JAVASCRIPT: SINGLE PAGE APPLICATIONS

WHAT IS A SINGLE PAGE APPLICATION?

- A *Single Page Application* (SPA) is a web application that loads a single HTML page and dynamically updates that page as the user interacts and navigates through the site.
  - Using *AJAX*, we can update only the portions of the page that change, instead of having to reload the entire page.
- There are many benefits to using an SPA, including:
  - A more native app-like experience for the user
  - Decreased load time and amount of data that needs to be transferred from server to client
  - Improved user experience because the data can be loaded in the background

WHAT IS SENNA.JS?

- *Senna.js* makes the SPA magic happen in Liferay.
- It is a lightweight, fast, open-source *Single Page Application* engine.
- It was built by a team of developers here at Liferay as well as a bunch of great contributors.
- *Senna.js* has been integrated into DXP to leverage the benefits of an SPA.
- For detailed documentation and examples, you can check out *http://sennajs.com/docs*.

HOW DOES IT WORK?

- After the SPA is initially loaded, all subsequent navigations are handled without a full page reload.
- Additional content is loaded using `XMLHttpRequest` via History API, which is able to update the URL without refreshing the page, therefore making it so your dynamic site can be shared and bookmarked.
- Having a URL for each state of your page also makes it so that the content can be fully crawled by search engines.
- There are many features *Senna.js* provides to make sure our SPAs meet these needs.

SENNA.JS FEATURES

- **SEO & Bookmarkability**: Sharing or bookmarking a link should always display the same content. Search engines are able to index that same content.
- **Hybrid rendering**: Ajax + server-side rendering allows disabling pushState at any time, allowing progressive enhancement.
  - The way you render the server-side doesn't matter. It can be simple HTML fragments or even template views.
- **State retention**: Scrolling, reloading, or navigating through the history of the page should get back to where it was.
- **UI feedback**: When some content is requested, it indicates to the user that something is happening.
- **Pending navigations**: Blocks UI rendering until data is loaded, then displays the content at once
  - It's important to give some UX feedback during loading.

ADDITIONAL SENNA.JS FEATURES

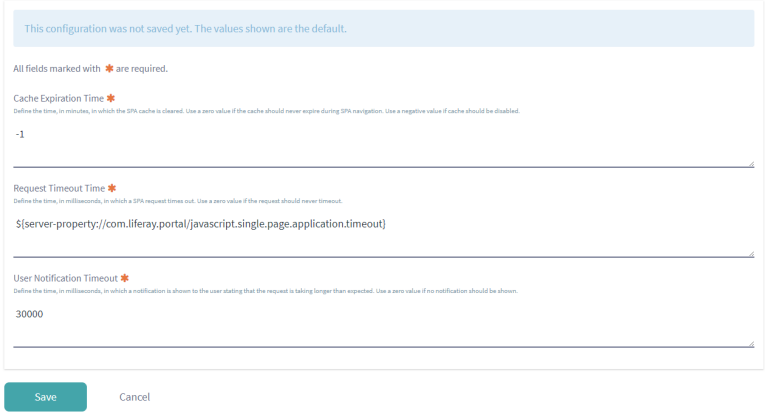
- **Timeout detection**: Timeout if the request takes too long to load or when navigating to a different link while another request is pending
- **History navigation**: By using History API, you can manipulate the browser history, so you can use the browser's back and forward buttons.
- **Cacheable screens**: Once you load a certain surface, this content is cached in memory and is retrieved later on without any additional request.
- **Page resources management**: Evaluate scripts and stylesheets from dynamically loaded resources.
  - Additional content loaded using XMLHttpRequest can be appended to the DOM. For security reasons, some browsers won't evaluate `<script>` tags from the new Fragment.
  - The SPA engine should handle extracting scripts from the content and parsing them, respecting the browser contract for loading scripts.

ENABLING SPA IN LIFERAY

- Enabling SPA in Liferay is simple.
- All that is needed is to have the `frontend-js-spa-web` module deployed and enabled.
  - This module is included with Liferay by default.
- SPA is also enabled by default and requires no changes to your workflow or existing code.

CUSTOMIZING SPA SETTINGS

- It's easy to customize the behavior of your *Single Page Application*.
- Depending on what you need to configure, you can customize SPA options in one of two places:
  - Caching and timeout settings can be configured in *System Settings*.
  - Options for disabling SPAs are set within specific elements.
- To configure settings via *System Settings* in Liferay, go to *Control Panel→Configuration→System Settings*, click the *Foundation* tab, and then click on *Frontend SPA Infrastructure*.



SPA SETTINGS OPTIONS

- You will find a number of options in the *Frontend SPA Infrastructure* section of *System Settings*:

1. **Cache Expiration Time**: The time, in minutes, in which the SPA cache is cleared.
    - A zero value means the cache should never expire during SPA navigation.
    - A negative value means the cache should be disabled.
2. **Request Timeout Time**: The time, in milliseconds, in which a SPA request times out.
    - A zero value means the request should never timeout.
3. **User Notification Time**: The time, in milliseconds, in which a notification is shown to the user stating that the request is taking longer than expected.
    - A zero value means no notification should be shown.

DISABLING SPA IN LIFERAY

- To disable SPA across all of Liferay, you can add the following line to the `portal-ext.properties` file:

```
javascript.single.page.application.enabled = false
```

DISABLING SPA IN SPECIFIC ELEMENTS

- If there is a certain application or page that you don't want to be part of the SPA, you have some options:
  - Blacklist the application to disable SPA for the entire application
  - Use the `data-senna-off` annotation to disable SPA for a specific form or link
- To blacklist an application from SPA, open your application class and set the `_singlePageApplication` property to false:

```
_singlePageApplication = false;
```

  - If you prefer, you can set this property to false in your *portlet.xml* instead by adding the following property to the `<portlet>` section:

```
<single-page-application>false</single-page-application>
```

- To disable SPA for a specific form or link, add the `data-senna-off` attribute to the element and set the value to `true`. For example:

```
<a data-senna-off="true" href="/pages/page2.html">Page 2</a>
```

LEVERAGING THE SPA LIFECYCLE

- Sometimes it is necessary to know the navigation status.
- *Senna.js* makes this easy by exposing lifecycle events that represent state changes in the application. The available events are:
  - **beforeNavigate**: Fires before navigation starts. Event payload: `{ path: '/pages/page1.html', replaceHistory: false }`
  - **startNavigate**: Fires when navigation begins. Event payload: `{ form: '<form name="form"></form>', path: '/pages/page1.html', replaceHistory: false }`
  - **endNavigate**: Fired after the content has been retrieved and inserted onto the page. Event payload: `{ form: '<form name="form"></form>', path: '/pages/page1.html' }`
- These events can be leveraged easily by listening for them on the Liferay global object.

```
Liferay.on('beforeNavigate', function(event) {
    alert("Get ready to navigate to " + event.path);
});
```

GLOBAL LISTENERS

- SPAs provide several improvements that benefit your site, but there may be some additional maintenance required as a result.
- In a traditional navigation scenario, every page refresh resets everything, so you don't have to worry about what's left behind.
- In a SPA scenario however, global listeners such as `Liferay.on` or `Liferay.after` or body delegates can become problematic.
- Every time you execute these global listeners, you add yet another listener to the globally persisted Liferay object which results in multiple invocations of those listeners.

DETACHING EVENT LISTENERS

- To prevent this, you need to listen to the navigation event in order to detach your listeners.
- For example, you would use the following code to detach the event listeners of a global category event:

```
var onCategory = function(event) {...};

var clearPortletHandlers = function(event) {
    if (event.portletId === '<%= portletDisplay.getRootPortletId() %>') {
        Liferay.detach('onCategoryHandler', onCategory);
        Liferay.detach('destroyPortlet', clearPortletHandlers);
    }
};


Liferay.on('category', onCategory);
Liferay.on('destroyPortlet', clearPortletHandlers);
```

Notes: