



April 2018, IPT Course
Introduction to Spring 5

Introduction to REST & HATEOAS

Trayan Iliev

tiliev@iproduct.org

<http://iproduct.org>

Copyright © 2003-2018 IPT - Intellectual
Products & Technologies

About me



Trayan Iliev

- CEO of IPT – Intellectual Products & Technologies
- Oracle® certified programmer 15+ Y
- end-to-end reactive fullstack apps with Java, ES6/7, TypeScript, Angular, React and Vue.js
- 12+ years IT trainer
- Voxxed Days, jPrime, jProfessionals, Java2Days, BGOUG, BGJUG, DEV.BG speaker

Agenda for This Session

- ❖ Multimedia and Hypermedia – basic concepts
- ❖ Service Oriented Architecture (SOA),
- ❖ Cloud computing and client side mashups
- ❖ REpresentational State Transfer (REST)
- ❖ Hypermedia As The Engine Of Application State (HATEOAS)
- ❖ New Link HTTP header
- ❖ Richardson Maturity Model of Web Applications
- ❖ Cross Origin Resource Sharing

Where to Find the Code?

Java Web Development projects and examples are available @ GitHub:

<https://github.com/iproduct/course-java-web-2021>



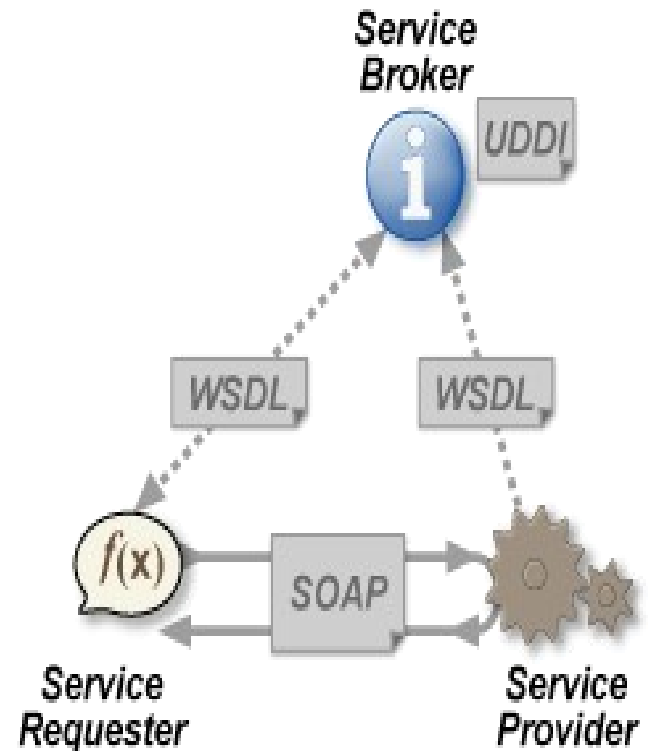
Service Oriented Architecture (SOA) – Definitions

Thomas Erl: SOA represents an open, agile, extensible, federated, composable architecture comprised of autonomous, QoS-capable, vendor diverse, interoperable, discoverable, and potentially reusable services, implemented as Web services. SOA can establish an abstraction of business logic and technology, resulting in a loose coupling between these domains. SOA is an evolution of past platforms, preserving successful characteristics of traditional architectures, and bringing with it distinct principles that foster service-orientation in support of a service-oriented enterprise. SOA is ideally standardized throughout an enterprise, but achieving this state requires a planned transition and the support of a still evolving technology set.

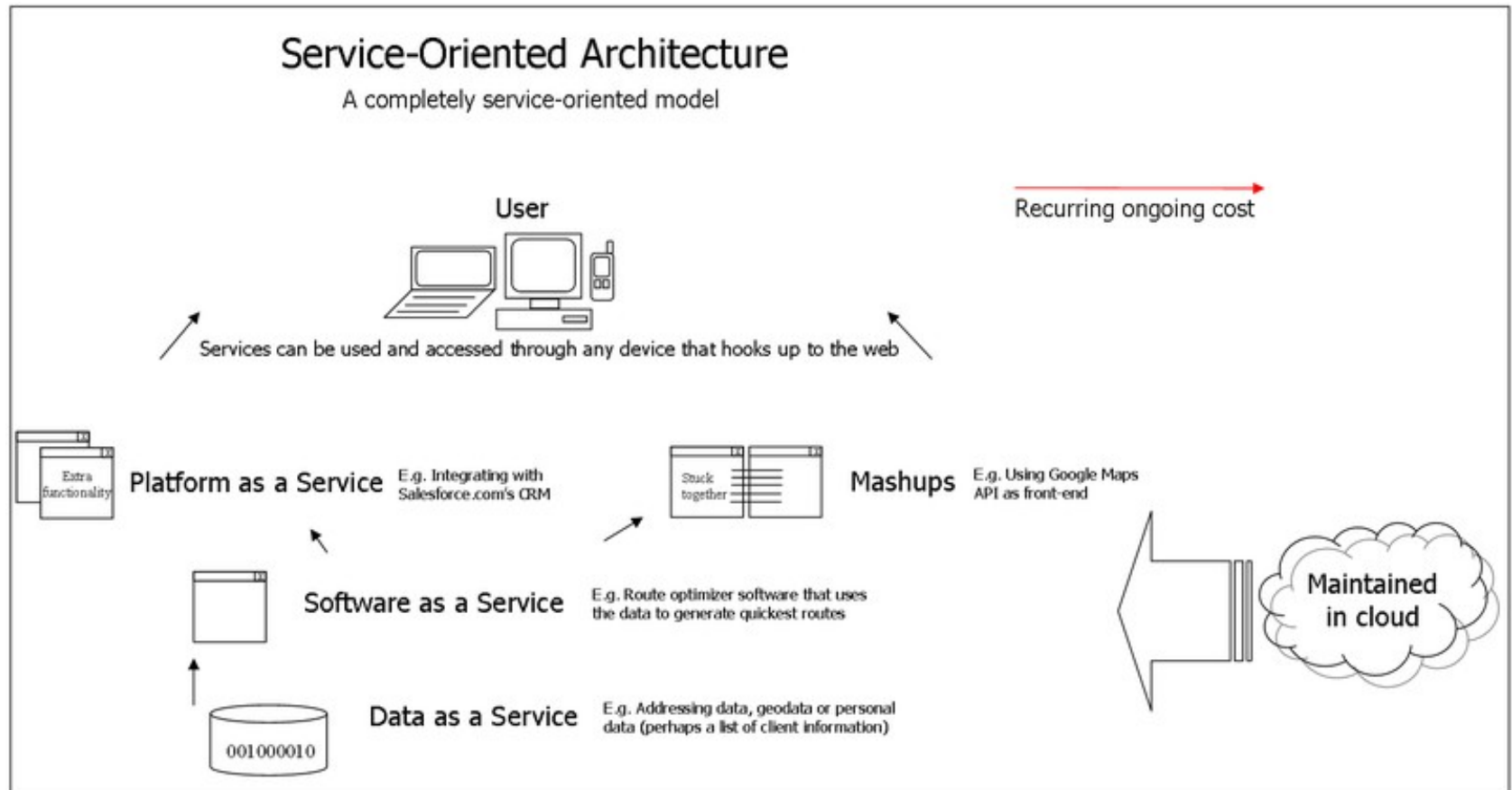
References: Erl, Thomas. serviceorientation.org – About the Principles, 2005–06

Classical Web Services - SOAP + WSDL

- Web Services are:
- components for building distributed applications in SOA architectural style
- communicate using open protocols
- are self-descriptive and self-content
- can be searched and found using UDDI or ebXML registries (and more recent specifications – WSIL & **Semantic Web Services**)

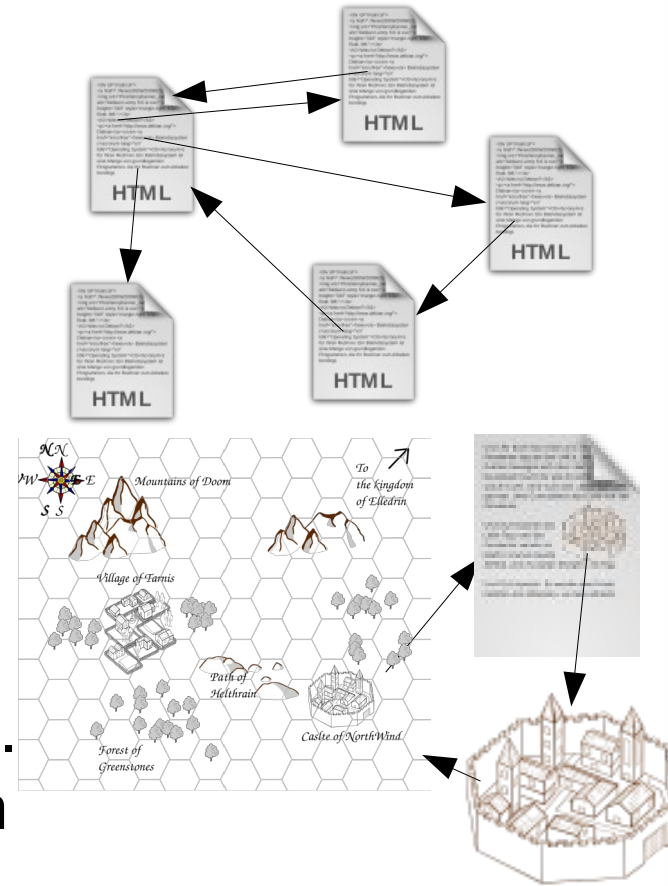


Service Oriented Architecture (SOA)




Hypertext & Hypermedia

- ❖ **Hypertext** is structured text that uses logical links (hyperlinks) between nodes containing text
- ❖ **HTTP** is the protocol to exchange or transfer hypertext
- ❖ **Hypermedia** - extension of the term hypertext, is a nonlinear medium of information which includes multimedia (text, graphics, audio, video, etc.) and hyperlinks of different media types (e.g. image or animation/video fragment can be linked to a detailed description).



Multipurpose Internet Mail Extensions (MIME)

- ❖ Different types of media are represented using different text/binary encoding formats – for example:
 - Text -> plain, html, xml ...
 - Image (Graphics) -> gif, png, jpeg, svg ...
- ❖ Multipurpose Internet Mail Extensions (MIME) allows the client to recognize how to handle/present the particular multimedia asset/node:
Ex.: **Content-Type: text/plain**


Media Type Media SubType (format)
- ❖ More examples for standard MIME types:
https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/MIME_types
- ❖ Vendor specific media (MIME) types:
application/vnd.*+json/xml

HTTP Request Structure

GET /context/Servlet **HTTP/1.1**

Host: *Client_Host_Name*

Header2: Header2_Data

...

HeaderN: HeaderN_Data

<Празен ред>

POST /context/Servlet
HTTP/1.1

Host: *Client_Host_Name*

Header2: Header2_Data

...

HeaderN: HeaderN_Data

<Празен ред>

POST_Data

HTTP Response Structure

HTTP/1.1 200 OK

Content-Type:
application/json

Header2: Header2_Data

...

HeaderN: HeaderN_Data

<Празен ред>

```
[{ "id":1,  
  "name":"Novelties in Java EE 7 ...",  
  "description":"The presentation is ...",  
  "created":"2014-05-10T12:37:59",  
  "modified":"2014-05-10T13:50:02",  
},  
{ "id":2,  
  "name":"Mobile Apps with HTML5 ...",  
  "description":"Building Mobile ...",  
  "created":"2014-05-10T12:40:01",  
  "modified":"2014-05-10T12:40:01",  
}]
```

Architectural Properties

According to **Dr. Roy Fielding** [Architectural Styles and the Design of Network-based Software Architectures, 2000]:

- Performance
- Scalability
- Reliability
- Simplicity
- Extensibility
- Dynamic evolvability
- Customizability
- Configurability
- Visibility

- ❖ All of them should be present in a desired Web Architecture and REST architectural style tries to preserve them by consistently applying several **architectural constraints**

REST Architecture

According to **Roy Fielding** [Architectural Styles and the Design of Network-based Software Architectures, 2000]:

- Client-Server
- Stateless
- Uniform Interface:
 - Identification of resources
 - Manipulation of resources through representations
 - Self-descriptive messages
 - Hypermedia as the engine of application state (HATEOAS)
- Layered System
- Code on Demand (optional)

Representational State Transfer(REST)

- ❖ REpresentational State Transfer (REST) is an architecture for accessing distributed hypermedia web-services
- ❖ The resources are identified by URIs and are accessed and manipulated using an HTTP interface base methods (GET, POST, PUT, DELETE, OPTIONS, HEAD, PATCH)
- ❖ Information is exchanged using representations of these resources
- ❖ Lightweight alternative to SOAP+WSDL -> HTTP + Any representation format (e.g. JavaScript™ Object Notation – JSON)

Representational State Transfer(REST)

- ❖ Identification of resources – URIs
- ❖ Representation of resources – e.g. HTML, XML, JSON, etc.
- ❖ Manipulation of resources through these representations
- ❖ Self-descriptive messages - Internet media type (**MIME type**) provides enough information to describe how to process the message. Responses also explicitly indicate their **cacheability**.
- ❖ Hypermedia as the engine of application state (aka **HATEOAS**)
- ❖ Application contracts are expressed as **media types** and **[semantic] link relations** (**rel** attribute - RFC5988, "Web Linking")

Hypermedia As The Engine Of Application State (HATEOAS) – New Link Header (RFC 5988) Example

Content-Length →1656

Content-Type →application/json

Link →<http://localhost:8080/polling/resources/polls/629>;
rel="prev"; type="application/json"; title="Previous poll",
<http://localhost:8080/polling/resources/polls/632>;
rel="next"; type="application/json"; title="Next poll",
<http://localhost:8080/polling/resources/polls>;
rel="collection"; type="application/json"; title="Polls
collection", <http://localhost:8080/polling/resources/polls>;
rel="collection up"; type="application/json"; title="Self
link", <http://localhost:8080/polling/resources/polls/630>;
rel="self"

Example: URLs + HTTP Methods

Uniform Resource Locator (URL)	GET	PUT	POST	DELETE
Collection, such as http://api.example.com/comments/	List the URIs and perhaps other details of the collection's members.	Replace the entire collection with another collection.	Create a new entry in the collection. The new entry's URI is assigned automatically and is usually returned by the operation.	Delete the entire collection.
Element, such as http://api.example.com/comments/11	Retrieve a representation of the addressed member of the collection, expressed in an appropriate Internet media type.	Replace the addressed member of the collection, or if it does not exist, create it.	Not generally used. Treat the addressed member as a collection in its own right and create a new entry in it.	Delete the addressed member of the collection.

Source: https://en.wikipedia.org/wiki/Representational_state_transfer

Advantages of REST

- ❖ Scalability of component interactions – through layering the client server-communication and enabling load-balancing, shared caching, security policy enforcement;
- ❖ Generality of interfaces – allowing simplicity, reliability, security and improved visibility by intermediaries, easy configuration, robustness, and greater efficiency by fully utilizing the capabilities of HTTP protocol;
- ❖ Independent development and evolution of components, dynamic evolvability of services, without breaking existing clients.
- ❖ Fault tolerant, Recoverable, Secure, Loosely coupled

Richardson's Web Maturity Model

According to **Leonard Richardson** [Talk at QCon, 2008 – <http://www.crummy.com/writing/speaking/2008-QCon/act3.html>]:

- ❖ **Level 0 – POX**: Single URI (XML–RPC, SOAP)
- ❖ **Level 1 – Resources**: Many URIs, Single Verb (URI Tunneling)
- ❖ **Level 2 – HTTP Verbs**: Many URIs, Many Verbs (CRUD – e.g Amazon S3)
- ❖ **Level 3 – Hypermedia Links Control the Application State = HATEOAS** (Hypertext As The Engine Of Application State) == **truely** RESTful Services

RESTful Patterns and Best Practices

According to **Cesare Pautasso**

[\[http://www.jopera.org/files/SOA2009-REST-Patterns.pdf\]](http://www.jopera.org/files/SOA2009-REST-Patterns.pdf):

- Uniform Contract
- Content Negotiation
- Entity Endpoint
- Endpoint Redirection
- Distributed Response Caching
- Entity Linking
- Idempotent Capability

REST Antipatterns and Worst Practices

According to **Jacob Kaplan-Moss**

[\[http://jacobian.org/writing/rest-worst-practices/\]](http://jacobian.org/writing/rest-worst-practices/):

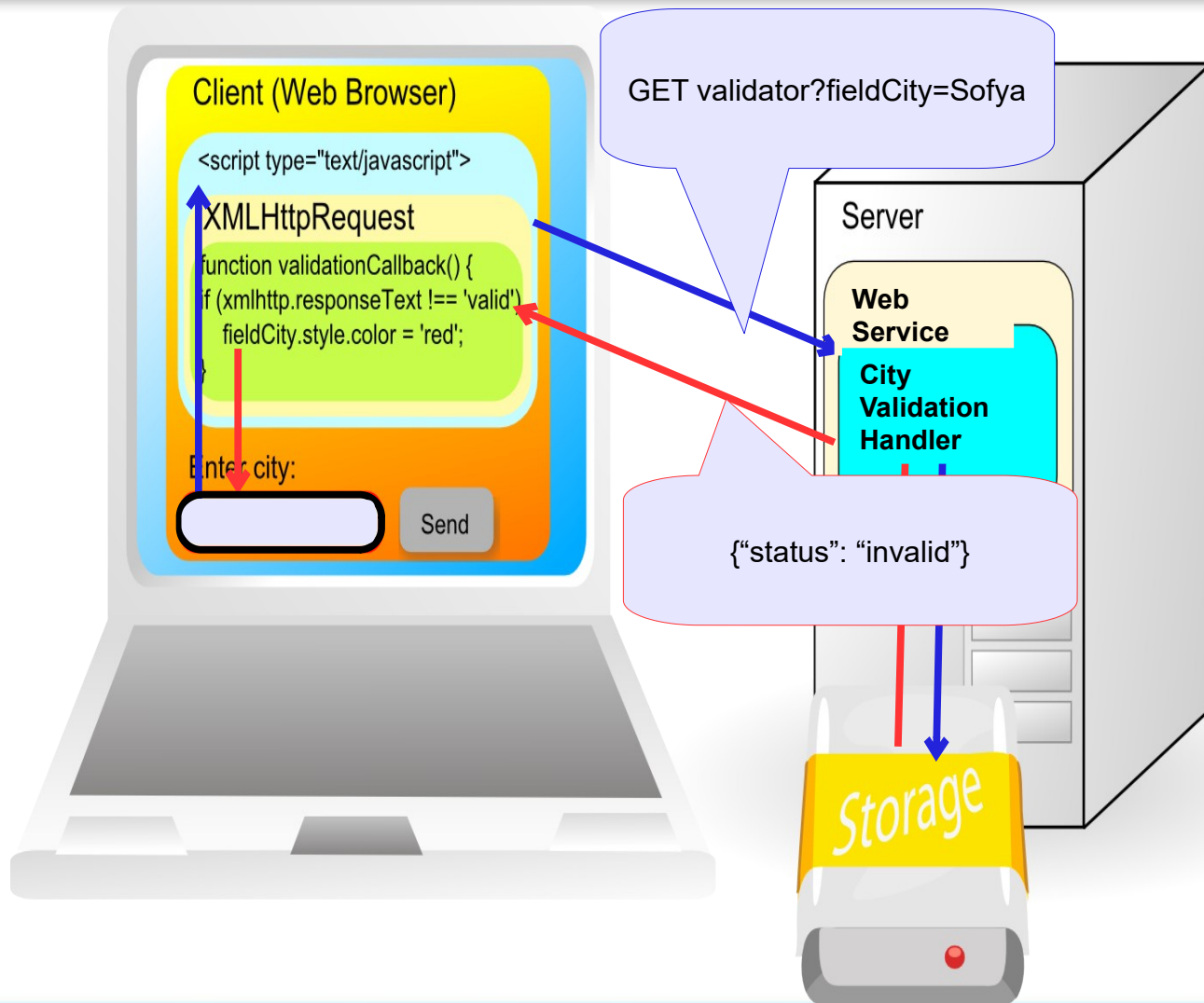
- Conflating models and resources
- Hardcoded authentication
- Resource-specific output formats
- Hardcoded output formats
- Weak HTTP method support (e.g. tunnel everything through GET/POST)
- Improper use of links
- Couple the REST API to the application

AJAX and Traditional Web Applications

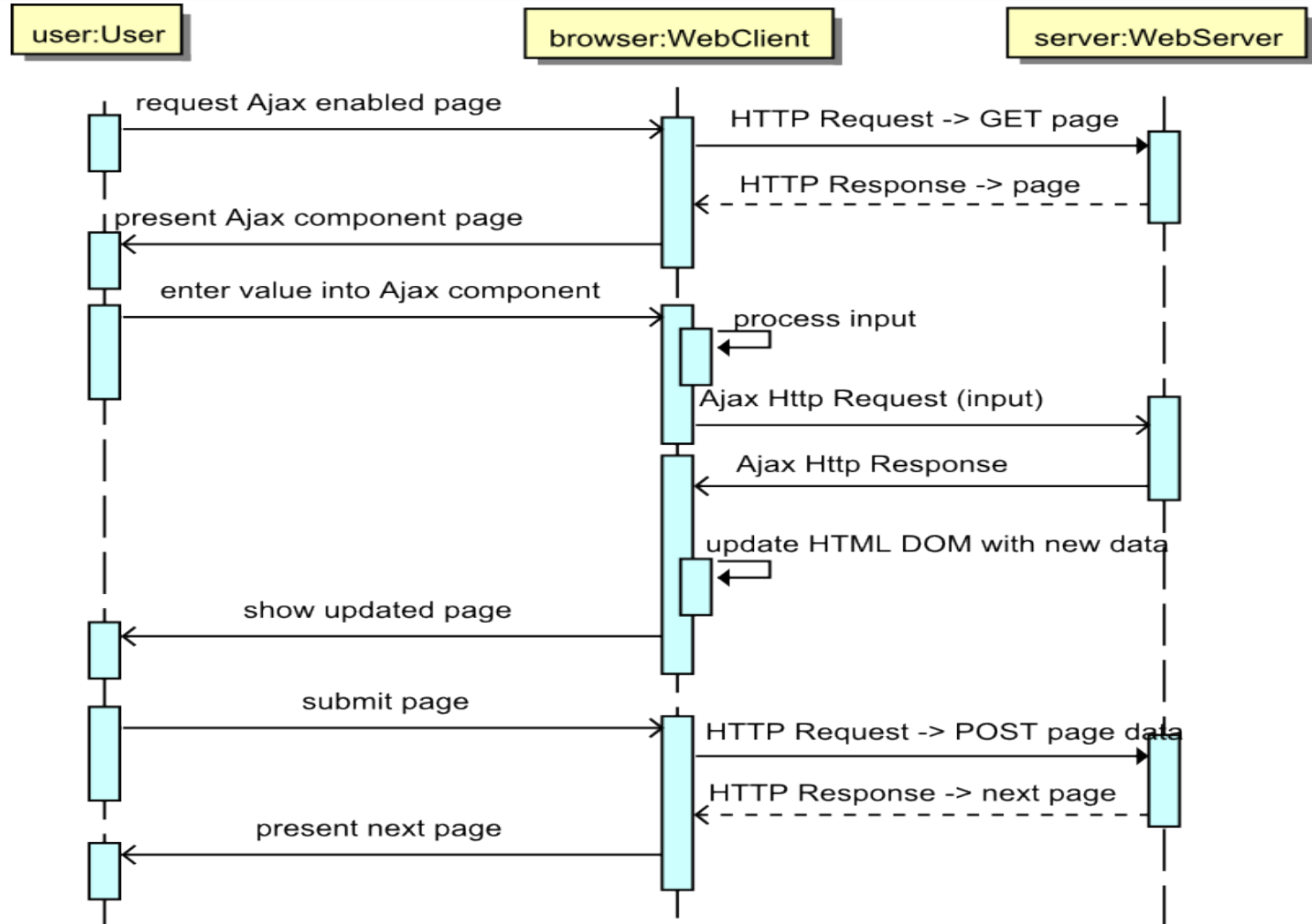
Main difference:

- **Ajax apps** are based on processing of **events** and **data**
- **Traditional web applications** are based on presenting **pages** and **hyperlink transitions** between them

Asynchronous JavaScript and XML (AJAX)



Asynchronous JavaScript and XML - AJAX



Basic Structure of **Asynchronous** AJAX Request

```
if (window.XMLHttpRequest) { // IE7+, Firefox, Safari, Chrome, Opera,  
    xmlhttp=new XMLHttpRequest();  
} else { // IE5, IE6  
    xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");  
}  
xmlhttp.onreadystatechange = function(){  
    if (xmlhttp.readyState==4 && xmlhttp.status==200){  
        callback(xmlhttp);  
    }  
}  
xmlhttp.open(method, url, true);  
xmlhttp.setRequestHeader("Content-type","application/x-www-form-  
urlencoded");  
xmlhttp.send(paramStr);
```

Callback function

isAsynchronous = **true**

XMLHttpRequest.readyState

Code	Meaning
1	After the XMLHttpRequest.open() has been called successfully
2	HTTP response headers have been successfully received
3	HTTP response content loading started
4	HTTP response content has been loaded successfully

Fetch API

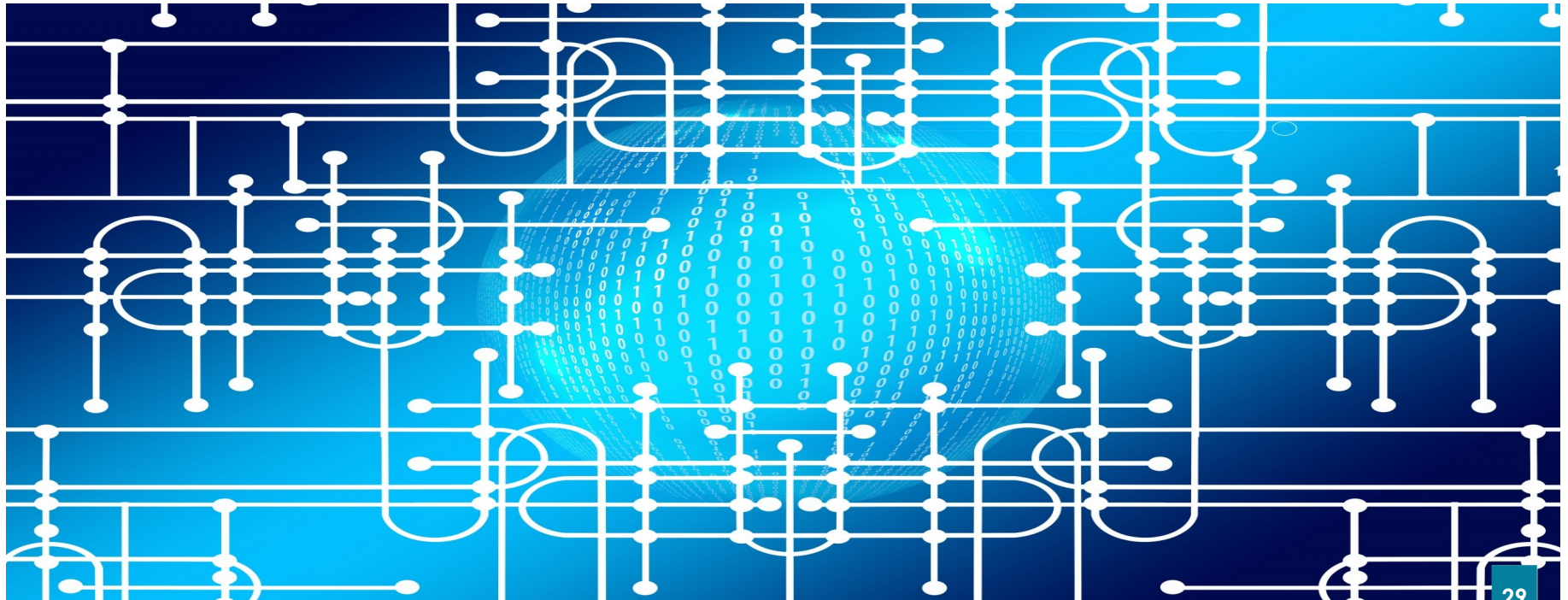
[https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API ,
<https://blog.logrocket.com/axios-or-fetch-api/>]

- The **Fetch API** provides an interface for fetching resources like `XMLHttpRequest`, but more powerful and flexible feature set.
- `Promise<Response> WorkerOrGlobalScope.fetch(input[, init])`
 - `input` – URI of the resource that you wish to fetch – url string or Request
 - `init` - custom settings that you want to apply to the request:
 - **method**: (e.g., GET, POST),
 - **headers**,
 - **body**: (Blob, BufferSource, FormData, URLSearchParams, or USVString),
 - **mode**: (cors, no-cors, or same-origin),
 - **credentials**: (omit, same-origin, or include. to automatically send cookies this option must be provided),
 - **cache**: (default, no-store, reload, no-cache, force-cache, or only-if-cached),
 - **redirect**: (follow, error or manual),
 - **referrer**: (default is client),
 - **referrerPolicy**: (no-referrer, no-referrer-when-downgrade, origin, origin-when-cross-origin, unsafe-url),
 - **integrity**: (subresource integrity value of request)

Fetch Demo Using Async – Await

```
async function init() {  
  try {  
    const userResult = await fetch("user.json");  
    const user = await userResult.json();  
    const gitResp = await fetch(`http://api.github.com/users/{user.name}`);  
    const githubUser = await gitResp.json();  
    const img = document.createElement("img");  
    img.src = githubUser.avatar_url;  
    document.body.appendChild(img);  
    await new Promise((resolve, reject) => setTimeout(resolve, 6000));  
    img.remove();  
    console.log("Demo finished.");  
  } catch (err) { console.log(err); }  
}
```

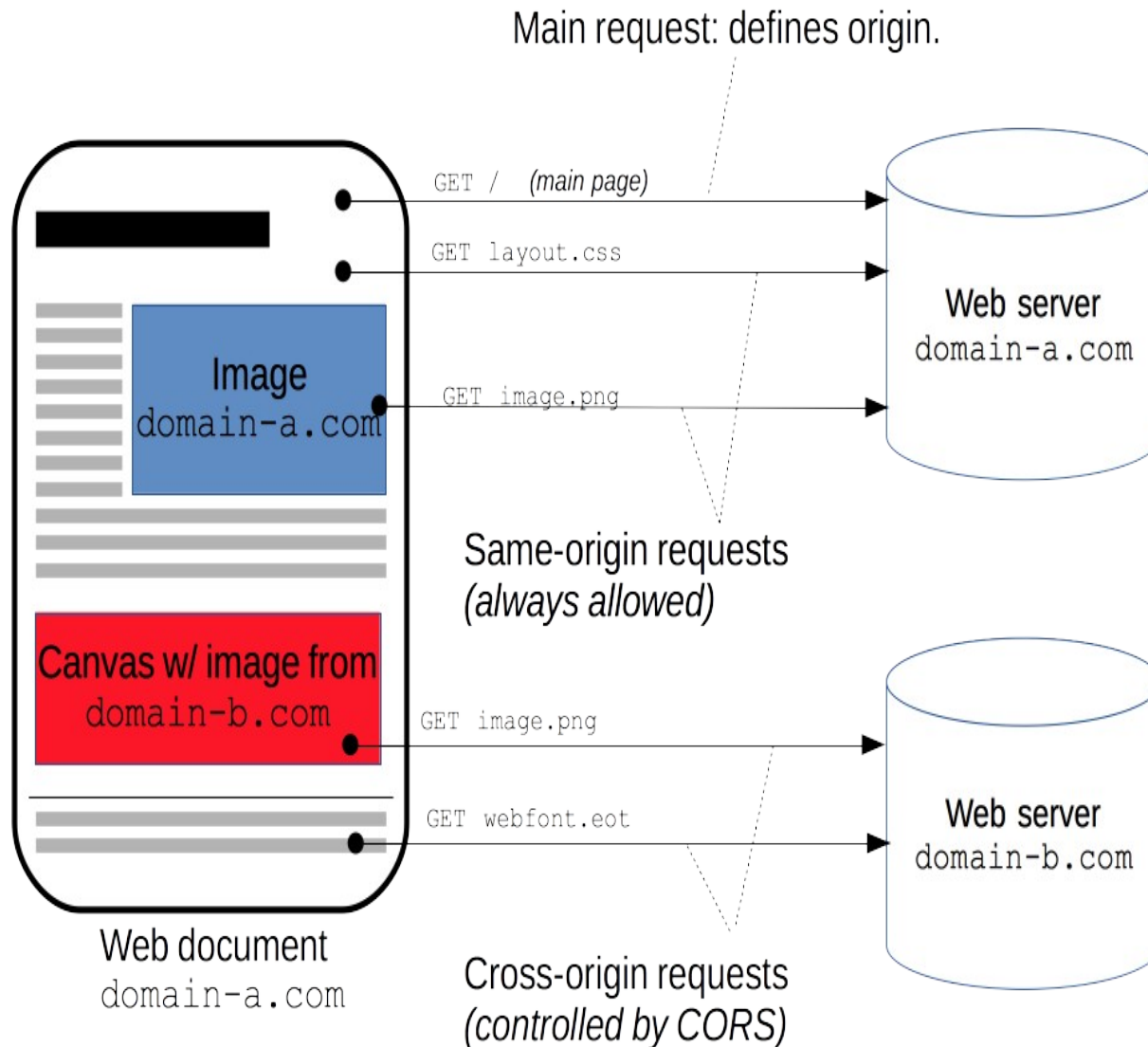

Cross-Origin Resource Sharing (CORS)



Cross-Origin Resource Sharing (CORS)

- Enables execution of **cross-domain requests** for web resources by allowing the server to decide which scripts (from which domains – **Origin**) are allowed to receive/manipulate such resources, as well as which HTTP Methods (GET, POST, PUT, DELETE, etc.) are allowed.
- In order to implement this mechanism, when the HTTP methods differs from simple GET, a **preflight OPTIONS request** is issued by the web browser, in response to which the server returns the **allowed methods, custom headers**, etc. for the requested web resource and script Origin.

Cross-Origin Resource Sharing (CORS)



Cross-Origin Resource Sharing (CORS)

- Allows serving requests to domains that are different from the domain of the script is loaded from.
- The server decides which requests to serve, based on the **Origin** of the script issuing the request, the method (**GET, POST**, etc.), **credentials/cookies**, and the **custom headers** to be sent.
- When the method is different from **simple GET, HEAD or POST**, or the **Content-Type** is different from **application/x-www-form-urlencoded, multipart/form-data**, or **text/plain**, a **preflight OPTIONS request** is mandated by the specification.
- In response to **preflight OPTIONS request**, the server should return which **origins, methods, headers, credentials/cookies** are allowed in cross-domain requests to that server, and for **how long**.

New HTTP CORS Headers – Simple Request

- HTTP GET request:

GET /crossDomainResource/ HTTP/1.1

Referer:

http://sample.com/crossDomainMas
hup/

Origin: http://sample.com

- HTTP GET response:

Access-Control-Allow-Origin:
http://sample.com

Content-Type: application/xml

CORS - Simple Request:



New HTTP CORS Headers – PUT / DELETE/ Custom Content/Headers

- HTTP OPTIONS preflight request:

OPTIONS /crossDomainPOSTResource/ HTTP/1.1

Origin: http://sample.com

Access-Control-Request-Method: POST

Access-Control-Request-Headers: MYHEADER, Content-Type

- HTTP response:

HTTP/1.1 200 OK

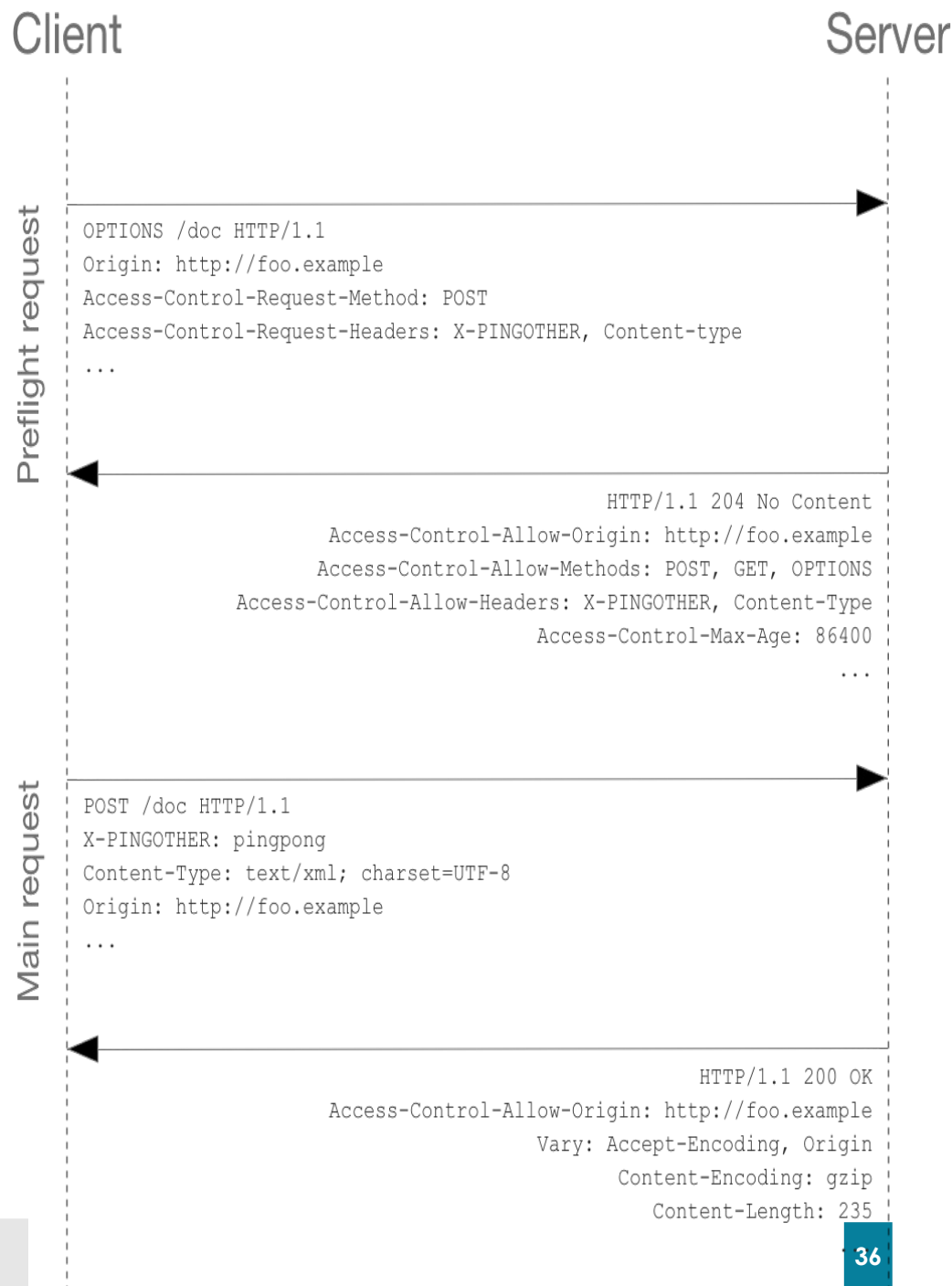
Access-Control-Allow-Origin: http://sample.com

Access-Control-Allow-Methods: POST, GET, OPTIONS

Access-Control-Allow-Headers: MYHEADER, Content-Type

Access-Control-Max-Age: 864000

CORS – POST / PUT / DELETE/ Custom Content/Headers:



Thank's for Your Attention!



Trayan Iliev

**CEO of IPT – Intellectual Products
& Technologies**

<http://iproduct.org/>

<http://robolearn.org/>

<https://github.com/iproduct>

<https://twitter.com/trayaniliev>

<https://www.facebook.com/IPT.EACAD>

<https://plus.google.com/+IproductOrg>