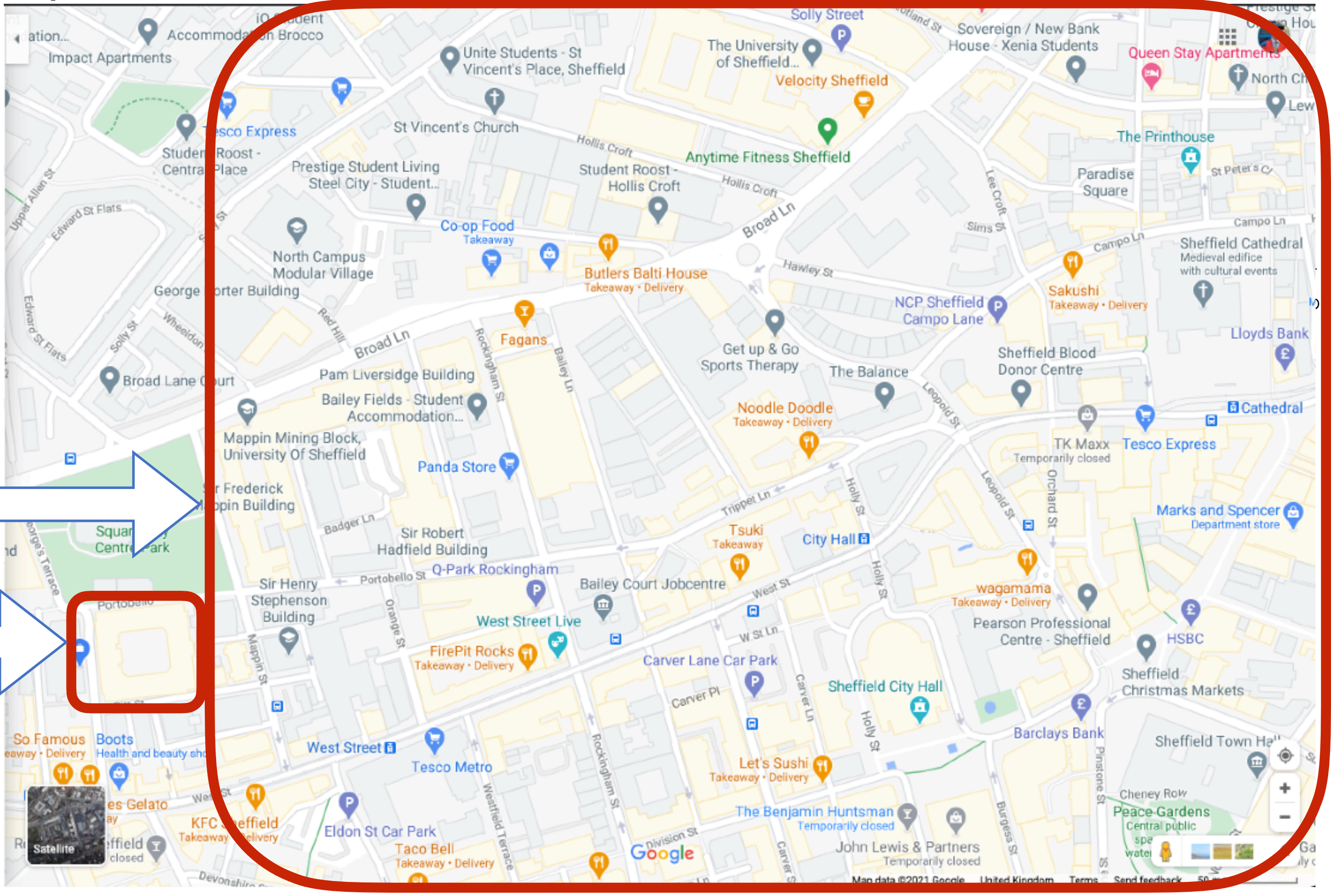
- Suppose I have this map. Regent Court is not visible
  - it is just outside

it is more or less here



In a traditional rest interaction

- I would have to reissue the request for the entire map
  - 90% of the map I already had
  - it is a serious waste



UNIVERSITÀ DI TORINO

The communication we would like

- We would like the client
  - to be able to request just the missing tiles of the map and reuse those that are already there
  - to allow to use the map while the missing tiles are fetched
- Advantages:
  - Less traffic
  - The browser page would not freeze while the data is fetched
    - Better user experience
- We can do that with Ajax
  - a Javascript middleware built in in the browser which intercepts the requests to the server and may help optimising them
    - being event based, while communication is ongoing, the browser is still alive and usable by the user

UNIVERSITÀ DI TORINO

AJAX

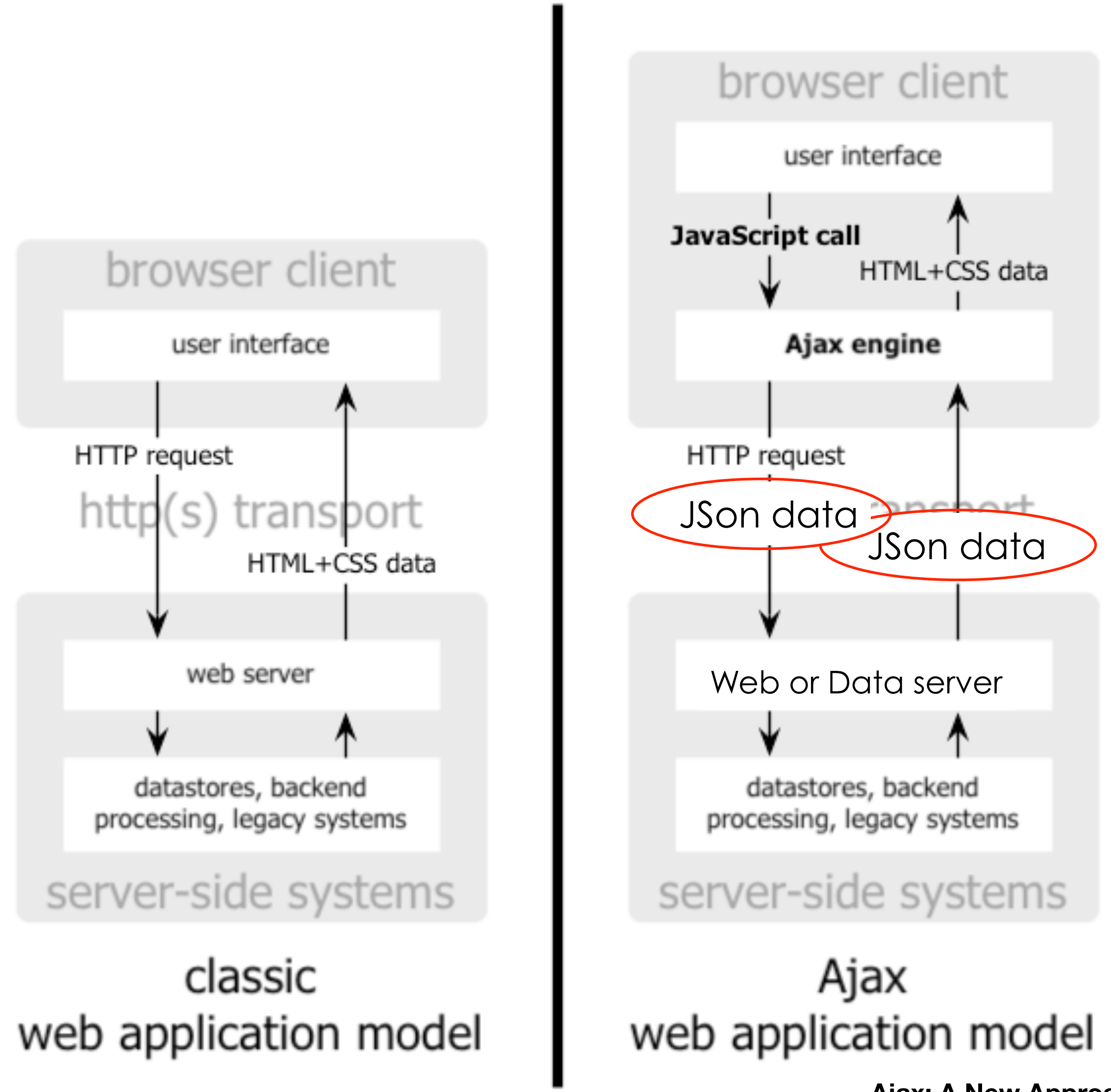
- Ajax is a Javascript library built-in in the browser
- It provides a filter between any html/javascript request and any server answer
- It allows
  - exchanging data between browser and server
    - rather than just fetching routes with serialised data and returning html
  - detaching the browser when a query is sent to the server
    - so that the user can still interacts while the data is fetched

Ajax is different

Ajax: A New Approach to Web Applications
by Jesse James Garrett
http://adaptivepath.com/publications/essays/archives/000385.php

- An Ajax application eliminates the start-stop-start-stop nature of interaction on the Web by introducing an intermediary — an Ajax engine — between the user and the server
- The Ajax engine is built in in the browser it will
    - render the user interface
    - communicate with the server on the user's behalf.
- The Ajax engine makes any communication with the server asynchronous
    - so the browser can interact with the user while they wait for a user interaction
    - The user is never staring at a blank browser window and an hourglass icon, waiting around for the server to do something

Ajax exchanges data rather than HTML pages



Ajax can exchange several data formats
we will use json

T

Ajax: A New Approach to Web Applications
by Jesse James Garrett
http://adaptivepath.com/publications/essays/archives/000385.php

What does the Ajax engine do?

- User actions that normally would generate an HTTP request takes the form of a JavaScript call to the Ajax engine instead.
- Any response to a user action that doesn’t require a trip back to the server — such as simple data validation, editing data in memory, and even some navigation — the engine handles on its own.
- If the engine needs something from the server in order to respond
  - if it’s submitting data for processing, loading additional interface code, or retrieving new data
  - the engine makes those requests asynchronously, usually using JSon, without stalling a user’s interaction with the application

The original Ajax code

We will not use this

Create object

Post to server

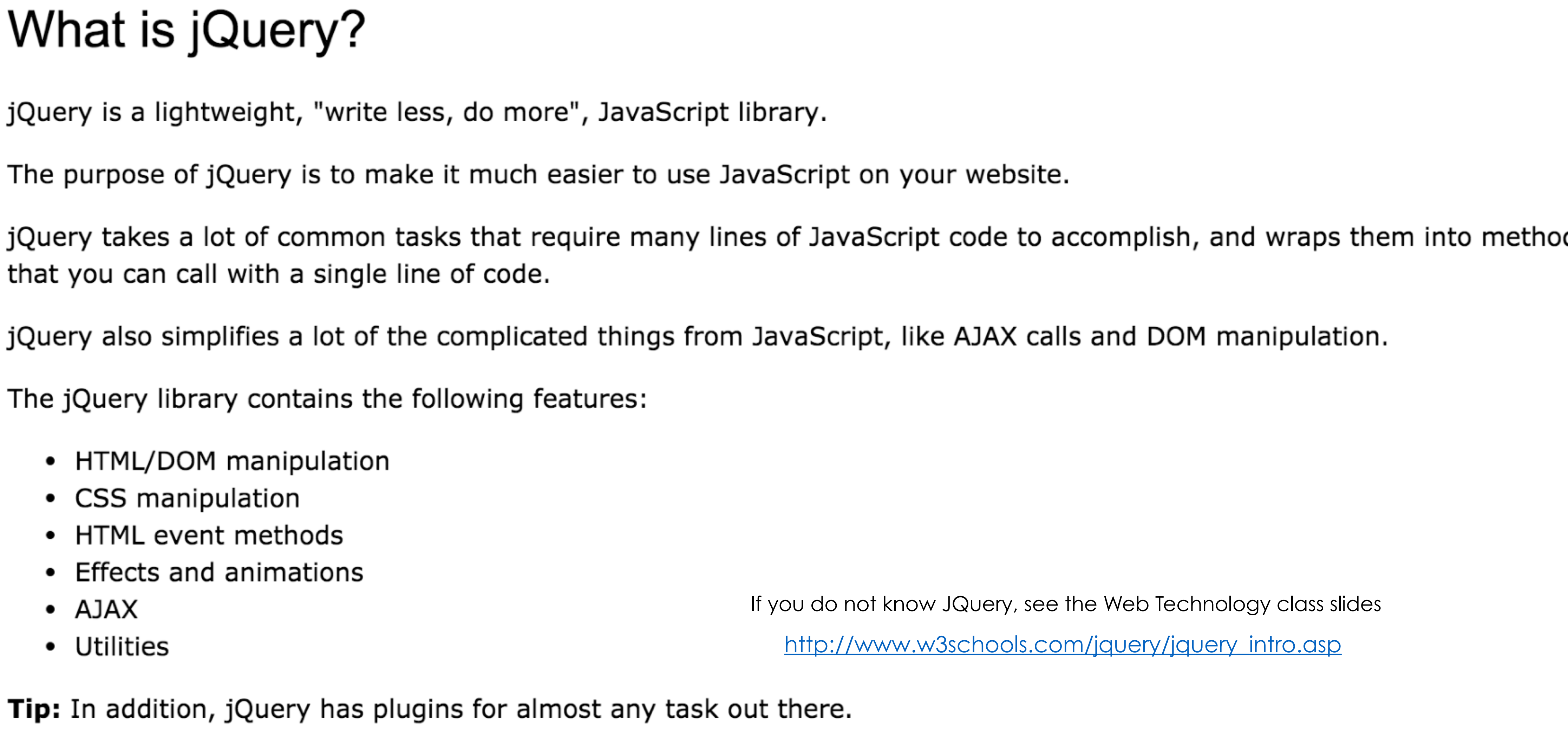
```
<html><head>
        <script>
        function submitForm() {
            var xhr;
            try {  xhr = new ActiveXObject('Msxml2.XMLHTTP');   }
            catch (e)     {
                try {   xhr = new ActiveXObject('Microsoft.XMLHTTP');    }
                catch (e2)         {
                  try {  xhr = new XMLHttpRequest();     }
                  catch (e3) {  xhr = false;   }
                }     }
            xhr.onreadystatechange  = processChange;
           xhr.open(POST, "data.txt",  true);
           xhr.setRequestHeader("Content-type","application/x-www-form-urlencoded");
           var namevalue=encodeURIComponent(document.getElementById("name").value)
           var agevalue=encodeURIComponent(document.getElementById("age").value)
           var parameters="name="+namevalue+"&age="+agevalue
            xhr.send(parameters)

        }

function processChange()    {
         if(xhr.readyState  == 4)        {
              if(xhr.status  == 200)
                  document.ajax.dyn="Received:"  + xhr.responseText;
              else
                 document.ajax.dyn="Er
         }     };

        </script>
        </head>
        </body> </html>
```

What to do when server responds
(processChange is a function that will be called)

When the function terminates, the request has been sent and the browser is free to do something else

When server responds, do something

We will not even use JQuery



What is jQuery?

jQuery is a lightweight, "write less, do more", JavaScript library.

The purpose of jQuery is to make it much easier to use JavaScript on your website.

jQuery takes a lot of common tasks that require many lines of JavaScript code to accomplish, and wraps them into metho
that you can call with a single line of code.

jQuery also simplifies a lot of the complicated things from JavaScript, like AJAX calls and DOM manipulation.

The jQuery library contains the following features:

- HTML/DOM manipulation
- CSS manipulation
- HTML event methods
- Effects and animations
- AJAX
- Utilities

Tip: In addition, jQuery has plugins for almost any task out there.

If you do not know JQuery, see the Web Technology class slides

http://www.w3schools.com/jquery/jquery_intro.asp

The Generic operation using jQuery

```
  <head>
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></scr
  </head>
  …
<script>
sendAjaxQuery(url, data){
    $.ajax({
        url: url,
        type: "POST",
        data: data,

        contentType: 'application/js
        error: function (...) {
               // do something here
        },
        success: function (response) {
               // do something here
        }
    });}
</script>
```

the url to contact

declare the action (POST)

the data to send (no need to stringily if declared datatype is JSON - done automatically by jQuery)

if an error is returned (http response code >= 300)

http response code 200<=x<300

UNIVERSITÀ DI TORINO

GET vs. POST

- Get

$.get(URL,callback);

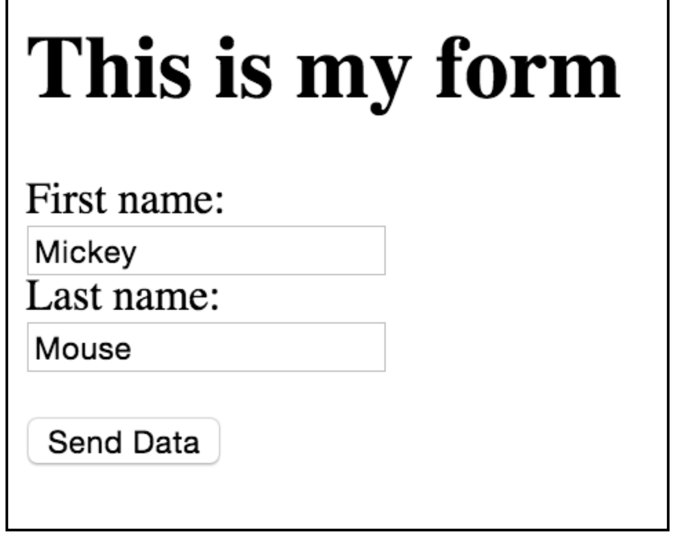
```
$("button").click(function(){
  $.get("demo_test.asp", function(data, status){
    alert("Data: " + data + "\nStatus: " + status);
  });
});
```

- Post will have also the post parameters

```
$("button").click(function(){
  $.post("demo_test_post.asp",
  {
    name: "Donald Duck",
    city: "Duckburg"
  },
  function(data, status){
    alert("Data: " + data + "\nStatus: " + stat
  });
});
```

$.post(URL,data,callback);

FORM



```
<!DOCTYPE html>
<html>
<head lang="en">
    <meta charset="UTF-8">
    <title>Ajax form</title>
    <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.2/jquery.min.js">
    </script>
</head>
<body>
<h1>This is my form</h1>
```

FORM MUST return false when Ajax is used

```
<form id="myForm" onsubmit="return false;" >
    First name:<br>
    <input type="text" name="firstname" value="Mickey">
    <br>
    Last name:<br>
    <input type="text" name="lastname" value="Mouse">
    <br><br>
    <button id="sendButton" onclick=sendData()>Send Data</button>
</form>
```

rather than using submit for form, just declare a button and the

UNIVERSITÀ
DI TORINO

Serialise form

```
function sendData() {
    var form = document.getElementById('myForm');
    sendAjaxQuery('/index.html',
JSON.stringify(getFormData());
}
```

Important: stringify

Form serialisation creates a structure containing all the data in the form.

```
function getFormData() {
  // Get a reference to the form
  const form = document.getElementById('myForm');

  // Create an empty object to store the form data
  const formData = {};

  // Iterate through the form elements
  for (const element of form.elements) {
    if (element.name) {
      formData[element.name] = element.value;
    }
  }
  return formData;
```

Ajax

```
// no need to JSON parse the result, as we are using
// contentType: ‘application/json', so JQuery knows
// it and unpacks the object before returning it
```

```
<script>
    function sendAjaxQuery(url, stringified-data) {
   $.ajax({
       url: url ,
       data: stringified-data,
        contentType: ‘application/json',
       type: 'POST',
       success: function (dataR) {
           var ret = dataR;
           // in order to have the object printed by alert
           // we need to JSON stringify the object
            // otherwise the alert will just print ‘[Object]’
           alert('Success: ' + JSON.stringify(ret));
       },
       error: function (xhr, status, error) {
           alert('Error: ' + error.message);
       }
   });
```

```
// Always use contentType: ‘application/json' rather than
// as the latter may have unpredictable behaviour in my e
```

```
data must be JSON stringify-ed before calling ajax
```

```
You MUST declare type JSON
```

Server (routes/index.js)

```
var express = require('express');
var router = express.Router();
var bodyParser= require("body-parser");


/* GET home page. */
router.get('/postFile.html', function(req, res, next) {
  res.render('index', { title: 'My Form' });
});

router.post('/index.html', function(req, res, next) {
  var body= req.body;
    res.writeHead(200, { "Content-Type": "application/json"});
    res.end(JSON.stringify(body));
});


module.exports = router;
```

`if we wanted to access the form fields, e.g.: req.body.firstname`

You MUST declare type JSON and Stringify before returning

Important!

- if while debugging you discover on either side (client or server) this situation:
  - you expect to receive an object like:

```
{"firstname":"Mickey","lastname":"Mouse","year":"1900","age":118}
```

  - and you receive in body-parser or by Ajax:

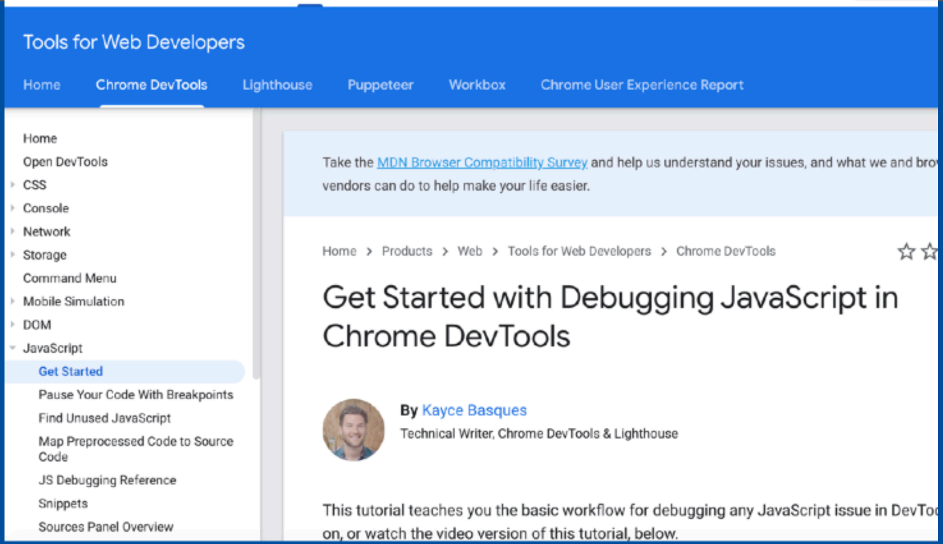
A structure

A field name that in reality is the content of your data structure in quotes

A value that makes no sense

```
{ '{"firstname":"Mickey","lastname":"Mouse","year":"1900","age":118}' : ""}
```

Then it means that you are either **not stringify-ing the data** OR **not using contentType: 'application/json'** OR in general making a mess with **JSON stringify/parse** https://developers.google.com/web/tools/chrome-devtools/javascript


Tools for Web Developers

Home | Chrome DevTools | Lighthouse | Puppeteer | Workbox | Chrome User Experience Report

Home
Open DevTools
CSS
Console
Network
Storage
Command Menu
Mobile Simulation
DOM
JavaScript
Get Started
Pause Your Code With Breakpoints
Find Unused JavaScript
Map Preprocessed Code to Source Code
JS Debugging Reference
Snippets
Sources Panel Overview

Take the MDN Browser Compatibility Survey and help us understand your issues, and what we and bro vendors can do to help make your life easier.

Home > Products > Web > Tools for Web Developers > Chrome DevTools

Get Started with Debugging JavaScript in Chrome DevTools

By Kayce Basques
Technical Writer, Chrome DevTools & Lighthouse

This tutorial teaches you the basic workflow for debugging any JavaScript issue in DevTo on, or watch the video version of this tutorial, below.

Because of course you use a debugger when programming

If you do not, you have better learn everything about debugging professionally or becoming a hairdresser (which personally I feel it is the most pointless job in the world, sa you may well understand by looking at me!!)

Whoever is caught debugging using printouts on the console only will be executed on the lab's podium ;)

Seriously: you must learn to debug as a pro - this will shorten your development time and will make your life easier

What you should know

- Why we need an event driven, flexible client server interaction with callbacks also in the communication between browser and server
- How to use Ajax (via JQuery)
- How to catch error codes and success codes
- How to create a callback to act on the results

Questions?

© Prof. Fabio Cravegna, Università di Torino