Available online www.jsaer.com

***Journal of Scientific and Engineering Research*, 2020, 7(12):250-257**

Research Article

ISSN: 2394-2630
CODEN(USA): JSERBR

Lightning Web Components: A Modern Programming Model for the Lightning Platform

Raja Patnaik

Email: raja.patnaik@gmail.com

Abstract This paper explores the adoption and evolution of Lightning Web Components (LWC), introduced by Salesforce in 2018, to leverage modern web standards and JavaScript for building efficient applications on the Lightning Platform. The paper discusses LWC's technical architecture, integrating MVC principles and Shadow DOM for enhanced UI separation, style encapsulation, and component modularity. It details the development lifecycle of LWCs, including debugging and testing frameworks that ensure component robustness and performance. Additionally, real-world application cases demonstrate LWC's significant impact on streamlining development processes, enhancing application scalability, and improving maintainability. The paper underscores the transformative role of LWC in advancing web development within Salesforce's ecosystem.[5]

Keywords LWC, Salesforce, low-code tool, programming tool

1. Introduction

Salesforce introduced a standard-based Javascript model in the Lightning Web Component, enabling millions of developers to build apps on the Lightning Platform using familiar tools on December 13, 2018 [2]. LWC is standard-based, high-performing, and user-friendly, leveraging modern JavaScript features like ES6+, allowing faster performance and better end-user experiences. The interoperability and the flexibility of easy integration with other tools enhance the developer service and drive innovation. The core benefits of LWC are as follows [1]:

- **Standardized for Enhanced Productivity:** Developers can use Javascript features like classes, modules, and imports while developing in LWC, which uses the modern web language with support for ES6+.
- **Faster Performance:** Most of the LWC components are run by the browser locally, resulting in faster execution.
- **Compatible with other LWC Components:** LWC is compatible with Lightning Components like Aura (An existing programming model launched by Salesforce in 2015).
- **Drive Low-Code Development:** LWC, combined with other Salesforce low-code tools like Lightning App Builder and Lightning Flow, drives low-code development with clicks and simple drag and drop.
- **Limitless Possibilities:** LWC, combined with Salesforce pro-code tools like Salesforce DX, Enterprise services like Salesforce Einstein or Salesforce IoT, and the other low-code modules, allows developers to build large-scale customer experiences and extend the functionality of the CRM.

Developers can rest assured that aura components and Lightning web components not only can coexist but also seamlessly interoperate, sharing the same high-level services:

Aura components and Lightning web components can coexist on the same page. Aura components can include Lightning web components, and both share the same base Lightning components, which have already been

implemented as Lightning web components. Additionally, Aura and Lightning web components share the same underlying services, such as Lightning Data Service and User Interface API.

Figure 1: Web Stack Transformation [9]

2. Architecture and Functionalities

A. Technical Architecture

The Salesforce Lightning Web Components (LWC) architecture, a contemporary innovation in web development within the Salesforce ecosystem, presents a nuanced adaptation of the traditional Model-View-Controller (MVC) framework. This architectural divergence is primarily manifested in its dual-controller model, which enables a more dynamic interaction between the client and server components [3]. Lightning Web Components represents Salesforce's implementation of lightweight frameworks based on web standards. It makes use of custom elements, templates, shadow DOM, decorators, modules, and various new language constructs available in ECMAScript 7 and beyond.

- The model is integrally associated with Salesforce Objects such as Account, Contact, Employee, etc., illustrating LWC's data-centric aspect.
- The view is represented through HTML and CSS components, facilitating user interface design and styling.
- The controller, a pivotal component of the LWC architecture, is divided into a client-side controller, implemented through JavaScript, and a server-side controller, implemented through APEX. This distinction enhances the framework's ability to handle complex business logic and data manipulation tasks efficiently.
- The introduction of Shadow DOM in LWC significantly changed the framework's architecture by providing encapsulation and isolation of components. Shadow DOM is a web standard that ensures a component's internal workings, styles, and behaviors are hidden and unaffected by external styles or scripts, leading to-

Isolated Styling: Styles defined within a component don't bleed out, and external global styles don't affect the component.

Scoped JavaScript: The behavior inside the component does not affect the global scope, ensuring component-specific functionalities remain isolated.

Leveraging Shadow DOM, Salesforce's Lightning Web Components (LWC) offer enhanced component reuse, maintainability, and performance. This encapsulation technique allows for isolated component styling, creating a more consistent user interface. Additionally, it streamlines event propagation and CSS isolation, leading to quicker event handling and preserving component design integrity and application security [4].

From a development perspective, the LWC component bundle typically includes four types of files: .html, .css, .js, and .js-meta.xml. The latter two are mandatory, with the .js file containing the JavaScript logic executed in the browser and the .js-meta.xml file storing metadata such as version information, component visibility, and placement details (e.g., visibility on the Home page or Record Detail page). While the .html and .css files are optional, they are instrumental in defining the component's structure and style.

The LWC framework facilitates a robust client-server interaction model, streamlining the development process within the Salesforce platform. By offering a systematic and adaptable approach to component-based web development, LWC meets the changing requirements of modern online applications inside the Salesforce ecosystem.

B. Development Life-Cycle

In Lightning Web Components (LWC), lifecycle hooks offer a systematic framework essential for managing the various stages of a component's existence within the application. These hooks provide specific entry points in the component's lifecycle, enabling developers to execute code at critical phases like creation, rendering, re-rendering, and removal from the DOM.

The key hooks include [6]:

- **Constructor():** Invoked when a component instance is created, initializing component variables and properties.
- **ConnectedCallback():** Executes when the component is inserted into the DOM, suitable for setup or initialization tasks.
- **RenderedCallback():** Called after the component is rendered, ideal for post-render adjustments or actions.
- **DisconnectedCallback():** Invoked upon the component's removal from the DOM, used for cleanup activities.
- **ErrorCallback():** Handles errors within the component's lifecycle or event handlers.

This structured approach ensures precise control over component behavior and facilitates efficient resource management, contributing significantly to the performance and reliability of LWC-based applications.

C. Debugging and Testing

Debugging and testing Lightning Web Components (LWC) is essential for their development. Constructed using standard HTML and JavaScript, these components operate across various supported browsers on desktop platforms. It is reported that developers may utilize conventional browser and JavaScript debugging tools, such as Chrome DevTools, to debug their components. A Lightning Page can be created to facilitate debugging with the added component. Subsequently, this page can be loaded in the preferred browser where inspection and debugging tools are applied. The debugging process for LWC has been optimized specifically for Chrome DevTools [7].

In production mode, LWC code is optimized for performance, with JavaScript code being minified to reduce its size. However, in debug mode, JavaScript code is easier to read and debug, allowing developers to see the uncompiled source code. To enable debug mode, follow the instructions in the Salesforce documentation.

Tools such as Jest, a JavaScript testing framework, are utilized by developers to perform unit testing on LWC components in isolation. Jest tests can be run locally on your system without connecting to your browser or Salesforce org [8].

The debugging and testing of LWC components are conducted using standard browser and JavaScript debugging tools, such as Chrome DevTools, alongside testing frameworks like Jest. These methods are employed to verify the functionality and performance of the components.

3. LWC Component Structure

The basic structure of an LWC includes:

- HTML file for the component's layout
- JavaScript file for the component's logic
- CSS file for the component's styles
- XML file for the component's metadata

These files work together to create a modular, reusable, and encapsulated web component within the Salesforce platform.

The LWC file structure would look like this:

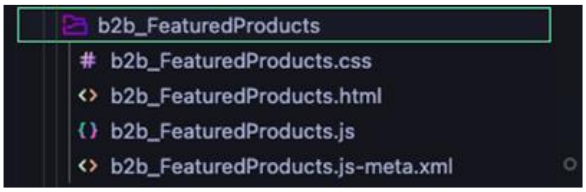

Figure 2: LWC File Structure

HTML Template (.html file) - Defines the component's HTML structure and markup.

```html
<template>
  <div class="myComponent">
    <h1>{title}</h1>
    <p>{description}</p>
  </div>
</template>
```

Figure 3: LWC HTML File

JavaScript Controller (.js file) - Contains the component's logic, event handling, and data processing using JavaScript.

```javascript
import { LightningElement, api, track } from 'lwc';

export default class MyComponent extends LightningElement {
  @api title;
  @track description = 'This is a description';
}
```

Figure 4: LWC JavaScript File

CSS Styles (.css file) - Provides the component's styling, scoped to the component to avoid global CSS conflicts.

```css
.myComponent {
  font-family: Arial, sans-serif;
}
h1 {
  color: blue;
}
```

Figure 5: LWC CSS file

Configuration File (-meta.xml file) - Defines the metadata for the component, including where it can be used (e.g., in Lightning App Builder, Community Builder, etc.).

```xml
<?xml version="1.0" encoding="UTF-8"?>
<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">
  <apiVersion>54.0</apiVersion>
  <isExposed>true</isExposed>
  <targets>
    <target>lightning__AppPage</target>
    <target>lightning__RecordPage</target>
    <target>lightning__HomePage</target>
  </targets>
</LightningComponentBundle>
```

Figure 6: LWC Meta File

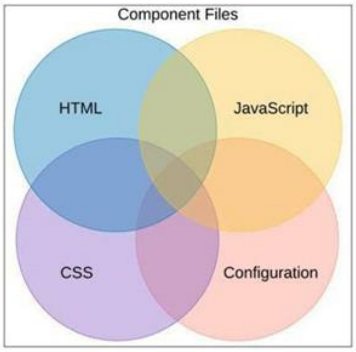

Figure 7: LWC Component Structure [13]

4. Literature Review

The literature review on Lightning Web Components (LWC) synthesizes advancements and technical implementations from multiple authoritative sources. Initially introduced in December 2018, LWC is described by Salesforce as leveraging modern JavaScript to enhance application development on its Lightning Platform, focusing on high performance and streamlined processes [1][2]. The integration of MVC architecture where it highlights its impact on improving UI and business logic separation, thereby enhancing maintainability and scalability [3]. Further insights into the transition from Aura components to LWC are provided, emphasizing performance enhancements and coding simplicity [4]. The role of Shadow DOM in ensuring component modularity and design integrity is detailed, supporting the development of robust and scalable applications . Lifecycle management practices and debugging strategies are also discussed, emphasizing the importance of effective issue resolution and component efficiency [6][7]. Finally, Jest testing is noted for its critical role in improving component quality through early error detection [8]. These collective insights underscore the robust framework provided by LWC for developing efficient, scalable applications. Lightning Web Components not only coexist and interoperate with the original Aura programming model, but also deliver unmatched performance. Aura Components leverage their own component model, templates, and a modular development programming model. Lightning Web Components are built on recent web standards, such as web components, custom elements, Shadow DOM, and more.

Figure 8: LWC Services on top of Core Stack of Web Standards [5]

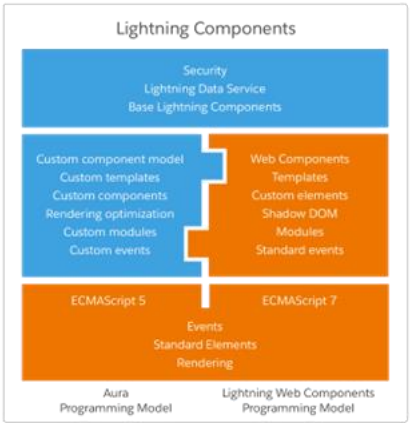

Figure 9: Coexistence and Interoperability [9]

Harnessing the latest web stack, LWC boasts numerous advantages over aura components.

- Better performance of an LWC Application.
- Built on Modern web standards.
- Compatible with New and Existing Aura components
- Optimizing website speed for faster loading times.
- Improved security, enhanced testing, and better browser compatibility.
- The ease of development.

5. Lightning Web Components Performance Best Practices

Data Loading and Server Calls

- **Use Lightning Data Service (LDS)** for efficient data operations and reduced server calls.
- **Implement Lazy Loading** to load data only when necessary.
- **Batch Requests** to minimize the number of network round-trips.
- **Cache Data** to avoid repeated server calls for unchanged information.

Component Design and DOM Manipulation

- **Compose Smaller Components** to create lightweight and reusable pieces.
- **Minimize Re-renders** by using track and @api properties appropriately.
- **Optimize DOM** updates to ensure only necessary parts are re-rendered.

CSS, JavaScript, and Event Handling

- **Scope CSS with Modules** to avoid global style conflicts.
- **Simplify CSS Rules** to prevent rendering slowdowns.
- **Attach Event Listeners Wisely** and use **Event Delegation** to handle events efficiently.

Media Optimization and Profiling

- **Lazy Load Images** to improve load times.
- **Optimize Image Sizes** for quicker loading.
- **Profile Components** with tools like Chrome DevTools to identify performance issues.
- **Monitor with Salesforce Lightning Usage App** for insights and performance tracking.

6. Research Method

As part of an experiment to test its capabilities and performance as a programming tool, a section of a large project was converted from Aura component to LWC. The project in question is Dream House, a real estate company that uses the Salesforce platform to help brokers manage their properties and customers find their dream homes. The experiment focused on updating the two most complicated pages of the project, namely "Property Finder" and "Property Explorer," using LWC and additional customizations.

Property Finder - It is a page that allows customers to search for properties using various filters. The results are then displayed as a list of matching homes' photographs, with simple property details displayed when a property is clicked or selected.
Property Explorer - It is similar to "Property Finder" but displays results as pin drops on a map instead of individual photos.
The conversion of the components from Aura to LWC was a comprehensive task. Twenty-two components across the two pages, 'Property Finder' and 'Property Explorer,' were transformed using the most straightforward method possible to achieve the best ROI (balance of time invested and performance gains). This significant number of components converted underscores the scale of the task and its impact on the project.
The Lightning Platform uses a standard metric called Experienced Page Time (EPT) to measure page load time. EPT measures the time taken to load a page from the moment a user clicks to navigate to a page until the page is fully loaded. The measurements were conducted in Lightning Platform's performance lab.
Salesforce Experienced Page Time (EPT) measures the time for a user to fully experience a Salesforce page, from initiation to full interactivity. EPT provides a comprehensive view of page performance, including all loading and rendering phases.
Network Time: The time taken for the request to travel to the server and for the server to send back the response.Network Time: The time taken for the request to travel to the server and for the server to send back the response.
Server Processing Time: The time taken by the Salesforce server to process the request and prepare the response.
Browser Processing Time: The time taken by the user's browser to interpret and render the page.
EPT is crucial for understanding the actual user experience as it covers the entire page load journey, offering a more accurate representation of performance. It helps identify bottlenecks and improve overall page performance and user satisfaction.

7. Result

Page	Cache State	Aura (EPT, ms)	LWC (EPT, ms)	% Improvement
Property Finder	Cold	1080	813	-24.72%
Property Finder	Warm	428	157	-63.32%
Property Explorer	Cold	912	890	-2.41%
Property Explorer	Warm	289	197	-31.83%

Figure 10: Aura to LWC Lab Measurements [10]

After converting DreamHouse from Aura to LWC, we noticed significant improvements. The development experience aligned more with current web front-end development standards and patterns, and substantial performance gains were observed. Lab measurements indicated improvements ranging from 2.4 percent to 63 percent.

7. Conclusion

In conclusion, Lightning Web Components offers a modern, performant, and developer-friendly framework that aligns with the latest web standards. By embracing LWC, Salesforce developers can build sophisticated, scalable, and maintainable applications that provide exceptional user experiences on the Lightning Platform. LWC's event-driven architecture helps in building highly interactive and responsive applications. The advantages extend to the ease of maintaining and updating the codebase due to the declarative nature of these components. Overall, LWCs enable developers to create innovative solutions that support business advancement and enhance user satisfaction.

References

[1]. Salesforce Announces Lightning Web Components, a Modern JavaScript Programming Model for Building Apps on the Lightning Platform in 2018, https://www.salesforce.com/news/press-releases/2018/12/13/salesforce-announces-lightning-web-components-a-modern-javascript-programming-model-for-building-apps-on-the-lightning-platform/J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.

[2]. Introducing Lightning Web Components in Dec 2018, https://www.salesforce.com/news/press-releases/2018/12/13/salesforce-announces-lightning-web-components-a-modern-javascript-programming-model-for-building-apps-on-the-lightning-platform/.

[3]. Salesforce LWC - A Client-Centric MVC Architecture in Oct 2020, https://www.linkedin.com/pulse/salesforce-lwc-mvc-architecture-prasanta-kumar-pardhi/.

[4]. Embracing Modernity with Lightning Web Components (LWC): Shifting from Aura to Salesforce's Cutting-Edge Front-end Development in Sept 2023, https://scrumdigital.com/blog/shifting-from-aura-to-salesforces-cutting-edge-front-end-development/.

[5]. Introducing Lightning Web Components 2018 https://developer.salesforce.com/blogs/2018/12/introducing-lightning-web-components

[6]. Lifecycle hooks in Lightning Web Component in Dec 2020, https://jayakrishnasfdc.wordpress.com/2020/12/06/lifecycle-hooks-in-lightning-web-component/.

[7]. Debug Lightning Web Components, 2019 https://developer.salesforce.com/blogs/2019/02/debug-your-lightning-web-components

[8]. JEST - Testing Lightning Web Components 101, Aug 2020, https://www.linkedin.com/pulse/jest-testing-lightning-web-components-101-swarna-gopalan/.

[9]. Lightning web components - Episode 1 - An Introduction 2018 https://www.slideshare.net/developerforce/lightning-web-components-episode-1-an-introduction

[10]. Case Study: DreamHouse Gains Speed by Switching to LWC , 2019 https://developer.salesforce.com/blogs/2019/06/case-study-dreamhouse-gains-speed-by-switching-to-lwc

[11]. Lightning Components Performance Best Practices , 2017 https://developer.salesforce.com/blogs/developer-relations/2017/04/lightning-components-performance-best-practices

[12]. Lightning Web Components Performance Best Practices, 2020 https://developer.salesforce.com/blogs/2020/06/lightning-web-components-performance-best-practices

[13]. Lightning Web Components Basics , 2019 https://trailhead.salesforce.com/content/learn/modules/lightning-web-components-basics/push-lightning-web-component-files