Conception d'Applications Interactives : Applications Web

Séance #1 - Côté navigateur

3/3 - ES6, Web Components

ES6

Not your parents' JavaScript

What is ES6?

ES6

ECMAScript 6

Latest standardized version of JavaScript

Transpilers

Browsers don't support ES6 yet

Transpilers compile ES6 to ES5

`let` and constants

`let`

```
function getPonyFullName(pony) {
  if (pony.isChampion) {
    let name = 'Champion ' + pony.name;
    return name;
  }
  // name is not accessible here
  return pony.name;
}
```

constants

```
const poniesInRace = 6;
poniesInRace = 7; // SyntaxError

const PONY = {};
PONY.color = 'blue'; // works
PONY = {color: 'blue'}; // SyntaxError
```

Creating and destructuring

Creating objects

```
function createPony() {
  const name = 'Rainbow Dash';
  const color = 'blue';
  return { name, color };
}
```

Destructuring assignment

```
const options = { timeout: 2000, isCache: true };

let { timeout: httpTimeout, isCache: httpCache } = options;

// In ES5:
// var httpTimeout = options.timeout;
// var httpCache = options.isCache;
```

Default parameters & rest operator

Default parameters

```
function getPonies(size = 42, page = 1) {
// ...
  server.get(size, page);
}
```

Rest operator

```
function addPonies(...ponies) {
  for (let pony of ponies) {
    poniesInRace.push(pony);
  }
}
```

Classes

Classes

```
class Pony {
  constructor(color) {
    this.color = color;
  }
  toString() {
    return `${this.color} pony`;
  }
}
const bluePony = new Pony('blue');
console.log(bluePony.toString());
```

Getters and setters

```
class Pony {
  get color() {
    return this._color;
  }
  set color(newColor) {
    this._color = newColor;
  }
}
const pony = new Pony();
pony.color = 'red';
console.log(pony.color);
```

Inheritance

```
class Animal {
  speed() {
    return 10;
  }
}
class Pony extends Animal {
  speed() {
    return super.speed() + 10;
  }
}
const pony = new Pony();
console.log(pony.speed());
// 20, as Pony overrides the parent method
```

```
class Animal {
  constructor(speed) {
    this.speed = speed;
  }
}
class Pony extends Animal {
  constructor(speed, color) {
    super(speed);
    this.color = color;
  }
}
const pony = new Pony(20, 'blue');
console.log(pony.speed); // 20
```

Promises

Declaring promises

```
const getUser = function (login) {
  return new Promise(function (resolve, reject) {
    // async stuff, like fetching users
    if (response.status === 200) {
      resolve(response.data);
    } else {
      reject('No user');
    }
  });
};
```

Using promises

```
getUser(login)
  .then(function (user) {
    return getRights(user);
   // getRights returning a promise
  })
  .then(function (rights) {
    return updateMenu(rights);
  })
  .catch(function (error) {
    console.log(error);
    // if getUser or getRights fails
  })
```

Arrow functions

Arrow functions

```
getUser(login)
  .then(function (user) {
    return getRights(user);
  })
  .then(function (rights) {
    return updateMenu(rights);
  })

getUser(login)
  .then(user => getRights(user))
  .then(rights => updateMenu(rights))
```

`this` stays lexically bounded!

```
const maxFinder = {
  max: 0,
  find: function (numbers) {
    numbers.forEach(element => {
      if (element > this.max) {
        this.max = element;
      }
    });
  }
};
maxFinder.find([2, 3, 4]);
console.log(maxFinder.max);
```

Collections: maps and sets

Maps & Sets

```
const cedric = { id: 1, name: 'Cedric' };
const users = new Map();
users.set(cedric.id, cedric); // adds a user
console.log(users.has(cedric.id)); // true
console.log(users.size); // 1
users.delete(cedric.id); // removes the user

const users = new Set();
users.add(cedric); // adds a user
console.log(users.has(cedric)); // true
console.log(users.size); // 1
users.delete(cedric); // removes the user
```

Iterating on a collection

```
for (let user of users) {
  console.log(user.name);
}
```

Template literals

```
const fullname = 'Miss ' + firstname + ' ' + lastname;

// basic templating system
const fullname = `Miss ${firstname} ${lastname}`;

// Multiline support
const template = `<div>
  <h1>Hello</h1>
</div>`
```

Modules

In `races_service.js`

```
export function bet(race, pony) {
  // ...
}
export function start(race) {
  // ...
}
```

Named imports

```
import { start as startRace } from
    './races_service';

startRace(race);
```

In another file

```
import { bet, start } from './races_service';

// later
bet(race, pony1);
start(race);
```

Default exports

```
// pony.js
export default class Pony {
}
// races_service.js
import Pony from './pony';
```

Du l'outillage

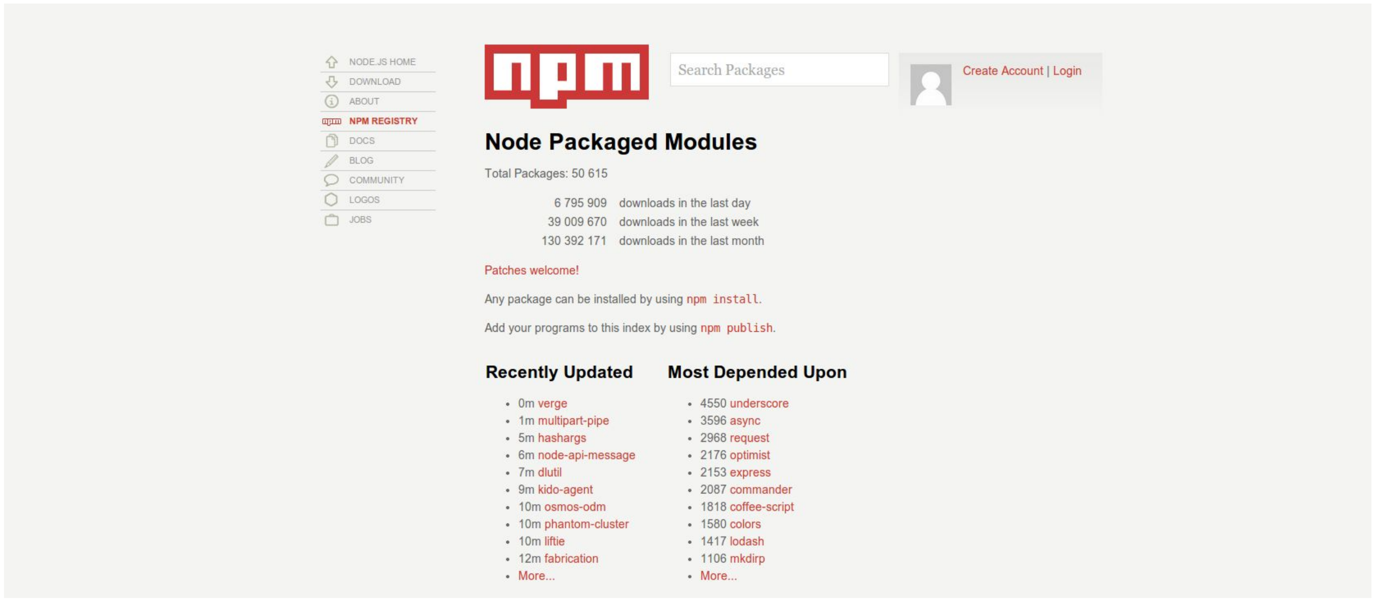
npm, package.json...

What is npm?

- NodeJS built modularly
  - Each functionality in a package

- npm is the official package manager for Node.js
  - runs through the command line
  - manages dependencies for an application
  - install applications available on the npm registry

What is npm registry?



Node Packaged Modules

Total Packages: 50 615

```
  6 795 909   downloads in the last day
 39 009 670   downloads in the last week
130 392 171   downloads in the last month
```

Patches welcome!

Any package can be installed by using `npm install`.

Add your programs to this index by using `npm publish`.

Recently Updated

- 0m verge
- 1m multipart-pipe
- 5m hashargs
- 6m node-api-message
- 7m dlutil
- 9m kido-agent
- 10m osmos-odm
- 10m phantom-cluster
- 10m liftie
- 12m fabrication
- More...

Most Depended Upon

- 4550 underscore
- 3596 async
- 2968 request
- 2176 optimist
- 2153 express
- 2087 commander
- 1818 coffee-script
- 1580 colors
- 1417 lodash
- 1106 mkdirp
- More...

NodeJS equivalent to Maven Central

http://npmjs.org

npm behind a corporate proxy

- Proxy must be defined as environment variable

```
export http_proxy=user:password@proxy.example.com:3128
export https_proxy=user:password@proxy.example.com:3128
```

  - Potential problem with `proxy-pac`…

- Using of `npm config` could be needed

```
npm config set proxy http://proxy.example.com:3128
npm config set https-proxy http://proxy.example.com:3128
```

- If necessary use credentials:

```
npm config set proxy http://user:password@proxy.example.com:3128
npm config set proxy https://user:password@proxy.example.com:3128
```

Exercise: our first NodeJS app

package.json

Either written by hand or using `npm init`

```
{
 "name": "awesome-test",
 "main": "server.js"
}
```

server.js

Main file

```
console.log('Hello World');
```

Run the project using `node server.js`

Restarting a Node Application on File Changes

- NodeJS won't restart when file changes are made
  - We need a 3rd party package for that: nodemon

```
npm install -g nodemon
```

  - Then use nodemon instead of node command

```
nodemon server.js
```

Installing packages

To install a package for our app we add it to packages.json

- By manually writing the dependency

```
{
  "name": "awesome-test",
  "main": "server.js",
   "dependencies": {
      "express": "~4.8.6"
   }
}
```

- By using the command line

```
npm install express --save
```

An HTTP server in pure NodeJS

package.json

Either written by hand or using npm init

```json
{
 "name": "http-server",
 "main": "server.js"
}
```

index.html

Static index file

```html
<!DOCTYPE html>
<html lang="en">
<head>
        <meta charset="UTF-8">
        <title>Super Cool Site</title>
</head>
<body>
        <h1>Hello Universe!</h1>
</body>
</html>
```

An HTTP server in pure NodeJS

```
// get the http and filesystem modules
var http = require('http')
var fs = require('fs');
// create our server using the http module
http.createServer(function(req, res) {
          // write to our server. set configuration for the response
          res.writeHead(200, {
           'Content-Type': 'text/html',
           'Access-Control-Allow-Origin' : '*'
          });
          var readStream = fs.createReadStream(__dirname + '/index.html');
          // send a message
          readStream.pipe(res);
}).listen(1337);
// tell ourselves what's happening
console.log('Visit me at http://localhost:1337');
```

Web Components

Reinventing the wheel...
and this time making it round

Example : the Google+ button

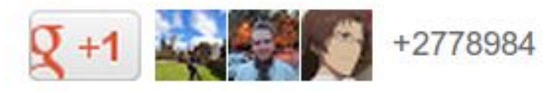
- If you want to place a Google+ button in your page

```
<!-- Place this tag where you want the +1 button to render. -->
<div class="g-plusone" data-annotation="inline"
data-width="300"></div>

<!-- Place this tag after the last +1 button tag. -->
<script type="text/javascript">
 (function() {
   var po = document.createElement('script');
   po.type = 'text/javascript';
   po.async = true;
   po.src = 'https://apis.google.com/js/plusone.js';
   var s = document.getElementsByTagName('script')[0];
   s.parentNode.insertBefore(po, s);
 })();
</script>
```

Example : the Google+ button

And what I would like is simple

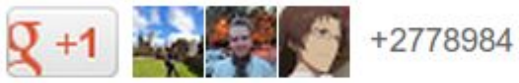
```
<g:plusone></g:plusone>
```



Example : the Google+ button

- To be fair, Google already makes it simpler

```
<script type="text/javascript" src="https://apis.google.com/js/plusone.js">
</script>...
<g:plusone></g:plusone>
```

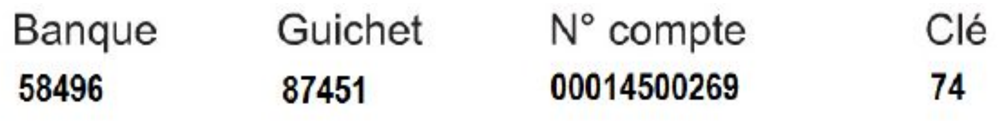


- They create directives with JS to emulate components
  - AngularJS approach
  - Respecting the spirit of the future standard
  - Working in current browsers

Totally non standard...

Another example : the RIB

- If you're French, you know what a RIB is
  - A banking account identification number



Banque	Guichet	N° compte	Clé
58496	87451	00014500269	74

- To show a RIB in HTML:
  - All styling & surface control must be done elsewhere by CSS and JS

```
<div class="rib">58496 87451 00014500269 74</div>
```

- What I would like
  - A semantic tag
  - With encapsulated styling and surface controlling

```
<x-rib banque="58496" guichet="87451" compte="00014500269" cle="74" />
```

But we already can do that!

- In most modern frameworks we can already do components, in a way or another
  - And all those ways are different!
  - Using different JavaScript libraries
  - Almost no component sharing between frameworks

- W3C's works aim to make a standard way
  - Supported natively by all browsers
  - Allowing for component reuse

Web Components : a W3C standard

- Web Components standard is being worked at W3C
    - We all know what this means
        - Clue : HTML5

They will work for years, bickering and fighting

Browsers and devs will use the WiP document

The 4 pillars of the Web Components

- Templates
- Shadow DOM
- Custom Elements
- Imports

Templates

The clone wars

Image: Instructables

Templates before `<template>`

- How did we do templating before
  - Using `display:none` or `hidden`

```
<div id="commentTemplate" class="comment" hidden>
   <img src=""> <div class="comment-text"></div>
</div>
```

  - Putting it inside a `script`
    - Type unknown to browser, it isn't interpreted
    - Markup easily recovered via .innerHTML and reused
    - Approach used by many template engines

```
<script id="commentTemplate" type="text/template">
   <div class="comment">
       <img src=""> <div class="comment-text"></div>
   </div>
</script>
```

The `<template>` tag

- Uniformising those approach with a new tag

```
<template id="commentTemplate">
   <div>
       <img src="">
       <div class="comment-text"></div>
   </div>
</template>
```

- Content inside the tag is parsed but not interpreted
  - HTML not shown
  - Resources are not loaded
  - Scripts not executed

Template instantiation

- Create the elements in page by cloning the template

```
<template id="commentTemplate">
   <div class="comment">
       <img src=""> <div class="comment-text"></div>
   </div>
</template>

<script>
     function addComment(imageUrl, text) {
      var t = document.querySelector("#commentTemplate");
      var comment = t.content.cloneNode(true);

      // Populate content.
      comment.querySelector('img').src = imageUrl;
      comment.querySelector('.comment-text').textContent = text;
      document.body.appendChild(comment);
     }
</script>
```

Shadow DOM

Join the shadowy side,
young padawan

Image: Springfield Punx

Encapsulation

- Each component should have
  - Public interface
  - Private inner code
- When using a component
  - You manipulate the interface only
  - You don't need to know anything about inner code
  - No conflict between inner code and outside code

Encapsulation before Shadow DOM

- Only a way :

`<innerFrame>`

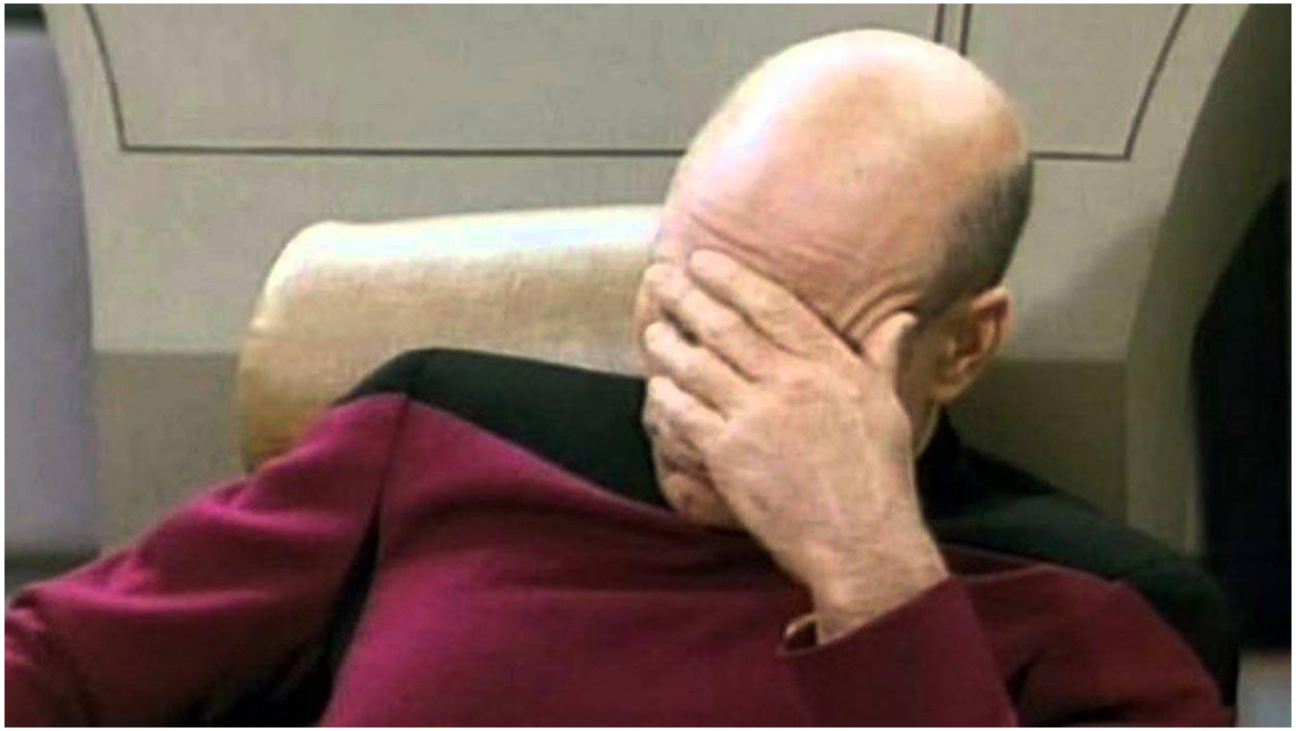
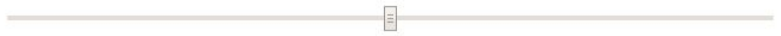


Image : Star Trek the Next Generation

Your browser cheats on you

- Considerer this simple slider

```html
<input id="foo" type="range">
```



  - How does the browser deal with it?
    - With HTML, CSS and JS!
  - It has a movable element, I can recover it's position
    - Why cannot see it in DOM tree?

```html
<video width="320" height="240" controls>
  <source src="movie.mp4" type="video/mp4">
</video>
```

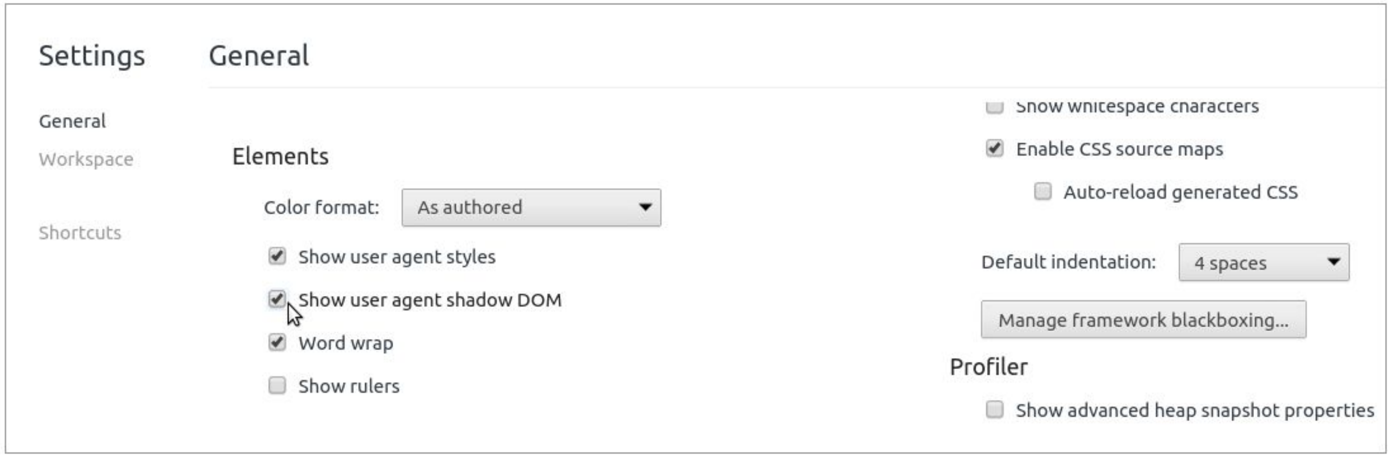
Browsers hide DOM sub-trees for standard components
They have a public interface and hidden inner code

That's Shadow DOM!

My name is DOM, Shadow DOM

- Shadow DOM: ability of the browser to
  - Include a DOM subtree into the rendering
  - But not into the main document DOM tree

- In Chrome you can see the Shadow DOM
  - By activating the option in Inspector



Settings — General

General
Workspace
Shortcuts

Elements

Color format: As authored

- [✓] Show user agent styles
- [✓] Show user agent shadow DOM
- [✓] Word wrap
- [ ] Show rulers

- [ ] Show whitespace characters
- [✓] Enable CSS source maps
  - [ ] Auto-reload generated CSS

Default indentation: 4 spaces

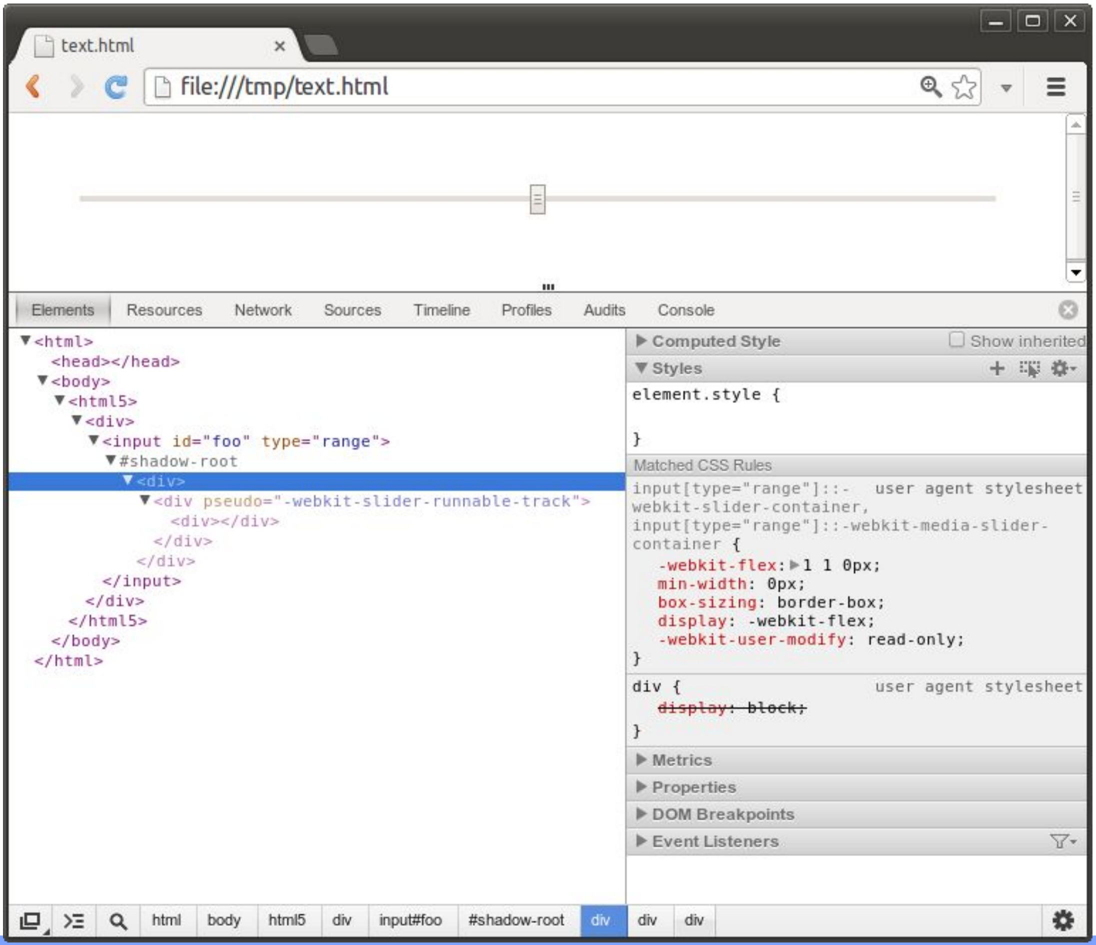
Manage framework blackboxing...

Profiler

- [ ] Show advanced heap snapshot properties

Looking into the Shadow

For the slider :



```
▼<html>
   <head></head>
 ▼<body>
   ▼<html5>
     ▼<div>
       ▼<input id="foo" type="range">
         ▼#shadow-root
           ▼<div>
             ▼<div pseudo="-webkit-slider-runnable-track">
                 <div></div>
               </div>
             </div>
         </input>
       </div>
     </html5>
   </body>
 </html>
```

```
element.style {

}
input[type="range"]::-   user agent stylesheet
webkit-slider-container,
input[type="range"]::-webkit-media-slider-
container {
   -webkit-flex: ▶1 1 0px;
   min-width: 0px;
   box-sizing: border-box;
   display: -webkit-flex;
   -webkit-user-modify: read-only;
}
div {                    user agent stylesheet
   ~~display: block;~~
}
```

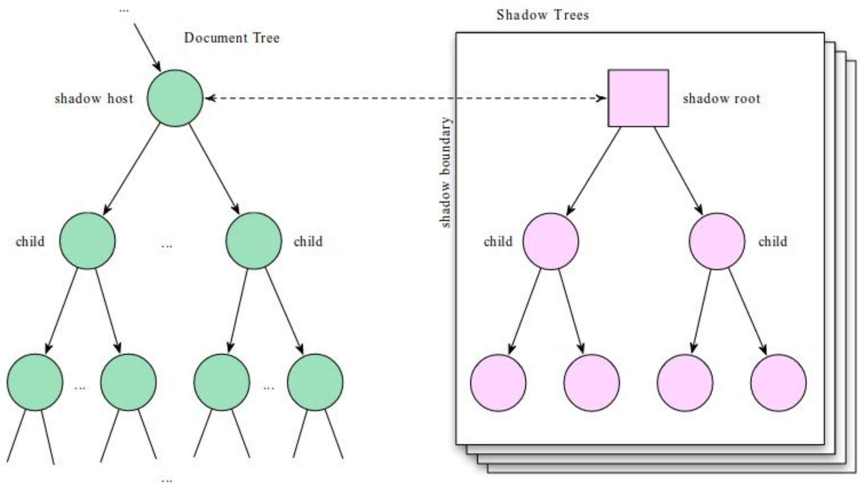
Shadow DOM is already here

- Browser use it everyday...
  - For their inner needs
  - Not exposed to developers
- Web Components makes Shadow DOM available
  - You can use it for your own components

Image: Springfield Punx

Using Shadow DOM

- There is a host element
  - A normal element in DOM tree



- A shadow root is associated to the host
  - Using the createShadowRoot method
  - The shadow root is the root of the hidden DOM tree

Image: W3C

Using Shadow DOM

- Quick and dirty Shadow DOM

```
<div id="emptyHost"></div>
<script>
    var host = document.querySelector('#emptyHost');
    var root = host.createShadowRoot();
    root.innerHTML = "<h1>Not empty anymore!</h4>";
</script>
```

  - DOM tree only shows

```
<div id="emptyHost"></div>
```

  - Rendered HTML shows

```
Not empty anymore!
```

  - Markup in `innerHTML` is ugly

Using Shadow DOM

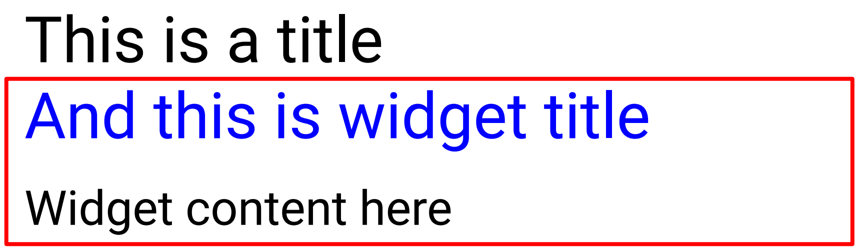
- Shadow DOM with templates

```
<div id="emptyHost"></div>
<template id="commentTemplate"> [...] </template>

<script>
 var host = document.querySelector('#emptyHost');
 var shadowRoot = host.webkitCreateShadowRoot();

 function addComment(imageUrl, text) {  [...]  }

 function addShadowedElement() {
   var instanceTemplate =
     addComment("http://lostinbrittany.org/avatar.png",
                "This is a nice comment made by a nice guy");
   shadowRoot.appendChild(instanceTemplate);
 }
</script>
```

Shadow DOM et CSS

- CSS defined in the Shadow DOM remains there
- Outside styles don't affect Shadowed content

```
<h1>This is a title</h1>
<div id="widget">
 #document-fragment
 <style>
   div {border: solid 1px red;}
   h1 {color: blue;}
 </style>
 <h1>And this is widget title</h1>
 <div>Widget content here</div>
</div>
```


This is a title

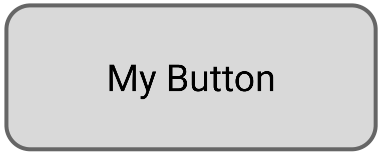
And this is widget title

Widget content here

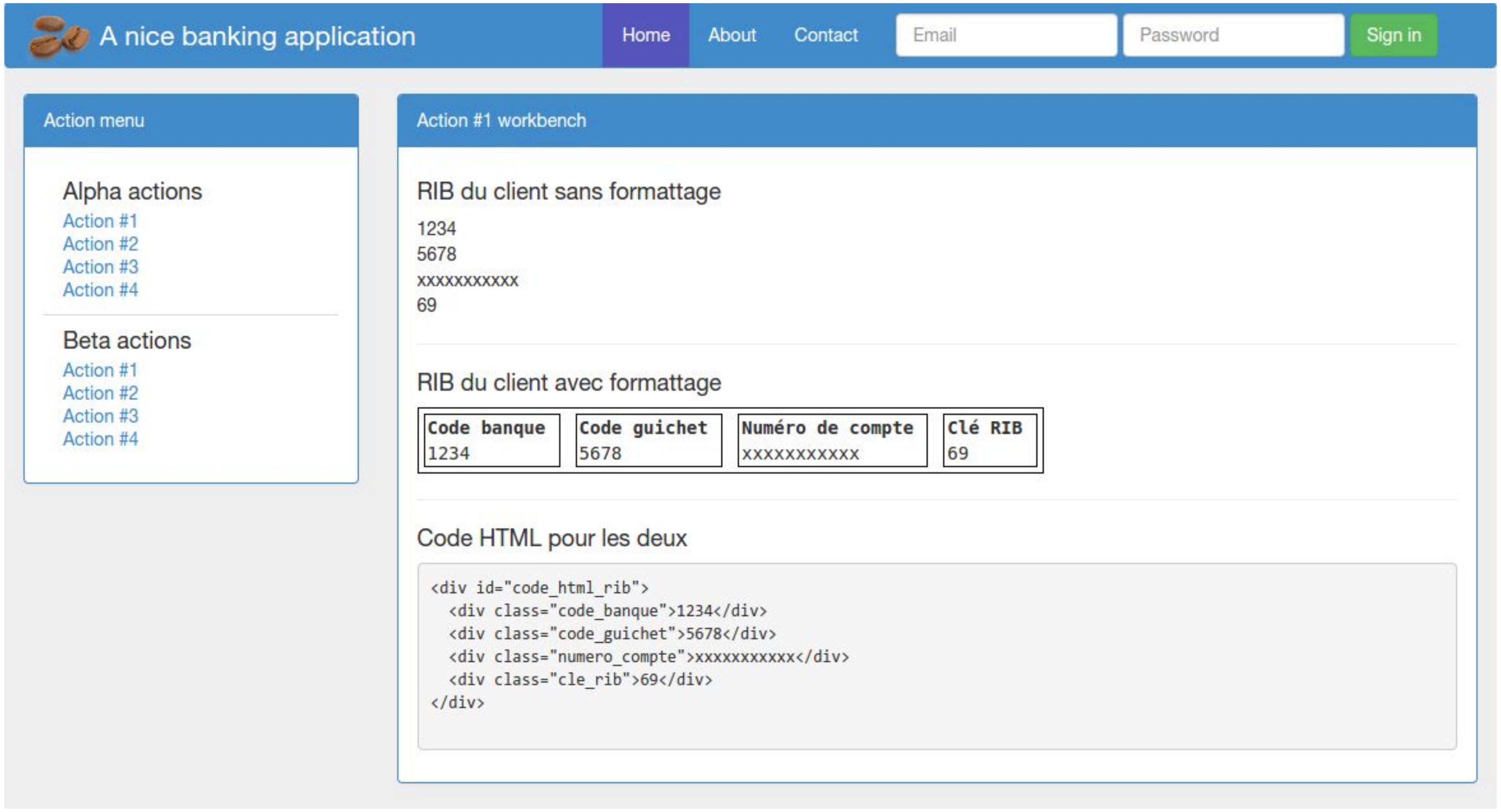
Shadow DOM et CSS

- Styling the host element : @host

```
<template id="template">
   <style>
     @host {
       button { border-radius: 5px; }
     }
   </style>
   <content></content>
</template>

<button id="host">My Button</button>

<script>
     var host = document.querySelector('#host');
     var root = host.createShadowRoot();
     var shadowContent =
          document.querySelector("#template").content.cloneNode(true);
     root.appendChild(shadowContent);
</script>
```



Example



RIB du client sans formattage

1234
5678
xxxxxxxxxxx
69

RIB du client avec formattage

Code banque	Code guichet	Numéro de compte	Clé RIB
1234	5678	xxxxxxxxxxx	69

Code HTML pour les deux

```
<div id="code_html_rib">
  <div class="code_banque">1234</div>
  <div class="code_guichet">5678</div>
  <div class="numero_compte">xxxxxxxxxxx</div>
  <div class="cle_rib">69</div>
</div>
```

Elemental mayhem !

Image: The Brick Blogger

Custom elements : the HTML side

- An element encloses template, lifecycle and behaviour
  - Templates done with `<template>` tag

```html
<!-- Template Definition -->
<template id="template">
  <style>
    ...
  </style>
  <div id="container">
    <img src="http://webcomponents.org/img/logo.svg">
    <content select="h1"></content>
  </div>
</template>

<!-- Custom Element usage -->
<x-component>
  <h1>This is Custom Element</h1>
</x-component>
```

Custom elements: the JavaScript side

- An element encloses template, lifecycle and behaviour
  - JavaScript to define behaviour and register the element

```
var proto = Object.create(HTMLElement.prototype);
proto.createdCallback = function() {
  // Adding a Shadow DOM
  var root = this.createShadowRoot();
  // Adding a template
  var template = document.querySelector('#template');
  var clone = document.importNode(template.content, true);
  root.appendChild(clone);
}
var XComponent = document.registerElement('x-component', {
  prototype: proto
});
```

Extending other elements

- To create element A that extends element B,
  element A must inherit the prototype of element B

```
var MegaButton = document.registerElement('mega-button'
  prototype: Object.create(HTMLButtonElement.prototype),
  extends: 'button'
});
```

Polymer

Webcomponents for today's web

Polymer

- A Google project
    - Introduced in Google I/O 2013
    - Built on top of Web Components
    - Designed to leverage the evolving web platform
    - Version 1.0 released at Google IO 2015

Version 3.0 released at Google IO 2018

Oh yeah!

Polymer

- Principes:
  - Use the platform
    - Use to the maximum the native APIs and capabilities of browsers
    - Don't reinvent the wheel
  - Everything is a component
    - Encapsulation is needed for a component oriented application
  - Extreme pragmatism
    - Boilerplate is bad
    - Anything repetitive should be re-factored into a component
      - Handled by Polymer itself or
      - Added into the browser platform itself

Conclusion

That's all folks!

Image: dcplanet.fr

Thank you !

Thank
You
Gracias
Merci
Arigato
bolzïn
Shukuria
Tashakkur
Grazie
Bïyan
Juspaxar
Mehrbani
Dankscheen
suksama
Shukria
atu
Taghanyelay
Efcharisto
Tingki