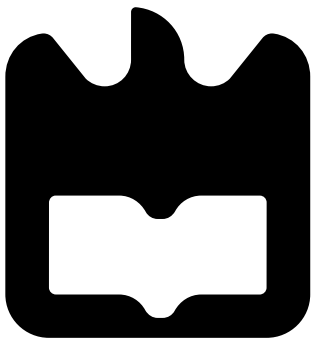
Universidade de Aveiro Departamento de Electrónica, Telecomunicações e Informática,
2015

Sérgio Daniel Marques Cunha

Conceção e Desenvolvimento de uma Plataforma de Componentes Web em HTML5

HTML 5 Framework Design and Development Based on Web Components



Sérgio Daniel Marques Cunha

Conceção e Desenvolvimento de uma Plataforma de Componentes Web em HTML5

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Sistemas de Informação, realizada sob a orientação científica do Doutor Hélder Troca Zagalo, Professor do Departamento de Electrónica, Telecomunicações e Informática, da Universidade de Aveiro. O trabalho descrito nesta dissertação foi realizado na empresa Nokia Solutions and Networks, sob a supervisão do Engenheiro Nuno Miguel Tavares de Sousa, Engenheiro de Desenvolvimento de Software da Nokia Solutions and Networks.

Sérgio Daniel Marques Cunha

HTML 5 Framework Design and Development Based on Web Components

Dissertation submitted to the University of Aveiro to fulfill the requirements to obtain a Master's degree in Information Systems, held under the scientific guidance of Professor Hélder Troca Zagalo, Department of Electronics, Telecommunications and Computer Science from the University of Aveiro. The work described in this thesis was carried out in the company Nokia Solutions and Networks, under the supervision of Engineer Nuno Miguel Tavares de Sousa, Software Development Engineer at Nokia Solutions and Networks.

o júri / the jury

presidente / president

Professor Doutor Joaquim Arnaldo Carvalho Martins
Professor Catedrático da Universidade de Aveiro

vogais / examiners committee

Professor Doutor Hélder Troca Zagalo
Professor Auxiliar da Universidade de Aveiro
(Orientador)

Professor Doutor Fernando Joaquim Lopes Moreira
Professor Associado da Universidade Portucalense
(Arguente Principal)

agradecimentos

Gostaria de agradecer ao meu orientador, o Professor Doutor Hélder Troca Zagalo, por todo o apoio e conselhos ao longo deste ano. Gostaria também de agradecer à Nokia Solutions and Networks por me ter proporcionado esta experiência de aprendizagem e evolução, assim como a todas as pessoas que direta ou indiretamente apoiaram, ajudaram a crescer, pelo esforço e pela maneira como me receberam.

Não posso por fim deixar de agradecer aos meus pais, ao meu irmão e a todos os meus amigos por serem pacientes, darem força, animo e estarem sempre presentes. Obrigado.

palavras-chave HTML5, Componentes Web, JavaScript, AngularJS

resumo Com a evolução da web e com uma maior dependência das aplicações web, surge a necessidade de se desenvolver aplicações web de uma forma mais rápida e modular, deixando então de se repetir pedaços de código e permitindo identificar de uma forma mais eficiente o que está a ser introduzido na aplicação. Com a introdução do conceito de Componentes Web surgem os conceitos de componentes modulares, reutilizáveis e isolados com os quais é possível desenvolver aplicações de uma forma mais rápida e eficaz. Este trabalho surge numa colaboração entre a empresa Nokia Solutions and Networks e a Universidade de Aveiro, onde foi iniciada a construção de uma plataforma de componentes e implementação dos mesmos num produto existente, o Nokia Performance Manager.

keywords HTML5, Web Components, JavaScript, AngularJS

abstract With evolution of web and with a more dependency on web applications, emerge the need to develop web applications in a easy and modular way. With a modular approach it is possible to stop replicate code and allow the identification of components in a application. With the introduction of Web Components emerge the concept of modular, reusable and isolated components with which it is possible to develop applications in a easy and faster way. This work was a collaboration between the company Nokia Solutions and Networks and University of Aveiro, where was started the construction of a framework of components and implementation of the components at the Nokia Performance Manager, an existing product.

Contents

List of Figures

List of Tables

Acronyms

API	Application Programming Interface
ARIA	Accessible Rich Internet Applications
ATDD	Acceptance Test-Driven Development
CI	Continuous Integration
CSS	Cascade Style Sheet
DOM	Document Object Model
E2E	End to End
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IDE	Integrated Development Environment
LTE	Long-Term Evolution
MVC	Model View Control
NCName	Non-Colonized Name
MXML	Magic/Macromedia eXtensible Markup Language
TDD	Test Driven Development
UI	User Interface
W3C	World Wide Web Consortium
WAI	Web Accessibility Initiative

Chapter 1

Introduction

1.1 Context

The development of web applications have evolved over the past years, and with this evolution came the great appearance of new tools, frameworks and libraries, that help on developing web applications. Web applications are used nowadays to deliver any type of content to the end users. With a need to develop new and better applications in short periods of time it is desired a reformulation on the actual state of developing web applications. The need to create applications in short periods of time is not a problem, but the time spent on design and creation of areas is. The actual solution is to create content and reproduce the implementation by repeating over and over again, this approach has a huge problem, when it is time to do changes or corrections on some duplicated content, that some times have specifications. To reduce this handicap in production time some frameworks and libraries introduced a kind of templating technique who allow to create bundles of code that can be reused and distributed as desired. But this solutions is not the best, and to uniform the implementation and to meet the requirements from the developers, the World Wide Web Consortium (W3C) has proposed new standards for the version 5 of HyperText Markup Language (HTML), the Web Components working draft standards, that will help on future development of web content. With these working draft standards it is expected to have more commitment on the possibilities to build web applications in a more effective way that enhance the development and solutions for the described problems.

In order to change the way on how web applications are built, with an isolation of concepts in components by giving each component a self explanatory name, an encapsulation of DOM content allowing to use the same components without interfering with each other, a template that will scaffold the component with the desired structure and at last the HTML import which allow to import each of the components to the web application, were created the actual working draft standards of Web Components. The Web Components are trying to simplify the development, reuse and distribution of components that can even be extended as expected

for each usage.

1.2 Objectives

Facing the demand of developing components that can be used and distributed, in this dissertation the current working draft standards of Web Components and alternative solutions to implement components were studied and analyzed and different solutions were experimented on how to implement components based on Web Components. With the key objectives:

- study the working draft standards of Web Components;
- study the actual possibilities to implement components;
- design and develop a framework to create components;
- develop components in the proposed framework;
- study a possible solution to integrate the developed components with a real solution;
- integrate the components with the actual solution.

1.3 Contributions

This dissertation was developed in an enterprise environment, working on a telecommunication company called Nokia Solutions and Networks. A company renowned for its products in telecommunication market, widely used and that has a constant urge to improve and to do better every day. There was a need to improve the present frontend implementation of Nokia Performance Manager, into an actual, versatile and modular implementation. From this dissertation resulted several contributions to the business.

At first, the solution made is now under usage to continuously develop more components and build a better framework of components, offering the possibility to build better applications and faster in future. Secondly there was an improvement of a present solution on Nokia Solutions and Networks, that has already made the integration of the developed components into this solution and delivered to clients.

1.4 Document Structure

This document is structured in 4 chapters:

- Customizable Web - in this chapter is described the working draft standards of Web Components and frameworks and libraries that allow to create components. Also, a comparison is done in order to perceive the loading time and compatibility within different browsers;

- Developing Web Components in AngularJS - in this chapter are presented the environment to develop and the implementation of the components;
- A case study of Web Components integration based in AngularJS - in this chapter the production methodologies and product are presented with an overview over the integration done;
- Conclusion - in this chapter is summarized the conclusions of the work done, and some future work.

Chapter 2

Customizable Web

In order to pave a way toward a solution and to acquire knowledge and to contextualize on the technologies involved, an investigation over the current state of Web Components was made.

On this chapter, a resume over the current state of Web Components working draft standards will be made, in order to acquire knowledge and better understand the possibilities given from HTML5 to build custom widgets that can be distributed and reused. Also, an overview on the concerns of implementing Web Components today and what to consider to implement them. In the end, an effort to provide and present a list of actual and relevant solutions on how to build components with a non native approach.

2.1 Web Components

The web has evolved and so have the requirements of the users and developers to develop faster and without entropy. The appearance of frameworks that allow developers to create templates, manipulate JavaScript with encapsulation and create custom tags to smooth the development and stimulate reuse of existing pieces of code, pushed the web development to another level. The Web Components working draft standards were created to fill the gap in the HTML standards. The proposed standards focus on the needs of developers of reusable components, namely: code reuse, simplified development and proper script and style encapsulation. The Web Components working draft standards, until today, are still a draft and in continuous development. A draft stage does not mean that it can not be used: they can be used but they are on a continuous evolution and the work done based on a draft may need to be refactored. The Web Components working draft standards are: **Shadow DOM**, **Custom Elements** and **HTML Imports**, the **Template** is already part of the HTML5 recommendation[1].

Web Components give the possibility to develop code which reflects the name that is being declared instead of using abstract and common tags that are agnostic and do not give any

information on what is being done. Also, the Web Components allow to avoid blend of code with encapsulation.

2.1.1 Custom Elements

When writing or reviewing existing code, a developer prefers to write less comments and more self explanatory code. For example, having functions with suggestive names tells to who are seeing the code for the first time what a function does. In HTML the concept of code with proper names does not exist, only in JavaScript, but Custom Elements brings the possibility to specify, on a web page, what each of the tags do. In other programming languages, the developer can give names to functions and variables. In HTML, if the code of a page is inspected there are a lot of div tags, the entropy to understand what the tags do is huge. A Custom Element should have a self explanatory name, helping the developer to create pieces of code and wrap that content within a specific HTML tag, as for instance the **video** tag[2]. With Custom Elements it is possible to:

- Define new HTML/DOM elements;
- Create elements that extend from another element;
- Bundle together a set of functionalities into one tag;
- Extend the API of a DOM element.

The Custom Element must follow some rules in name tagging. It must be a sequence of characters, must match the NCName production, must contain a hyphen and must not contain any uppercase letter[2].

Registering New Elements

New elements must be registered for the web browser acknowledge the element when parsing the web page. The Custom Elements working draft standard defines the **document.registerElement()** method that enables developers to register new elements. This function takes two arguments, first is the name of the new element and the second is an optional object that prototypes an existing element from which element it inherits, as seen on listing 2.1.

Listing 2.1: Registering new element

```
1 var loremIpsum = document.registerElement('lorem-ipsum');
2 document.body.appendChild(new loremIpsum());
```

The call to document.registerElement('lorem-ipsum') teaches the browser this new element and creates a constructor that can be used to create instances.

Extending elements

Custom Elements allow to extend existing HTML elements as well as other custom elements. To extend an element, the new prototype element must be given to **registerElement()**. as seen on listing 2.2.

Listing 2.2: Extending Native Elements

```
1 var superSmall = document.registerElement('super-small', {
2     prototype: Object.create(HTMLSmallElement.prototype),
3     extends: 'small'
4 });
5 <small is="super-small">
```

Custom Elements that are inherited from Native Elements(ex: 2.2) need to be declared as "element foo is bar". This extension is called *type extension custom elements*, as they inherit from a **HTMLElement**, as seen on listing 2.3.

Listing 2.3: Extending Custom Elements

```
1 var loremIpsumExtended = document.registerElement('
    lorem-ipsum-extended', {
2     prototype: Object.create(HTMLElement.prototype);
3     extends: 'lorem-ipsum'
4 });
```

To simply inherit from a Custom Element and create a **lorem-ipsum-extended** tag, it should inherit its prototype from **lorem-ipsum** (ex: 2.3)

2.1.2 Shadow DOM

Nowadays, web sites contain many animations and styling with CSS, processing components and behavior of elements with JavaScript. How can many CSS and JavaScript components developed by different teams work on the same page without spoiling previous or future functions or style? Encapsulation is necessary to ensure the components work without interfering with each other, giving more focus on the component being developed and not on how to change existing code of the page. Some of the problems that the lack of encapsulation can introduce are: collision of stylesheets; unintentional changes to widget structures; JavaScript collisions and malfunctions; leak of private information and a non working page.

Shadow DOM introduces encapsulation on HTML. This standard might be changed, because it is still a draft, but nowadays a similar implementation is used under browser engines to encapsulate the flow of each tag. As an example the video and audio tag present on HTML5[3].

A web page is composed by a document tree also known as **node tree**. The root of tree that makes the web page is called the root node. The node tree elements can have zero or more node trees associated.

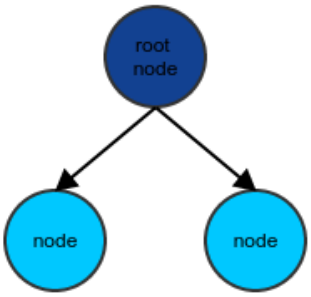


Figure 2.1: DOM tree example

A **shadow host** is the element wherewith all DOM nodes will attach, constructing a tree of nodes. This shadow host, hosts a shadow root, the main node of the tree.

The **shadow root** is an element added to the shadow host, which will be the root node to the Shadow DOM. This root node can have various trees of DOM elements.

How Shadow DOM works

A developer must create a **shadow host** and a **shadow root**. This will create a new DOM tree that can now be filled with elements. The elements that will be part of the **shadow root** can be included as *innerHTML* or *appendChild*. With *appendChild* it is possible to use another component already, defined by the template where the content that will be part of the **shadow root** is defined. With *innerHTML* the inclusion is made in-line. The inclusion with *innerHTML* can be done to replace all the content inside a node, because it replaces all the content with the defined on *innerHTML*. The problem of encapsulation within the same tags is solved, as can be seen in the following example. The color applied on the *h1* tag only applies to outer shadow DOM elements, the elements inside have their own styles. As seen on listing 2.4.

Listing 2.4: Element Inclusion Example

```
1 <html>
2 <head>
3   <meta charset="utf-8">
4   <title>Shadow DOM</title>
5   <style>
6     h1 {color: blue};
7   </style>
8 </head>
```

```
<body>
  <h1>Blue Title</h1>
  <div id='shadowHost'></div>
  <template>
    <h1>appendChild</h1>
  </template>
  <script>
    var template = document.querySelector('template');
    var host = document.getElementById('shadowHost');
    var root = host.createShadowRoot();
    root.innerHTML = '<h1>inlineHTML</h1>';
    root.appendChild(document.importNode(template.content, true));
  </script>
</body>
</html>
```

Tree of Trees

The Shadow DOM is a tree filled with DOM nodes. As it was seen in figure 2.1 there is one root node, the first tree element and consequent nodes. Also, a Shadow DOM can have multiple trees inside a DOM tree. See figure 2.2 for an example. When there is a shadow root that points to an existing child node of a shadow tree, that child node and their child nodes becomes disabled. The new tree became part of the existing one, as if it made a replacement of the nodes.

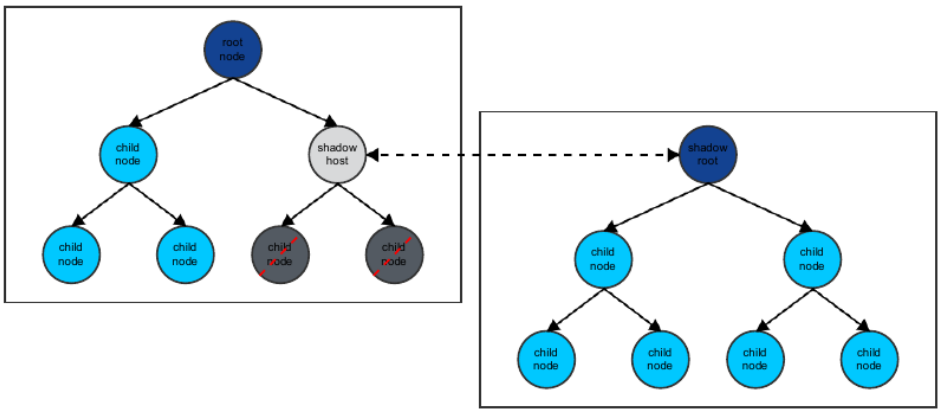


Figure 2.2: DOM tree of tree example

2.1.3 Templates

The template definition on Oxford Dictionaries in subsection Computing is “A preset format for a document or file”[4]. When the HTML was developed and implemented, the

template was not a requirement and was not integrated in the standard. As the time and the requirements have evolved, also, the developers require the ability to template code in HTML.

There are different solutions to template content: Underscore, Mustache, Handlebars, etc. They have helped in various ways developers to create pieces of code to reuse and avoid the replication of code. But these solutions have their own implementation and interpretation of what a template is. They have the pros and cons of using a third party solutions: they are addictions that were designed to simplify the creation of an application, but they fail when you mix several third party solutions on a single application. As it was an imperative that a template were a missing piece from HTML, the W3C created the template standard, for a native implementation on different browsers. Therefore the developer does not need to import or use third party frameworks or libraries[5]. The **template** element provides a method to include fragments of HTML, which are called **template contents**.

Advantages of templating

Wrapping content inside a template gives the following advantages:

- The content inside template is inactive and hidden inside the DOM, it is only rendered by request.
- The content of the template is considered not to be in the document. Trying to access an element by its id, using *document.getElementById()* for example, will not return any child node.
- Templates can be placed anywhere, inside **header** or **body** and contain any content.

Declare a template

The template element should be declared first to be used and must contain an id. This id will be used to register the template on the DOM. An example is present on listing 2.5.

Listing 2.5: Template Example

```
1 <template id="helloTemplate">
2     <h1>Hello Template!</h1>
3 </template>
```

Activate a template

After declaring a template, it must be registered on the DOM to be used by an application. There is more than one way to activate a template. One of them is by appending the template content using **document.importNode()**. An example is present on listing 2.6.

Listing 2.6: Example Activation Template

```
//fetch the template
var template = document.querySelector('#helloTemplate');
//clone the template
var templateClone = document.importNode(template.content, true);
//append the cloned template to body
document.body.appendChild(templateClone);
```

Template Example

Below there is an example on how to declare, activate and use a template element, on listing 2.7.

Listing 2.7: Template Example

```
<button onclick="templateUse()">Click!</button>
<div id="container"> </div>
<script>
  function templateUse() {
    var content = document.querySelector('template').content;
    var p = content.querySelector('p');
    p.textContent = parseInt(p.textContent) + 1;
    //Activate and use of Template
    document.querySelector('#container').appendChild(
        document.importNode(content, true));
  }
</script>
//Declaration of template
<template>
  <div>Total clicks: <p>0</p> </div>
</template>
```

2.1.4 HTML Imports

With HTML Imports is possible to import HTML files into the current document, as it currently does for CSS and JavaScript. Importing an HTML page will include its content in the current web page and apply all CSS styles to it and run any JavaScript defined in it. The possibility to import a page that contains imports for CSS and JavaScript allows to include themes, libraries, frameworks, sections of applications and also complete applications[6]. To make an import, simply do a link as currently is made for CSS or JavaScript inside the head of the page. An example on listing 2.8.

Listing 2.8: Import Demo

```
<head>
    <link rel="import" href="plugins.html">
</head>
```

2.2 Implementing Web Components today

The proposed standards that make up the Web Components Initiative are still in draft, but it is possible to implement Web Components today to some extent. To do that the developer needs to know the possible solutions and alternatives to native implementations that are available for Web Browsers that do not support Web Components. A developer should also be aware of several important issues when developing a Web Component. For instance: is the component Accessible? What browsers does it support? Does it allow templating? Is it testable? In this section will look in more detail at these aspects.

2.2.1 Accessibility

The World Wide Web was first created to share knowledge and information. Today it is used for almost everything: from social applications to gaming and real time meetings. The information shared on the web is not only text, but is enriched with images, animations, video, graphics, a lot of content that someone with disabilities cannot see and enjoy. In particular case people who rely on screen readers and who are not able to use a mouse. This has led to the development of standards that allows all users to interact with the application. The applications that use those standards are commonly called Accessible Rich Internet Applications (ARIA)[7].

For instance, when a web developer creates a collapsible tree of widgets in HTML with CSS and JavaScript. To non-disabled user it looks like a collapsible tree, but without proper semantic, all the tree may not be perceptible or operable by a user with disabilities[8].

Incorporating the WAI-ARIA is a way for the author to incorporate proper semantics in custom widgets and make these widgets accessible, usable and interoperable with assistant technologies.

WAI-ARIA relies on user agents, they are the applications responsible for interpreting the given semantics. If the application has complex structures and the elements are not associated to a WAI-ARIA role with the appropriate states and properties, there won't be enough semantic information for user agents to process.

2.2.2 Browser Compatibility

When implementing Web Components the developer needs to be aware of the purpose and the target audience of the application: what is the the purpose and the target audience of the application. Web Components rely on W3C Standards that not all browsers support. The

Web Components working draft standards are still in a draft stage, except for the Template Standard. Web Browsers are still making their implementations. The only browsers that support natively all the working draft standards are Chrome and Opera. Firefox and Safari only support Template, Firefox also have the: Custom Element, Shadow DOM and HTML Import working draft Standards implementations on roadmap but are not implemented yet. Internet Explorer does not support any of the working draft standards and they are still under consideration for implementation[9][10].

There is a workaround to have a glimpse of Web Components on browsers that do not have a native implementation. The community has found a way to fill the gap on non supporting web browsers and use web components working draft standard across modern web browsers today, the polyfills.

Table 2.1: Compatibility

	IE 11	Chrome	Firefox	Safari
Custom Element	✗	✓	◆	✗
HTML Imports	✗	✓	◆	✗
Templates	✗	✓	✓	✓
Shadow DOM	✗	✓	◆	✗

(a) ◆ - The feature can be activated by setting a flag to on, but it is not active by default.

2.2.3 Polyfills

A polyfill, as mentioned in 2.2.2, fills the gaps in the Web Browser to create support for all the Web Components proposed standards. This behavior can be achieved across all modern browsers, that is: latest Firefox, Chrome and Opera, Safari 7+ and Internet Explorer 10+. This implementation is made in JavaScript[11]. If a browser already has implemented any of the working draft standards, then the polyfill will detect the native support and switch to the fast path.

The polyfills are also available in separate components. If it is only required the Shadow DOM standard support, then it can be loaded separately and avoid unused content. If the Web Components polyfill is included, then the Template, Shadow DOM, Custom Elements and HTML Import working draft standards are available to the application.

2.2.4 Templating

As seen on description of the template standard, a template allows to reuse code, to avoid duplicated code. Reusing an already crafted piece of independent code will result in a better experience for the programmer.

To develop the web components today one of the “must have” features on choosing the framework/library is the template ability. If this template ability is compliant with the actual

standard it will make it easy to migrate in the future were the Template Standard is widely implemented on browsers.

2.2.5 Testability

When developing new applications, or simply developing new features for already established applications, there is always a concern if the outcome will be the expected and also to minimize the scatter of unwanted code. There are many ways to test an application, but from a Web Component perspective there are two types of test to support: unit tests and end to end tests. Unit tests, as the name suggests, are used to test units of code. Unit testing has become an accepted best practice in software development. Having testable code brings a lot of desired benefits, for instance: code modularity, code readability and a safe net to make changes without breaking existing functionality.

There are third party libraries and framework that facilitate the development of tests. In Web Components there is an environment to test the components, the Web Component Tester[12]. The tests are done in JavaScript and can run in any browser, to check for problems within different engines.

2.2.6 Web Components Compatibility/Integration

Web Components are a major building block on the development of Web Applications. They enable the coexistence of Web Components from different developers or vendors on the same Web Page without additional effort. The components are innocuous in the DOM: they are just DOM nodes. Any framework or library working in the page does not know that those nodes are Web Components: for them they are simple DOM nodes. Web Components can coexist on the same page, but if Web Components are used with other frameworks the developer will get the same problems as when mixing several frameworks on the same page. For instance, mixing Web Components with AngularJS might cause problems. AngularJS provides something similar to Web Components through its directives. Directives work well and do not interfere with each other if properly developed, but you can not treat a directive as a standard Web Component. When mixing AngularJS directives, as explained in depth in section 2.3.3, and Web Components the developer must be conscious of the problems that might occur since AngularJS directives are not true Web Components. Other frameworks, like Polymer, do not have problems when mixing their components with Web Components because they just provide a thin layer over native Web Components that makes writing standard Web Components easier.

After all the research done on the proposed standards which may be used to create Web Components, the conclusion is that it is premature to use those working draft standards on projects that have a need to be stable and maintainable over some years without deep changes to its core. The safer route for a long term application is to choose a framework or library

that provides a development environment similar to what Web Components provide, since the proposed standards may change and force to redo already crafted components.

2.3 Framework's/Library's Analysis

To implement a native web component it is required that all of the proposed standards mentioned on section 2.1 are implemented, taking care of all tiny aspects. What if the web components implementation can be simplified using a tiny layer, a framework or a library to implement web components? In this section will explore the major alternatives to develop a Web Components based methodology. One important feature that will be used over this section is the mutation observer. **Mutation Observer** is a function present in the JavaScript engine, that a developer can use to check for DOM changes. This feature is important due to the need to check for changes in elements on any application in order to take actions within every changed element. Every time that the terminology mutation is applied, it refers to a change in a element through mutation observer[13][14][15].

2.3.1 Polymer

A framework ahead of its time, allowing web components to be developed today, for Evergreen browsers[1]. Polymer project extends the web components and makes it easier to create any component, from a single banner to a complete application[16]. To be able to implement all of these components so early, the Polymer project, together with the help from the community, developed a "Foundation" as shown on figure 2.3. The missing features from browsers which are present on web components working draft standards were implemented by making polyfills (section 2.2.3).

Polymer introduces an abstraction from the implementation of the web components native APIs. With Polymer it is possible to craft HTML elements with complex computation behind. The focus of this framework is to get back to basics and implement elements on HTML that are as agnostic as possible, to build powerful applications maintaining full adherence to base standards[17].

Listing 2.9: Polymer Example

```
1 <polymer-element name="hello-world" noscript>
2   <template>
3     <h1>Hello World!</h1>
4   </template>
5 </polymer-element>
```

[1] Browsers that automatically updates themselves.

One example of the abstraction created by Polymer is how a component is declared. As can be seen on example 2.9, in Polymer the developer does not have to write any JavaScript to declare a new web component, as it would be required if the standards in section 2.1 were used.

Architecture

The Polymer framework is structured in a 4 layers model that make the Polymer platform, see figure 2.3.

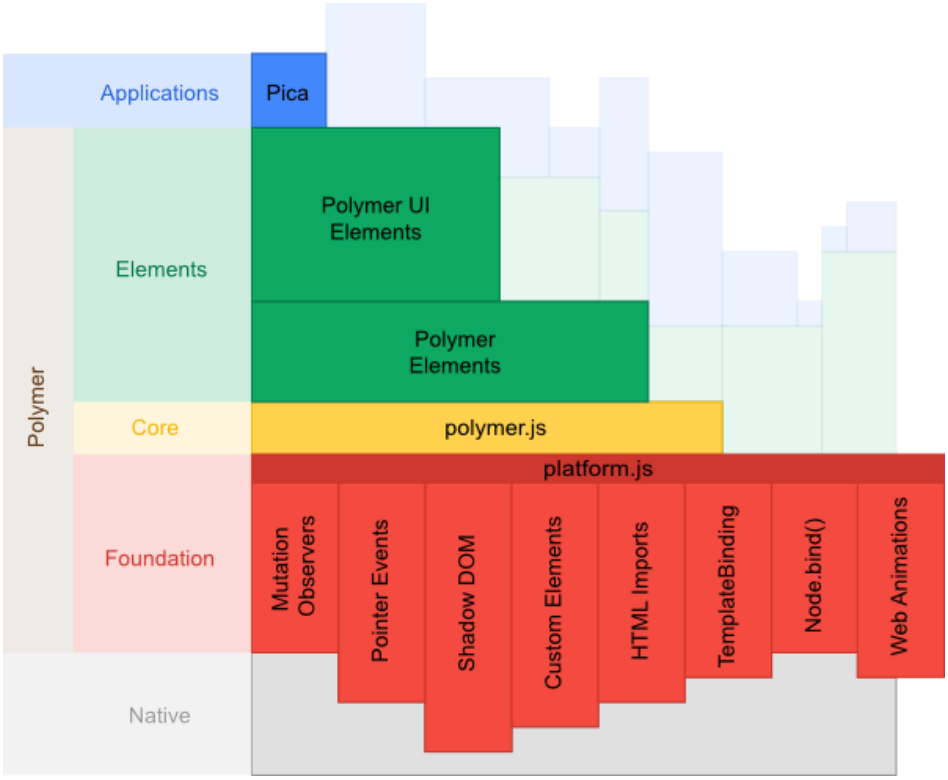


Figure 2.3: Polymer Architecture

The **Native** and **Foundation** described in this figure are the emerging web standards, some are already implemented as native in browsers and others are built in platform.js[2]. Both Foundation (polyfills) and Native are in the same level, if there is some feature present on a browser, then the polyfills would not load that requirement. Ex: Chrome, latest version, already has Shadow DOM implemented, the Shadow DOM from the polyfills wont be loaded because the browser already supports it. The **Core** layer is the Polymer framework implementation on top of the Web Components, this layer is the responsible, like a maestro, on

[2]The platform.js has changed to be a webcomponents.js provided by the Web Components initiative. Also, for polymer 0.8, the developers are using a subset of the standard and their own polyfill

how foundation and native work together. The **Elements** layer are the Custom Elements implemented and available on Polymer[18]. As example the "core-icon" displays an icon[19].

2.3.2 React

React is a JavaScript library to build and develop user interfaces. The React implementation differs from Web Components, is more focused on user interface. The key benefits to implement a component with React is the complete independence from the polyfills. One of the main features from React is the way the information is presented on the page and how it is always watching for changes to update the view. This is possible because of a specific implementation for DOM manipulation: the Virtual DOM.

Virtual DOM

The Virtual DOM creates, as the name suggest, a virtual node element. An element in React virtual DOM is a **ReactElement**. A different approach to DOM elements, creating an independent implementation of DOM, goes away from native implementation and from the compatibility problems.

Listing 2.10: React Element

```
1 var root = React.createElement('h1');
```

One major aspect of React is the higher performance obtained from how Virtual DOM works. The React uses several clever techniques that minimize the DOM operations and consequently reduce the response time. Virtual DOM creates a simple representation on DOM of a DOM sub-tree. The nodes are mapped in the DOM as usual and React uses Virutal DOM to create their duplicated and simpler version, avoiding creating and accessing DOM nodes. Everytime that a Virtual DOM node changes, the function "shouldComponentUpdate" will return true and will only update the node that needs to be updated, avoiding as possible the DOM mutations[20][21].

Architecture

React alone does not have an architecture as it is a user interface library, it can be used with any framework but there is an application that compose React architecture, it is Flux. Flux uses a unidirectional flow of data. Flux was developed with the insight to create an architecture to React, with a specific and focus on uni directional biding of information. Instead of using a MVC model, it uses a dispatcher, a store and a view. The view layer is the responsible to propagate every changed through a central dispatcher. Then the dispatcher will update the store and the view to match the changes[22].

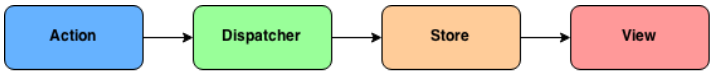


Figure 2.4: Flux Architecture

2.3.3 AngularJS

AngularJS is a framework following the MVC model, rich in features and was designed to create dynamic web pages. It was designed to allow the development of rich applications in HTML, adding the template technique to extend the behavior of the application. AngularJS has a behavior next to Web Components implementation. Almost all of the application work is done on the browser side. It adds useful features to browser such as: **Data Binding**, **Dependency Injection** and **Directives**[23]. In a next Chapter will be dedicated a proper topic on creating AngularJS directives because this can be used to mirror Web Components functionality. Their proponents defend that writing applications with AngularJS is writing less and doing more, this approach is also protected by Unit and E2E testing.

Data Binding

AngularJS uses **Two-Way Data Binding**, a bidirectional method of data binding between the view and the model, from MVC. It allows to maintain a permanent data synchronization between the view and the data model. See figure 2.5 to see the work-flow.

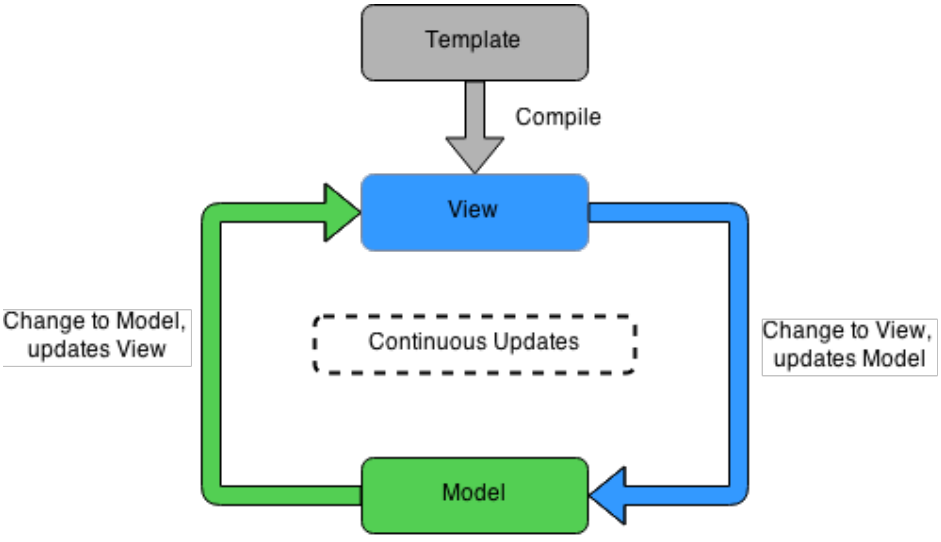


Figure 2.5: AngularJS Two-Way Data Binding

Dependency Injection

A dependency in AngularJS is every component created on it that can be used by other components, can be services provided out of the box by AngularJS, like the HTTP service, but can also be other services, directives or modules developed by a third party. In AngularJS, when developing a component, only need to declare its dependencies since AngularJS takes care of injecting them. With AngularJS it is possible to include the services only by referencing them when making the prototype of what will be done. Having dependency injection gives a good abstraction and flexibility to only include what is needed and AngularJS will do the rest. Recurring to dependency injection is possible to test each component separately.

Directives

While web browsers do not provide a native implementation of the Web Components proposed Standards, the AngularJS team developed a way to create web components. With directives in AngularJS it is possible to have a close approach to web components. A Directive allows to create a new tag, extend existing tags, create attributes to use on any tag and add behavior. A Directive can be used in four different ways, as shown on listing 2.11. The main advantage of directives is that it does not need any layer to work without issues on any browser supporting AngularJS.

Listing 2.11: Angular Directives Type

```
1 <angular-directive></angular-directive>      //Element
2 <div angular-directive='lorem '></div>      //Attribute
3 <!-- directive: angular-directive lorem -->   //Comment
4 <div class="angular-directive: lorem;"></div> //Class
```

Scope

The AngularJS uses the scope as object to the application model, where all the scopes are arranged in a hierarchical structure. The scope structure is a DOM of the application. Scope is an execution context for expressions which will result in events[24].

2.3.4 Others

There are also other libraries that due to a lack of a more visible interest from the community and also because a lack of documentation will only be mentioned as other framework's/libraries in order to keep an eye on them.

X-Tags

The X-Tag is a JavaScript library developed and supported by Mozilla, with the purpose of bringing a part of Web Components to developers. With X-Tag it is possible to use the Custom Element, the HTML Import and Mutation Observers, built upon polyfills[25].

Brick

Brick is not a framework: it is a collection of components. The main focus of Brick is to simplify the development process of the user interfaces, UIs. It does not focus on how the application works on the background. Brick wants to simplify the design of the UI and give the possibility to focus on implementing the ideas from developers. This UI collection of components use the Custom Element standard, keeping the integration with other frameworks under a standard and known API[26].

After an individual overview over the possibilities from frameworks or libraries available, what are the possibilities of using a native implementation or using framework or library instead? In order to address the question a brief comparison can be seen in Table 2.2.

Native implementation and Polymer are similar, both depend on polyfills to work properly and they are the state of the art. React and AngularJS are similar, both have proprietary implementations, mirroring the components which approach the proposed standards. React is still unstable to use as long term library, and AngularJS is well established.

Table 2.2: Features comparison

	Native	Polymer	React	AngularJS
Custom Element	✓	✓	◆	◆
Shadow DOM	✓	✓	◆	◆
Templates	✓	✓	◆	◆
HTML Import	✓	✓	✗	◆
Testability	✓	✓	✓	✓
Browser Compatibility	■	■	✓	✓
Web Components Compatibility	✓	✓	✗	✗
Web Components Integration	✓	✓	✗	✗
Stability	✗	✗	✗	✓

(a) ◆ - The solution has a proprietary implementation of this feature.

(b) ■ - The solution depends from a third party implementation to work properly.

The implementation will not be as working draft standard of Web Components, because the handicap to implement a web component and the problems to make it work without any issue on the majority of Web Browsers will be huge, and for now is not affordable, and also one major demand of this implementation is the stability of the solution.

2.4 Analysis on benchmark and compatibility from principal tools

As information and knowledge enthusiasts, when reading or studying news, articles or simply watching a conference, we pose more requirements on the applications to be more and more polished. Speed matters. As a user I do not want to wait more than some 2 or 3 seconds to start reading or watching the content that is important to me. When constructing a web application, the developer must take some actions to avoid a high number of requests and apply techniques to accelerate the rendering of a web page.

The tests methodology used to gather the total load time of each page in different browsers and a visual check over the print screen taken at the end of each test, that tell if the page was correctly loaded. The validation if the page is correctly loaded and is usable will also be made on manual testing. To collect the data, a set o 10 tests was done for each pair framework/browser. The set of tests were done on the following browsers: Chrome, Firefox, Safari, Internet Explorer 9, Internet Explorer 10 and Internet Explorer 11. The tests focus on Polymer, React and AngularJS. To be more agnostic as possible, the way that was chosen to test the framework/library, was through TodoMVC[3]. All the implementations of a Todo List in MV*, were developed mainly by contributors to the framework/library development, which leads to the inference that is an optimal implementation. The native implementation was not possible to test, due to a lack of an agnostic implementation on TodoMVC or in any similar site. In order to be agnostic and take advantage of an existing solutions, all the tests where executed through WebPageTest[4]. The WebPageTest was developed by AOL[5], but now is on GitHub[6], and is mainly maintained by Google[27]. The instance was run through the WebPageTest site and all the tests where made on the same location, Dulles, VA USA, with the connection always set to Cable (5/1 Mbps 28ms RTT). The aim of this set of tests is to determine the performance level of each framework/library and check some incompatibilities on browsers.

2.4.1 Results

The tests made are identified in two types, a *first view* which are the first time that the page is visited, were the browser is opened and the cache and cookies are clean. Then it is executed the *repeated view* where the page is rendered after a first view procedure. The repeated view uses the resources as a coming back to a visited site.

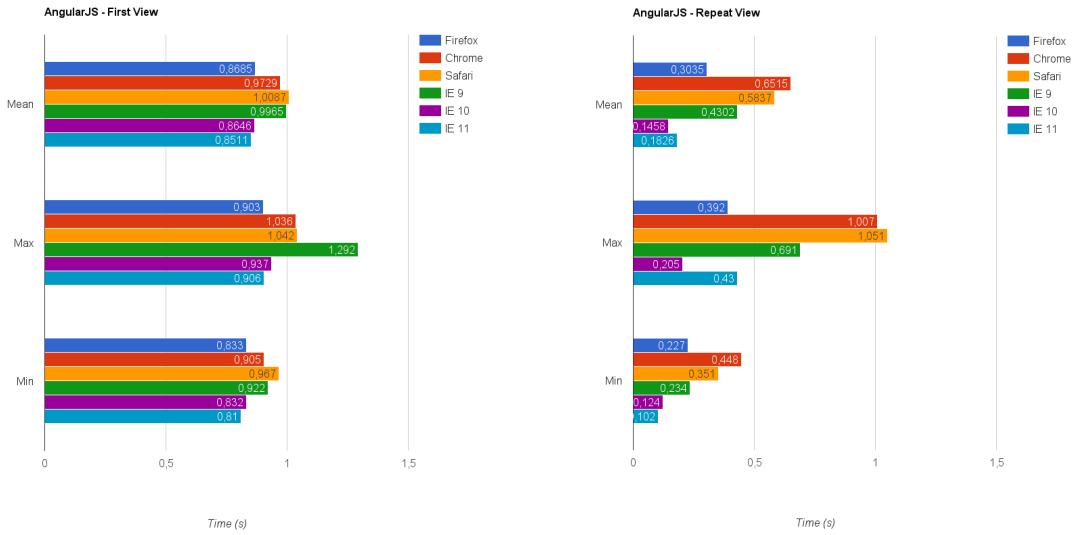
[3]http://todomvc.com/
[4]http://www.webpagetest.org
[5]http://dev.aol.com/
[6]https://github.com/WPO-Foundation/webpagetest

AngularJS

On AngularJS at the first view, the results in average have a response around the 0,8 seconds and 1 seconds in all browsers, which are a good response time. On figure 2.6a are mapped the results of the first view test set.

On the repeated view benchmark from AngularJS, as seen on figure 2.6b, the response time have improved as expected, but with an inexplicable result on Chrome, that ended with the worst average of all the browsers tested. Knowing that AngularJS and Chrome are developed by Google® was expected that the results were better.

In both test types, the page was rendered as expected and is operational in both browsers.



(a) AngularJS - First View Graphic Comparison (b) AngularJS - Repeat View Graphic Comparison

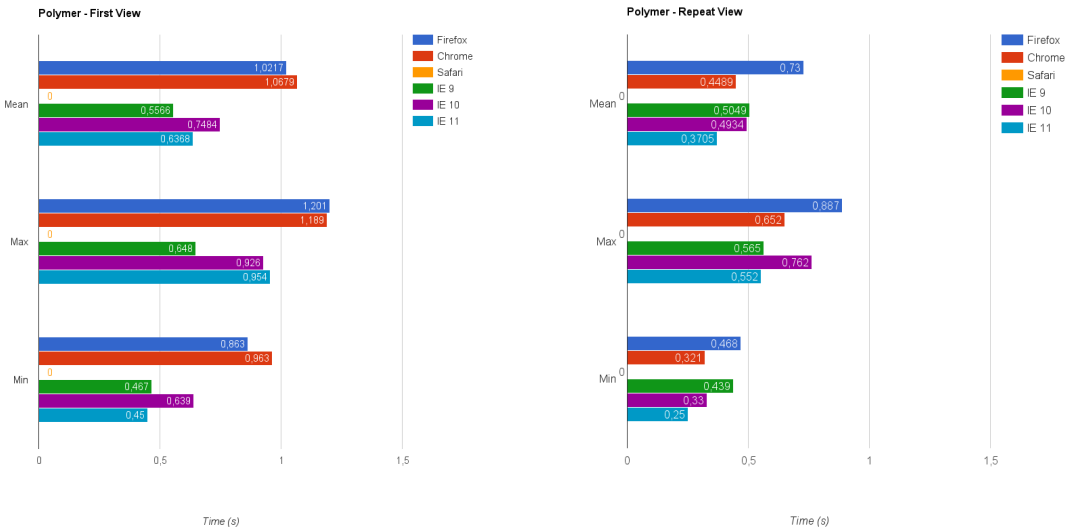
Polymer

On the first view of Polymer test set, as seen on figure 2.7a, the average results have a close behavior on Firefox and Chrome, around the 1s, and better performance on IE 9, 10 and 11. The page was rendered correctly on Firefox, Chrome, Internet Explorer (from version 9 to 11) but on Safari for Windows the page was not rendered and consequently is not functional, this specific case needs a mention as this version is not supported by Apple®. The same page was tested on Safari in Mac OS X and the page has rendered and is functional as expected.

In the repeated view, as seen on figure 2.7b, the better load times are in the engines that was expected a worst performance, due to the lack of implementation of used standards in this example, and the need to use polyfills. But the response time was fast in every browser

excluding Safari in Windows. Also, it is needed a mention on the worst performance from Firefox.

On Internet Explorer the data collected from console seems that was not collected properly, but when inspecting the network activity on page load, all resources was loaded properly.

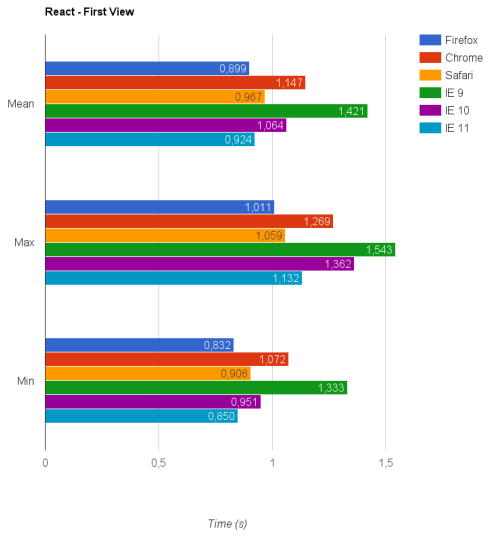


(a) Polymer - First View Graphic Comparison (b) Polymer - Repeat View Graphic Comparison

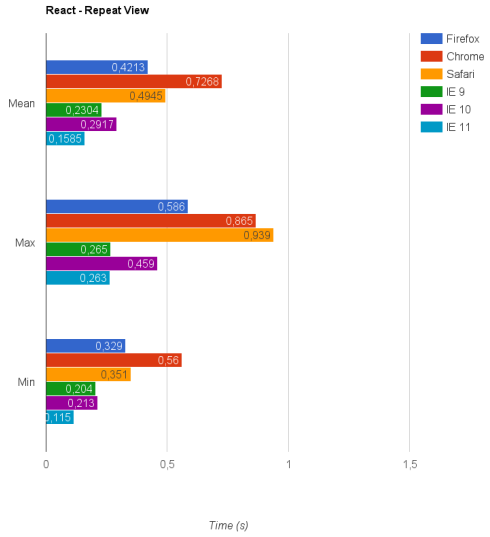
The performance of the tests on each browser in Polymer may be affected by the need to include the polyfills in some cases. Even if the page is rendering on Chrome, a browser that have the web components working draft standards implementation. On Polymer, the results can be even better when the standards are applied cross-browser.

React

On React in the first view, as seen on figure 2.8a, the results are sparser in comparative to AngularJS and Polymer. There are 3 browsers that have rendered the page on an average of more than 1 second.



(a) React - First View Graphic Comparison



(b) React - Repeat View Graphic Comparison

On the repeated view, as seen on figure 2.8a, the worst average result is from Chrome, having 0,7 seconds to open the page. The browsers that have exceeded the other and in this case maybe the results are not reliable is Internet Explorer (from version 9 to 11).

On all tests taken in React there was detected problems in every screenshot done by the WebPageTest that permit the user to check if the page was rendered as desired over all Internet Explorer browsers (from version 9 to 11). Although when making manual testing on all the instances of Internet Explorer, the page is rendered and works as expected. Which suggests that the tests results from Internet Explorer (from version 9 to 11), seems to be unreliable, considering the problems on capturing the screenshot on each test, and if the screenshot was not taken correctly, the load time of the page maybe is not correctly measured.

On the overall test results of React, are very sparse going against the more uniform time collected from AngularJS and Polymer.

2.4.2 Comparative

As a final overview of the executed performance tests, all the tested framework/library will be put face to face in a comparative, allowing the comparison of the average performance results of each one. As seen on figure 2.9 is possible to conclude that AngularJS is the most constant in performance, with results close to each other, but the one that have better results is Polymer. As can be seen easily on figure 2.10.

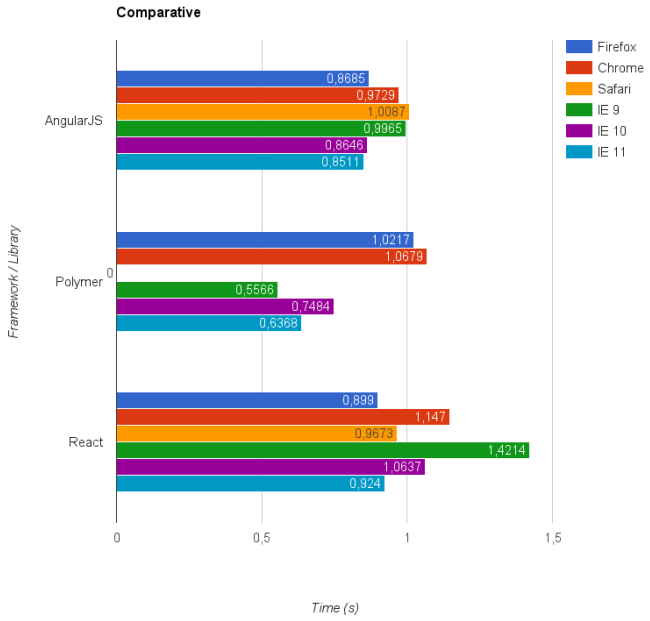


Figure 2.9: Framework / Library Comparative - Average Time

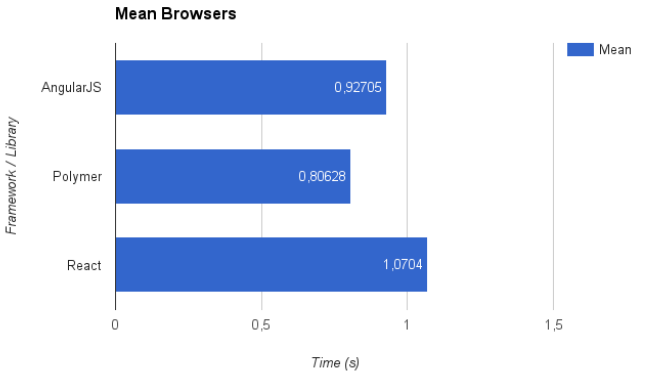


Figure 2.10: Framework / Library Comparative

With the present data on this section there are insights and inferences to be taken. The framework with better result is the one that has an implementation that focus on a native implementation, even the Internet Explorer, from version 9 to 11, that does not support Web Components and have required the polyfills to fill the gap on the engine. React was a surprise, due to the simple implementation mentioned in section 2.3.2, to be the one that had a less good result. But even React is not a bad result, with a response around 1 second is not

meaningful.

With the insight that this is a simple project with lower complexity it is still possible to analyze the results, even that they can be different with a big project that is more complex. Having a overview on all the results from the first view, the differences on rendering in time and in the expected result was not meaningful to have an individual mention. On a repeated view the results are scattered. Chrome and Safari are the browser with the poorest performance, but with no issues to the user.

2.4.3 Community

When it is time to decide on which framework or library to use, the community can also help to chose, either by how many results on articles or questions on a programmers community, such as StackOverflow[7]. On the case that the solutions are developed based on community contributes, the amounts of forks, contributions and commits on Github[8] on each project, easily show the acceptance of the community and the commitment of the developers to develop and upgrade the solution. Also, the community on these cases are a main help when problems appears, on solving bugs and adding new features, with the request from the community. So, having a framework or library that have a huge community on background will help on development of the framework or library and to use it.

On the table 2.3, a collection of data can be analyzed. The AngularJS community shows to be the most active with more contributors and commits. Also, the amount of questions on StackOverflow about AngularJS are greater than Polymer and React together. Having all this data as insight the AngularJS is the framework that has more commitment from the community.

Table 2.3: Community comparison

	Github				Twitter	StackOverflow
	Forks	Stars	Contributors	Commits	Followers	Questions
AngularJS	16.231	38.496	1246	6784	65.472	94.073
Polymer	893	9.688	47	2774	22.789	2.501
React	3.090	21.859	481	4486	21.481	2.712

(a) Data collected on 13-05-2015

2.5 Chapter Summary

Throughout this chapter, several topics and concepts were explained on how to build a web component. To have a better understanding on how web components are paving the future, but already appearing in web applications development.

[7] http://stackoverflow.com/
[8] https://github.com/

After analyzing the web components standard, the aspects that should be addressed when creating an web application and the framework's/libraries it is now possible to draw some conclusions on future implementation. If there is no demand on supporting old browsers and was made the assumption that an application is being built on technology that may change in the future, then the Native Web Components or the Polymer are the perfect solution. Otherwise, if there is a demand and restriction to support a determined web browser, then a more stable and established solution is priority.

With the target of creating and implement the built components, resultant from this work, into an existent product some considerations are needed to have. The actual implementation is a complex, huge and built with Flex application who need to improve into a modular approach, where everyone can build components to specific areas and distribute or reuse already crafted components. With this insight it is needed to have a stable, reusable, with accessibility support and testable implementation environment. AngularJS meet all the requirements presented earlier, with special emphasis: accessibility already supported and ready to use; wide compatibility with browsers which allow to develop without the concerns of developing with workarounds, in special with older versions of browsers; an consolidated marketplace, lots of developers using AngularJS to build their applications; the approach used in AngularJS is similar to native implementation. AngularJS is a versatile framework that allows implementing an application today while keeping the options open for a future version with newer standards. It also has a good community and a good track for acceptance from developers.

Chapter 3

Developing Web Components in AngularJS

After taking the decision that AngularJS will be the framework used to build components. This chapter will focus on all requirements to the components being developed, implementation of the components, an overview on architecture and on defining a module and a directive, the components built and tools used within the process. Also, a brief on testing a component with AngularJS and the process of testing and the process to create documentation for each component. At the end of the chapter it is shown the problems and the solutions encountered.

3.1 Requirements

Following what has been said as the objectives of this work, it is necessary to clearly determine what are the main requirements for this project. All the requirements have a high priority, there are not a requirement that is less important than the other. The implementation is meant to be built with components that are:

- Modular;
- Reusable;
- Automatically Quality Tested;
- Documented.

3.1.1 Modular

During application development it is normal to break the system into several modules, where each module has a specific and isolated responsibility. The same applies to each module in the web components to be developed. A component should solve only a single problem and should be able to be reused on other components and applications.

3.1.2 Reusable

In parallel on development of modular components, the requirement of having reusable components need to be taken in consideration at the same time. With reusable components, each component is designed and developed to solve problems in various projects. The component should allow customization's to fulfill the specific requirements of each project and avoid the development of similar components that are meant to be extended from a main component.

3.1.3 Automatically Quality Tested

Quality is a commitment of the company. To ensure the components are always checked for quality, all quality tests should run automatically. As a guidance to this quality and test approach, Test Driven Development (TDD) is used to ensure that each component is always tested and only has the behavior it needs to deliver its service.

In order to ensure that development or maintenance of a component does not break other components, the Continuous Integration (CI) practice is used to ensure that every component does not contain bugs and has the desired quality. Further in the next section will be presented an overview on CI.

3.1.4 Documented

Using components from third party developers requires a frequent use of documentation to know how to use them and all the features available. Like using third party components, in the development stage of the components there is a need to create documentation that explains and allows the live visualization of the components.

To keep the documentation and examples close to the code, the code and documentation examples are inserted in the code as comments. During the build, all the source code is scanned and, from these comments, the documentation set is generated for all the components.

3.2 Implementing Web Components in AngularJS

After describing all the requirements, (see section 3.1) were needed to define the framework of components structure, and present the tools used during the development. As a start point to create the development of framework, it was needed to design the architecture of the application, to acknowledge how the components will interact with each other. Finally, the guidelines on how to develop components with AngularJS was needed to create an uniform method of development.

3.2.1 Framework Architecture

Following the requirements identified in the previous subsections, the final solution were built with AngularJS directives with an approach based on working draft standards of Web Components, as described on section 2.1. To implement directives as described, each AngularJS directive has the requirement that each directive can be distributed, according the requirement of modularity and reusability. When building an AngularJS application it was needed a *module*, which is the container of any resource used, as example: directive, controller, among others. Following, in order to create modular components, each component need to be included inside of an independent module. With this approach its provided the modularity and reusability requirements. Each AngularJS directive has a *template* and *controller* that created the behavior of the directive. The framework architecture is present on figure 3.1, where is present a module which communicates with every directive inside the framework and imported into the module. Inside of each directive and inside a module are the template and controller.

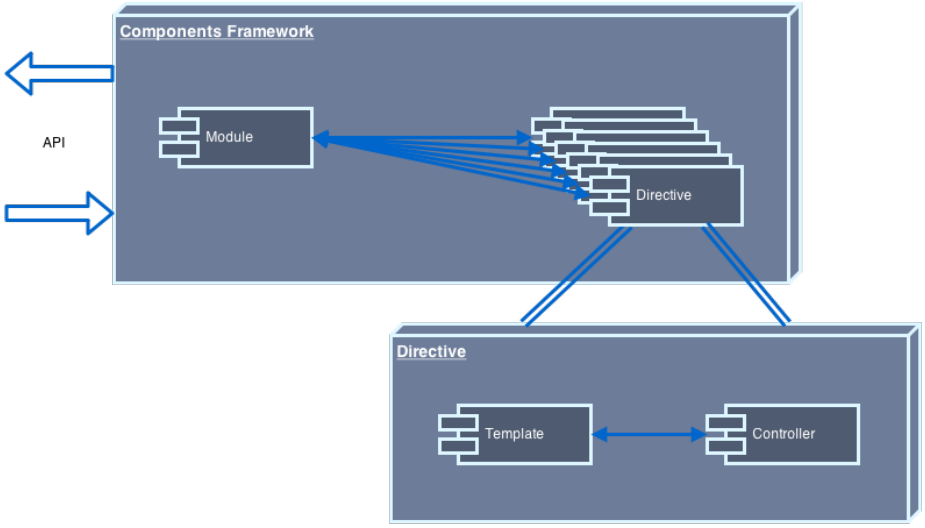


Figure 3.1: AngularJS Component Architecture

From the *Automatically Quality Tested* requirement, which is converted into a process, who will result in a framework of components, that can be distributed and used as needed. The processes of verifying the quality and testing are automatic and will generate a bundle of components. The framework of components will be as shown in figure 3.2, where all the component are ready to be imported and used.

3.2.2 Defining a Module

Every AngularJS application needs a module as the application's container. This container organizes the JavaScript code and avoids declaring variables in global namespace. Inside the

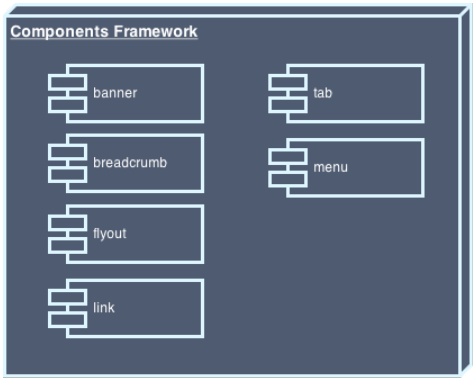


Figure 3.2: Components Framework

module it were built controllers, directives, services, filters, among others. As a good practice from the AngularJS team, each feature must have a module and for each directive and filter in specific, as they are expected to be reusable[28]. Creating multiple modules as recommended, may seem a little confusing on how to integrate and build an application with every module separated and how they interact with each other. As explained in the section 2.3.3, each component in AngularJS has an array to declare dependencies. There are a module that aggregate every desired module for an application to run. Also, when developing each feature separated by modules, also help on testing each feature separately and on isolating some cases where is desired to integrate multiple features and multiple modules. With these insights and with the requirements to create reusable and modular components, each directive was created with separated modules.

A module can be defined in two different ways:

- declaring it as sort of "function" as can be (see figure listing 3.1);
- assigning a module to a variable and construct the module appending content to it (see figure listing 3.2).

Listing 3.1: Angular Module

```
1 angular.module('NameOfModule', [])
2 .controller('calculator', function() {
3   //Here is where the application is built.
4   this.result = this.number + this.number;
5 });
```

Listing 3.2: Angular Module with Assign

```
1 var appModule = angular.module('NameOfModule', []);
2
3 appModule.controller('calculator', []){
```

```
4   //Here is where the application is built.
5   this.result = this.number + this.number;
6 };
```

The module declaration has two arguments: the first is the name of the component; the second is an optional list of dependencies.

3.2.3 Defining a Directive

As already mentioned in subsection 2.3.3, a directive in AngularJS is similar to a Web Component, which allows to create new DOM elements that will be interpreted by AngularJS HTML compiler, and assign behavior or transform the element. The process of creating and know how to create a Directive in AngularJS will be succinctly described in the next sections.

For this example on how to create a directive, the focus is to create a directive that can be reused and distributed. To create a directive that needs to be reusable and distributed there is a need to include the directive on a module. In listing 3.3 on line number 1, was created the module who will contain this directive. The module presented allow to reuse and import the directive on other AngularJS applications. On line number 2, it is defined the name of the directive, following the camelCase notation, and it is defined the function with a *return* that has[23]:

- **restrict** - Regarding the type of directive that is being defined, defining the declaration type of the directive. It can be: E - Element, A - Attribute, C - Class and M - Comment;
- **replace** - With a boolean value, defines if the element created will be replaced by the template elements or if it stays encapsulated;
- **link** - Every time there is a need to handle DOM inside a directive, it is needed to include a link with a function that will perform the desired operations;
- **template/templateUrl** - The template will define the layout of the directive, that can be defined inline with a **template** or from an external resource with **templateUrl**;
- **scope** - The directive can request a scope from three different types: **inherited**, **isolated** and **shared**;
- **transclude** - With transclude, it is possible the integration of other content inside the directive;
- **controller** - The controller will allow to create an API to share with other directives;
- **compile** - Compile allows to manipulate DOM in the compilation phase.

Listing 3.3: Directive myName

```
1 angular.module('myName', [])
```

```
2       .directive('myName', function(){
3         return{
4           restrict: 'E',
5           replace: true,
6           transclude: 'true',
7           link: function (scope, elements, a){
8             scope.completeName = a.first + ' ' + a.last;
9           },
10          template: "<h1>{{completeName}}</h1><div ng-transclude></div>"
11        };
12      });
```

Listing 3.4: Using Directive

```
1  <body ng-app="APP">
2    <div>
3        <my-name first="Sergio" last="Cunha"> </my-name>
4    </div>
5    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.2/
         angular.min.js"> </script>
6    <script>
7      var APP = angular.module('APP',['myName']);
8    </script>
9    <script src="myName.js"> </script>
10 </body>
```

After the definition and inclusion of the directive in the AngularJS application, it can be used with the created and registered element. Listing 3.4 shows an example with an element "my-name" which receives a "first" and "last" attributes that will define the first name and the last name. The my-name directive is also ready to distribute and reuse.

3.2.4 Components built with directives

After an overview on how to create a directive and how it works, the developed directives are presented. As a first requirement, the component needed to develop is a banner, which depends on other components such as tab, link and menu. After the banner being completely built, one more component will be built, a flyout. In the following subsections will be explain in more detail what has been done. To distinguish the created components from the existing, and avoid future name collisions in HTML native tags, the "ot" initials, standing for "Orange Touch" the used scheme for design inside Nokia, were added at the beginning of each component.

Banner

In any page, one of the main areas is the banner, where usually information of different sorts is accessible, for instance, the title and menus to access any area of the application, such as login, preferences, help, content, among others. The banner component is not a single component. A banner is a combination of various components like a single banner without actions, menus and links. All this components will enrich each other and, by working together, it results in a component combined by the banner, menus and links.

In the banner isolated component, was build a template to present the banner and included variables on scope to do binding of the data between the template and scope when it is rendered.

An application depends on menus and sub menus to show the available contents in an organized view. The <ot-menu> component gets an array of objects from the banner component with a title, URL, active flag and a children array with the submenu objects, which have the same properties as the parent objects. The menu will parse the information and build the component according to the children it has, to build the proper menu. This directive contains a *transclude* that allows to insert dynamic portions of code in the middle of the template.

Some directives can be dynamically enriched with other elements, elements that, for AngularJS uses transclude, for example, sometimes can be a list, other times are single, but separated items. When transclude is used, that content is appended to the scope of the directive.

As an example, a string will be added between the defined tags on the "my-name" directive that will be transcluded into the directive and render as if the content was the original from the directive, as seen on figure listing 3.5. The transclusion is possible, because the *transclude* flag was defined as true on listing 3.3.

Listing 3.5: Using Transclude

```
1 <div>
2     <my-name first="Pero" last="Dunvjak">
3         This content will be transcluded into the directive!
4     </my-name>
5 </div>
```

Figure 3.3: Banner Condensed

The link component was created with the requirement of having text with a link to do an action. The requirement properties to this component are a text string, a reference corresponding to a link or any other action and can have an icon parsed as string, which is the class from CSS. The <ot-link> component will parse the information received as attributes,

the text and reference, and will present the content after transmitting the attributes contents to the included template. The <ot-link> component was created when all the links on the page had a specific format. It can have an icon, and in some cases, the text inside a link needs to be shortened. In order to validate if there is an icon, the validation is done on the template with an inline validation using the "ng-if" directive, which will validate if an assertion is true or false. If the assertion is true, then it will be included in a class with the type of icon specified in the attribute icon. In any link component, the text length should not be bigger than 20 characters. If it is bigger, the title needs to be shortened. The text will only have 20 characters and include "..." at the end, with the complete name parsed to title element to show a tooltip with the full name.

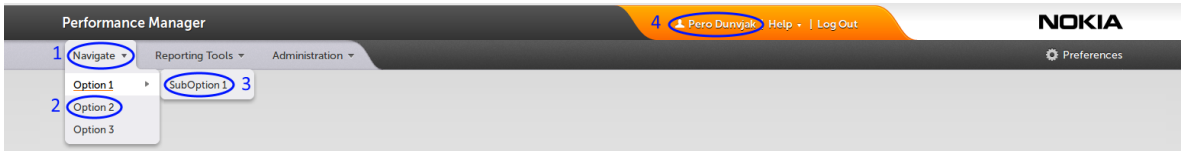


Figure 3.4: Banner Expanded

On the figure 3.4 the components described before as part of banner, are highlighted:

1. Menu component, that will hold options;
2. Here is an option component, responsible to enrich the menu;
3. A submenu is present here with more options inside;
4. One example of using the <ot-link> component with an icon.

All the components mentioned above are placed inside the banner component to create the behavior pretended.

Flyout

In the best interest of the user, it became necessary to implement a component that the user could manage all the opened areas. This can be seen as a quick access menu that permits the user to revisit its previous choices and report generations. Also, the flyout is expected to be hidden with a grip to open and close, as seen in figure 3.5, where a single grip is visible. To construct the flyout component, was needed to get an array of objects that correspond to all the user opened areas. That array should have the area ID and also the active ID that informs us which area is opened. If a new area is opened then a new tab will be added to the flyout, through a new array of objects received. To change or close an opened area as the click action is triggered in case of close button will be asked to close or if its a click on an open are to change, it will change the current area to the clicked. In the end, after closing the

area or switching to the desired area, an action is triggered to send an object to the flyout with all the opened and the active areas in case of something goes wrong and the tab could not be closed or switched. An example of the flyout with some opened areas as can be seen in figure 3.6.

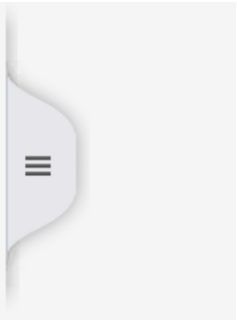


Figure 3.5: Flyout Closed

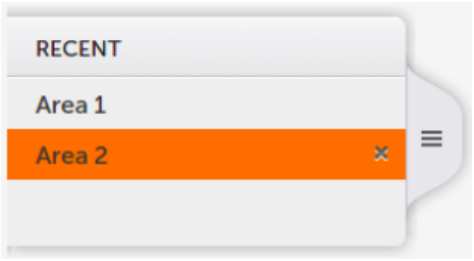


Figure 3.6: Flyout Open

3.2.5 Tools used in development

As a part of the development, it is pretended to create a build process that will simplify the development, including quality and testing mechanisms and also distribution process. In order to achieve this goal, a set of tools was used to manage packaging, building, verifying code quality and testing the developed components.

Bower

Web sites have various dependencies and to manage each one of them is a hard work to do. From frameworks, to libraries and utilities, all dependencies can be managed through Bower. Bower is a dependency manager for web, focused on frontend.

In a not optimized development and build environment, the developer must go to each site and download all the dependencies needed. The developer must also be aware on new versions to update the system and the procedure is usually the same: download and install. With Bower the only requirement is to do a *bower search "package"* followed by *bower install "package" –save*. Bower keeps the project dependencies in a "bower.json" file. When it is time

to update, the only thing needed to update is to do *bower update*. If some package breaks, then Bower allows the developer to install a specific version of the *bower install "package#version"* dependency[29].

Docular

Docular is a documentation generator built in AngularJS that also uses Grunt and Node.js[30]. Since it parses all the scripts or non-script, the documentation can be inserted within code or in separated files, by searching for comments referring to "@doc" inside a comment block for the declaration on script or only for "@doc" in documentation files. Inside the documentation, there are a set of attributes required:

- "@doc" where will be defined what will be documented;
- "@name" defines the name of the attribute;
- "@description" where is described the code being documented;
- "@restrict" in case of documenting a directive in AngularJS to define if it is an E, A or C;
- "@param" to describe the parameters;
- "@example" where is built an example usage which will be parsed by docular and present the example within an iframe.

EditorConfig

The EditorConfig is a tool that help to maintain consistency on coding style, from editor to editor and from developer to developer. It provides a simple way to ensure freedom for the developers when choosing the editor while ensuring that the coding style is the same. It requires a plugin for the IDE or Editor to read the patterns defined in the *.editorconfig* file. The EditorConfig file set all the required code style, from specifying tab as space, indentation, among other for the project[31].

Grunt

When developing an application and, in this specific case, components for a web page, there are some task that are repeated over the time such as: compilation, minification, testing, among others. These tasks can be automated to make the developer's job easier, allowing to focus on other tasks[32]. Grunt is a task runner in JavaScript, that help on some recurrent tasks. An example of a grunt configuration file is present on listing 3.6. Where can be seen an small example of a grunt configuration file, that do a quality analysis on code with JSHint, and a watcher to keep analyzing and warn for lack of quality. The JSHint will be described later in this subsection.

Listing 3.6: Example Grunt Configuration

```
// Wrapper function that encapsulates Grunt configurations
module.exports = function(grunt) {

  // Load the plugins needed
  grunt.loadNpmTasks('grunt-contrib-jshint');
  grunt.loadNpmTasks('grunt-contrib-watch');

  // Initialization of configuration objects
  grunt.initConfig({
    jshint: {
      files: ['Gruntfile.js', 'src/**/*.js', 'test/**/*.js'],
      options: {
        globals: {
          jQuery: true
        }
      }
    },
    watch: {
      files: ['<%= jshint.files %>'],
      tasks: ['jshint']
    }
  });

  // Setup the tasks
  grunt.registerTask('default', ['jshint']);

};
```

JSHint

When writing code in any project, the quality of the code is a constant concern. What today is a small piece of code and is not difficult to understand, may become complex, very hard to understand and iterate. JSHint is a community-driven tool to check for errors and potential problems in code. The goal of JSHint is to allow JavaScript developers to write better code without worrying. As previously mentioned, a project in the beginning is small, but tends to become huge and have more problems based on complexity of functions, or even on mistakes, such as syntax error, that could lead to hours of debugging, leaks, bugs, slow processing, among others. JSHint can be configured to match the requirements patterns of quality that the team wants the project to match and configure the integrated development environment (IDE) to use the configurations of JSHint and raise a flag each

time a fault in quality is detected[33]. As JavaScript is a dynamic typed language, where there is no compilation which may lead into syntax mismatch, with JSHint embedded with IDE the developer is constantly warned when there are errors and problems on the current implementation and enforce the developer to correct the problem prior to commit the code, avoiding the problems mentioned above.

Jasmine

Jasmine is used to support Unit Testing and the use of the TDD methodology during application development. Jasmine is a behavior-driven development framework to test JavaScript code. It does not have any dependency on other framework or DOM. Writing tests on jasmine is easy, for example a collection of test. Inside Jasmine, the tests are separated within **suites** and **specs**. A suite is where the test domain is defined. A suite begins with a call to *describe* jasmine function followed by a name and a function. Inside of suite are defined the specs. A spec is the test pretended to run, with an assertion at the end. It is expected that after some actions the test will assert true or false. A spec is like a suite, but instead of being called with *describe* is with *it* and takes a string and a function. An example can be seen on figure listing 3.7. In jasmine, everything is a function, it is pure JavaScript. Every test in Jasmine can have portions of code to implement or generate data for a specific test.

Listing 3.7: Example Jasmine Test

```
1  describe('A suite of tests', function() {
2    var a = true;
3
4    it('Assert that var a is true', function(){
5      expect(a).toBe(true);
6    });
7
8    it('Assert that var b + c is not calculated incorrectly', function()
       {
9      var b = 1;
10     var c = 2;
11
12     expect(b+c).not.toBe(4);
13   });
14 });
```

Karma

When running several tests in different browsers, one thing that could help is a tool that runs the desired tests in a collection of browsers. Karma is a tool that invokes a web server

and runs the tests over the code on each browser. The process of defining a browser can be done manually or automatically. If done manually, the tester should open the browser that the code is being tested within a server where Karma is running, usually *http://localhost:9876/*. This tool can run the set of test in two formats ways, a manual an automatic. On automatic, the tester sets the array of browsers where are going to be tested, an instance of each browser is initiated and the bundle of tests run for each one of them, with summary of the results in the end. On manual the user star the karma server and then open the link of the karma server on the browsers where want the tests to run, at the end of the tests the result is the same as in automatic mode. Karma also helps on Test Driven Development (TDD), since it can keep watching for changes in any file and then runs the tests again[34].

3.3 Testing a component

One major part of development is testing the component to see if everything was correctly developed and has the expected behavior. Following one of the requirements of the framework of components, the TDD methodology should be used.

3.3.1 Test Driven Development

The Test Driven Development (TDD) is a methodology on how to write code with quality that gives feedback at each interaction. By following TDD, at the first interaction, it is needed to think in the feature. It is expected that this feature have a predefined behavior and for that it is needed that it is separated in components. Following TDD, it clarifies the acceptance criteria over the next component, helps isolating a component that is completely independent, allows to detect errors, allows to make the code clean by refactoring and, doing TDD, all the unit tests that need to be done in order to test the feature, are already made during development. With TDD, the feedback on the quality of the implementation and design is constant[35][36].

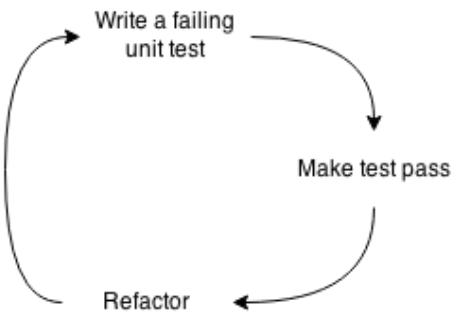


Figure 3.7: TDD cycle

To do TDD there are three laws:[37]

1. Do not write code unless there is a unit test failing;
2. Do not write more than a unit test that is sufficient to fail. Not compiling is also failing;
3. Do not write more production code once the test passes the current failing test.

3.3.2 Unit Testing

A unit test is a short program that will assert a determined behavior in short pieces of code, written by developers. The result of every test is a boolean and assert if the test passes or fails[38][39].

3.4 Problems on implementation

During implementation has occurred some problems, some related with lack of knowledge on how AngularJS works, but the main problem that was faced is related to transclude.

3.4.1 Transclude

During the first implementation, transclusion was used to include other directives in templates without specifying them in the template. This quickly become an issue when was pretended to include different directives in more than one part of the template. By the time of writing, AngularJS only allowed one transclusion area in a template. To solve this issue two solutions were pondered. The first one was using a multi-transclusion plugin[1]. The second solution was to include the directives and other code directly in the template rather than defining it later in the implementation. Since the components that needed this feature could not exist without the template part that had to be transcluded, it was decided to include it directly in the template avoiding unnecessary imports and code bloat.

3.5 Chapter Summary

In this chapter were presented the requirements for the construction of components in AngularJS. Also, a architecture proposal was implemented with all the insights from the requirements and AngularJS constraints. The actual architectural proposal was defined with all the concerns described, but also were keep in mind that the actual architecture is only a transitional architecture, when there is not implemented the API to access data as expected, which simplifies and improve the implementation quality. After a brief on how to build components who are modular and reusable, all the components built were presented with an overview of the complete component. This was the main goal of this work, to implement a method to develop components, and then develop components with it. This implementation

[1] https://github.com/zachsnow/ng-multi-transclude

also includes the testing methodology followed, which is part of the requirements to test each component.

On the end of the chapter a section with problem and solution is presented, with insights that help on future development.

Chapter 4

A case study of Web Components integration based in AngularJS

A challenge that was always present in the reformulation of already implemented applications is that is not always possible to build a new application, instead it is only possible to integrate with the existing one. The development of components, since the beginning that it was driven to integrate with an existent product based on legacy code. The integration of the new development with legacy code, adds a layer of complexity, it is needed to update the application from the old solution into a modern and up to date solution. In this case, and with the present study, the chosen solution are the components developed in AngularJS. To be able to integrate the components with the present solution is required, at first to know the application that will be migrated, how it was build and in what technologies, second, is needed to embrace the development team methodologies and, at last, do the integration.

The integration was done in a product from Nokia Solutions and Networks, the Nokia Performance Manager, that is an application for network monitoring and data visualization.

The integration process is not like building an application from scratch. The application needs to be stable and a deep restructuring can not be done, due to the huge costs on such task that will not allow it to be done in a short term period. Also, there are features being developed which can not stop. Doing the parallel task of rebuilding the application it would be a tentative of achieving the present state of the application and it would not be possible because of the continuous development of new features. Having these insights, the integration can not be done as a project running in parallel. The solution is to identify the main areas that need to be restructured, develop the components to those areas and integrate them with the application. After each restructuring, it is needed to identify the next area and do the process once again. The restructuring is expected to be easy at each interaction. With more components developed, there are areas that will only need to use the already crafted components.

The main focus of this chapter is the integration of the developed components with the

present solution. The present solution will be presented as all the methodologies used in the development, problems and challenges from this approach.

4.1 Nokia Performance Manager

In a world using mobile communications, the requirements to have better quality and less faults on network are a demand. With more data generated by mobile broadband and Long-Term Evolution (LTE) services defining new quality and performance indicators. The objective of taking a 360° overview over network status, the network operators have to manage thousands of indicators that allow to manage and optimize the network operation. Nokia Performance Manager is a tool, developed by Nokia Solutions and Networks, that allows to manage the mountain of data generated every day. Mapping the information on various and helpful data visualization methods, like bar, scatter, pie, among other charts which allow to identify degradation and prioritizing network quality. Performance Manager can handle up to 70 billion network counters daily[40].

Flex is a framework that allows the development of multiplatform applications, having as a requirement the dependency of Flash. It appeared to fill the gap that Javascript brought to web applications. It is comprised of different components: ActionScript and MXML. The ActionScript is intended to produce the application's core even if it requires a server to communicate with. The MXML is the layout, the layer that will present the actions and change the application.

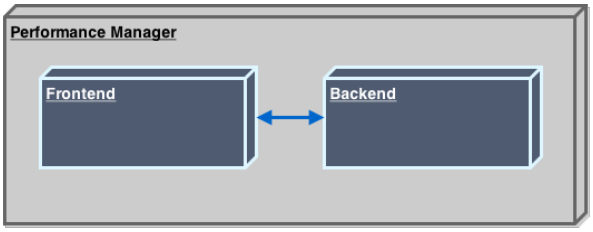


Figure 4.1: Performance Manager Architecture

Currently, Performance Manager main goal is to evolve from Flex due to its problems in terms of performance and mixing of concepts. Some of these problems are: single thread, deprecated by Adobe, event driven, data visualization components are not good and security among others.

The current implementation as already said, has some mixture of concepts, the frontend and backend logic are not isolated, which leads into a dependency from backend solution with the frontend solution, this mixture is due to business development on frontend. The architecture of Performance Manager is displayed on figure 4.1.

One of the answers from the Adobe team about using Flex, when they were questioned

about using Flex or HTML (version 5): "In the long-term, we believe HTML5 will be the best technology for enterprise application development"[41].

4.1.1 Benefits from this work

The implementation of components in the present application will result in a restructuring of the application on a new and innovative solution which leads to:

- **Modular application** - An application divided and completely separated into modules;
- **Less development time** - With components to do specific tasks, the time needed to create new features will be reduced;
- **Less bugs** - Restructuring the present application from a legacy solution without tests to a TDD approach;
- **Clean and restructure present solution** - Doing the migration of the application into components will facilitate the restructuring of the present solution.

With a modular application it is easy sharing and reusing components in new modules. A modular approach helps on testing each module separately. After the initial phase of developing components, where all the components need to be created, it will be easier to create new modules and develop frontend areas. When creating new areas and there is a component that does not have the desired behavior, then the component can be extended to provide the desired behavior. In case that the requirement is specific, then the extension should be created in for of new component that extend the behavior of the first. As the restructuring proceeds, the implemented tests also increase, leading to higher quality software, and so expecting to have less bugs. Also, the restructuring gives the ability to rethink the present implementation from frontend to backend, cleaning and improving the solution.

4.2 Mode of Operation

4.2.1 Methodologies

On any job, the best way to know if the job has been done correctly is by having feedback from coworkers, employers, clients, friends and others. Any kind of feedback that can be gathered is good to evolve and evaluate if the job is correctly done. For a programmer, the feedback is also important to determine the code quality and consequently the product quality. When it is time to deploy the work it is also time to check for feedback, detect problems, determine solutions and evolve. When programming, it is crucial to have feedback systems in any cycle[35], avoiding all the possible problems and misunderstandings that can come from not testing our work.

This solution has the requirement to fulfill a quality pattern and, for that, is required that was followed development and testing methodologies were followed.

Agile

The agile development paradigm can be applied in various methodologies, such as: extreme programming, kanban, scrum, and others, but on this specific case it is used scrum, since it was already used by the team. Agile development use time-iterations, adaptive planning and development. Those practices allow teams to respond and adapt quickly with changes on priorities. With an agile development methodology, it is expected that the development focus on agility, simplicity, communication and focus on programming instead of documentation[42][43].

With Scrum the teams have flexibility to select the work and take care of problems, by following this guidelines:

- The team and the project owner create a list of prioritized task or features to be done on a sprint. This actions are mapped on the product backlog;
- At the end of each sprint, the tasks on the backlog are evaluated to know how much weight they have. Also, a demo is done to show to the product owner what has been done;
- The team does a fast daily stand-up meeting that should not surpass ten to fifteen minutes, where everyone updates the status on what they are working and if they have blockers. This is a stand-up meeting;
- A person is designated as scrum master and has the task to facilitate the work, to remove or get some-one to remove blockers from stand-up meeting.

4.2.2 Continuous Integration

Ensuring that an application has the expected behavior and does not have bugs is a constant concern. With unit and integration tests, these techniques are useless if they are not integrated with the other development team. An application is not developed only by one or two programmers and, even in this case, there is a need to have a source code repository to have version control over the code, to share the project with colleagues, to do the final build of the project to deploy. By bringing all these concepts together, the concept of Continuous Integration (CI) appears. A project should have an environment where the application under development will be tested with all the commits from everyone.

A CI environment is designed to facilitate and to always have a feedback of the project integration of features, either if they are going well or if they are causing some problems with the existing ones. For a CI system to work properly, a server must be running to do a couple of tasks. This is not a requirement, but it is easy to have those tasks automated.

Before committing the changes to the version control, it is expected that the local repository will be updated to match all the latest committed changes. Then, after building the application on a local machine, it is time to do the tests and check if anything has failed, if

some fail in this process, in building or testing, then a failure will show. If something fails, the problem should be analyzed and fixed. The process should restart, and, if everything was done with success, then it is time to commit the changes to source code repository.

The CI server will check if there were commits made on source code repository and then it will build the application. If some errors have occurred, it will trigger a warning and stop the process. If everything was correct while building, then it will run a set of tests which can vary from unit, integration to regression. Check if there are some errors, it will warn and can stop the process if the accepted amount of failed tests were achieved.

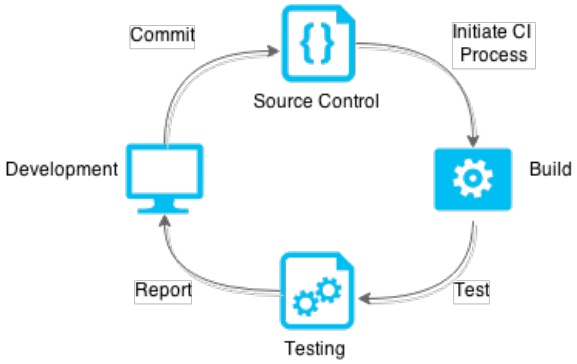


Figure 4.2: CI cycle

Integration Test

After developing a component with TDD, it is time to push them to repository, test component with all other components and check for faults that may occur. An application is ready if all the components are working properly, either way the application is failing. The integration test is a drill that runs all the single unit test together, asserting a specific or various behaviors[38].

To be able to do integration tests in this projects the chosen solution was Jasmine with Karma.

End-to-End

End-to-End tests look to the test environment as a black box and the tests run on the visible environment. Usually this type of test is used to do a complete test, from one point of application to another, testing if every component work as expected in the final solution[44].

Robot

When it is time to test if everything is working as expected by doing acceptance testing or acceptance test-driven development (ATDD), the tool that was already used is Robot. Robot

is a framework for automated testing, with a keyword-driven testing approach. One main advantage of this framework is that it is generic and can be extended through test libraries and allow the user to create new higher-level keywords from existing ones using the syntax used in the creation of test cases[45].

4.2.3 Sonar

On behalf of quality code, the inclusion of a tool that analyses and gives feedback on code quality will be used SonarQube[1]. SonarQube will be used because it is currently used by the team and the code can be aggregated in one single tool. With a sonar, it is possible to manage code quality in seven different axes: Architecture and Design, Coding Rules, Comments, Complexity, Duplications, Potential Bugs and Unit Tests. Whether it is important to take actions on code quality, at a file, module, project being developed, the dashboards available aggregate all the pertinent information to have an overview of the code quality. The platform is composed of three components: Database, Analyzer and Web Server.

With SonarQube running analysis on code, feedback is provided on how the code is structured, doing analysis on every line of code, checking for issues on code and quantifying each of the issues by severity and ordered from worst to inoffensive: blocker, critical, major, minor and info. Also, SonarQube will analyze code covered by unit tests, documentation, duplication, complexity, design, among others. Any issue identified will have a weight which will add a penalty to correct them, the issues are quantified within a technical debt in working days and in cost of operation.

4.3 Integration

After creating the components a new challenge comes: how is it possible to migrate the present application?

At first, it may seem impossible to include components built with HTML in a Flex application, but after a brainstorm, two solutions were identified:

- 1st solution, a complete remake of the application, which is impossible, due to its high complexity and a need to still develop content in the present application and maintain the already installed versions;
- 2nd solution, insert the components as they are being developed, doing some adaptations on the current application to support them.

In order to build the application with the second possibility the architecture of the application has changed to contain the "Interface for communication" 4.3.1 and the components from "Components Framework". As a result, the architecture will be as seen on figure 4.3, an

[1] http://www.sonarqube.org/

integration of the old frontend with The interface created to communicate with framework of components as a new frontend layer, using by now the flex to communicate with the backend. Over this process of integration all the methodologies described earlier in this chapter were followed with the integration of the developed components in section 3.2 in CI and on Sonar, to control and test quality, also they were integrated to allow everyone to use the created components, and in future start the development of new components with a continuous integration process. Furthermore, in the integration process the agile methodologies were followed since I was integrated in a development team in order to integrate the components with the product.

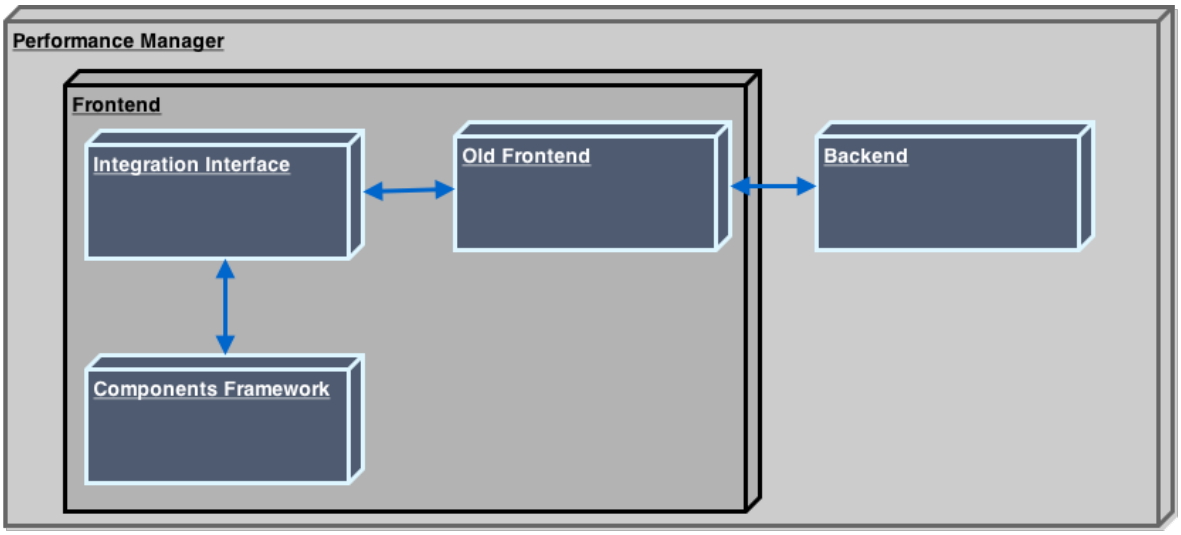


Figure 4.3: Application Architecture

4.3.1 Interface for communication

With no possible communication with the backend due to a lack of web service layer and since the old frontend will continue to exist for an undefined time, and as already said the present solution needs some maintenance, the solution was to create communications with flex. The communication with flex was created using "ExternalInterface.addCallback("javascriptFunction", flexFunction)". This interface is a function on flex that create a link between Flex and JavaScript, defines a function name to be called from JavaScript, "javascriptFunction" and will tell old frontend to run the specified function, "flexFunction". The result from this external interface can be a return value that will be used in JavaScript or actions in flex. To implement each of the components there was a need to create various interfaces to share the desired data with each component. For each of these interfaces it was also needed to parse and collected information inside of old frontend prior to share through interface. In this process there is also a need to create listeners inside the old frontend that will call JavaScript

functions if are triggered actions from old frontend.

4.3.2 Integration Problems

While process of integrating components was made, some problems were identified:

- lack of web service layer to get the information from the source and not through other implementations will simplify the process and wold not add another layer of complexity in the future when its time to shutdown the flex application;
- when implementing interfaces inside flex, the complexity to create a component increases a little, because of the need to create a structure and some times to do adaptations on the component to work properly with the current implementation.

After the integration of the components and with adjustments in flex to replace the old items, some problems with integration appears, as the test that uses menus our tabs, that now are in the flyout, need to be corrected. Also due to an adjustment on the flex container the content was not rendering as expected and was needed to make corrections on the flex application to show the content correctly.

4.3.3 Achievements

At the final stage of the integration it is possible to see that the integration and the first area of components were implemented and integrated resulting in an already released version of software, as shown on figure 4.4.

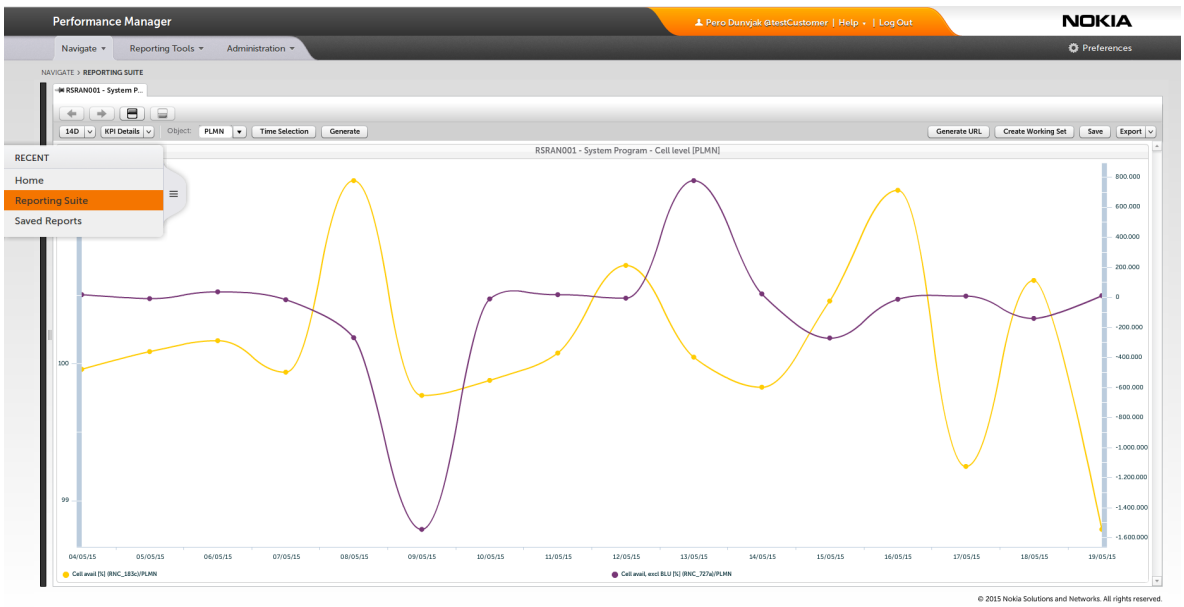


Figure 4.4: Product with Integration

With the present process of developing and integration was possible to identify problems with the old solution and were already pointed to review in future improvements.

4.4 Chapter Summary

Throughout this chapter were described the environment where the developed components will be integrated, the mode of operations of the team and the need to include the components on the present methodologies. In the end the integration process is described with all the adaptations on the architecture of the old solution to the present solution. Even with all the constraints to integrate the components with the old solution, it was visible that it was surpassed concluding in the achievements.

Chapter 5

Conclusions and Future Work

5.1 Conclusion

In this study of a framework to develop components were assessed the current possibilities that allow the creation of web components or a lock alike components based on the working draft standards of Web Components. It was possible to acknowledge the improvements coming with the introduction of Web Components in HTML, but also, the actual limitations with an implementation based on the current compatibility and knowing that they are not standards yet and can change any time. However there are solutions presented in order to solve this problem, and within the solutions AngularJS was the chosen solution due to its ability to create components that have the same behavior as if they were web components. Also, an acceptance from the programmers community was important on the final decision, because it is expected that this decision is for a long term implementation.

The solution adopted to develop components in AngularJS was a close approach to web components, focused on: reusability, modularity and encapsulation. With the design of the framework of components it was possible to "standardize" the development of components and also simplify a future migration into native components. Since the methodology of development is similar and simple the only need is to adapt to the coding style, because there will be a template, custom element, behavior as in controller and imports. The main concepts are already implemented.

To implement these components in AngularJS at first it was very hard to keep focused on producing isolated components. But with training it start to increasingly became easy to implement isolated components. Also, with the need to create components there is an overhead on identifying the areas to migrate and in isolating the existing areas into components, leading into an increasing time to develop each component. With this approach and with AngularJS there were some cases who led to acknowledge that this implementation of components was complex. This complexity could maybe been avoided with an implementation with native or Polymer, but its only assumptions.

On the integration of the framework of components with Nokia Performance Manager, the solution to do the integration was only temporary. Since there is not a web services layer implemented. This implementation have increased the complexity on development, but even with this constraints, it was possible to actually integrate the components. Also noted that, the developed components was already implemented and delivered in a product release.

5.2 Future Work

The current framework of components is functional and implemented on Nokia Performance Manager as already said, but it lacks on the other areas that need to be migrated into components. Since the components were only implemented into two different areas, it is imperative to continue the development of components in order to do a full migration of the product. Also, this migration will need to be followed by the development of web services to shutdown the old frontend.

The current implementation depends on the developer to create the main structure of the page, that is a handicap when there is time to reformulate or upgrade some application. In order to be easier and to jump this problem, have emerged the need to create a scaffold of HTML and CSS that is common for each application and that also will enable the use of any framework. With a page that is ready to develop and can use all the developed components, will simplify even more the development of new applications.

As a final remark on future work, and since the application manages large amount of data, there is a need to create components that allow the visualization of data in tables, charts among others, in an efficient way, in performance and visualization.

Bibliography

[1] HTML5 - Template. http://www.w3.org/TR/html5/scripting-1.html#the-template-element.

[2] W3C. Custom elements. http://w3c.github.io/webcomponents/spec/custom/, 2015. [Online; accessed 27-March-2015].

[3] W3C. Shadow dom. http://w3c.github.io/webcomponents/spec/shadow/, 2015. [Online; accessed 27-March-2015].

[4] Oxford University Press. template - definition of template in english from the oxford dictionary. http://www.oxforddictionaries.com/definition/english/template, 2015. [Online; accessed 27-March-2015].

[5] 4.11.3 Template — HTML5. http://www.w3.org/TR/html5/scripting-1.html#the-template-element.

[6] W3C. Html imports. http://w3c.github.io/webcomponents/spec/imports/, 2015. [Online; accessed 27-March-2015].

[7] WAI-ARIA Overview. http://www.w3.org/WAI/intro/aria.php.

[8] Web Accessibility initiative. Wai-aria overview. http://www.w3.org/WAI/intro/aria.php, 2014. [Online; accessed 26-November-2014].

[9] Microsoft. modern.ie. https://status.modern.ie/, 2015. [Online; accessed 24-February-2015].

[10] Alexis Deveria. Can i use. http://caniuse.com/, 2015. [Online; accessed 24-February-2015].

[11] WebComponents. Polyfills. http://webcomponents.org/polyfills/, 2014. [Online; accessed 12-December-2014].

[12] Google. Polymer/web-component-tester . github. https://github.com/Polymer/web-component-tester, 2015. [Online; accessed 30-March-2015].

[13] MutationObserver - Web API Interfaces — MDN. https://developer.mozilla.org/en/docs/Web/API/MutationObserver.

[14] Mutation observers (Windows). https://msdn.microsoft.com/en-us/library/dn265034(v=vs.85).aspx.

[15] W3C DOM4 Mutation Observers. http://www.w3.org/TR/dom/#mutation-observers.

[16] Polymer Authors. Welcome - polymer. https://www.polymer-project.org/0.5/, 2015. [Online; accessed 23-March-2015].

[17] Polymer Authors. Understanding polymer. https://www.polymer-project.org/0.5/docs/start/everything.html, 2015. [Online; accessed 20-March-2015].

[18] Element collections - Polymer. https://www.polymer-project.org/0.5/docs/elements/.

[19] core-icon - Polymer core elements. https://www.polymer-project.org/0.5/docs/elements/core-icon.html.

[20] Facebook Inc. React (virtual) dom terminology — react. https://facebook.github.io/react/docs/glossary.html, 2015. [Online; accessed 25-March-2015].

[21] Facebook Inc. Advanced performance — react. https://facebook.github.io/react/docs/advanced-performance.html, 2015. [Online; accessed 25-March-2015].

[22] Flux — Application Architecture for Building User Interfaces. https://facebook.github.io/flux/docs/overview.html#content.

[23] Google. Angularjs: Developer guide: Introduction. https://docs.angularjs.org/guide/introduction, 2015. [Online; accessed 30-March-2015].

[24] AngularJS: Developer Guide: Scopes. https://docs.angularjs.org/guide/scope.

[25] X-Tag - Web Components Custom Element Polylib. http://www.x-tags.org/.

[26] Mozilla. Mozilla brick. http://brick.mozilla.io/v2.0/blog, 2015. [Online; accessed 18-March-2015].

[27] WebPagetest - About. http://www.webpagetest.org/about.

[28] Google. AngularJS: Developer Guide: Modules. https://docs.angularjs.org/guide/module.

[29] Bower. http://bower.io/.

[30] Docular. http://grunt-docular.com/.

[31] EditorConfig. http://editorconfig.org/.

[32] Grunt. http://gruntjs.com/.

[33] JSHint. http://jshint.com/.

[34] Karma - How It Works. http://karma-runner.github.io/0.12/intro/how-it-works.html.

[35] Steve Freeman and Nat Pryce. *Growing Object-Oriented Software Guided by Tests*. Addison-Wesley, 2009.

[36] Robert C Martin. *Clean Code: A Handbook of Agile Software Craftsmanship*, volume 1. Prentice Hall, 2008.

[37] Robert C. Martin. Professionalism and test-driven development. *IEEE Software*, 24(3):32–36, 2007.

[38] Roy Osherove. The Art of Unit Testing: with Examples in .NET. 2009.

[39] Guide to Agile Practices. http://guide.agilealliance.org/guide/unittest.html.

[40] Performance & Capacity Management — Nokia Networks. http://networks.nokia.com/portfolio/products/operations-support-systems/performance-manager.

[41] Your Questions About Flex. http://blogs.adobe.com/flex/2011/11/your-questions-about-flex.html, 2011.

[42] Kenneth S. Rubin. *Essential Scrum: A Practical Guide to the Most Popular Agile Process*. 2012.

[43] Manifesto for Agile Software Development. http://www.agilemanifesto.org/.

[44] Lisa Crispin, Janet Gregory, Agile Practice Lead, Object Mentor, and Ideaca Knowledge Services. *Agile Testing: A Practical Guide for Testers and Agile Teams*. 2009.

[45] Robot Framework. http://robotframework.org/.