

Bachelor's thesis

Degree Programme in Information Technology

2019

Bui Trung Tan

WEB COMPONENTS

– Concept and Implementation



BACHELOR'S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Degree programme in Information Technology

2019 | 46

Bui Trung Tan

WEB COMPONENT

- Concept and Implementation

This thesis was conducted as research on Web Component by analyzing its specification and usage. The purpose of this investigation was to introduce and implement web components in a web application.

This thesis discusses Web Component concepts and their related specifications, such as Shadow DOM, HTML Template and HTML Custom Element. This study also examines lit-element library and shows how to utilize this library for web development.

The implementation of web components was demonstrated in a Phonebook App prototype. This project creates new web components and uses them together with third-party web components to build a complete web application. The result of this project is a Progressive Web App running on client-side with basic features for managing personal contacts. To put this prototype into production, this thesis explains a deployment process using Heroku service, a popular platform for publishing web application without complicated configurations.

KEYWORDS:

Web Component, DOM, Shadow DOM, HTML Template, Custom Element, LitElement, Heroku

CONTENTS

LIST OF ABBREVIATIONS (OR) SYMBOLS	5
1 INTRODUCTION	6
2 WEB TECHNOLOGIES	7
2.1 HTML and CSS	7
2.2 Javascript	7
2.3 Browser	8
3 WEB COMPONENT CONCEPTS	9
3.1 DOM	9
3.2 Shadow DOM	11
3.3 HTML Template	15
3.4 HTML Custom Element	18
3.5 Web Component	21
4 LIT-ELEMENT LIBRARY	24
4.1 Polymer CLI	24
4.2 LitElement component	25
4.3 Templates	26
4.4 Styles	27
4.5 Attributes and Properties	29
5 IMPLEMENTATION	30
5.1 Phonebook App project	30
5.2 Use case diagrams	30
5.3 Development	35
5.4 Deployment	39
6 CONCLUSION	43
REFERENCES	44

FIGURES

Figure 1. Browser popularity in March 2019 [11].	8
Figure 2. The role of DOM between original source code and web page.	9
Figure 3. Original source code of a simple web page.	10
Figure 4. DOM structure of a simple web page.	10
Figure 5. The simple web page in user's device.	11
Figure 6. Shadow DOM inside regular DOM [17].	12
Figure 7. Attaching a Shadow DOM.	13
Figure 8. DOM structure of a web page with Shadow DOM.	14
Figure 9. Result of Shadow DOM.	14
Figure 10. Compatibility of Shadow DOM with different browsers.	15
Figure 11. Example of using HTML Template element.	16
Figure 12. Before and after processing HTML Template content.	16
Figure 13. Document fragment of a HTML Template element.	17
Figure 14. Compatibility of HTML template with different browsers.	18
Figure 15. Example of a custom element.	19
Figure 16. Compatibility of Custom Element with different browsers.	20
Figure 17. An example web application using web components.	21
Figure 18. Javascript declaration of the CardTicket web component.	22
Figure 19. Example web application using the CardTicket web component.	23
Figure 20. Loading Polyfills library.	23
Figure 21. Declaration of the CardTicket web component using LitElement.	26
Figure 22. Loop and Conditional syntax.	27
Figure 23. LitElement static style property definition.	28
Figure 24. LitElement external style definition.	29
Figure 25. Use-case diagram of Phonebook App.	31
Figure 26. Phonebook App project structure.	36
Figure 27. Phonebook App user interface in wide-screen devices.	38
Figure 28. Phonebook App user interface in small-screen devices.	38
Figure 29. Start the local server using Polymer CLI.	39
Figure 30. Run web component unit tests in a local computer.	40
Figure 31. Bundling the application for deployment.	40
Figure 32. Create a new Heroku app.	41

TABLES

Table 1. Use case "View contact list".	31
Table 2. Use case "Add a new contact".	32
Table 3. Use case "Remove a contact".	32
Table 4. Use case "View contact details".	33
Table 5. Use case "Update contact details".	33
Table 6. Use case "Bookmark a contact".	34

LIST OF ABBREVIATIONS (OR) SYMBOLS

API	Application Program Interface
CSS	Cascading Style Sheets
DOM	Document Object Model
ES	ECMA Script, a scripting-language specification standardized by ECMA International organization
ES6	ECMA Script version 6, or ECMAScript 2015
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
LESS	Leaner Style Sheets
NPM	Node Package Management
SASS	Syntactically awesome style sheets
URL	Uniform Resource Locator
WHATWG	Web Hypertext Application Technology Working Group, the organization which maintains and evolves HTML since 2004

1 INTRODUCTION

Web development has been growing rapidly in recent years. According to the Stackoverflow 2018 survey, JavaScript is the most popular programming language since 2012. [1] This programming language allows web applications to be functional like a native application in different platforms. For example, a Progressive Web App can run on mobile devices without a network connection, supporting features such as Push Notification, Geolocation and Camera Access.

With the evolution of Javascript, web development is moving from server to client. Traditionally, a website is developed by a server-side programming language such as PHP, ASP, or Java. The server returns HTML, CSS and Javascript content to the client device. This methodology has some drawbacks, such as server overloading or poor network connection. The new approach is to let the server produce and send only the necessary data to the client while Javascript running in the client-side processes the data and displays it to the users. [2]

This new trend requires front-end development – or client-side development – to be more mature. In software development, DRY (Don't Repeat Yourself) is the most fundamental principle. It states that “every piece of knowledge or logic must have a single, unambiguous representation within a system.” [3]. Web Component is a modern approach of this principle by moving duplicated parts of web applications into components.

Web Component is becoming a popular standard and being supported by many web browsers. By following this standard, developers can create web components which are reusable across web applications or organizations.

This thesis was carried out as a research project on the topic “Web Component – Concept and Implementation”. By examining Web Component specification, the thesis aims to clarify the principles of Web Component and their related technologies. Thus the Phonebook App prototype was developed which illustrates how to implement web components and use them in a web application.

2 WEB TECHNOLOGIES

This chapter explains technologies related to web development, especially front-end web development such as HTML, CSS, Javascript, and browsers. These are also understood as client-side technologies, since they affect a user's interaction directly.

2.1 HTML and CSS

HTML (Hypertext Markup Language) is the markup language to create structure and content for a web page. HTML encloses different parts with elements. An HTML element appears in the HTML source code as a *tag*, such as the `<body>` tag which represents the main content of the web page. Some HTML tags give meaning and style to elements. For example, a text inside a `` tag is usually rendered as a bold text and considered as important content. [4]

CSS (Cascading Style Sheets) is the language to redefine styles of HTML elements, such as layout, colors and font styles. In modern web design, CSS has an important role in adjusting the web page appearance to different devices and screen sizes. [5] CSS can be embedded inside an HTML element via the `style` attribute, or separated into `<style>` elements or external files. The style definition can be applied to every element in a web page. [6]

2.2 Javascript

Javascript is the programming language which adds interactivity to a web page by dynamically changing the HTML content or CSS styles. Traditionally, Javascript is understood as client-side script, which is executed by a web browser to interact with the opening web page. [7] This limitation is extended nowadays, as Javascript can run on a background process to support features like push notification or background sync. [8]

Using Node.js as a Javascript runtime environment, Javascript source code can be executed as a standalone application and interact with the operating system, allowing Javascript to be a server-side programming language like PHP, Java or Python. Node.js

uses event-driven architecture with non-blocking operations that makes it become a high-performance server-side language. [9]

There are hundreds of thousand Node.js libraries written by the developer community. NPM is the dependency manager tool for Node.js applications. It arranges the libraries in place and manages version conflicts so that libraries can be integrated into Node.js applications. [10]

2.3 Browser

A browser is an application which runs on a user's device to render a web page from HTML, CSS, and Javascript content. Figure 1 shows a report from W3Counter website about the most common browsers in March 2019. Chrome is an open-source browser, developed by Google Inc., which is leading the browser market with 63.6%, followed by the Safari browser from Apple Inc with 13.2%. Other browsers such as Mozilla Firefox, Microsoft Internet Explorer and Edge, Opera are becoming less popular.

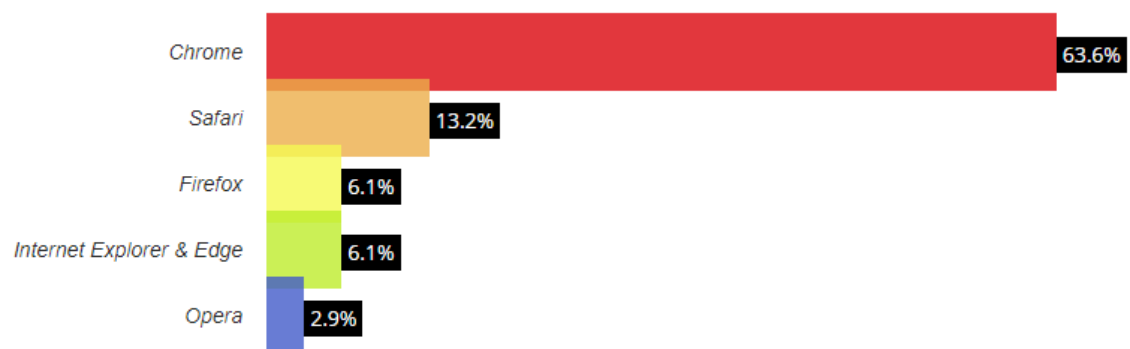


Figure 1. Browser popularity in March 2019 [11].

3 WEB COMPONENT CONCEPTS

When programming software applications, developers try to reuse source code in different ways. In Functional Programming, common pieces of code are put into functions in order to be recalled quickly by the function's name. In Object-oriented Programming, common variables and functions are wrapped into an object as properties and methods.

In web development, developers also try to do the same. However, the web environment includes many other factors. For example, it includes different types of source code such as HTML, CSS and Javascript. Also, when these codes are put together into a web page, they can affect each other. Therefore, encapsulating an implementation in web development is more complicated. It requires a same set of source code must always render the same elements in browsers whenever it is reused and does not affect other elements unexpectedly.

Web Component is introduced as a suite of technologies to encapsulate web source codes into a single component, making it easier to be reused in any web application. [12]

This chapter explains fundamental technologies related to Web Component, including DOM, Shadow DOM, HTML Template, HTML Custom Element, and how they are combined to create web components.

3.1 DOM

The DOM (Document Object Model) is a tree data structure that represents a web page – or web document - in the way that a browser can understand. From the developer's aspect, DOM is an application programming interface (API) for Javascript can interact with a web document such as accessing its elements and modifying element's properties. [13]



Figure 2. The role of DOM between original source code and web page.

As described in Figure 2, elements in a web page appear as original source code when they were developed by a developer. After being sent to a browser via network connection, the browser translates the source code into a DOM structure then render the web page on the user's device. Javascript running in the web browser can modify the web page by changing elements in the DOM structure.

The original source code can be similar to DOM structure, but can also be very different. For example, Figure 3 shows the original source code of a simple HTML web page.

```

1  <!DOCTYPE html>
2  <html>
3    <body onload="prepare()">
4      <h1>My First Heading</h1>
5      <p></p>
6    </body>
7    <script>
8      function prepare() {
9        document.querySelector('p').innerText = "This is a dynamic content.";
10     }
11   </script>
12 </html>

```

Figure 3. Original source code of a simple web page.

However, the DOM structure is different. As in Figure 4, there is a `<head></head>` section added by the browser. The browser also executed the Javascript code and modified the `<p>` element by adding the text "This is a dynamic content."

```

<html>
  <head></head>
  ▼<body onload="prepare()">
    <h1>My First Heading</h1>
    <p>This is a dynamic content.</p>
    ▼<script>
      function prepare() {
        document.querySelector('p').innerText = "This is a dynamic content.";
      }
    </script>
  </body>
</html>

```

Figure 4. DOM structure of a simple web page.

After that, the web page was rendered with contents based on this DOM structure as shown in Figure 5.



Figure 5. The simple web page in user's device.

Each element in the DOM structure is called as a *node*. To program with a node, a Javascript developer need to work with the DOM interface which represents that node. For example, in Figure 4, `document.querySelector('p')` returns a `HTMLParagraphElement` interface.

3.2 Shadow DOM

There are many problems related to scoping when building web applications using traditional HTML, CSS and Javascript [14], such as:

- Style override: the document styles may override each other when they are shared across the whole web page.
- Script alteration: the document Javascript may alter some parts of unexpected elements.
- ID overlap: the `id` attribute of elements in a document can be duplicated, which can lead to selection issues.

“Shadow DOM fixes CSS and DOM. It introduces scoped styles to the web platform.”

[15] The benefits from Shadow DOM includes:

- Scoped CSS: styles defined inside Shadow DOM can only be applied for elements of it.
- Isolated DOM: elements inside a Shadow DOM is invisible to the global web page.

- Simplified CSS: ID and class names of HTML elements can be more generic as they are not conflicted with other elements outside of its Shadow DOM.

Creating Shadow DOM

Shadow DOM can be attached into elements of the regular DOM. Those elements are: article, aside, blockquote, body, div, footer, h1, h2, h3, h4, h5, h6, header, main, nav, p, section, span and valid custom elements. [16]

Figure 6 illustrates the relationship between Shadow DOM and the regular DOM.

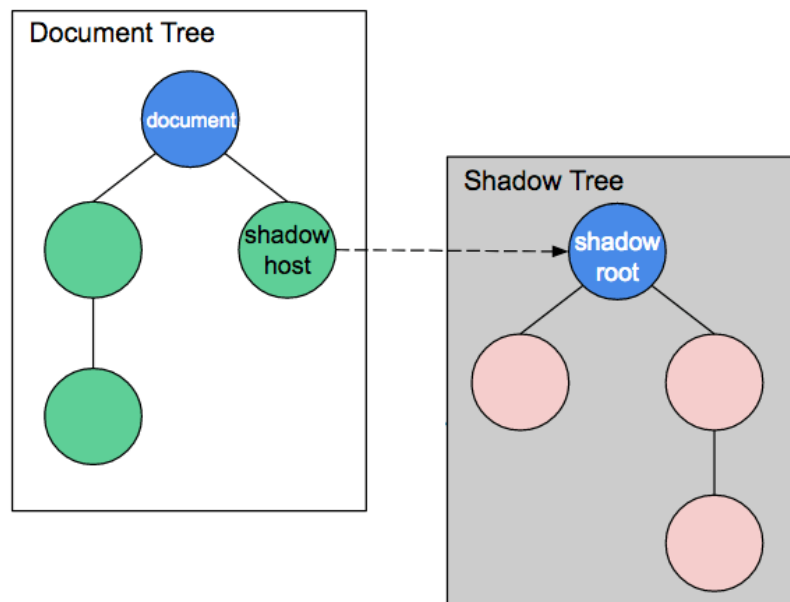


Figure 6. Shadow DOM inside regular DOM [17].

The Document Tree represents a regular DOM structure as a tree with different nodes. For example, the top-level node is `document`, which represents the web page. Shadow DOM introduces some new terminologies, including:

- Shadow host: the root node of the Shadow DOM which can be access from the regular DOM
- Shadow tree: the detail DOM structure of the shadow host
- Shadow root: the most top-level node inside a Shadow DOM

A shadow host can be created and attached into a regular DOM element using the `attachShadow()` Javascript method. This method returns the shadow root, which can be modified as a regular HTML element.

Figure 7 demonstrates how a new shadow host is attached to a `<div>` element. There are style definitions for `h1` elements with the same CSS selectors. The first one `h1 { color: green; font-size: 1.5rem }` is defined in the web document and will be applied for elements in the regular DOM tree. The second one `h1 { color: blue; font-size: 1rem }` is defined inside the Shadow DOM and only affects elements inside the shadow tree.

```

1  <!DOCTYPE html>
2  <html>
3    <body onload="prepare()">
4      <style>
5        | h1 { color: green; font-size: 1.5rem }
6      </style>
7      <h1>Heading of the global web page</h1>
8      <div></div>
9    </body>
10   <script>
11     function prepare() {
12       var wrapper = document.querySelector('div');
13       var shadowRoot = wrapper.attachShadow({ mode: 'open' });
14       shadowRoot.innerHTML = `
15         <style>
16           | h1 { color: blue; font-size: 1rem }
17         </style>
18         <h1>Heading inside a Shadow DOM</h1>
19       `;
20     }
21   </script>
22 </html>

```

Figure 7. Attaching a Shadow DOM.

The original source code above is translated to the DOM structure as Figure 8. There is a `#shadow-root (open)` element as the root node of the shadow tree.

```

<html>
  <head></head>
  ▼<body onload="prepare()">
    <style>
      h1 { color: green; font-size: 1.5rem }
    </style>
    <h1>Heading of the global web page</h1>
    ▼<div>
      ▼#shadow-root (open)
        <style>
          h1 { color: blue; font-size: 1rem }
        </style>
        <h1>Heading inside a Shadow DOM</h1>
      </div>
    ▶<script>...</script>
  </body>
</html>

```

Figure 8. DOM structure of a web page with Shadow DOM.

As a result, in Figure 9, the two h1 elements were rendered differently on the browser. Even CSS selectors are the same, they do not overlap each other.

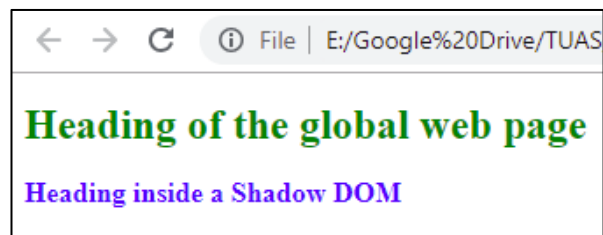


Figure 9. Result of Shadow DOM.

Compatibility

Shadow DOM is not fully supported by all browsers. Its compatibility can be checked by an online validation tool as in Figure 10. [18] Firefox, Chrome and Opera browsers show a full support for Shadow DOM feature. Safari, Microsoft IE and Edge have a limited support or no support at all.

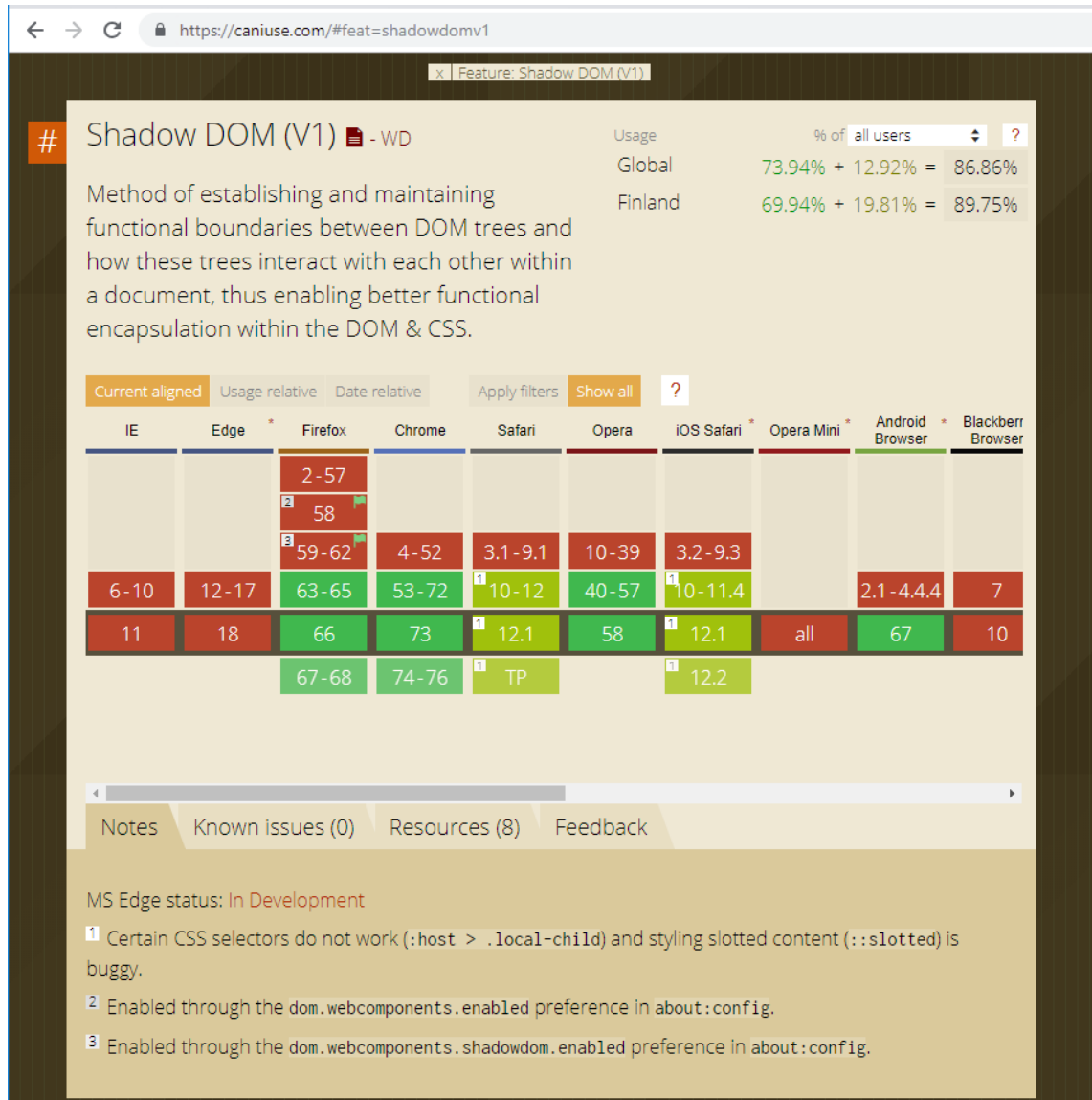


Figure 10. Compatibility of Shadow DOM with different browsers.

3.3 HTML Template

HTML Template is a special HTML element which encloses a client-side content including CSS and Javascript, making it completely be disabled from the web application but may subsequently be instantiated during runtime using JavaScript. [19]

```

1 <!DOCTYPE html>
2 <html>
3   <body>
4     <style>
5       h1 { color: green; font-size: 1.5rem }
6     </style>
7     <h1>HTML Template example</h1>
8     <template id="my-template">
9       <style>
10        h1 { color: blue; font-size: 1rem }
11      </style>
12      <h1>This is content from a template.</h1>
13    </template>
14    <button onclick="loadContent()">Load content</button>
15  </div></div>
16 </body>
17 <script>
18   function loadContent() {
19     var template = document.querySelector("#my-template");
20     var wrapper = document.querySelector('div');
21     var shadowRoot = wrapper.attachShadow({ mode: 'open'});
22     shadowRoot.innerHTML = template.innerHTML;
23   }
24 </script>
25 </html>

```

Figure 11. Example of using HTML Template element.

Figure 11 gives an example of using Template element. Its content is defined as a normal HTML element but is not rendered. When a user clicks on “Load content” button, a Javascript function is executed to insert the template content into a Shadow DOM, as shown in Figure 12.

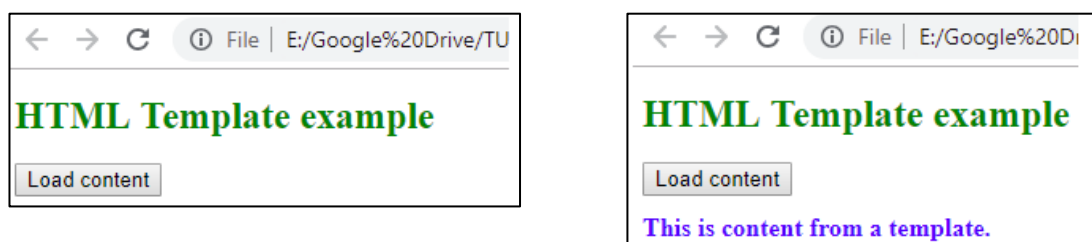


Figure 12. Before and after processing HTML Template content.

In the DOM structure, a HTML Template element wraps a fragment of HTML markup inside a `#document-fragment` node, as shown in Figure 13. This helps to prevent any side-effect of the template content on the web page.

```
<html>
  <head></head>
  ▼ <body>
    <style>
      h1 { color: green; font-size: 1.5rem }
    </style>
    <h1>HTML Template example</h1>
    ▼ <template id="my-template">
      ▼ #document-fragment
        <style>
          h1 { color: blue; font-size: 1rem }
        </style>
        <h1>This is content from a template.</h1>
      </template>
      <button onclick="loadContent()">Load content</button>
      <div></div>
      ▶ <script>...</script>
    </body>
  </html>
```

Figure 13. Document fragment of a HTML Template element.

Compatibility

As shown in Figure 14, HTML Template feature is widely supported by most of the browsers except Microsoft IE.

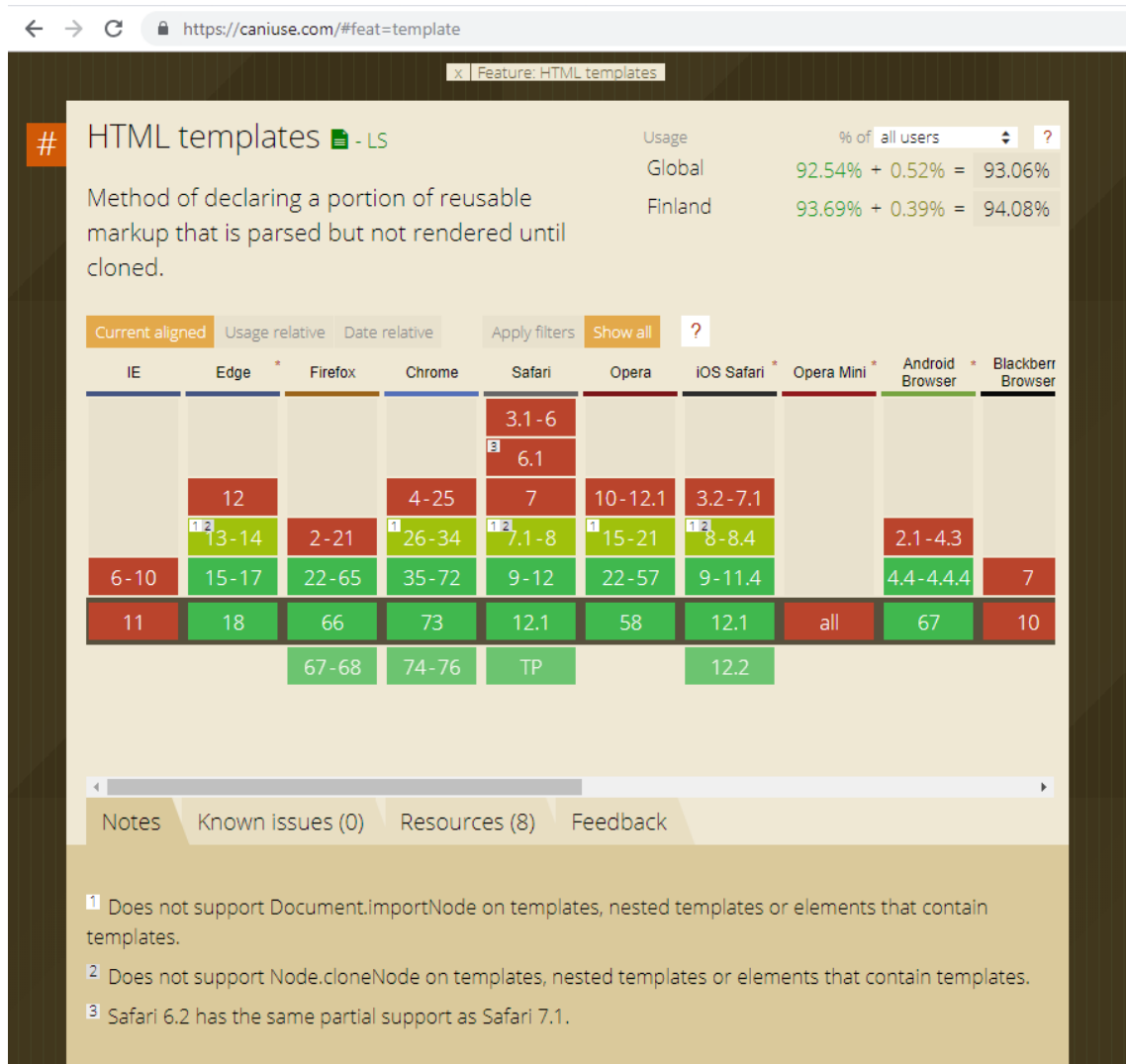


Figure 14. Compatibility of HTML template with different browsers.

3.4 HTML Custom Element

There are many standard HTML elements which are supported by most of the browsers. With the power of Javascript, `HTMLElement` interface allows web developers to define new types of HTML elements, or custom elements. [20]

By extending the `HTMLElement` interface, a custom element has all the functionalities of a standard HTML element plus its own functionalities. With a semantic name, a custom element improves the readability of a web document. A valid custom element name must match the requirements from WHATWG community [21] including:

- The name must have at least two paths starting with an alphabet lower case character, dividing by a dash "-".
- The name must not be any of the preserved names.

Figure 15 shows an example of a simple custom element named `<my-element>`. In the body of the web page, this element is used like a standard HTML element, with a customized attribute named `color`. The element is defined as a Javascript class `MyElement`. In line 38, the `customElements` property of `window` interface helps to register a new custom element, making it usable in the web document.

```

1  <!DOCTYPE html>
2  <html>
3    <body>
4      <h1>HTML Custom Element example</h1>
5      <my-element color="blue"></my-element>
6      <button onclick="switchColor()">Switch color</button>
7    </body>
8    <script>
9      function switchColor() {
10       var element = document.querySelector('my-element');
11       element.color = element.color === 'green' ? 'blue' : 'green';
12     }
13
14     class MyElement extends HTMLElement {
15       static get observedAttributes() {
16         return ['color'];
17       }
18
19       get color() {
20         return this.getAttribute('color');
21       }
22       set color(value) {
23         this.setAttribute('color', value);
24       }
25
26       attributeChangedCallback(name, oldValue, newValue) {
27         if (name === 'color') {
28           this.innerHTML = `<style>* {color: ${newValue}}</style>${this._getExampleContent()}`;
29         }
30       }
31
32       _getExampleContent() {
33         return `
34         <h3>This is content of the custom element.</h3>
35         <p>Current color: ${this.color}</p>`;
36       }
37     };
38     window.customElements.define('my-element', MyElement);
39   </script>
40 </html>

```

Figure 15. Example of a custom element.

In the example above, the `my-element` element has a `color` attribute with its initial value is `blue`. This setting will trigger the `attributeChangedCallback` method because `color` attribute is under observation of the browser, as defined in `observedAttributes` method. This attribute can be adjusted in the same way as a standard HTML element using the Javascript function such as `switchColor` function in this example.

Compatibility

In Figure 16, Custom Element shows a similar compatibility to Shadow DOM. It is supported in open-source browsers such as Firefox and Chrome, but can not be fully functional in Safari, IE and Edge.



Figure 16. Compatibility of Custom Element with different browsers.

3.5 Web Component

Web component is a meta-specification made possible by four other specifications [22]:

- The Custom Elements specification
- The Shadow DOM specification
- The HTML Template specification
- The ES Module specification

ES Module is a new feature of Javascript 2015, or ES6, which allows developers to separate the Javascript logic into different parts – or modules – by using Module Exports and recall the exported modules at different places by using Module Imports [23].

The combination of the four specifications above allows developers to define their own custom elements with encapsulated and isolated styles in Shadow DOM, that can be reused many times as templates by ES Module. A custom element which matches these specifications is considered a web component.

Figure 17 demonstrates HTML source code of a simple web application which has several `card-ticket` elements with different attributes. Each `card-ticket` element is an instance of the web component. This web component is defined in the `CardTicket.js` file and is imported using ES6 Module syntax.

```
2 <html>
3   <body>
4     <h1>Web Component Example</h1>
5     <card-ticket></card-ticket>
6     <card-ticket color="blue" content="Ticket 1"></card-ticket>
7     <card-ticket color="green" content="Ticket 2"></card-ticket>
8   </body>
9   <script type="module">
10    import { CardTicket } from './CardTicket.js';
11    window.customElements.define('card-ticket', CardTicket);
12  </script>
13 </html>
```

Figure 17. An example web application using web components.

The `CardTicket.js` file is shown in Figure 18. It is similar to a HTML custom element, but the element's content is encapsulated by Shadow DOM. Therefore, CSS declarations are not leaked into other elements in the application.

In modern web component development, the web component's content is managed by Javascript and would not be rendered to the web document by default. This approach reduces the used of HTML Template and gives Javascript the full power to control the content.

```

1  class CardTicket extends HTMLElement {
2      static get observedAttributes() {
3          return ['color', 'content'];
4      }
5
6      constructor() {
7          super();
8          this.root = this.attachShadow({ mode: 'open' });
9          this.color = "black";
10         this.content = "(No content)";
11         this.isSelected = false;
12         this.addEventListener('click', () => this._selected());
13     }
14
15     _selected() {
16         this.isSelected = !this.isSelected;
17         this.connectedCallback();
18     }
19
20     attributeChangedCallback(name, oldValue, newValue) {
21         this[name] = newValue;
22     }
23
24     connectedCallback() {
25         this.root.innerHTML = `<style>
26             :host {
27                 display: inline-grid; border-radius: .3rem;
28                 min-width: 8rem; min-height: 8rem;
29                 text-align: center; margin: .5rem;
30                 border: 1px solid #AAA; cursor: pointer;
31                 ${this.isSelected ? `box-shadow: 0px 0px .3rem .3rem rgba(100,100,100,0.25);` : ``}
32             }
33             * { color: ${this.color} }
34             h1 { font-size: 1.2rem; text-transform: uppercase;}
35             p {font-size: .9rem; }
36         </style>
37         <h1>${this.color}</h1><p>${this.content}</p>`;
38     }
39 };
40
41 export {CardTicket};

```

Figure 18. Javascript declaration of the CardTicket web component.

As a result, in Figure 19, the application renders three similar tickets with different styles. Clicking on a ticket will update styles for and only for the target ticket.

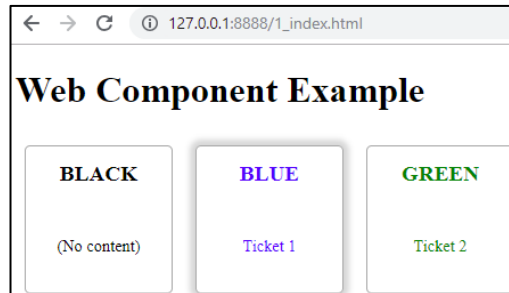


Figure 19. Example web application using the CardTicket web component.

Compatibility

As the combination Shadow DOM, HTML Template, Custom Element and ES Module, Web Component requires each specification to be supported by the browsers. Some browsers are still in the updating process to support the standards for Web Component. In the meantime, Polyfills library simulates the missing browser capabilities as closely as possible. [24]

In a web project, Polyfills library `@webcomponents/webcomponentsjs` can be installed via npm. After that, the web application can load this library as a normal Javascript file as shown in Figure 20, enabling a better support for Web Component in different browsers.

```
<!-- load webcomponents bundle, which includes all the necessary polyfills -->  
<script src="node_modules/@webcomponents/webcomponentsjs/webcomponents-bundle.js"></script>  
  
<!-- load the element -->  
<script type="module" src="my-element.js"></script>  
  
<!-- use the element -->  
<my-element></my-element>
```

Figure 20. Loading Polyfills library.

4 LIT-ELEMENT LIBRARY

“LitElement is a simple base class for creating fast, lightweight web components that work in any web page with any framework.” [25] A developer can create a new web component by extending this class instead of the `HTMLElement` interface. This class is packed in lit-element library.

The lit-element library is developed by Polymer Project team, funded by Google Inc. This team is also the author of Polymer library for web component development, but this library was deprecated and lit-element is recommended instead.

LitElement base class gives developers some advantages, such as:

- **Fast and light:** the library has lit-html template engine, which helps to render HTML content inside a Shadow DOM and update only the dynamic parts of the web page.
- **Friendly declarative:** the element is defined in Javascript with all the powerful features of this programming language. LitElement also supports a clearer syntax for defining properties, data binding, templates, styles and events.
- **Highly compatible:** LitElement follows Web Component specification, enabling its compatibility with other front-end web frameworks.

This chapter investigates lit-element library including project configuration and the main features of LitElement base class.

4.1 Polymer CLI

Polymer CLI is the official tool for lit-element projects and Web Component. After being installed, it runs as a command line with a set of options for a developer can initialize projects with pre-defined structures, start a development server for a project, run Web Component unit tests and more.

The common command lines include:

- `polymer build`: bundles the project for production. The process includes simplifying source code and fulfilling configurations for a complete application.

- `polymer serve`: runs a local web server for the project. This server helps developers to test the web application in a browser without the deployment process.
- `polymer test`: runs the unit tests in the `test` directory of the project.

The installation instruction of Polymer CLI can be found in the official website <https://polymer-library.polymer-project.org> of The Polymer Project Authors.

4.2 LitElement component

Figure 21 shows an implementation of CardTicket component using LitElement base class. This web component has the same functionalities and styles as the implementation without LitElement in Figure 18.

In general, the basic steps to create a web component with LitElement includes:

- Use ES6 Module Import to load the LitElement base class and the html helper function from lit-element library.
- Define a new class which is a descendant of LitElement base class.
- Implement the `render` method to define the content of the component.
- Register the component's HTML tag with the browser.

Comparing the two implementations of CardTicket component, there are two obvious advantages from using LitElement:

- Shadow DOM is used implicitly without any manual step.
- Attributes of the element are understood as the component's properties. Therefore, developers do not need to update those properties manually by using `attributeChangedCallback` method.

The two implementations share the same way of usage. Each requires the web application to import the Javascript file first, then it can be used as a normal HTML tag, as shown in Figure 17.

However, to start a lit-element project, a developer needs to run a local server using `polymer serve` as mentioned in Section 4.1. This server resolves the dependencies from ES6 Module Import statements for the browser can load them properly.

```

1  import { LitElement, html } from 'lit-element';
2
3  class CardTicket extends LitElement {
4    static get properties() {
5      return {
6        color: String, content: String, isSelected: Boolean
7      };
8    }
9
10   constructor() {
11     super();
12     this.color = "black";
13     this.content = "(No content)";
14     this.isSelected = false;
15     this.addEventListener('click', () => this._selected());
16   }
17
18   _selected() {
19     this.isSelected = !this.isSelected;
20     this.connectedCallback();
21   }
22
23   render() {
24     return html`<style>
25       :host {
26         display: inline-grid; border-radius: .3rem;
27         min-width: 8rem; min-height: 8rem;
28         text-align: center; margin: .5rem;
29         border: 1px solid #AAA; cursor: pointer;
30         ${this.isSelected ? `box-shadow: 0px 0px .3rem .3rem rgba(100,100,100,0.25);` : ``}
31       }
32       * { color: ${this.color} }
33       h1 { font-size: 1.2rem; text-transform: uppercase;}
34       p {font-size: .9rem; }
35     </style>
36     <h1>${this.color}</h1><p>${this.content}</p>`;
37   }
38 }
39 customElements.define('card-ticket', CardTicket);

```

Figure 21. Declaration of the CardTicket web component using LitElement.

4.3 Templates

The template for a LitElement component is defined in `render` method of the element class. This method returns a `TemplateResult` object by using `html` helper function from `lit-html` template engine.

As can be seen in Figure 21, properties of the element class can be inserted into the template using string substitution `${ }` syntax. Since then, content of the template will be updated asynchronously with property's changes, and only the affected DOM parts will be re-rendered.

A recommended implementation for an element's template is to make it as a pure function of the element's properties:

- The `render` method should not change element's property values.
- Only the element's properties can affect the template.
- The `render` method should always return the same result with the same property values.

After being rendered, the DOM structure of a web component should not be changed by any other factor except its properties.

Lit-html template engine supports flexible syntaxes to render a template based from property values, such as looping and conditionals, as shown in Figure 22.

```
html`
<!-- Loop syntax -->
<ul>
  | ${this.myArray.map(i => html`<li>${i}</li>`)}
</ul>

<!-- Conditional syntax -->
${this.myBool?
  | html`<p>Render some HTML if myBool is true</p>`:
  | html`<p>Render some other HTML if myBool is false</p>`}
`;
```

Figure 22. Loop and Conditional syntax.

Data binding is another powerful feature of lit-html engine. Text content, attributes, boolean attributes, properties and events of a child element can be binded to a value from its parent element and automatically updated whenever that value is changed. [26]

Data binding is one-direction from parent to child element. To share data in the opposite direction, a child element can fire an event for a parent element to capture it.

4.4 Styles

By default, style is rendered inside Shadow DOM. It can be defined in a separate method of the element class, in the `<style>` tag of the element's template, or in an external stylesheet.

Define styles as a static property

LitElement supports defining CSS in the `styles` static property of the element class. This property should contain the styles which can be applied for all instances of a web component across the whole web application. This approach helps improving performance when rendering components.

The `styles` property can be defined with a `css` helper function, as shown in Figure 23.

```
import {LitElement, css} from 'lit-element';

class MyElement extends LitElement {
  static get styles() {
    return css`
      :host {
        display: block;
      }`;
  }
}
```

Figure 23. LitElement static style property definition.

Define styles in the element's template

This approach is useful when styles can be changed depend on the component property's values. Figure 21 shows an example that styles of a `<card-ticket>` element are varied from its attributes `color`, `content` and its state `isSelected`. Therefore, styles of CardTicket component need to be defined in `render` method.

Define styles in an external stylesheet

External stylesheet is designed for loading CSS generated from a CSS preprocessor such as SASS or LESS. It is defined by a `<link>` tag in the component template, as shown in Figure 24.

```
class MyElement extends LitElement {  
  render() {  
    return html`  
      <link rel="stylesheet" href="./styles.css">  
    `;  
  }  
}
```

Figure 24. LitElement external style definition.

However, it comes with some side-effects:

- External styles require additional HTTP requests. The component could be rendered without styles when the requests are in process.
- The external URL in `href` attribute depends on the application but not the component itself. Thus, this URL may not be found in different web applications.

4.5 Attributes and Properties

A web component manages their data via its attributes and properties. When the component is used in a web application by a HTML tag, this tag creates an instance of the web component and may contain some attributes. In other words, attributes belong to a specific instance of the web component.

By default, an attribute is reflected as a property in the element class definition. Inside the class, other logics could be implemented depending on the properties. Properties can be declared in `static get properties()` method. Figure 21 shows an example that `color`, `content` and `isSelected` are three properties of `CardTicket` component, which `color` and `content` were used as attributes of `CardTicket` instances in the web application as in Figure 17.

5 IMPLEMENTATION

This chapter describes the implementation of the Phonebook App prototype and demonstrates how lit-element library can help to develop and integrate web components into a complete web application. The implementation includes modeling, development, testing and deployment. The prototype focused on working with web components instead of advanced functionalities.

5.1 Phonebook App project

The Phonebook App helps users to manage their contacts by adding, removing or updating a contact information. A user can also bookmark a contact to put it into the bookmark list.

As a prototype, there are some limitations related to business features such as lacking advanced functionalities for searching for a contact, sorting the contact list, etc. Besides, data from users is stored in the device's memory and will be removed when reloading the app.

This is a Progressive Web App with responsive design and offline access features. It can be installed on mobile devices like a native mobile application. [27]

5.2 Use case diagrams

Use case diagrams visualize the functional requirements of a software application. In Figure 25, the actor for Phonebook App is a user who accesses the application. This user can act on different features of the app, such as viewing contact list or adding a new contact, etc. Each feature is represented as a use case in an oval.

The <<extend>> relationship between use cases indicates a behavior which can be done under a certain use case. For example, in Figure 25, a user can remove a contact while viewing the contact list.

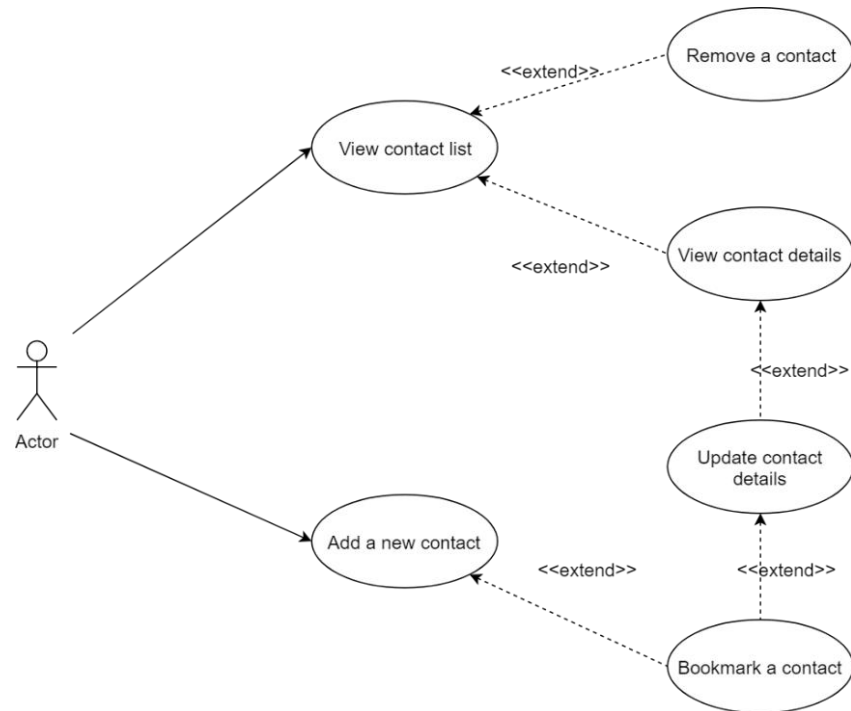


Figure 25. Use-case diagram of Phonebook App.

Tables 1 to 6 explain each use case using the Rational Unified Process format [28] including brief description, the basic and the alternative flow of behaviors, special requirements, the system status before and after the use case and the extension points.

Table 1. Use case "View contact list".

Use case 1: View contact list	
Brief discussion	This use case allows the user to view the list of all contacts storing in the app.
Basic flow	The use case starts when the user opens the app via a web browser.
Alternative flow	- There is no contact in the app: + The app shows "There is no contact" message.
Special requirements	The app must have one or more contacts in its memory.
Before the use case	The app has not opened.
After the use case	- Success: the app shows list of contacts in the Contacts section - Failure: the error message is shown.
Extension points	While viewing contact list, the user can remove a contact (use case 3) or view contact detail (use case 4).

Table 2. Use case "Add a new contact".

Use case 2: Add a new contact	
Brief discussion	This use case allows the user to input a new contact to the app.
Basic flow	<ul style="list-style-type: none"> - The use case starts when the user clicks on "Add Contact" button on the left sidebar. - The app displays the "Add Contact" form. - The user enters the contact information. - The user clicks "Add" button.
Alternative flow	<ul style="list-style-type: none"> - The user clicks "Cancel" button in the "Add Contact" form. + The app closes the form.
Special requirements	(none)
Before the use case	The app is opened.
After the use case	<ul style="list-style-type: none"> - Success: the app closes the form and shows the new contact in the contact list. - Failure: the app shows no changes.
Extension points	While creating a new contact, the user can bookmark the new contact (use case 6).

Table 3. Use case "Remove a contact".

Use case 3: Remove a contact	
Brief discussion	This use case allows the user to remove a contact from the app.
Basic flow	The use case starts when the user clicks on the (X) icon on top-right corner of a contact in the contact list.
Alternative flow	<ul style="list-style-type: none"> - There is no contact in the app: + The app shows "There is no contact" message.
Special requirements	There is at least one contact in the app.
Before the use case	The app shows list of contacts.
After the use case	<ul style="list-style-type: none"> - Success: the target contact is removed. - Failure: the target contact remains unchanged.
Extension points	(none)

Table 4. Use case "View contact details".

Use case 4: View contact details	
Brief discussion	This use case allows the user to view detail information of a contact.
Basic flow	<ul style="list-style-type: none"> - The use case starts when the user clicks on a contact in the contact list. - The app shows the "Update contact" form with detail information of the selected contact. - The user clicks "Cancel" button. - The app closes the form.
Alternative flow	<ul style="list-style-type: none"> - The user clicks "Update" button in the "Update Contact" form. + The app closes the form with the updated information.
Special requirements	There is at least one contact in the app.
Before the use case	The app shows the contact list.
After the use case	The app backs to its previous state without any change.
Extension points	While viewing contact details, the user can update the contact information (use case 5).

Table 5. Use case "Update contact details".

Use case 5: Update contact details	
Brief discussion	This use case allows the user to update detail information of a contact.
Basic flow	<ul style="list-style-type: none"> - The use case starts when the user clicks on a contact in the contact list. - The app shows the "Update contact" form with detail information of the selected contact. - The user updates the contact information. - The user clicks on "Update" button.
Alternative flow	<ul style="list-style-type: none"> - The user clicks "Cancel" button in the "Update Contact" form. + The app closes the form.
Special requirements	There is at least one contact in the app.

Before the use case	The app shows the contact list.
After the use case	<ul style="list-style-type: none"> - Success: the form is closed, and the app shows the updated information of the edited contact. - Failure: the edited contact remains unchanged.
Extension points	While updating contact detail, the user can bookmark the contact (use case 6).

Table 6. Use case "Bookmark a contact".

Use case 6: Bookmark a contact	
Brief discussion	This use case allows the user to add a contact into the bookmark list.
Basic flow	<ul style="list-style-type: none"> - The use case starts when the user creates a new contact or updates a contact information. - The app shows the "Add contact" form or "Update contact" form with a checkbox "Add to bookmark". - The user ticks the "Add to bookmark" checkbox. - The user clicks on "Add" or "Update" button.
Alternative flow	<ul style="list-style-type: none"> - The user clicks "Cancel" button in the forms. + The app closes the form.
Special requirements	There is at least one contact in the app.
Before the use case	The app shows the "Add contact" form or "Update contact" form.
After the use case	<ul style="list-style-type: none"> - Success: the app shows the edited contact in Bookmark list. - Failure: the edited contact remains unchanged.
Extension points	In the "Update contact" form, if the user removes the tick in "Add to bookmark" checkbox and click "Update" button, the contact is removed from the Bookmark list.

5.3 Development

Development Environment

The Phonebook App prototype was developed in multiple operating systems including Windows 10 and MacOS Mojave 10.14. The source code is published in Github.com at <https://github.com/tanbt/PolymerPractice/tree/master/phonebook-app>.

The editor had been used during development was Visual Studio Code. This is a free and lightweight application of Microsoft Corporation with built-in support for HTML, CSS, Javascript and Typescript languages. A developer can install external extensions to upgrade this editor with more features.

System requirements

For the development environment, the project requires Node.js 10 and Npm to be installed already in the system. The instruction of Node.js and Npm installation can be found in its official website <https://nodejs.org>. Also, dependencies of the project need to be installed by running `npm install` command at the project directory.

Visual Studio Code should be installed as the suggested editor, but other text editors are also acceptable. The project does not depend on a specific operating system and does not have any special hardware requirements.

Git is the Source Code Management tool for managing changes of the source code during development process. This tool can be installed via its official website <https://git-scm.com>.

Project structure

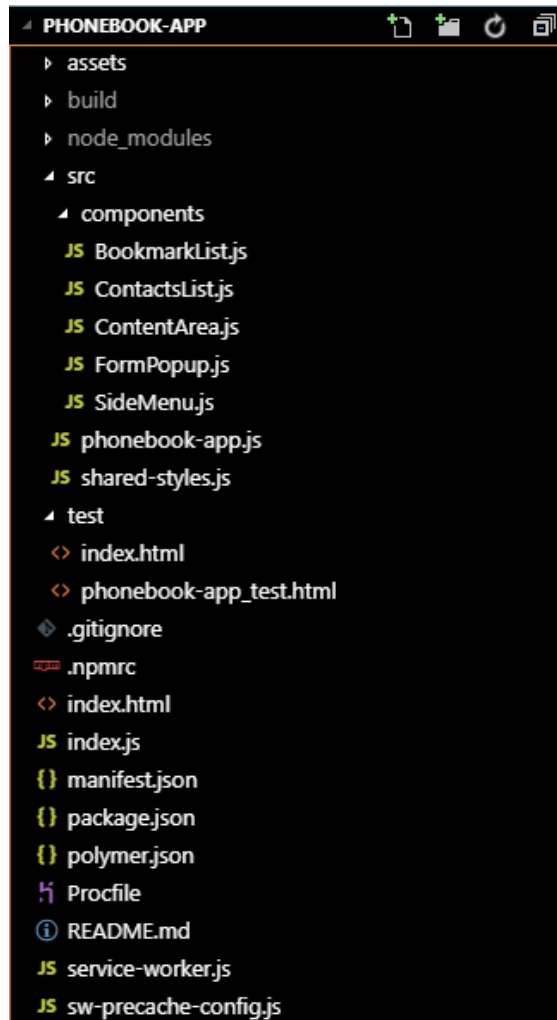


Figure 26. Phonebook App project structure.

Figure 26 depicts the structure of Phonebook App project. This structure contains source code of the application and configuration for development and deployment processes:

- `assets` directory contains static resources such as app icon, logo and pictures.
- `build` directory is generated automatically after running `polymer build` command, it contains the output of this command line.
- `node_modules` directory contains the npm dependencies, such as `lit-element` library, `Polymer CLI` tool, open-source web components from the community and other libraries. This directory is generated automatically after running `npm install` command.

- `src` directory contains the source code of the app, written in Javascript. The `components` sub-directory contains web components for different parts of the application. `phonebook-app.js` file wraps other components inside a single `<phonebook-app>` component. `shared-styles.js` file defines some CSS styles to be shared across components.
- `test` directory contains unit test cases for web components developed by `lit-element` library.
- `.gitignore` file indicates files or directories that Git should skip. Usually, they are generated files and directories.
- `.npmrc` and `package.json` defines configurations for npm such as list of dependencies, scripts to be run under npm, name and description of the project.
- `index.html` is the entry point of the app. It initializes the app with manifest configurations and service worker, loading polyfills and the `<phonebook-app>` element.
- `index.js` and `Procfile` define configurations for deploying the app to Heroku service.
- `manifest.json`, `service-worker.js` and `sw-precache-config.js` contains configurations for Polymer CLI can generate manifest information and offline caching feature.
- `README.md` provides a brief description of the project.

Application User Interface

Figure 27 shows the Phonebook App user interface in a wide-screen device. The sidebar appears on the left with the app logo and “Add contact” button. A user can resize this bar by moving its right edge.

In small-screen devices such as smartphones or tablets, the left sidebar is hidden and the logo appears in the top-right corner, as shown in Figure 28. In this user interface, a user can click on the logo to open the “Add contact” form.

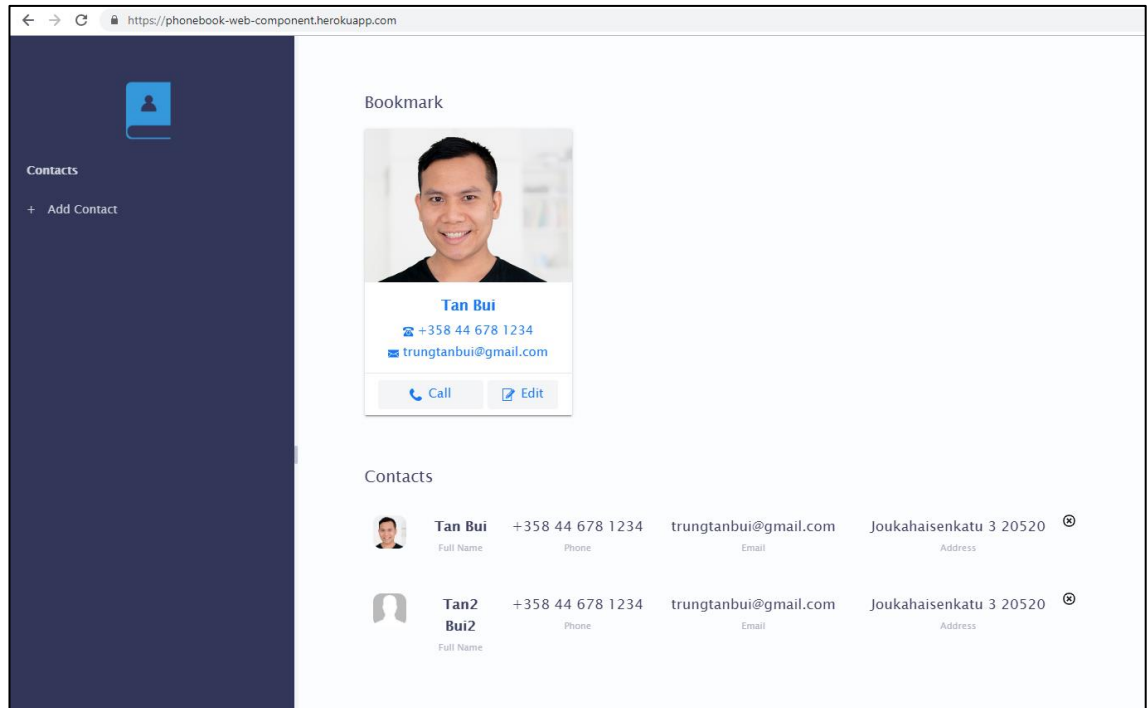


Figure 27. Phonebook App user interface in wide-screen devices.

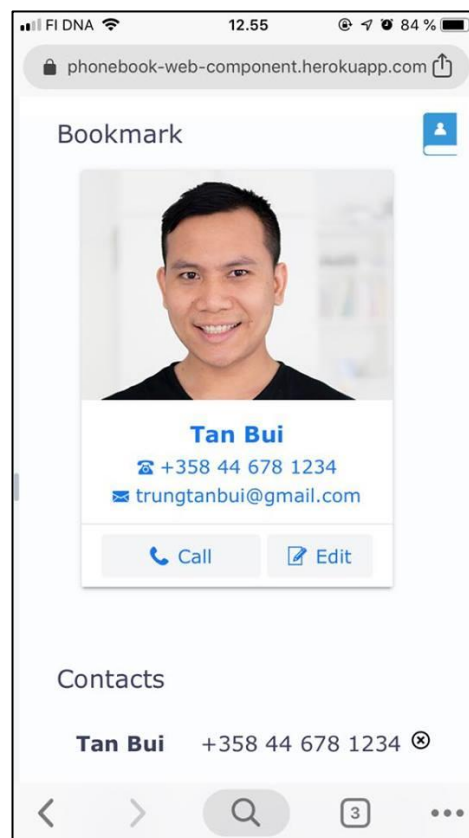


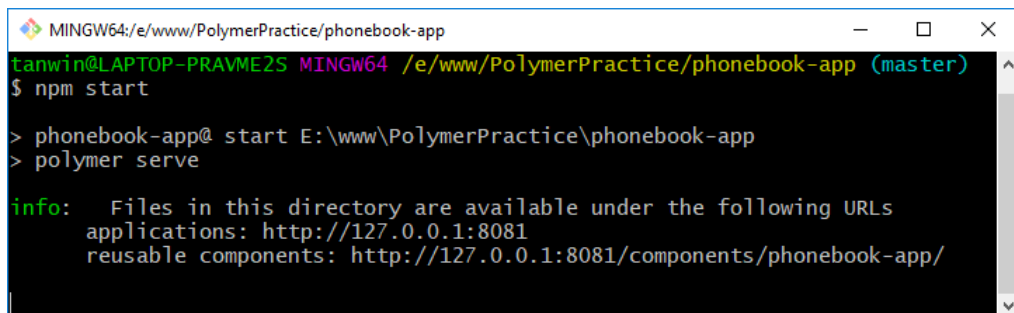
Figure 28. Phonebook App user interface in small-screen devices.

5.4 Deployment

Run the app in a local computer

After the successful completion of the development, the app can be started locally in the development computer using `npm start` command. This command runs `polymer serve` command implicitly. Polymer CLI tool starts the local server on a random port.

For example, Figure 29 shows a local server running on port 8081. The app can be accessed by a web browser in the local computer at the address `http://127.0.0.1:8081`.

A screenshot of a terminal window titled 'MINGW64:/e/www/PolymerPractice/phonebook-app'. The prompt is 'tanwin@LAPTOP-PRAVME2S MINGW64 /e/www/PolymerPractice/phonebook-app (master)'. The user enters '\$ npm start'. The output shows '> phonebook-app@ start E:\www\PolymerPractice\phonebook-app' and '> polymer serve'. Below this, an 'info:' message states: 'Files in this directory are available under the following URLs', 'applications: http://127.0.0.1:8081', and 'reusable components: http://127.0.0.1:8081/components/phonebook-app/'.

```
MINGW64:/e/www/PolymerPractice/phonebook-app
tanwin@LAPTOP-PRAVME2S MINGW64 /e/www/PolymerPractice/phonebook-app (master)
$ npm start

> phonebook-app@ start E:\www\PolymerPractice\phonebook-app
> polymer serve

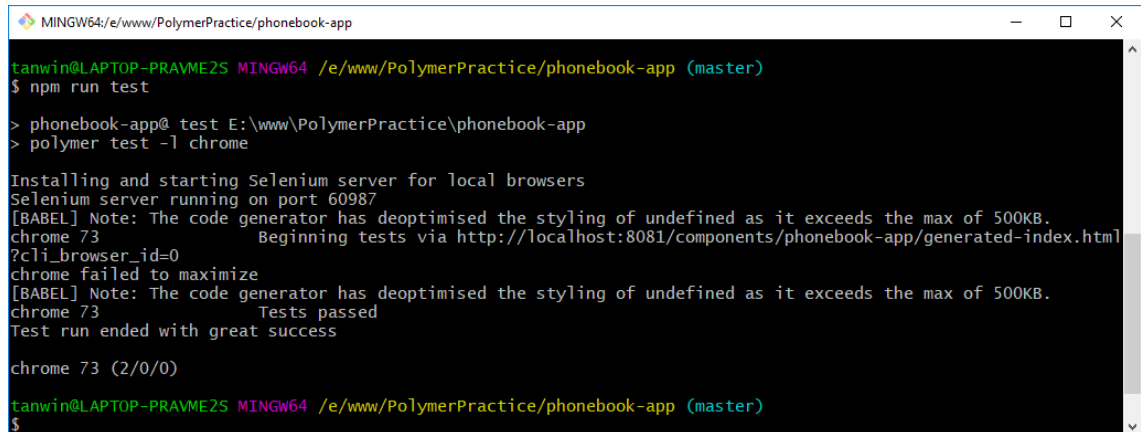
info: Files in this directory are available under the following URLs
      applications: http://127.0.0.1:8081
      reusable components: http://127.0.0.1:8081/components/phonebook-app/
```

Figure 29. Start the local server using Polymer CLI.

Run web component unit tests in a local computer

The web component unit test cases can be run in the development computer using `npm run test` command, which contains `polymer test` command to execute the test. As shown in Figure 30, there are two test cases had passed successfully.

This project is not intended to cover all the testing scenarios but introduces web component unit test implementation via two simple test cases.



```

MINGW64:/e/www/PolymerPractice/phonebook-app
tanwin@LAPTOP-PRAYME25 MINGW64 /e/www/PolymerPractice/phonebook-app (master)
$ npm run test

> phonebook-app@ test E:\www\PolymerPractice\phonebook-app
> polymer test -l chrome

Installing and starting Selenium server for local browsers
Selenium server running on port 60987
[BABEL] Note: The code generator has deoptimised the styling of undefined as it exceeds the max of 500KB.
chrome 73 Beginning tests via http://localhost:8081/components/phonebook-app/generated-index.html
?cli_browser_id=0
chrome failed to maximize
[BABEL] Note: The code generator has deoptimised the styling of undefined as it exceeds the max of 500KB.
chrome 73 Tests passed
Test run ended with great success

chrome 73 (2/0/0)

tanwin@LAPTOP-PRAYME25 MINGW64 /e/www/PolymerPractice/phonebook-app (master)
$

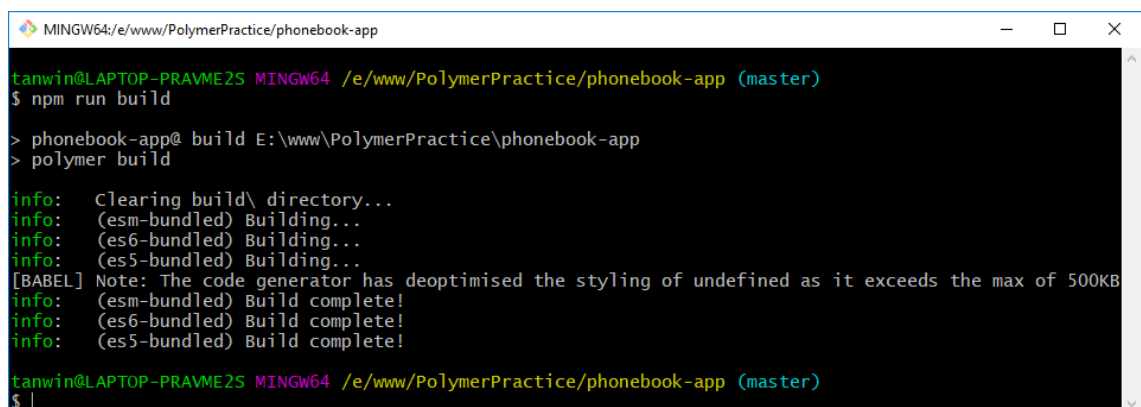
```

Figure 30. Run web component unit tests in a local computer.

Build the app for deployment

By executing `npm run build` command, Polymer CLI reads configurations in `polymer.json` file and generates different versions of the app in different bundles. Each bundle targets a specific browser support. For example, `esm-bundled` contains the generated app that can run on modern browsers which support ES Module, while `es5-bundled` supports Javascript 2009 for old browsers.

Figure 31 shows an examples that Phonebook App were packaged in to `es5-bundled`, `es6-bundled` and `esm-bundled`.



```

MINGW64:/e/www/PolymerPractice/phonebook-app
tanwin@LAPTOP-PRAYME25 MINGW64 /e/www/PolymerPractice/phonebook-app (master)
$ npm run build

> phonebook-app@ build E:\www\PolymerPractice\phonebook-app
> polymer build

info: Clearing build\ directory...
info: (esm-bundled) Building...
info: (es6-bundled) Building...
info: (es5-bundled) Building...
[BABEL] Note: The code generator has deoptimised the styling of undefined as it exceeds the max of 500KB
info: (esm-bundled) Build complete!
info: (es6-bundled) Build complete!
info: (es5-bundled) Build complete!

tanwin@LAPTOP-PRAYME25 MINGW64 /e/www/PolymerPractice/phonebook-app (master)
$ |

```

Figure 31. Bundling the application for deployment.

Each of these bundles can be uploaded to a web server and run independently as a final production of the application.

Deploy the app using Heroku service

Heroku is a cloud service for deploying web applications from different platforms, such as Node.js, Python, Java, Php, Ruby, etc. It is fast and scalable without complex configurations.

After registering a free account on Heroku.com website, a user can create a new app, as shown in Figure 32.

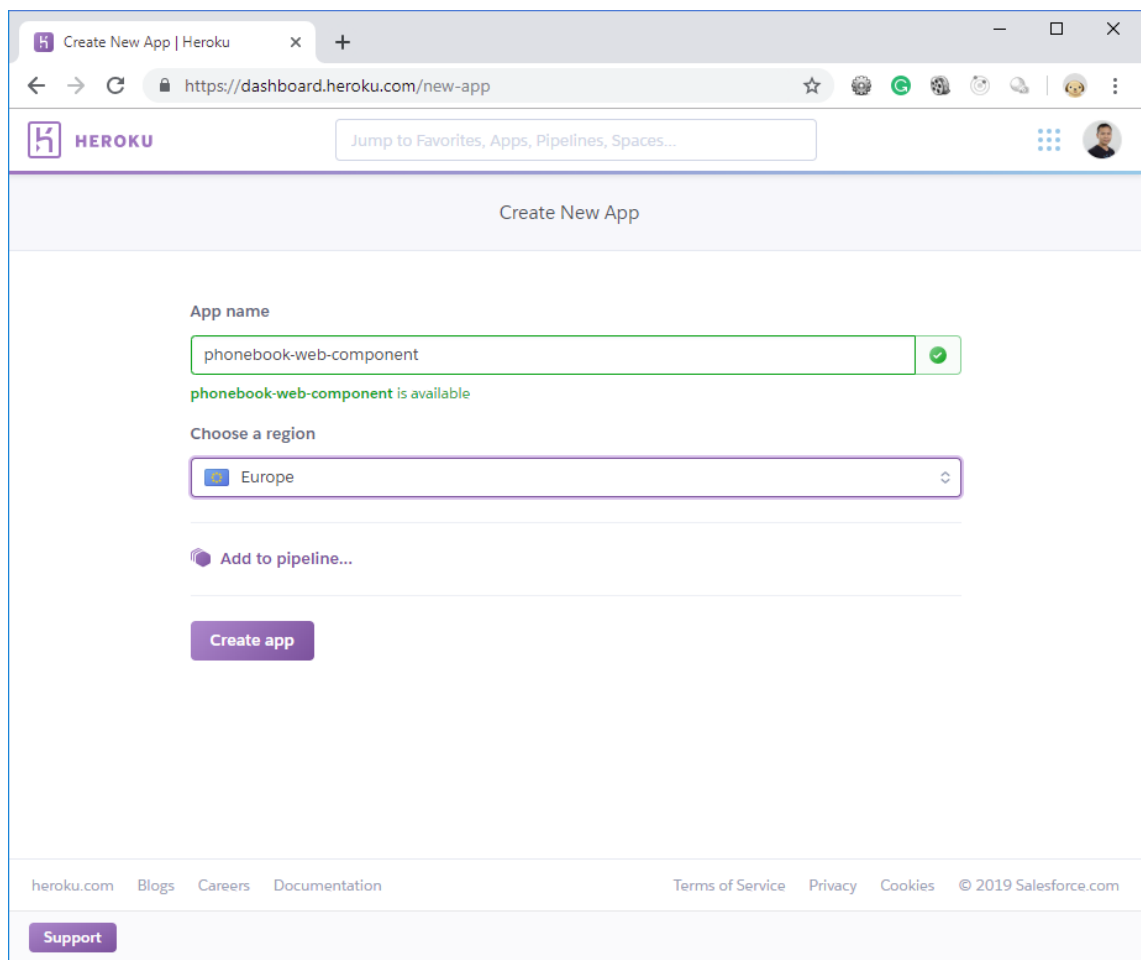


Figure 32. Create a new Heroku app.

To push an app from a local computer to Heroku service, a developer can use Heroku CLI tool, [29] which runs in the Command Prompt of Windows operating system or the in the Terminal of Linux and MacOS operating systems.

After installing Heroku CLI and logging in with a Heroku account using `heroku login` command, a developer can add the Heroku app repository to the project by running the

command `heroku git:remote -a the-app-name`, with `the-app-name` is the name defined in Figure 32.

Hence, each time the developer pushes a change into Heroku app repository by the command `git push heroku master`, the Heroku service automatically runs `npm run build` to update the app bundles then starts the app with configurations in `Procfile` file in the project.

For this project, the Phonebook App were deployed and can be accessed by any device with an internet connection at <https://phonebook-web-component.herokuapp.com/>.

6 CONCLUSION

The main goals of the thesis were to study and implement web components. A significant amount of time was invested in Web Component specification and its related technologies. The thesis did not only focus on theoretical concepts but also demonstrated them with simple examples.

The Phonebook App prototype did not include advanced features but aimed to implement several web components and integrate them into a complete web application, plus deploying the application to a production environment. To sum up, the project was managed successfully and the goals of the thesis were achieved.

Even though Web Component has not been popular in most of the web applications nowadays, its theory is important and had been implemented in different methodologies such as React Component.

The Microsoft Corporation also announced the plan of supporting Custom Element and Shadow DOM for Microsoft Edge. There are large companies that also started using web components for their products, such as Vaadin, Youtube, Tesla.

Web Component is not a silver bullet for web development. It introduces a modern approach to develop web applications which is more concise, fast, and reusable. Web Component is being supported and standardized and it is not an overstatement to conclude that Web Component might be the future of web application development.

REFERENCES

- [1] Stack Overflow. (2018). *Stack Overflow Developer Survey 2018*. [online] Available at: <https://insights.stackoverflow.com/survey/2018> [Accessed 14 Apr. 2019].
- [2] Wahlin, D. (2012). *Moving from Server-Side to Client-Side Web Development*. [online] IT Pro. Available at: <https://www.itprotoday.com/web-application-management/moving-server-side-client-side-web-development> [Accessed 4 May 2019].
- [3] Hunt, A., Thomas, D. and Cunningham, W. (2015). *The pragmatic programmer : from journeyman to master*. Boston: Addison-Wesley, p.27.
- [4] MDN Web Docs. (2019b). *HTML basics*. [online] Available at: https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/HTML_basics [Accessed 3 May 2019].
- [5] <https://www.facebook.com/lifewire> (2018). *A Look at What CSS Is (Cascading Style Sheets)*. [online] Lifewire. Available at: <https://www.lifewire.com/what-is-css-3466390> [Accessed 3 May 2019].
- [6] Codecademy. (2019). *Inline Styles in HTML | Codecademy*. [online] Available at: <https://www.codecademy.com/articles/html-inline-styles> [Accessed 3 May 2019].
- [7] Hiring | Upwork. (2018). *What is JavaScript? Bring Interactivity & Animation to the Web*. [online] Available at: <https://www.upwork.com/hiring/development/what-is-javascript/> [Accessed 3 May 2019].
- [8] Google Developers. (2019a). *Introducing Background Sync | Web | Google Developers*. [online] Available at: <https://developers.google.com/web/updates/2015/12/background-sync> [Accessed 3 May 2019].
- [9] Patel, P. (2018). *What exactly is Node.js?* [online] freeCodeCamp.org. Available at: <https://medium.freecodecamp.org/what-exactly-is-node-js-ae36e97449f5> [Accessed 14 Apr. 2019].
- [10] Npmjs.com. (2019). *npm | npm Documentation*. [online] Available at: <https://docs.npmjs.com/cli/npm> [Accessed 14 Apr. 2019].

- [11] W3counter.com. (2019). *W3Counter: Global Web Stats - March 2019*. [online] Available at: <https://www.w3counter.com/globalstats.php?year=2019&month=3> [Accessed 14 Apr. 2019].
- [12] MDN Web Docs. (2019). Web Components. [online] Available at: https://developer.mozilla.org/en-US/docs/Web/Web_Components [Accessed 14 Apr. 2019].
- [13] W3.org. (2019). *What is the Document Object Model?* [online] Available at: <https://www.w3.org/TR/WD-DOM/introduction.html> [Accessed 14 Apr. 2019].
- [14] Patel, Sandeep Kumar (2015). *Learning Web Component Development*. Birmingham: Packt Publishing, p.18.
- [15] Google Developers. (2019). *Shadow DOM v1: Self-Contained Web Components*. [online] Available at: <https://developers.google.com/web/fundamentals/web-components/shadowdom> [Accessed 14 Apr. 2019].
- [16] MDN Web Docs. (2019a). *Element.attachShadow()*. [online] Available at: <https://developer.mozilla.org/en-US/docs/Web/API/Element/attachShadow> [Accessed 14 Apr. 2019].
- [17] MDN Web Docs. (2019d). *Using shadow DOM*. [online] Available at: https://developer.mozilla.org/en-US/docs/Web/Web_Components/Using_shadow_DOM [Accessed 3 May 2019].
- [18] Caniuse.com. (2019). *Can I use... Support tables for HTML5, CSS3, etc.* [online] Available at: <https://caniuse.com> [Accessed 14 Apr. 2019].
- [19] MDN Web Docs. (2019b). *The Content Template element*. [online] Available at: <https://developer.mozilla.org/en-US/docs/Web/HTML/Element/template> [Accessed 14 Apr. 2019].
- [20] HTML5 Rocks - A resource for open web HTML5 developers. (2013). *Custom Elements: defining new elements in HTML - HTML5 Rocks*. [online] Available at: <https://www.html5rocks.com/en/tutorials/webcomponents/customelements/> [Accessed 14 Apr. 2019].

- [21] Whatwg.org. (2019b). *HTML Standard*. [online] Available at: <https://html.spec.whatwg.org/multipage/custom-elements.html#valid-custom-element-name> [Accessed 14 Apr. 2019].
- [22] Webcomponents.org. (2019). *webcomponents.org - Discuss & share web components*. [online] Available at: <https://www.webcomponents.org/specs> [Accessed 14 Apr. 2019].
- [23] Ecma-international.org. (2015). *ECMAScript 2015 Language Specification – ECMA-262 6th Edition*. [online] Available at: <https://www.ecma-international.org/ecma-262/6.0/> [Accessed 14 Apr. 2019].
- [24] Webcomponents.org. (2019b). *webcomponents.org - Discuss & share web components*. [online] Available at: <https://www.webcomponents.org/polyfills/> [Accessed 14 Apr. 2019].
- [25] Polymer-project.org. (2018). *Introduction – LitElement*. [online] Available at: <https://lit-element.polymer-project.org/guide> [Accessed 16 Apr. 2019].
- [26] Polymer-project.org. (2018b). *Templates – LitElement*. [online] Available at: <https://lit-element.polymer-project.org/guide/templates#bind-properties-to-child-elements> [Accessed 19 Apr. 2019].
- [27] Google Developers. (2019a). *Progressive Web Apps | Web | Google Developers*. [online] Available at: <https://developers.google.com/web/progressive-web-apps/>.
- [28] Cockburn, A. (2016). *Writing Effective Use Cases*. Addison-Wesley, pp.124–125.
- [29] Heroku.com. (2019). *The Heroku CLI | Heroku Dev Center*. [online] Available at: <https://devcenter.heroku.com/articles/heroku-cli> [Accessed 20 Apr. 2019].