FORMS AND VALIDATION

SETTING UP VALIDATION ON FORMS

- Form, input, and validation tags are provided in Liferay to make creating forms easy and flexible.
- The `<aui:form>` taglib sets up the necessary code (HTML, JS), utilizes Liferay JavaScript APIs, calls the validation framework, and submits the form data to the back-end.
- The "input" taglibs (`<aui:input>`, `<liferay-ui:input-*>`, etc.) generate the necessary code (HTML, CSS, JS) to keep a consistent form UI.
  - These taglibs also initialize the input's state - e.g., `value`, `disabled`, `readonly`.
- The `<aui:validator>` taglib pairs with some input taglibs to provide validation rules for those inputs.
  - This way, you can ensure your users are entering proper data, formatted the way you expect, before it gets sent to the back-end for final validation and processing.

INPUT VALIDATION SUPPORT

- The inputs that support validators natively are:
  - `<aui:input>`
  - `<aui:select>`
  - `<liferay-ui:input-date>`
  - `<liferay-ui:input-search>`

CREATING A FORM

- Here is an example of a simple form.

```
<aui:form action="<%= myActionURL %>" method="post" name="myForm">
    <aui:input label="My Text Input" name="myTextInput" type="text" value='
    <%= "My Text Value" %>' />

    <aui:button type="submit" />
</aui:form>
```

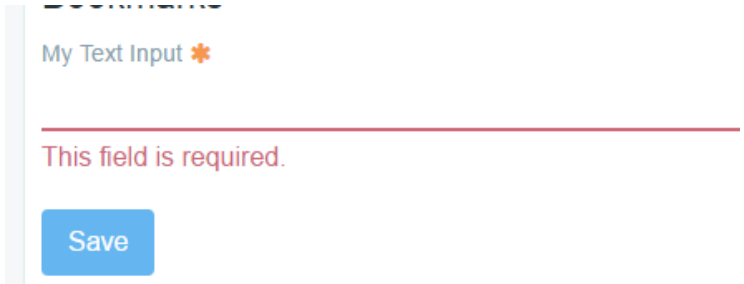
- Nothing too complex about this, but it ends up outputting all the “magic” that’s needed to pass the data to the back-end.

ADDING VALIDATION

- What if we want to ensure the user enters the required data? We add validators.

```
<aui:input label="My Text Input" name="myTextInput" type="text" value='
<%= "My Text Value" %>'>
    <aui:validator name="required" />
</aui:input>
```

- This will force the user to enter something into the input before the form is submitted.



ADDING VALIDATION

- Let's say we wanted to restrict the value to a number between 0 and 10.
- We can add additional validators. Each one will need to pass in order for the form to submit.

```
<aui:validator name="required" />
<aui:validator name="number" />
<aui:validator name="range">[0,10]</aui:validator>
```

- We can even customize the error message.

```
<aui:validator errorMessage="Please enter how many fingers you have."
name="range">[0,10]</aui:validator>
```

- There's a wide variety of built-in validators available. Check the full documentation for details: *https://dev.liferay.com/develop/tutorials/-/knowledge_base/6-2/using-the-alloyui-validator-tag#available-validation-rules*

DYNAMICALLY/CONDITIONALLY REQUIRING AN INPUT

- Sometimes you'll want to validate an input based on the value of another input.
- You can do this by checking for that condition in a JavaScript function in the `required` validator's body.

```
<aui:input label="My Checkbox" name="myCheckbox" type="checkbox" />

<aui:input label="My Text Input" name="myTextInput" type="text">
    <aui:validator name="required">
        function() {
            return AUI.$('#<portlet:namespace />myCheckbox').prop('checked');
        }
    </aui:validator>
</aui:input>
```

CUSTOM VALIDATION

- So far, we've only seen validator rules that come with AUI.
- Suppose you need something a little more custom or advanced.
- You can write your own validator and optionally supplement it with built-in validators.

```
<aui:input label="Email" name="email" type="text">
    <aui:validator name="email" />
    <aui:validator errorMessage="Only emails from @example.com are allowed."
    name="custom">
        function(val, fieldNode, ruleValue) {
            var regex = new RegExp(/@example\.com$/i);
            return regex.test(val);
        }
    </aui:validator>
</aui:input>
```

- This example runs the regular `email` validator and your `custom` domain validator.

ADD ADDITIONAL VALIDATION VIA JAVASCRIPT

- Sometimes you need to dynamically add additional validation after the page has rendered.
- Maybe some additional inputs were added to the DOM via an AJAX request.
- To do this, you'll need to access the `Liferay.Form` object.

```
<aui:script use="liferay-form">
    var form = Liferay.Form.get('<portlet:namespace />myForm');
    // ...
```

- With the `Liferay.Form` object, you can now `get()` and `set()` `fieldRules`.
- `fieldRules` are the JavaScript equivalent of all the validators attached to the form.

CUSTOM VALIDATOR EXAMPLE

```
<aui:script use="liferay-form">
    var form = Liferay.Form.get('<portlet:namespace />myForm');
    var oldFieldRules = form.get('fieldRules');
    var newFieldRules = [
        {
            body: function (val, fieldNode, ruleValue) {
                return (val !== '2');
            },
            custom: true,
            errorMessage: 'must-not-equal-2',
            fieldName: 'fooInput',
            validatorName: 'custom_fooInput'
        },
        {
            fieldName: 'fooInput',
            validatorName: 'number'
        }
    ];
    var fieldRules = oldFieldRules.concat(newFieldRules);
    form.set('fieldRules', fieldRules);
</aui:script>
```

MANUAL VALIDATION

- You may need to execute validation on an input based off of some event not typical of user input or validate a related input at the same time.
- You can do this by accessing the `formValidator` object and calling the `validateField()` method, passing the input's `name`.

MANUAL VALIDATION EXAMPLE

```
<aui:input label="Old Title" name="oldTitle" type="text">
    <aui:validator errorMessage="The New Title cannot match the Old Title"
    name="custom">
        function(val, fieldNode, ruleValue) {
            <portlet:namespace />checkOtherTitle('<portlet:namespace />newTitle
            // check if old and new titles are a match
            return !match;
        }
    </aui:validator>
</aui:input>
<aui:input label="New Title" name="newTitle" type="text">
    <!-- same custom validator -->
</aui:input>
<aui:script use="liferay-form">
    function <portlet:namespace />checkOtherTitle(fieldName) {
        var formValidator = Liferay.Form.get('<portlet:namespace />myForm').
        formValidator;

        formValidator.validateField(fieldName);
    }
</aui:script>
```

Notes: