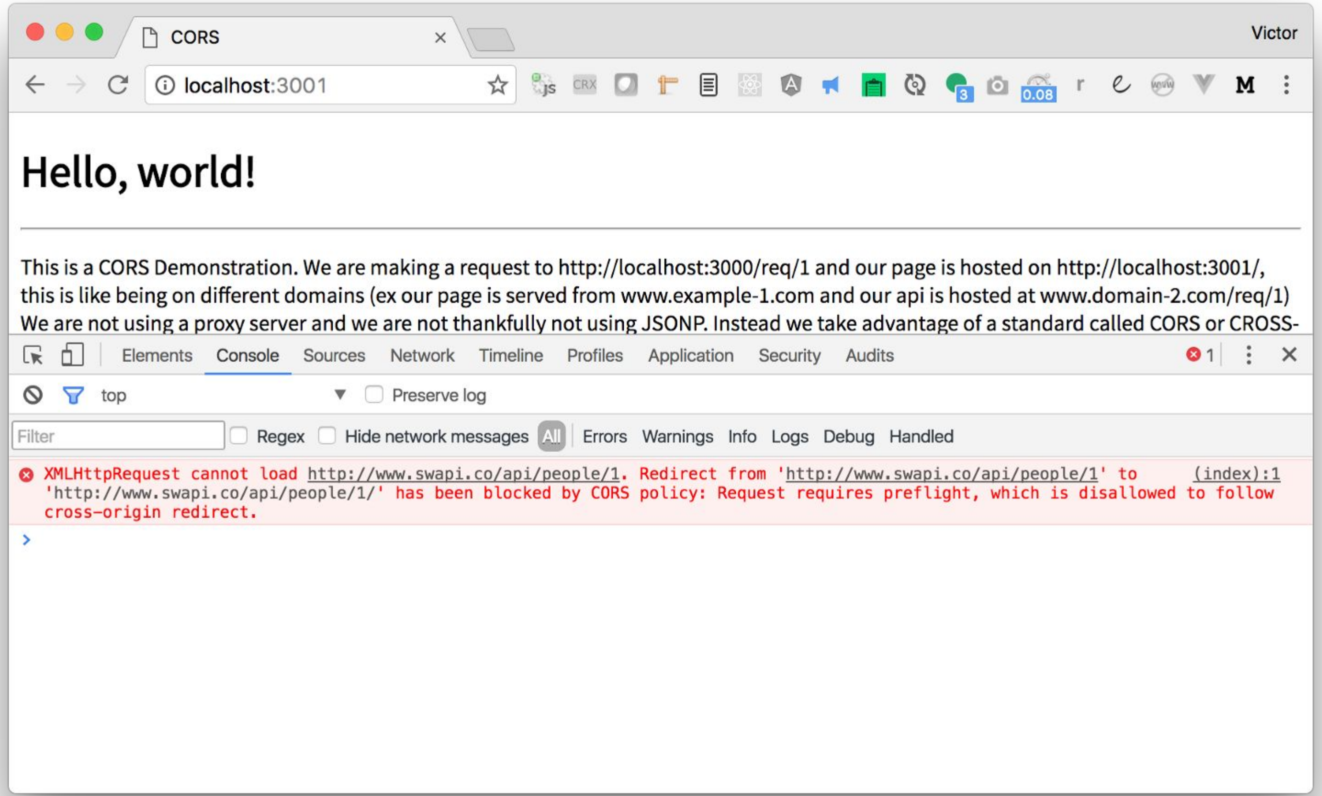
AN INTRODUCTION TO FULLSTACK FOR BEGINNERS

MODULO 7 – INTEGRATING WITH BACKENDS

Matteo Colombo Speroni
Danilo D'Alessandro
Armando Esposito
Marco Montalbano

Starting from an example



Hello, world!

This is a CORS Demonstration. We are making a request to http://localhost:3000/req/1 and our page is hosted on http://localhost:3001/, this is like being on different domains (ex our page is served from www.example-1.com and our api is hosted at www.domain-2.com/req/1) We are not using a proxy server and we are not thankfully not using JSONP. Instead we take advantage of a standard called CORS or CROSS-

```
XMLHttpRequest cannot load http://www.swapi.co/api/people/1. Redirect from 'http://www.swapi.co/api/people/1' to    (index):1
'http://www.swapi.co/api/people/1/' has been blocked by CORS policy: Request requires preflight, which is disallowed to follow
cross-origin redirect.
```

SOP - Same Origin Policy

*Browser **restrict the access** to the elements in the Document Object Model (DOM) of a page only to the scripts having the exact **same origin** of the application serving that page.*

Eg: http://my.domain.com/dir/index.html

URL	OUTCOME	REASON
http://my.domain.com/dir2/other.html	Success	
https://my.domain.com/dir2/other.html	Failure	Different **protocol**
http://my.domain.com:**8080**/dir2/other.html	Failure	Different **port**
http://**other**.domain.com/dir2/other.html	Failure	Different **host**

SOP - Why Bypassing - Techniques

Big companies with a ***base domain*** (eg. **company.com**) make use of multiple ***subdomains*** to deploy all different services such as **login.company.com**, **media.company.com**, ..

There are many techniques for bypassing SOP check policy:

1. Manipulating the origin of documents:
   a. The document.property
2. Exploiting allowed cross-origin embeddings:
   a. JSONP
   b. The iframe hack
3. Avoiding browser-to-server cross-origin calls:
   a. Cross-document messaging
   b. Server side proxies
4. Restricting access with origin whitelisting:
   a. WebSockets
   b. Cross-Origin Resource Sharing

JSONP - Json With Padding

Request data from a server in a different domain, taking advantage of the fact that browsers do not enforce the Same Origin Policy on <script> tags

Limitations:

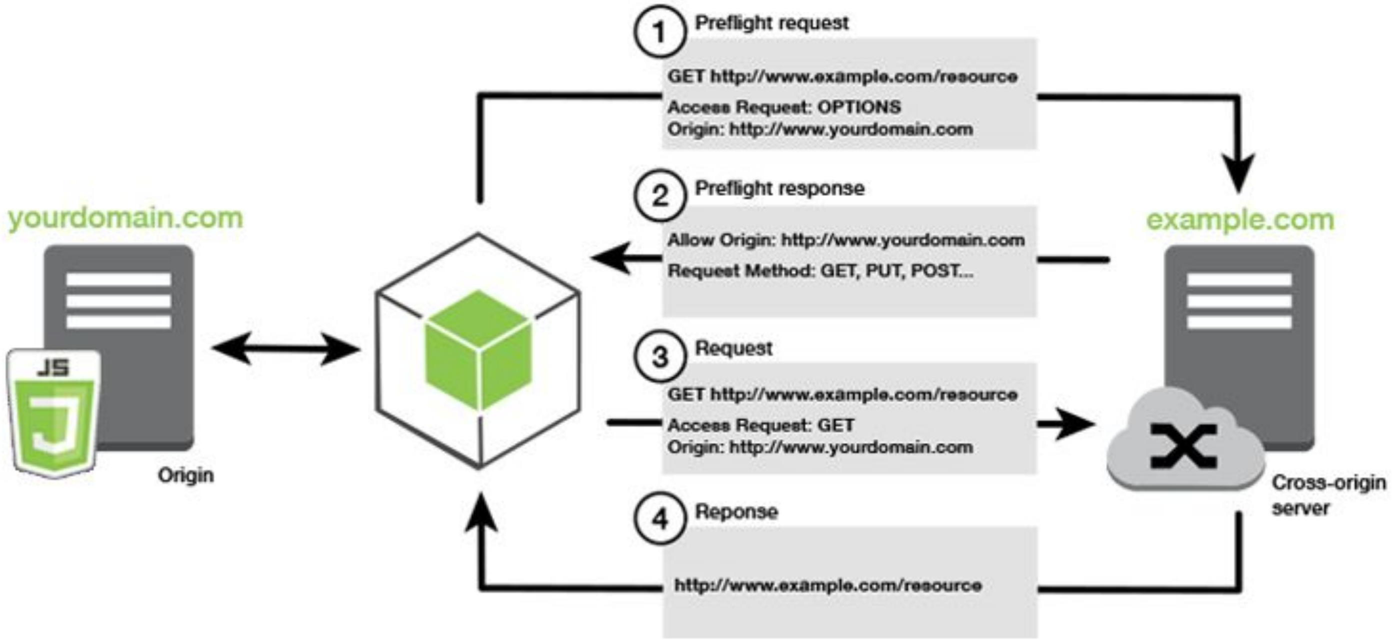
- Only GET HTTP requests
- Error handling more difficult
- Need modifications on Server Side
- Security Issues

```
function jsonCallback(json){

  console.log(json);

}


$.ajax({

  url: "http://run.plnkr.co/plunks/v8xyYN64V4nqCshgjKms/data-2.json",

  dataType: "jsonp"

});
```

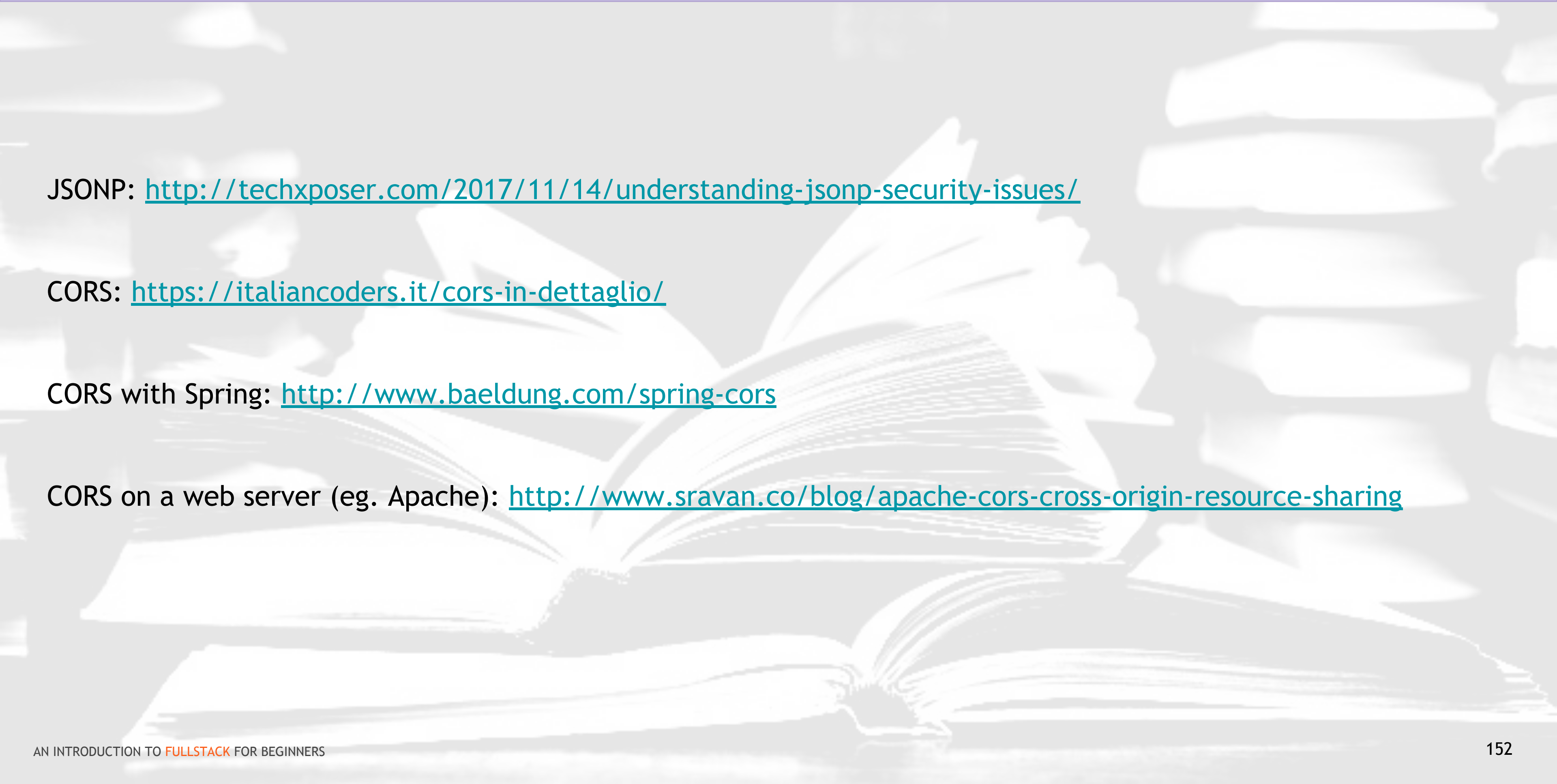
CORS - Cross-Origin Resource Sharing

The W3C Recommended Standard for performing cross-origin requests. It Allows servers to setup cross-origin access control rules to enable cross-origin data transfer in a controlled way.

In general this configuration is made at Web Server level (eg. Apache) or in some cases at application level.


yourdomain.com
JS
Origin
1 Preflight request
GET http://www.example.com/resource
Access Request: OPTIONS
Origin: http://www.yourdomain.com
2 Preflight response
Allow Origin: http://www.yourdomain.com
Request Method: GET, PUT, POST...
3 Request
GET http://www.example.com/resource
Access Request: GET
Origin: http://www.yourdomain.com
4 Reponse
http://www.example.com/resource
example.com
Cross-origin server

Useful links



JSONP: http://techxposer.com/2017/11/14/understanding-jsonp-security-issues/

CORS: https://italiancoders.it/cors-in-dettaglio/

CORS with Spring: http://www.baeldung.com/spring-cors

CORS on a web server (eg. Apache): http://www.sravan.co/blog/apache-cors-cross-origin-resource-sharing