DOM Event Handling. HTTP Clients. REST. Novelties in ECMAScript 6. Webpack 2/3

Trayan Iliev

IPT – Intellectual Products & Technologies
e-mail: tiliev@iproduct.org
web: http://www.iproduct.org

Agenda - I

1. JavaScript HTML DOM – Document Object Model (DOM) Object tree, W3C DOM standard Core DOM and HTML DOM
2. DOM objects, properties, methods and events.
3. DOM Events and event listeners. Browser event models – DOM Level 0, Traditional model (using properties), DOM Level 2, and Microsoft event handling models.
4. Scheduling asynchronous behaviors (setInterval(), setTimeout(), clearInterval(), clearTimeout()).
5. Working with forms and validation – Forms API
6. HTTP Client API – AJAX requests using XMLHttpRequest, HTTP request/response methods, headers and content types

Agenda - II

7. Practical HTTP Client programming using jQuery. jQuery Deferred and ES6 Promises, AJAX + JSON, JSON with Padding (JSONP)
8. Axios - promise based HTTP client for the browser and Node.js
9. Novelties in ECMAScript 6 (ECMAScript 2015, Harmony) – class and constructor syntax, let and var, function lambdas (=>), Promises
10. Bootstrapping an ES6 project using Webpack 2 and Babel
11. Writing reusable components as ES6 classes

Where is The Code?

JavaScript Application Programming code is available @GitHub:

https://github.com/iproduct/Course-Multimedia-FMI

Event Handling Models in JavaScript

- DOM Level 0 (original Netscape model)

```
<a href="#" onclick= "alert('I\'m clicked!'); return false;" />
```

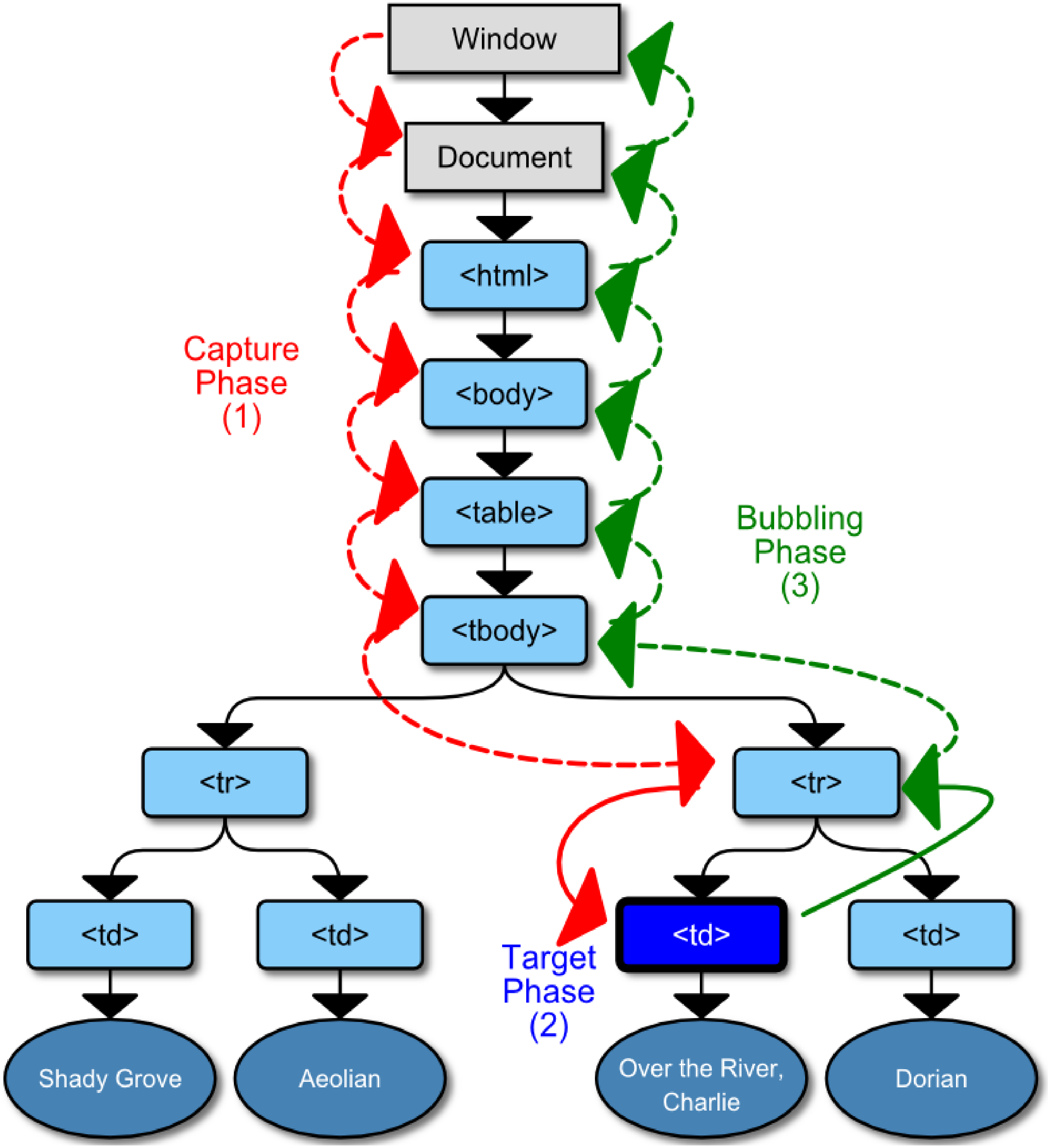
- Traditional model (as properties)

```
anElem.onclick = function() { this.style.color = 'red'; }
```

- can register multiple event handlers:

```
var oldHandler = (anElem.onclick) ? anElem.onclick : function (){ };
anElem.onclick = function () {oldHandler(); this.style.color = 'red'; };
```

- Microsoft Event Handling Model
- DOM Level 2 Event Handling Model
- DOM Level 3 Event Handling Model



Source: UI Events W3C Working Draft, 04 August 2016, https://www.w3.org/TR/DOM-Level-3-Events/, Copyright © 2016 W3C® (MIT, ERCIM, Keio, Beihang). W3C liability, trademark and document use rules apply.

W3C DOM Level 2 Event Handling Model

- Three phases in event handling life-cycle:
  - Capturing phase – from document to target element
  - At Target phase – processsing in the target element
  - Bubbling phase – returns back from target to document
- All events go through Capturing phase, but not all through Bubbling phase – only low level (raw) events
- event.stopPropagation() - stops further processing
- event.preventDefault() - prevents standards event processing
- Register/deregister event handlers:

```
anElement.addEventListener('click', eventListener, false)
anElement.removeEventListener('click', eventListener, false)
```

Microsoft Event Handling Model

- Register/deregister event handlers:

```
anElement.attachEvent('onclick', eventListener)
anElement.detachEvent('onclick', eventListener)
```

- Callback function *eventListener* does not receive *event* object:

```
function crossBrowserEventHandler(event) {
  if(!event) event = window.event;  ... // processing follows ... }
```

- No Capturing phase – every element has methods setCapture() and releaseCapture()
- from document towards target element
- window.event.cancelBubble = true; // stops bubbling -a
- window.event.returnValue=false; // prevents default action

W3C DOM Level 2 Events and APIs

Име на интерфейса	Събития
Event	abort, blur, change, error, focus, load, reset, resize, scroll, select, submit, unload
MouseEvent	click, mousedown, mousemove, mouseout, mouseover, mouseup
UIEvent	DOMActivate, DOMFocusIn, DOMFocusOut

Asynchronous JavaScript & XML - AJAX

- Ajax – A New Approach to Web Applications, J. Garrett February, 2005

  http://www.adaptivepath.com/publications/essays/archives/000385.php
- Presentation based on standards HTML 5 / XHTML, CSS
- Dynamic visualisation and interaction using Document Object Model (DOM)
- Exchange and manipulation of data using XML and XSLT or JavaScript Object Notation (JSON)
- Asynchronous data fetch using **XMLHttpRequest**
- And JavaScript who wrapps everything above in one application

AJAX and Traditional Web Applications

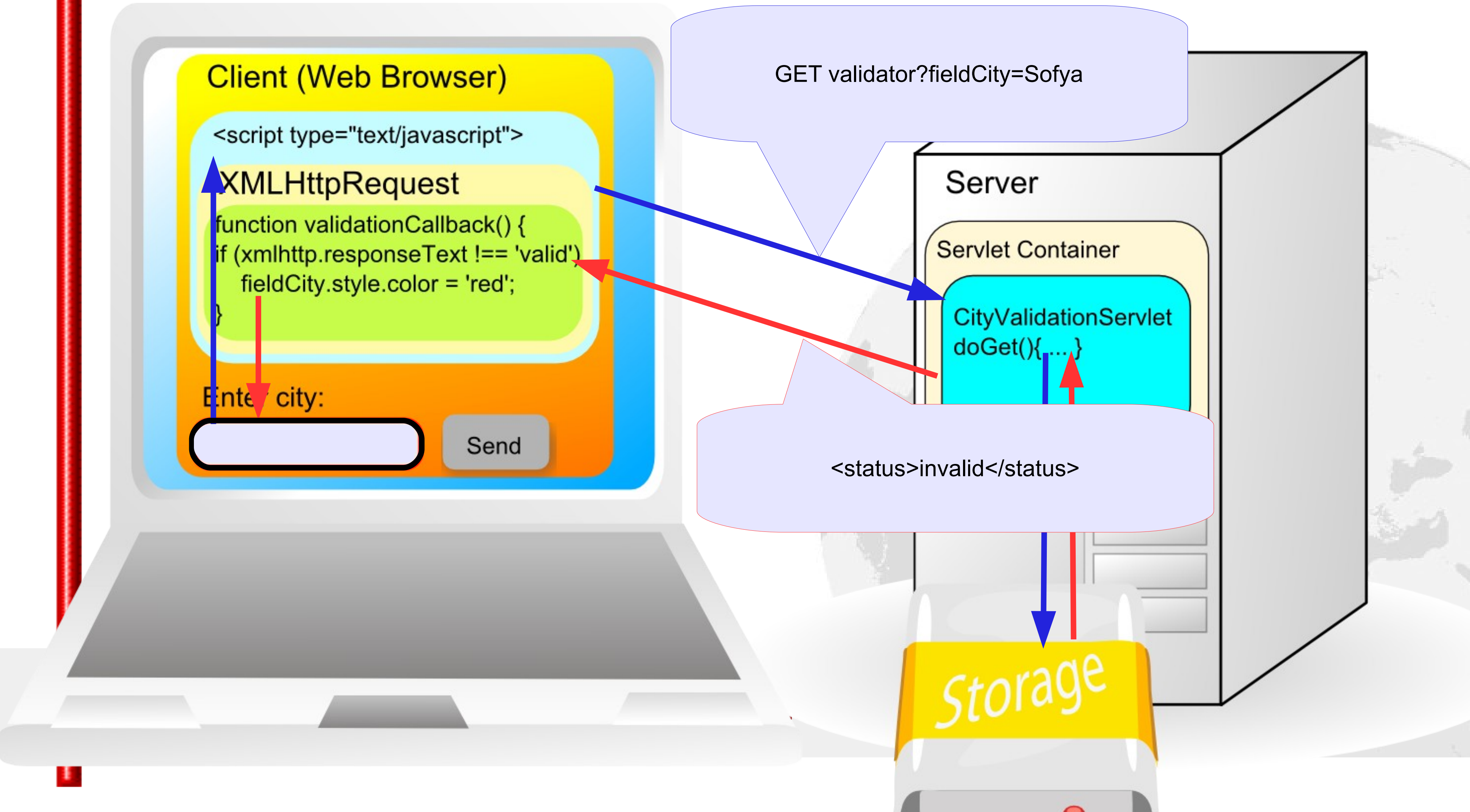
Main difference:

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

Problems connected with AJAX (1)

- Sandboxing
- Scripting switched off
- Spped of client processing
- Time for script download
- Loosing integrity
- Search engine indexing
- Accessibility
- More complex development
- More complex profiling – 2 cycles
- Cross Domain AJAX

AJAX Interactions



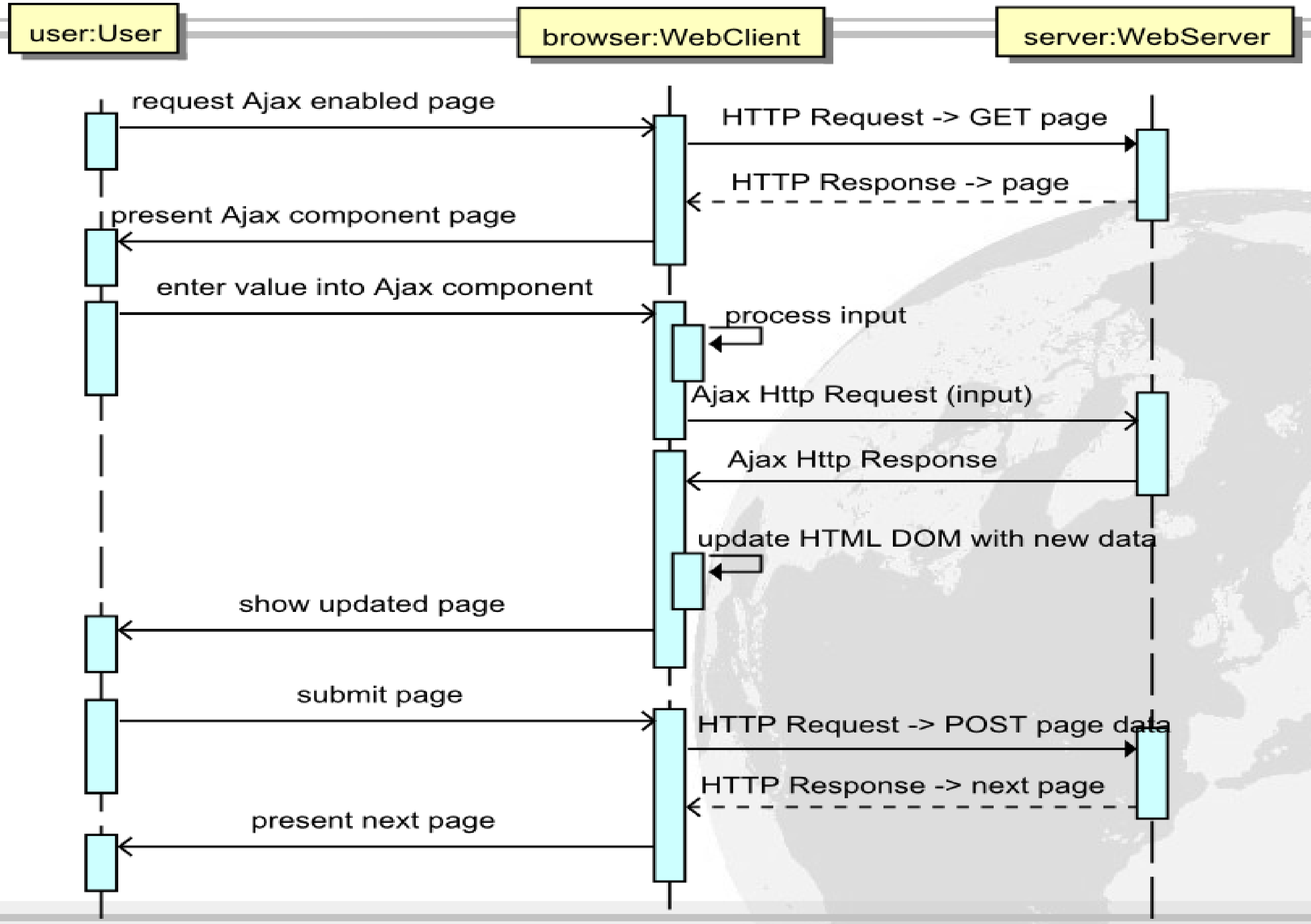
```
<script type="text/javascript">
function validationCallback() {
if (xmlhttp.responseText !== 'valid')
    fieldCity.style.color = 'red';
}
```

```
CityValidationServlet
doGet(){.....}
```

GET validator?fieldCity=Sofya

<status>invalid</status>

AJAX Interactions Flowchart


user:User
browser:WebClient
server:WebServer
request Ajax enabled page
HTTP Request -> GET page
HTTP Response -> page
present Ajax component page
enter value into Ajax component
process input
Ajax Http Request (input)
Ajax Http Response
update HTML DOM with new data
show updated page
submit page
HTTP Request -> POST page data
HTTP Response -> next page
present next page

Basic Structure of Synchronous AJAX Request

```
var method = “GET”;
var url = “resources/ajax_info.html”;

if (window.XMLHttpRequest)   {// IE7+, Firefox, Safari, Chrome, Opera,
        xmlhttp=new XMLHttpRequest();
    }  else {// IE5, IE6
        xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
    }
}

xmlhttp.open(method, url, false);
xmlhttp.send();
document.getElementById("results").innerHTML =
        xmlhttp.responseText;
```

isAsynchronous = false

AJAX Request with XML Processing and Authentication

```
if (window.XMLHttpRequest)   {// IE7+, Firefox, Safari, Chrome, Opera,
        xmlhttp=new XMLHttpRequest();
    }  else {// IE5, IE6
        xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
    }
}
xmlhttp.open(“GET”, “protected/product_catalog.xml”, false,
                                                          “trayan”, “mypass”);
xmlhttp.send();
if (xmlhttp.status == 200 &&
           xmlhttp.getResponseHeader("Content-Type") == "text/xml") {
        var xmlDoc = xmlhttp.responseXML;
        showBookCatalog(xmlDoc);  // Do something with xml document
    }
}
```

AJAX Request with XML Processing (2)

```
function showBookCatalog(xmlDoc){
        txt="<table><tr><th>Title</th><th>Artist</th></tr>";
    var x=xmlDoc.getElementsByTagName("TITLE");
    var y=xmlDoc.getElementsByTagName("AUTHOR");
    for (i=0;i<x.length;i++) {
        txt=txt +"<tr><td>"
            + x[i].firstChild.nodeValue
            + "</td><td>"+ y[i].firstChild.nodeValue
            + "</td></tr>";
    }
    txt += "</table>"
    document.getElementById("book_results").innerHTML=txt;
}
```

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

Код	Значение
1	след като XMLHttpRequest.open() е извикан успешно
2	заглавните части на отговора на HTTP заявката (HTTP response headers) са успешно получени
3	начало на зреждане на съдържанието на HTTP отговора (HTTP response content)
4	съдържанието на HTTP отговора е заредено успешно от браузъра

Browser Independent AJAX Request

```
function getXMLHTTP() {
    var xmlhttp = null;
   if (typeof XMLHttpRequest != "undefined") {
        xmlhttp = new XMLHttpRequest();
    } else {
        try {
            xmlhttp = new ActiveXObject("Msxml2.XMLHTTP");
        } catch (e) { }
        if (xmlhttp == null) {
            try {
                xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
            } catch (e) { }
        }
    }
    return(xmlhttp);
}
```

HTTP Request Headers

- В **HTTP 1.0** всички заглавни части са опционални
- В **HTTP 1.1** са опционални всички заглавни части без **Host**
- Необходимо е винаги да се проверява дали съответната заглавна част е различна от **null**

HTTP Requests Status Codes - RFC2616

- **Accept**
- **Accept-Charset**
- **Accept-Encoding**
- **Accept-Language**
- **Accept-Language**
- **Authorization**
- **Connection**
- **Content-Length**
- **Cookie**
- **Host**
- **If-Modified-Since**
- **If-Unmodified-Since**
- **Referer**
- **User-Agent**

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",
 }]
```

Response Status Codes

- 100 Continue
- 101 Switching Protocols
- 200 OK
- 201 Created
- 202 Accepted
- 203 Non-Authoritative Information
- 204 No Content
- 205 Reset Content
- 301 Moved Permanently
- 302 Found
- 303 See Other
- 304 Not Modified
- 307 Temporary Redirect
- 400 Bad Request
- 401 Unauthorized
- 403 Forbidden
- 404 Not Found

Response Status Codes

- **405 Method Not Allowed**
- **415 Unsupported Media Type**
- **417 Expectation Failed**
- **500 Internal Server Error**
- **501 Not Implemented**
- **503 Service Unavailable**
- **505 HTTP Version Not Supported**

HTTP Response Headers

- Allow
- Cache-Control
- Pragma
- Connection
- Content-Disposition
- Content-Encoding
- Content-Language
- Content-Length
- Content-Type
- Expires
- Last-Modified
- Location
- Refresh
- Retry-After
- Set-Cookie
- WWW-Authenticate

Axios – Promise HTTP Client for Browser & Node [https://github.com/mzabriskie/axios]

- Make XMLHttpRequests from the browser
- Make http requests from node.js
- Supports the Promise API
- Intercept request and response
- Transform request and response data
- Cancel requests
- Automatic transforms for JSON data
- Client side support for protecting against XSRF

Axios: GET Request Handling
[https://github.com/mzabriskie/axios]

```
function getUserAccount() {
  return axios.get('/user/12345');
}


function getUserPermissions() {
  return axios.get('/user/12345/permissions');
}


axios.all([getUserAccount(), getUserPermissions()])
  .then(axios.spread(function (acct, perms) {
    // Both requests are now complete
```

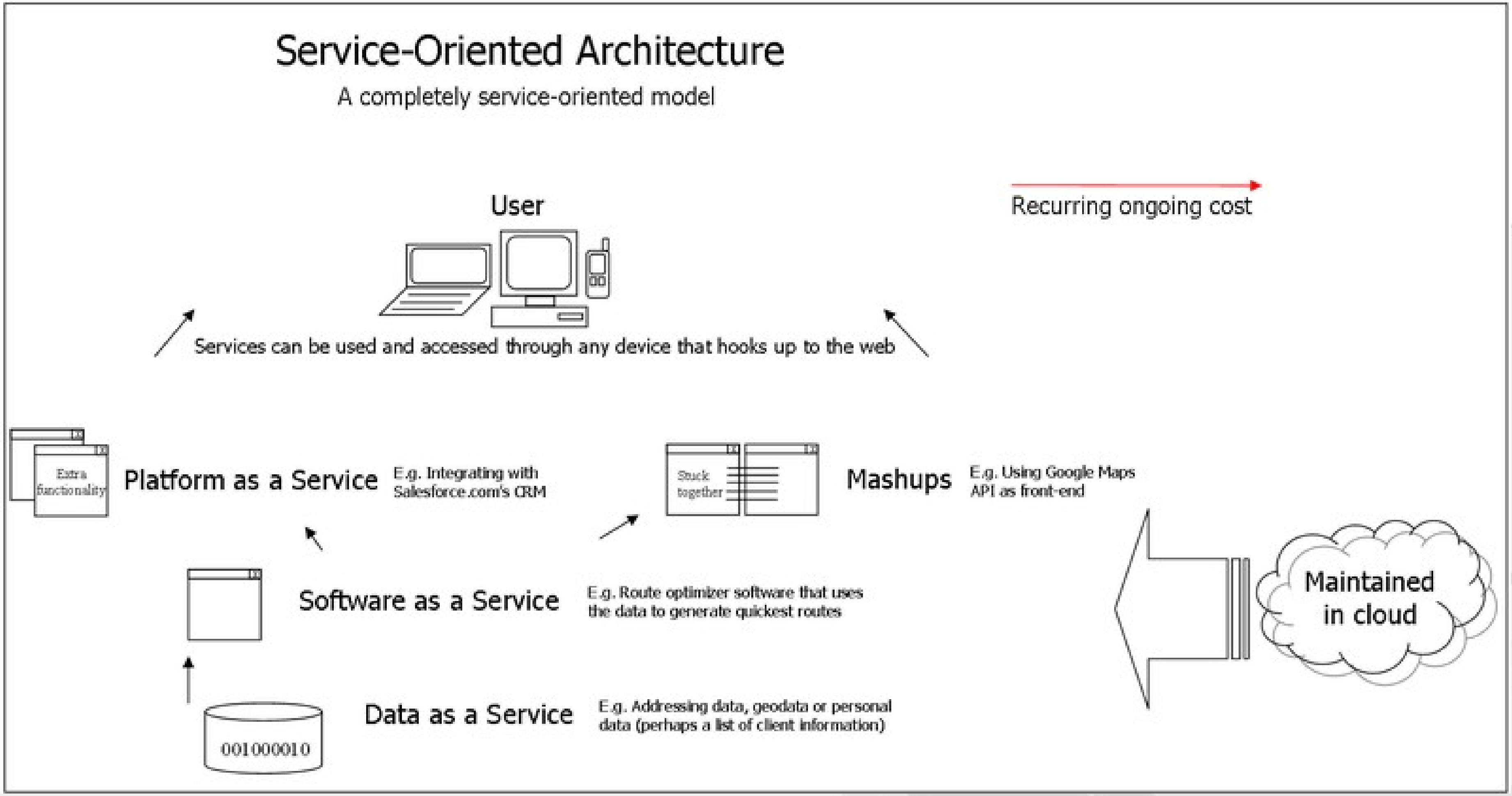
Axios: POST Request Handling [https://github.com/mzabriskie/axios]

```
axios.post('/user', {
    firstName: 'Fred',
    lastName: 'Flintstone'
  })
  .then(function (response) {
    console.log(response);
  })
  .catch(function (error) {
    console.log(error);
  });
```

Axios: Cancelable Promise [https://github.com/mzabriskie/axios]

```
var CancelToken = axios.CancelToken;
var source = CancelToken.source();
axios.get('/user/12345', {
  cancelToken: source.token
}).catch(function(thrown) {
  if (axios.isCancel(thrown)) {
    console.log('Request canceled', thrown.message);
  } else {  // handle error
  }
});
// cancel the request
source.cancel('Operation canceled by the user.');
```

Service Oriented Architecture (SOA)


Service-Oriented Architecture
A completely service-oriented model
User
Recurring ongoing cost
Services can be used and accessed through any device that hooks up to the web
Extra functionality
Platform as a Service
E.g. Integrating with Salesforce.com's CRM
Stuck together
Mashups
E.g. Using Google Maps API as front-end
Software as a Service
E.g. Route optimizer software that uses the data to generate quickest routes
Maintained in cloud
001000010
Data as a Service
E.g. Addressing data, geodata or personal data (perhaps a list of client information)

Source: http://en.wikipedia.org/wiki/File:SOA_Detailed_Diagram.png,
Author: JamesLWilliams2010, License: Creative Commons Attribution 3.0 Unported

REST Architectural Properties

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

- Performance
- Scalability
- Reliability
- Simplicity
- Extensibility
- Dynamic evolvability
- Cusomizability
- 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 Architectural Constraints

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)

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 tolerat, Recoverable, Secure, Loosely coupled

Representational State Transfer (REST) [1]

- 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 HHTP 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) [2]

- 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 realtions (**rel** attribute - RFC5988, "Web Linking")

[Source: http://en.wikipedia.org/wiki/Representational_state_transfer]

Simple Example: URLs + HTTP Methods

Uniform Resource Locator (URL)	GET	PUT	POST	DELETE
Collection, such ashttp://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 ashttp://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*

Richardson's Maturity Model of Web Services

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

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

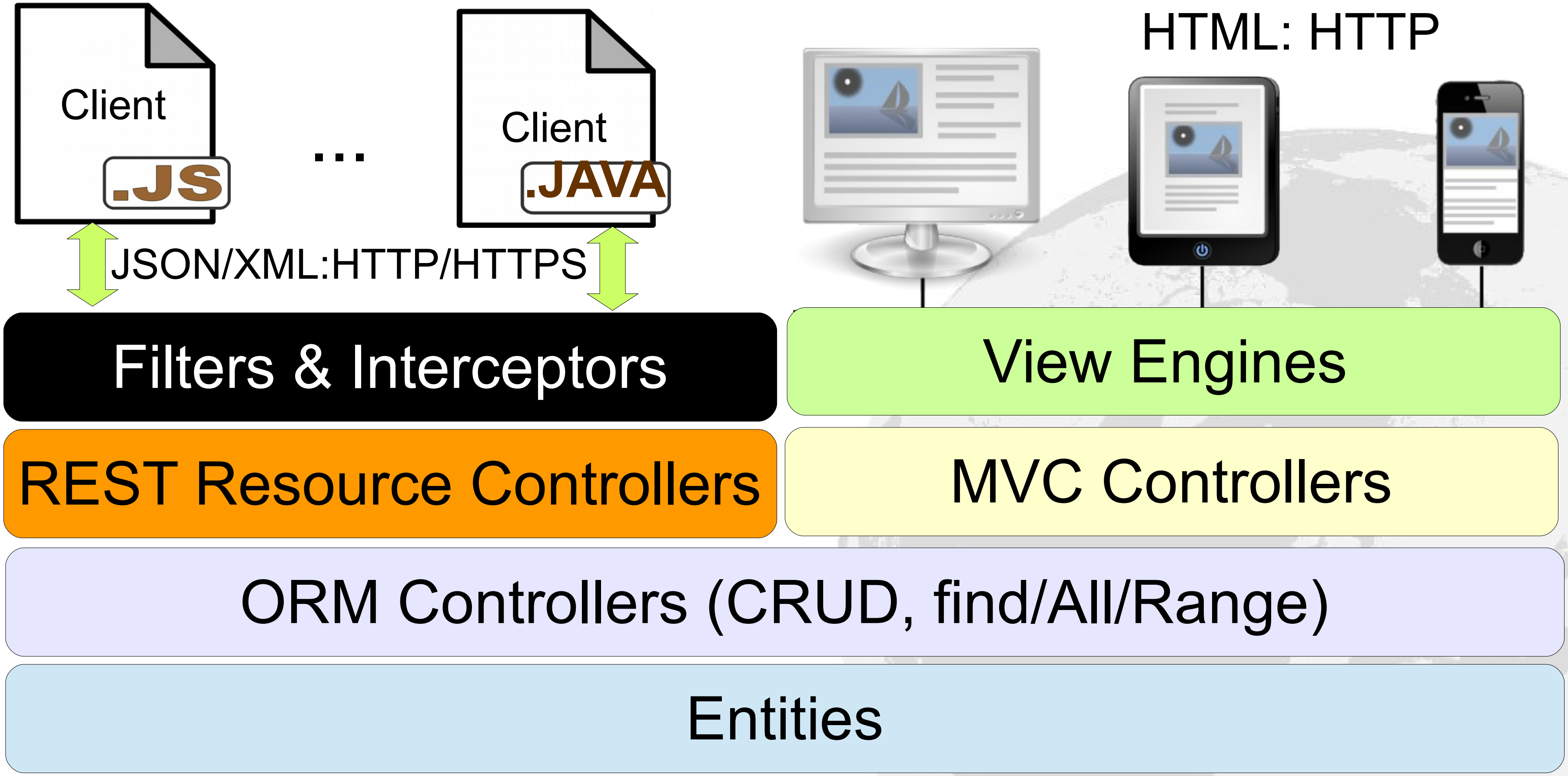
```
Content-Length →1656
Content-Type →application/json
Link →<http://localhost:8080/polling/resources/polls/629>; rel="prev";
type="application/json"; title="Previous poll", <http://localhost:8080/
polling/resources/polls/632>; rel="next"; type="application/json";
title="Next poll", <http://localhost:8080/polling/resources/polls>;
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"
```

Web Application Description Language (WADL)

- XML-based file format providing machine-readable description of HTTP-based web application resources – typically RESTful web services
- WADL is a W3C Member Submission
  - Multiple resources
  - Inter-connections between resources
  - HTTP methods that can be applied accessing each resource
  - Expected inputs, outputs and their data-type formats
  - XML Schema data-type formats for representing the RESTful resources
- But WADL resource description is static

Източник: http://en.wikipedia.org/wiki/File:Webservices.png, автор: H. Voormann
Лиценз: Creative Commons Attribution 3.0 Unported

N-Tier Architectures



Cross-Origin Resource Sharing (CORS)

- Позволява осъществяване на заявки за ресурси към домейни различни от този за извикващия скрипт, като едновременно предоставя възможност на сървъра да прецени към кои скриптове (от кои домейни – Origin) да връща ресурса и какъв тип заявки да разрешава (GET, POST)
- За да се осъществи това, когато заявката е с HTTP метод различен от GET се прави предварителна (preflight) OPTIONS заявка в отговор на която сървъра връща кои методи са достъпни за съответния Origin и съответния ресурс

Нови заглавни части на HTTP при реализация на CORS

- HTTP GET заявка

```
GET /crossDomainResource/ HTTP/1.1
Referer: http://sample.com/crossDomainMashup/
Origin: http://sample.com
```

- HTTP GET отговор

```
Access-Control-Allow-Origin: http://sample.com
Content-Type: application/xml
```

Нови заглавни части на HTTP при реализация на POST заявки при CORS

- HTTP OPTIONS preflight request

```
OPTIONS /crossDomainPOSTResource/ HTTP/1.1
Origin: http://sample.com
Access-Control-Request-Method: POST
Access-Control-Request-Headers: MYHEADER
```

- 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
Access-Control-Max-Age: 864000
```

EcmaScript 6 – ES 2015, Harmony [https://github.com/lukehoban/es6features]

A lot of new features:

- arrows
- classes
- enhanced object literals
- template strings
- destructuring
- default + rest + spread
- let + const
- iterators + for..of
- Generators
- unicode
- Modules + module loaders
- map + set + weakmap + weakset
- proxies
- symbols
- subclassable built-ins
- Promises
- math + number + string + array + object APIs
- binary and octal literals
- reflect api
- tail calls

Fetch API

[https://developer.mozilla.org/en-US/docs/Web/API/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 - 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)

ES6 Classes [http://es6-features.org/]

```
class Shape {
    constructor (id, x, y) {
        this.id = id
        this.move(x, y)
    }
    move (x, y) {
        this.x = x
        this.y = y
    }
}
```

```
class Rectangle extends Shape {
    constructor (id, x, y, width, height)
{
        super(id, x, y)
        this.width  = width
        this.height = height
    }
}
class Circle extends Shape {
    constructor (id, x, y, radius) {
        super(id, x, y)
        this.radius = radius
    }
}
```

Block Scope Vars: let [http://es6-features.org/]

```
for (let i = 0; i < a.length; i++) {
    let x = a[i]
    …
}
for (let i = 0; i < b.length; i++) {
    let y = b[i]
    …
}
```

```
let callbacks = []
for (let i = 0; i <= 2; i++) {
    callbacks[i] =
        function () { return i * 2 }
}

callbacks[0]() === 0
callbacks[1]() === 2
callbacks[2]() === 4
```

ES6 Arrow Functions and this

- ECMAScript 6:

```
this.nums.forEach((v) => {
    if (v % 5 === 0)
        this.fives.push(v)
})
```

- ECMAScript 5:

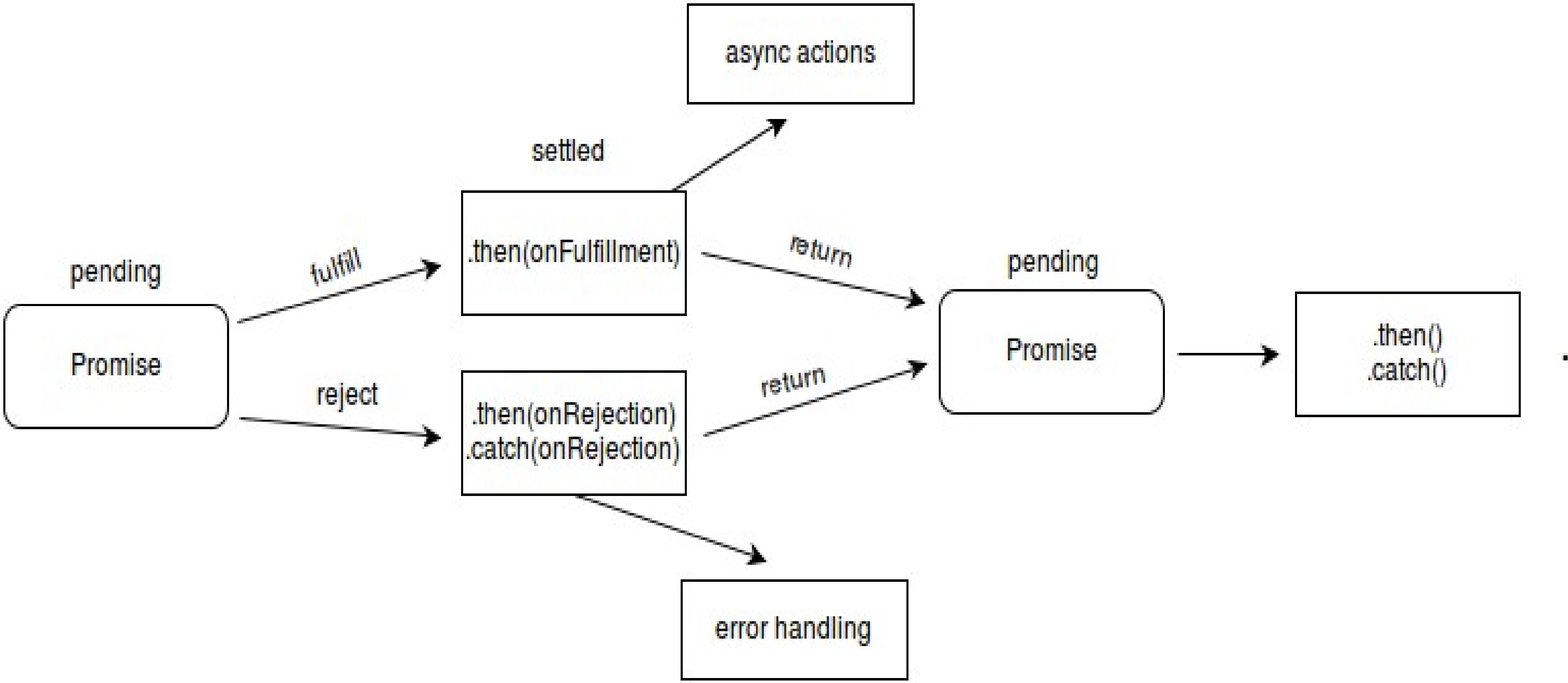
```
var self = this;
this.nums.forEach(function (v) {
    if (v % 5 === 0)
        self.fives.push(v);
});
```

Source: http://wiki.commonjs.org/wiki/Modules/1.1

ES6 Promises [http://es6-features.org/]

```
function msgAfterTimeout (msg, who, timeout) {
    return new Promise((resolve, reject) => {
        setTimeout(() => resolve(`${msg} Hello ${who}!`), timeout)
    })
}
msgAfterTimeout("", "Foo", 1000).then((msg) => {
    console.log(`done after 1000ms:${msg}`);
    return msgAfterTimeout(msg, "Bar", 2000);
}).then((msg) => {
    console.log(`done after 3000ms:${msg}`)
})
```

ES6 Promises



Source: *https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise*

Combining ES6 Promises

```
function fetchAsync (url, timeout, onData, onError) { … }
fetchPromised = (url, timeout) => {
    return new Promise((resolve, reject) => {
        fetchAsync(url, timeout, resolve, reject)
    })
}
Promise.all([
    fetchPromised("http://backend/foo.txt", 500),
    fetchPromised("http://backend/bar.txt", 500),
    fetchPromised("http://backend/baz.txt", 500)
]).then((data) => {
    let [ foo, bar, baz ] = data
    console.log(`success: foo=${foo} bar=${bar} baz=${baz}`)
}, (err) => {
    console.log(`error: ${err}`)
})
```

JavaScript Module Systems - CommonJS

- math.js:

```
exports.add = function() {
    var sum = 0, i = 0, args = arguments, len = args.length;
    while (i < len) {
        sum += args[i++];
    }
    return sum;
};
```

- increment.js:

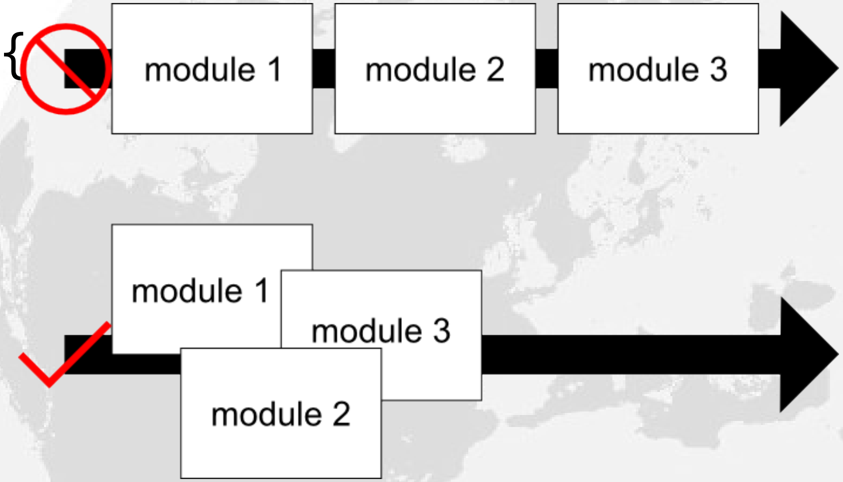
```
var add = require('./math').add;
exports.increment = function(val) {
    return add(val, 1);
};
```

Source: http://wiki.commonjs.org/wiki/Modules/1.1

JavaScript Module Systems – AMD I

```
//Calling define with module ID, dependency array, and factory
//function
define('myModule', ['dep1', 'dep2'], function (dep1, dep2) {
    //Define the module value by returning a value.
    return function () {};
});

define(["alpha"], function (alpha) {
    return {
       verb: function(){
           return alpha.verb() + 2;
       }
    };
});
```



JavaScript Module Systems - AMD II

- Asynchronous module definition (AMD) – API for defining code modules and their dependencies, loading them asynchronously, on demand (lazy), dependencies managed, client-side

```
define("alpha", ["require", "exports", "beta"],
  function(require, exports, beta) {
    exports.verb = function() {
      return beta.verb();
      //OR
      return require("beta").verb();
    }   });

define(function (require) {
   require(['a', 'b'], function (a, b) {//use modules a and b
   });
});
```

JavaScript Module Systems – ES6

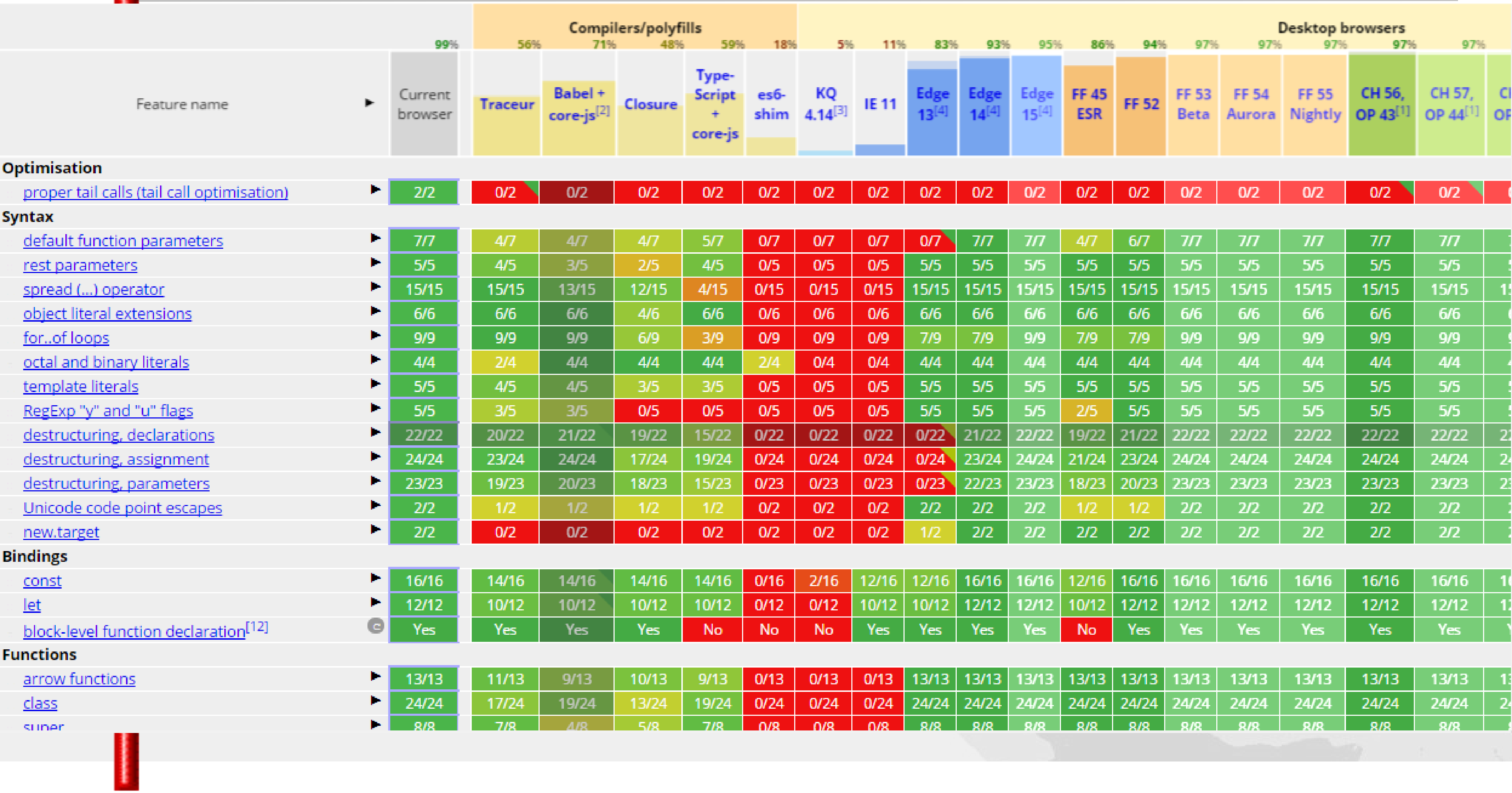
-
  ```
  //  lib/math.js
  export function sum (x, y) { return x + y }
  export var pi = 3.141593
  ```

-
  ```
  //  someApp.js
  import * as math from "lib/math"
  console.log("2π = " + math.sum(math.pi, math.pi))
  ```

-
  ```
  //  otherApp.js
  import { sum, pi } from "lib/math"
  console.log("2π = " + sum(pi, pi))
  ```

-
  ```
  // default export from hello.js and import
  export default () => ( <div>Hello from React!</div>);
  import Hello from "./hello";
  ```

EcmaScript 6 Compatibility [http://kangax.github.io/compat-table/es6/]



		Compilers/polyfills					Desktop browsers											
	99%	56%	71%	48%	59%	18%	5%	11%	83%	93%	95%	86%	94%	97%	97%	97%	97%	97%
Feature name ▶	Current browser	Traceur	Babel + core-js[2]	Closure	Type-Script + core-js	es6-shim	KQ 4.14[3]	IE 11	Edge 13[4]	Edge 14[4]	Edge 15[4]	FF 45 ESR	FF 52	FF 53 Beta	FF 54 Aurora	FF 55 Nightly	CH 56, OP 43[1]	CH 57, OP 44[1]
Optimisation																		
proper tail calls (tail call optimisation) ▶	2/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2
Syntax																		
default function parameters ▶	7/7	4/7	4/7	4/7	5/7	0/7	0/7	0/7	0/7	7/7	7/7	4/7	6/7	7/7	7/7	7/7	7/7	7/7
rest parameters ▶	5/5	4/5	3/5	2/5	4/5	0/5	0/5	0/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5
spread (...) operator ▶	15/15	15/15	13/15	12/15	4/15	0/15	0/15	0/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15
object literal extensions ▶	6/6	6/6	6/6	4/6	6/6	0/6	0/6	0/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6
for..of loops ▶	9/9	9/9	9/9	6/9	3/9	0/9	0/9	0/9	7/9	7/9	9/9	7/9	7/9	9/9	9/9	9/9	9/9	9/9
octal and binary literals ▶	4/4	2/4	4/4	4/4	4/4	2/4	0/4	0/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4
template literals ▶	5/5	4/5	4/5	3/5	3/5	0/5	0/5	0/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5
RegExp "y" and "u" flags ▶	5/5	3/5	3/5	0/5	0/5	0/5	0/5	0/5	5/5	5/5	5/5	2/5	5/5	5/5	5/5	5/5	5/5	5/5
destructuring, declarations ▶	22/22	20/22	21/22	19/22	15/22	0/22	0/22	0/22	0/22	21/22	22/22	19/22	21/22	22/22	22/22	22/22	22/22	22/22
destructuring, assignment ▶	24/24	23/24	24/24	17/24	19/24	0/24	0/24	0/24	0/24	23/24	24/24	21/24	23/24	24/24	24/24	24/24	24/24	24/24
destructuring, parameters ▶	23/23	19/23	20/23	18/23	15/23	0/23	0/23	0/23	0/23	22/23	23/23	18/23	20/23	23/23	23/23	23/23	23/23	23/23
Unicode code point escapes ▶	2/2	1/2	1/2	1/2	1/2	0/2	0/2	0/2	2/2	2/2	2/2	1/2	1/2	2/2	2/2	2/2	2/2	2/2
new.target ▶	2/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	1/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2
Bindings																		
const ▶	16/16	14/16	14/16	14/16	14/16	0/16	2/16	12/16	12/16	16/16	16/16	12/16	16/16	16/16	16/16	16/16	16/16	16/16
let ▶	12/12	10/12	10/12	10/12	10/12	0/12	0/12	10/12	10/12	12/12	12/12	10/12	12/12	12/12	12/12	12/12	12/12	12/12
block-level function declaration[12]	Yes	Yes	Yes	Yes	No	No	No	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes
Functions																		
arrow functions ▶	13/13	11/13	9/13	10/13	9/13	0/13	0/13	0/13	13/13	13/13	13/13	13/13	13/13	13/13	13/13	13/13	13/13	13/13
class ▶	24/24	17/24	19/24	13/24	19/24	0/24	0/24	0/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24
super ▶	8/8	7/8	4/8	5/8	7/8	0/8	0/8	0/8	8/8	8/8	8/8	8/8	8/8	8/8	8/8	8/8	8/8	8/8

Developing Sinagle Page Apps (SPA) in 3 steps

1) Setting up a build system – *npm, webpack, gulp* are common choices, *babel, typescript, JSX, CSS preprocessors (SASS, SCSS, LESS), jasmine, karma, protractor, Yeoman/ Slush, live servers*

2) Designing front-end architecture components – *views & layouts* + *view models* (presentation data models) + *presentation logic* (event handling, messaging) + *routing paths* (essential for SPA)

   Better to use component model to boost productivity and maintainability.

3) End-to-end application design – front-end: wireframes → views, data entities & data streams → service API and models design, sitemap → router config

Creating New Project: NPM + WebPack

[https://www.sitepoint.com/beginners-guide-to-webpack-2-and-module-bundling/]

```
mkdir my-project
cd my-project
npm init
npm install webpack webpack-dev-server  --save-dev
touch index.html src/index.js webpack.config.js
npm install babel-core babel-loader babel-preset-es2015 --save-dev
npm install css-loader style-loader sass-loader node-sass --save-dev
npm install file-loader url-loader --save-dev
npm install extract-text-webpack-plugin
```

In package.json:

```
"scripts": {
    "start": "webpack-dev-server --inline --hot",
    "watch": "webpack --watch",
    "build": "webpack -p"
},
```

Simple WebPack – webpack.config.js (1)

[https://www.sitepoint.com/beginners-guide-webpack-module-bundling/]

```
const path = require('path');

module.exports = {
  context: path.resolve(__dirname, 'src'),
  entry: './index.js',
 output: {
    path: path.resolve(__dirname, 'dist'),
    filename: 'bundle.js'
 },

...
```

Simple WebPack - webpack.config.js (2)

[https://www.sitepoint.com/beginners-guide-webpack-module-bundling/]

```
 module: {
  rules: [{
      test: /\.js$/,
      include: path.resolve(__dirname, 'src'),
   use: [{
        loader: 'babel-loader',
    options: {
     presets: [
            ['es2015', { modules: false }]
     ]
    }
   }]
  }]
 }
};
```

Webpack Project Bootstraping

Installing Webpack:
https://webpack.js.org/guides/installation/

Getting Started with Webpack:
https://webpack.js.org/

Webpack 2 configuration explained :
https://webpack.js.org/configuration/

A Beginner's Guide to Webpack 2 & Module Bundling :
https://www.sitepoint.com/beginners-guide-webpack-module-bundling/

Webpack Tutorials

Webpack: An Introduction (Angular website):
https://angular.io/docs/ts/latest/guide/webpack.html

SurviveJS – Webpack tutorial (more advanced, older Webpack version):
http://survivejs.com/webpack/introduction/

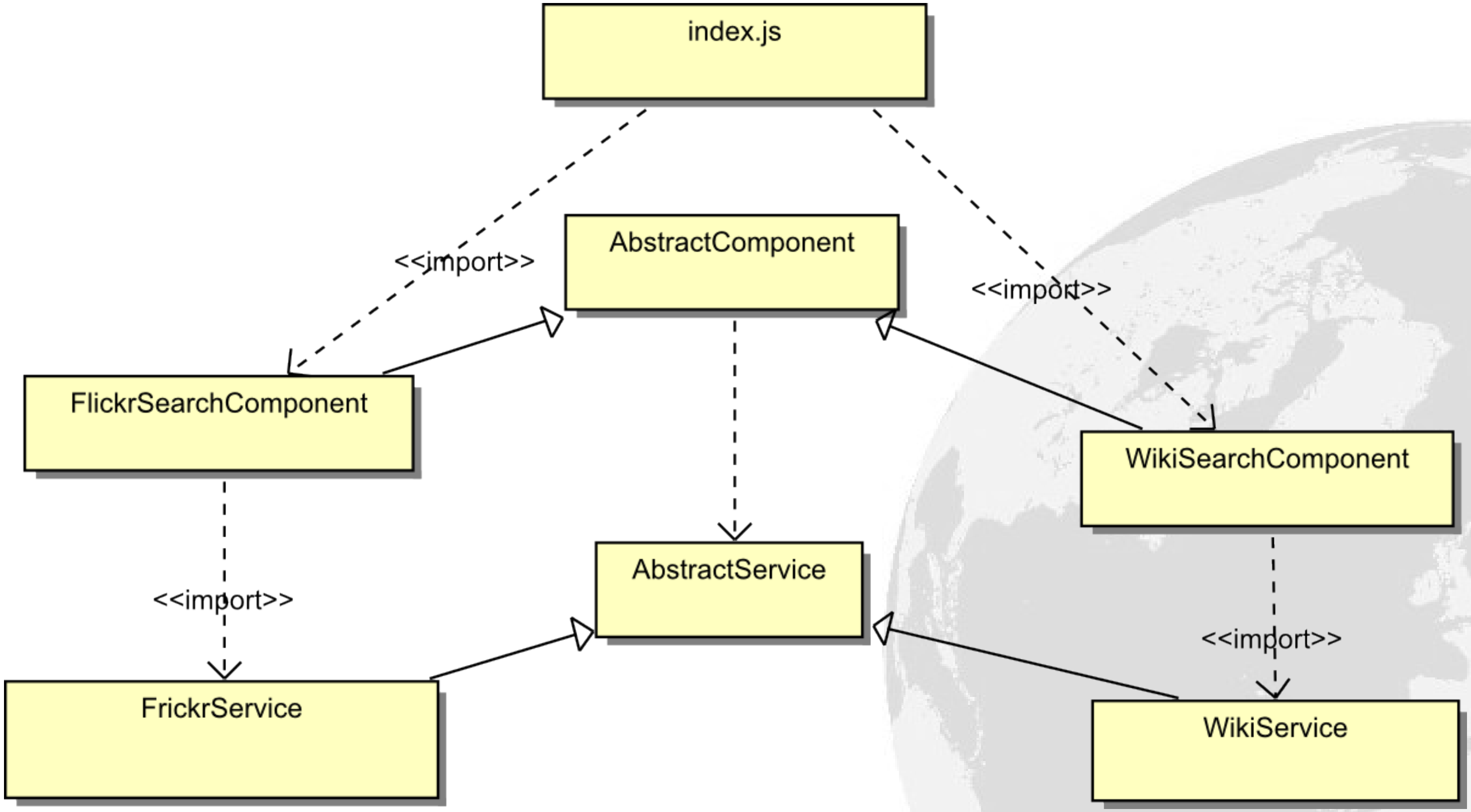
WebPack 2 Loaders and Plugins

- Loaders are transformations (functions running in node.js) that are applied on a resource file of your app
- For example, you can use loaders to to load ES6/7 or JSX
- Loaders can be chained in a pipeline to the resource. The final loader is expected to return JavaScript
- Loaders can be synchronous or asynchronous
- Loaders accept query parameters – loader configuration
- Loaders can be bound to extensions / RegExps
- Loaders can be published / installed through npm
- Plugins can give loaders more features

WebPack Loaders [https://webpack.js.org/loaders/]

- babel-loader - turns ES6 code into vanilla ES5 using Babel
- file-loader - emits the file into the output folder and returns the url
- url-loader - like file loader, but returns Data Url if file size <= limit
- extract-loader - prepares HTML and CSS modules to be extracted into separate files (alt. to ExtractTextWebpackPlugin)
- html-loader - exports HTML as string, requiring static resources
- style-loader - adds exports of a module as style to DOM
- css-loader - loads css file resolving imports and returns css code
- sass-loader - loads and compiles a SASS/SCSS file
- postcss-loader - loads and transforms a CSS file using PostCSS
- raw-loader - lets you import files as a string

Webpack Demo Structure



References [1]

- jQuery JS library - http://jquery.com/
- Representational state transfer (REST) in Wikipedia – http://en.wikipedia.org/wiki/Representational_state_transfer
- JavaScript Object Notation (JSON) – http://www.json.org/
- Fielding's blog discussing REST – http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven
- Representational state transfer (REST) in Wikipedia – http://en.wikipedia.org/wiki/Representational_state_transfer
- Hypermedia as the Engine of Application State (HATEOAS) in Wikipedia – http://en.wikipedia.org/wiki/HATEOAS
- JavaScript Object Notation (JSON) – http://www.json.org/

Thanks for Your Attention!

Questions?