Web Accessibility

Overview Last modified: 04/25/2018 11:25pm

Learn about the Sugar Accessibility Plugin for Sidecar.

Introduction

Making your application accessible -- per the standards defined by the W3C's WAI specifications -- is a hard thing to do and even harder to maintain. The goal of the **Sugar Accessibility Plugin for Sidecar** is to automatically apply rules to your rendered HTML, so that you don't have to be concerned with all of the intricacies of accessibility.

With respect to programmatically applying accessibility rules, you can generally assume that the rules fall into one of three categories:

1. Rules that are dependent on the context of the element's use and cannot be applied programmatically because the context is never clear.
2. Rules that can be applied programmatically, but only when the context is clear.
3. Rules that can always be applied programmatically.

We plan to continue to develop this plugin to address more and more accessibility concerns in an abstract way, with the intention of completely covering the latter two cases. In the meantime, the plugin handles two very specific cases: and (2) One from the third category.

How It Works

The plugin listens to the `render` event for all `View.Component` objects that are created. Anytime a component is rendered, the plugin runs its own plugins (hereinafter referred to as "helpers") on the component. Each of these helpers is responsible for determining if any modifications are necessary in order for the component's HTML to meet accessibility standards and then carrying out those changes in the DOM. This behavior is done automatically as a part of the sidecar framework, so you do not need to do anything to start using it.

Sometimes, a component that you write will modify the HTML after it has been rendered. The plugin has no means of becoming aware of these changes to the HTML and you will have to tell it to look for rules to apply. One example is found in`View.Fields.Base.ActionmenuField`.

When the user selects all records in a list view, an alert is flashed indicating that all visible records are now selected. Within this alert, a link can be clicked to select all records, even those that are not currently visible. A new `onclick` event listener is registered for this link. And because this link is added to the DOM after the component is rendered, the author of the component must make sure that the new HTML meets accessibility requirements. Here is how that is done:

```
var $el = $('a#select-all');
```

```
$el.on('click', function() {...});
app.accessiblity.run($el, 'click');
```

This will only run the `click` helper. If you want to run all helpers, then call `app.accessiblity.run($el)` (without the second parameter). But be aware that some helpers only support `View.Component` objects, while others support either `View.Component` objects or jQuery DOM elements. So running all helpers on a jQuery DOM element may fail, or at least fail to apply some accessibility rules as expected.

When the logger is configured for `debug` mode, messages are logged indicating which helpers are being run and which helpers could not be run when intended.

Plugins

A plugin (or helper) is a module that applies an accessibility rule to a `View.Component` (or jQuery DOM element). At runtime, these helpers can be found in `app.accessibility.helpers` and implement a `run` method. The `run` method takes a `View.Component` (or jQuery DOM element) and then checks its HTML to determine if anything needs to be done in order to make the HTML compliant with accessibility standards related to the rule or task with which the helper is concerned. If any changes are necessary, the helper then modifies the HTML to comply.

Click

The `click` helper is responsible for making an element compliant with accessibility standards when click events are bound to said element. Since this helper only deals with `onclick` events, it only inspects elements within the component that include `onclick` event listeners.

If the tag name cannot be determined for an element being inspected, then there is no way of knowing whether or not the element is accessible. Thus, the element is assumed to be compliant.

Inherently focusable elements are those elements that require no intervention. These elements include:

- button
- input
- select
- textarea

Conditionally focusable elements are those elements that require intervention under certain circumstances. In the case of `<a>` and `<area>` tags, these elements are compliant as long as they contain an `href` attribute. These elements include:

- a
- area

All other elements are not inherently focusable and require a `tabindex` attribute of `-1` if a `tabindex`attribute does not already exist. This helper adds `tabindex="-1"` to any elements within the component that are not compliant.

When the logger is configured for `debug` mode, messages are logged...

1. In the event that no `onclick` events were found within the component. Thus, no action is taken.
2. In the event that an element being inspected has no tag name.
3. To report the type of element being made compliant.
4. To report the type of element that is already found to be compliant.

Label

The `label` helper adds an aria-label to the form element found within the component. This helper only inspects elements that can be found via the component's `fieldTag` selector and is consdered a "best effort" approach.

This helper will only work on `View.Field` components since it is extremely unlikely for the `fieldTag` selector for`View.Layout` or `View.View` components to match form elements.

A form element is considered to be compliant if the `aria-label` attribute is already present or if its tag is not one that requires a label. These elements include:

- button
- input
- select
- textarea

This helper adds `aria-label="{label}"` to the element that needs to be made compliant. `View.Field.label` is the label that is assigned to the attribute. The component must be a `View.Component`. Plain jQuery DOM elements are not sufficient since they do not include a`label`property.

API

SUGAR.accessibility.init()

Initializes the accessibility module to execute all accessibility helpers on components as they are rendered. This is called by the application during bootstrapping.

SUGAR.accessibility.run()

Loads the accessibility helpers that are to be run and executes them on the component.

Arguments

Name	Type	Required	Description
component	View.Component/jQuery	true	The element to test for accessibility compliance.
helper	String/Array	false	One or more names of specified helpers to run. All registered helpers will be run if undefined.

Returns

Void

SUGAR.accessibility.whichHelpers()

Get the helpers registered on a specific element.

Name	Type	Required	Description
helper	String/Array	true	One or more names of specified helpers to run.

Returns

Array - The accessibility helpers that were requested. Filters out any named helpers that are not registered. All registered helpers are returned if no helper names are provided as a parameter.

SUGAR.accessibility.getElementTag()

Generates a human-readable string for identifying an element. For example, `a[name="link"][class="btn btn-link"][href="http://www.sugarcrm.com/"]`. Primarily used for logging purposes, this method is useful for debugging.

Arguments

Name	Type	Required	Description
$el	jQuery	true	The element for which the tag should be generated.

Returns

String - A string representing an element's tag, with all attributes. The element's selector, if one exists, is returned when a representation cannot be reasonably generated.