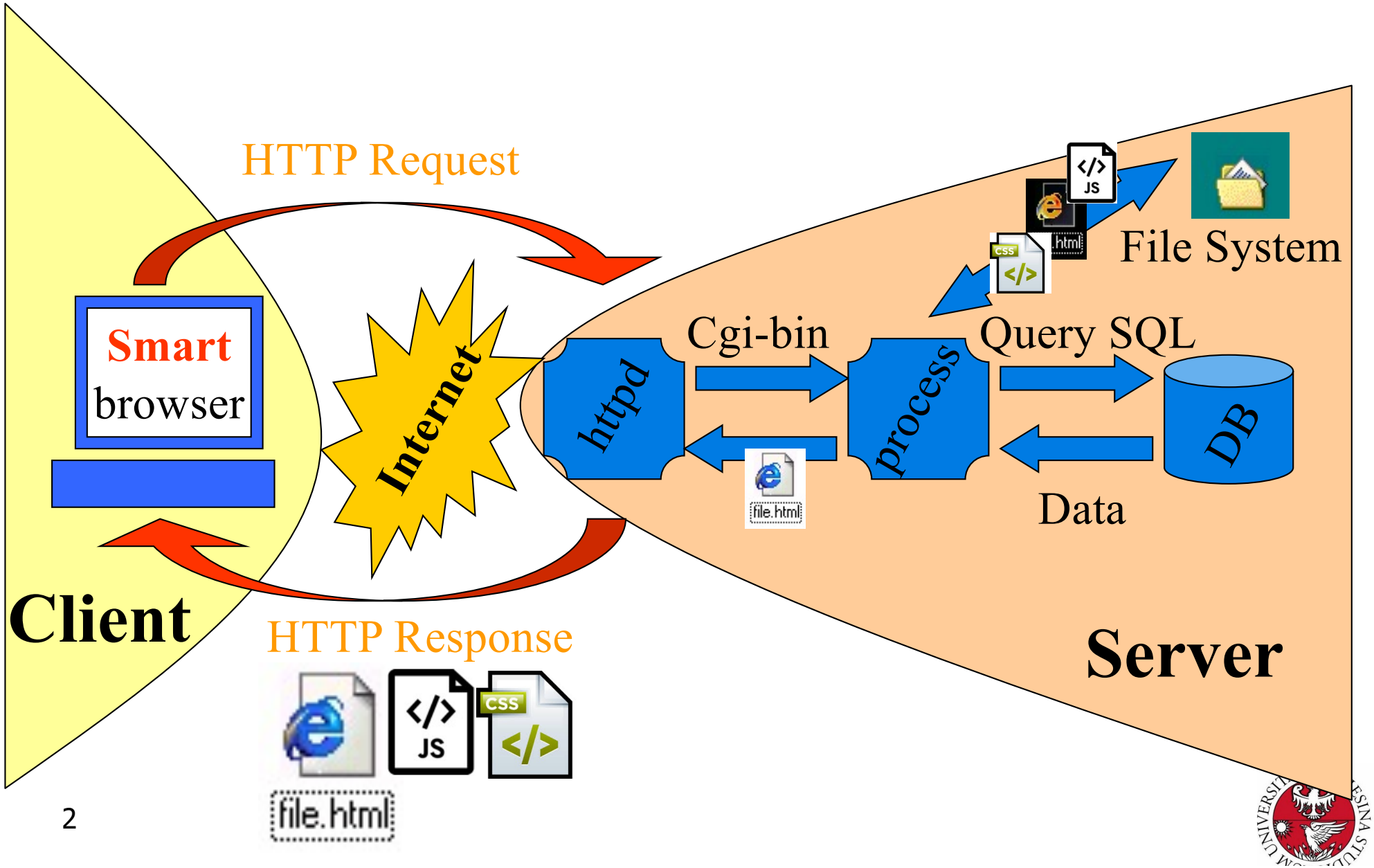
The Responsiveness problem (and solution: Ajax)


HTTP Request
Smart
browser
Internet
Client
HTTP Response
file.html
JS
CSS
Server
File System
Cgi-bin
Query SQL
httpd
process
DB
Data
file.html

Reactive Web Design

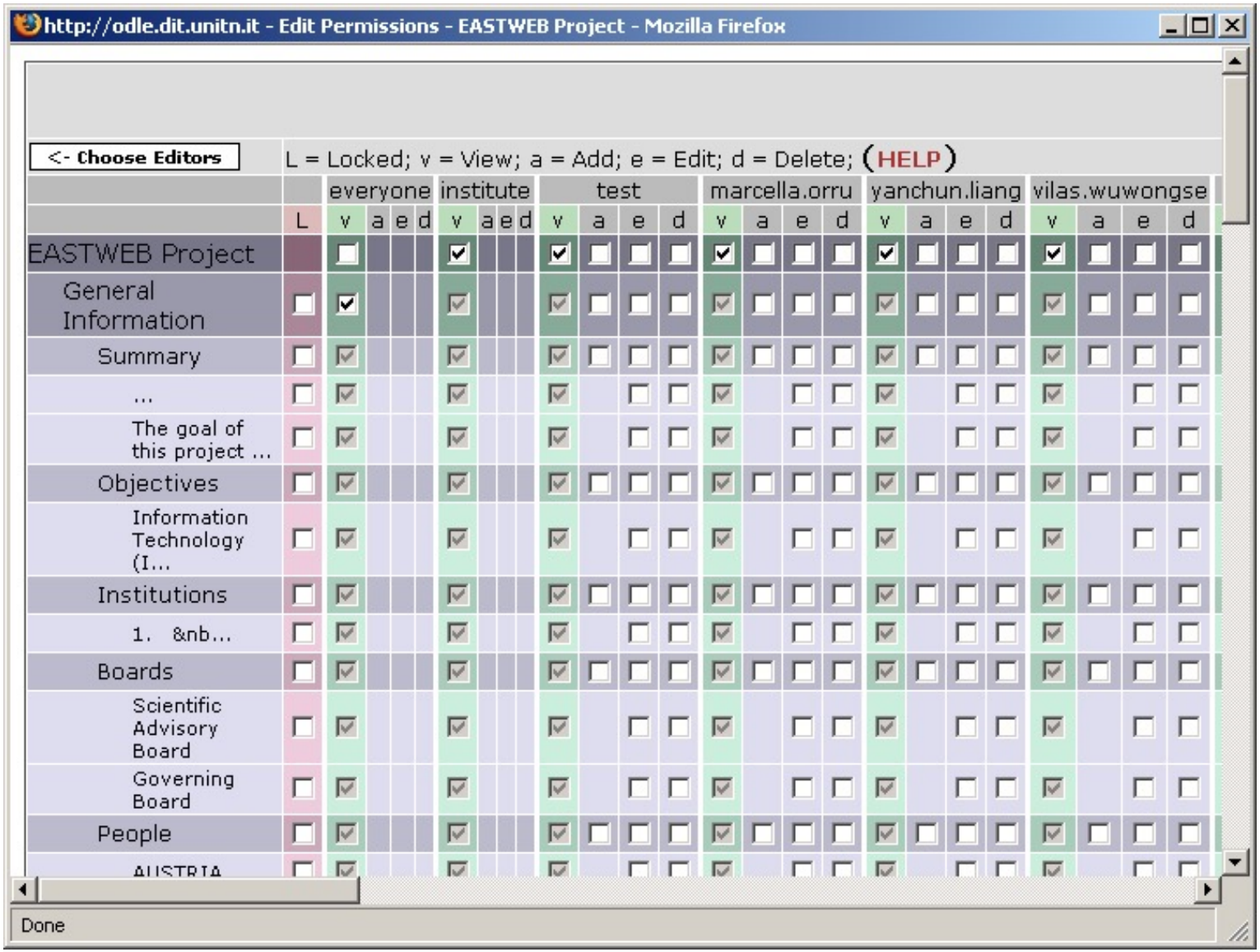
Reactive Web Design: a set of techniques that can be used to build sites that always feel fast and responsive to user input regardless of the network speed or latency.

Reactive programming is programming with **asynchronous data streams**.

Responsive Web Design

Responsive design is an approach to **web** page creation that makes use of flexible layouts, flexible images and cascading style sheet media queries. The goal of **responsive design** is to build **web** pages that detect the visitor's screen size and orientation and change the layout accordingly.

The form nightmare…



Ajax !

- not a technology in itself: it is a term coined in 2005 by Jesse James Garrett: “Asynchronous JavaScript + XML”.
- new development technique
- blur the line between web-based and desktop applications.
- rich, highly responsive and interactive interfaces

Ajax was born as:

- dynamic presentation based on XHTML + CSS;
- dynamic display and interaction using Document Object Model;
- data exchange and manipulation using XML e XSLT;
- asynchrounous data fetching using XMLHttpRequest;
- JavaScript as glue.

How does Ajax work?

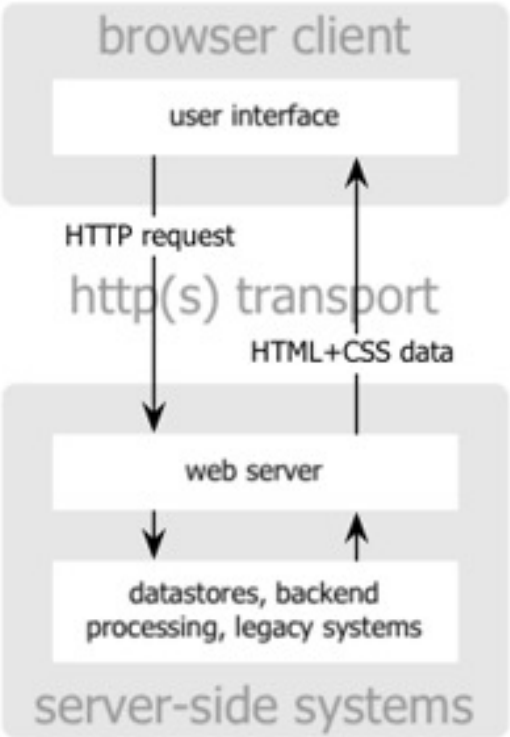
- The core idea behind AJAX is to make the communication with the server asynchronous, so that data is transferred and processed in the background.

- As a result the user can continue working on the other parts of the page without interruption.

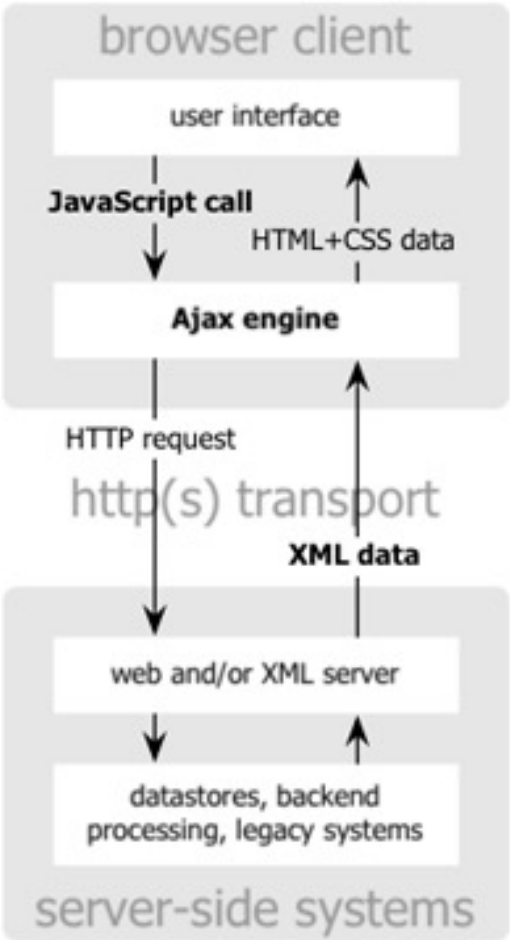
- In an AJAX-enabled application only the relevant page elements are updated, only when this is necessary.

The paradigms

1.0



classic
web application model

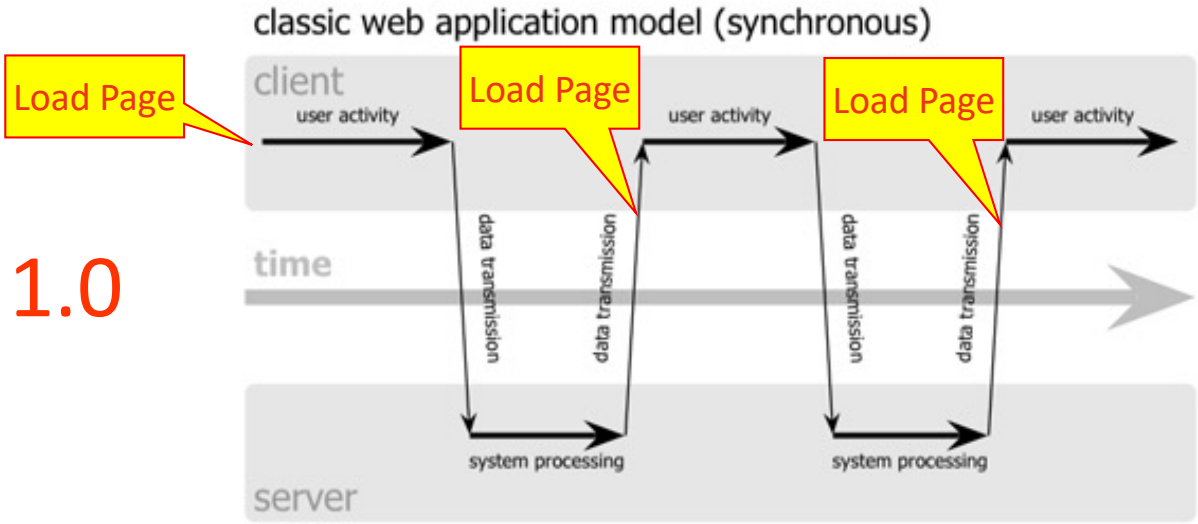


Ajax
web application model

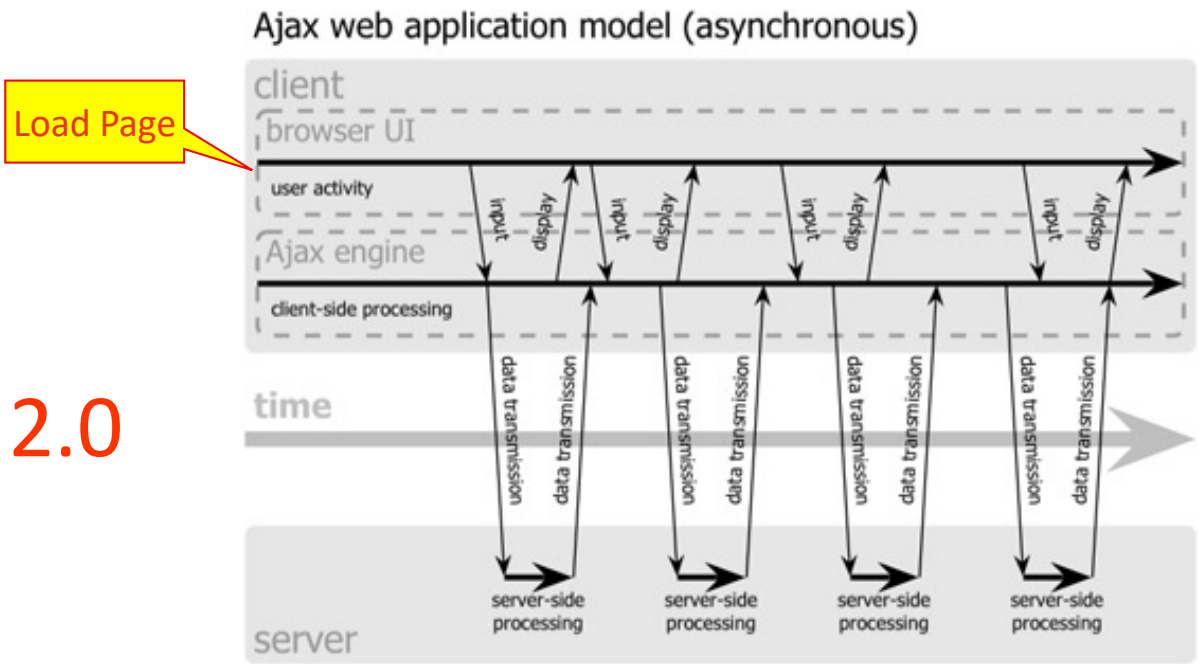
2.0

Pictures after
Jesse James Garrett

The models



1.0



2.0

Pictures after
Jesse James Garrett

The heart and history of Ajax

- First used after Microsoft implemented Microsoft *XMLHTTP* COM object that was part of The Microsoft® XML Parser (IE 5.1)
- Similarly supported by a Mozilla Javascript object *XMLHttpRequest (Mozilla 1.0, Firefox, Safari 1.2 etc.)*
- Massively used by Google

Other labels for the same technology were Load on Demand, Asynchronous Requests, Callbacks, Out-of-band Calls, etc.

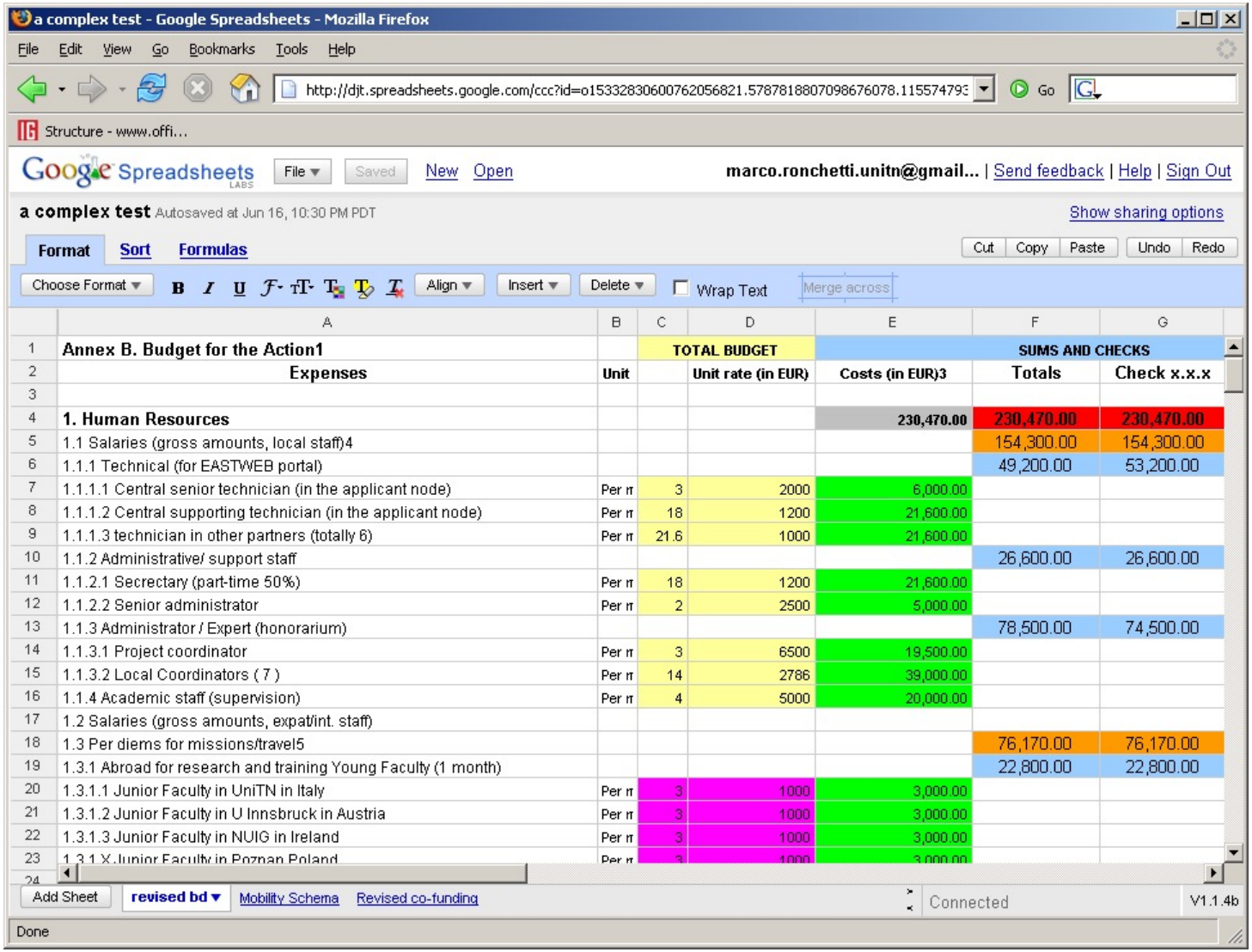
Ajax code

```
if (window.XMLHttpRequest) { // Mozilla, Safari, ...
    http_request = new XMLHttpRequest();
} else if (window.ActiveXObject) { // IE
    http_request = new ActiveXObject("Microsoft.XMLHTTP");
}
```

Ajax - advantages

- Rich applications in browsers
- No issues with installation
- Built on existing infrastructure (TCP/IP, SSL, HTTP, XML…)

The (impressive!) result



	A	B	C	D	E	F	G
1	Annex B. Budget for the Action1		TOTAL BUDGET		SUMS AND CHECKS		
2	Expenses	Unit		Unit rate (in EUR)	Costs (in EUR)3	Totals	Check x.x.x
3							
4	1. Human Resources				230,470.00	230,470.00	230,470.00
5	1.1 Salaries (gross amounts, local staff)4					154,300.00	154,300.00
6	1.1.1 Technical (for EASTWEB portal)					49,200.00	53,200.00
7	1.1.1.1 Central senior technician (in the applicant node)	Per m	3	2000	6,000.00		
8	1.1.1.2 Central supporting technician (in the applicant node)	Per m	18	1200	21,600.00		
9	1.1.1.3 technician in other partners (totally 6)	Per m	21.6	1000	21,600.00		
10	1.1.2 Administrative/ support staff					26,600.00	26,600.00
11	1.1.2.1 Secrectary (part-time 50%)	Per m	18	1200	21,600.00		
12	1.1.2.2 Senior administrator	Per m	2	2500	5,000.00		
13	1.1.3 Administrator / Expert (honorarium)					78,500.00	74,500.00
14	1.1.3.1 Project coordinator	Per m	3	6500	19,500.00		
15	1.1.3.2 Local Coordinators (7)	Per m	14	2786	39,000.00		
16	1.1.4 Academic staff (supervision)	Per m	4	5000	20,000.00		
17	1.2 Salaries (gross amounts, expat/int. staff)						
18	1.3 Per diems for missions/travel5					76,170.00	76,170.00
19	1.3.1 Abroad for research and training Young Faculty (1 month)					22,800.00	22,800.00
20	1.3.1.1 Junior Faculty in UniTN in Italy	Per m	3	1000	3,000.00		
21	1.3.1.2 Junior Faculty in U Innsbruck in Austria	Per m	3	1000	3,000.00		
22	1.3.1.3 Junior Faculty in NUIG in Ireland	Per m	3	1000	3,000.00		
23	1.3.1.X Junior Faculty in Poznan Poland	Per m	3	1000	3,000.00		

Ajax - advantages

- Better Performance and Efficiency
  - small amount of data transferred from the server. Beneficial for data-intensive applications as well as for low-bandwidth networks.
- More Responsive Interfaces
  - the improved performance give the feeling that updates are happening instantly. AJAX web applications appear to behave much like their desktop counterparts.
- Reduced or Eliminated "Waiting" Time
  - only the relevant page elements are updates, with the rest of the page remaining unchanged. This decreases the idle waiting time.
- Increased Usability
  - Users can work with the rest of the page while data is being transferred in the background.

XMLHttpRequest

```
function loadDoc() {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
     document.getElementById("demo").innerHTML =
              this.responseText;
    }
  };
  xhttp.open("GET", "ajax_info.txt", true);
  xhttp.send();
}
```

Asynchronous Programming!

XMLHttpRequest – Getting static resources

```
function loadDoc() {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
     document.getElementById("demo").innerHTML =

            this.responseText;
    }
  };
  xhttp.open("GET", "ajax_info.txt", true);
  xhttp.send();
}
```

Getting dynamic resources with GET

```
function loadDoc() {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
     document.getElementById("demo").innerHTML =
              this.responseText;
    }
  };
  xhttp.open("GET", "myservlet?param1=27", true);
  xhttp.send();
}
```

Note: if you want to avoid getting cashed results, add a fake parameter with the current time, e.g.

```
xhttp.open("GET", url + ((/\?/).test(url) ? "&" : "?") + (new Date()).getTime());
```

See https://www.w3schools.com/jsref/jsref_regexp_test.asp to understand the code above

Getting dynamic resources with POST

```
function loadDoc() {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
     document.getElementById("demo").innerHTML =

            this.responseText;
    }
  };
  xhttp.open("POST", "ajax_info.txt", true);
  xhttp.setRequestHeader("Content-type",
      "application/x-www-form-urlencoded");

  xhttp.send("nome=Dorothea&lname=Wierer");
}
```

1) add an HTTP header with setRequestHeader().
2) Specify the data you want to send in the send() method

XMLHttpRequest methods

new XMLHttpRequest()	Creates a new XMLHttpRequest object
abort()	Cancels the current request
getAllResponseHeaders()	Returns header information
getResponseHeader()	Returns specific header information
open(*method, url, async, user, psw*)	Specifies the request *method*: the request type **GET** or **POST** *url*: the file location *async*: true (asynchronous) or false (synchronous) *user*: optional user name *psw*: optional password
send()	Sends the request to the server Used for GET requests
send(*string*)	Sends the request to the server. Used for POST requests
setRequestHeader()	Adds a label/value pair to the header to be sent

XMLHttpRequest properties

Property	Description
onreadystatechange	Defines a function to be called when the readyState property changes
readyState	Holds the status of the XMLHttpRequest. 0: request not initialized 1: server connection established 2: request received 3: processing request 4: request finished and response is ready
responseText	Returns the response data as a string
responseXML	Returns the response data as XML data
status	Returns the HTTP status-number of a request, e.g. 200: "OK" 403: "Forbidden" 404: "Not Found"
statusText	Returns the status-text (e.g. "OK" or "Not Found")

And make sure that you…

- Preserve the Normal Page Lifecycle – as much as possible!
- Reflect Control State on the Server – in real-life scenarios there is no use of simply rendering controls on the page.
- Support Cross-Browser usage – there are different implementation of the XmlHttpRequest object. You should make sure that all AJAX components you choose operate properly on various browsers and platforms.
- Ensure proper Operation when Cookies are Disabled – support cookieless sessions.

And make sure that you…

- Give visual feedback - When a user clicks on something in the AJAX user interface, they need immediate visual feedback
- Keep the Back button – make sure that the Back button in your application functions on every page of the site.
- Use links for navigation – avoid the temptation to use links as an interface on your AJAX application to change the state of your application. Users have been trained over many years to expect a link to “take” them somewhere, so give them what they expect.
- Use human-readable links – people like to pass the addresses of useful web pages to each other. Make sure your application supports URLs that people can share easily, so not too long or complex.

Adapted from: www.telerik.com/documents/Telerik_and_AJAX.pdf

And make sure that you…

- Don’t bloat the code – make sure that your application uses as little client-side scripting as possible. This reduces download time for the page and also reduces the processor requirements on the client browser, so results in a faster browser experience.
- Follow UI conventions – AJAX is a world of possibilities, but when it comes to user interface the best is invariably the familiar. If you’re creating a user interface for a specific feature, the place to start is by replicating an existing successful interface and looking at what your clients expect. Also remember that although it may be cool to implement drag-and-drop, few people may realize the interface relies on it.
- Don’t scroll – users like to feel in control, so if they have moved the scrollbar to a specific place, don’t move the page somewhere else.
- Reduce page loads – do as much as you can to reduce the number of page loads the user has to do to achieve their goal.

Adapted from: www.telerik.com/documents/Telerik_and_AJAX.pdf

AJAX Tutorial and reference

JS AJAX

AJAX Intro
AJAX XMLHttp
AJAX Request
AJAX Response
AJAX XML File
AJAX PHP
AJAX ASP
AJAX Database
AJAX Applications
AJAX Examples

https://www.w3schools.com/js/js_ajax_intro.asp

https://developer.mozilla.org/en-US/docs/Web/Guide/AJAX

Cross Origin

For security reasons, browsers restrict cross-origin HTTP requests initiated from scripts.

CORS - Cross-Origin Resource Sharing

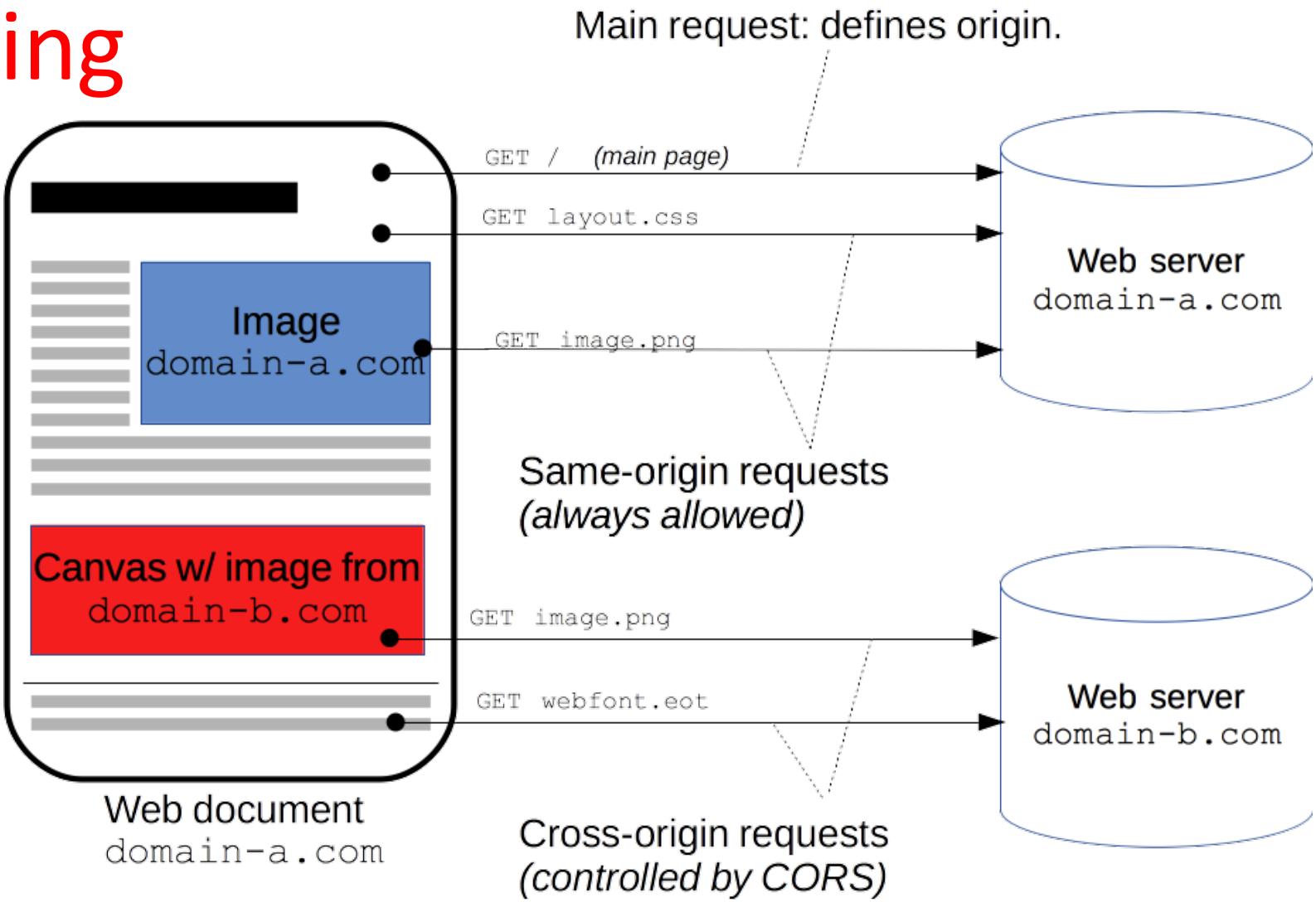


immagine da https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS

CORS

- **Cross-site XMLHttpRequest**
- Modern browsers support cross-site requests by implementing the Cross-Origin Resource Sharing (CORS) standard. As long as the server is configured to allow requests from your web application's origin, XMLHttpRequest will work. Otherwise, an INVALID_ACCESS_ERR exception is thrown.
- CORS failures result in errors, but for security reasons, specifics about the error *are not available to JavaScript*. All the code knows is that an error occurred.

CORS

```
const xhr = new XMLHttpRequest();
const url = 'https://bar.other/resources/public-data/';
xhr.open('GET', url);
xhr.onreadystatechange = someHandler;
xhr.send();
```

What the browser will send to the server:

```
GET /resources/public-data/ HTTP/1.1
Host: bar.other
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS
X 10.14; rv:71.0) Gecko/20100101 Firefox/71.0
Accept: text/html,application/xhtml+xml,
application/xml;
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Connection: keep-alive
Origin: https://foo.example
```

How the server will respond:

CORS

```
HTTP/1.1 200 OK
Date: Mon, 01 Dec 2008 00:23:53 GMT
Server: Apache/2
Access-Control-Allow-Origin: *
Keep-Alive: timeout=2, max=100
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: application/xml

[…XML Data…]
```

Alternatives:

```
Access-Control-Allow-Origin: https://foo.example
```

For more detail, see https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS

Adding CORS to Apache

- To add the CORS authorization to the header using Apache, simply add the following line inside either
  the <Directory>, <Location>, <Files> or <VirtualHost> sections of your server config (usually located in a *.conf file, such as httpd.conf or apache.conf), or within a .htaccess file:

```
Header set Access-Control-Allow-Origin "*"
```

- https://enable-cors.org/server_apache.html

- https://poanchen.github.io/blog/2016/11/20/how-to-enable-cross-origin-resource-sharing-on-an-apache-server

Ajax

- https://www.tutorialspoint.com/ajax/ajax_technology.htm