Magento U

Student Guide

Fundamentals of Magento 2 Development: Unit 3.5

magento.com/training/overview

Version 2.1

Magento® U

training@magento.com

magento.com/training/overview

magento.com/training/catalog/certification

Contents – Unit 3.5

About This Guide

This guide uses the following symbols in the notes that follow the slides.

Symbol	Indicates…
	A note, tip, or other information brought to your attention.
	Important information that you need to know.
	A cross-reference to another document or website.
	Best practice recommended by Magento

1. Unit 3.5 Home Page

1.1 Fundamentals of Magento 2 Development - Unit 3.5

Notes:

Fundamentals of Magento 2 Development v. 2.1

Software version: Magento 2 v.2.1.0

1.2 Unit 3.5 Home Page

Notes:

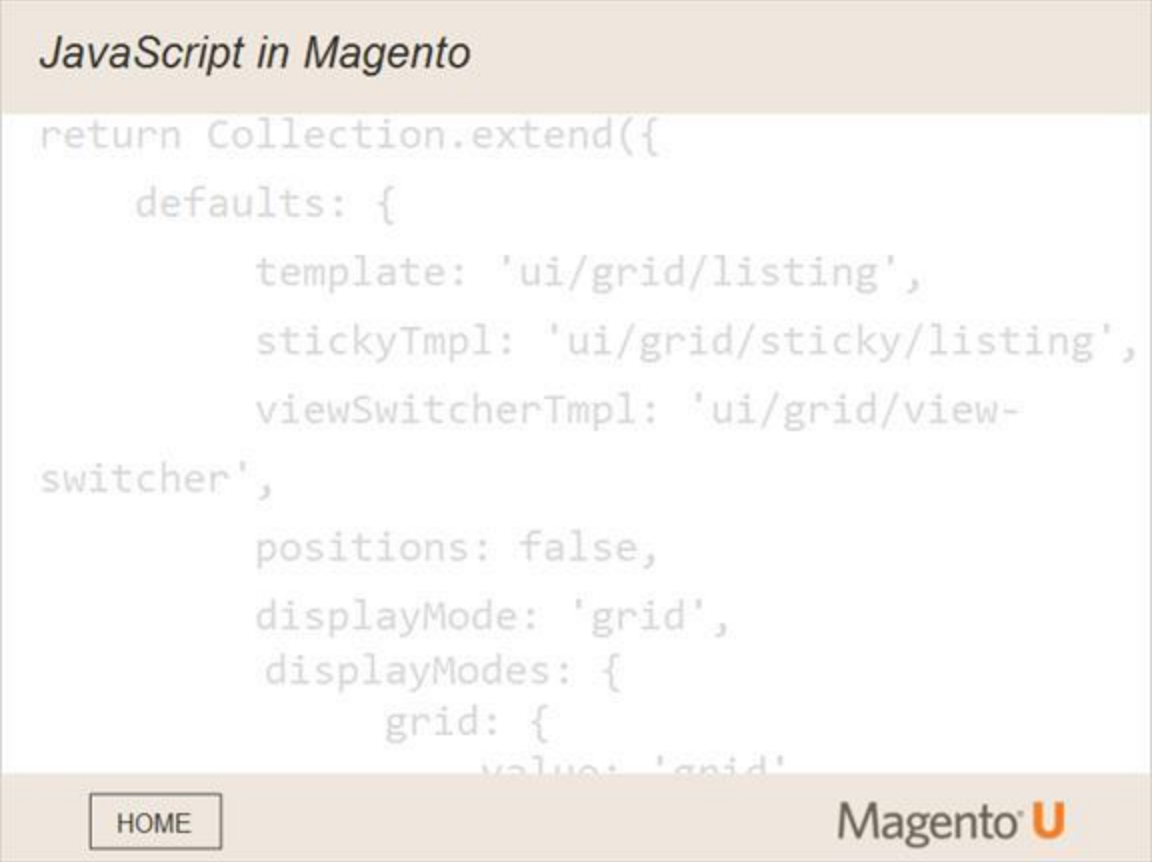
Unit 3.5 of the Magento 2 Fundamentals course contains three modules.

The suggested flow of the course is indicated by the arrows.

However, you are free to access any of the modules, at any time, by simply clicking the Home button on the bottom of each slide.

2. JavaScript in Magento

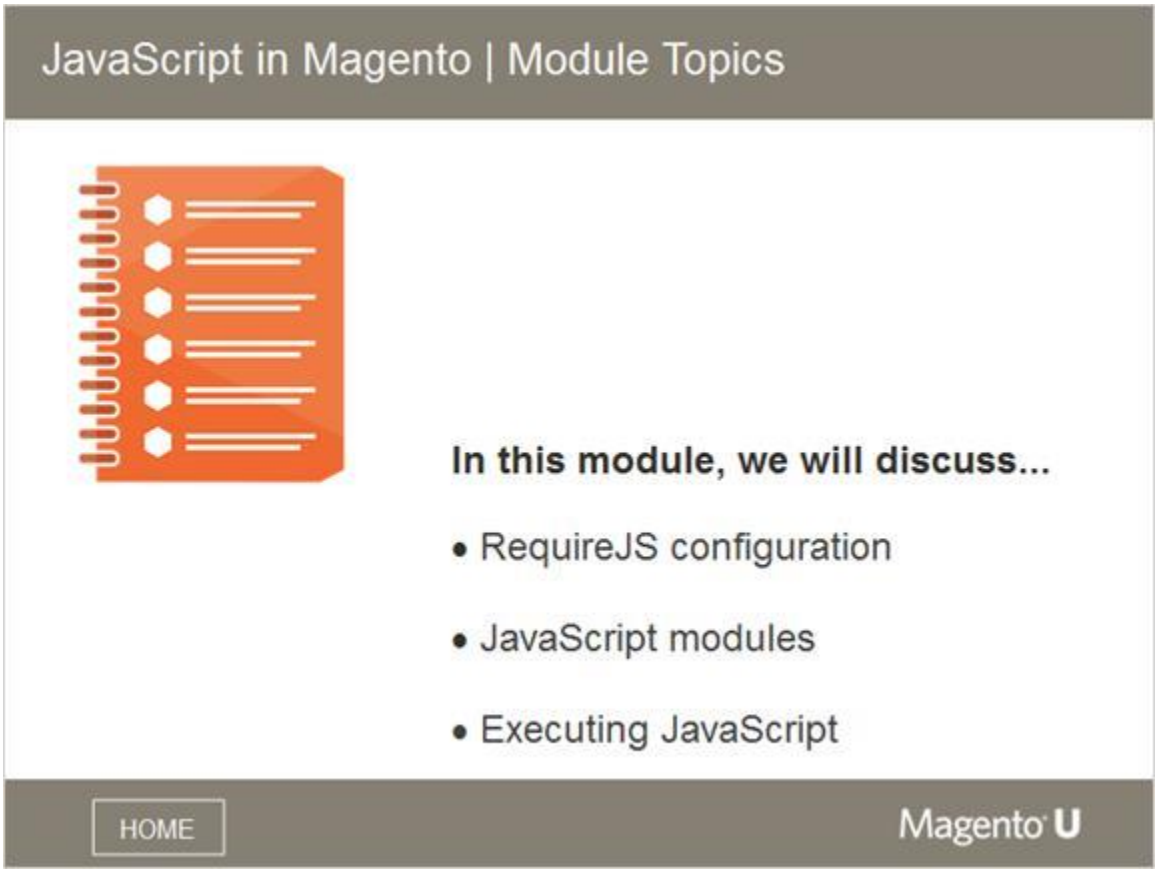
2.1 JavaScript in Magento



Notes:

In this module, we will discuss how JavaScript is used in Magento 2.

2.2 JavaScript in Magento | Module Topics



Notes:

Magento 2 uses JavaScript (JS) in specific ways that differ from the use of JS in other applications.
In this module, we will focus more general topics, such as RequireJS configuration, JS modules, and JS execution,

Note: We are not teaching JS in this module - only the key aspects of how it is used in Magento 2. To refresh or deepen your knowledge of JS, there are many resources available on the web, such as the free course offered by w3schools.com: http://www.w3schools.com/js/

2.3 JavaScript in Magento | Overview

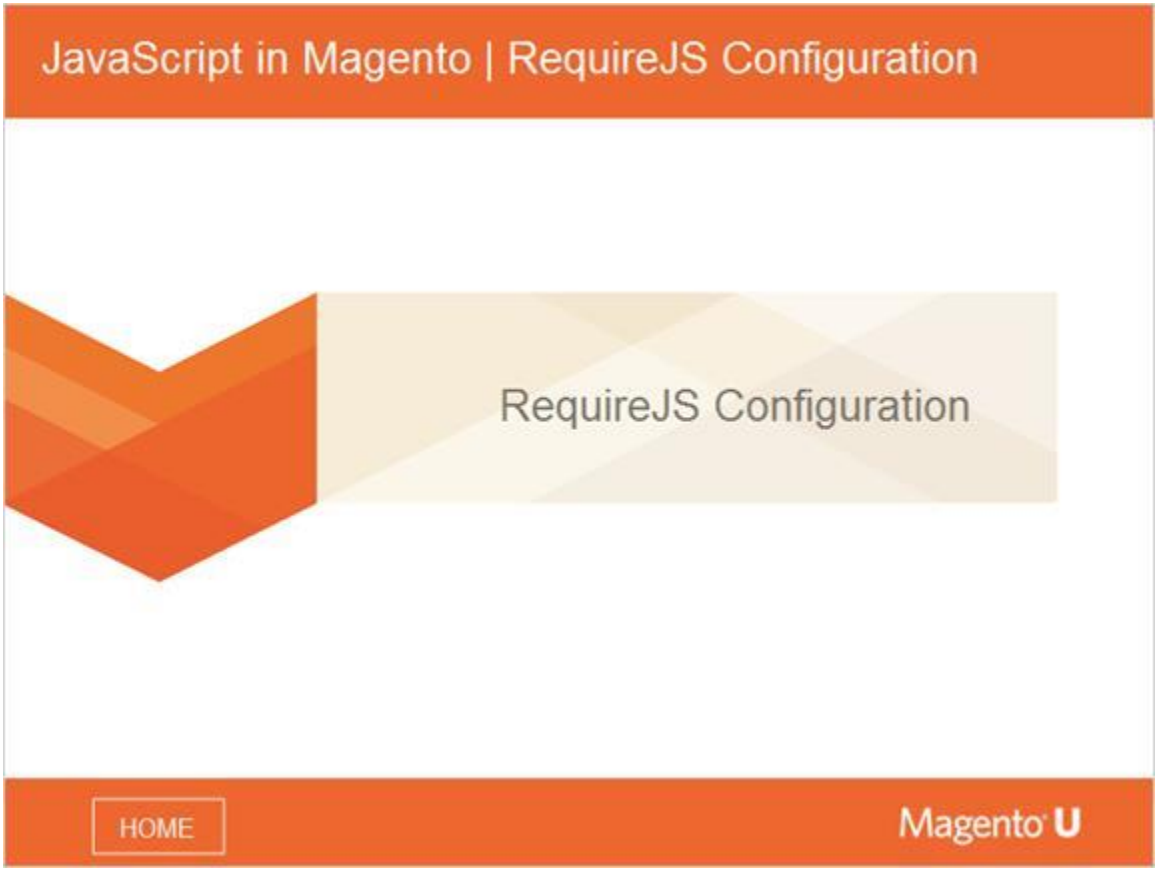
Notes:

Magento has its own JavaScript framework, which we will briefly overview in this section.

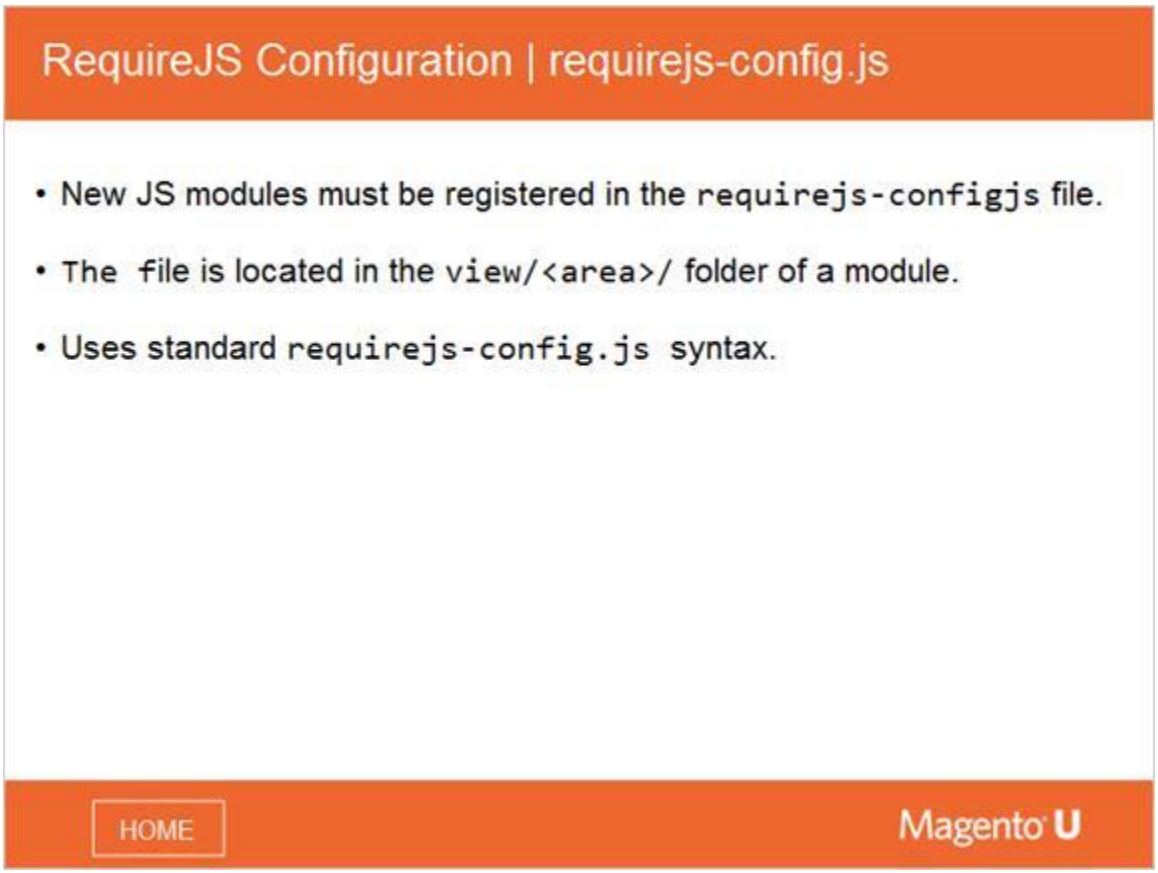
The key to JavaScript file organization in Magento2 is `RequireJS` and `AMD` modules.

Refer to the official RequireJS documentation if you would like additional guidance on this topic:
http://requirejs.org

2.4 JavaScript in Magento | RequireJS Configuration



2.5 RequireJS Configuration | requirejs-config.js



Notes:

The `Requirejs-config` file is a standard tool in the RequireJS framework, which allows for the declaration of all modules used on a page.

Magento2 generates a generic `requirejs-config.js` file based on the `requirejs-config` files from each module. This file should be located in the `view/<area>` folder of a module, and should follow a standard `requirejs-config` syntax.

We will see an example on the next slide.

2.6 RequireJS Configuration | requirejs Config Example

RequireJS Configuration | requirejs Config Example

```
/**
 * Copyright © 2016 Magento. All rights reserved. See COPYING.txt for details.
 */

var config= {
    map: {
        '*': {
            compareItems:           'Magento_Catalog/js/compare',
            compareList:            'Magento_Catalog/js/list',
            relatedProducts:        'Magento_Catalog/js/related-products',
            upsellProducts:         'Magento_Catalog/js/upsell-products',
            productListToolbarForm: 'Magento_Catalog/js/product/list/toolbar',
            catalogGallery:         'Magento_Catalog/js/gallery',
            priceBox:               'Magento_Catalog/js/price-box',
            priceOptionDate:        'Magento_Catalog/js/price-option-date',
            priceOptionFile:        'Magento_Catalog/js/price-option-file',
            priceOptions:           'Magento_Catalog/js/price-options',
            priceUtils:             'Magento_Catalog/js/price-utils',
            catalogAddToCart:       'Magento_Catalog/js/catalog-add-to-cart'
        }
    }
};
```

HOME

Magento U

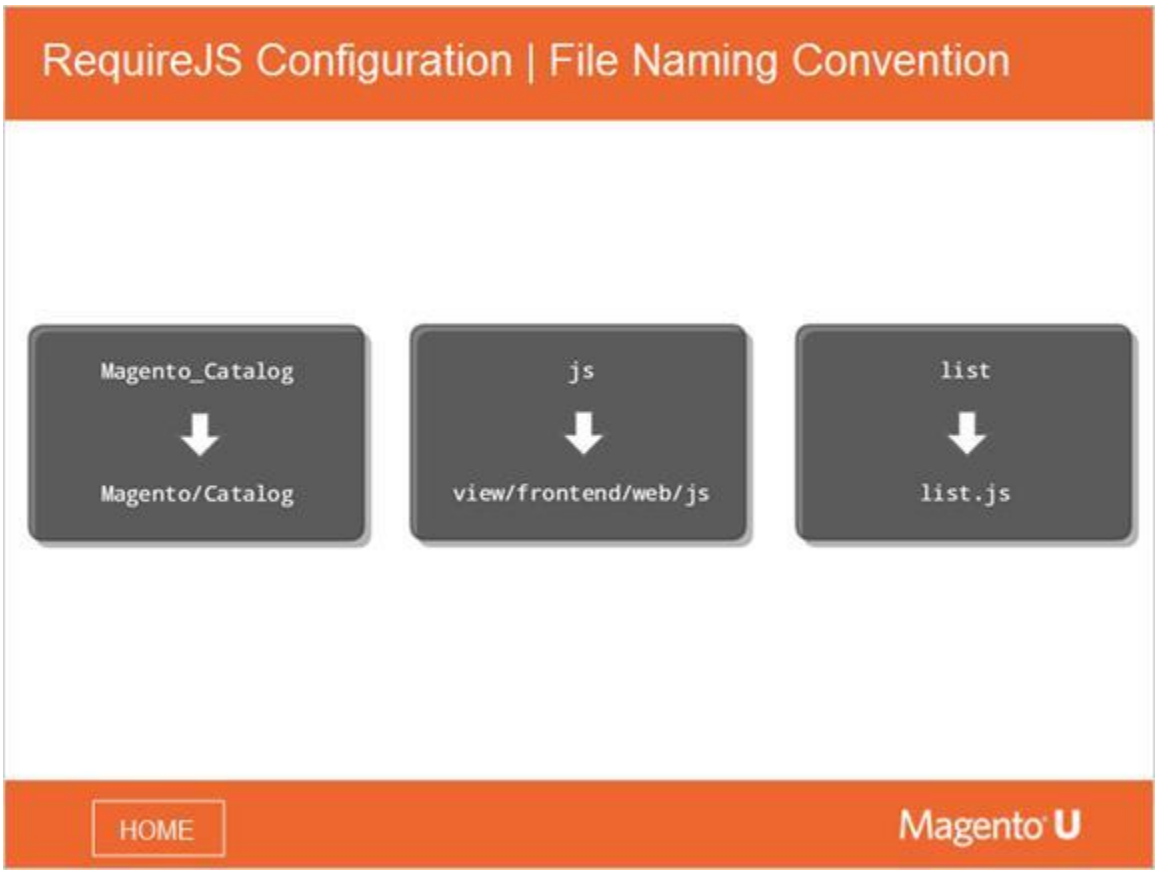
Notes:

This is an example of a typical `requirejs-config` file structure. A developer should follow this structure to add a new JavaScript module.

Each module declaration is presented as **Name** (for example, `compareItems`) and **Path** (`Magento_Catalog/js/compare`). We will see later how the path is translated into a real path to a physical file.

Name is used inside JavaScript modules for communication and execution.

2.7 RequireJS Configuration | File Naming Convention



Notes:

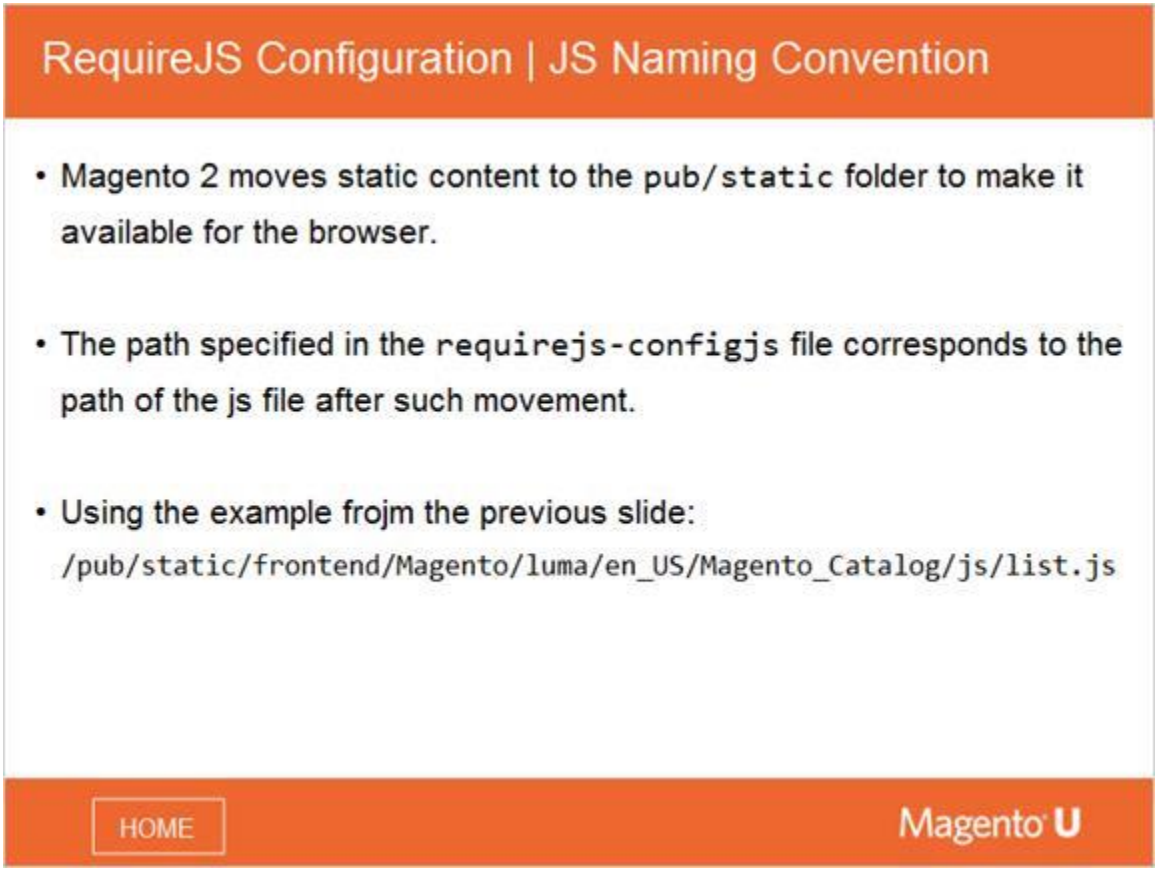
This slide shows how a path from the `requirejs-config` file is translated into a real path.

The first chunk of a path defines a module (`Magento/Catalog` in this case).

The second chunk is usually, but not always, a subfolder of the **web** folder. So, if the `requirejs-config` file is located in the `view/frontend` folder, the second chunk defines a relative path under that folder to the directory which contains the JS file.

The final chunk is a JavaScript file name, without ".js".

2.8 RequireJS Configuration | JS Naming Convention

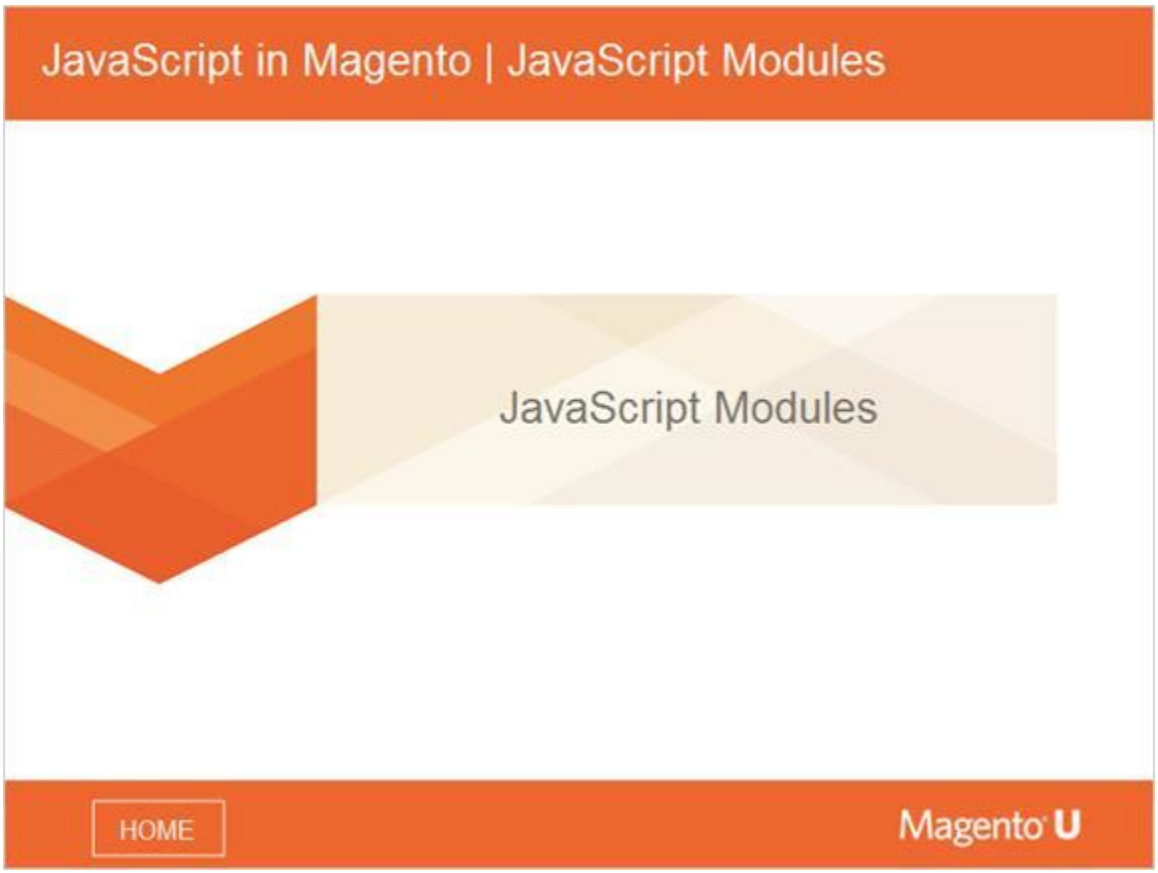


Notes:

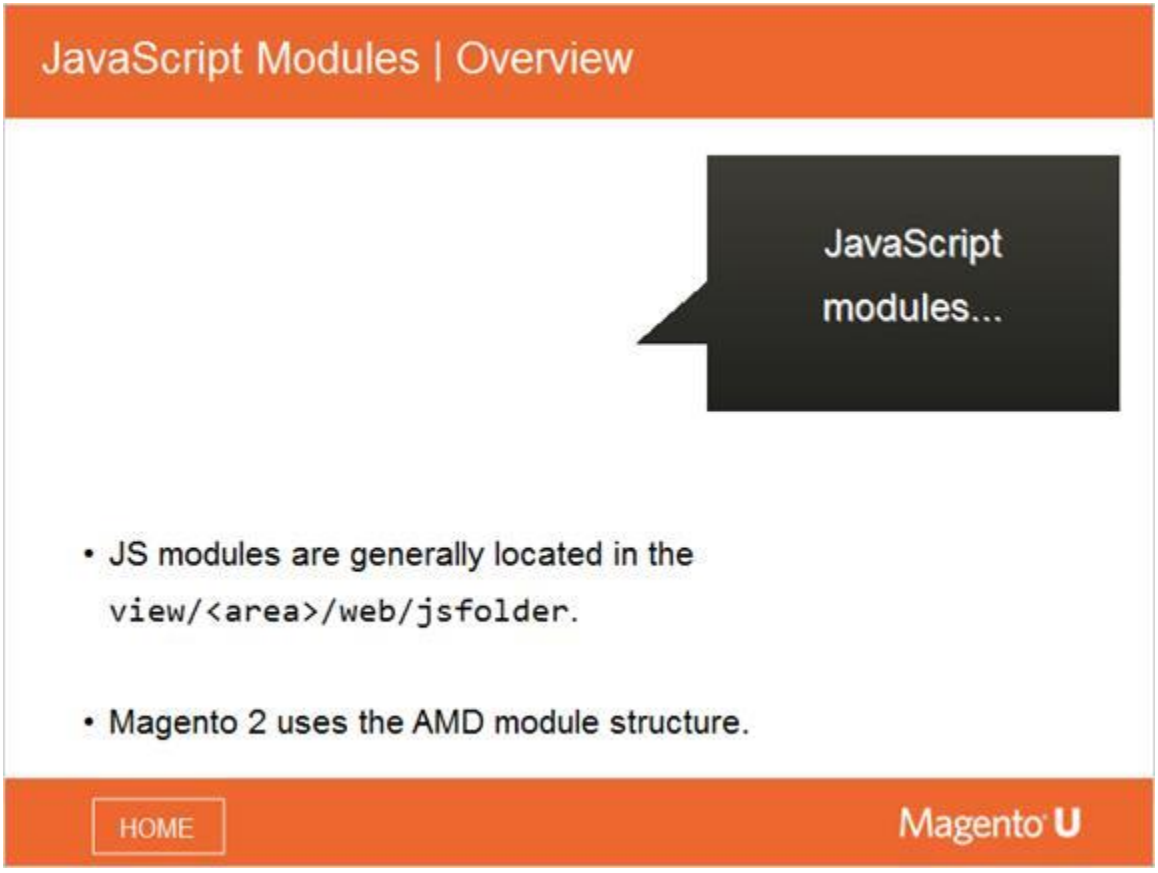
There is a reason why a `requirejs-config` path may look a bit unusual… It is because of the way Magento 2 displays static content.

The static content has to be moved to the `pub/static` folder, in a process called **static content deploy**. Afterwards, the path to the JavaScript file will be the same as in the `requirejs-config`, as shown in the third bullet of the slide.

2.9 JavaScript in Magento | JavaScript Modules



2.10 JavaScript Modules | Overview



Notes:

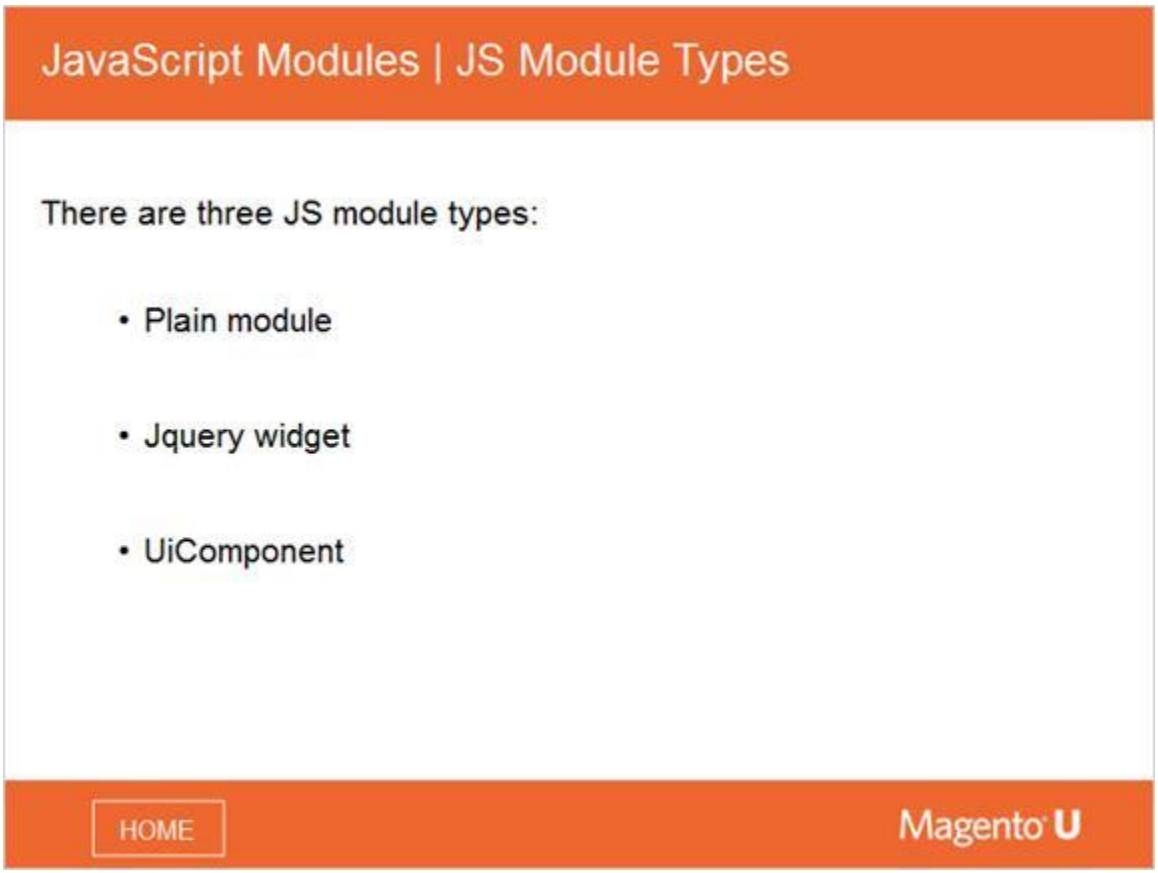
Normally, JS modules are located in the `view/<area>/web/jsfolder`, but they can be located in the `view/<area>/web/js`.

There is an informal rule that regular JS modules are in the `web/jsfolder` and `uiComponentsjsmodules` (covered later) are in the `web/js/view` folder.

As mentioned previously, Magento 2 uses AMD modules.

Refer to the documentation to gain a better understanding:
http://requirejs.org/docs/whyamd.html

2.11 JavaScript Modules | JS Module Types



Notes:

There are three types of JavaScript modules in Magento 2:

- **Plain module:** A custom JS file that follows AMD rules. It does not have any specific pre-defined structure (other than that dictated by AMD).
- **Jquery widgets:** Currently very popular, and are widely used in Magento 2. These widgets must have a specific structure in order to follow AMD requirements. (We will see an example later.)
- **UiComponent:** A Magento2 invention; we will cover them in the next section.

2.12 JavaScript Modules | Plain JS Module

JavaScript Modules | Plain JS Module

```
define([
        'jquery',
        'Magento_Customer/js/model/authentication-popup',
        'Magento_Customer/js/customer-data'
    ],
    function ($, authenticationPopup, customerData) {
        'use strict';

        return function (config, element) {
            $(element).click(function (event) {
                var cart = customerData.get('cart'),
                customer = customerData.get('customer');
                event.preventDefault();
                if (!customer().firstname && cart().isGuestCheckoutAllowed === false)
                {
                    authenticationPopup.showModal();
                    return false;
                }
                location.href = config.checkoutUrl;
            });
        };
    }
);
```

`define` section with a list of dependencies

returns a function with 2 params: config, element

HOME

Magento U

Notes:

This is an example of the plain JS module. It consists of two major parts. The **define** statement, which declares all dependencies and returns the statement. Because of the way in which Magento 2 executes the JavaScript, they usually return a function with two parameters: `element` and `config`.

Technically, it could be anything, but in this case the module must be executed using the "`require`" function (as a regular `requirejs` module), which is not typical for Magento 2. We will cover the ways to execute JS modules later in this section.

2.13 JavaScript Modules | jquery Widget

JavaScript Modules | jquery Widget

```
define([
    "jquery",
    "jquery/ui"
], function($){
    "use strict";

    $.widget('mage.shoppingCart', {
            // Code that is specific to Jquery widgets.
            //Config and element are available as this.config and this.element
    });

    return $.mage.shoppingCart;
});
```

HOME

Magento U

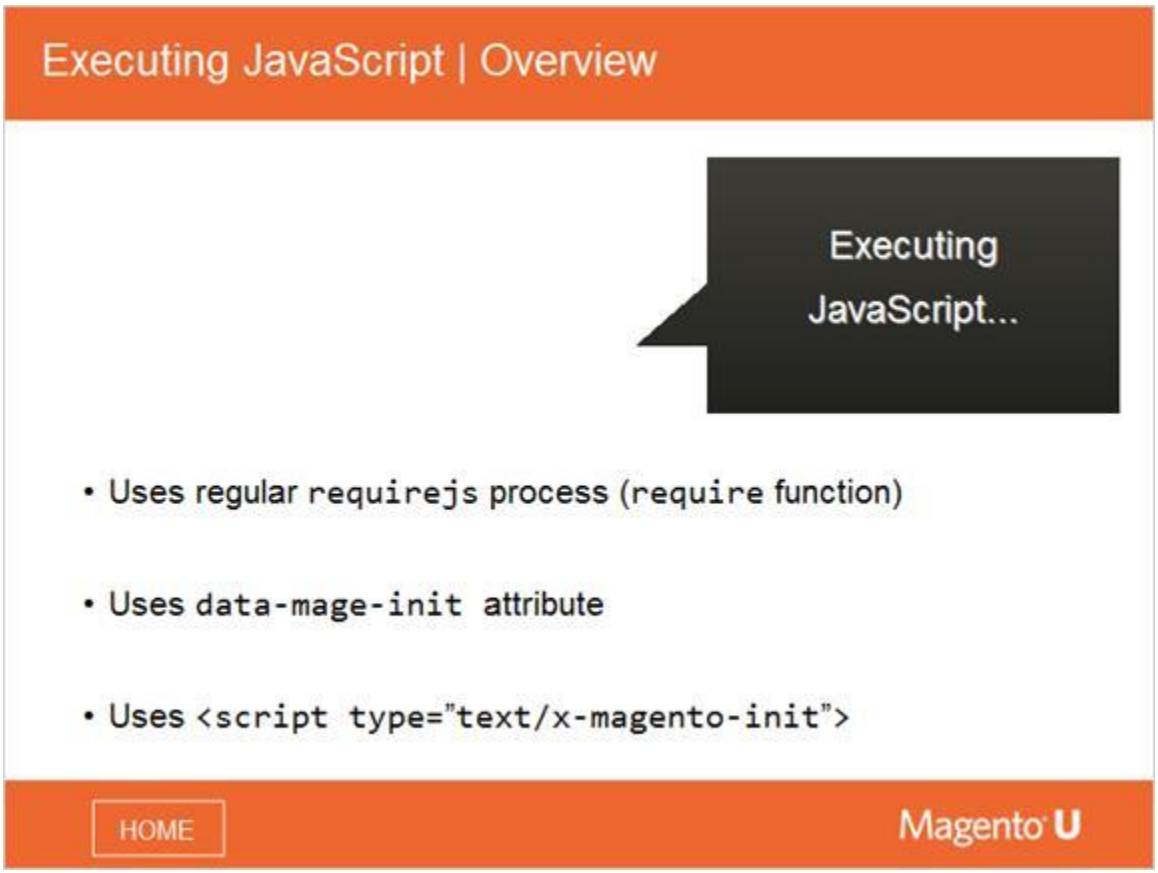
Notes:

Here is a `Jquery` widget example. As you can see, it has the same `define` statement as a plain module, but the return part is slightly different. It uses `config` and `element` parameters, but now they are hidden inside of its structure.

This is a typical approach -- to create (or extend from an existing) `jquery` widget by passing an object with parameters and functions into the `$.widget()` function.

2.14 JavaScript in Magento | Executing JavaScript

2.15 Executing JavaScript | Overview



Notes:

We will now take a look at how JavaScript is executed in Magento 2.

There are three ways to execute a JS module in Magento 2:

- Use a regular `requirejs` process with a "`require`" function.
- Use special `data-mage-init` attribute of a specific DOM element. (Remember the `config` and `element` parameters in the plain module? The "`element`" corresponds to the DOM element of the `data-mage-init` attribute, which ensures a module is executed "on" a specific DOM element.)
- Sometimes, you don't need a specific DOM element connected to a `javascriptor`; you need multiple elements. In this case, you should use the `<script type="text/x-magento-init">` approach.

2.16 Executing JavaScript | Regular require() Call

Executing JavaScript | Regular require() Call

```
<script>
   require(["jquery", "jquery/ui", "Magento_Swatches/js/swatch-renderer"], function ($)
   {
       $('.swatch-layered.color')
           .find('[option-type="1"],
                  [option-type="2"],
                  [option-type="0"],
                  [option-type="3"]')
           .SwatchRendererTooltip();
   });
</script>
```

HOME

Magento U

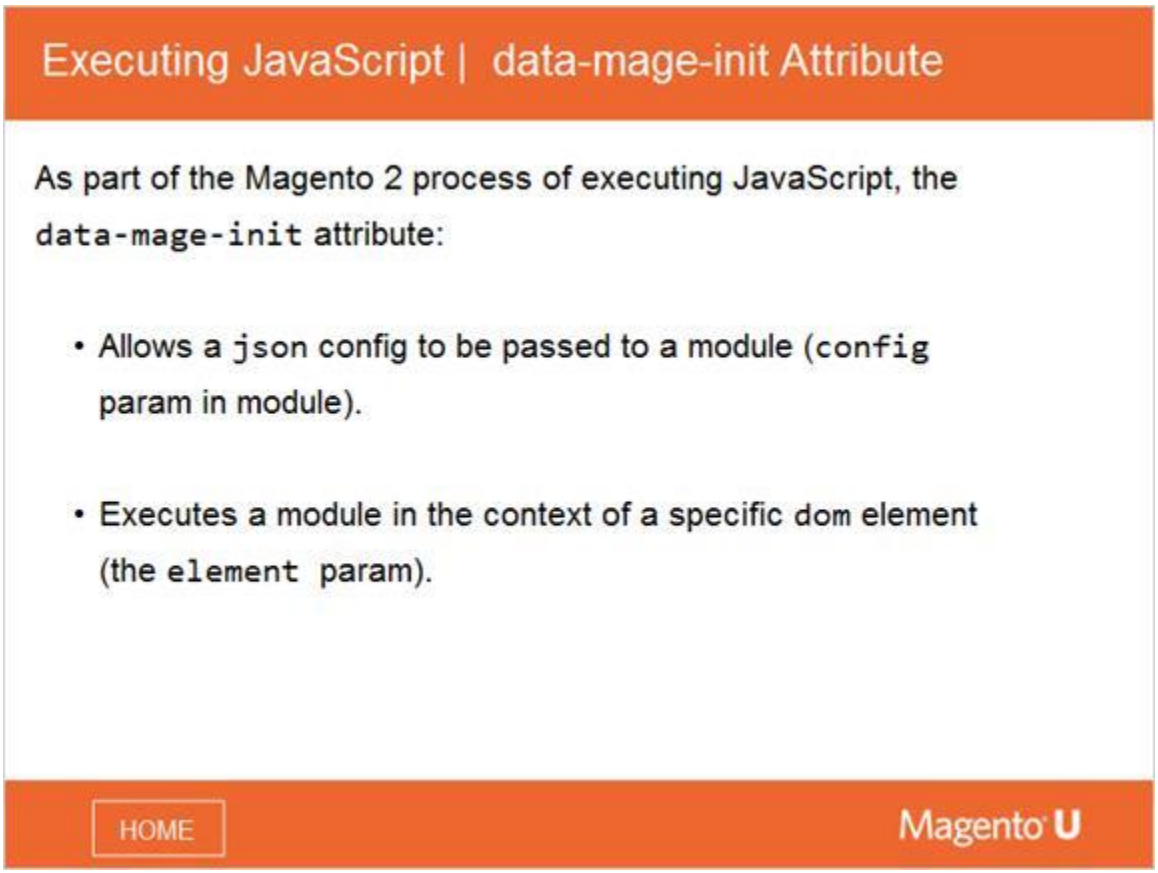
Notes:

This is an example of the "`require`" function usage.

It is standard `requirejs` practice. You call the `require` function with two parameters: first, a list of dependencies, and second, a function that has those dependencies as a parameter.

You can manipulate a module's properties and methods inside of a function. Refer to the `requirejs` documentation for more details.

2.17 Executing JavaScript | data-mage-init Attribute

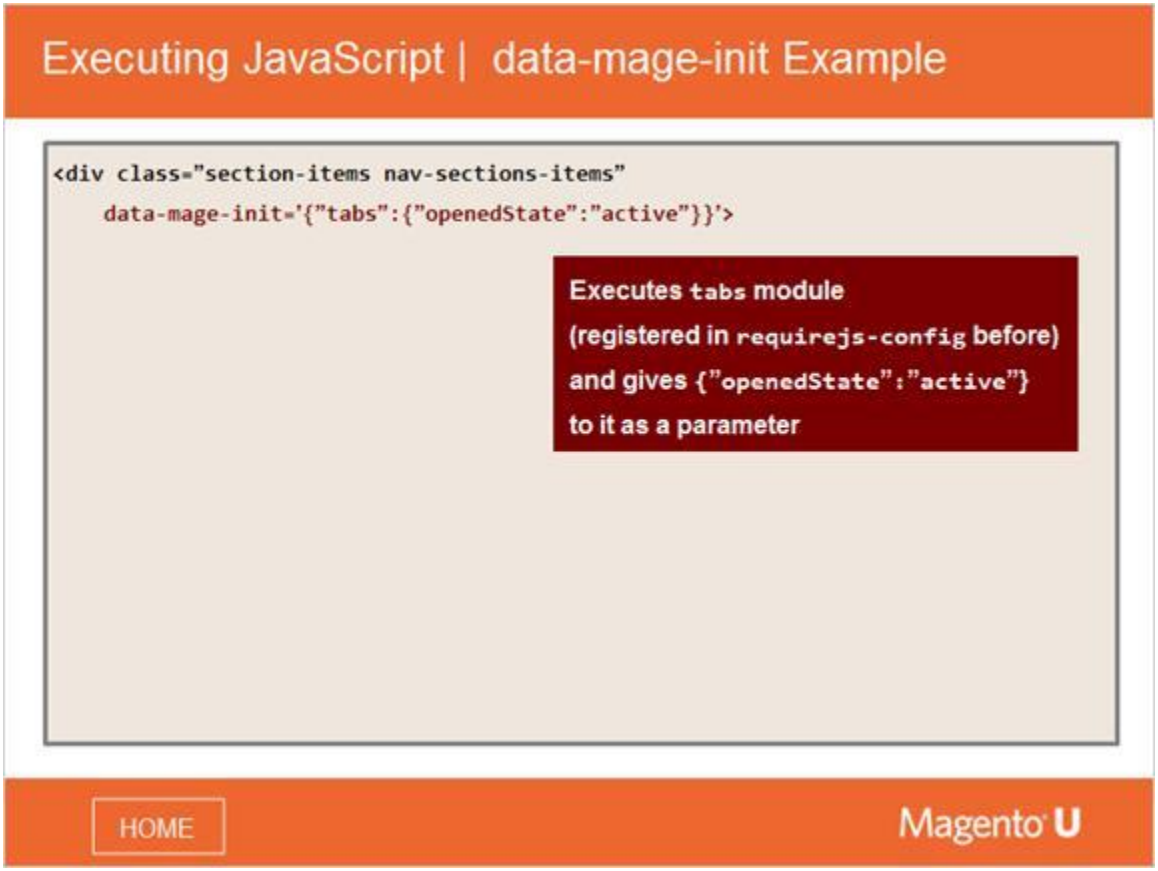


Notes:

The `data-mage-init` attribute is a Magento 2 -specific way of executing JavaScript. It works in a very simple way.

Assume you have a module that does something with a DOM element (like `<div>`). We define the `data-mage-init` attribute and set its value to the `json` object , which lists JS modules and configuration for them. That configuration will later be passed as a “config” parameter to a plain JS module.

2.18 Executing JavaScript | data-mage-init Example

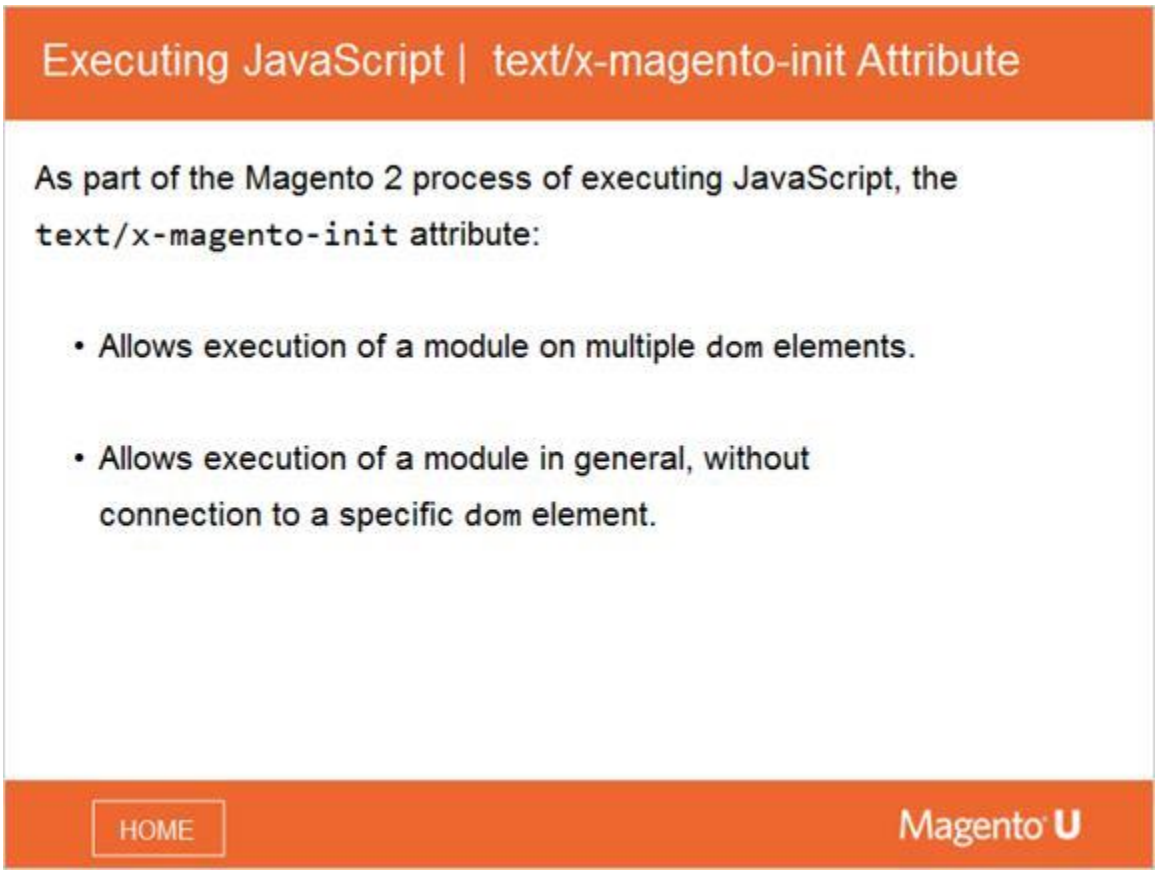


Notes:

This example shows a typical way of executing a JS module using the `data-mage-init` attribute.

Notice its value -- a `json` object that lists all the modules ("tabs" in our example, but it could be more than one module) and `config` for each module ({"`openedState`": "`active`"} in our example).

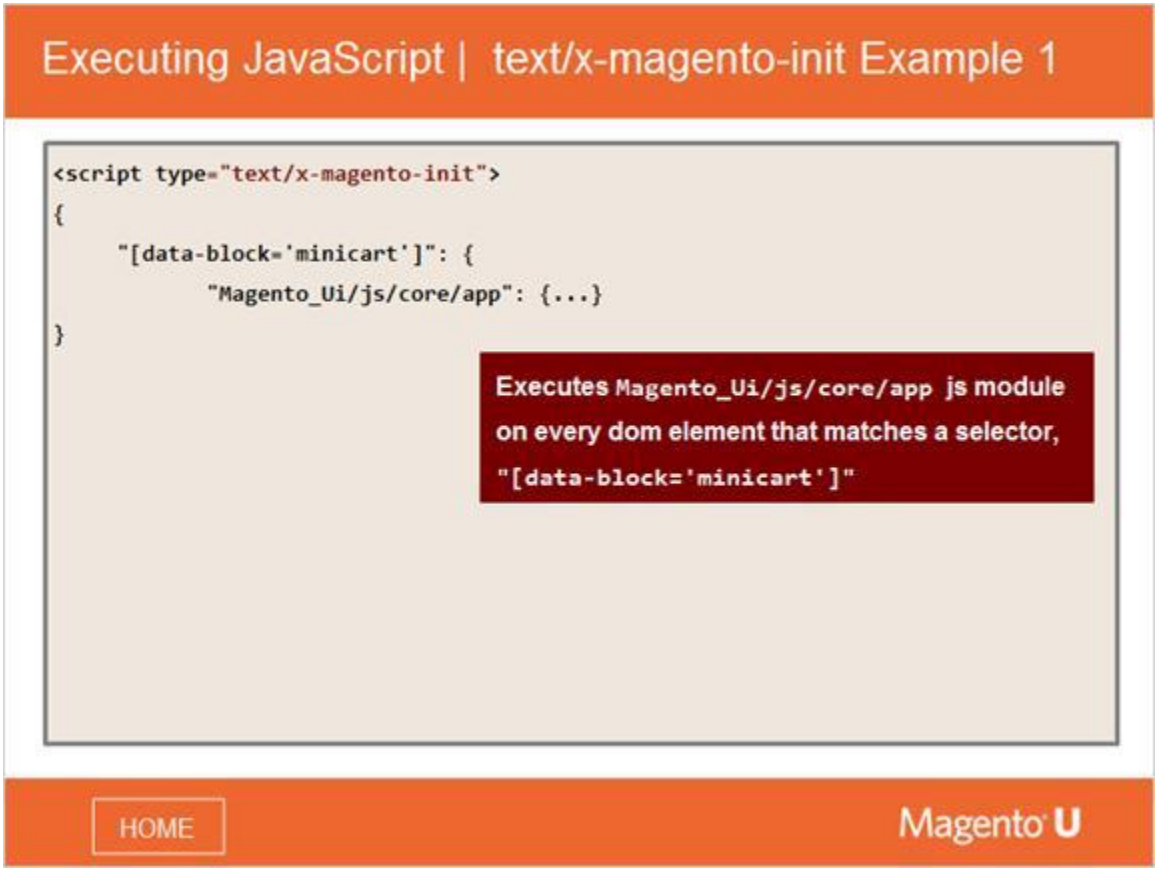
2.19 Executing JavaScript | text/x-magento-init Attribute



Notes:

The `text/x-magento-init` approach allows the execution of a JS module without connecting it to a specific DOM-node or multiple nodes.

2.20 Executing JavaScript | text/x-magento-init Example 1



Notes:

This code example demonstrates how to connect a module to multiple nodes (all that match a selector `[data-block='minicart']`).

Its syntax is pretty straightforward, and matches `data-mage-init` syntax -- with a list of modules as keys and their config as values.

2.21 Executing JavaScript | text/x-magento-init Example 2

Executing JavaScript | text/x-magento-init Example 2

```
<script type="text/x-magento-init">
{
    "*": {
        "Magento_Customer/js/customer-data": {…}
        }
    }
</script>
```

Executes `Magento_Customer/js/customer-data` js module without connection to a specific dom element

HOME

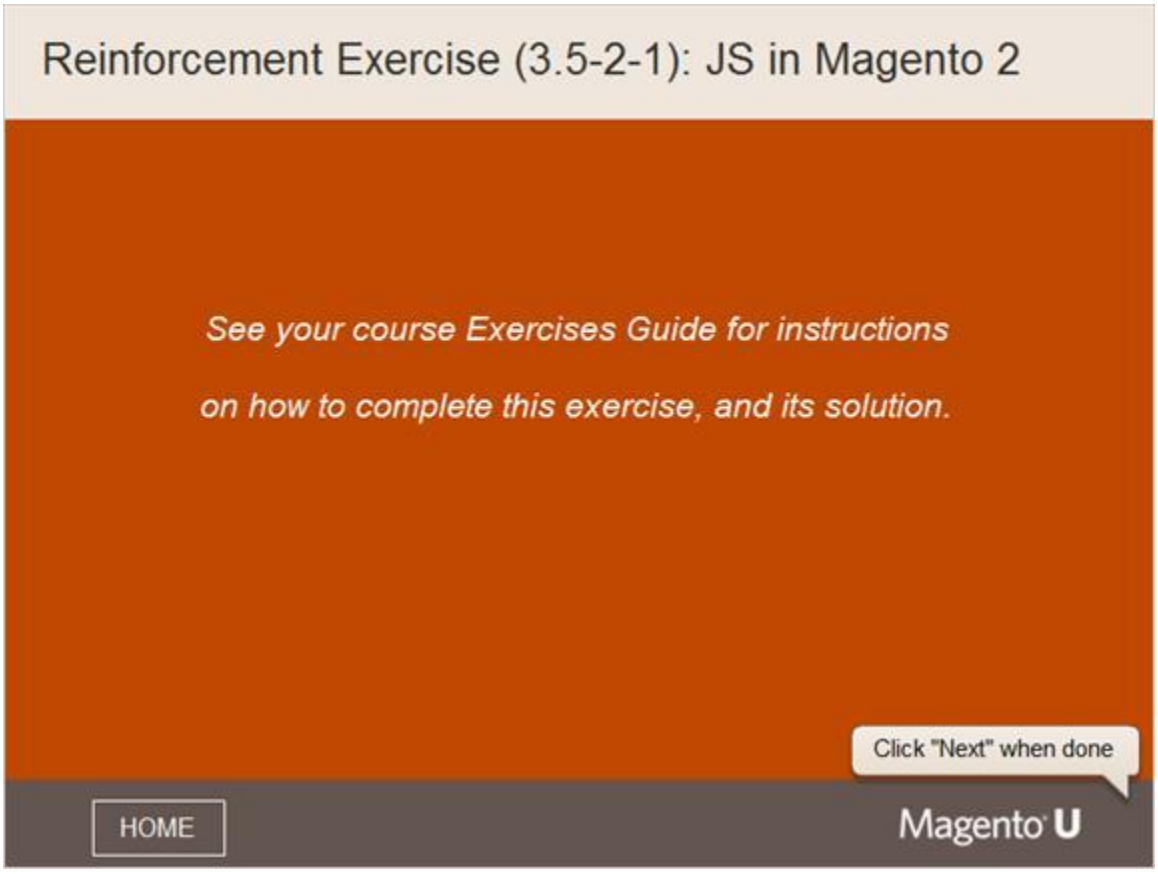
Magento U

Notes:

And here you see an example of a JS module execution, without connection to a specific DOM-node.

All you have to do is specify '*' instead of a selector, as in the previous example. Otherwise, the syntax is the same.

2.22 Reinforcement Exercise (3.5-3-1): JS in Magento 2

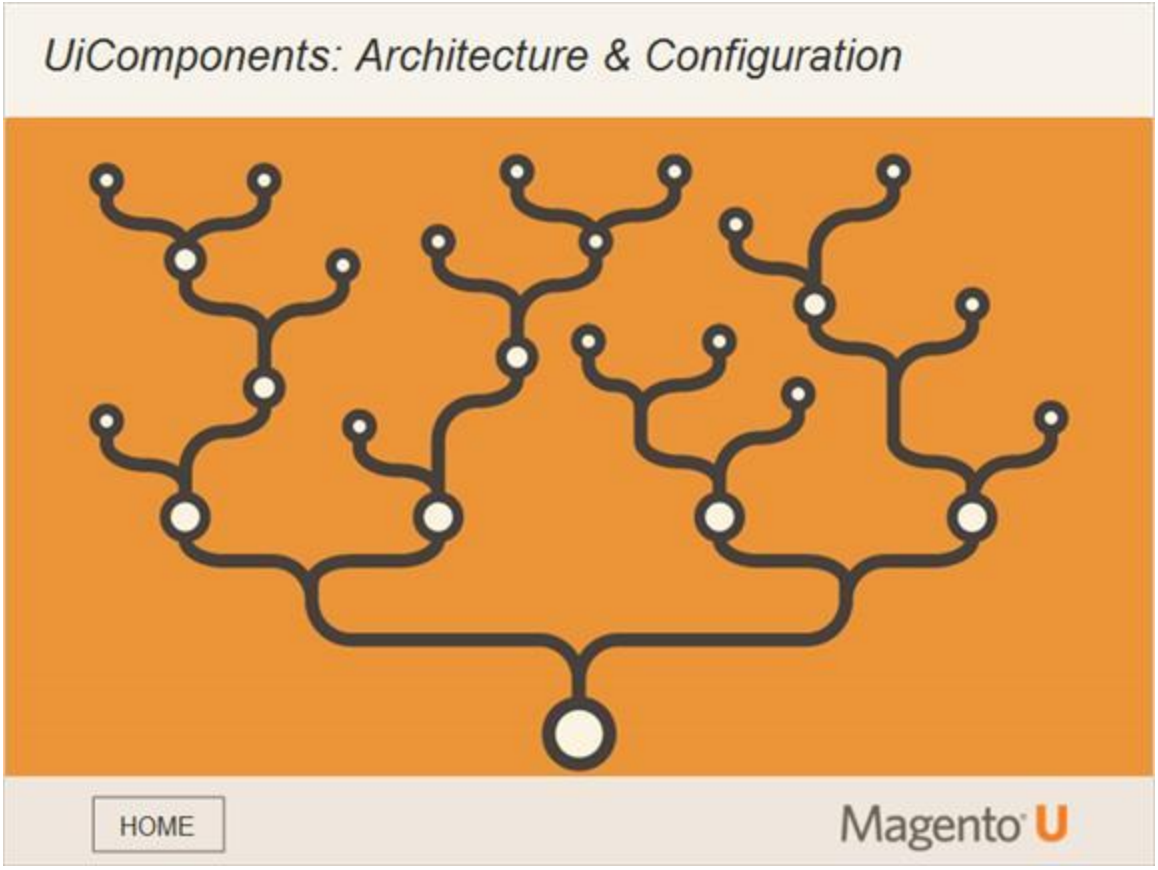


Notes:

See your course Exercises Guide for instructions on how to complete this exercise, and the solution.

3. UI Components

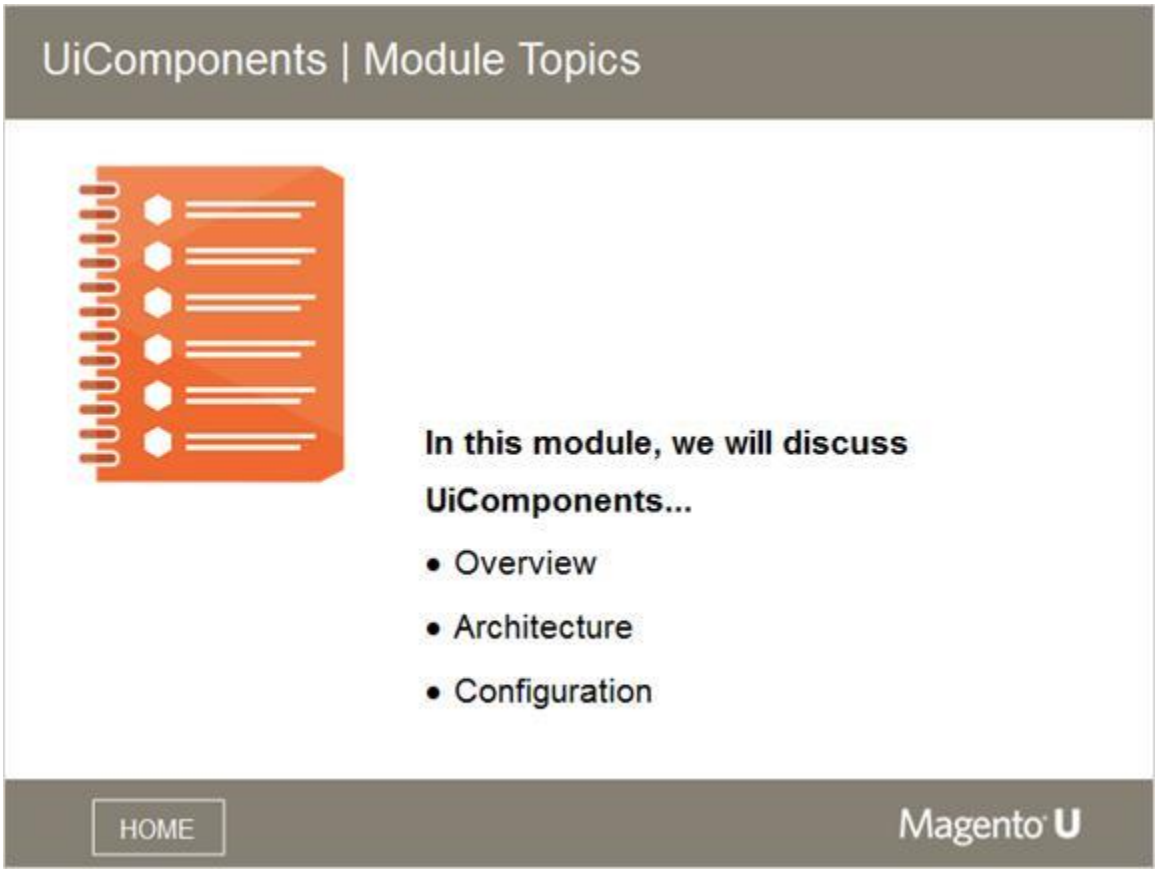
3.1 UiComponents: Architecture & Configuration



Notes:

This module discusses UI components in more depth.

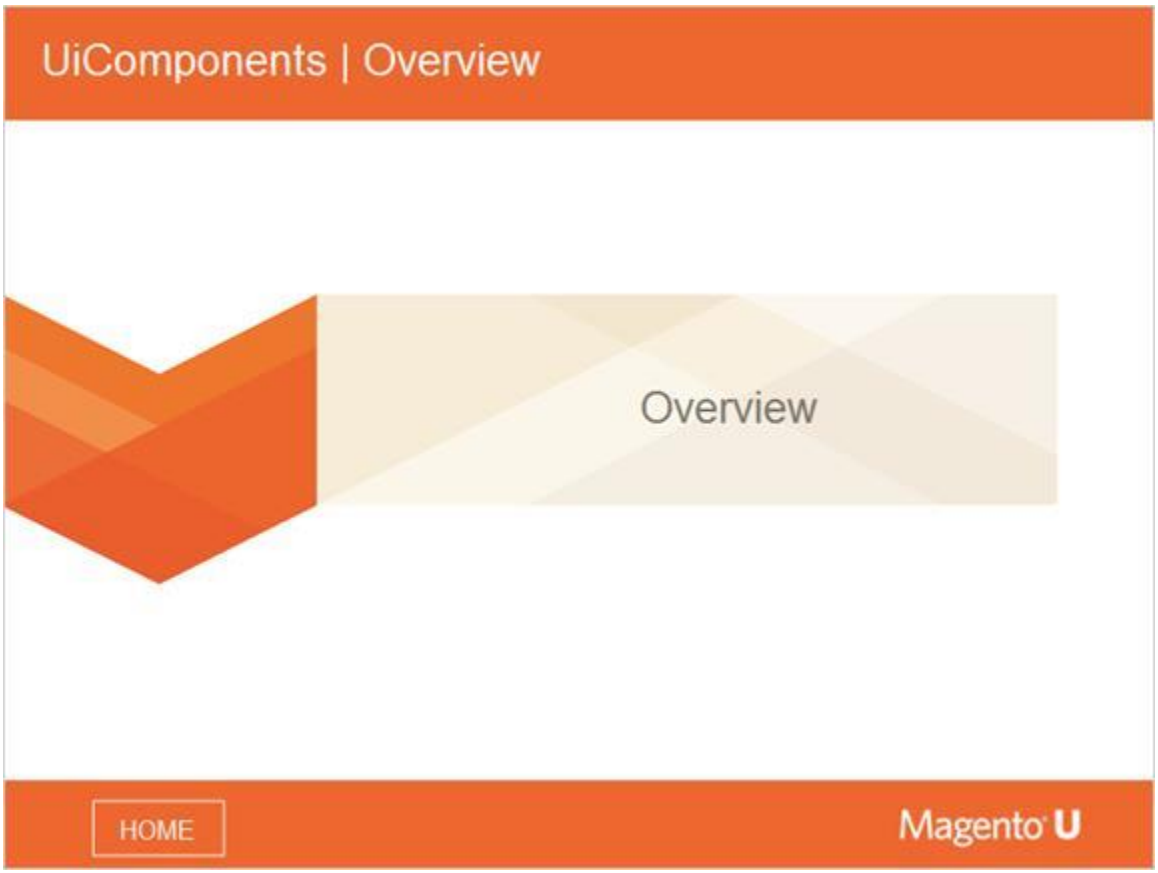
3.2 UiComponents | Module Topics



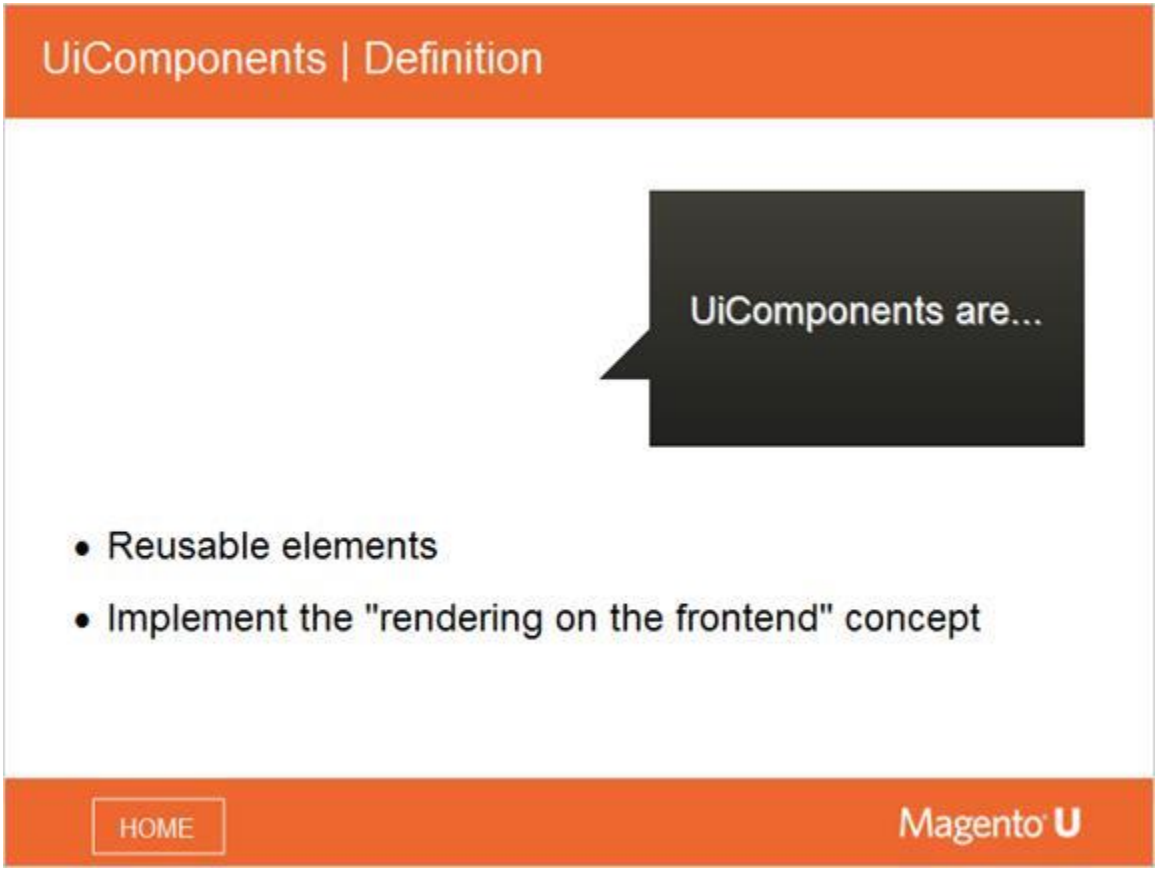
Notes:

In this module we will discuss UiComponents:
- Overview
- Architecture
- Configuration

3.3 UiComponents | Overview



3.4 UiComponents | Definition

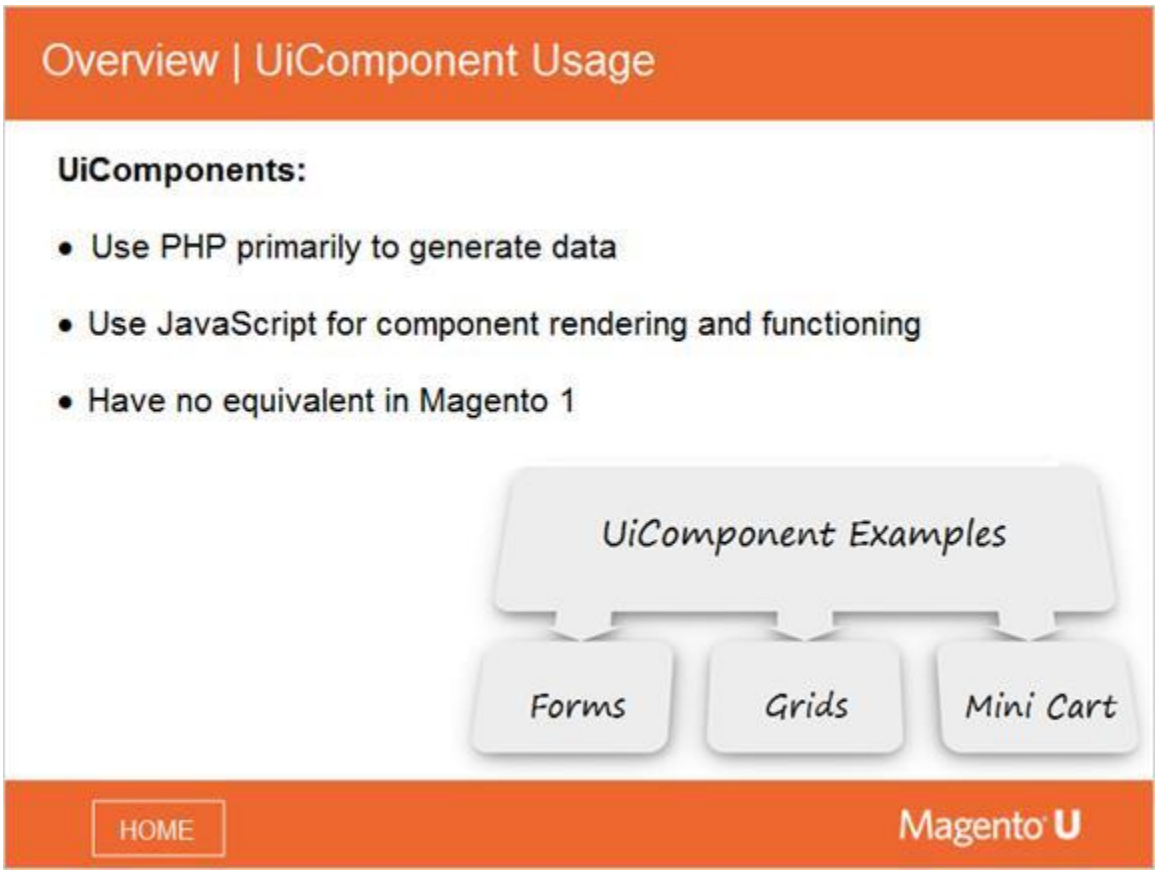


Notes:

UiComponents are reusable elements (pieces of an interface) generated and managed by JavaScript.

These components implement the "rendering on the frontend" concept.

3.5 Overview | UiComponent Usage



Notes:

Some examples of UiComponents are shown on the slide.

In Magento 1, there was nothing equivalent to UiComponents. The closest elements were form elements from the `lib/Varien/Data/Form` library, but they were all rendered in PHP.

In Magento 2, UiComponents also use PHP, but primarily for generating data for the component. The most important aspects of component rendering and functioning are done in JavaScript.

In this course, we will discuss JavaScript only as it relates to UiComponent architecture and the backend.

3.6 Overview | UiComponents & Blocks Comparison

Overview | UiComponents & Blocks Comparison

	UiComponent	Block
Root class	Magento\Ui\Component\AbstractComponent	Magento\Framework\View\Element\AbstractBlock
Interface	Magento\Framework\View\Element\UiComponentInterface *extends BlockInterface*	Magento\Framework\View\Element\BlockInterface
Templates	xhtml, html, knockout templates	phtml
Data	Uses own infrastructure to obtain dates; Allows data sharing between components.	Obtains data required for itself; No sharing between blocks.
Configuration	Configured by their own xml files	Configured by layout.
Rendered	On the frontend by JavaScript.	On the backend by PHP.

HOME

Magento U

Notes:

This table presents similarities and differences between UiComponents and Blocks.

As you can see, both blocks and UiComponents have one thing in common: they both implement `BlockInterface`, which means they use the `toHtml()` method.

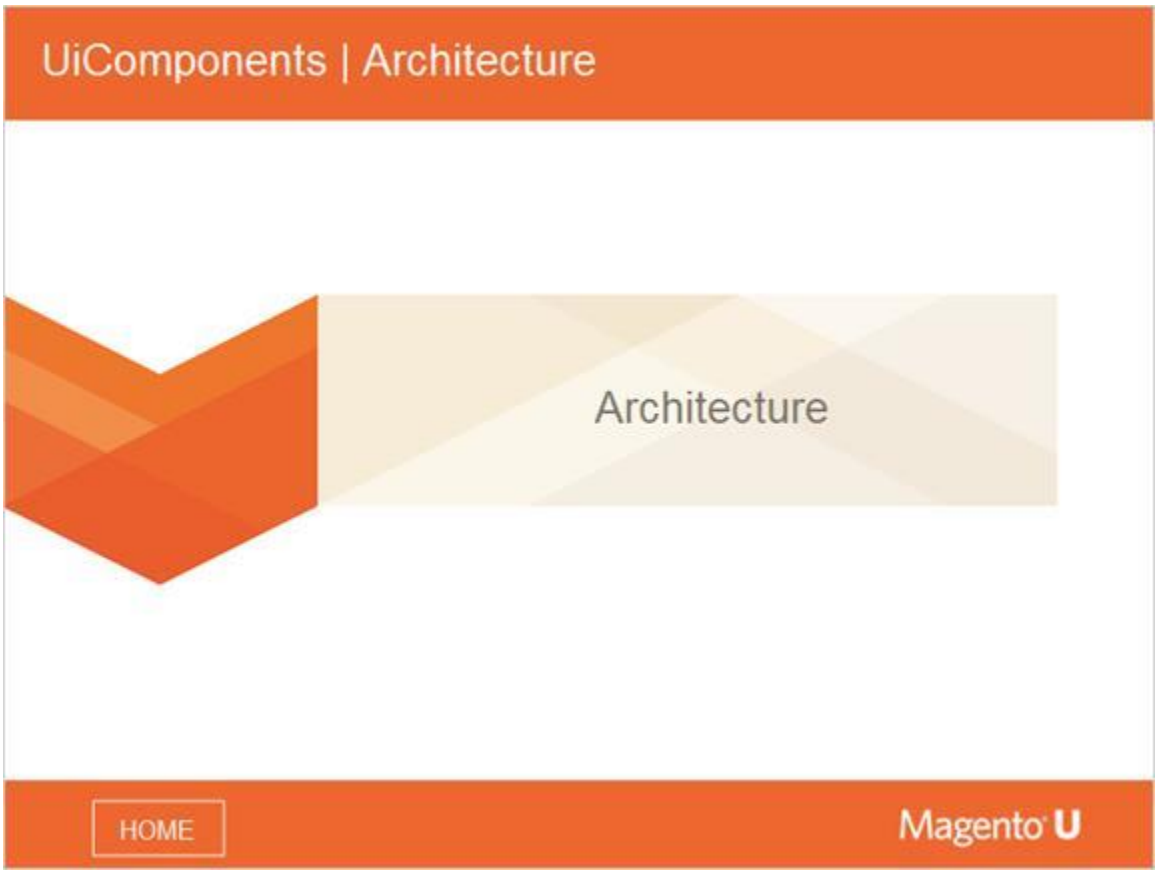
UiComponents have their own templates -- xhtml and html files -- while blocks use phtml templates.

Special classes generate data for UiComponents, while blocks serve as the data containers themselves.

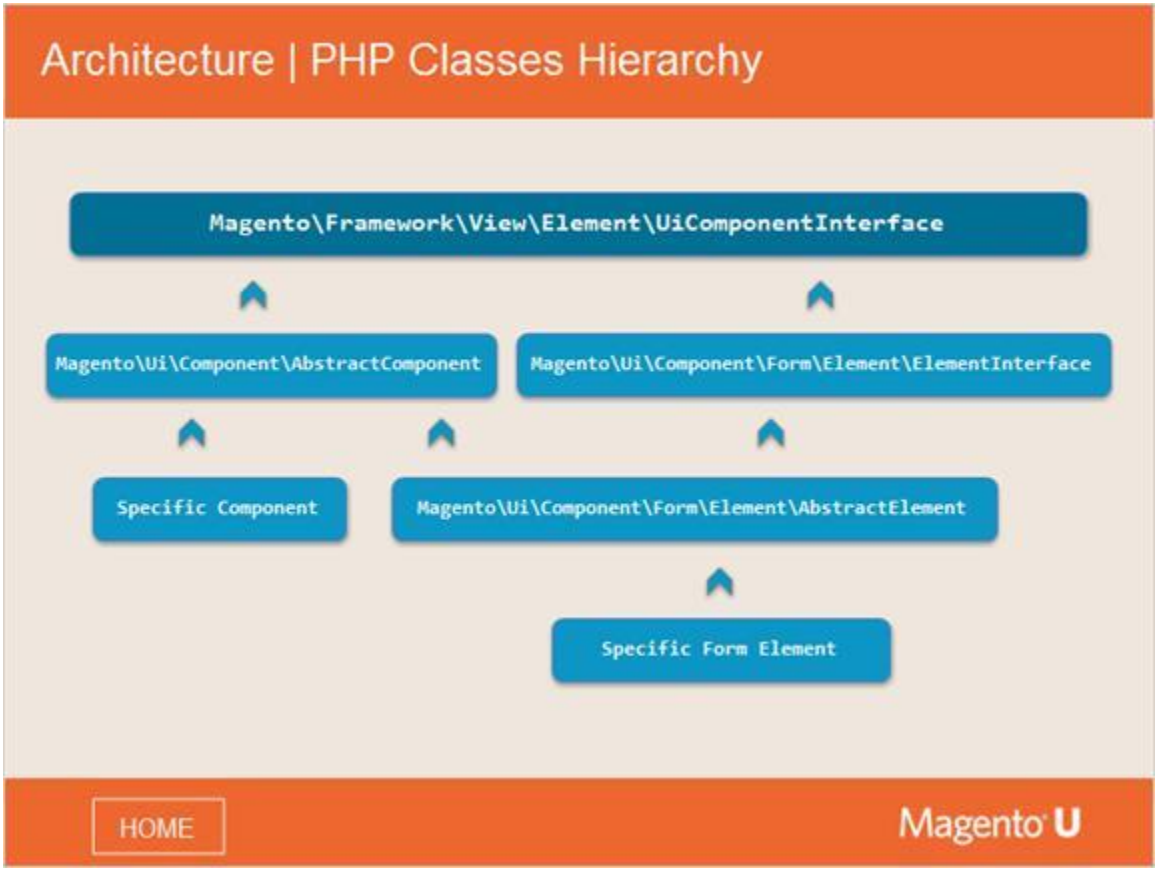
Also, UiComponents have their own configuration files, while there are no configuration files for blocks (other than layout).

And, as mentioned earlier, UiComponents are rendered by JavaScript while blocks are rendered by PHP.

3.7 UiComponents | Architecture



3.8 Architecture | PHP Classes Hierarchy



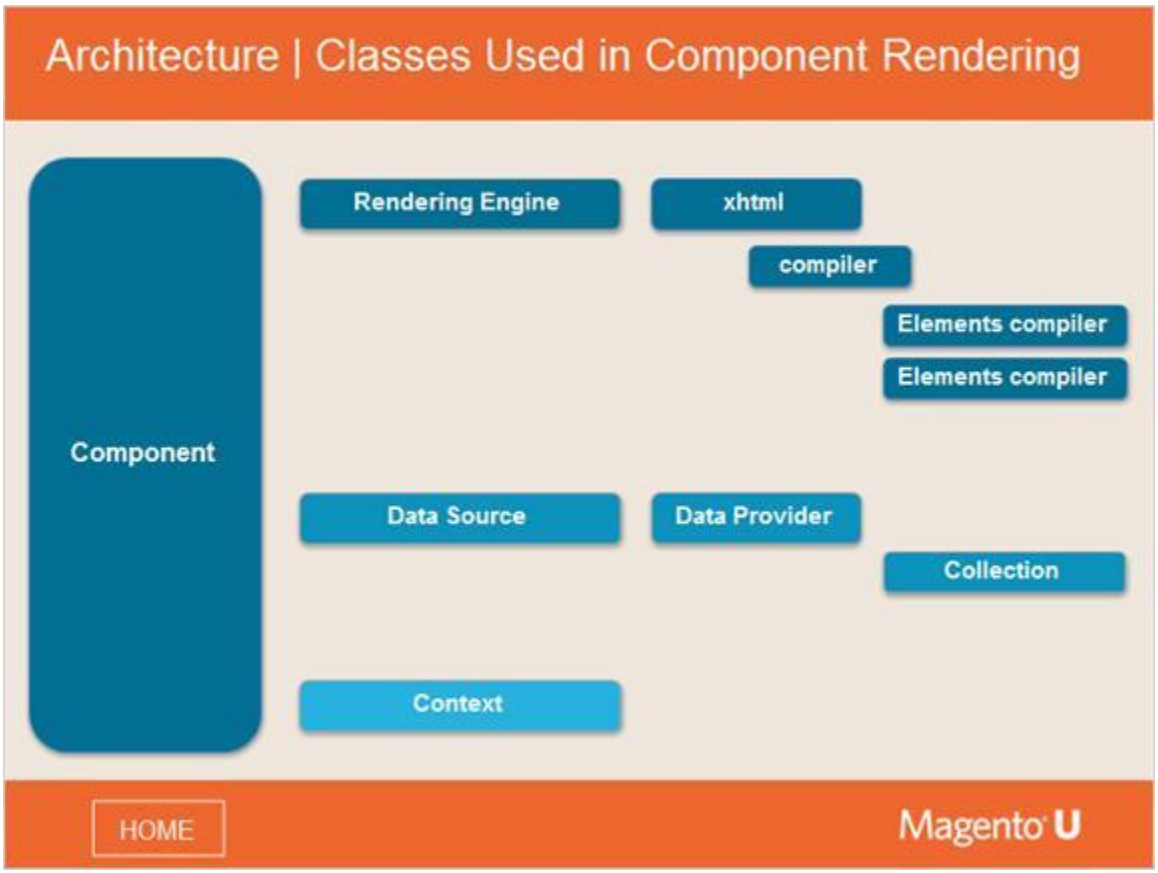
Notes:

As shown on this diagram, every UiComponent has to extend `UiComponentInterface` (which extends `BlockInterface`).

You can also see two types of UiComponents: form elements and "other" (extending `AbstractComponent`).

On the slides that follow, we will examine `UiComponentInterface` and `AbstractComponent` in more detail.

3.9 Architecture | Classes Used in Component Rendering



Notes:

This slide displays key classes involved in component rendering. They are presented here at a high level.

Each component usually contains a `Context` object. These objects encapsulate other objects and data that the component might need.

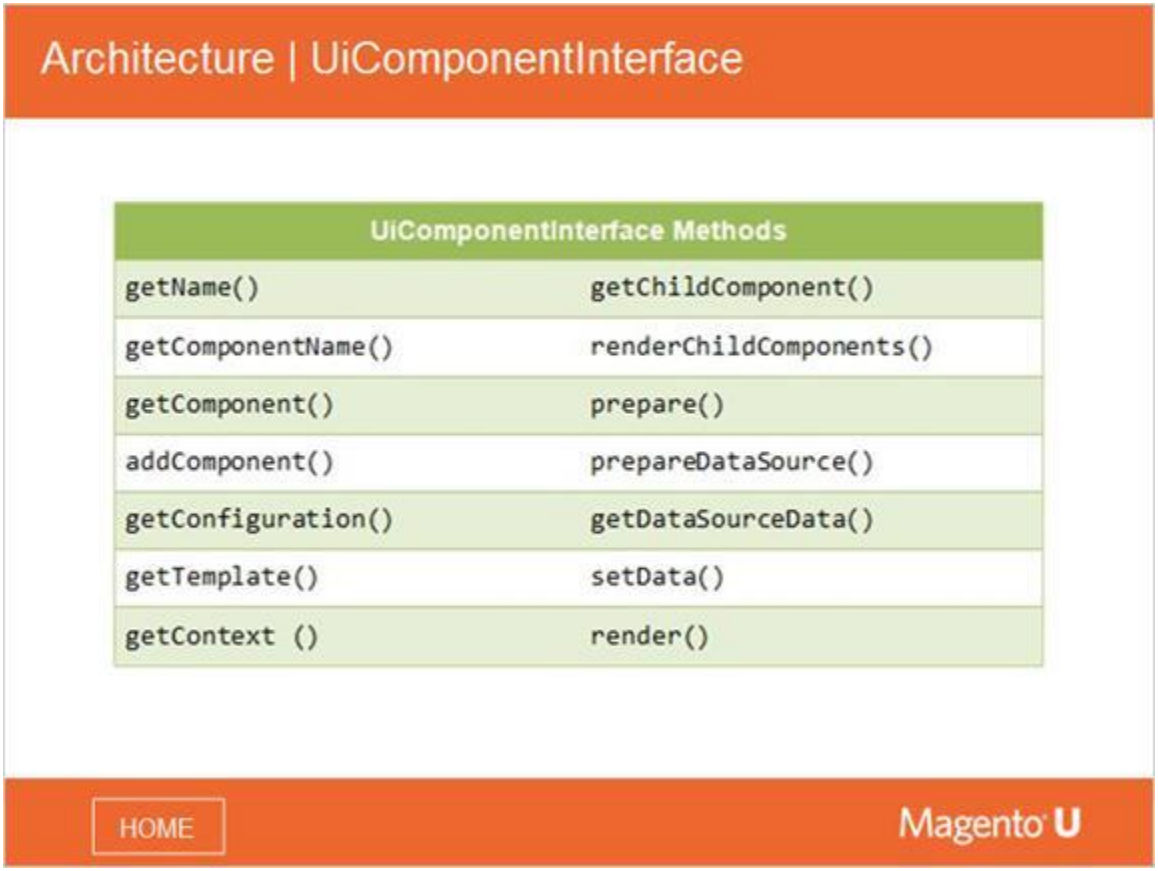
Usually components are hierarchical, so we have a root component and its children (for example, a grid and columns, filters, etc.; a form and elements).

The root component usually starts with an xhtml template which will be compiled in PHP. This process is initiated by the `toHtml()` method.

A further rendering process then continues on the JavaScript side.

The data for a component is provided by two special classes: `DataSource` and `DataProviders`.

3.10 Architecture | UiComponentInterface



UiComponentInterface Methods	
getName()	getChildComponent()
getComponentName()	renderChildComponents()
getComponent()	prepare()
addComponent()	prepareDataSource()
getConfiguration()	getDataSourceData()
getTemplate()	setData()
getContext ()	render()

Notes:

This table presents a listing of `UiComponentInterface` methods.

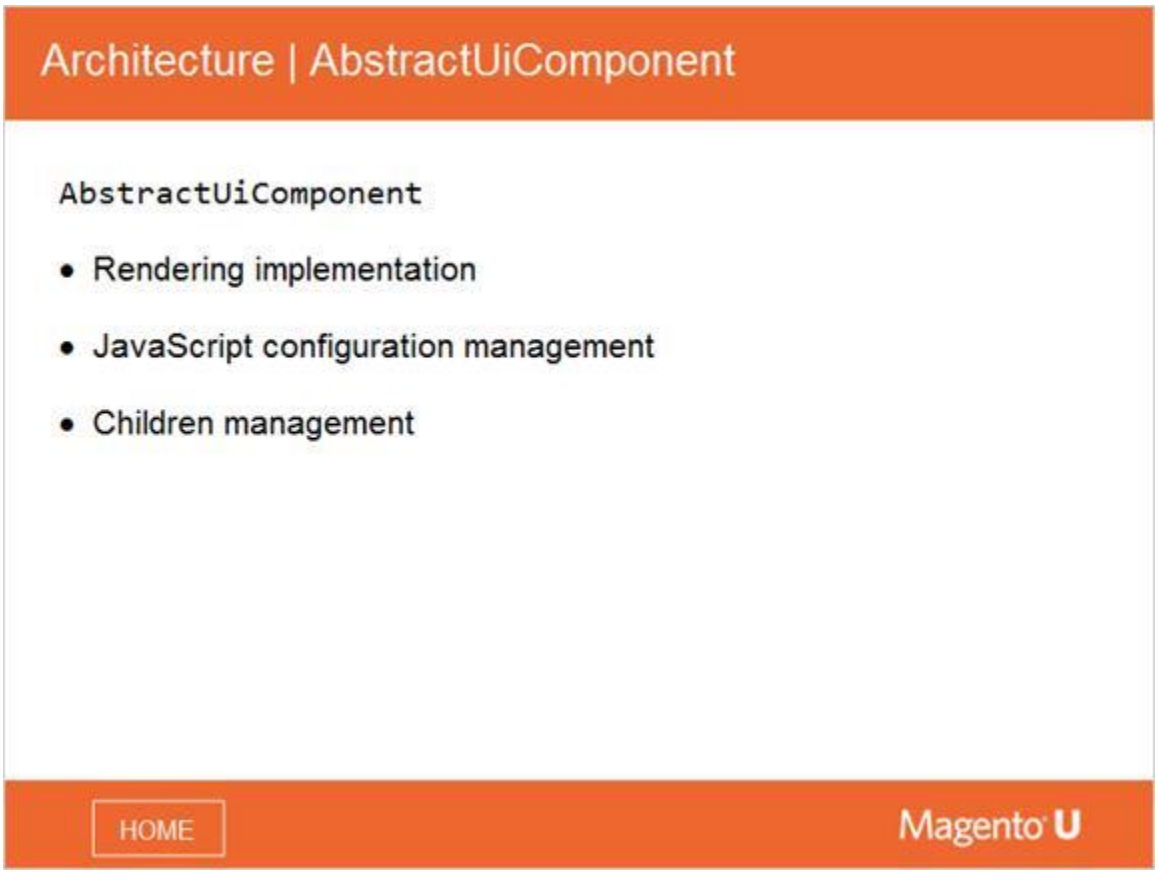
Most of them are self-explanatory, but some require clarification.

`render()`: The `render()` method is responsible for rendering a component on the PHP side. As discussed earlier, the process only includes compilation of an xhtml file, while the component itself (which usually consists of html templates) will be rendered by JavaScript.

`prepare()`: The `prepare()` method is where component-specific operations may be located.

`getDataSourceData()`: The `getDataSourceData()` method extracts data for the UiComponent that goes into the JavaScript config.

3.11 Architecture | AbstractUiComponent



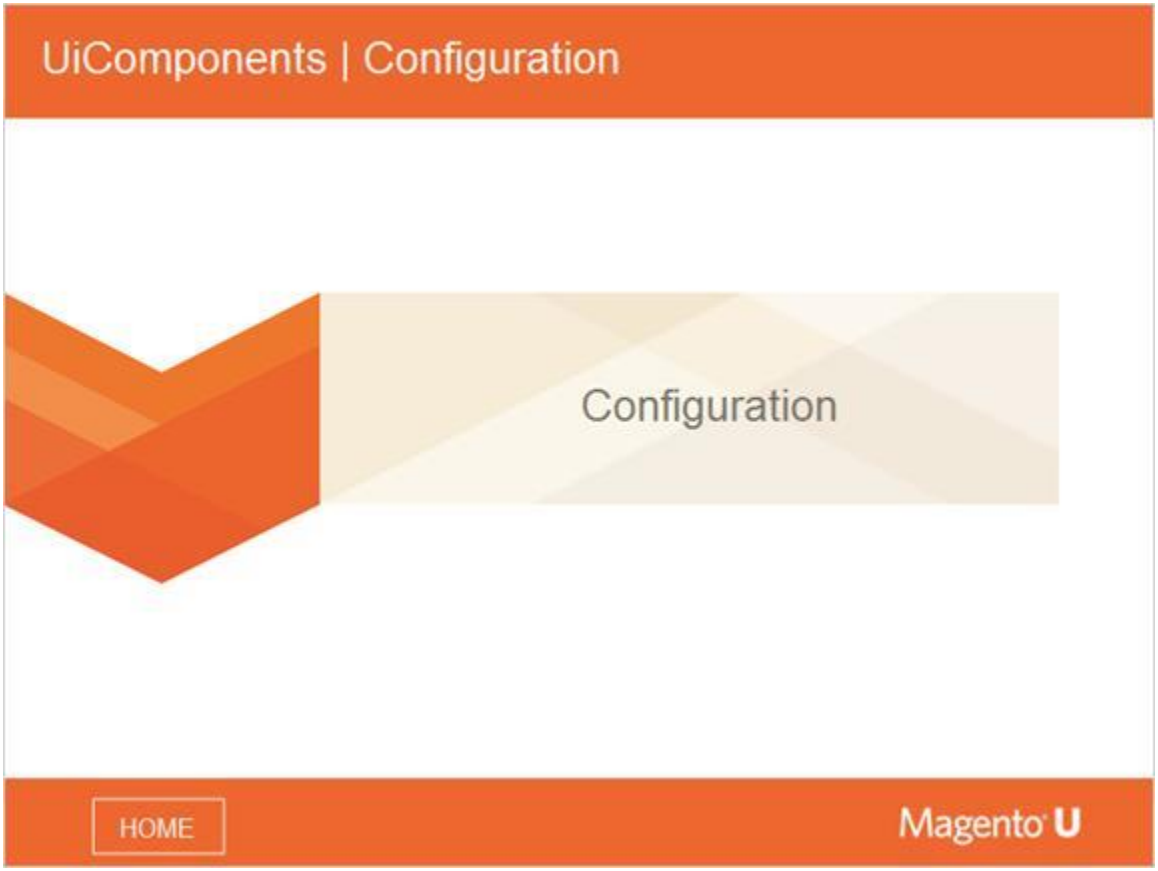
Notes:

Here are the key responsibilities of the `AbstractUiComponent` class (`Magento\Ui\Component\AbstractComponent`).

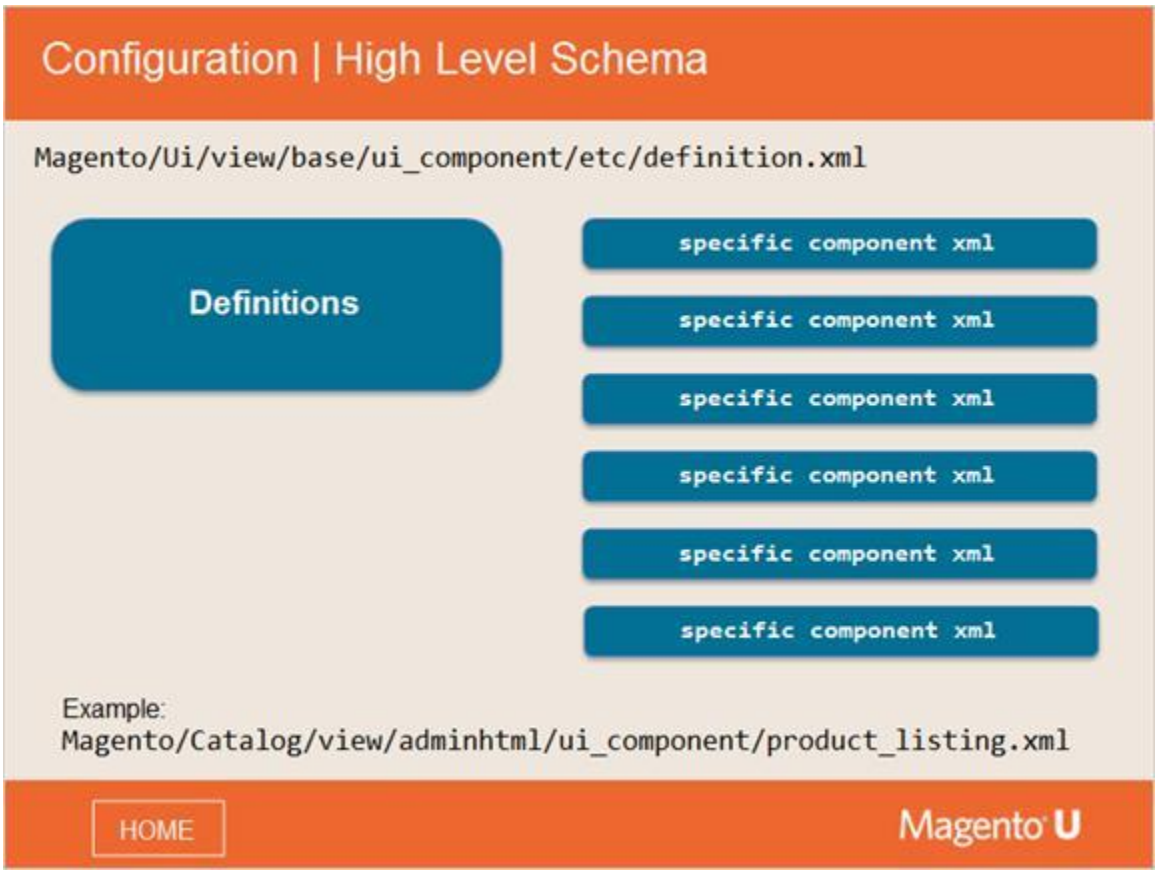
Its methods can be divided into three groups, as shown on the slide.

Their content is quite straightforward. Take a few minutes to explore this class in your installation.

3.12 UiComponents | Configuration



3.13 Configuration | High Level Schema



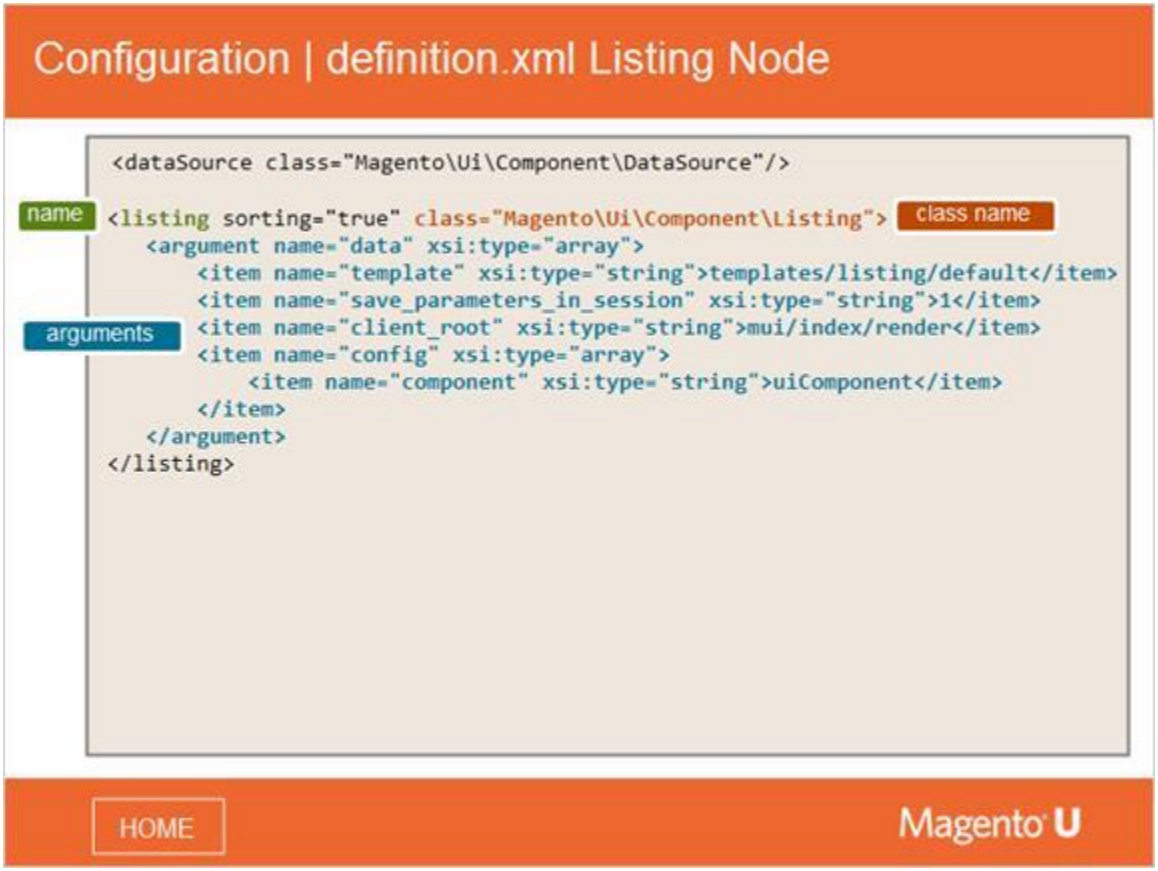
Notes:

At a high level, there is a definition file (`Magento/Ui/view/base/ui_component/etc/definition.xml`) and files that are specific to each UiComponent (for example: `Magento/Catalog/view/adminhtml/ui_component/product_listing.xml`).

Definition files contain basic information about all the components that exist in a system.

Specific component configuration files extend definitions by adding relevant information about a component -- its data source, children, and so on.

3.14 Configuration | definition.xml Listing Node



Notes:

Here is an overview of the `definition.xml` file.

The file consists of a list of component nodes (such as listing, pages, and so on). This slide displays the listing node; the next slide displays the paging node.

Each node has a certain structure, in which is defined:

- **Class name**: PHP class that corresponds to the UiComponent.
- **List of arguments**: Some of the arguments are generic, but very important, like `js_config`, `template`, and so on.

3.15 Configuration | definition.xml Paging Node

Configuration | definition.xml Paging Node

name

class name

arguments

```
<paging class="Magento\Ui\Component\Paging">
   <argument name="data" xsi:type="array">
      <item name="config" xsi:type="array">
         <item name="component" xsi:type="string">
                                 Magento_Ui/js/grid/paging/paging</item>
            <item name="displayArea" xsi:type="string">bottom</item>
            <item name="storageConfig" xsi:type="array">
               <item name="provider" xsi:type="string">
                                 ns = ${ $.ns }, index = bookmarks</item>
               <item name="namespace" xsi:type="string">current.paging</item>
         </item>
      </item>
   </argument>
</paging>
```

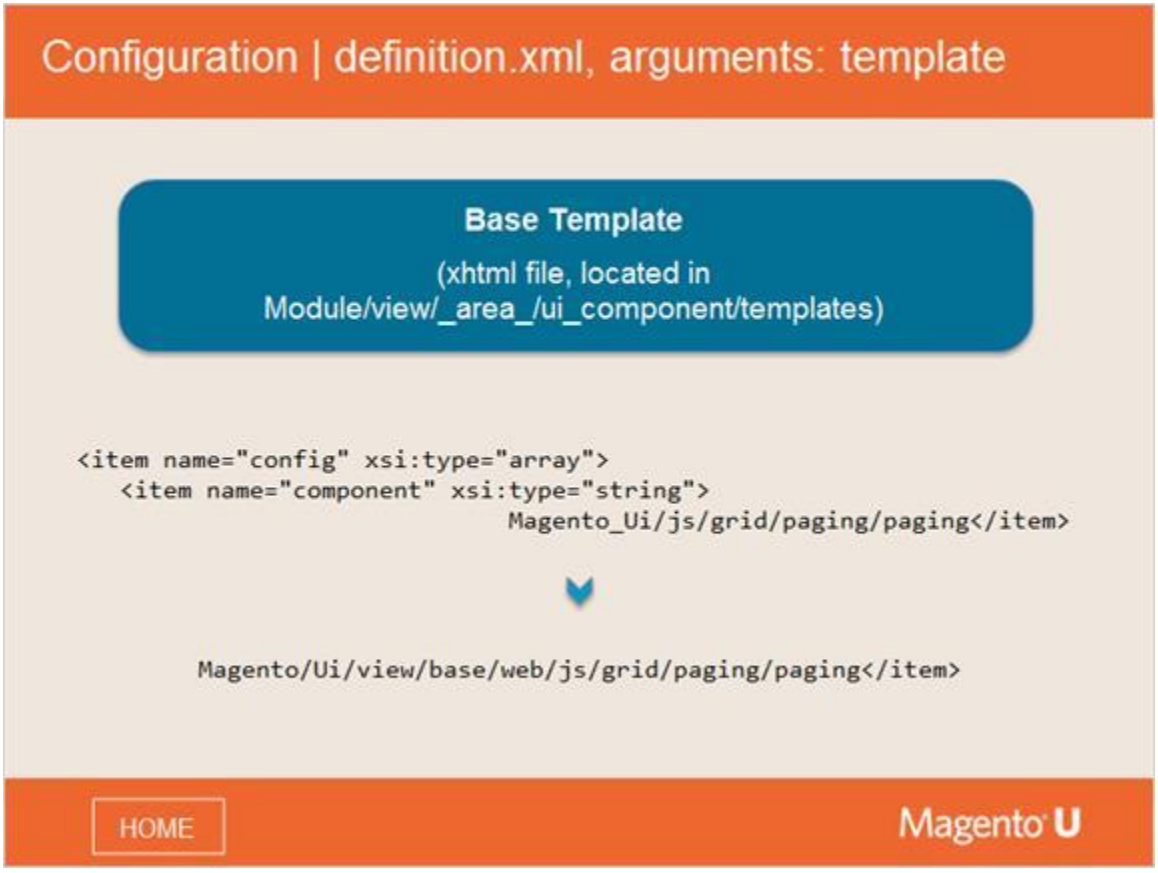
HOME

Magento U

Notes:

Here is the paging node of the `definition.xml` file.

3.16 Configuration | definition.xml, arguments: template



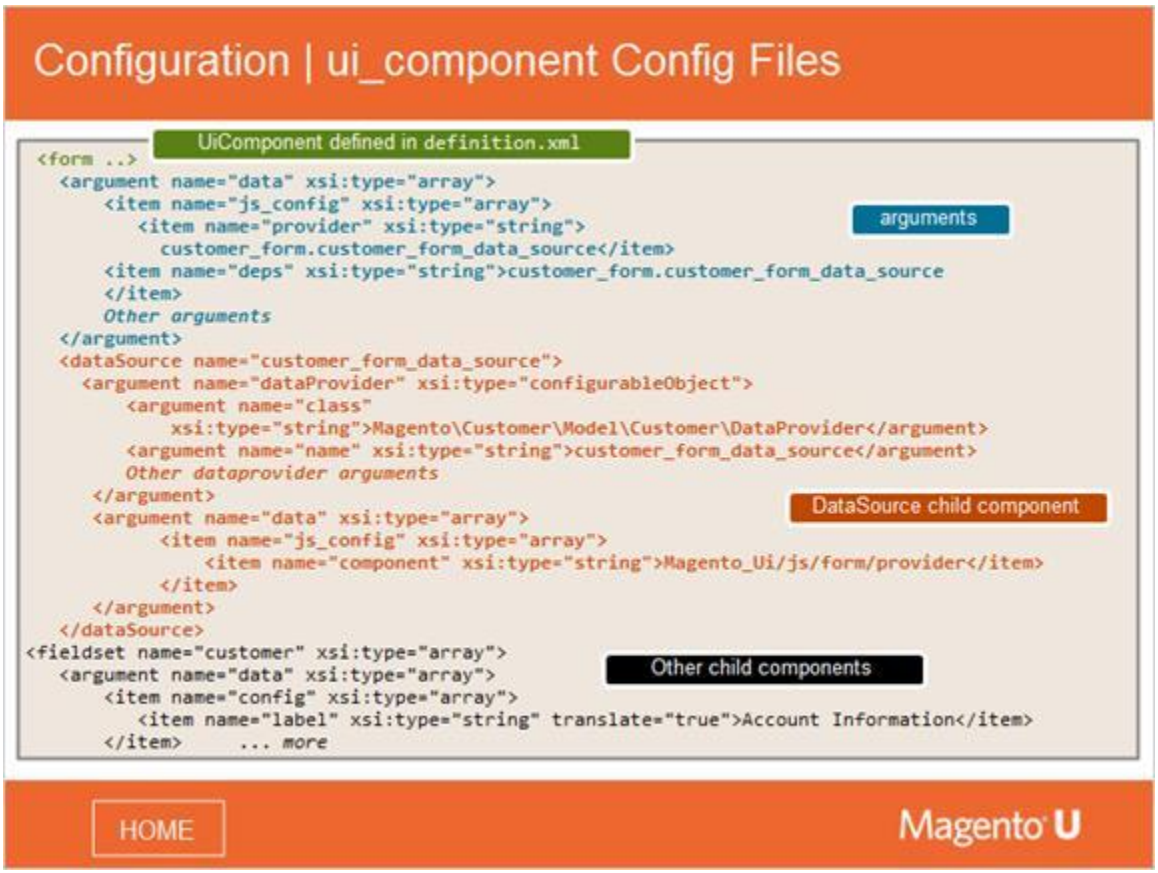
Notes:

The `template` item in an `arguments` array defines an xhtml file. This file serves as a container, so it doesn't contain "real" code for rendering a component. The "real" code is taken from `*.html` files.

It is important to realize that the xhtml template will be compiled (into `html` code) by the component's class. Magento 2 provides its own compilers for xhtml.

We will examine their function in the "Templates" section.

3.17 Configuration | ui_component Config Files




```
<form ..>
  <argument name="data" xsi:type="array">
      <item name="js_config" xsi:type="array">
         <item name="provider" xsi:type="string">
           customer_form.customer_form_data_source</item>
      <item name="deps" xsi:type="string">customer_form.customer_form_data_source
      </item>
      Other arguments
  </argument>
  <dataSource name="customer_form_data_source">
    <argument name="dataProvider" xsi:type="configurableObject">
        <argument name="class"
            xsi:type="string">Magento\Customer\Model\Customer\DataProvider</argument>
        <argument name="name" xsi:type="string">customer_form_data_source</argument>
        Other dataprovider arguments
    </argument>
    <argument name="data" xsi:type="array">
          <item name="js_config" xsi:type="array">
              <item name="component" xsi:type="string">Magento_Ui/js/form/provider</item>
          </item>
    </argument>
  </dataSource>
<fieldset name="customer" xsi:type="array">
  <argument name="data" xsi:type="array">
      <item name="config" xsi:type="array">
         <item name="label" xsi:type="string" translate="true">Account Information</item>
      </item>     ... more
```

Notes:

Let's now move to the component's specific configuration file.

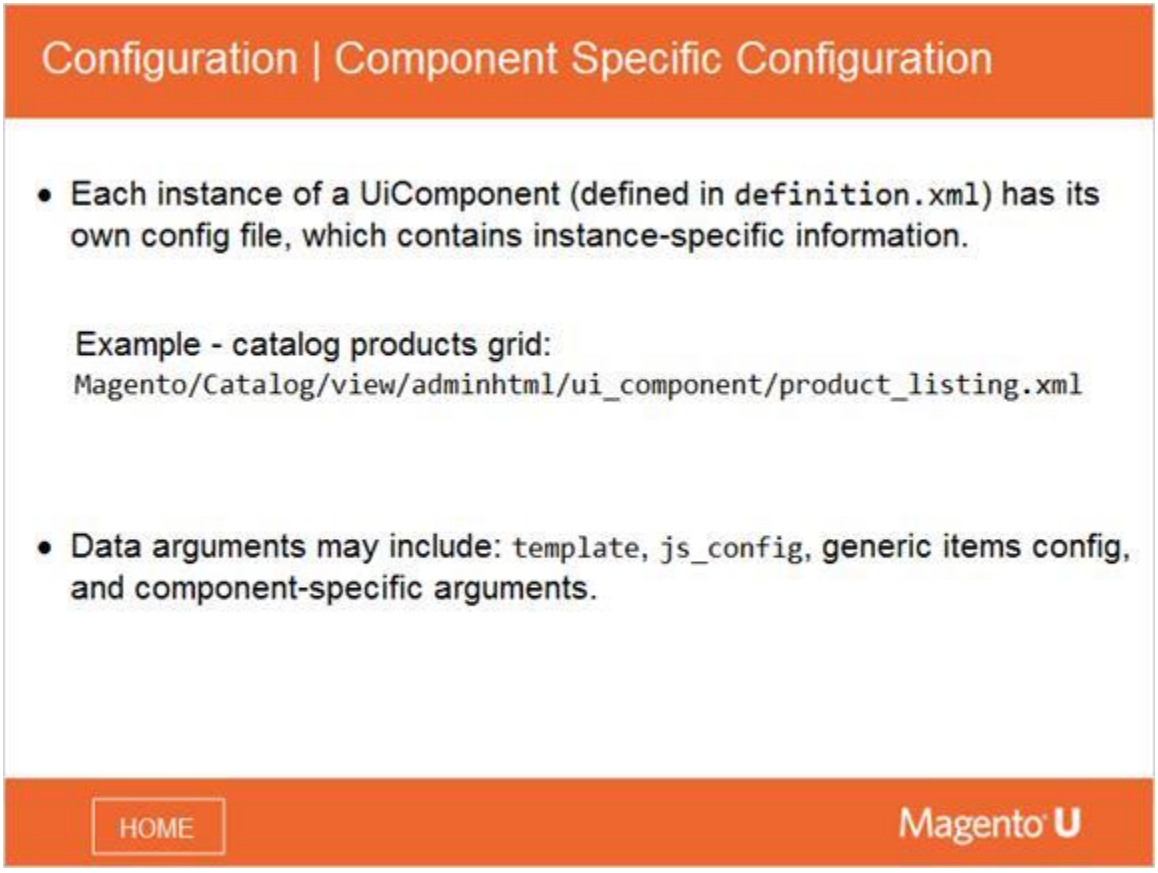
It is usually located in the `view/(base|frontend|adminhtml)/ui_component` folder of a Magento module.

Typical configuration files consist of the following elements:

- Component's name (*green*). The first node of a file defines the component; it should match one of the components declared in `definition.xml`.
- Arguments (*blue*) specific to this component.
- DataSource subcomponent (*orange*). This is another component which provides data.
- Child components (*black*).

We'll discuss each element in more detail.

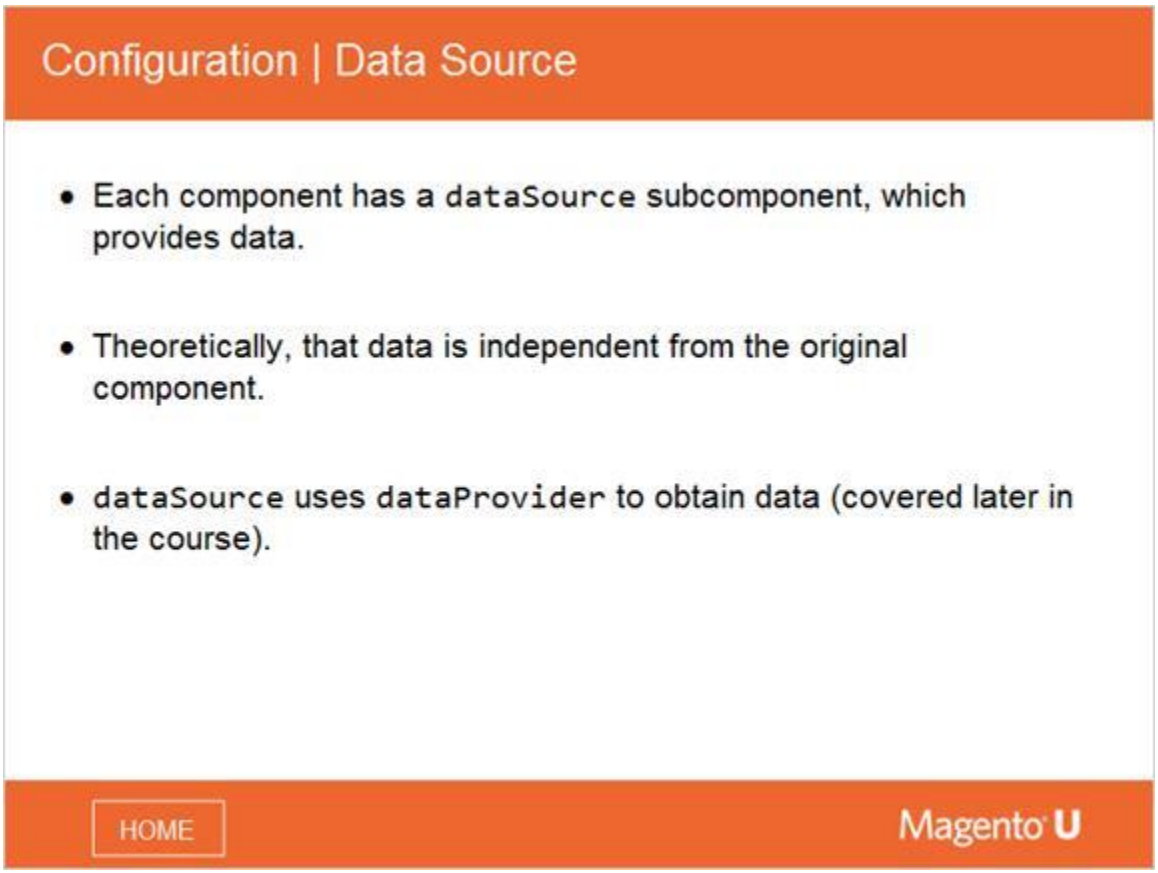
3.18 Configuration | Component Specific Configuration



Notes:

An instance specific configuration file extends the original component's definition with instance-specific data (for example, how to generate data for a component). Take a few moments to look at the products listing configuration in the example file shown on the slide.

3.19 Configuration | Data Source



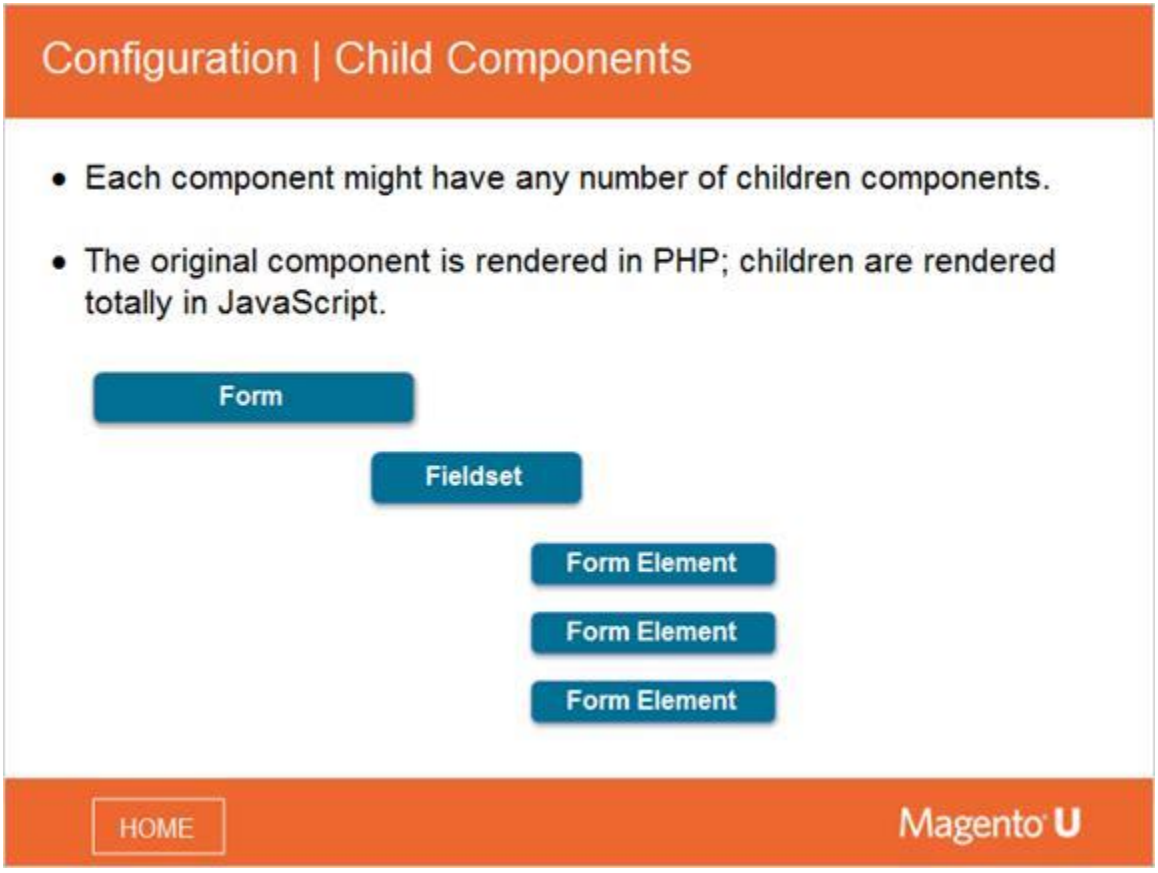
Notes:

The UiComponent framework uses the following approach in dealing with data…

Every component has a `dataSource` subcomponent, which delivers data.

While it may seem like the data subcomponent should be independent of the original component -- so other components on the page could also use the data -- a component almost always has its own `dataSource`, which provides data exclusively to it.

3.20 Configuration | Child Components



Notes:

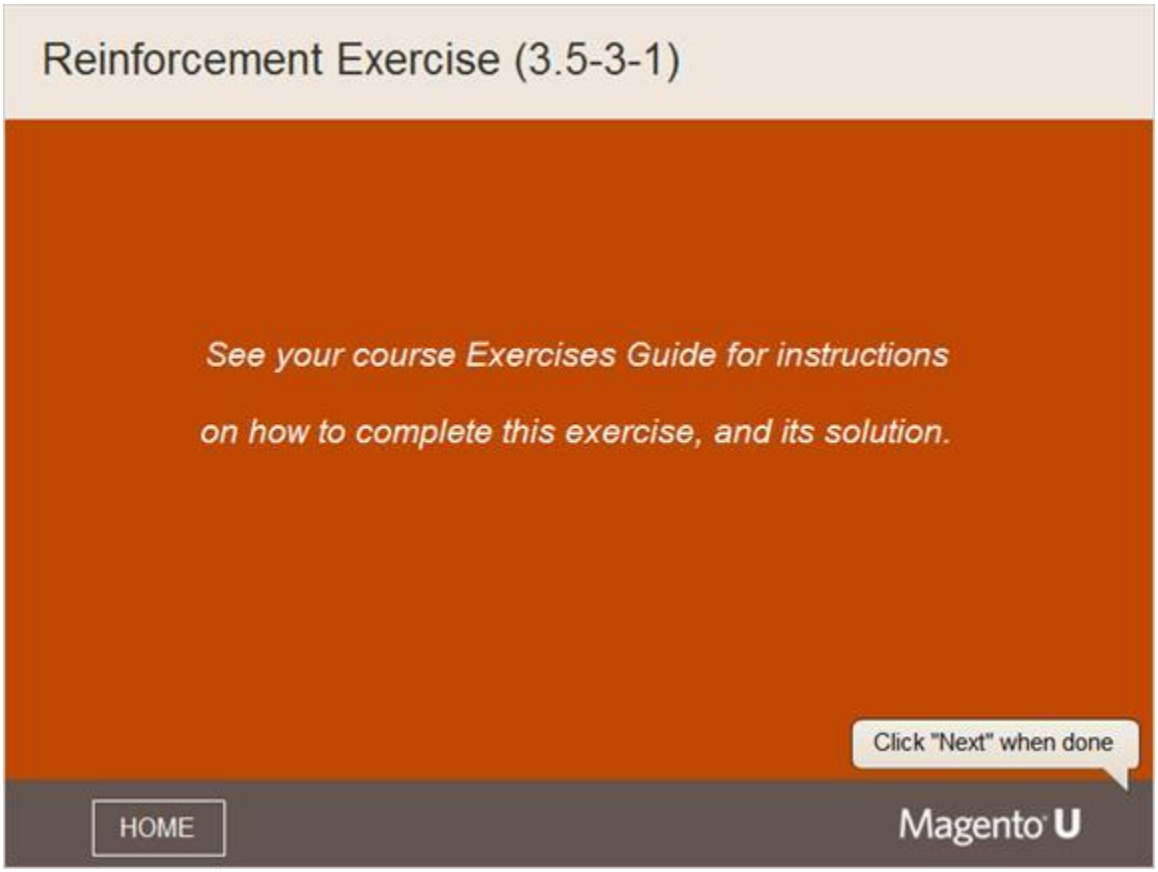
Usually components are composite -- for example, Form has elements, Grid has filters, and so on.

A component's JavaScript is usually aware of its children and knows how to render them.

It is important that the `toHtml()`, and therefore `render()`, PHP methods are only executed for the original (root) component and compile the xhtml template.

Everything else, including sub-component rendering, is in JavaScript (using the knockout templates engine and html templates).

3.21 Reinforcement Exercise (3.5-3-1)

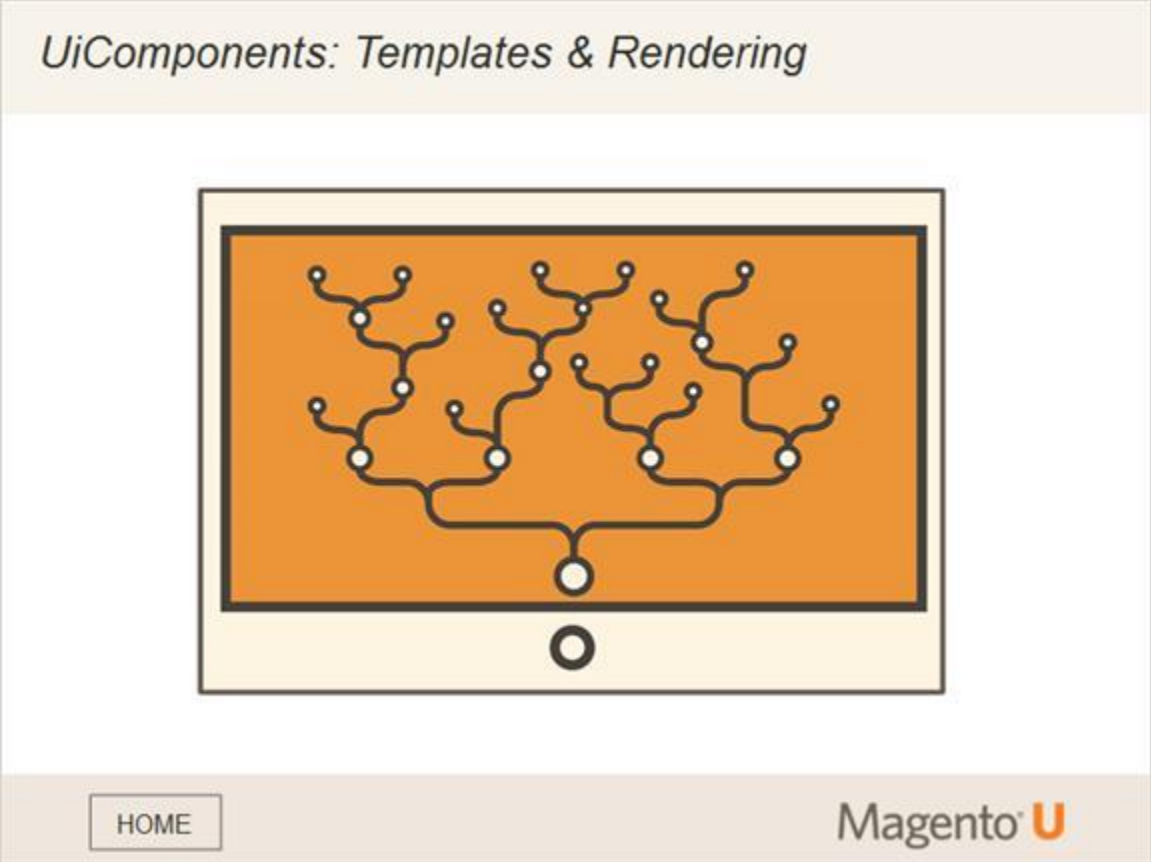


Notes:

See your course Exercises Guide for instructions on how to complete this exercise, and the solution.

4. UI Components

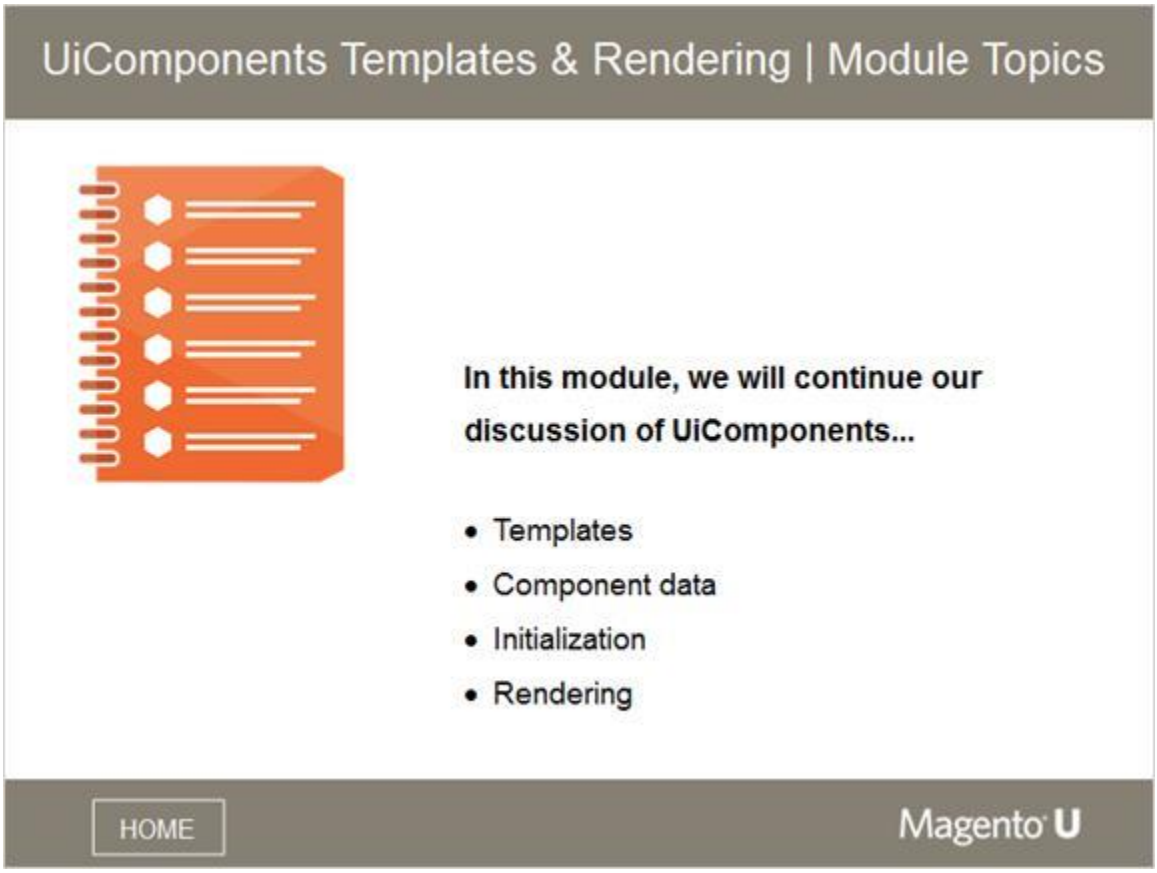
4.1 UIComponents: Templates & Rendering



Notes:

This module discusses UI components in more depth.

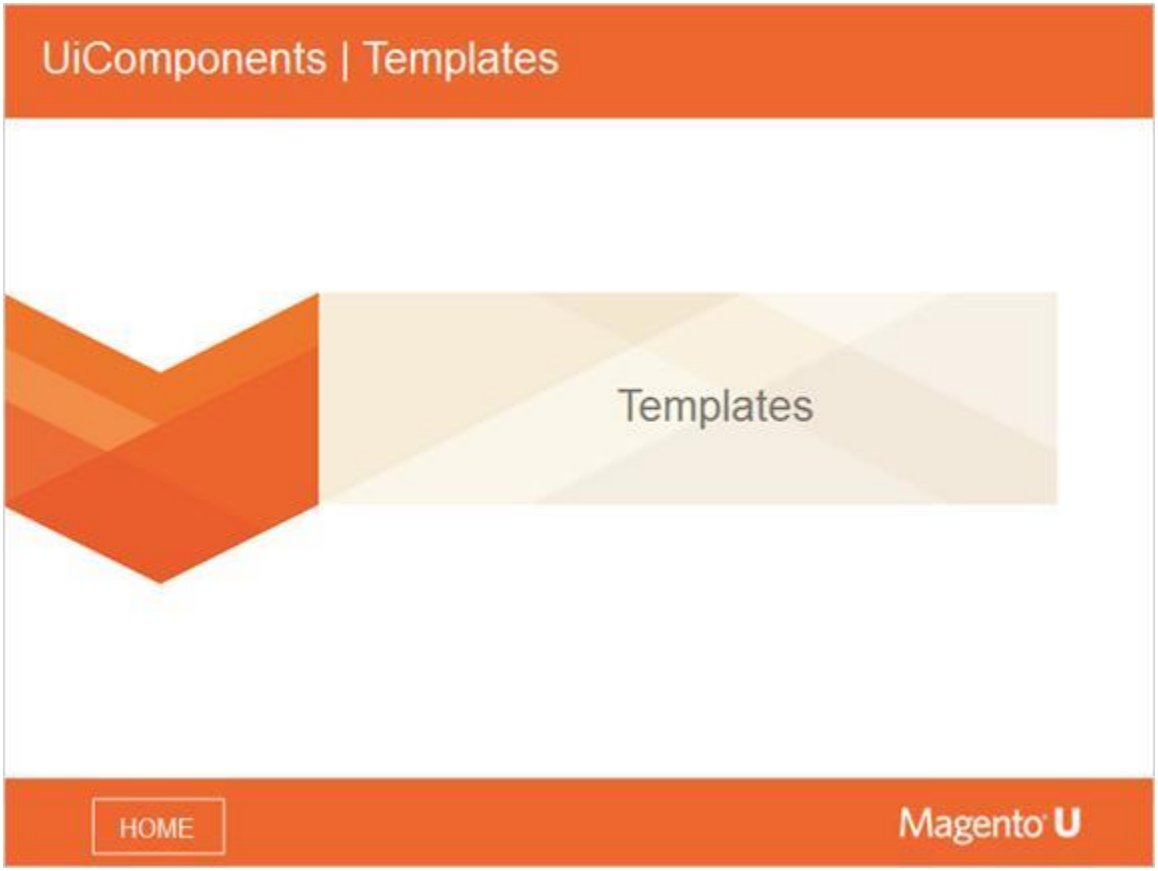
4.2 UiComponents Templates & Rendering | Module Topics



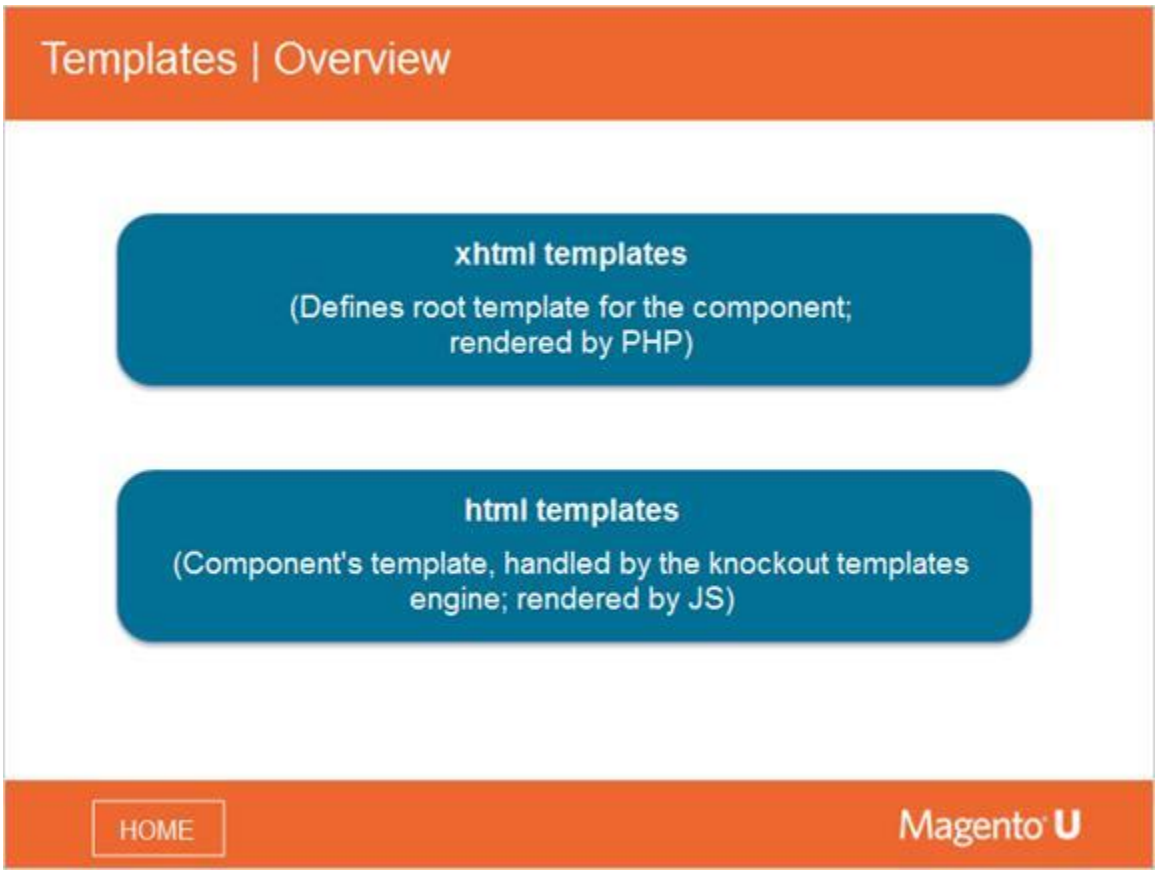
Notes:

In this module we will continue our discussion of UiComponents:
- Templates
- Component data
- Initialization
- Rendering

4.3 UiComponents | Templates



4.4 Templates | Overview



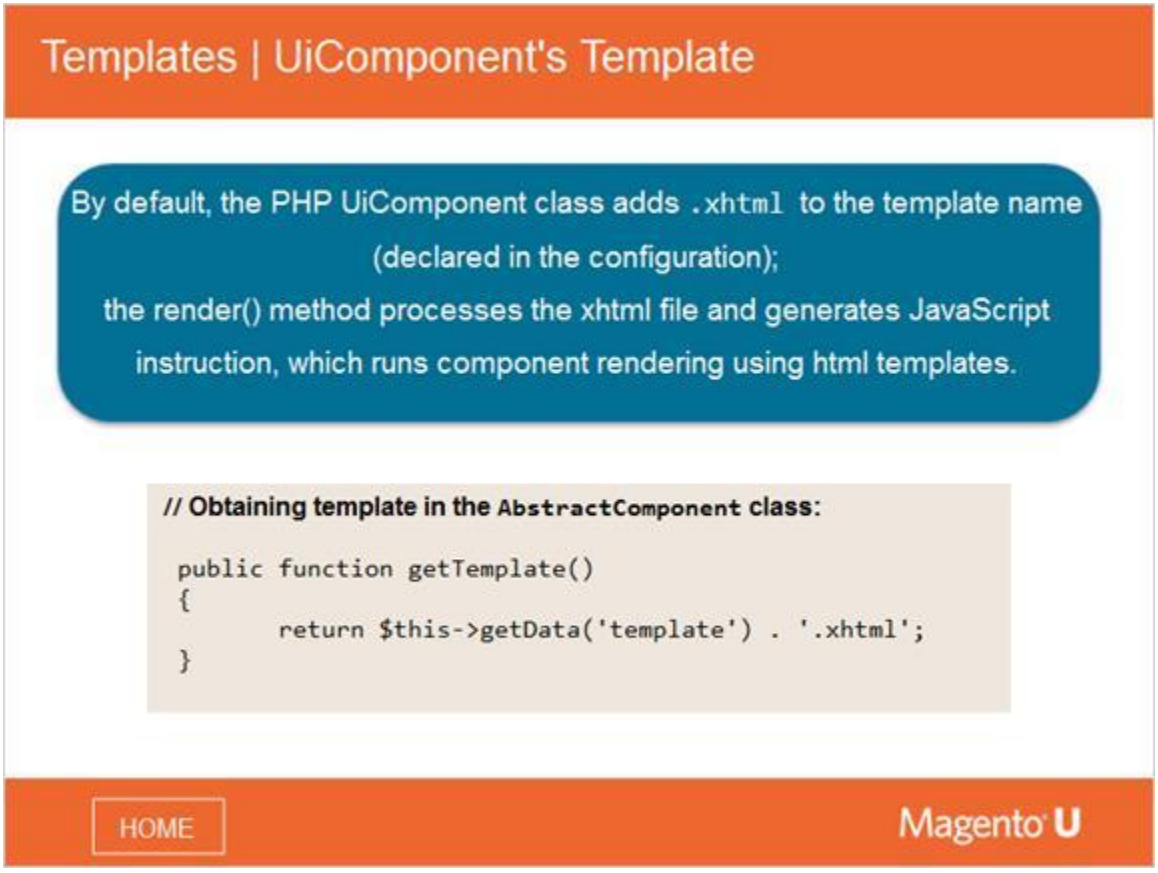
Notes:

As displayed on the slide, there are 2 types of templates used in UiComponents: xhtml and html templates.

XHTML is a "template" for the root component and is rendered in PHP.

HTML is an actual template rendered in JavaScript (using the knockout templates engine).

4.5 Templates | UiComponent's Template



Notes:

The code on this slide displays the `getTemplate()` method implementation, which is defined in the `AbstractComponent`.

As you can see, it always adds an `.xhtml` extension to the file. The xhtml template is used as a container, where JavaScript will be inserted to render the component and its children.

4.6 Templates | xhtml Template Declaration

Templates | xhtml Template Declaration

```
// Declaration in definition.xml (orange text)

<listing sorting="true" class="Magento\Ui\Component\Listing">
    <argument name="data" xsi:type="array">
        <item name="template" xsi:type="string">
          templates/listing/default</item>
        <item name="save_parameters_in_session" xsi:type="string">1</item>
        <item name="client_root" xsi:type="string">mui/index/render</item>
```

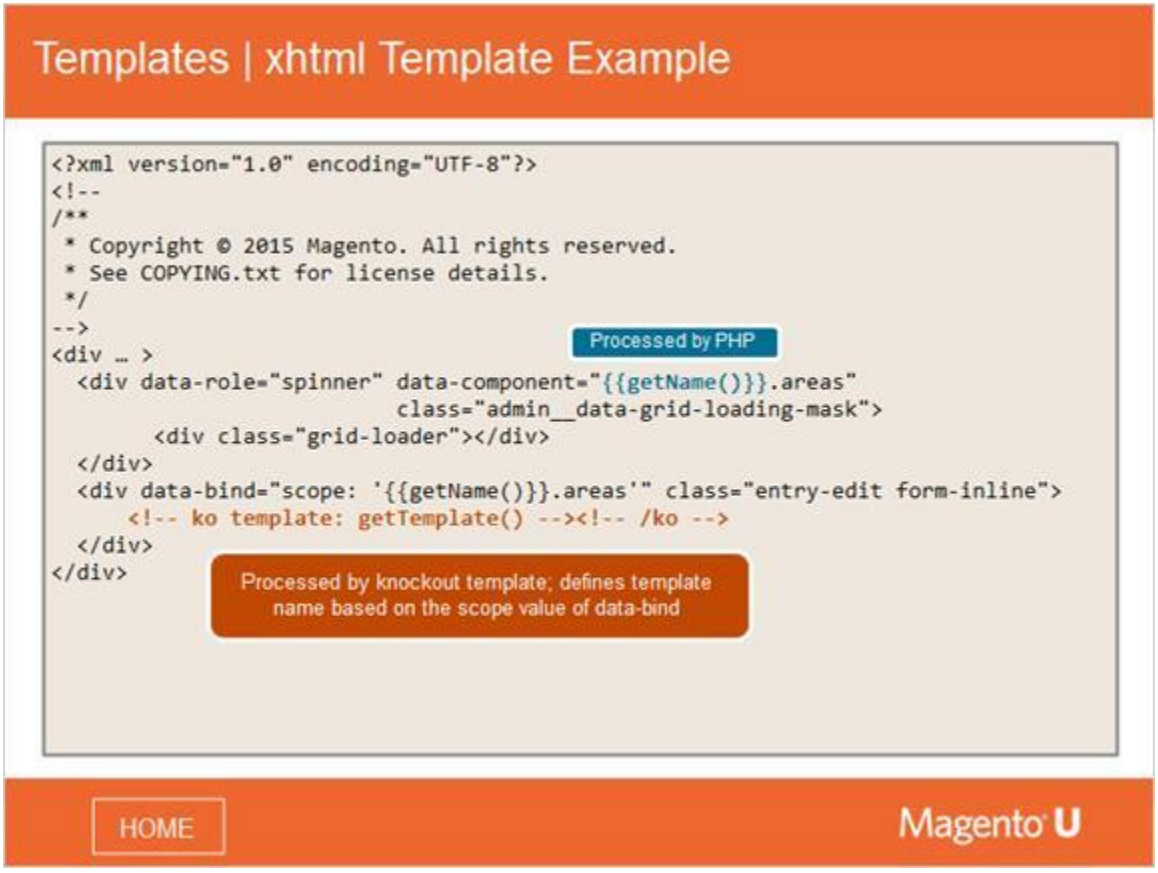
HOME

Magento U

Notes:

A code example of the xhtml template declaration in `definition.xml`.

4.7 Templates | xhtml Template Example



Notes:

And finally, the .xhtml file itself.

This particular example is: `Magento/Ui/view/base/ui_component/templates/form/default.xhtml`.

You can see it contains `div` containers.

Elements contained within the curly braces `{{..}}` are what the Magento compiler will process. There are two types of elements that could be included in the braces.

One is a component's method (the method of a corresponding component's PHP class), and the other is a variable.

In addition to processing the double brace contents `{{..}}`, a JavaScript element with component configuration, data, and modules will be generated and inserted at the bottom of the file.

Note the comment in brown `<!-- ko template: getTemplate() --><!-- /ko -->`. This will be processed by the knockout template, and will use an html file as a template.

4.8 Templates | html Template Declaration

Templates | html Template Declaration

```
// html template declaration in config

<filterSelect class="Magento\Ui\Component\Filters\Type\Select">
        <argument name="data" xsi:type="array">
            <item name="config" xsi:type="array">
                <item name="template" xsi:type="string">
                    ui/grid/filters/field</item>
            </item>
      </argument>
</filterSelect>
```

HOME

Magento U

Notes:

This slide shows how an html template could be declared in a component's xml. The template file declared in the configuration can be found at: `Magento/Ui/view/base/web/templates/grid/filters/filter.html`

Note that the "template" item should be inside a config element.

4.9 Templates | html Template Declaration

Templates | html Template Declaration

```
// Magento/Ui/view/base/web/templates/form/element/input.html

<!--
/**
 * Copyright © 2015 Magento. All rights reserved.
 * See COPYING.txt for license details.
 */
-->
<input class="admin__control-text" type="text"
    data-bind="
        event: {change: userChanges},
        value: value,
        hasFocus: focused,
        valueUpdate: valueUpdate,
        attr: {
            name: inputName,
            placeholder: placeholder,
            'aria-describedby': noticeId,
            id: uid,
            disabled: disabled
    }" />
```

HOME

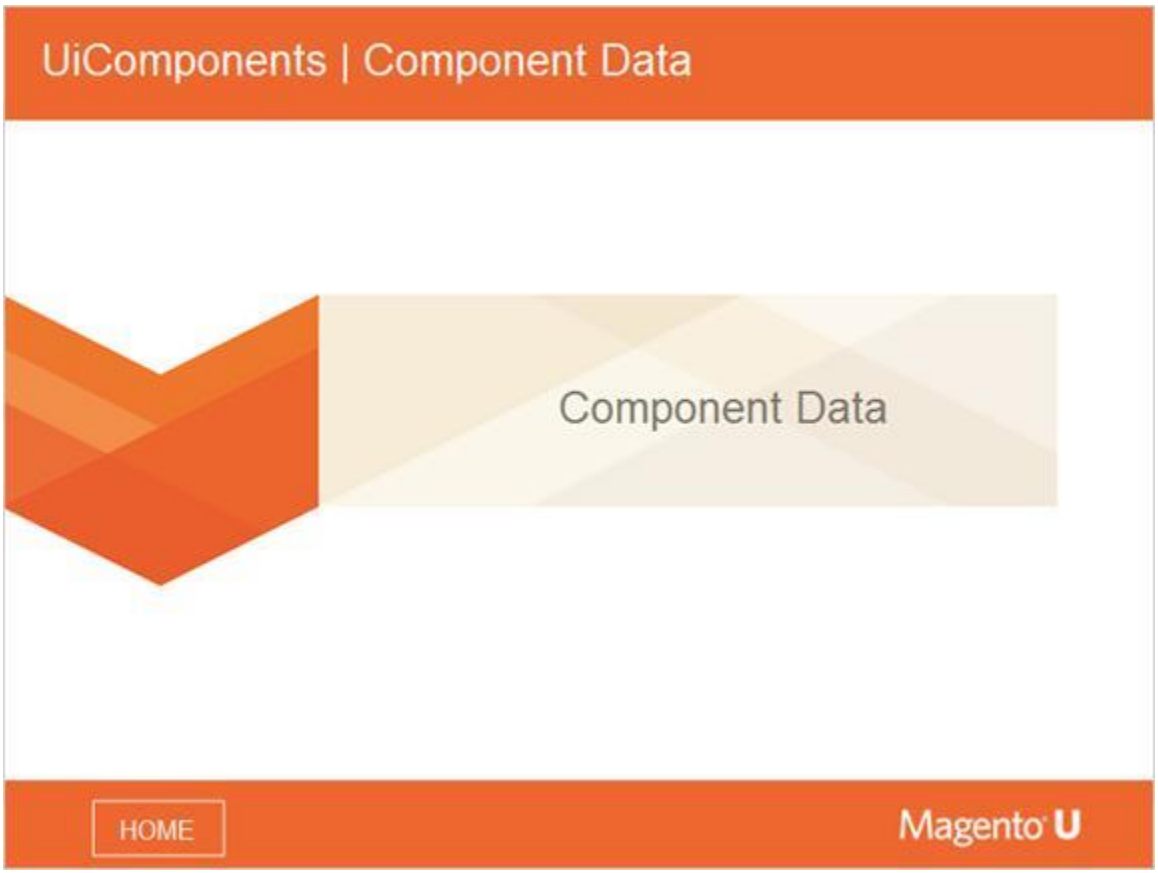
Magento U

Notes:

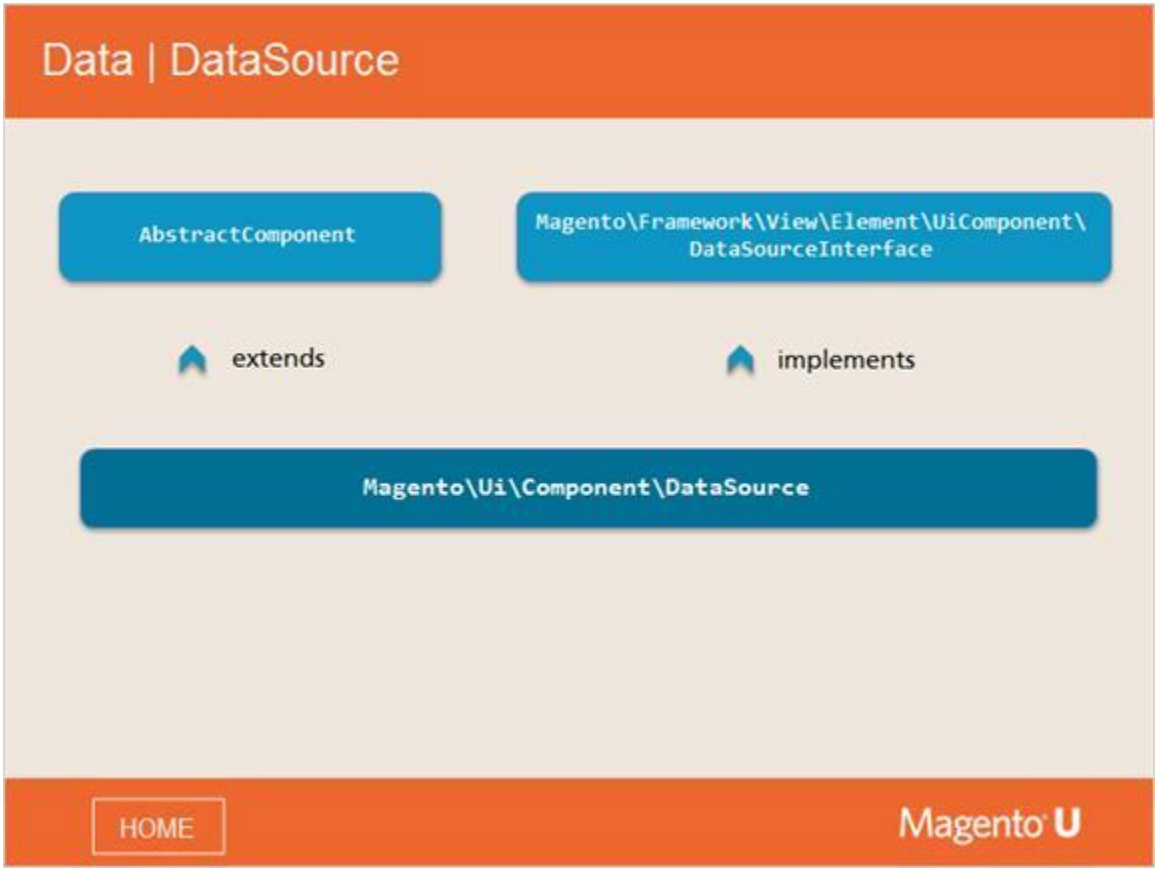
And finally, here is an example of an html template for the input component.

This template will be processed by the knockout template engine.

4.10 UiComponents | Component Data



4.11 Data | DataSource



Notes:

The hierarchy for the `DataSource` component is depicted above.

As you know, each component should have its own PHP class. For the `DataSource`, there is the interface `DataSourceInterface`, which every `DataSource` implementation has to implement.

The UiComponent `DataSource` commonly extends `AbstractComponent`.

4.12 Data | DataProvider

Data | DataProvider

- Implements:

  `Magento\Framework\View\Element\UiComponent\DataProvider\DataProviderInterface`

- Wraps data
- Provides methods to access, filter, and sort data
- Is used by `DataSource` component
- Examples:

  `Magento\Framework\View\Element\UiComponent\DataProvider\DataProvider`
  `Magento\Catalog\Ui\DataProvider\Product\ProductDataProvider`

HOME

Magento U

Notes:

The `DataSource` does not actually contain data. Instead, it obtains data from the `DataProvider`, the class that is responsible for data. Example: For the grid UiComponent, its `DataProvider` will wrap the collection with data.

4.13 Data | Accessing Data in JavaScript

Data | Accessing Data in JavaScript

```
<listing xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:module:Magento_Ui:etc/ui_configuration.xsd">
    <argument name="data" xsi:type="array">
        <item name="js_config" xsi:type="array">
            <item name="provider" xsi:type="string">
                                  product_listing.product_listing_data_source</item>
            <item name="deps" xsi:type="string">
                                  product_listing.product_listing_data_source</item>
        </item>
        <item name="spinner" xsi:type="string">product_columns</item>
    </argument>
    <dataSource name="product_listing_data_source">
        <argument name="dataProvider" xsi:type="configurableObject">
            <argument name="class" xsi:type="string">
                              Magento\Catalog\Ui\DataProvider\Product\ProductDataProvider</argument>
            <argument name="name" xsi:type="string">product_listing_data_source</argument>
            <argument name="primaryFieldName" xsi:type="string">entity_id</argument>
            <argument name="requestFieldName" xsi:type="string">id</argument>
            <argument name="data" xsi:type="array">
                <item name="config" xsi:type="array">
                    <item name="component" xsi:type="string">Magento_Ui/js/grid/provider</item>
                    <item name="update_url" xsi:type="url" path="mui/index/render"/>
                </item>
            </argument>
        </argument>
    </dataSource>
```

HOME

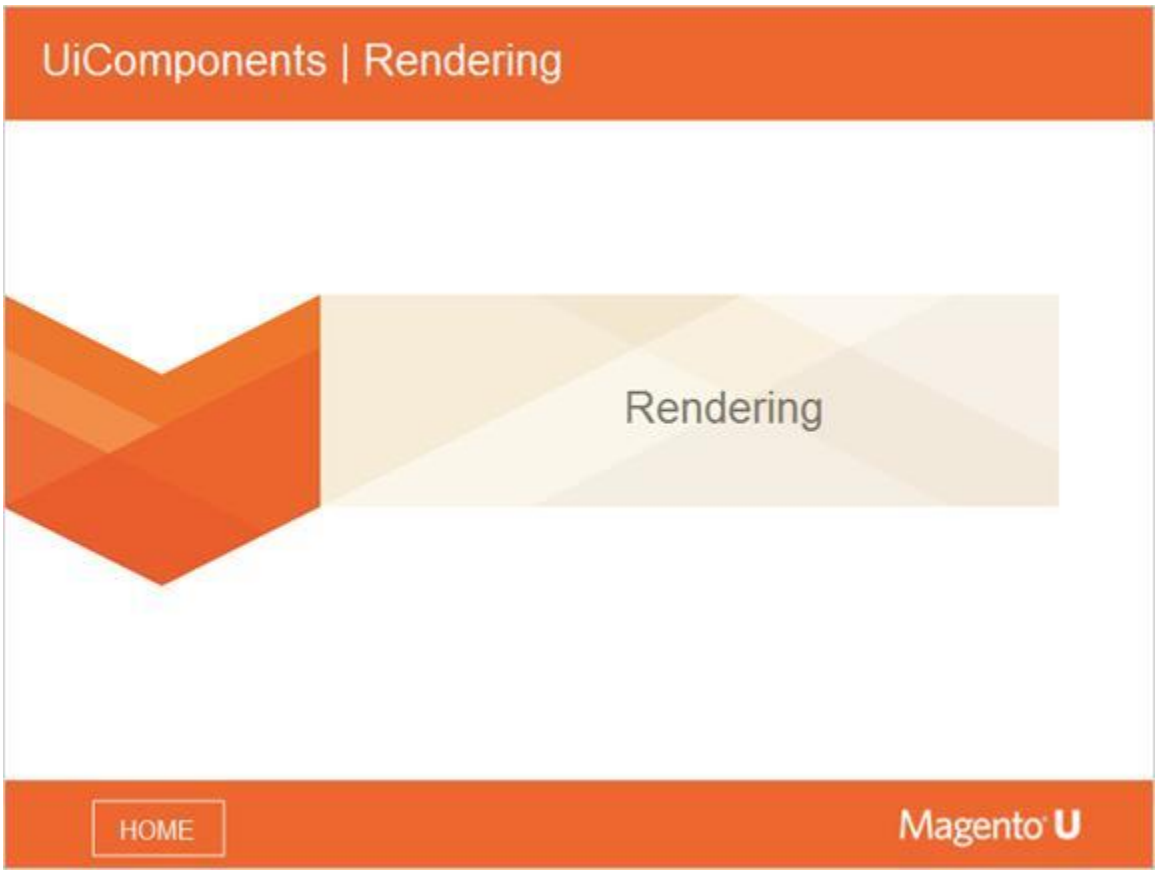
Magento U

Notes:

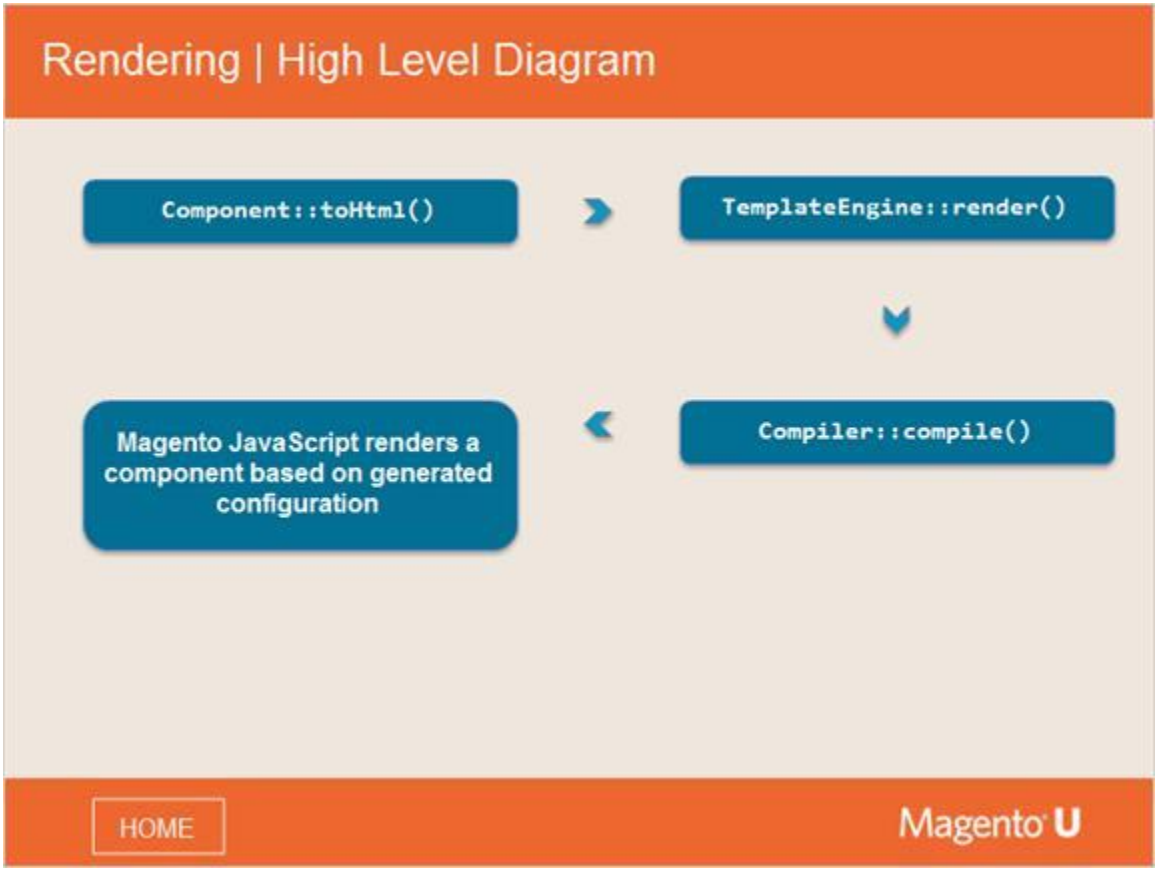
This slide displays how a `DataProvider` for a particular UiComponent can be specified.

In your installation of Magento 2, take the time to open the `ProductDataProvider` and explore it. You will see that most of its operations are involved in wrapping around similar collection operations.

4.14 UiComponents | Rendering



4.15 Rendering: High Level Diagram



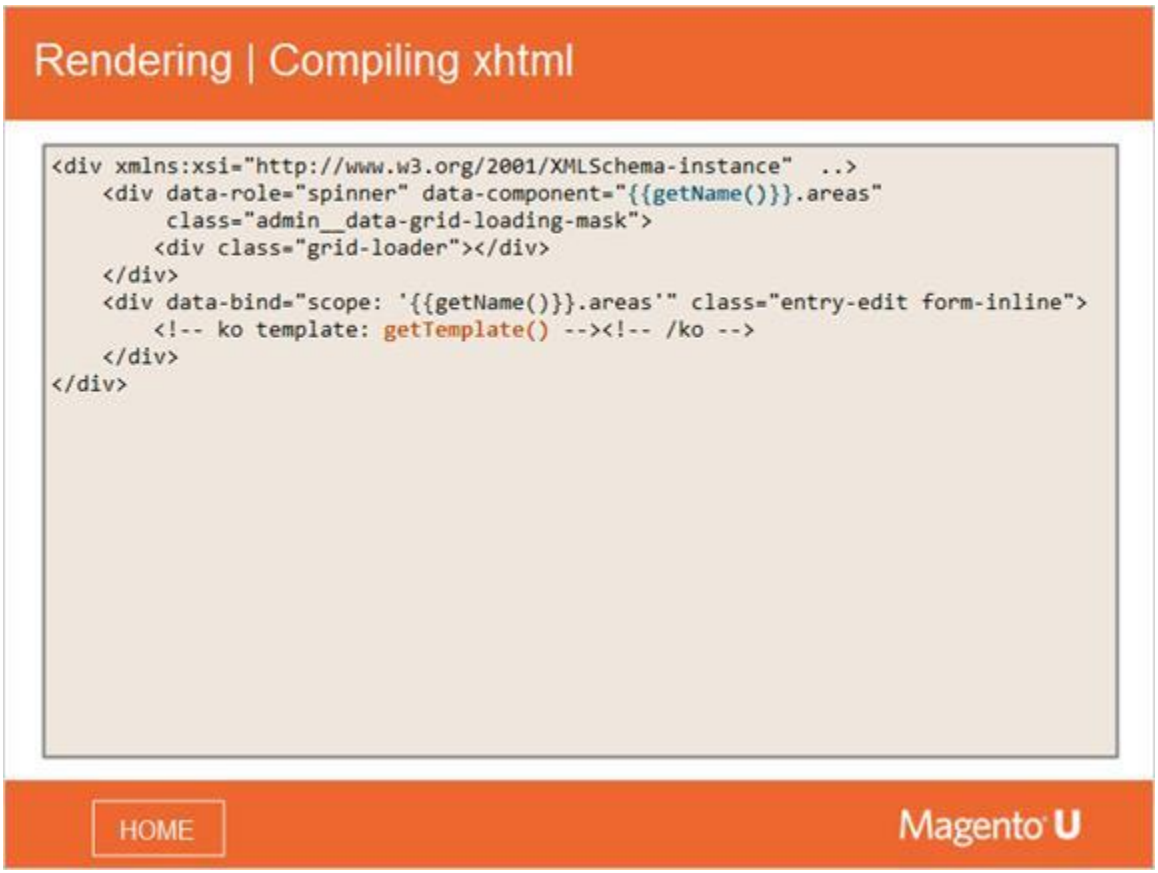
Notes:

This slide presents a high-level diagram of the rendering process.

> **Note:** This diagram only includes “PHP rendering,” which results in `*.xhtml` file processing. The “real” rendering is conducted by JavaScript.

Usually, when working with a UiComponent, you won’t interact with the classes depicted (`Compiler`, `TemplateEngine`), so they will not be covered in detail in this course.

4.16 Rendering | Compiling xhtml



Notes:

`{{getName()}}`
The PHP compiler parses all statements contained within {{..}} and either executes corresponding methods or obtains required variables.

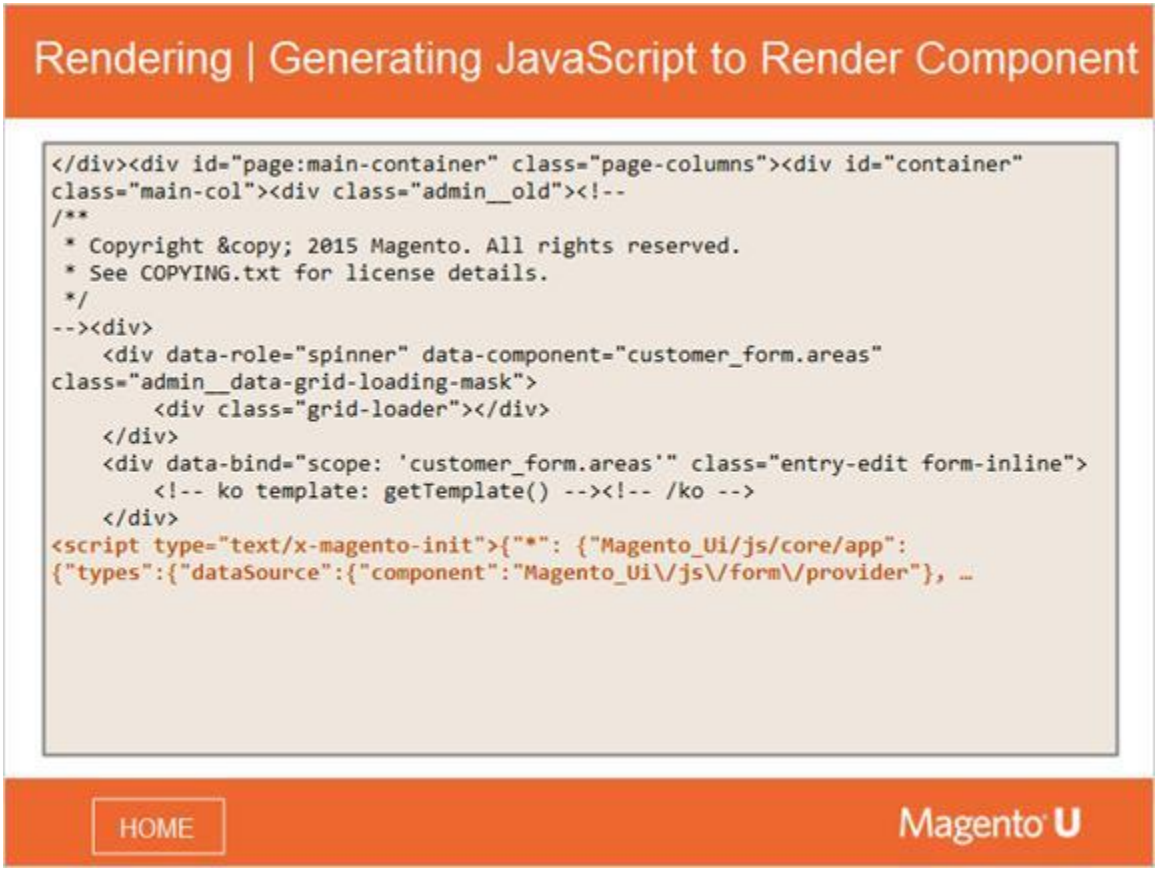
`getTemplate()`
The knockout templates engine will obtain a real component's template (html file) based on the scope.

Here we can see exactly what the Magento compiler does with the xhtml template. It processes all the {{..}} statements, as discussed earlier.

Every statement is either a call to a method of the UiComponent's class, or to a variable.

- **Note:** The `getTemplate()` is not inside of {{..}} so it will not be processed by PHP. It will be processed by the knockout templates engine when a page is loaded in a browser.

4.17 Rendering | Generating JavaScript to Render Component



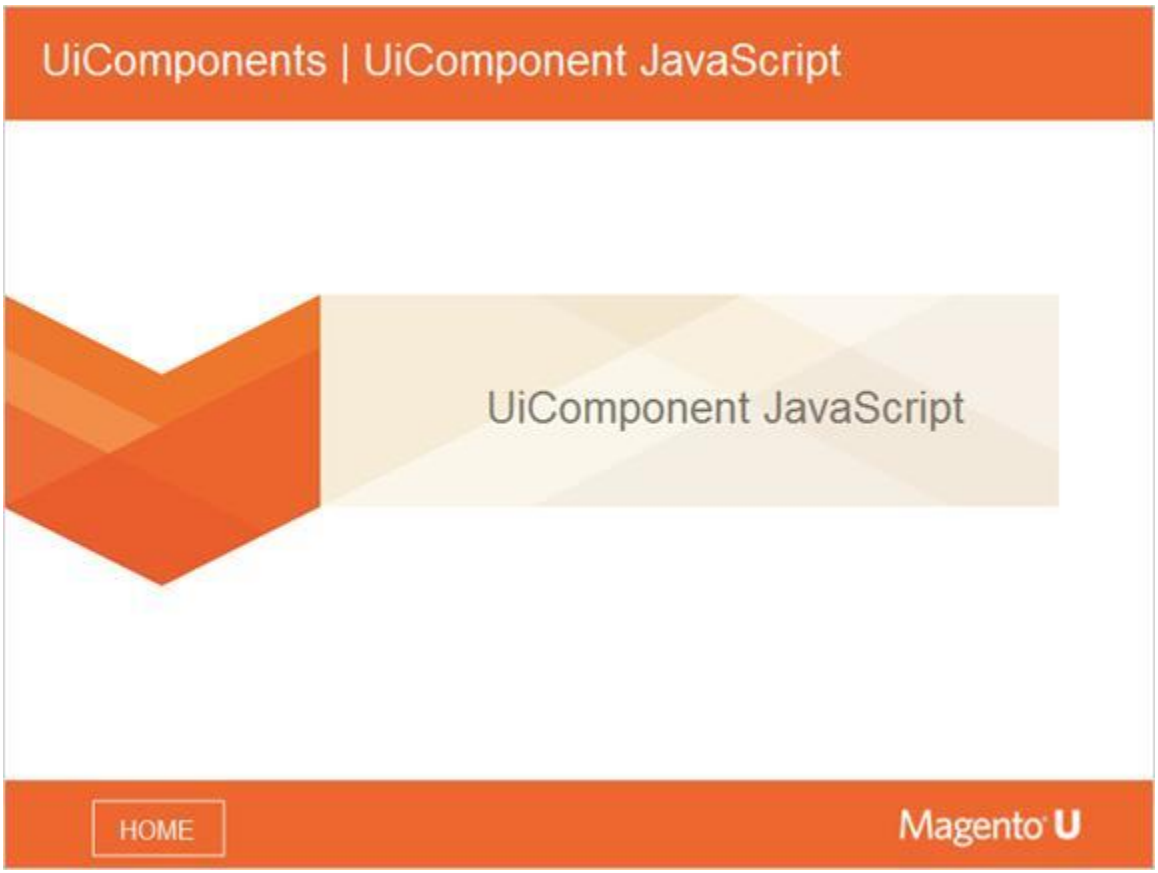
Notes:

This slide shows us the result of an xhtml compilation, which is delivered to the browser.

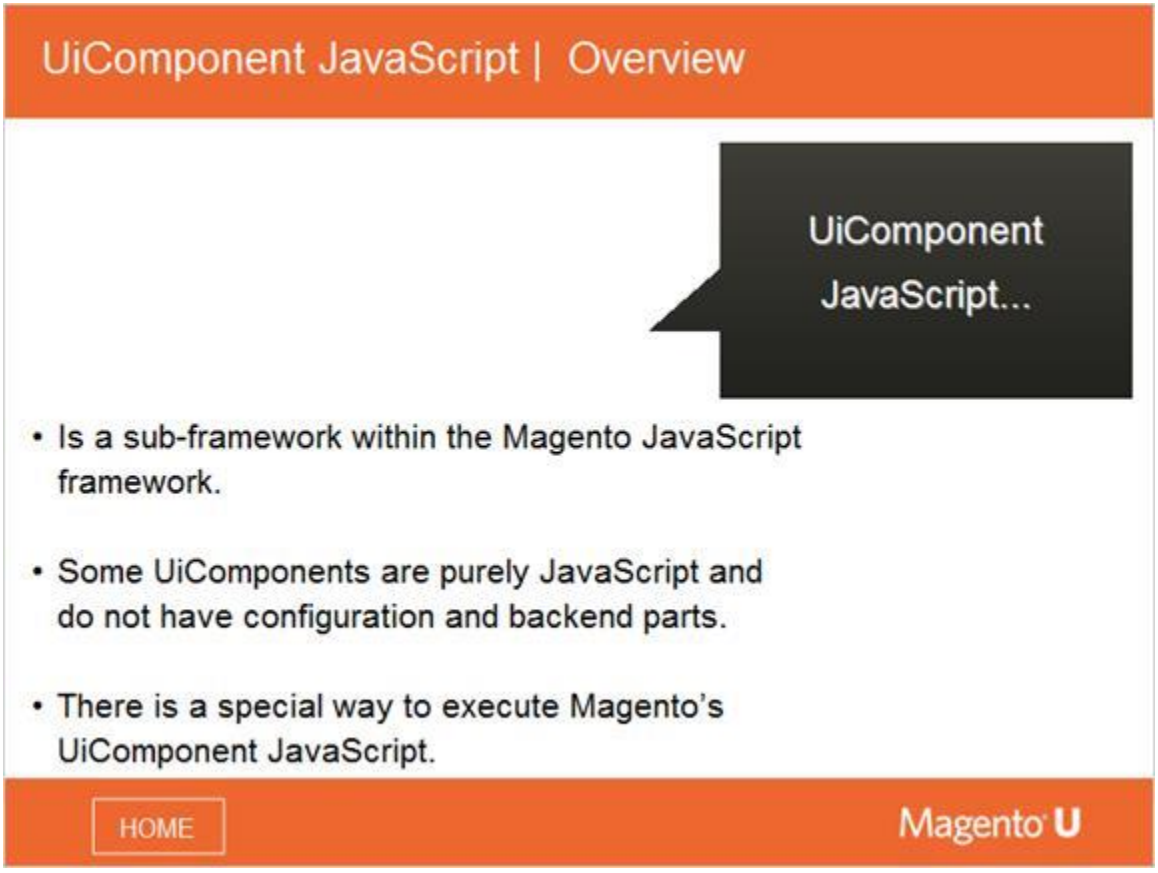
As you can see, there are no {{..}} statements but there is still `getTemplate()`.

Pay particular attention to the `<script>` code at the bottom. This lengthy line of JavaScript includes configuration of the component, its children, and all the data exported from the `DataProvider`.

4.18 UiComponents | UiComponent JavaScript



4.19 UiComponent JavaScript | Overview



Notes:

Earlier in this section, we discussed two of the three types of JavaScript modules in Magento 2:
- Flat modules
- Jquery widgets

Now, we'll take a brief look at the third type, UiComponent JavaScript.

This is very complex topic and requires a deep knowledge of JavaScript (which is beyond the scope of this course).

UiComponent is a special type of a module that has its own structure, and uses its own JavaScript sub-framework.

Because of its complex structure, UiComponent is also executed in a unique way, differing from what we've seen before (`require` function, `data-mage-init` attribute, and `text/x-magento-init` property of `<script>`).

4.20 UiComponent JavaScript | Code Example

UiComponent JavaScript | Code Example

```
define([
   'ko',
   'underscore',
   'Magento_Ui/js/lib/spinner',
   'rjsResolver',
   'uiLayout',
   'uiCollection'
], function (ko, _, loader, resolver, layout, Collection) {
   'use strict';

   return Collection.extend({
        defaults: {
            template: 'ui/grid/listing',
            stickyTmpl: 'ui/grid/sticky/listing',
            viewSwitcherTmpl: 'ui/grid/view-switcher',
            positions: false,
            displayMode: 'grid',
            displayModes: {
                grid: {
                    value: 'grid',
                    label: 'Grid',
                   template: '${ $.template }'
                }
            },
```

HOME

Magento U

Notes:

This code from `Magento/Ui/view/base/web/js/grid/listing.js` is an example of the listing component's JavaScript.

We won't describe the structure of UiComponent's JavaScript framework, as this topic is beyond the scope of this course.

However, this example does demonstrate that the UiComponent JavaScript module is different from the other JS modules we've seen before.

Note: Some of the elements of the UiComponent JavaScript framework, such as `uiLayout`, `uiCollection`, and so on. Typically, UiComponent JavaScript returns an extension of either a `UiCollection` or `UiElement` object.

4.21 UiComponent JavaScript | Execution

UiComponent JavaScript | Execution

To execute UiComponent JavaScript, the following things must happen:

- The backend `UiComponent` PHP class must generate its configuration from `definition.xml` and instance-specific XML file (for example, `product_listing.xml`).
- The backend `DataProvider` must generate data for the instance to use.
- All this data must be exported to a page to become available to the component's JavaScript module.

HOME

Magento U

Notes:

We have learned enough about UiComponents to know that its execution is not a simple process.

This slide describes key steps that must be taken to get UiComponents executed and rendered on the page.

4.22 UiComponent JavaScript | Execution Example

Notes:

This code from `Magento/Ui/view/base/web/js/grid/listing.js` is an example of the Listing component's JavaScript.

This is a small portion of the Product Listing UiComponent's execution script from the products grid page.

Please take the time to visit that page, and examine all the related JavaScript and its structure. You will notice it is a huge amount of JS, and includes both data and configuration.

It is very important to focus on the JS execution itself. You will see that it is executed through the special module: `Magento_Ui/js/core/app`. This module is used to execute UiComponent JavaScript, and differs greatly from the other UiComponent JavaScript modules in Magento 2.

4.23 Reinforcement Exercise (3.5-4-1)

Notes:

See your course Exercises Guide for instructions on how to complete this exercise, and the solution.

4.24 End of Unit 3.5

Notes: