


Building a Multilingual Static Website: A Step-by-Step Guide

 medium.com/@nohanabil/building-a-multilingual-static-website-a-step-by-step-guide-7af238cc8505

Noha Nabil

August 10, 2023



Photo by [Jaye Haych](#) on [Unsplash](#)

A multilingual website is a website that offers content in more than one language. This can be a great way to reach a wider audience and improve your website's SEO.

In this article, I'll walk you through the process of building a static website that supports multiple languages using **HTML**, **CSS**, and **JavaScript**.

Let's start by laying the foundation for our multilingual website. In your HTML file, make sure to include the appropriate language attribute in the `<html>` tag to specify the default language. For example, to set English as the default language:

```
<html lang="en">
<!-- Your website content here -->
</html>
```

Additionally, include the necessary `<meta>` tags for character encoding and viewport settings:

```
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <!-- Other meta tags and stylesheets -->
</head>
```

Creating Language Options

To allow users to switch between languages, let's integrate language options into the navigation bar. Here's an example of how you can achieve this:

```
<nav>
  <ul>
    <li><a href="index.html" onclick="changeLanguage('en')">English</a></li>
    <li><a href="index.html" onclick="changeLanguage('ar')">عربي</a></li>
  </ul>
</nav>
```

Updating Main Content

Let's dive into the heart of your website — the main content. This is where you'll display the core information about your site in different languages. The key here is to ensure that your JavaScript code dynamically updates the content based on the selected language.

In your HTML code, you've structured the main content as follows:

```
<main>
  <h1 data-i18n="welcome_message"></h1>
  <p data-i18n="about_us"></p>
</main>
```

Here, the `data-i18n` attribute in the `<h1>` and `<p>` tags marks these elements for language-specific content. To make these elements display the appropriate translations, the JavaScript code provided later will work its magic.

Language Data and JSON Files

For each language, create a JSON file to store translations for different elements on your website. For instance, you might have an `en.json` file for English and an `ar.json` file for Arabic. Here's a simplified example of the structure of these files:

```

/* en.json */
{
  "welcome_message": "Welcome to Our Website!",
  "about_us": "Learn more about us...",
  "footer_text": "© 2023 All rights reserved."
}

/* ar.json */
{
  "welcome_message": "!مرحبًا بكم في موقعنا",
  "about_us": "...اعرف المزيد عنا",
  "footer_text": "...جميع الحقوق محفوظة © 2023"
}

```

Updating Content Dynamically

To dynamically update content based on the selected language, we'll use JavaScript. Here's how you can achieve this:

```

// Function to update content based on selected language
function updateContent(langData) {
  document.querySelectorAll('[data-i18n]').forEach(element => {
    const key = element.getAttribute('data-i18n');
    element.innerHTML = langData[key];
  });
}

```

The `updateContent` function iterates through elements with the `data-i18n` attribute and updates their text content using the corresponding translation from the language data.

Setting Language Preferences

To remember the user's language preference, we'll use the `localStorage` API. This allows us to store and retrieve data locally in the user's browser:

```

// Function to set the language preference
function setLanguagePreference(lang) {
  localStorage.setItem('language', lang);
  location.reload();
}

```

Whenever the user selects a language, the `setLanguagePreference` function is called to store the preference and reload the page with the new language settings.

Fetching Language Data

To make your multilingual static website work, you need a way to retrieve the language-specific content stored in your JSON files. This is where the `fetchLanguageData` function comes into play.

```
// Function to fetch language data
async function fetchLanguageData(lang) {
  const response = await fetch(`languages/${lang}.json`);
  return response.json();
}
```

This function takes the selected language as an argument and constructs a URL to fetch the corresponding JSON file from the `languages/` directory. When the `fetch` request is made, it returns a promise that resolves into a Response object. The `response.json()` method is then used to parse the JSON content from the Response object.

In simpler terms, the `fetchLanguageData` function is responsible for getting the language-specific content stored in your JSON files, which will later be used to update your website's content dynamically. This is a pivotal step in ensuring that your website displays the correct translations for each language.

Changing the Display Language

Now that we've covered the foundation of your multilingual static website, it's time to explore how users can actually switch between different languages. The `changeLanguage` function is at the core of this process.

```
// Function to change language
async function changeLanguage(lang) {
  await setLanguagePreference(lang);

  const langData = await fetchLanguageData(lang);
  updateContent(langData);
  toggleArabicStylesheet(lang); // Toggle Arabic stylesheet
}
```

This function is your gateway to creating a seamless transition between languages. Let's break down what's happening here:

1. `await setLanguagePreference(lang)`: This line first sets the user's language preference using the `setLanguagePreference` function we discussed earlier. By storing this preference in `localStorage`, you ensure that the user's chosen language persists even after they leave the website.
2. `const langData = await fetchLanguageData(lang)`: The `fetchLanguageData` function is used to retrieve the language-specific content for the selected language. This content, stored in JSON files, includes translations for various elements on your website.

Linking Stylesheets and JavaScript

Before we wrap up, let's cover the essential step of linking your stylesheets and JavaScript file to your HTML. Properly linking these files is crucial for the functionality and aesthetics of your multilingual static website.

Stylesheet Linking

In your `<head>` section, you have a link to your main stylesheet:

```
<link rel="stylesheet" href="./assets/css/style.css">
```

This line connects your HTML file to the `style.css` file located in the `assets/css/` directory. This stylesheet likely contains the basic styling for your website's layout and appearance.

When working with multilingual websites, especially when different languages require different styles, it's important to consider how your styles will adapt. For instance, you might need separate styles for Arabic text to ensure proper alignment and typography.

JavaScript Linking

Similarly, your JavaScript file is linked at the end of the `<body>` section:

```
<script src="assets/js/script.js"></script>
```

This line connects your HTML to the `script.js` file located in the `assets/js/` directory. This JavaScript file contains the functions that make your multilingual features and content updates possible.

Keep in mind that as you develop your multilingual website, your JavaScript code may evolve to include more features or optimizations. Regularly reviewing and updating your JavaScript file is a crucial part of maintaining your website's functionality.

Dynamic Stylesheet Switching

Different languages often require different styles to ensure proper typography and layout. Let's create a function to toggle the Arabic stylesheet based on language selection:

```
// Function to toggle Arabic stylesheet based on language selection
function toggleArabicStylesheet(lang) {
  const head = document.querySelector('head');
  const link = document.querySelector('#styles-link');

  if (link) {
    head.removeChild(link); // Remove the old stylesheet link
  } else if (lang === 'ar') {
    const newLink = document.createElement('link');
    newLink.id = 'styles-link';
    newLink.rel = 'stylesheet';
    newLink.href = './assets/css/style-ar.css'; // Path to Arabic stylesheet
    head.appendChild(newLink);
  }
}
```

Note: Here I created another stylesheet for arabic language "style-ar.css".

This function ensures that the correct stylesheet is applied when switching between languages.

Page Initialization

Finally, we'll use the `DOMContentLoaded` event to initialize the page's content and styles based on the user's preferred language:

```
// Call updateContent() on page load
window.addEventListener('DOMContentLoaded', async () => {
  const userPreferredLanguage = localStorage.getItem('language') || 'en';
  const langData = await fetchLanguageData(userPreferredLanguage);
  updateContent(langData);
  toggleArabicStylesheet(userPreferredLanguage);
});
```

The `DOMContentLoaded` event triggers the code once the page's HTML content has been fully loaded. It retrieves the user's preferred language from `localStorage`, updates the content, and toggles the stylesheet accordingly.

You can find the whole project on Github [Here](#).

Conclusion

By following these steps, you can create a dynamic and user-friendly multilingual static website. Users will be able to seamlessly switch between languages, ensuring that your website's content is accessible to a wider audience. Remember to keep your JSON translation files up to date as your website evolves, and continue to provide an excellent user experience for visitors around the world.

Creating a multilingual static website requires careful attention to detail and a solid understanding of HTML, CSS, and JavaScript. However, the effort is well worth it as you enhance accessibility and engagement for users from various linguistic backgrounds.