Lab 1. Basic Setup

In this lab you will create and use a web component in HTML5.

> duration: 30 minutes

Step 1. Create an HTML page and implement a basic namebadge

Create a directory 'polymer_labs' and navigate into it.
Install NodeJS from it's website and use npm to install lite-server globally

```
$ sudo npm install lite-server -g
```

Create a file 'index.html' inside your current folder and write a html page
that uses a name badge like this:

```
<html>
  ...
  <name-badge></name-badge>
  ...
</html>
```

Run the page inside Chrome. What is the output? How does Chrome handle the unknown tag?

Step 2. Registering the custom element

Open the index.html and import the polyfill for custom elements. This polyfill is provided in the starter files inside the '_labs/Lab. Web Components' folder.

Write script in the page to register and create an instance of the name-badge element. For now, just add dummy content into the innerHTML property for the custom element. We will get back to this later.

Hint: Use customElements.define('name-badge', class extends HTMLElement {}); to define the new element.

Give the new element some style, markup and fake data..
If you are not of the creative type, copy and paste the following script into your HTML file:

```
customElements.define('name-badge', class extends HTMLElement {
        constructor() {
            super();
            this.innerHTML = `
              <style>
              .badge { width:200px;text-align:center; }
              .name { padding:10px;border-radius:5px 5px 0px 0px; border-
style:solid;border-color:black;border-width:2px 2px 0px 2px;background-
color:red;color:white;font-weight:bold;}
              .details { padding:10px;font-style:italic;border-radius:0px 0px
5px 5px; border-style:solid;border-color:black;border-width:1px 2px 2px
2px;background-color:white;}
              </style>
              <div class="badge">
              <div class="name">John Gorter</div>
              <div class="details">Trainer Info Support bv</div>
              </div>
            `;
        }
    });
```

Step 3. Add an attribute to the badge

The badge is kind of uncustomizable. Let's change that... Open the 'index.html' into your favorite editor and change the declaration into:

```
<name-badge name="Another User"></name-badge>
```

Try to write the code to reflect this attribute into the content.

Your code should look similar to the following:

```
customElements.define('name-badge', class extends HTMLElement {
        constructor() {
            super();
            this.innerHTML = `
              <style>
              .badge { width:200px;text-align:center; }
              .name { padding:10px;border-radius:5px 5px 0px 0px; border-
style:solid;border-color:black;border-width:2px 2px 0px 2px;background-
color:red;color:white;font-weight:bold;}
              .details { padding:10px;font-style:italic;border-radius:0px 0px
5px 5px; border-style:solid;border-color:black;border-width:1px 2px 2px
2px;background-color:white;}
              </style>
              <div class="badge">
              <div class="name"><span id="name"></span></div>
              <div class="details">Trainer Info Support bv</div>
              </div>
            `;
            this.querySelector("#name").innerText = this.contactname;
        }
       get contactname() {
          if (this.hasAttribute('contactname')){
            return this.getAttribute('contactname');
          }
          return "John Gorter";
       }
       set contactname(val) {
         this.setAttribute('contactname', val);
         this.querySelector("#name").innerText = val;
       }
    });
```

Try to change the name and use the Chrome Dev Tools to see the attribute reflect the changes.

```
 Tip:
 You can use $0.contactname = 'another value' inside the console after
selecting the name-badge element
 to change the contact property and see the results
```

Step 4. Use Templates to create better HTML

All the markup for our namebadge is inside a string. This is not best practice at all! Let's use templates to make the component more maintainable and better structured.

Take all the markup from the namebadge and stuff it into a template tag. Give the template an id for later use:

```
  <template id="#mytemplate">
  ... markup here ...
  </template>
```

Change the code in the constructor to use the template upon constructing the element. The following code demonstrates the loading and insertion of the template:

```
var temp = document.querySelector("#template").content;
document.appendChild(document.importNode(temp, true));
```

Test to see if all works as expected!

Your complete HTML page should look something like this:

```
<!DOCTYPE html>
<html>
    <head>
        <script src="./CustomElements.js"></script>
    </head>
    <body>
        <name-badge contactname="Another Person"></name-badge>
    </body>
    <template id="mytemplate">
         <style>
              .badge { width:200px;text-align:center; }
              .name { padding:10px;border-radius:5px 5px 0px 0px; border-
style:solid;border-color:black;border-width:2px 2px 0px 2px;background-
color:red;color:white;font-weight:bold;}
              .details { padding:10px;font-style:italic;border-radius:0px 0px
5px 5px; border-style:solid;border-color:black;border-width:1px 2px 2px
2px;background-color:white;}
            </style>
            <div class="badge">
            <div class="name"><span id="name"></span></div>
            <div class="details">Trainer Info Support bv</div>
            </div>
    </template>
    <script>
    customElements.define('name-badge', class extends HTMLElement {
        constructor() {
            super();
            var temp = document.querySelector("#mytemplate").content;
            this.appendChild(document.importNode(temp, true));
            this.querySelector("#name").innerText = this.contactname;
        }
       get contactname() {
          if (this.hasAttribute('contactname')){
            return this.getAttribute('contactname');
          }
          return "John Gorter";
       }
       set contactname(val) {
         this.setAttribute('contactname', val);
         this.querySelector("#name").innerText = val;
       }
    });
    </script>
</html>
```

Step 5. Use the Shadow DOM to encapsulate the element

Open the HTML page and add the following HTML into the body of the page,
just below the opening body tag:

```
<div><h1>Polymer Trainer List</h1></div>
<hr/>
```

Change the style inside the template for the name badge and add the following style-rule:

```
    div { font-family:arial;color:gray;}
```

Run the page inside the browser and note how every text implicitly is arial and colored gray.

Our new style rule changed the whole document. That was not the intention. Let's implement Shadow DOM.

Change the script where the template is instantiated to create a shadowDomRoot. Attach the content of
the template to this shadowDomRoot.

Your code should look something like:

```
    customElements.define('name-badge', class extends HTMLElement {
        constructor() {
            super();
            var temp = document.querySelector("#mytemplate").content;
            var root = this.createShadowRoot();
            root.appendChild(document.importNode(temp, true));
            root.querySelector("#name").innerText = this.contactname;
        }
       get contactname() {
          if (this.hasAttribute('contactname')){
            return this.getAttribute('contactname');
          }
          return "John Gorter";
       }
       set contactname(val) {
         this.setAttribute('contactname', val);
         this.querySelector("#name").innerText = val;
       }
    });
```

Refresh the page. If all went well the document should be displaying a black header, whilst the details of
the name-badge is still gray.

Step 6. Use HTMLImports to structure the end result

The page is beginning to grow, can you image what the page would look like when we were to create a lot of new elements? How are these elements even reusable when they are coded into the page.

Create a new file inside the same directory. Name the file 'name-badge.html'. Copy and paste all code and markup related to this element into the newly created file.

Open the index.html and use the following include link

```
  <link  rel="import" href="name-badge.html"/>
```

to include the namebadge into your page.

Open the name-badge.html file and change the code in the script to:

- Get the the template from the import and add it to the body of the document

```
// first script to execute.. add the template to the document structure -->
document.body.appendChild(document.querySelector('link[rel="import"]').import.q
uerySelector("#mytemplate"));
```

- Define the custom tag and use the template from the import to stamp out a new DOM into the Shadow DOM.

```
// inside the definition, grab the template from the body..
var temp = document.querySelector("#mytemplate").content;
```

If all went well, your code should look like this:

```
    <template id="mytemplate">
         <style>
              div { font-family:arial;color:gray;}
              .badge { width:200px;text-align:center; }
              .name { padding:10px;border-radius:5px 5px 0px 0px; border-
style:solid;border-color:black;border-width:2px 2px 0px 2px;background-
color:red;color:white;font-weight:bold;}
              .details { padding:10px;font-style:italic;border-radius:0px 0px
5px 5px; border-style:solid;border-color:black;border-width:1px 2px 2px
2px;background-color:white;}
            </style>
            <div class="badge">
            <div class="name"><span id="name"></span></div>
            <div class="details">Trainer Info Support bv</div>
            </div>
    </template>
    <script>
    // Add the template to the document structure -->

document.body.appendChild(document.querySelector('link[rel="import"]').import.q
uerySelector("#mytemplate"));
    // Define the custom element
    customElements.define('name-badge', class extends HTMLElement {
       connectedCallback() {
            var temp = document.querySelector("#mytemplate").content;
            var root = this.createShadowRoot();
            root.appendChild(document.importNode(temp, true));
            root.querySelector("#name").innerText = this.contactname;
       }
       get contactname() {
          if (this.hasAttribute('contactname')){
            return this.getAttribute('contactname');
          }
          return "John Gorter";
       }
       set contactname(val) {
         this.setAttribute('contactname', val);
         this.querySelector("#name").innerText = val;
       }
    });
    </script>
```

Note how we used the connectedCallback instead of the constructor to delay the creation of the template to the moment it is actually used in the document.

Test the setup.

This time, we have to run the browser from a webserver, since the browser needs to get additional resources and does this by making an AJAX request.

Open a NodeJS prompt and navigate to the labfolder where the index.html file resides and run:

```
$ lite-server
```

Check to see if the page still works.

Extra Exercise

The name is an attribute but the details of our name-badge is still hard-coded. Try to complete the
name-badge by allowing inner tags to the name-badge for the details pane.

For example:

```
<name-badge><section details>CEO Apple</section></name-badge>
```

should work

The final solution in the docs folder contain a fully working example, compare your results with the files from the solution.

Summary

We made use of 4 features from the HTML5 spec to create web components. We've used Custom Elements, Templates, Shadow DOM and HTMLImports. Each of these features have a purpose of their own but combined they create an opportunity to write web components.

However, to creat reusable web components, we have written a lot of boilerplate code.
This is where Polymer and other libraries help. Let's dive into Polymer..

-= End of lab =-