Full-stack Development with Node.js and React.js

Trayan Iliev, http://www.iproduct.org/

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

Trayan Iliev

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

Oracle®, Java™ and JavaScript™ are trademarks or registered trademarks of Oracle and/or its affiliates.
Microsoft .NET, Visual Studio and Visual Studio Code are trademarks of Microsoft Corporation.
Other names may be trademarks of their respective owners.
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-node-express-react
Event Handling Models in JavaScript

- DOM Level 0 (original Netscape model)

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

- Traditional model (as properties)

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

- can register multiple event handlers:

```javascript
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
W3C DOM Level 2 Event Handling Model

- Three phases in event handling life-cycle:
  - Capturing phase – from document to target element
  - At Target phase – processing 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

<table>
<thead>
<tr>
<th>Интерфейс</th>
<th>Събития</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event</td>
<td>abort, blur, change, error, focus, load, reset, resize, scroll, select, submit, unload</td>
</tr>
<tr>
<td>MouseEvent</td>
<td>click, mousedown, mousemove, mouseout, mouseover, mouseup</td>
</tr>
<tr>
<td>UIEvent</td>
<td>DOMActivate, DOMFocusIn, DOMFocusOut</td>
</tr>
</tbody>
</table>
Asynchronous JavaScript & XML - AJAX

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

Client (Web Browser)

```javascript
XMLHttpRequest

function validationCallback() {
    if (xmlhttp.responseText !== 'valid') {
        fieldCity.style.color = 'red';
    }
}

Enter city:

Send
```

Server

GET validator?fieldCity=Sofya

```xml
<status>invalid</status>
```

Servlet Container

CityValidationServlet

doGet() { ... }

Storage
AJAX Interactions Flowchart

- User requests Ajax enabled page
- Browser presents 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
- Server presents next page
  - HTTP Request -> POST page data
  - HTTP Response -> next page
Basic Structure of **Synchronous** AJAX Request

```javascript
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");
}

isAsynchronous = false

xmlhttp.open(method, url, false);
xmlhttp.send();
document.getElementById("results").innerHTML = xmlhttp.responseText;
```
AJAX Request with XML Processing and Authentication

```javascript
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

```javascript
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);
```
### XMLHttpRequest.readyState

<table>
<thead>
<tr>
<th>Код</th>
<th>Значение</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>след като XMLHttpRequest.open() е извикан успешно</td>
</tr>
<tr>
<td>2</td>
<td>заглавните части на отговора на HTTP заявката (HTTP response headers) са успешно получени</td>
</tr>
<tr>
<td>3</td>
<td>начало на зреядане на съдържанието на HTTP отговора (HTTP response content)</td>
</tr>
<tr>
<td>4</td>
<td>съдържанието на HTTP отговора е заредено успешно от браузъра</td>
</tr>
</tbody>
</table>
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

- In HTTP 1.0, all header parts are optional.
- In HTTP 1.1, all header parts are optional except for Host.
- It is necessary to always check if the corresponding header part is different from null.
HTTP Requests Status Codes - RFC2616

- **Accept**
- **Accept-Charset**
- **Accept-Encoding**
- **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

```json
[{
  "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

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>Continue</td>
</tr>
<tr>
<td>101</td>
<td>Switching Protocols</td>
</tr>
<tr>
<td>200</td>
<td>OK</td>
</tr>
<tr>
<td>201</td>
<td>Created</td>
</tr>
<tr>
<td>202</td>
<td>Accepted</td>
</tr>
<tr>
<td>203</td>
<td>Non-Authoritative Information</td>
</tr>
<tr>
<td>204</td>
<td>No Content</td>
</tr>
<tr>
<td>205</td>
<td>Reset Content</td>
</tr>
<tr>
<td>301</td>
<td>Moved Permanently</td>
</tr>
<tr>
<td>302</td>
<td>Found</td>
</tr>
<tr>
<td>303</td>
<td>See Other</td>
</tr>
<tr>
<td>304</td>
<td>Not Modified</td>
</tr>
<tr>
<td>307</td>
<td>Temporary Redirect</td>
</tr>
<tr>
<td>400</td>
<td>Bad Request</td>
</tr>
<tr>
<td>401</td>
<td>Unauthorized</td>
</tr>
<tr>
<td>403</td>
<td>Forbidden</td>
</tr>
<tr>
<td>404</td>
<td>Not Found</td>
</tr>
</tbody>
</table>

---

Copyright © 2003-2016 IPT – Intellectual Products & Technologies Ltd. All rights reserved.
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
jQuery

- jQuery is a fast and concise JavaScript Library that simplifies HTML document traversing, event handling, animating, and Ajax interactions for rapid web development [http://jquery.com/]
- Lightweight Footprint - about 31KB in size (Minified and Gzipped)
- Easy-to-use but powerful – Ajax, Attributes, Callbacks Object, Core, CSS, Data, Deferred Object, Dimensions, Effects, Events, Forms, Internals, Manipulation, Miscellaneous, Offset, Plugins, Properties, Selectors, Traversing, Utilities
- Widespread JS library with many third-party plugins
jQuery [2]

- Supports CSS 3 selectors and much more
- JQuery 3.x browser support – IE: 9+, Chrome, Edge, Firefox, Safari: Current/ -1, Opera: Current, Safari iOS: 7+, Android 4.0+
- Supports own layout and presentation widgets – jQueryUI
  - Interactions – Drag/Droppable, Resizable, Selectable, Sortable
  - Widgets – Accordion, Autocomplete, Button, Datepicker, Dialog, Menu, Progressbar, Slider, Spinner, Tabs, Tooltip
  - Effects – Add Class, Color Animation, Effect, Hide, Remove Class, Show, Switch Class, Toggle, Toggle Class
  - Utilities – Position, Widget Factory
- Supports custom themes (CSS)
jQuery Mobile

- jQuery Mobile: Touch-Optimized Web Framework for Smartphones & Tablets – A unified, HTML5-based user interface system for all popular mobile device platforms, built on the rock-solid jQuery and jQuery UI foundation. Its lightweight code is built with progressive enhancement, and has a flexible, easily themeable design [http://jquerymobile.com/]

- jQuery Mobile is a separate project for building standard compliant (HTML 5, CSS 3, WAI-ARIA) mobile applications
jQuery Mobile recently provided a web-based drag-and-drop UI builder – Codiga, that produces .HTML, .CSS and .JS basic files on-the-fly.
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]

```javascript
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]

```javascript
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]

```javascript
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)

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
- 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 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 tolerant, 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 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) [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

<table>
<thead>
<tr>
<th>Uniform Resource Locator (URL)</th>
<th>GET</th>
<th>PUT</th>
<th>POST</th>
<th>DELETE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collection, such as (<a href="http://api.example.com/comments/%5C">http://api.example.com/comments/\</a>)</td>
<td>List the URIs and perhaps other details of the collection's members.</td>
<td>Replace the entire collection with another collection.</td>
<td>Create a new entry in the collection. The new entry's URI is assigned automatically and is usually returned by the operation.</td>
<td>Delete the entire collection.</td>
</tr>
<tr>
<td>Element, such as (<a href="http://api.example.com/comments/11%5C">http://api.example.com/comments/11\</a>)</td>
<td>Retrieve a representation of the addressed member of the collection, expressed in an appropriate Internet media type.</td>
<td>Replace the addressed member of the collection, or if it does not exist, create it.</td>
<td>Not generally used. Treat the addressed member as a collection in its own right and create a new entry in it.</td>
<td>Delete the addressed member of the collection.</td>
</tr>
</tbody>
</table>

Source: [https://en.wikipedia.org/wiki/Representational_state_transfer](https://en.wikipedia.org/wiki/Representational_state_transfer)
Richardson's Maturity Model of Web Services


- **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
N-Tier Architectures

- Entities
  - ORM Controllers (CRUD, find/All/Range)
  - MVC Controllers
  - REST Resource Controllers
  - Filters & Interceptors

Client

JSON/XML: HTTP/HTTPS

HTML: HTTP
Cross-Origin Resource Sharing (CORS)

- Позволява осъществяване на заявки за ресурси към домейни различни от този за извикващия скрипт, като едновременно предостявя възможност на сървъра да преценени 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 заявкка 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
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
ES6 Classes [http://es6-features.org/]

```javascript
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

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

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

- ECMAScript 5:
  ```javascript
  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

Combining ES6 Promises

```javascript
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

```javascript
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";
## Full-stack Development with Node.js and React.js

### EcmaScript 6 Compatibility


<table>
<thead>
<tr>
<th>Feature Name</th>
<th>Current browser</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Optimisation</strong></td>
<td></td>
</tr>
<tr>
<td>proper tail calls (tail call optimisation)</td>
<td>2/2</td>
</tr>
<tr>
<td><strong>Syntax</strong></td>
<td></td>
</tr>
<tr>
<td>default function parameters</td>
<td></td>
</tr>
<tr>
<td>rest parameters</td>
<td></td>
</tr>
<tr>
<td>spread (...) operator</td>
<td></td>
</tr>
<tr>
<td>object literal extensions</td>
<td></td>
</tr>
<tr>
<td>for of loops</td>
<td></td>
</tr>
<tr>
<td>octal and binary literals</td>
<td></td>
</tr>
<tr>
<td>template literals</td>
<td></td>
</tr>
<tr>
<td>RegExp ‘v’ and ‘u’ flags</td>
<td></td>
</tr>
<tr>
<td>destructuring declarations</td>
<td></td>
</tr>
<tr>
<td>destructuring assignment</td>
<td></td>
</tr>
<tr>
<td>destructuring parameters</td>
<td></td>
</tr>
<tr>
<td>Unicode code point escapes</td>
<td></td>
</tr>
<tr>
<td>new.target</td>
<td></td>
</tr>
<tr>
<td><strong>Functions</strong></td>
<td></td>
</tr>
<tr>
<td>arrow functions</td>
<td></td>
</tr>
<tr>
<td>class</td>
<td></td>
</tr>
<tr>
<td>spread</td>
<td></td>
</tr>
<tr>
<td>super</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>56%</td>
<td>71%</td>
<td>48%</td>
<td>59%</td>
<td>18%</td>
<td>5%</td>
<td>11%</td>
<td>83%</td>
<td>93%</td>
<td>95%</td>
<td>85%</td>
<td>94%</td>
<td>97%</td>
<td>97%</td>
<td>97%</td>
<td>97%</td>
<td>97%</td>
<td>97%</td>
</tr>
<tr>
<td>Desktop browsers</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

Copyright © 2003-2016 IPT – Intellectual Products & Technologies Ltd. All rights reserved.
Developing Single 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/]

```bash
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 2 – webpack.config.js (1)
[https://www.sitepoint.com/beginners-guide-to-webpack-2-and-module-bundling/]

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

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

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

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

Installing Webpack 2:
https://webpack.js.org/guides/installation/

Getting Started with Webpack 2:
https://webpack.js.org/guides/get-started/

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

A Beginner’s Guide to Webpack 2 & Module Bundling:
Webpack Tutorials

Webpack 2: An Introduction (Angular 2 website):
https://angular.io/docs/ts/latest/guide/webpack.html

N. Dabit – Beginner’s guide to Webpack:
https://medium.com/@dabit3/beginner-s-guide-to-webpack-b1f1a3638460

SurviveJS – Webpack tutorial (more advanced):
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 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 2 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 2 Main Plugins

- **CommonsChunkPlugin** - generates chunks of common modules shared between entry points and splits them to separate bundles.
- **ExtractTextWebpackPlugin** - extracts CSS from your bundles into a separate file (e.g. app.bundle.css).
- **CompressionWebpackPlugin** - prepare compressed versions of assets to serve them with Content-Encoding.
- **I18nWebpackPlugin** - adds i18n support to your bundles.
- **HtmlWebpackPlugin** - simplifies creation of HTML files (index.html) to serve your bundles.
- **ProvidePlugin** - automatically loads modules, whenever used.
- **UglifyJsPlugin** - tree transformer and compressor which reduces the code size by applying various optimizations.
Webpack Demo Structure

- index.js
- AbstractComponent
  - FlickrSearchComponent
  - AbstractService
  - FrickrService
- WikiSearchComponent
  - WikiService
References [1]

- jQuery JS library - http://jquery.com/
- JavaScript Object Notation (JSON) – http://www.json.org/
- JavaScript Object Notation (JSON) – http://www.json.org/
Thanks for Your Attention!

Questions?