WEB COMPONENTS

Project report

From - 01.09.2023

To - 30.09.2023

Pawan R
Integra Micro System
4,Bellary Road, Jakkur,
yashoda nagar, Bengaluru
560064.

Overview

Project Name: Web components Development

Project Duration: September 1, 2023 - October 31, 2023

Project Team: Pawan R [Intern Dev]

Introduction

This documentation provides an overview of our dynamic web component project, showcasing its capabilities and practical use cases. Explore how to seamlessly integrate web components, define user-specific attributes, manage component deletion, and enable inter-component communication.

Project Description

Web components are a set of web platform API's that allow you to create reusable custom elements in web documents and web applications. They are a key part of modern web development, enabling developers to encapsulate and reuse UI components across different frameworks and libraries. This project aims to develop a set of custom Web Components that can be used in various web projects.

Objectives

1. Develop a set of reusable web components: Create a collection of custom web components that cover common UI elements and functionalities, such as buttons, modals, forms, and more.
2. Ensure Cross-Browser Compatibility: Ensure that the web components are compatible with major web browsers, including Chrome, Firefox, Safari and Edge.
3. Document Component: Provide clear and comprehensive documentation for each web component, including usage examples, and installation instructions.
4. Test and debug: Implement automated testing and debugging procedures to catch and resolve issues early in the development process.
5. Demonstration and Integration: Showcase the usage of these Web Components in a sample web application, demonstrating their flexibility and reusability.

Scope of work

The scope of work for this project includes the following tasks:

1. Requirements Gathering: Collaborate with stakeholders to determine the specific UI components needed and gather requirements for each component.
2. Design and Development: Design and develop each Web Component using HTML, CSS, and JavaScript. Follow best practices for code organization and maintainability.
3. Cross-Browser Testing: Test the Web Components on various web browsers to ensure compatibility and fix any issues that arise.
4. Documentation: Create detailed documentation for each Web Component, including usage instructions, code examples.
5. Testing and Debugging: Implement automated testing procedures and debugging tools to identify and resolve issues.
6. Sample Application: Develop a sample web application that showcases the use of these Web Components in real-world scenarios.
7. Integration with Package Managers: Publish the Web Components to package registries (e.g., npm) for easy integration into different web projects.
8. Maintenance and Updates: Plan for ongoing maintenance and updates to address any future compatibility issues or feature enhancements.

Hardware Requirements

- A desktop or laptop with an internet connection.
- Hard disk with enough space.
- 4GB of RAM.
- Windows operating system.

Non Functional Requirements

1. Security: The subsystem should provide a high level of security and integrity of the data held by the system.
2. Performance and Response time: The system should have high performance rate when executing user's input and should be able to provide feedback or response within a short time span usually 50 seconds for highly complicated tasks and 20 to 25 seconds for less complicated tasks.
3. Error handling: Error should be considerably minimized and an appropriate error message that guides the user to recover from an error should be provided. Validation of user's input is highly essential. Also the standard time taken to recover from an error should be 15 to 20 seconds.
4. Availability: This system should always be available for access at 24 hours, 7 days a week. Also in the occurrence of any major system malfunctioning, the system should be available in 1 to 2 working days.
5. Ease of use: Considered the level of knowledge possessed by the users of this system, a simple but quality user interface should be developed to make it easy to understand and required less training

Software Requirements

1. HTML
2. Java Script
3. CSS

HTML

HTML is an acronym that stands for HyperText Markup Language.

HyperText: HyperText simply means "Text within Text". A text has a link within it, is a hypertext. Every time you click on a word that brings you to a new webpage, you have clicked on a hypertext.

Markup language: A markup language is a programming language that is used to make text more interactive and dynamic. It can turn a text into images, tables, links, etc. An HTML document is made of many HTML tags and each HTML tag contains different content

Java Script

JavaScript is a dynamic computer programming language. It is lightweight and most commonly used as a part of web pages, whose implementations allow client-side scripts to interact with the user and make dynamic pages. It is an interpreted programming language with object-oriented capabilities.

CSS

CSS (Cascading Style Sheets) is a fundamental component of web development. It's a stylesheet language used to define the visual presentation and layout of web pages written in HTML and XML. CSS allows web designers and developers to control the colors, fonts, spacing, and positioning of elements on a webpage.

With its cascading nature, CSS rules can be applied hierarchically, giving designers fine-grained control over the appearance of web content. Whether it's creating responsive designs for various screen sizes or ensuring a consistent look and feel across a website, CSS plays a crucial role in enhancing the aesthetics and user experience of the web.

Web Components Overview

Web components represent a suite of web platform APIs that enable developers to create custom, reusable, and encapsulated HTML tags. These tags can seamlessly integrate into web pages and applications, offering a powerful solution for building modular and highly interactive web interfaces. Here's an in-depth look at the key aspects of web components:

Custom Elements

Custom Elements are a core feature of web components that allow developers to define their custom HTML elements. These elements can encapsulate a set of behaviors, properties, and methods, making them highly reusable across different parts of a web application. Custom elements adhere to the Web Component standards, ensuring broad cross-browser compatibility.

Key points about Custom Elements:

- Custom elements are defined using the `customElements.define()` method, specifying a name and an associated class that extends `HTMLElement`.
- They provide a consistent and self-contained way to create new HTML tags with custom behavior.
- Custom elements are invoked using standard HTML syntax, for example, `<my-custom-element></my-custom-element>`.

Shadow DOM

Shadow DOM is a fundamental aspect of web components that enables the encapsulation of the component's internal structure, styles, and JavaScript logic. The Shadow DOM ensures that the component's structure and styles do not interfere with the rest of the web page, avoiding naming conflicts and unintended side effects.

Key points about Shadow DOM:

- It is a hidden, separate DOM subtree within a custom element.
- Shadow DOM can contain its HTML, CSS, and JavaScript, isolated from the main document's context.
- This encapsulation protects component internals and promotes reusability.

HTML Templates

HTML Templates allow you to define the structure of a web component's content without rendering it immediately. This is especially useful for creating dynamic content that can be cloned and manipulated as needed.

Key points about HTML Templates:

- Templates are defined using the <template> tag in HTML.
- They can include placeholders for data that can be filled in dynamically.
- Templates can be cloned, manipulated, and attached to the Shadow DOM when needed, allowing for efficient and flexible content rendering.

Cross-Browser Compatibility

Web components are designed to be highly compatible across various modern web browsers. However, it's essential to consider compatibility requirements for older browsers. You can use tools like web component polyfills to extend support to older browser versions.

Integration with JavaScript Libraries and Frameworks

Web components are versatile and can be seamlessly incorporated into any JavaScript library or framework that operates with HTML. This flexibility makes them a valuable asset for enhancing web-based interfaces within your preferred development ecosystem.

In summary, web components offer a standardized and versatile approach to creating reusable and encapsulated elements that can enhance the interactivity and modularity of your web applications. With Custom Elements, Shadow DOM, HTML Templates, and JavaScript modules, you can create encapsulated, self-contained components that are easy to maintain and integrate into a wide range of web development projects.

Getting started

This section will guide you through the initial steps to set up and run the project locally.

Prerequisite

Before you get started, make sure you have the following prerequisites in place:

1. Web Browser: Ensure you have a modern web browser installed. This project is designed to work with a wide range of browsers for maximum compatibility.

2. Development Environment: You will need a code editor or integrated development environment (IDE) of your choice to work with the project files. Popular options include Visual Studio Code, Sublime Text, or WebStorm.

Installation

Once you have the prerequisites in place, follow these steps to install and set up the Dynamic Web Components project:

1. Clone the Repository:
    - Use Git to clone the project repository to your local machine:

      ```
      git clone https://github.com/pawanrgowda/web-components.git
      ```

    - If you prefer, you can also download the repository as a ZIP file from the project's GitHub page.
2. Navigate to the Project Directory:
    - Open your terminal or command prompt and change your working directory to the project's root folder:

      ```
      cd web-components
      ```

Running the project locally with Live Server(VS Code)

If you prefer to use the Live Server extension in Visual Studio Code, follow these steps:

Install the Live Server Extension:

1. Open your Visual Studio Code editor.
    - Go to the Extensions view by clicking the square icon on the left sidebar or using the shortcut Ctrl+Shift+X.
    - Search for "Live Server" and click the "Install" button for the Live Server extension offered by Ritwick Dey.
2. Open Your Project Folder:
    - Open your Dynamic Web Components project folder in Visual Studio Code.
3. Launch Live Server:
    - Right-click on the HTML file that serves as the entry point of your project (e.g., index.html).
    - From the context menu, select "Open with Live Server."
    - This action will automatically start a local development server and open your project in your default web browser.

4. Access Your Project:
    - Your project will be accessible at a live server URL, such as http://127.0.0.1:5500/index.html. You can see the URL in the status bar at the bottom of the VS Code window.

You're now running your Dynamic Web Components project using the Live Server extension in Visual Studio Code. Enjoy a live-reloading development experience with easy access to your project in the browser.

Features included

Dynamic Web Component Integration:

One of the project's core functionalities lies in its dynamic web component integration. This innovative capability enables developers to seamlessly inject web components into the web application at runtime. This dynamic insertion empowers designers to create highly interactive and adaptable user interfaces, responding to user needs in real-time.

User-Defined Attributes:

At the heart of this project is the ability for users to define and assign attributes to web components. This feature allows for a high degree of personalization, ensuring that these components can be tailored to specific requirements. Users can configure attributes that influence the appearance, behavior, and functionality of these components, offering a deeply customized user experience.

Deletion Option:

To maintain the application's agility and flexibility, a deletion option is provided. This feature permits the removal or disconnection of web components from the application. It's a crucial component management tool, offering developers a way to streamline their application by removing unnecessary or outdated components, ensuring optimal performance and maintainability.

Inter-Component Communication:

Beyond just web application to component interaction, this project showcases the capabilities of inter-component communication. It introduces two distinct web components, one of which plays a vital role in this communication ecosystem. When a web component is clicked within the application, this specialized component retrieves and displays the attributes of the selected component in a modal box.

1. **Opening a Blog Modal on Product Selection**

```
document.body.addEventListener('click', function (e) {
    if (e.target && e.target.matches('product-card')) {
        let blogModal = document.querySelector('blog-modal');
        blogModal.setAttribute('open', true);
        blogModal.setAttribute('title', e.target.getAttribute('title'));
        blogModal.setAttribute('content', e.target.getAttribute('description'));
        blogModal.setAttribute('briefdescription',
e.target.getAttribute('briefdescription'));
    }
});
```

This JavaScript code enhances user interaction within your web application by enabling the dynamic opening of a "blog-modal" component when a product is selected. Here's a detailed breakdown of the code:

- **Event Listener Setup**

  `document.body.addEventListener('click', function (e) { ... }`

  This code adds an event listener to the entire document's body, responding to click events. It captures user interactions across your web page.

- **Element Matching**

  `if (e.target && e.target.matches('product-card')) { ... }`

  Within the event listener, it checks whether the clicked element, e.target, matches the selector 'product-card.' This condition ensures that the code executes when a specific type of element, likely representing a product, is clicked.

- **Accessing the Blog Modal**

  `let blogModal = document.querySelector('blog-modal');`

  The code fetches the "blog-modal" element from the DOM. This element serves as the container for displaying detailed information about the selected product when the modal is opened.

- **Setting Modal Attributes**

  `blogModal.setAttribute('open', true);`

  This line sets the 'open' attribute of the "blog-modal" to 'true.' This action likely triggers the modal's visibility, revealing the detailed content to the user.

  `blogModal.setAttribute('title', e.target.getAttribute('title'));`

  The code assigns the 'title' attribute of the "blog-modal" with the 'title' attribute obtained from the clicked 'product-card.' This effectively updates the title displayed within the opened modal to match the selected product's title.

  `blogModal.setAttribute('content', e.target.getAttribute('description'));`

  Similarly, this line updates the 'content' attribute of the "blog-modal" with the 'description' attribute retrieved from the clicked 'product-card.' This ensures that the product description is displayed within the modal.

  `blogModal.setAttribute('briefdescription', e.target.getAttribute('briefdescription'));`

  The 'briefdescription' attribute of the "blog-modal" is set using the 'briefdescription' attribute from the clicked 'product-card.' This action populates the modal with a brief description or summary of the selected product.

2. **Setting attributes of Blog Modal**

   ```
   instance.querySelector('.title').innerHTML = this['title'];
   instance.querySelector('.content').innerHTML = this['content'];
   instance.querySelector('.briefdescription').innerHTML = this['briefdescription'];
   shadowRoot.appendChild(instance);
   ```

This portion of the code is responsible for dynamically updating the content displayed within the BlogModal web component. It extracts specific elements within the modal's shadow DOM and populates them with the content stored in the corresponding properties of the web component instance.

- **Updating the Title**

  `instance.querySelector('.title').innerHTML = this['title'];`

  Here, the code targets an element with the CSS class "title" within the modal's shadow DOM using querySelector. It then updates the inner HTML of this element by setting it to the value of the title property of the web component instance. This operation ensures that the title displayed in the modal corresponds to the title property of the component instance.

- **Updating the Content**

  `instance.querySelector('.content').innerHTML = this['content'];`

  In a similar fashion, this code selects an element with the CSS class "content" and updates its inner HTML with the content stored in the content property of the web component instance. This allows the modal to display the main content of the component instance.

- **Updating the Brief Description**

  `instance.querySelector('.briefdescription').innerHTML = this['briefdescription'];`

  This line focuses on an element with the CSS class "briefdescription" and sets its inner HTML to the value of the briefdescription property of the web component instance. This is useful for providing a brief summary or description within the modal.

- **Appending the Instance**

  `shadowRoot.appendChild(instance);`

  After updating the content within the modal's shadow DOM, this step appends the modified instance back into the shadow DOM. This ensures that the changes made to the content are visually reflected within the BlogModal web component.

This functionality allows the BlogModal to dynamically display content based on the values of its title, content, and briefdescription properties. When users interact with your application and select a specific item, this code ensures that the modal presents the relevant information, creating a responsive and user-friendly experience.

This illustrates the seamless communication between different web components, opening the door to a plethora of possibilities for building modular and collaborative web applications.

Styling

Styling web components is a crucial aspect of creating a cohesive and visually appealing user interface. Web components offer various methods for styling, ensuring that your components can be seamlessly integrated into your web application's design.

Shadow DOM

Web components inherently provide style encapsulation through Shadow DOM. Styles defined within a component's Shadow DOM are isolated from external styles, reducing the risk of CSS conflicts.
Key considerations when styling web components with Shadow DOM:

1. Isolation: Styles within the Shadow DOM are isolated and do not affect the main document's styles, promoting encapsulation.
2. Scoped Styles: Styles defined within the Shadow DOM are specific to that component, preventing unintended global style changes.
3. Encapsulation: The Shadow DOM allows you to create self-contained components with encapsulated styles.

Use Cases

Web components are versatile and can be applied to a wide range of use cases in modern web development. Below are some practical scenarios and examples of how web components can enhance the functionality and user experience of web applications:

1. Custom UI Elements

Web components are an excellent choice for creating custom user interface elements that are not readily available in standard HTML. For example, you can build custom navigation menus, tabbed interfaces, date pickers, or sliders. This allows you to create a unique and branded look and feel for your web application.

2. Reusable Widget

Web components can encapsulate complex widgets or functionalities, making them easy to reuse across different parts of your application. This promotes code modularity and reusability.

3. Dynamic Forms

Web components are particularly useful for creating dynamic forms where users can input data. You can create form components that validate input, provide real-time feedback, and offer a seamless user experience.

4. Interactive Charts and Graphs

You can build web components that generate interactive charts and graphs, enabling users to visualize data in a dynamic and engaging way.

5. Collaborative Features

Web components can facilitate real-time collaboration features within your web application. For instance, you can create components for collaborative text editing, whiteboarding, or chat functionality.

6. Contextual Modals

Web components are perfect for creating contextual modals and pop-up windows that appear when users interact with specific elements or need to view additional information.

Conclusion

In summary, the Dynamic Web Components project presents an innovative and comprehensive exploration of web components' profound influence on modern web development. By offering dynamic integration, user-defined attributes, and seamless inter-component communication, this project underscores the versatility and potential of web components.

Web components, as exemplified here, serve as a cornerstone for developing web applications that are modular, interactive, and highly tailored to the end-users' needs. They empower developers to create personalized and engaging user interfaces while adhering to rigorous standards of cross-browser compatibility.

As you delve into the realm of web component development, it is our hope that this project has not only provided insights but also served as a source of inspiration. The future of web application development is bright, and web components play a pivotal role in shaping it.

In your endeavors with web components, we encourage you to explore the boundless opportunities they offer, pushing the boundaries of innovation and creating remarkable web experiences. We are excited to witness the extraordinary applications that will emerge as you harness the full potential of web components.