Bachelor’s thesis

Information and Communication Technology

2022 | 43

Krishna KC

DEVELOPING AND IMPLEMENTING WEB COMPONENTS

TURKU AMK

TURKU UNIVERSITY OF APPLIED SCIENCES

BACHELOR’S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Information and Communication Technology

2022| 43

Krishna KC

DEVELOPING AND IMPLEMENTING WEB COMPONENTS

Component-based frameworks and libraries have gained popularity over the years. Most of the web development projects are carried out using different tools and libraries. Integrating different frameworks and libraries in a single project has numerous advantages and benefits. All the component-based frameworks are developed from the concept of Web Components. The concept of Web Components makes it possible to break down and develop different parts of the web application to separate modules making each part reusable which in turn can be used within the same project or different projects seamlessly.

The main objectives of this thesis were to research different specifications of Web Components, implement them to create a dynamic reusable component without the help of libraries or frameworks, and, then reuse the created component in React JS which is the most popular framework of 2021.

This thesis summarizes the concepts and implementations of different technologies behind Web Component specifications, including Custom Elements, Shadow DOM, and HTML template. Similarly, the opportunities and challenges of Web Components are presented in general. Similarly, different easy-to-use and open-source tools needed are briefly presented. The component developed within the thesis provides an easy approach to defining a web component, its deployment, and reuse in applications based on React JS.

KEYWORDS:

HTML, CSS, JavaScript, Web Components, Custom Elements, DOM, Shadow DOM, HTML Template, Vanilla JavaScript, React JS, GitHub, GitHub Pages, MicroFrontend

CONTENTS

FIGURES

LIST OF ABBREVIATIONS

API	Application Programming Interface
CSS	Cascading Style Sheets
DOM	Document Object Model
ECMA	European Computer Manufacturers Association
HTML	Hypertext Markup Language
IDE	Integrated Development Environment
JS	JavaScript
MVC	Model-View-Controller
SEO	Search Engine Optimization
UI	User Interface
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
XML	Extensible Markup Language

1 INTRODUCTION

The growth of web development has increased the demand of web developers in 2020s. According to the Stackoverflow 2021 survey, JavaScript remains the most practiced language under the category Programming, scripting, and markup languages followed by HTML and CSS. [1] JavaScript and HTML along with CSS are based on the standards and specifications provided by Ecma International and W3C respectively.

Web Development is divided into two parts named frontend and backend Development. Frontend refers to the client-side programming that happens in the browser to which a user or client sees and interacts somehow. Whereas backend, known as server-side programming, occurs in the server and database that works behind to power the frontend features.[2]

Frontend developers use different primary web languages, libraries, extensions, and frameworks. All the languages have certain features and benefits in solving specific problems.[3] The primary language used and understood by browsers are:

- HTML
- CSS
- JavaScript
- jQuery

Libraries and frameworks provide powerful capacity and a straightforward approach to problem-solving compared to primary languages. Different libraries and frameworks have advantages, which can be easily implemented and used for utmost performance and user experience. Some of the popular libraries and frameworks used in web development are as follows:

- React library
- Vue.js
- Angular
- Svelte
- Ember

All frameworks and libraries are primarily component-based. These frameworks provide an easy learning curve, are scalable, and integrate better with backend services. Component-based frameworks are popular due to their straightforward approach over monolith or micro web services. Component-based development is aimed at designing and developing reusable components with encapsulated styles. Developers can deliver faster and better products with new features and experiences with open web technology and standards.[4]

Complications might arise when many people visit web pages, and the server has to continuously act upon requests from the clients or a large number of clients simultaneously. Due to large numbers of users and microservices, building software or products with component-based architecture has risen exponentially.[5] Some of the advantages of using component-based frameworks or systems are:

- **Robustness:** Components are isolated and help to minimize crashes of the whole system in different cases
- **Easy Maintenance:** Problems faced can be easily isolated and fixed without disturbing the rest of the codes
- **Flexibility:** Wide range of applications can be covered by reusing a single component.
- **Extensible:** New components can be created quickly, adding extra capabilities.

All the component-based frameworks or libraries are based upon the theory of web components. The thesis lays out detailed research about the concepts of web components, the creation of a web component without using frameworks or additional libraries and the integration of web components within an application based on a popular framework named React JS. Hence, a user card based on the concept of web components was developed that has a custom tag *<identity-card></identity-card>,* styles encapsulated in Shadow DOM, and template tag used for the markup of identity card. The thesis also illustrates the deployment of the created component and shows an easy approach to reusing the user card component in React JS.

The chapters in this thesis are divided into four main chapters.Chapter 1 provides an introduction to current frontend web development approach and provides an overview and advantages of component based frameworks.

Chapter 2 explains different specifications of Web Components including Custom Elements, Shadow DOM, and HTML Template as well as advantages and drawbacks of web components in current context when writing this thesis.

In Chapter 3, different tools and technologies that have been used as a part of implementing web components during this thesis are discussed. The selected tools are based on research of different technologies. The selected tools are open source and are easy to implement and light-weight in nature.

Similarly, Chapter 4 describes in detail how the tools described in Chapter 3, were used to define a web component, deploy and implement web component As a part of the implementation, a simple approach to define, deploy, and, implement a web component was laid out in Chapter 4.

2 WEB COMPONENTS

Web Components is a set of technologies that allows developers to create reusable custom elements with their functions encapsulated away from the rest of the code used in web apps.

Web Components help to maintain the reusability of fragments of codes. They help developers to manage source codes such as HTML, style sheets, and scripts easily without affecting the whole page. Web Components solve the problems related to slow application speed and code reuse, resulting in faster development and debugging for a product or service.

Web Components require three major base technologies, including Custom Elements, Shadow DOM, and HTML template. Each of these technologies is integrated to form a single web component that has a custom tag, scoped styles, and scripts limited to itself only. [6]

2.1 Custom Elements

A custom element is one of the basic specifications for developing Web Components. With the help of a custom element, developers can create new HTML tags, modify or add properties to existing titles, and extend the components made by other developers. A custom element is created using the base web technologies Vanilla JS, HTML, and CSS without external 3rd party libraries or frameworks.[7]

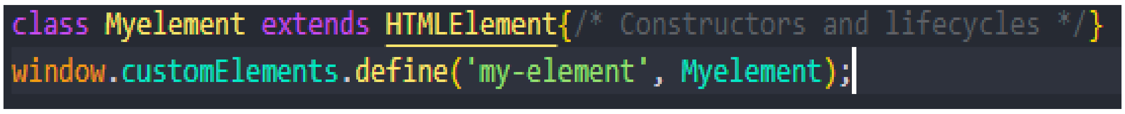
A JavaScript class defines a new element. A custom element can be extended to a basic HTMLElement or existing HTML tags such as buttons, paragraphs, etc. The binding of the class containing constructors and reactions with the tag name is achieved by using the *window.customElements.define* method. This method invokes and notifies the browser about a new tag. The method *window.customElements.define* has two parameters that contain the name of the HTML tag and the class extending the HTMLElement.[7]

There are specific rules about naming and exceptions that arise when creating a custom element. These rules are [8]:

- Tag names should not be common names but single words with a compulsion of the dash(-) so that the HTML parser can distinguish a custom element and regular elements. <x-bar>, <my-element> etc can be good tag names.
- The same tags cannot be registered more than once to prevent DOMException.
- Since there are only a few self-closing tags in HTML, a custom element cannot be a self-closing tag. For example: <my-element></my-element>.

The class contains constructors and lifecycle methods which are the logical parts of adding functionality to the element. There are specific rules to be followed inside constructors and reactions for a custom element which are as follows [9]:

- *super()* without any parameters must be called at the beginning to establish a strict prototype chain and *this* value**.**
- Only simple, early *return* or *return this* is allowed inside constructor body.
- DOM methods like *document.write()* or *document.open()* should be avoided inside constructor.



```
class Myelement extends HTMLElement{/* Constructors and lifecycles */}
window.customElements.define('my-element', Myelement);
```

Figure 1. Source code for creating a custom element.

Figure 1 shows the class 'Myelement' inheriting the basic HTML Element, which holds all the methods and behavior of a custom element, including constructors, lifecycle, and different attributes associated with a custom element. In the second line of the figure, a method for defining a custom element is used where, *my-element* is the name of the custom element, and *Myelement* is the reference to the class having methods and attributes to the custom element. The custom element can then be used as <my-element><my-element> just like a regular HTML tags.

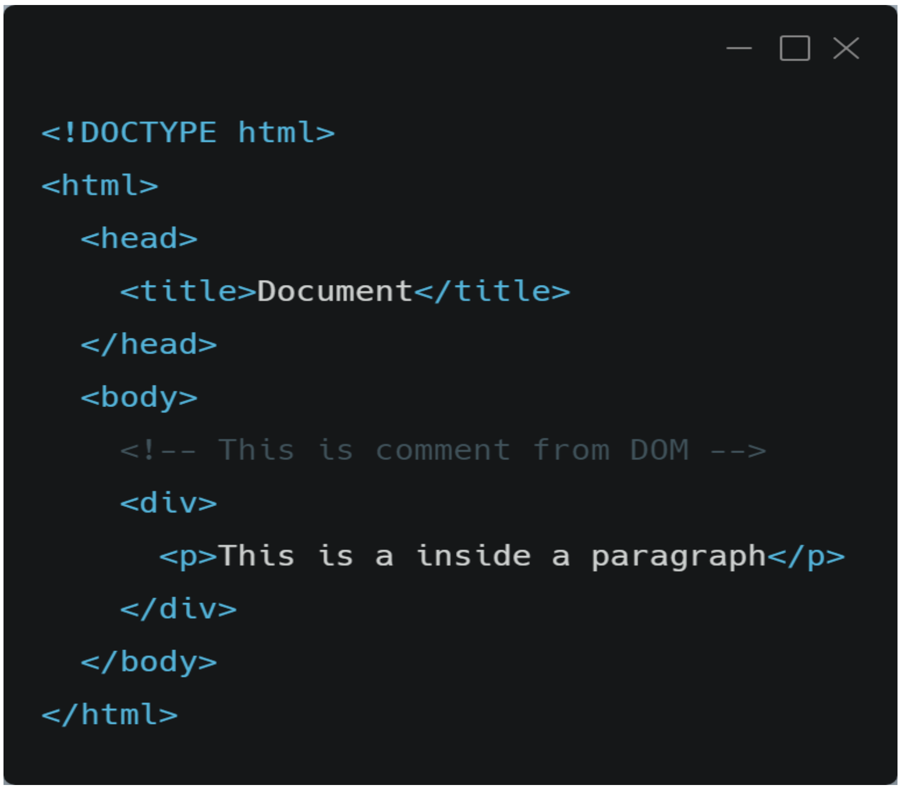
A custom element is fully supported in various browsers, including a new version of Chrome, Firefox, and Edge; however, Safari and other browsers have partial support for a custom element.

2.2 Shadow DOM

DOM

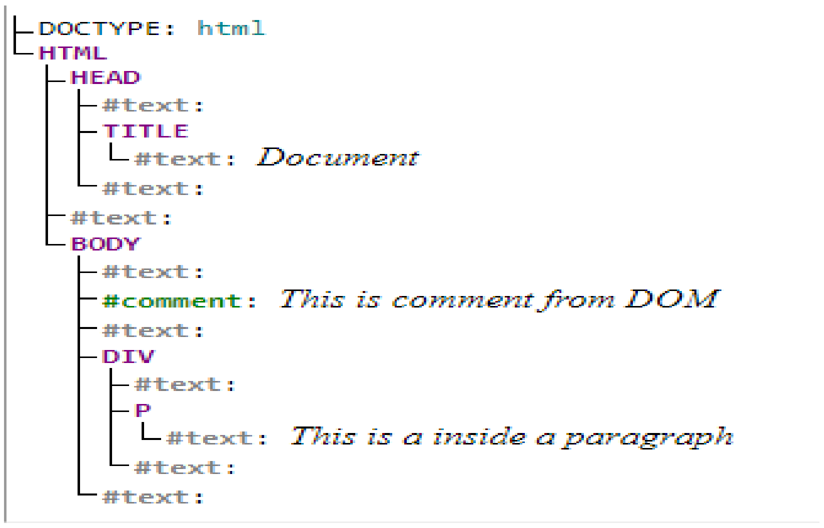
The DOM refers to an API that defines the logical structure of HTML or XML documents and methods to access and manipulate HTML elements. The DOM connects web pages and HTML documents to JavasScript or other programming languages. It is the representation of a webpage in a hierarchical way that benefits the programmers to go through the document for easy access and manipulation of tags, ids, classes, attributes, or elements using methods provided by the document object.[10]

The HTML source (Figure 2) can differ from the DOM structure since DOM represents the whole document in a browser containing all the document's contents. All the HTML documents are represented as a DOM tree in the browser (Figure 3). Everything in HTML source code has a place in DOM representation, including the comments. Nodes are then used for accessing, modifying, or manipulating certain parts of the documents. The structure of DOM is represented as a ”node tree” starting from the parent node. In the case of HTML documents, all the HTML tags, texts, and comments are denoted as element nodes, text nodes, and comment nodes respectively.[11]



```html
<!DOCTYPE html>
<html>
  <head>
    <title>Document</title>
  </head>
  <body>
    <!-- This is comment from DOM -->
    <div>
      <p>This is a inside a paragraph</p>
    </div>
  </body>
</html>
```

Figure 2. Simple HTML source code including a comment.



```
├DOCTYPE: html
└HTML
  ├HEAD
  │ ├#text:
  │ ├TITLE
  │ │ └#text: Document
  │ └#text:
  ├#text:
  └BODY
    ├#text:
    ├#comment: This is comment from DOM
    ├#text:
    ├DIV
    │ ├#text:
    │ ├P
    │ │ └#text: This is a inside a paragraph
    │ └#text:
    └#text:
```

Figure 3. HTML DOM tree.

Figure 3 shows the DOM tree structure of the HTML source code of Figure 2. HTML is the root node, *HEAD* and *BODY* is the child node of HTML, *TITLE*, *DIV*, *P* are element nodes and, text nodes are shown with a symbol '#' in the DOM tree.

Shadow DOM

Shadow DOM is a feature of DOM that allows a browser to include a subtree of DOM elements while rendering an HTML document (Figure 4). Shadow DOM is a document fragment attached to a host element but not in the main DOM. The host element can be any HTML element from where the shadow DOM starts and has its own scope.[12] The primary standards of shadow DOM are:

- **Shadow host**: It is the element or node that a shadow DOM is attached to.
- **Shadow tree**: It is a DOM tree of the shadow DOM where the shadow host is the root node.
- **Shadow boundary**: It is the end of the shadow tree from where the scope of shadow DOM ends and regular DOM continues.
- **Shadow root**: It is the root node of the shadow tree where shadow DOM is attached.

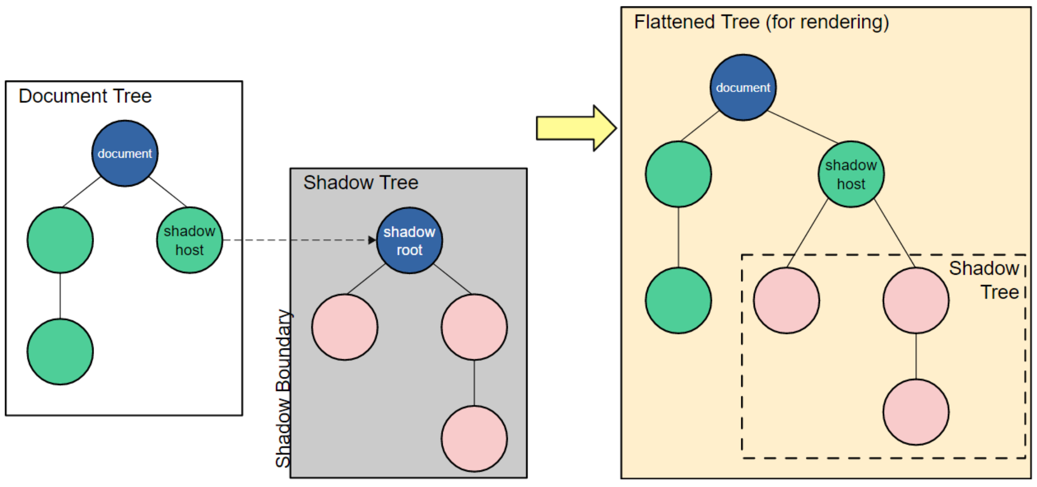


Figure 4. Shadow DOM in a DOM. [12]

In Fgure 4, the Shadow tree is attached to a shadow host from the main document tree, which has its own scope. The shadow host in document tree is shadow root. The flattened tree on the right side is used for rendering the whole DOM tree.

There are two main limitations of which elements can be used to create shadow root. [13]These include:

- Only one shadow root can be created for an element.
- The host of shadow tree can only be either custom element or elements such as *header, main, body, h1-h6 (heading tags), nav, p, section, div, span, footer, article, aside or blockquote.* In contrast, a shadow tree can not be attached to other elements such as *img*.

The growth of features in web application makes the applications to be more complex. The shadow DOM solves conflicts that arise between class names and IDs used by CSS hence providing easy solutions to developers to maintain and upgrade specific parts of the applications.

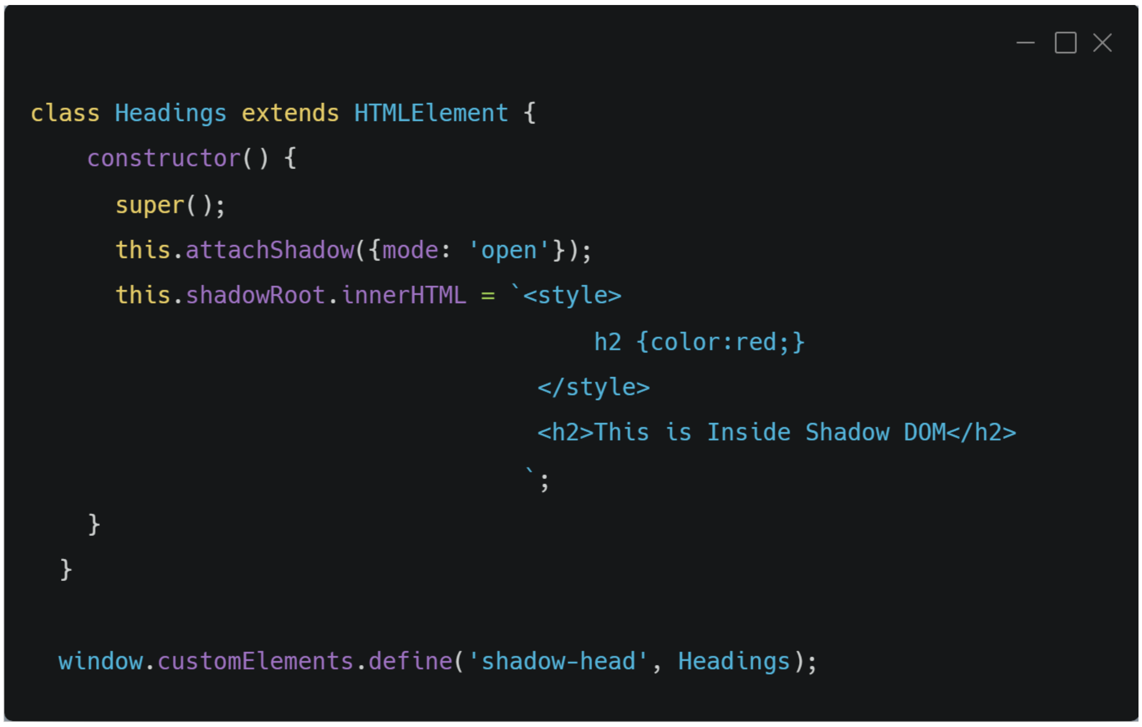
The shadow DOM helps developers create a child DOM inside the main DOM tree allowing them to build elements that are isolated from the rest of the codes. Some features of the elements developed using shadow DOM are as follows [14]:

- **Self Contained**: That cannot be accessed or manipulated from standard DOM methods and selectors like *document.querySelector().*

- **Scoped and simplified CSS**: CSS rules stated inside shadow DOM have limitations only inside shadow DOM, and the rules do not leak out, affecting other elements outside shadow DOM. In this way, simple selectors, ID, and classes can be used seamlessly.

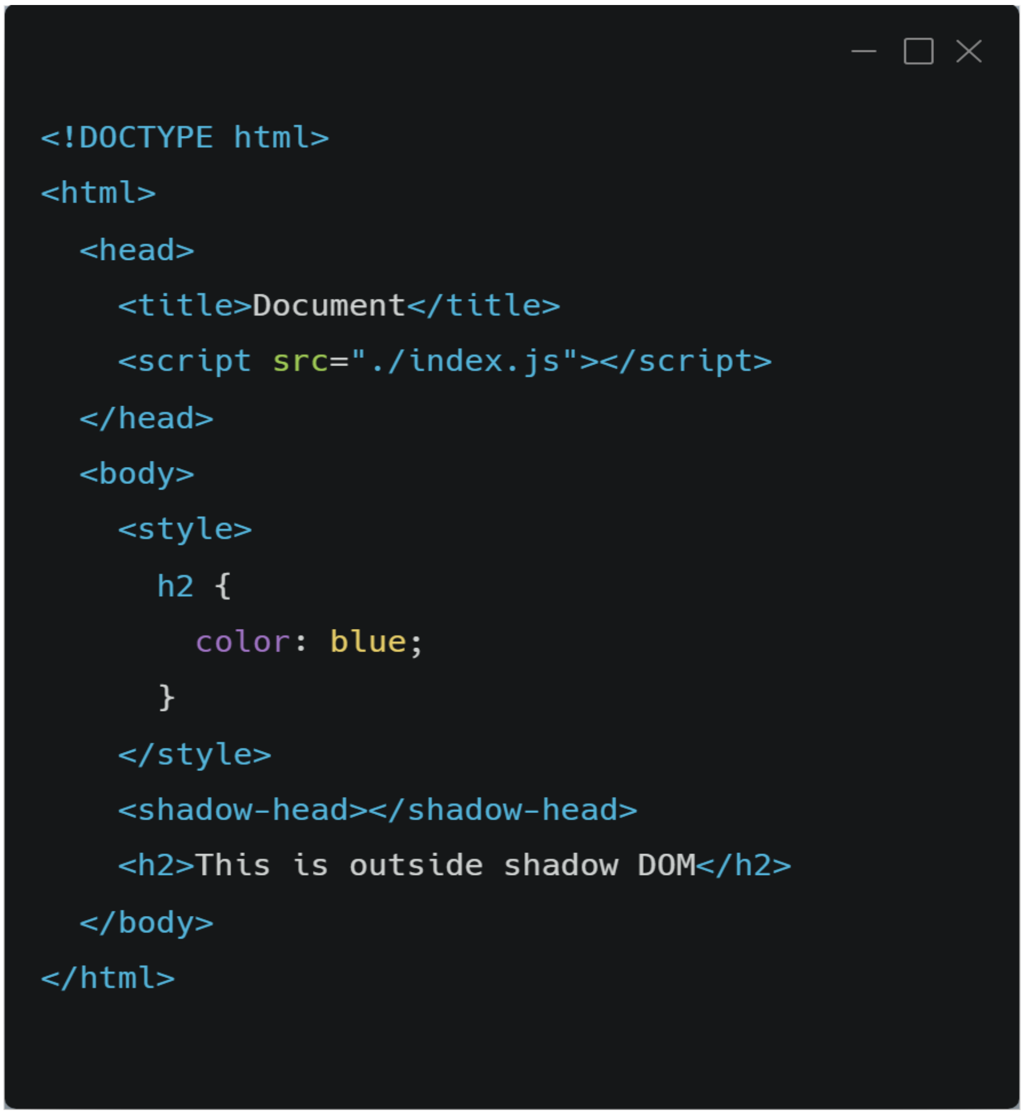
- **Productivity**: Shadow DOM increases productivity by allowing developers the ability to debug and maintain each element scoped inside the shadow DOM.

The method *attachShadow* is used for creating a shadow DOM to a custom element (Figure 5). The statement *this.attachShadow({mode:'open'})* helps bind the custom elements to include an isolated DOM tree (Figure 7). The mode open or closed is used, indicating if the shadow root can be modified outside the custom element or not respectively. The markup and styles that are attached to the shadow root are limited inside the custom element. The markup and styles can be added using regular DOM methods and they need to be attached to the *shadowRoot* (Figure 5).



```
class Headings extends HTMLElement {
    constructor() {
      super();
      this.attachShadow({mode: 'open'});
      this.shadowRoot.innerHTML = `<style>
                                          h2 {color:red;}
                                      </style>
                                      <h2>This is Inside Shadow DOM</h2>
                                     `;
    }
  }

  window.customElements.define('shadow-head', Headings);
```

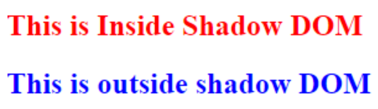
Figure 5. Shadow DOM in custom element.



```
<!DOCTYPE html>
<html>
  <head>
    <title>Document</title>
    <script src="./index.js"></script>
  </head>
  <body>
    <style>
      h2 {
        color: blue;
      }
    </style>
    <shadow-head></shadow-head>
    <h2>This is outside shadow DOM</h2>
  </body>
</html>
```

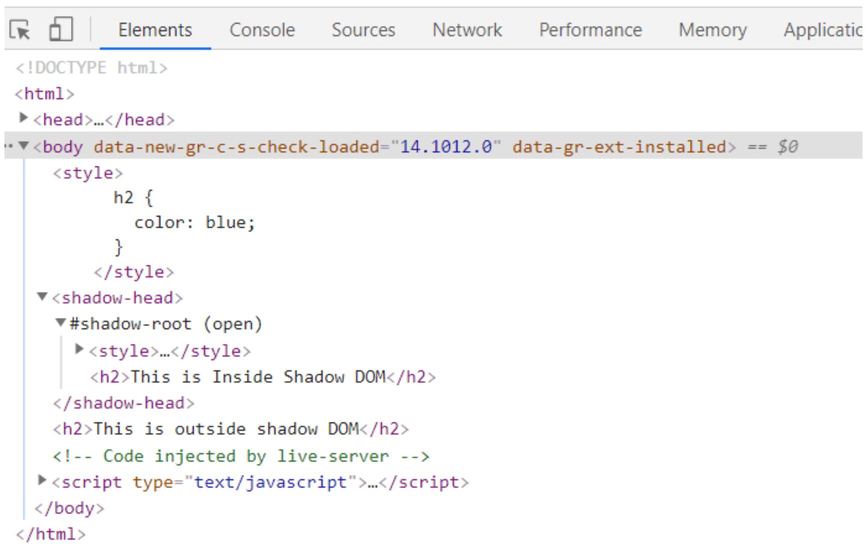
Figure 6. HTML including styles and Custom element.

In Figure 5, the shadow DOM is attached to a custom element named shadow-head using the method *attachShadow*. The DOM method *innerHTML* is used to add a markup and styles. <*shadow-head*></*shadow-head*> is defined as a custom element in a HTML source file as shown in Figure 6.



This is Inside Shadow DOM

This is outside shadow DOM



```
<!DOCTYPE html>
<html>
 ▶<head>…</head>
▼<body data-new-gr-c-s-check-loaded="14.1012.0" data-gr-ext-installed> == $0
    <style>
          h2 {
            color: blue;
          }
        </style>
  ▼<shadow-head>
    ▼#shadow-root (open)
      ▶<style>…</style>
       <h2>This is Inside Shadow DOM</h2>
    </shadow-head>
    <h2>This is outside shadow DOM</h2>
    <!-- Code injected by live-server -->
  ▶<script type="text/javascript">…</script>
  </body>
</html>
```

Figure 7. Shadow DOM in browser.

After the browser has rendered the codes as illustrated in Figures 5 and 6, the styles defined in HTML for *<h2>* do not override with the shadow root element *<h2>* having different styles. In Figure 7, a browser renders the HTML page with a custom element and developers tool shows the scope of shadow root.

The Shadow DOM is supported by all new browsers, including Firefox, Chrome, Opera, Safari, and the latest Edge browsers.

2.3 HTML Template

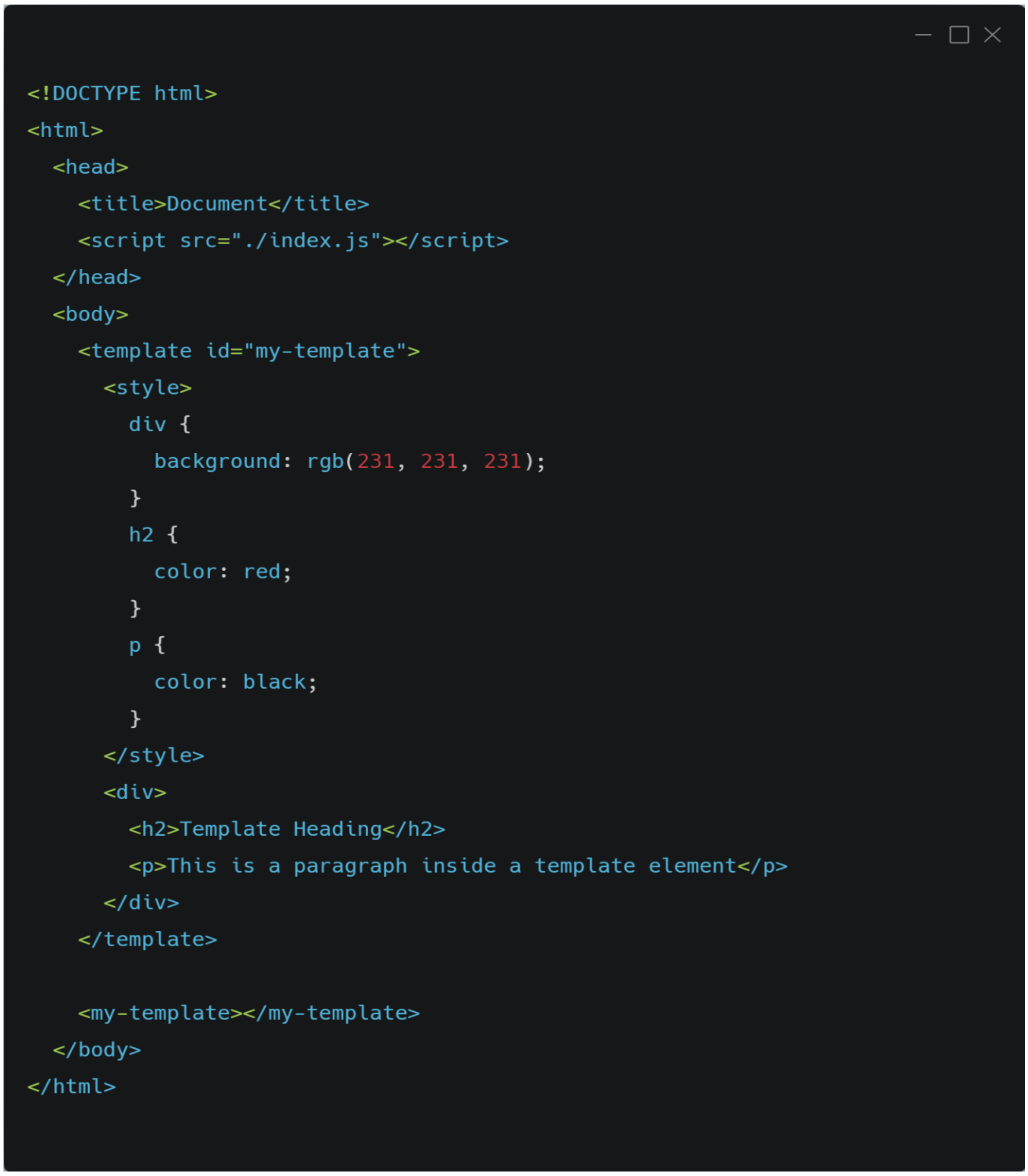
The HTML template is one of the powerful features of Web Components. It plays a crucial role in holding the markup structures for custom elements. The built-in HTML <*template*> element acts as a a storage for HTML markup.

The contents inside the template are ignored by the browser and are hidden when the page loads(Figure 9). The contents of the template are rendered using JavaScript (Figure 11). The HTML template makes it possible to define content or structure and save it for easy use across the webpage, instead of writing all HTML in theJavaScript source file (Figure 8).[15]

JavaScript is utilized for the HTML template. Web components work better when the template is used as the content of the shadow DOM. All the markup inside the template elements is wrapped inside a document-fragment node in a DOM tree (Figures 9 and 11). This prevents any other scripting or styles from overriding within the template content in a web page.

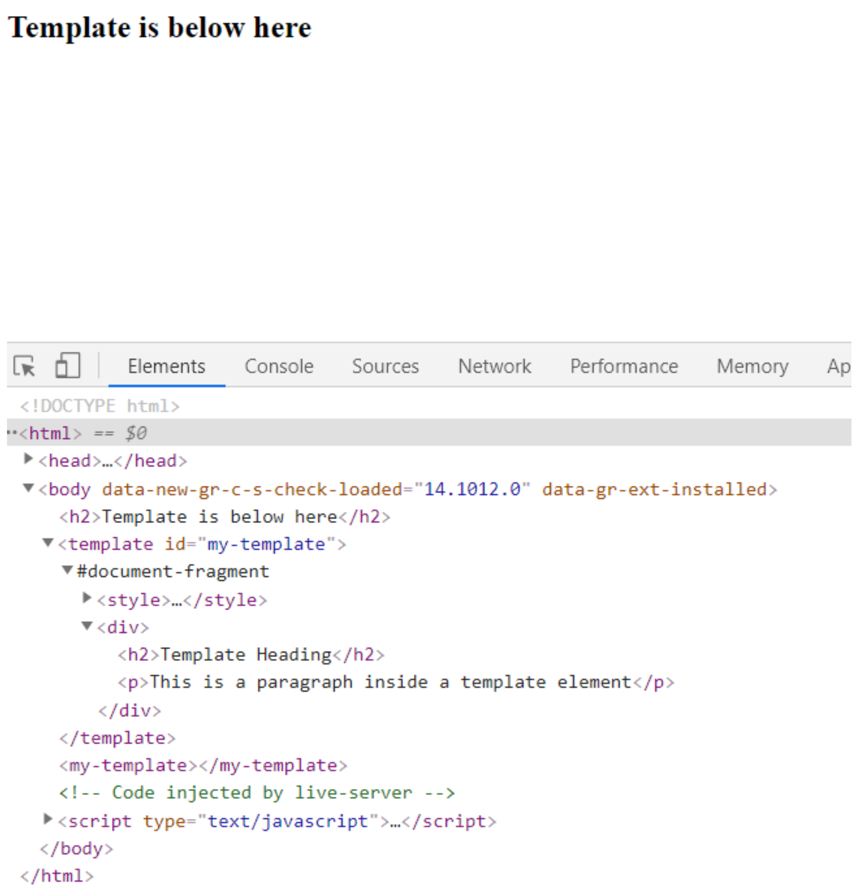
The main advantages of using the template are as follows[15]:

- The HTML template can hold any tags within it.
- The clone of the HTML template is used instead of the template itself when referenced and appended.
- Styles and Scripts can be inserted inside templates hence providing their own properties for the markup without overriding the other elements.
- The HTML template is supported in all modern web browsers except the Internet Explorer, which, however, can be solved by giving template style with *display: none* attribute.



```
<!DOCTYPE html>
<html>
  <head>
    <title>Document</title>
    <script src="./index.js"></script>
  </head>
  <body>
    <template id="my-template">
      <style>
        div {
          background: rgb(231, 231, 231);
        }
        h2 {
          color: red;
        }
        p {
          color: black;
        }
      </style>
      <div>
        <h2>Template Heading</h2>
        <p>This is a paragraph inside a template element</p>
      </div>
    </template>

    <my-template></my-template>
  </body>
</html>
```

Figure 8. Template element in HTML.

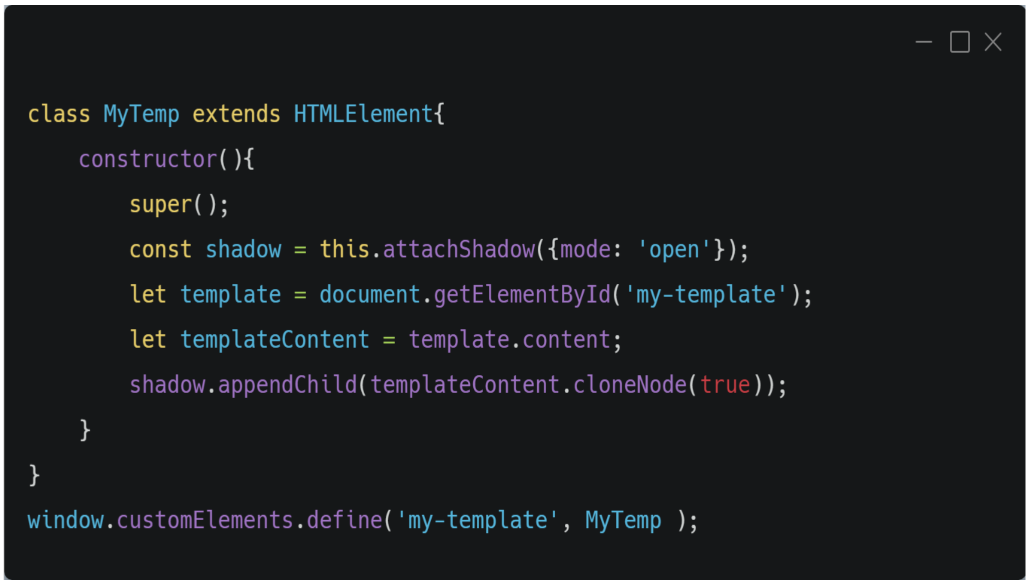


Template is below here

```
<!DOCTYPE html>
··<html> == $0
 ▶<head>…</head>
 ▼<body data-new-gr-c-s-check-loaded="14.1012.0" data-gr-ext-installed>
     <h2>Template is below here</h2>
   ▼<template id="my-template">
     ▼#document-fragment
       ▶<style>…</style>
       ▼<div>
           <h2>Template Heading</h2>
           <p>This is a paragraph inside a template element</p>
         </div>
     </template>
     <my-template></my-template>
     <!-- Code injected by live-server -->
   ▶<script type="text/javascript">…</script>
   </body>
</html>
```

Figure 9. Template before referenced in JavaScript.

Figure 8 shows an example of using template elements. In the developers tool of the browser, the template’s contents are wrapped inside a node called *#document-fragment* as shown in Figure 9.

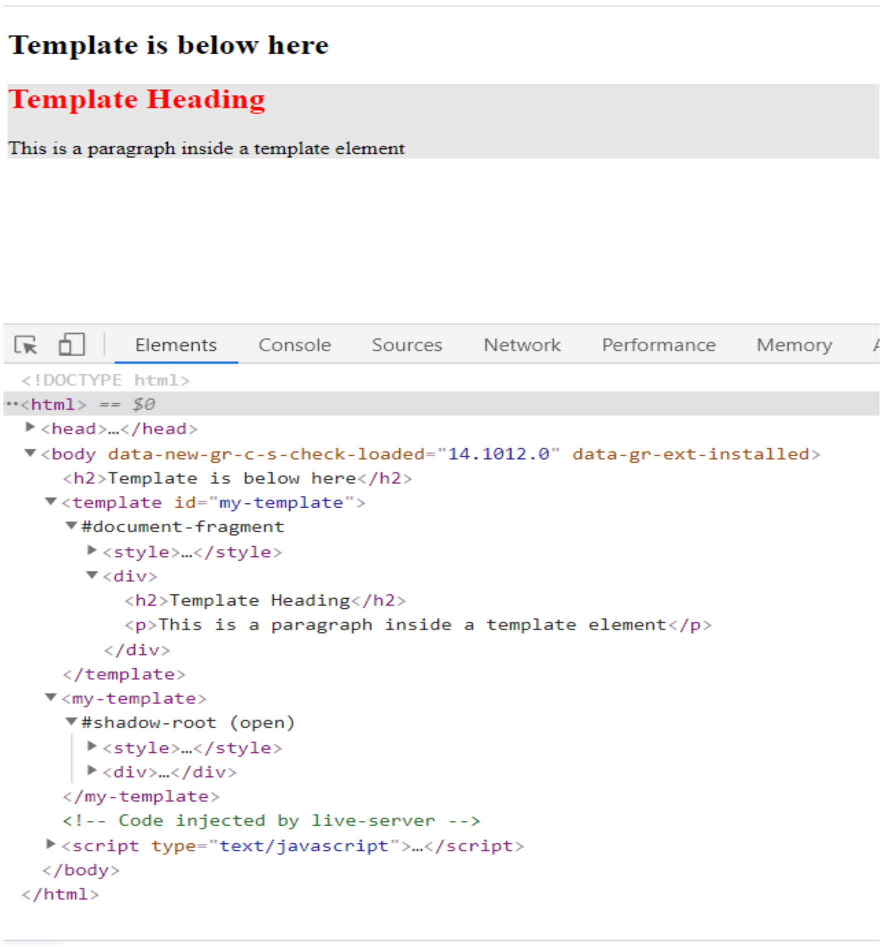


```
class MyTemp extends HTMLElement{
    constructor(){
        super();
        const shadow = this.attachShadow({mode: 'open'});
        let template = document.getElementById('my-template');
        let templateContent = template.content;
        shadow.appendChild(templateContent.cloneNode(true));
    }
}
window.customElements.define('my-template', MyTemp );
```

Figure 10. Appending the template to a custom element.

The template (Figure 10) is referenced by using the DOM method of *document.getElementById('my-template');*. The contents of the template are referenced using the method *template.content*. Finally, the template content is cloned and appended to the custom element with shadow DOM.

The contents of the template cloned as document-fragment are visible on the browser (Figure 11).



Template is below here

Template Heading

This is a paragraph inside a template element

```
<!DOCTYPE html>
··<html> == $0
  ▶<head>…</head>
  ▼<body data-new-gr-c-s-check-loaded="14.1012.0" data-gr-ext-installed>
     <h2>Template is below here</h2>
    ▼<template id="my-template">
      ▼#document-fragment
        ▶<style>…</style>
        ▼<div>
           <h2>Template Heading</h2>
           <p>This is a paragraph inside a template element</p>
         </div>
     </template>
    ▼<my-template>
      ▼#shadow-root (open)
        ▶<style>…</style>
        ▶<div>…</div>
     </my-template>
     <!-- Code injected by live-server -->
    ▶<script type="text/javascript">…</script>
   </body>
 </html>
```

Figure 11. Template after referenced in the custom element.

2.4 Advantages and Disadvantages of Web Components

The web applications are built using different technology stacks. The use of many stacks together to built a application could be challenging for a development teams. Web components make it possible for a developer to add extra functionalities within an application without the need to focus on different stacks [26].

Web components is used to create dynamic reusable components that can be used across different web applications and platforms with an easy approach. Web components is compatible to use with any JavaScript frameworks or no framework at all enabling developers having knowledge of different stacks to collaborate on the same project. Web Components is entirely based on Web Standard making them more futureproof than most of the frameworks[16].

Today, large companies have included Web components to build their features, including Firefox user interface and Apple Music Service [26].

2.4.1 Advantages

There are numerous benefits of using web components. The main advantage is its ability to bring different technology stacks like Angular, React, Vanilla JavaScript, Ember, and other frameworks and libraries together. Due to the varying benefits of one framework over another, better productivity can be gained using different stacks together.

Some of the benefits of Web Components are as follows [17]:

- **Micro Frontends Friendly**: Web Components works well with micro-frontend architecture which aims to provide possibility for multiple development teams associated with multiple frontend technologies and micro applications or services to work together.
- **Easy learning of Frameworks**: Since Web Components are based on JavaScript, learning Web Components makes a developer learn other JavaScript component-based frameworks easily.
- **Modular**: Web Components focuses on creating separate components resulting in a more accessible reusable design and reducing complexity.
- **Web Standard**: Web Components are based on Web Standards and W3C specifications, making them more standard and evolution in nature than other frameworks.
- **Framework support**: Web components can be used with almost all JavaScript frameworks and can be created with no framework, making the design more accessible and efficient among design teams.
- **Encapsulation**: Shadow DOM provides components with their DOM tree that cannot be accessed from the main document, making the styles and logic only apply to the specific element only.

2.4.2 Disadvantages

Web Components also have their opposing sides since they are based on evolving standards and have yet to overcome specific problems. Web components, in most cases, work very well; however, there are certain HTML elements that cannot be used with web components and are still to be developed for developers to rely on web components entirely.

Some of the challenges of Web Components faced by developers are as follows [18]:

- **Browser Compatibility**: The top browsers like Chrome, Safari, and Firefox fully support a web component built on the specifications. However, browsers like Safari or Edge still lack support for web components, including custom elements and shadow DOM. Due to this reason, Web components are not fully accessible from unsupported browsers.
- **Styling**: Since the styles of web components are encapsulated using Shadow DOM. When there is multiple web components, styles needed for each component should be added separately to ensure overall visual looks with other components.
- **Working with Forms**: Since web components is not a part of the regular DOM tree, Shadow DOM does not allow interaction with form elements like input.
- **Rendering to Server**: Rendering web components to a server-side requires extensible knowledge; however, other frameworks have simple solution to this problem.

3 TOOLS, FRAMEWORK, AND LIBRARY

3.1 Visual Studio Code

Visual Studio Code commonly known as VS Code is a free, lightweight, and open source IDE developed by Microsoft. According to Stack Overflow Developer Survey 2021 [27], more than 70 percentage of professional developers prefers VS code among different IDEs.

VS Code has built-in support for HTML, CSS, and JavaScript. Moreover, it has support for a wide range of programming languages. Despite being lightweight, VS code offers numerous benefits through its features like extensions, code linters, debugging tools as well as support for cloud and web development [28].

VS Code was used during the development of the user card component. Similarly, a live server was installed as an extension of Visual Studio Code that helps to create a local server for running HTML files.

3.2 Vanilla JS

Vanilla JS is a name that is used to refer a regular JavaScript. The use of inbuilt JavaScript methods and objects without the need of an external library or frameworks makes it one of the lightest programming languages. JavaScript is based on standards of ECMAScript language specifications. JavaScript is used to design and program the behavior of web pages during certain user interactions on the client-side [29].

Web Components that is created using vanilla JS are also known as native web components since they are built upon browser API without the need for additional frameworks or libraries. The main feature of using web components created using vanilla JS is that it does not require another library, compiler, runtime, or build tools[30].

Vanilla JS was used during the development of the user card component for every part including, custom elements, template, shadow DOM, and attributes related to the custom element.

3.3 GitHub and GitHub pages

GitHub is an online platform that provides services like tracking, storing and collaboration of source codes for developers on software development projects. GitHub is based on technology named Git that helps to track changes within files. GitHub is also used as social network within programmers that encourages developers to explore and contribute to different open source projects [31].

Developers and companies use GitHub for different purposes. They can store the code modifications and adapt and improve software from the repositories by managing possible conflicts from multiple developers. The version control in git helps track the complete history of the changes that occurred in the codes over time, assisting developers in identifying the problems and switching to the working version of source codes.[19]

GitHub pages provide hosting services for HTML, CSS, and JavaScript files. Web Components built from Vanilla JavaScript can easily be deployed through GitHub with the help of GitHub pages. GitHub pages can publish a website through an internal build process straight from the GitHub repository. Through GitHub pages, it is possible to deploy different sites named project, user, and organization. To publish a site, the repository that contains the source files of the project must be owned [20].

The deployment of a web component created in the thesis is achieved through different steps in order as follows [21]:

- A new or existing account to https://github.com/ is created.
- A new repository for the source files of the Web Component is made.
- Basic git commands are used to publish source files to the repository.
- Custom or private domain is selected for GitHub pages.

The page is published under the default domain provided by GitHub, i.e. *http(s)://<username>.github.io/<repository>*

3.4 React JS

React is a JavaScript library developed by a software engineer named Jordan Walke working at Facebook. React is an open-source component-based frontend library built upon Model View Controller (MVC) responsible for visual and interaction within the application. React works by dividing multiple UI into separate components. Each component has its property and function making it easy for developers to debug and write code efficiently [22].

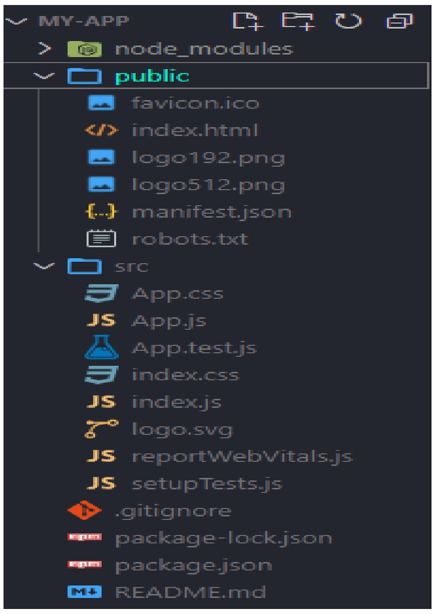
According to StackOverflow's 2021 surveys**,** React JS is the number 1 choice among the most popular web frameworks [23]. The following features of React have made its demand to be high among businesses and developers around the world [24]:

- **Simple and Easy implementation**: React is simple and easy to read and implement. This helps in learning and building the desired products required within less time, decreasing production costs.
- **Easy Maintenance**: React is based upon reusable components. React components can be easily used multiple times in a single app or various projects. Components ranging from small to significant wrapper components are beneficial during development, making it easier to maintain and reducing costs in the short and long term.
- **Dynamic and Robust**: React helps developers build interactive and dynamic applications for different platforms like web, Android, IOS, and IoT without affecting the quality of the application.
- **SEO friendly**: React focuses mainly on rendering speed by reducing page load times which is very beneficial for businesses to be in top ranks by various search engines like Google, Yahoo, Bing.
- **Easy Testing**: React allows developers to efficiently manage the output of different functions and events through localhost and helps to pinpoint the error making it easier to debug the code errors.

The application can be easily created in React JS. To be able to create a react app, Node JS must be installed on a local machine. Node js can be downloaded from the official website https://nodejs.org/en/. After the installation of Node js. React app can be simply created through the command line by moving to a directory where react application is created.

To create a react app (Figure 12), the commands used in the command line are as follows [25]:

- *npx create-react-app my-app*: npx is a package runner tool that is included in npm. Create-react-app is a pipeline to create a frontend react application. Whereas my-app is the folder and app name of the react app created.
- *cd my-app*: This command changes the directory to the react app created, which is needed to run the visual in localhost.
- *npm start*: It helps to begin the process for the development server and verifies packages, and starts the server at localhost.



```
MY-APP
  > node_modules
  v public
      favicon.ico
      index.html
      logo192.png
      logo512.png
      manifest.json
      robots.txt
  v src
      App.css
      App.js
      App.test.js
      index.css
      index.js
      logo.svg
      reportWebVitals.js
      setupTests.js
    .gitignore
    package-lock.json
    package.json
    README.md
```

Figure 12. Folder structure of React app.

Figure 12 shows the folder structure of a React application after creating it from the terminal. The public directory contains index.html and some image icons provided by default when the application is started. Similarly, the src folder contains source files for different JavaScript and CSS files, a crucial component of a react application.

4 DEFINING, DEPLOYING, AND IMPLEMENTING WEB COMPONENTS

This chapter explains the implementation of Web components by creating a simple user card component. The created component contains a template tag that holds the markup of the user card and the styles are encapsulated in shadow DOM. Similarly, JavaScript was used to dynamically render different values for different user cards.

In addition, the implementation describes how the created component can be deployed and implemented in a simple react project with the help of a simple tag that can be accessed through the internet.

4.1 Defining Web Component With Vanilla JS

User card component consists of *usercard.js* and *index.html* (Figure 14) within the folder *USERCARD* (Figure 13). The file *usercard.js* contains links and references of all the source codes for creation of the user card whereas *index.html* is used to view the created component and show the usage of a component in a simple HTML file.

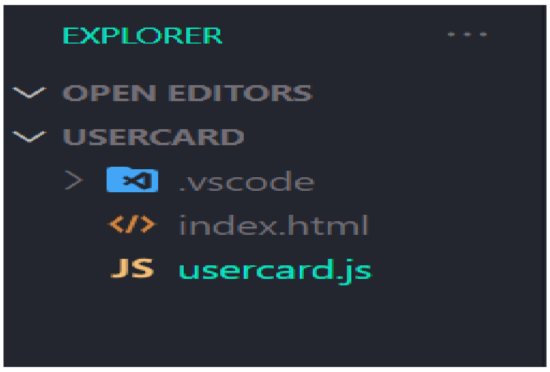
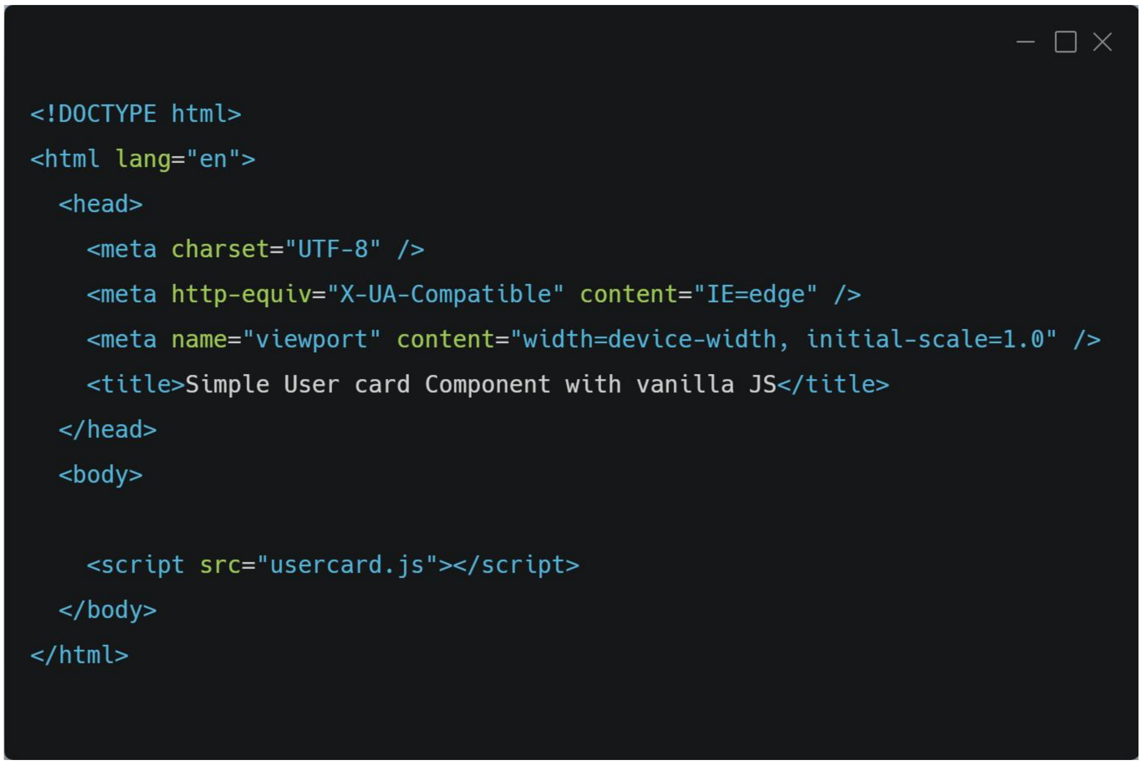


Figure 13. Folder structure of user card.



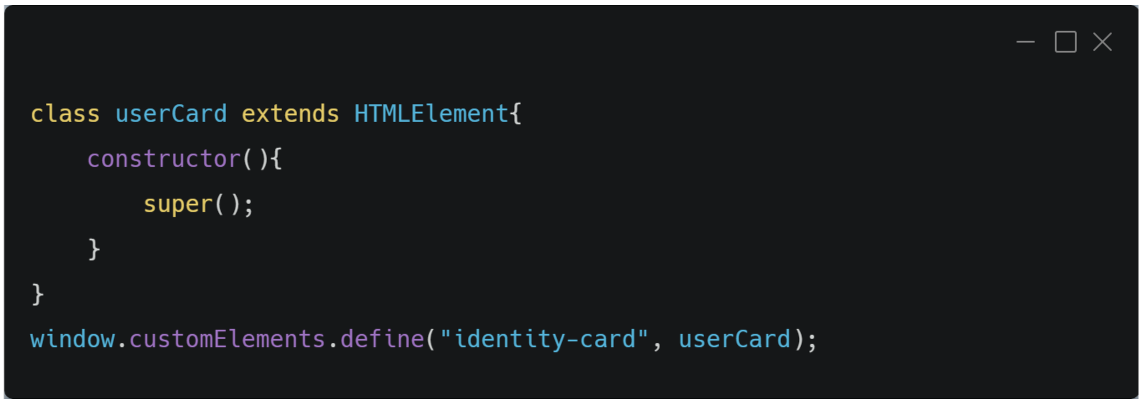
```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Simple User card Component with vanilla JS</title>
  </head>
  <body>

    <script src="usercard.js"></script>
  </body>
</html>
```

Figure 14. HTML markup with a source to usercard JavaScript file.

The contents of the *index.html* file includes Script tag with a reference to the JavaScript file (Figure 14).

A line *console.log(”I am connected”)* was added on a created usercard.js file to check if the script file is attached to the JavaScript file. The message inside console.log was viewed by opening the HTML file in the browser and checking the console tab in the chrome’s developer tool.

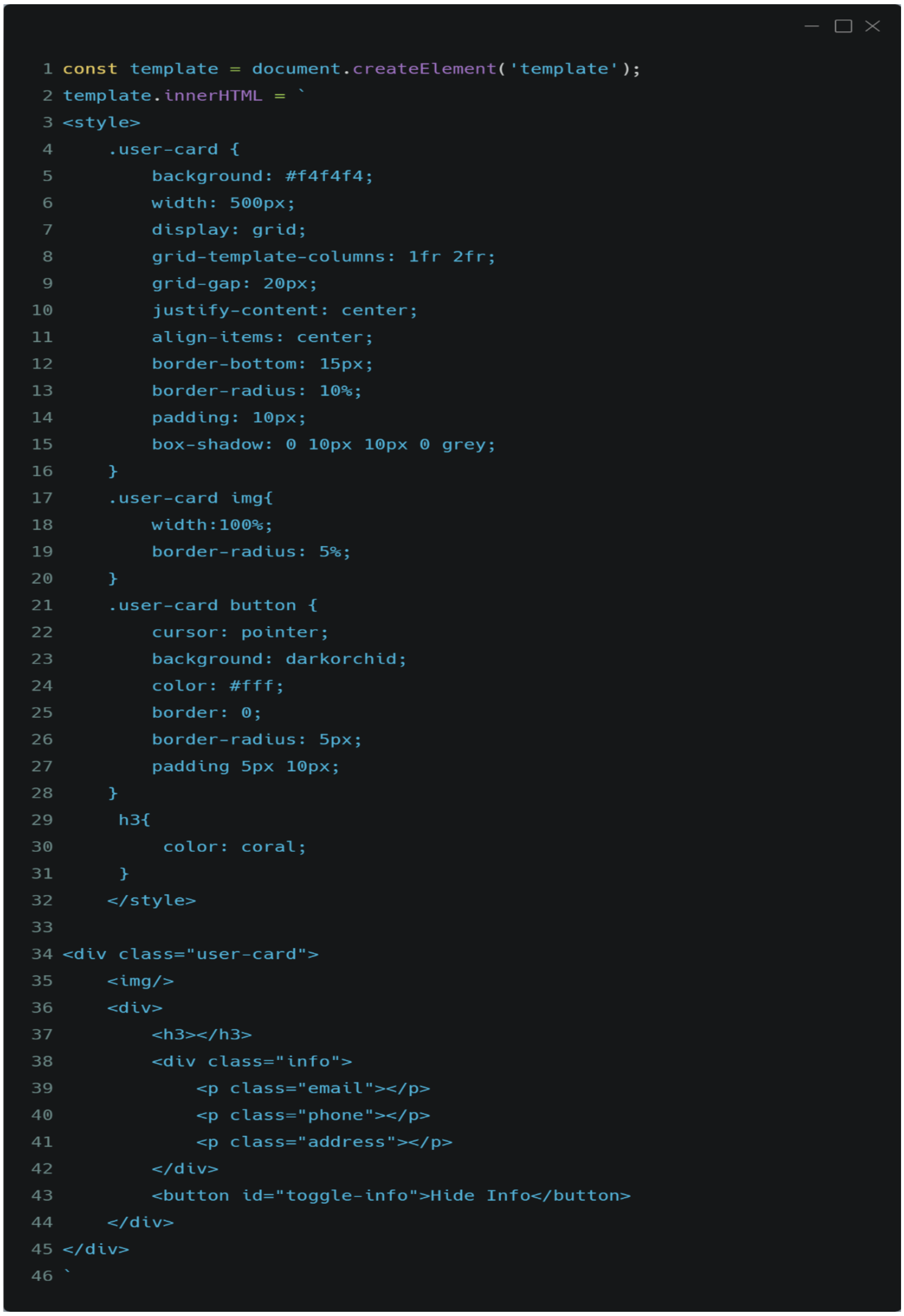
The custom element was defined and created using JavaScript class userCard that inherits a basic HTML element to hold the attributes for user card. Similarly custom element was registered and linked with the class with a tag name of *identity-card* (Figure 15).



```
class userCard extends HTMLElement{
    constructor(){
        super();
    }
}
window.customElements.define("identity-card", userCard);
```

Figure 15. Defining and registering the custom element.

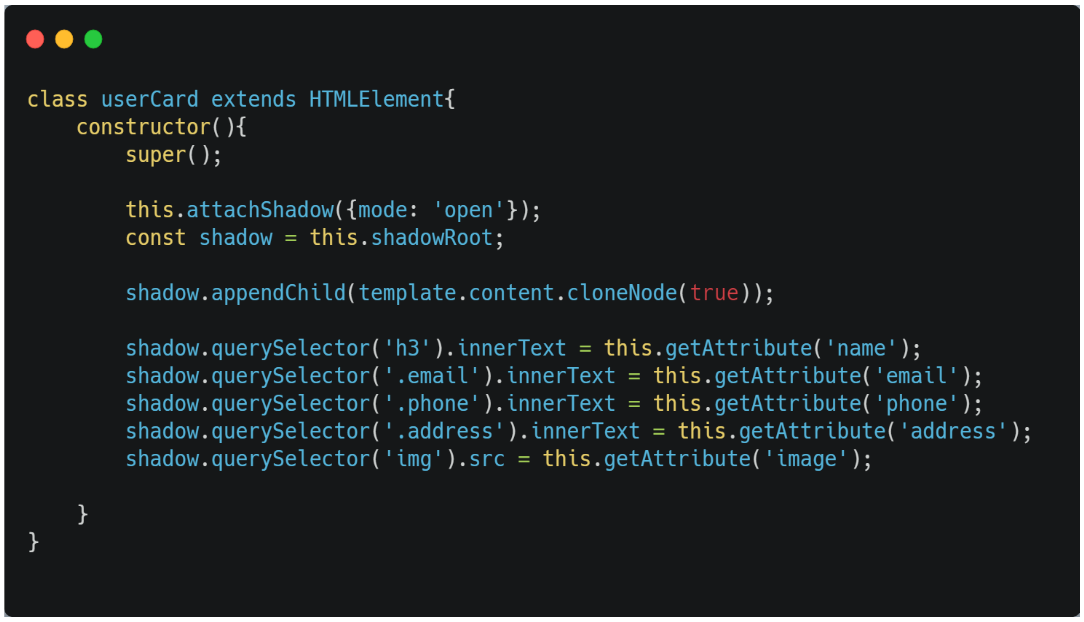
The template element (Figure 16) was created outside the UserCard class within the index.js file with the help of the DOM method named document.createElement('template'). The required CSS styles and HTML makup for user card was created within the template file (Figure 16)



```
const template = document.createElement('template');
template.innerHTML = `
<style>
    .user-card {
        background: #f4f4f4;
        width: 500px;
        display: grid;
        grid-template-columns: 1fr 2fr;
        grid-gap: 20px;
        justify-content: center;
        align-items: center;
        border-bottom: 15px;
        border-radius: 10%;
        padding: 10px;
        box-shadow: 0 10px 10px 0 grey;
    }
    .user-card img{
        width:100%;
        border-radius: 5%;
    }
    .user-card button {
        cursor: pointer;
        background: darkorchid;
        color: #fff;
        border: 0;
        border-radius: 5px;
        padding 5px 10px;
    }
     h3{
         color: coral;
     }
    </style>

<div class="user-card">
    <img/>
    <div>
        <h3></h3>
        <div class="info">
            <p class="email"></p>
            <p class="phone"></p>
            <p class="address"></p>
        </div>
        <button id="toggle-info">Hide Info</button>
    </div>
</div>
`
```

Figure 16. Template element with markup and styles for user card.

The statements *constshadow=this.*shadowRoot and *shadow.appendChild(template.content.cloneNode(true))* (Figure 17) are used inside the constructor of the component's class in order to use the component inside the shadow root of *identity-card* .

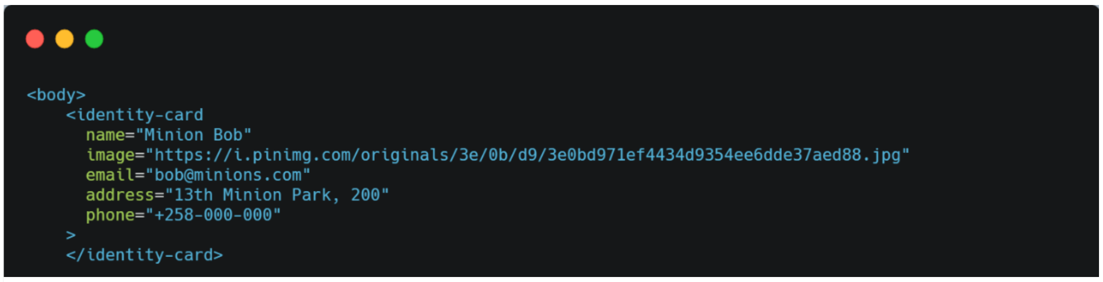


```
class userCard extends HTMLElement{
    constructor(){
        super();

        this.attachShadow({mode: 'open'});
        const shadow = this.shadowRoot;

        shadow.appendChild(template.content.cloneNode(true));

        shadow.querySelector('h3').innerText = this.getAttribute('name');
        shadow.querySelector('.email').innerText = this.getAttribute('email');
        shadow.querySelector('.phone').innerText = this.getAttribute('phone');
        shadow.querySelector('.address').innerText = this.getAttribute('address');
        shadow.querySelector('img').src = this.getAttribute('image');

    }
}
```

Figure 17. Defining attributes using query selectors.

After the template and its contents were cloned into the shadow root of the component, query selectors were used for referencing different fields like image, name, email, and phone. Similarly, attributes were used to dynamically allocate other parts of the contents of the user card (Figure 17).

The component created is finally tested using the tag (Figure 18) for custom element and attribute values within the *index.html* file.The custom element named *<identity-card></identity-card>* is loaded into the HTML file (Figure 18) just above the closing body tag.

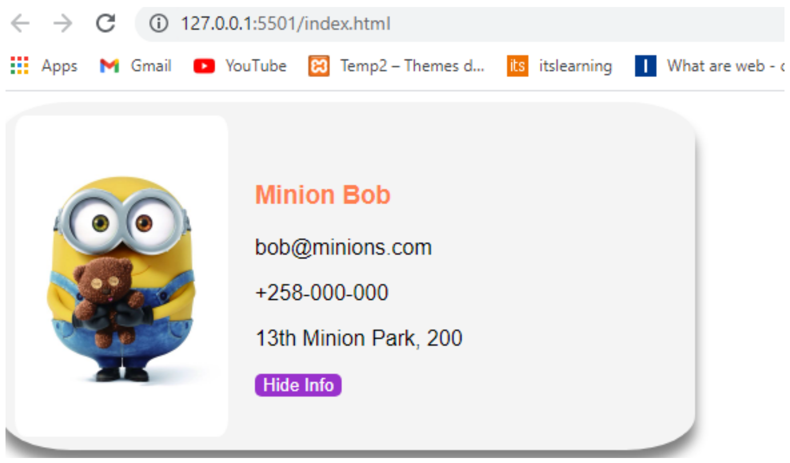


```
<body>
    <identity-card
      name="Minion Bob"
      image="https://i.pinimg.com/originals/3e/0b/d9/3e0bd971ef4434d9354ee6dde37aed88.jpg"
      email="bob@minions.com"
      address="13th Minion Park, 200"
      phone="+258-000-000"
    >
    </identity-card>
```

Figure 18. Implementation of web component inside body in HTML.

Similarly, inside the body, the *<identity-card></ identity-card>* is used with all the attributes defined in the script file.

Live server extension from VS Code was used to run index.html file in a localhost in a chrome browser (Figure 19). The final result of user card component along with its attributes was working smoothly on a browser (Figure 19).



Minion Bob

bob@minions.com

+258-000-000

13th Minion Park, 200

Figure 19. Identity card rendered by browser.

The complete web component for a user card was sucessfully defined by using VS Code and Vanilla JS. The resulted *<identity-card></ identity-card>* is a custom element that has CSS styles and markup within the HTML template tag that is attached into custom element's shadow DOM encapsulating styles within itself.

4.2 Deploying Web Components in Cloud

The repository for the source files was created after login into the GitHub account. Command-line was used to push the source files from the local machine to the Github repository. The directory in command line is changed to *D:\Thesis_web_components\usercard*, that contained the source files to the user card component. Then the following commands were used to establish a connection to the GitHub repository:

- **git init**: This basic git command was used to initialize the folder for tracking of changes.
- **Git remote add origin https://github.com/KKEEC/UserCardComponent**: It was used to establish a remote connection between the local machine and GitHub repository.

- **Git branch -M main**: The command was used to switch to the default branch named master.
- **Git push -u origin main**: This command was used to push the files from the local drive to the GitHub Repository.

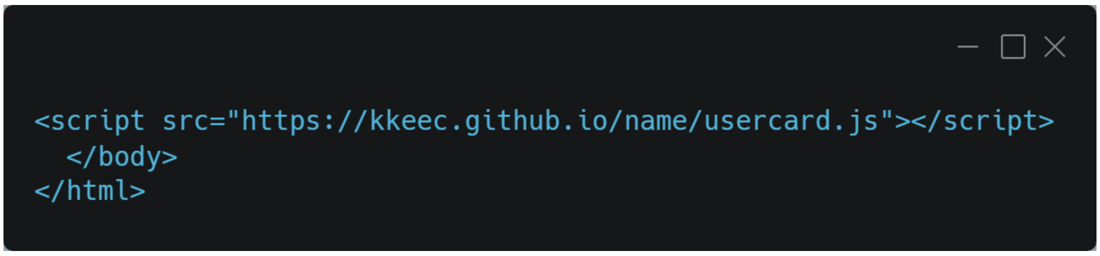
After the last command, the source files appear on the Github repository when opened from the web browser.

The repository was then deployed on a GitHub page after directing to the settings of the repository. The site for a custom element was easily deployed through the pages section under settings. After, the component was deployed, and the site was published with a default address of https://kkeec.github.io/UserCardComponent/.

4.3 Implementing Web Components within React JS application

When using a user card component on a react app, *index.html* located on public folder within react application as well as *App.js* located on src folder had to be used.

After the sources are deployed within GitHub, the main source file for *identity-card* named as *usercard.js* is sourced by providing simple url path (Figure 20). The source file *usercard.js* is accessed through a link *https://kkeec.github.io/UserCardComponent/usercard.js*. The latter part,*usercard.js* within the link, explicitly provides the path to *usercard.js* file.



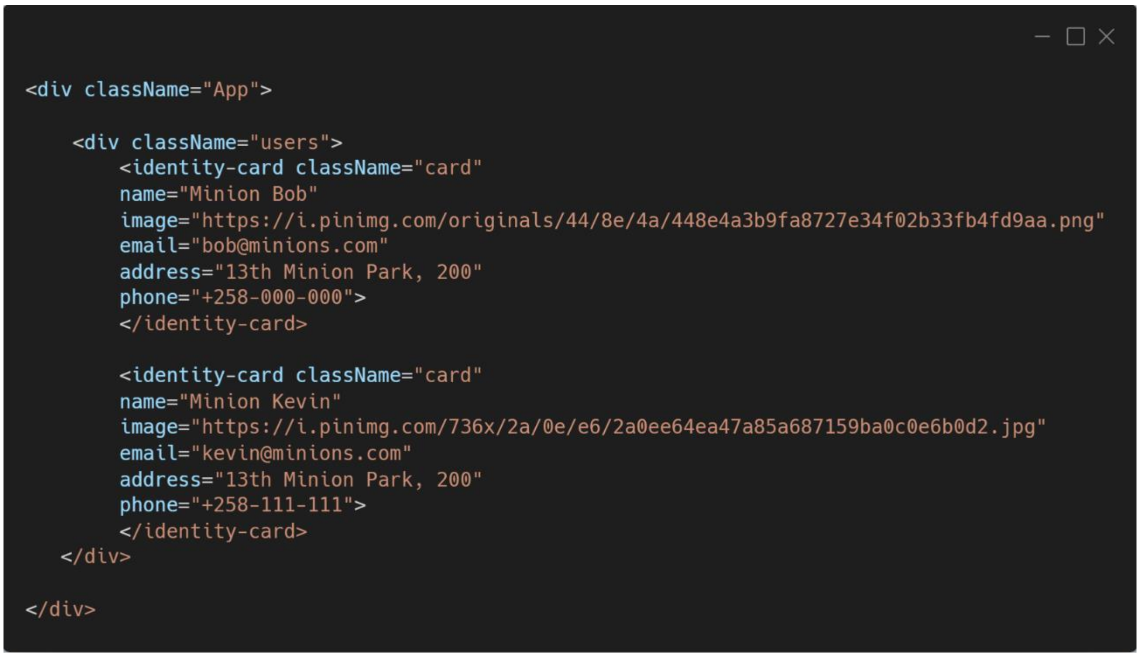
```
<script src="https://kkeec.github.io/name/usercard.js"></script>
  </body>
</html>
```

Figure 20. Script tag containing source link for identity card component.

The script tag with a source referencing to the url of GitHub page was added to *index.html* file within the react application (Figure 20).

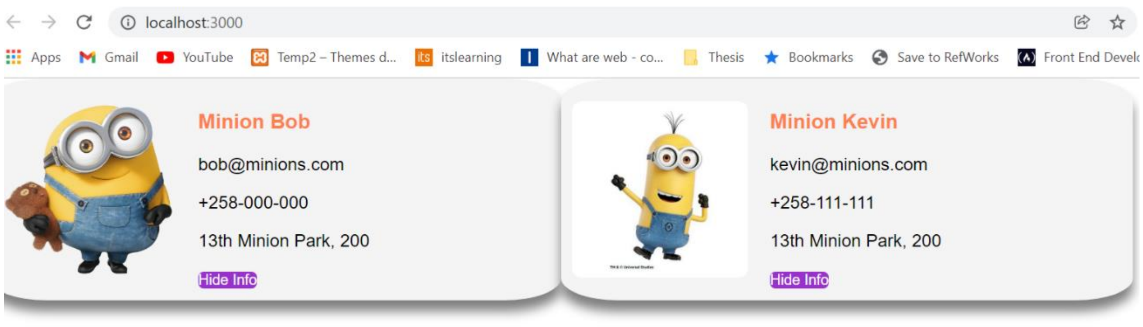
After the source has been provided to index.html in a react application, The web component was used in *App.js* file within the react application (Figure 21).

The two instances of *identity-card* were used with different attribute values for other persons within the react app (Figure 21) to check the user card to work with multiple values for attributes (Figure 22). Figure 22 displays two different user cards having different attribute values rendered by browser.



```
<div className="App">

    <div className="users">
        <identity-card className="card"
        name="Minion Bob"
        image="https://i.pinimg.com/originals/44/8e/4a/448e4a3b9fa8727e34f02b33fb4fd9aa.png"
        email="bob@minions.com"
        address="13th Minion Park, 200"
        phone="+258-000-000">
        </identity-card>

        <identity-card className="card"
        name="Minion Kevin"
        image="https://i.pinimg.com/736x/2a/0e/e6/2a0ee64ea47a85a687159ba0c0e6b0d2.jpg"
        email="kevin@minions.com"
        address="13th Minion Park, 200"
        phone="+258-111-111">
        </identity-card>
   </div>

</div>
```

Figure 21. Consuming idenity card component within React application.



Minion Bob

bob@minions.com

+258-000-000

13th Minion Park, 200

Minion Kevin

kevin@minions.com

+258-111-111

13th Minion Park, 200

Figure 22. Browser rendering the user card within react application.

5 CONCLUSION

The critical goals of the thesis were to study Web Components as well as the advantages and disadvantages of using Web Components. Similarly, tools and libraries were researched so that it was possibile to create web components with minimal configurations and common tools. The implementation part of the thesis provided a deeper insight to defining a web component using Vanilla JS without additional frameworks or libraries. The implementation part also helped better understanding of the deployment process of defined components within GitHub and how the deployed components were used within react application without additional dependencies.

The created user card component did not include advanced features but was dynamic considering the nature of component. Using different attributes and values made the component so that it be used in various projects and other frameworks as a part of the web page. The approach of the web development making modular pieces of components helped in better understanding and usage of web components. In summary, the implementation of Web Components helped to achieve the goals of the thesis.

Web Components is a modern approach to web development that is simple, fast, and reusable. This provides developers to easily maintain the codes of specific parts of applications with a more easy approach. Due to the nature of Web Components, different giant tech companies have started to use Web Components or component-based libraries in some way.

Web Components is slowly being standardized and supported across different platforms and browsers. Similarly, the powerful features of Web Components to work with other JavaScript-based frameworks and libraries enables people with different skills to collaborate on a project easily. Hence, It can be concluded that Web Components might evolve to be an integral part of web development in the future.

REFERENCES

[1] Stack Overflow. 2021. Stack Overflow Developer Survey 2021. [online] Available at: https://insights.stackoverflow.com/survey/2021#section-most-popular-technologies-programming-scripting-and-markup-languages [Accessed 10 May 2021].

[2] LAB, S.T.I. -12-31T04:41:28.822Z, 2019-last update, The Fundamentals of Front End and Back End Development. Available: https://sagaratechnology.medium.com/the-fundamentals-of-front-end-and-back-end-development-5973ac0910cf [May 25, 2021].

[3] TERRA, J. -08-07T09:28:03+05:30, 2019-last update, How to Become a Front End Developer - Skills, Roles, Salary Explained. Available: https://www.simplilearn.com/how-to-become-a-front-end-developer-article [May 25, 2021].

[4] HART, B., , What Is Component Based Development?. Available: https://www.perforce.com/blog/vcs/component-based-development [May 26, 2021].

[5] LAROCCA, K., -10-29T21:51:25.670Z, 2020-last update, 5 Benefits Of Component-Based Development. Available: https://medium.com/newyorkpublicradiodigital/5-benefits-of-component-based-development-90af513bb7d2 [June 1, 2021].

[6] SINGH, S.- Traditional vs modern web development ⚔. Available: https://dev.to/sunnysingh/traditional-vs-modern-web-development-1em8 [June 13, 2021].

[7] ADERINOKUN, I., 2018. What, exactly, is the DOM? *bitsofcode,* .BIDELMAN, E., , Custom Elements v1: Reusable Web Components | Web Fundamentals. Available: https://developers.google.com/web/fundamentals/web-components/customelements [May 17, 2021].

[8] Whatwg.org. (2019b). HTML Standard. [online] Available at: https://html.spec.whatwg.org/multipage/custom-elements.html#valid-custom-elementname [June 25. 2021].

[9] Html.spec.whatwg.org. 2021. HTML Standard. [online] Available at: https://html.spec.whatwg.org/multipage/custom-elements.html#custom-element-conformance [July 10, 2021].

[10] Developer.mozilla.org. 2021. Document Object Model (DOM) - Web APIs | MDN. [online] Available at: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model [July 25, 2021].

[11] MUCHIRI, A. Understanding the DOM Tree and Nodes. Available: https://www.alibabacloud.com/blog/understanding-the-dom-tree-and-nodes_596231 [July 28, 2021].

[12] Using Shadow DOM. Available: https://developer.mozilla.org/en-US/docs/Web/Web_Components/Using_shadow_DOM [September 5, 2021].

[13] Shadow tree. Shadow DOM. Available: https://javascript.info/shadow-dom [September 6, 2021].

[14] BIDELMAN, E., 2021. Shadow DOM v1: Self-Contained Web Components | Web Fundamentals. [online] Google Developers. Available at: https://developers.google.com/web/fundamentals/web-components/shadowdom [September 15, 2021].

[15] BIDELMAN, E., HTML's New Template Tag: standardizing client-side templating. Available: https://www.html5rocks.com/en/tutorials/webcomponents/template/ September 18, 2021].

[16] KAMBOJ, S. -02-07T09:15:54.169Z, 2020-last update, Web Components Basics and Performance Benefits. Available: https://medium.com/@spkamboj/web-components-basics-and-performance-benefits-f7537c908075 [October 5, 2021].

[17] PATEL, S. 2015. Learning web component development. Birmingham, UK: Packt Publishing, p.26.

[18] REVILL, L. -05-03T17:34:57.068Z, 2017-last update, Web Component Challenges. Available: https://blog.revillweb.com/web-component-challenges-a09ebc598d65 [Nov 1, 2021].

[19] BRADFORD, L.. What Is GitHub?. Available: https://www.thebalancecareers.com/what-is-github-and-why-should-i-use-it-2071946 [Nov 3, 2021].

[20] GitHub Docs. 2021. About GitHub Pages - GitHub Docs. [online] Available at: <https://docs.github.com/en/pages/getting-started-with-github-pages/about-github-pages> [Accessed 5 November 2021].

[21] CAMACHO, E. -05-05T23:54:33.426Z, 2018-last update, Creating and deploying a static website using Github Pages. Available: https://medium.com/@erickcodes/creating-and-deploying-a-static-website-using-github-pages-a634a588ed7d [Nov 10, 2021].

[22] PANDIT, N. What And Why React.js. Available: https://www.c-sharpcorner.com/article/what-and-why-reactjs/ [Nov 12, 2021].

[23] Stack Overflow. 2021. Stack Overflow Developer Survey 2021. [online] Available at: <https://insights.stackoverflow.com/survey/2021#section-most-popular-technologies-web-frameworks> [Accessed 12 November 2021].

[24] Deshpande C.What is React: Definition, Why ReactJS, its Features & Installation. Available: https://www.simplilearn.com/tutorials/reactjs-tutorial/what-is-reactjs [Nov 14, 2021].

[25] React. Create a New React App . Available: https://reactjs.org/docs/create-a-new-react-app.html [Nov 25, 2021].

[26] Viljami S. Design 2021. [online] Viljamis.com. Available at: https://viljamis.com/2019/why-we-use-web-components/ [Accessed 7 December 2021].

[27] Stack Overflow. 2021. *Stack Overflow Developer Survey 2021*. [online] Available at: <https://insights.stackoverflow.com/survey/2021#section-most-popular-technologies-integrated-development-environment> [Accessed 15 December 2021].

[28] MUSTAFEEZ Z. A. , What is Visual Studio Code?. Available: https://www.educative.io/edpresso/what-is-visual-studio-code [Dec 15, 2021].

[29] Developer.mozilla.org. 2021. JavaScript | MDN. [online] Available at: <https://developer.mozilla.org/en-US/docs/Web/JavaScript> [Accessed 17 December 2021].

[30] BOGAN, M. 2021. Web Component Solutions: A Comparison - DZone Web Dev. [online] dzone.com. Available at: https://dzone.com/articles/web-component-solutions-a-comparison [Accessed 18 December 2021].

[31] JUVILER, J. 2021. What Is GitHub? (And What Is It Used For?). [online] Blog.hubspot.com. Available at: <https://blog.hubspot.com/website/what-is-github-used-for> [Accessed 19 December 2021].