Web Applications 2017

Week 3, Slides 1

Content

- Web Service
- Representational State Transfer, REST
- Node/Express REST
- JEE, REST
- HATEOAS
- AJAX
- CORS

eqweq

Web Service

"A web service is a software system designed to support interoperable machine-to-machine interaction over a network." // W3c

Types
- SOAP
- REST
- ...

We only use REST

Readings
- SOAP vs REST

Background

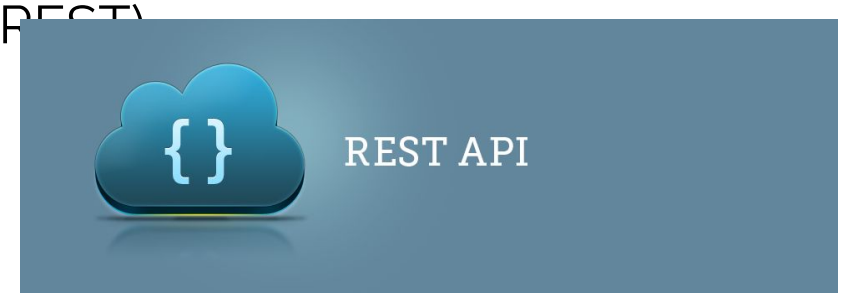
The Web is a marvelous “application”

- Has been up 24/7 for 30-40 years
- Has been able to expand many magnitudes
- More users, more data, more advanced services … the perfect application?

Wouldn’t it be good to build our application like that??

- So what are the key principles behind the Web?

Representational State Transfer (REST)



1. Identification of resources (nouns)
2. Manipulating of resources through representations
3. Self-descriptive messages (stateless)
4. Hypermedia as the engine of application state (HATEOAS)

Principles identified by Roy Fielding

A web service is a software system designed to support interoperable machine-to-machine interaction over a network.

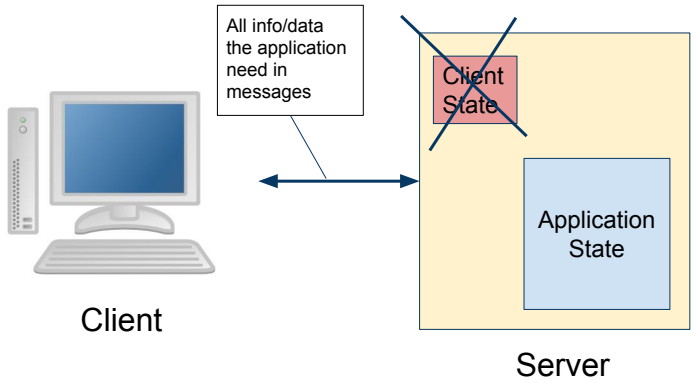
Readings

- Wikipedia, REST
- REST (Roy Fielding)
- HATEOAS

Practical Interpretation of REST

1. All resources accessible with URL's
2. Use JSON, XML or … as representation
3. HTTP is stateless and self descriptive
   (simple unified interface: GET, POST, …)
4. Embed links in response to present the options to the client

Stateless … What state?



Our application will have state, so how is REST stateless?

- Answer: No client state!
  - I.e no session on server
- Everything needed must be in request (also authorization, more later …)

Testing some RESTful Services

These need no API key

- Flickr Photo Service (try change format, last)
- Google Maps (try change coordinates)
- Konvert.me (try change IP)

Need API key

- Google Maps Driving Directions

Many public RESTful services available.

Readings

- Public APIs
- Programmable web
- API-key

Designing a REST API

Representation
"Accept: application/json" or
"Accept: application/xml" or ...

```
# Person registry service
# Get all persons
GET: http://localhost:8080/person/list

# Get single person
GET: http://localhost:8080/person/list/1

# Create new person
POST: http://localhost:8080/person/list --data "name=bertil"

# Update person
PUT: http://localhost:8080/person/list/1 --data "name=sara"

# Delete person
DELETE: http://localhost:8080/person/list/1

# Get primitive type
GET: http://localhost:8080/person/list/count
```

Readings

- Best Practices for REST API

REST Status Responses

Help the API consumers route their responses accordingly

- 200 OK - Response to a successful GET, PUT, PATCH or DELETE. Can also be used for a POST that doesn't result in a creation.
- 201 Created - Response to a POST that results in a creation. Should be combined with a Location header pointing to the location of the new resource
- 204 No Content - Response to a successful request that won't be returning a body (like a DELETE request)
- … more

What do we need Server side?

Something that can handle

- The verbs
- URL's
- Consume and Produce representations
- Return status responses

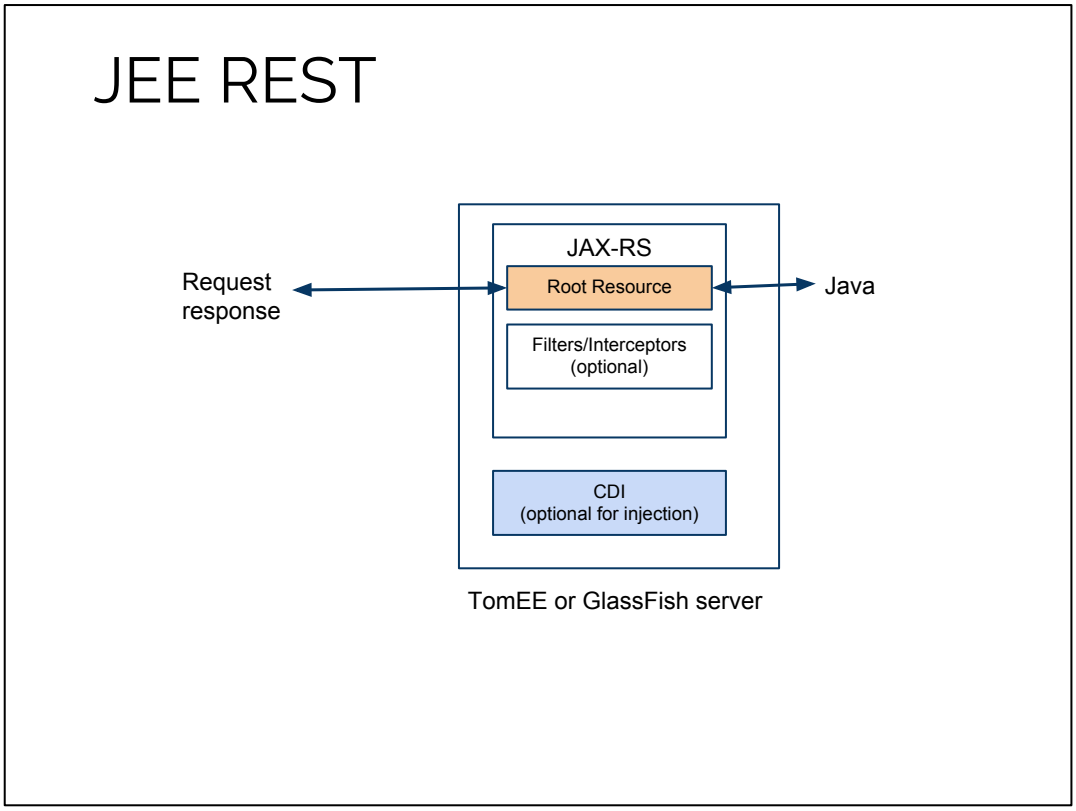
Node Express

- The verbs
  - Yes
- URL's
  - Yes, URL's possibly patterns (reg exp's)
  - URL may have parameters
- Consume and Produce representations
  - Yes, using header fields, send json, xml, ...
- Return status responses
  - Yes

Express Alternatives

Frameworks built on Express

- Not tested



Java API for RESTful Web Services, JAX-RS 2.0

Readings

- JAX-RS Jersey developer guide

Root Resource Class

```
@Path("persons")
public class PersonResource {

    @GET
    @Produces({MediaType.APPLICATION_JSON})
    public Response findAll() {
        return Response.ok(gson.toJson(persons)).build();
    }

    @POST
    @Consumes({MediaType.APPLICATION_FORM_URLENCODED})
    public Response add(@FormParam("name") String name) {
        // Set location and return 201
        UriBuilder ub = uriInfo.getAbsolutePathBuilder();
        ub.path(name);
        return Response.created(ub.build()).build();
    }
}
```

Also need ApplicationConfig class (generated)

Will be request scoped!

Also @PathParam @QueryParam

Root resource, the top level resource class, a class to handle HTTP requests (cousin to Servlet)

Readings

- JAX-RS Resources

Context

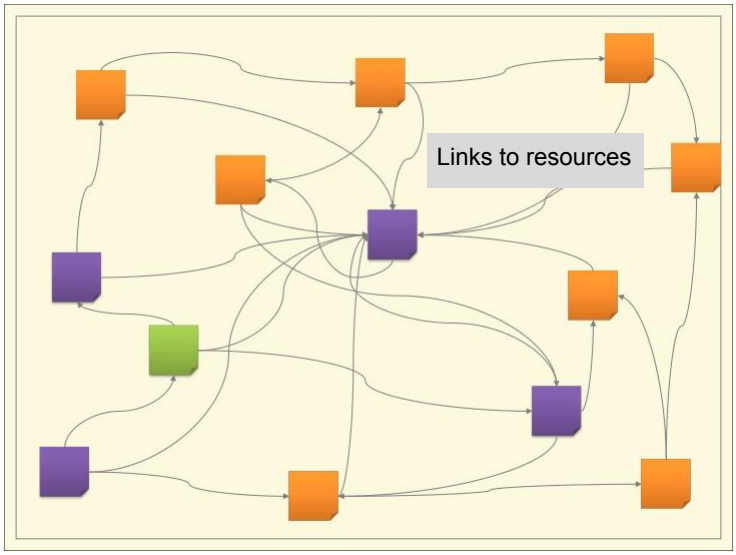
Possible to inject "low level" objects in resource classes using @Context annotation

```
@Context
private HttpHeaders headers;

@Context
private Request request;

@Context
private UriInfo uriInfo
```

HATEOAS



HATEOAS means that hypertext should be used to find your way through the API.

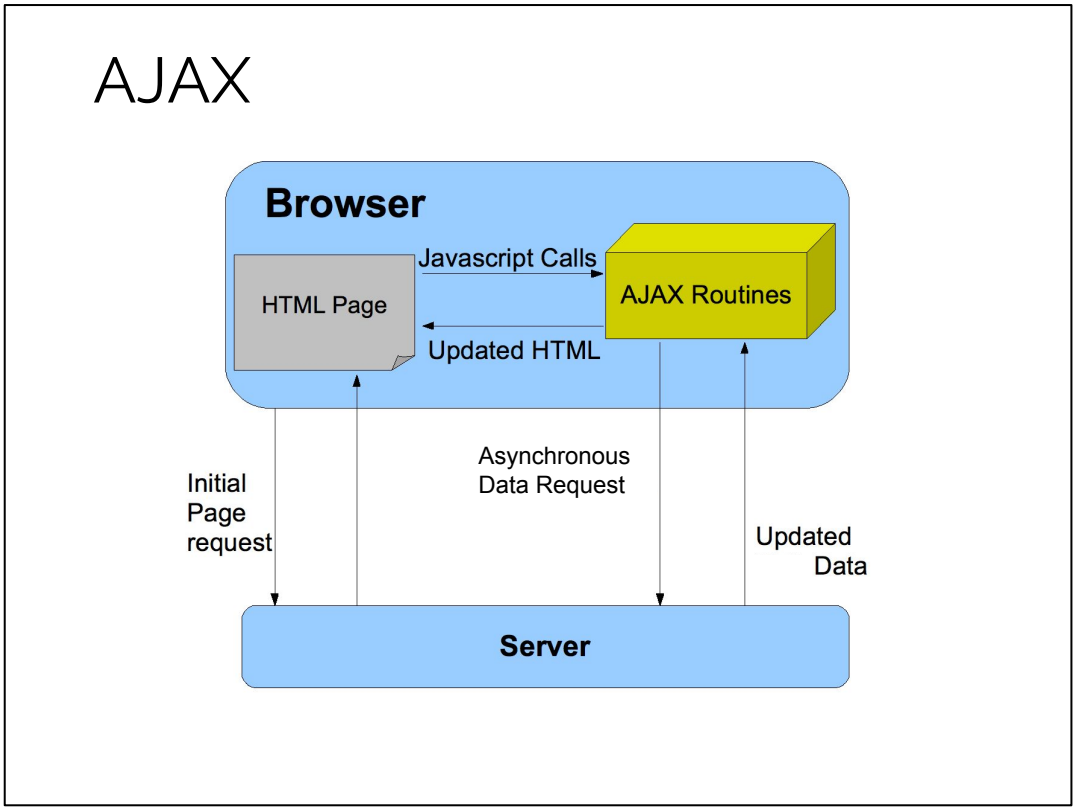
- The GET response contains requested data and …
- … links to associated data.
- API is self exposing! The hypertext is actually telling us what is allowed and what not

Implementing HATEOAS

```
GET /account/12345

HTTP/1.1 HTTP/1.1 200 OK
<?xml version="1.0"?>
<account>
    <account_number>12345</account_number>
    <balance currency="usd">100.00</balance>
    <link rel="deposit" href="/account/12345/deposit" />
    <link rel="withdraw" href="/account/12345/withdraw" />
    <link rel="transfer" href="/account/12345/transfer" />
    <link rel="close" href="/account/12345/close" />
</account>
```

Using XML



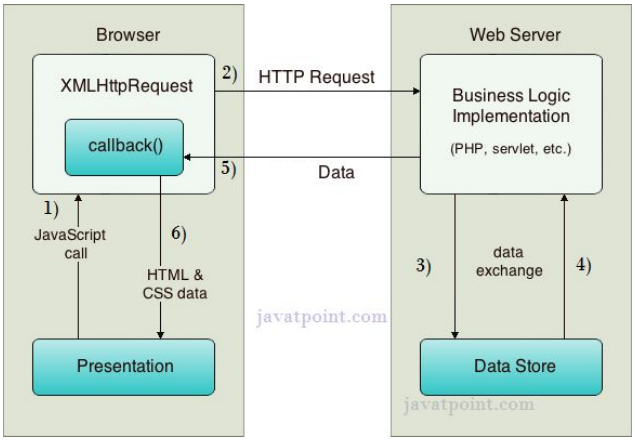
AJAX (or AJAX, short for asynchronous JavaScript and XML)

- Is a group of interrelated Web development techniques used on the client-side to create asynchronous Web applications.
- With Ajax, web applications can send data to and retrieve from a server asynchronously (in the background) without interfering with the display and behavior of the existing page

Readings

- AJAX MDN

XMLHttpRequest



XMLHttpRequest is a standard JS AJAX (low level) API. We don't use directly.

Readings

- XMLHttpRequest MDN

jQuery AJAX

GET (HTML)

```
$.load( …  URL … );
```

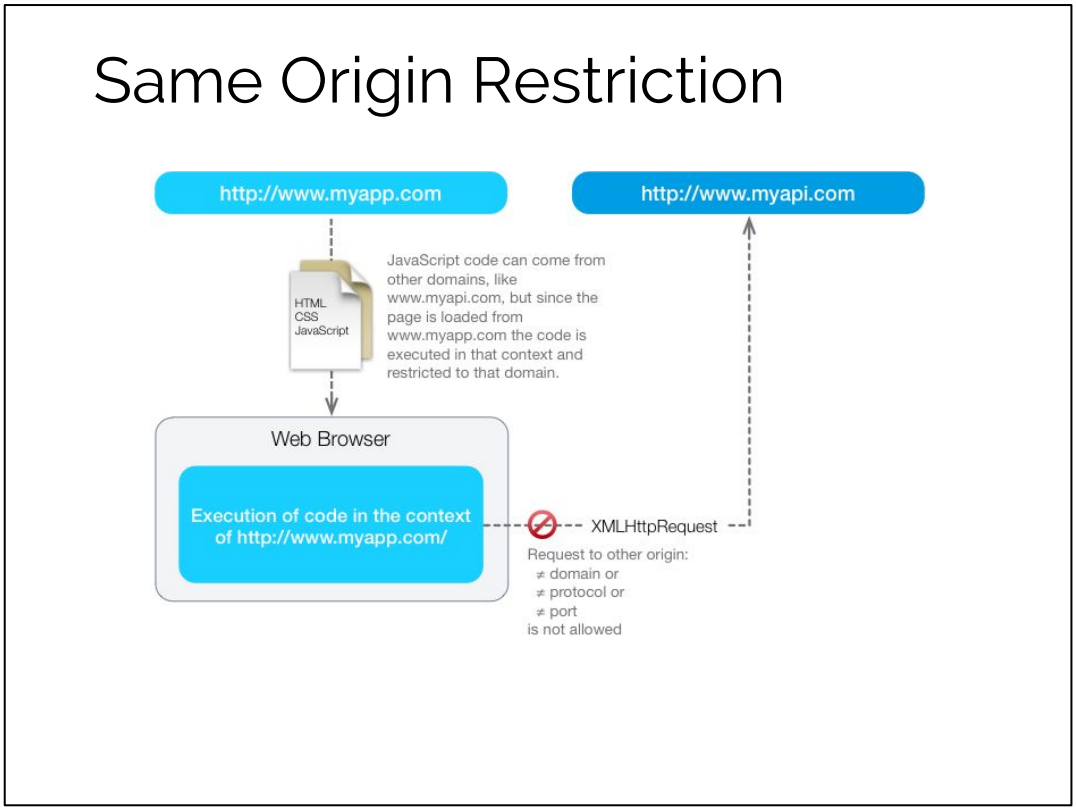
Get JSON

```
$.getJSON( … URL … );
```

Low level API

```
$.ajax({
  url: "test.html",
  context: document.body
}).then(function() {
  $( this ).addClass( "done" );
});
```

We use jQuery, promise based AJAX API (jQuery uses “deferreds” similar to Promises)

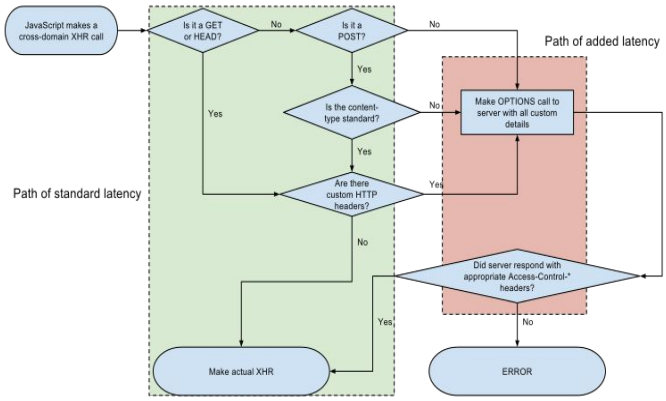


User agents allow content retrieve from one origin to interact freely with other content retrieved from that origin, but user agents restrict how that content can interact with content from another origin.

Readings

- Same origin policy Wikipedia
- Same origin policy
- Web origin

CORS



Better image

Readings

- CORS Wikipedia
- CORS W3C
- JSONP, hack to circumvent restrictions.

CORS Headers

```
// HTTP Response headers for CORS
response.header("Access-Control-Allow-Methods",
               "POST, GET, OPTIONS, PUT, DELETE");
response.header("Access-Control-Allow-Origin", "*");
response.header("Access-Control-Allow-Headers",
            "X-Requested-With, Content-Type");
```

Readings

- CORS MDN
- CORS W3C
- JSONP, hack to circumvent restrictions.