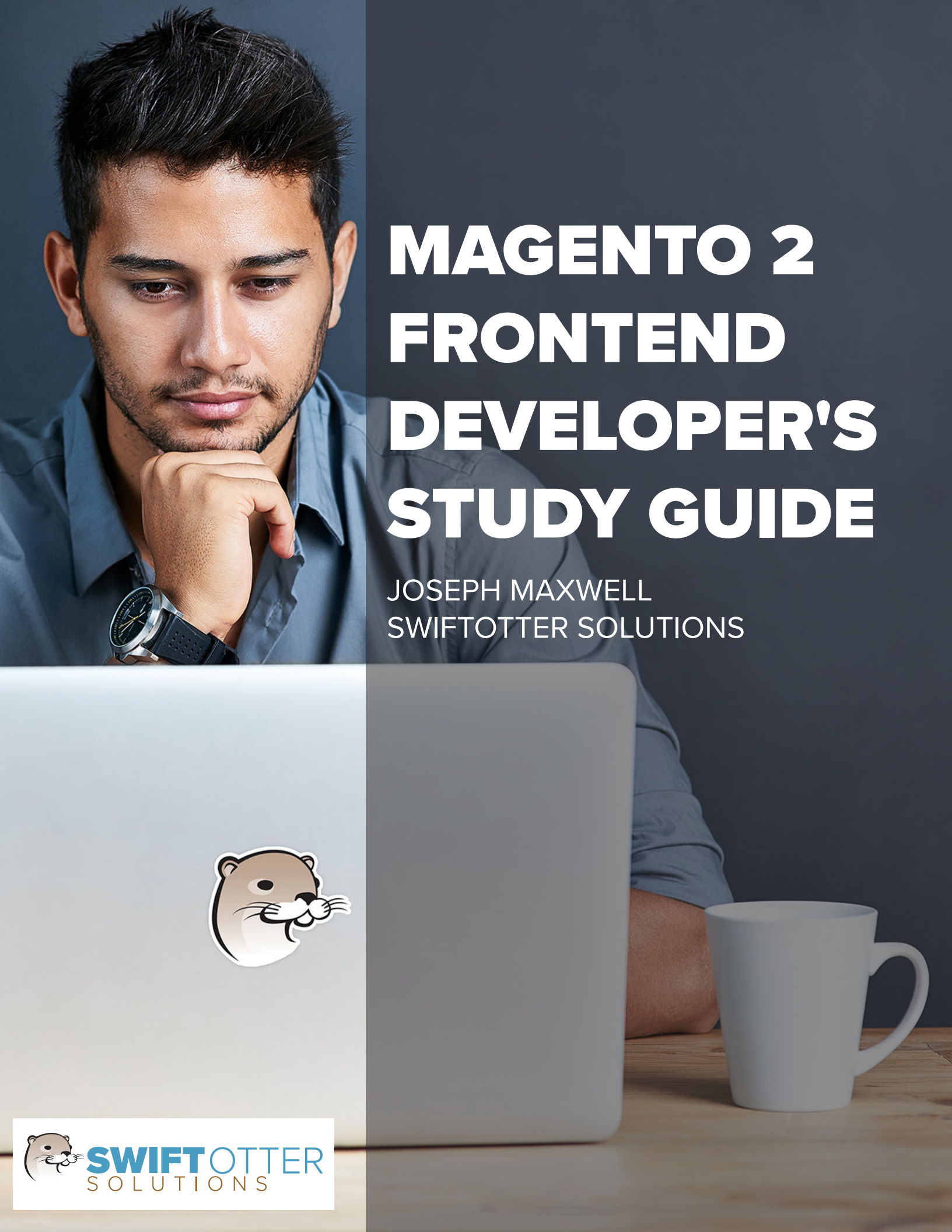

MAGENTO 2 FRONTEND DEVELOPER'S STUDY GUIDE

JOSEPH MAXWELL
SWIFTOTTER SOLUTIONS

SWIFTOTTER SOLUTIONS

INTRODUCTION

You downloaded the most comprehensive study guide for Magento 2 Frontend Development. The material covered provides excellent methodology for training developers who are new to these concepts.

This study guide answers the questions presented in Magento's study guide for this test. Reading this study guide is part of solid preparation for passing the test but does not guarantee a passing grade. What should yield a passing grade is extensive experience with Magento's frontend combined with rigorous and careful review of this study guide.

Most people who spend $260 on a test want confidence that they are likely to pass it. Therefore the next step of taking our practice test is critical. The practice test validates that you are ready for the test. Following completion of the practice test, you see questions you answered correctly and the ones you didn't.

Additionally, for a small fee, you can see your scores on the practice test by objective (showing your weak areas) and get the objective scores emailed to you. Think about it. After you take the real test, didn't you want to know where you were weak? Now, by taking SwiftOtter's practice test and choosing the $10 upgrade, you can know before you take the test how you are doing and where to study more.

All the best!
Joseph Maxwell

WE ARE SWIFTOTTER

We are focused, efficient, solution-oriented and use Magento as the agent for solving medium-sized ecommerce merchant's challenges. New sites, migrations and maintenance are our bread and butter. Our clients are our friends. Simple.

In addition, we provide second-level support for merchants with in-house development teams. While moving development in-house can save money, it often leaves holes in the support system. We patch those holes by being available to quickly solve problems that might be encountered.

This study guide demonstrates our commitment to excellence and our love for continuous learning and improvement. Enhancing the Magento developer community is good for everyone: developers, agencies, site owners and customers.

ACKNOWLEDGEMENTS

I would like to extend heartfelt gratitude to the following individuals who contributed in edits or comments to the development of the study guide. These people graciously donated their time to help them out. If you see them at a conference or on Twitter, please thank them.

I want to especially thank Jesse Maxwell, my brother, for his technical and grammatical review of this book. His deep frontend experience was invaluable.

- Vinai Kopp (Mage2.tv)
- Maria Kern (Netz98)
- Mark Cotton (Absolute Design)
- Ben Robie (DEG)
- Javier Villanueva (Media Lounge)

CONTENTS

Magento 2 Certified Professional
Frontend Exam Preparation eBook

1. CREATE THEMES

(4 QUESTIONS - 7%)

SWIFT OTTER SOLUTIONS

Themes in Magento 2 control much of the look and feel for the frontend and the backend. They use a combination of PHP, XML, HTML, CSS and JavaScript to achieve the desired look. Themes override or extend existing PHP, HTML, CSS and JavaScript while providing extra functionality.

Themes should only be for applying changes to existing code. New functionality should go into modules specific to the use case.

When compared with Magento 1, themes have many similarities: they are found in the app/design folder, they are structured in a package/theme path, and they use resemblant layout XML syntax.

Themes in Magento 2 also represent improvements, such as the use of containers, better modularity and extensibility, and more control over settings, to name a few.

1.1 DESCRIBE FOLDER STRUCTURE FOR LOCAL AND COMPOSER-BASED THEMES

Points to remember:

- Local themes are stored in app/design.
- Composer-based themes can be stored anywhere.
    - As default unless a specific alternative is stated, Composer based modules / themes will be located in the `vendor/` directory.
- `etc/`, `i18n/` and `web/` are directories found in most themes.

Local themes are stored in the `app/design` directory. If a theme is loaded through Composer, that theme can be located anywhere on the file system, but in most cases will use the default `vendor/` directory within Magento.

Magento uses the Composer autoloader. If you look at the Luma theme's composer.json, you will see the `autoload.files` node, whose value is `registration.php`.

As the Magento application starts up, Composer executes each file as specified in the `autoload.files` section. `registration.php` then registers itself as a theme. The theme is now available.

1.2 DESCRIBE THE DIFFERENT FOLDERS OF A THEME

`etc/`: this folder usually has one file: `view.xml`. `view.xml` provides configuration values for the theme in a structured format.

`i18n/`: this folder contains translations for the theme.

`media/`: this folder usually has one file: `preview.jpg`. The preview image provides a sample of what the theme will look like when activated.

`web/`: the files and directories here will be eventually downloaded by website visitors. In one form or another, they will ultimately be accessible from `pub/static`. LESS files will first be placed in `var/view_preprocessed` before being compiled and found in `pub/static`.

As a rule of thumb, Magento recommends not to use this directory but rather place customizations of the theme into the appropriate directory within the module directory where the functionality originates (for example, checkout customizations should be placed under the `Magento_Checkout/web directory`).

`css/`: location of base Magento stylesheets. These will be exported to the `pub/static/[area]/[package]/[theme]/[locale]/css` directory.

`css/source/`: LESS files that implement styles for basic UI elements. Most of these styles are mixins for global elements from the Magento UI library. `theme.less` is also located here, which overrides values for the default variables.

`fonts/`: web fonts that will be utilized in the theme.

`images/`: images that are included in the theme. These are images that will not frequently change. For example, you would include an icon here, but not a free shipping banner.

`js/`: theme-specific JS.

Module overrides:

When developing a theme, you will likely need to override another module's assets. These overrides reside in the theme folder, then the module's name. For example, in our SwiftOtter_Flex theme, we need to override `addtocart.phtml` in `Magento_Catalog`.

To accomplish this, we would copy `addtocart.phtml` into `app/design/SwiftOtter/Flex/Magento_Catalog/templates/product/view`.

The result is an easy-to-remember and replicable directory scheme.

HELPFUL LINKS:

- https://devdocs.magento.com/guides/v2.2/frontend-dev-guide/themes/theme-structure.html

1.3 DESCRIBE THE DIFFERENT FILES OF A THEME

`composer.json`: this provides basic instructions, telling Composer information about the module. The most important information is the `autoload.files` node where Composer is informed about `registration.php`.

`registration.php` (required): this registers the module as a theme with Magento. It is important to note that the component name is made up as follows:

- `adminhtml` or `frontend`
- /
- Package name (often the developing company)
- /
- Theme name

Examples:

- `frontend/SwiftOtter/Flex`
- `frontend/Magento/luma`
- `adminhtml/Magento/backend`

`theme.xml` (required): this file describes the theme to Magento. You will see a `title` node, a `parent` node (optional), and a `media/preview_image` (optional) node.

HELPFUL LINKS:

- https://devdocs.magento.com/guides/v2.2/frontend-dev-guide/themes/theme-structure.html

1.4 UNDERSTAND THE USAGE OF MAGENTO AREAS: ADMINHTML/BASE/FRONTEND

Local themes are found in one of two folders: `app/design/frontend` (available on the store front) or `app/design/adminhtml` (available in the admin panel).

However, a module's assets can either be made available to both the `adminhtml` and `frontend` areas or just one of the two areas individually. Files in the `view/base` directory are available to both the frontend and adminhtml areas.

Placing a file in the `frontend` or `adminhtml` directory, will override that file in the base theme, if it exists there.

The current area is determined in the `App\State class`, however this knowledge is not expected from a frontend developer.

HELPFUL LINKS:

- https://devdocs.magento.com/guides/v2.2/frontend-dev-guide/themes/theme-overview.html

Magento 2 Certified Professional
Frontend Exam Preparation eBook

2.
MAGENTO DESIGN CONFIGURATION SYSTEM

(4 QUESTIONS - 7%)

SWIFT OTTER SOLUTIONS

2.1 DESCRIBE THE RELATIONSHIP BETWEEN THEMES

What type of relationships can exist between themes?

Parent/child.

In the theme's theme.xml file, you can specify the `<parent/>` node, like: `<parent>Magento/blank</parent>`. As such, a theme can be a part of (or the end of) many layers of other themes.

This hierarchy is what is used to determine the fallback sequence for theme inheritance (see here and here).

What is the difference between a parent theme and a child theme?

The only difference is that the child theme is the theme that is currently selected for display in the specified area. Any theme can be chosen for display (whether or not it specifies a parent in `theme.xml`).

HELPFUL LINKS:

- https://devdocs.magento.com/guides/v2.2/frontend-dev-guide/themes/theme-inherit.html

How can the relationship between themes be defined and influenced?

The relationship is determined by the presence of the `<parent/>` node in `theme.xml`.

How is that taken into account when creating a custom theme or customizing an existing theme?

This information is useful to determine the fallback sequence. Fallback is used to determine a file to load in a sequence of other themes when the file does not exist in the current theme. The actual mechanism varies between file types (different for layout XML vs template files).

2.2 CONFIGURE THE DESIGN SYSTEM USING THE OPTIONS FOUND IN THE ADMIN UI UNDER CONTENT > DESIGN > CONFIGURATION

How do the configuration settings affect theme rendering?

Configuration settings provide an easy-to-use interface for detailing options for the theme that is applied to a store. They are **not** theme configuration, but rather store design configuration.

TIP: These settings are stored in the `core_config_data` table but are not found in Store > Configuration.

The configuration values are displayed in a grid format showing the website > store > store view hierarchy with an option to adjust configuration at each point.

These values are retrieved like any other store configuration value (see here, `getDefault`).

For backend developers, see here and here.

What happens if a theme is added or removed?

Even though `registration.php` is a required file in a theme directory, themes are not managed in the same way modules are. When you run `php bin/magento module:enable --all`, you will see that the new theme is not listed in the list.

The theme is automatically added to the `theme` table in the database. Themes are available in the admin panel's Content > Design > Configuration and can be selected in the site's Applied Theme dropdown list.

If a theme is removed, the default theme will automatically be used, but the theme's database record is not automatically removed.

HELPFUL LINKS:

- https://devdocs.magento.com/guides/v2.2/frontend-dev-guide/themes/theme-install.html

What common mistakes can be made in regard to these settings?

Here are a few mistakes that would be easy to make:

- Assuming that these store design configuration values are stored in another table.
- Assuming that these store design configuration values are associated with a theme and NOT a store.
- Not configuring the appropriate values for each store.

HELPFUL LINKS:

- https://docs.magento.com/m2/ee/user_guide/design/configuration.html

2.3 APPLY A TEMPORARY THEME CONFIGURATION TO A STORE VIEW USING THE OPTIONS FOUND IN THE ADMIN UI UNDER CONTENT > DESIGN > SCHEDULE

What is the purpose of this feature?

The purpose of this feature is to automate site design changes for specific occasions. For example, a merchant may configure a custom theme to display for the duration of the Christmas season.

How does it influence rendering if a design change is scheduled?

On the day the change is scheduled to start, the selected theme is configured as the primary theme for that store. Note that only one design change can be

scheduled at a time. If the start and end dates overlap another design change's start and end dates, Magento will throw an error and prevent you from saving.

```
The date range for this design change overlaps another design change for the specified store.
```

https://github.com/magento/magento2/blob/2.2-develop/app/code/Magento/Theme/Model/Design.php controls the design that is applied onto the website. Specifically, the loadChange() method responds with the currently active design change. This triggers the `theme_save_after` which clears the cache.

When the design change is scheduled, unless Varnish is enabled, the design change will be applied and removed automatically (see this page).

What happens at the end of a scheduled design?

Nothing. If FPC is disabled (should be only in development), the design change lookup will indicate that there is no design change or locate the next one to be utilized.

How is full page caching involved?

It is our understanding that full-page caching overrides the design changes, until the cache is purged or expires.

2.4 UNDERSTAND THE DIFFERENCES AND SIMILARITIES BETWEEN CONTENT > DESIGN > CONFIGURATION AND > SCHEDULE TO CONFIGURE THE DESIGN FALLBACK

What is the effect if both options are used at the same time?

The theme selected in Design Schedule overrides the theme in Design Configuration. However, while Design Configuration has no sense of time, the Design Schedule applies the theme between the dates specified.

Magento 2 Certified Professional
Frontend Exam Preparation eBook

3.
LAYOUT XML IN THEMES

(11 QUESTIONS - 18%)

SWIFT OTTER SOLUTIONS

3.1 DEMONSTRATE KNOWLEDGE OF ALL LAYOUT XML DIRECTIVES AND THEIR ARGUMENTS

What layout XML elements exist and what is their purpose?

The basic building blocks of layout XML elements are `<block/>` and `<container/>`. Most other layout instructions make modifications to blocks and containers.

Containers

Containers contain or group blocks and other containers. If you look in the root layout XML file, you will see the top-most element is a container.

Containers are helpful in providing a more extensible means of grouping and rendering child blocks because the HTML tag and class can be specified in XML. Using blocks with an iterator, or a ListText block does not have this advantage.

M1

MAGENTO 1

If you are coming from Magento 1, a great example of how containers are now leveraged is found in the difference of how the product page is rendered. While the output is quite similar, the layout XML behind it is very different:

- Magento 1
- Magento 2

However, the ListText block has the advantage that data values can be set via layout XML `<arguments>` node. Containers don't allow for `<arguments>`.

Notable container attributes (more)

`name`: required. This defines how you will reference this container from other areas in the system.

`htmlTag`: a tag to wrap the child block output. Required if `htmlClass` is specified.

`htmlClass`: the class(es) to apply to the wrapper HTML tag.

ReferenceContainer

The extensible aspect of a container is to be able to reference it throughout layout XML. This capability is leverged with the `<referenceContainer/>` tag (example).

```
    <referenceContainer name="product.info.type">


        <block name="additional.product.details"
    template="SwiftOtter::product.details.phtml">


            <arguments>


                <argument name="viewModel"
    type="xsi:object">SwiftOtter\Flex\Block\ProductDetails</
    argument>


            </arguments>


        </block>


    </referenceContainer>
```

The above example will include a new logic class (using the recommended Template Block + ViewModel approach) to render additional product details. The template from this example is automatically rendered to the page within the container `product.info.type`.

Blocks

Blocks are a foundational building unit for layout in Magento. They are the link between a PHP block class, which contains logic and a template which renders content. Blocks can have children and grandchildren (and so on). Information can be passed from layout XML into the block via the `<arguments/>` child node. For those of a hybrid discipline (backend developers) this could include view models or some static values to be utilized in the template.

Notable block attributes (more)

- `class`: defaults to \Magento\Framework\View\Element\Template and does not need to be specified. In fact, Magento recommends against specifying a class (see note in the header comments for the Template class in case the block only is required to output a template).
- `name`: used to interact with the block from other locations in the frontend.
- `before` / `after`: places the block before or after another element, as identified by its name. Or, you can specify a "-" (dash) to place the block at the beginning or end respectively.
- `template`: the path to the template. You should always use Magento module path notation. Example: `SwiftOtter_Hero::hero.phtml`. If the

module name is omitted Magento will attempt to determine it based on the block name, which fails if a generic block class is used or the block class is changed with layout or di XML (more details).

- `cacheable` (default `true`): if specified as `false`, the entire page will not be cached with Full Page Cache. Because this negatively impacts performance, it is best to never add to blocks that are on pages where the caching is applied. Instead, follow the directions in this Magento Stack Exchange post on how to utilize Magento's ESI system if the block content is public, that is, the identical for all visitors, or use customer-data sections in case the content is private (that is, specific for each visitor). More information on including private data on cached pages can be found at this link.

HELPFUL LINKS:

- https://devdocs.magento.com/guides/v2.2/frontend-dev-guide/layouts/xml-instructions.html

ReferenceBlock

Like containers, referencing another block allows you to affect the output of another block.

`<referenceBlock name="product.info.sku" remove="true"/>`: completely removes the block from layout.

`<referenceBlock name="product.info.sku" display="false"/>`: prevents a block from displaying (the associated PHP classes are still loaded).

```
<referenceBlock name="product.info.sku">

    <arguments>

        <argument name="template"
xsi:type="string">SwiftOtter_Flex::new-sku.phtml</argument>

    </arguments>

</referenceBlock>
```

The above example shows how to change the template of an existing block. Note that changing the template here will break fallback patterns that depend on the original file chosen. As such, this might be problematic for module developers.

For example, let's say you are developing a module and want to redirect the SKU's template to a custom one in your module. A merchant, who uses your module, installs a custom theme. When they browse to the product page, the SKU might not look correct because the theme they installed did not take into account the updated SKU template.

What is the purpose of the attributes that are available on blocks and other elements?

The idea is to place configuration into one location (alongside the other configuration that is controlling the look of the page). Layout XML configures containers and blocks. It would be easier to maintain extra details for containers and blocks when they are not buried in a mass of HTML. In addition, this opens

up the possibility for module developers to easily adjust these parameters without having to override many templates.

Other important layout instructions:

- `<update handle="customer_account"/>` (example): this includes instructions from another layout handle.
- `<move element="existing.element.name" destination="target.element"/>`: this moves an element (block or container) into another element (block or container).
    - The `move` element also contains `before` and `after` attributes so you can change the element's placement within a parent if so desired.

3.2 DESCRIBE PAGE LAYOUTS AND THEIR INHERITANCE

How can the page layout be specified?

The page layout is specified in a layout XML file, in the root `<page/>` node, like: `<page layout="2columns-left" ...></page>`

What is the purpose of page layouts?

The purpose is to create a structured and common set of layout instructions to render pages. Most pages on a website can be categories as fitting into a 1 column,

2 column, or 3 column container system. These page layouts can be selected throughout the admin panel to provide a specific layout by page.

How can a custom page layout be created?

To create a custom layout you must first define the layout, define the common layout instructions, and then utilize that layout (as seen in the first answer to this section). In our example below, we will create a text layout. This is similar to the two column layout example except the text area is narrower.

Page layouts only contain containers. This is different than the page configuration (XML files with a layout handle as their file name, and stored in `view/[area]/layout`) which can contain blocks and containers.

The definition of the layouts are stored in your module's `layouts.xml` file, like `app/code/SwiftOtter/Test/view/layouts.xml`. See the following example.

Defining the layout:

In a module's `view/[area]` directory, create a `layouts.xml` file:

```
<!-- app/code/SwiftOtter/Test/view/layouts.xml -->


<?xml version="1.0" encoding="UTF-8"?>


<page_layouts xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:View/
PageLayout/etc/layouts.xsd">


    <layout id="text">
```

```
        <label translate="true">Text layout</label>


    </layout>


</page_layouts>
```

Defining the common layout instructions:

```
<!-- app/code/SwiftOtter/Module/view/page_layout/text.xml
-->


<?xml version="1.0"?>


<layout xmlns:xsi="http://www.
w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:View/
Layout/etc/page_layout.xsd">


    <!-- notice foundation page layout of 2columns-right:
-->


    <update handle="2columns-right"/>


    <!-- Instead of adding to body tag which isn't possible
from page_layout xml -->


    <referenceContainer name="page.wrapper"
htmlClass="page-wrapper page-layout-2columns-right layout--
text"/>


</layout>
```

Where are the existing page layouts used?

Page layouts can be customized in the admin panel on products, categories, CMS pages.

How can the root page layout be specified for all pages and for specific pages?

Set the `layout` parameter on the `<page />` node to the ID of a layout specified in one of the `page_layout` XML files. The node can otherwise be blank if desired. Specific pages can be targeted using the layout XML's filename.

HELPFUL LINKS:

- https://devdocs.magento.com/guides/v2.2/frontend-dev-guide/layouts/layout-types.html

3.3 DEMONSTRATE UNDERSTANDING OF LAYOUT HANDLES AND CORRESPONDING FILES

A layout handle is a string that associates a layout XML file with a specific page or group of pages. It is how Magento correlates the fact that you are on the customer account page with needing to render the instructions found in this file.

While layout handles can be manipulated, Magento provides plenty of sensible defaults. The one you will most often see is derived from the path of the page, or more specifically, the controller that renders the page.

For the example below, we will look at the following URL: https://yoursite.test/customer/account. When browsing here, the `<body/>` tag contains the following classes: `account customer-account-index page-layout-2columns-left`.

The first thing to notice is the use of the `module/controller/action` syntax. If one of these parameters are not specified `index` is used instead.

The customer_account_index layout handle is built from the following elements:

- `customer`: the route ID, see here. Please note that the route ID can be different than the front name. When browsing to a page, the front name is used to match the URL with the controller. When building layout handles, the route's ID is used.
- `account`: the folder as part of the controller.
- `index`: the controller's action, as it was not specified in the URL above: see this link.

So how does this map to a layout XML file? Layout XML files are stored in a module's `view/[area]/layout` directory.

We can then take the layout handle and look for a file in the `layout` directory with that name. If you are using PHPStorm, a fast way to do this is by tapping the `Shift` key twice and entering the filename you wish to locate.

Any of the layout handles listed above can affect the output of a page. The `customer_account_index.xml` file will be specific to the customer dashboard.

How can the available layout handles for a given page be determined?

To determine what controller based layout handle applies to the current page, the quickest method is to look at the body class. One of the classes contains the path. Exchange the dashes for underscores to get the layout handle. For instance, the product page has a class of `catalog-product-view`. The layout handle would then be `catalog_product_view`.

To get the full list of handles that are available, briefly add the following to a template that is rendered on the page: `<?php var_dump($block->getLayout()->getUpdate()->getHandles()) ?>`. We are including another way to get the layout handles, below in the “What are the most commonly used layout handles?” section.

How do you add a new layout handle?

As detailed above, most layout handles are automatically generated. However, it is quite easy to add your own during the construction of a request.

To add a new layout handle, simple use the `update` layout instruction:

```
<update handle="new_handle_name"/>
```

What is the purpose of layout handles?

See above. The purpose is to connect layout XML instructions with a controller. This also decouples the rendering of HTML on the frontend, making it easy to control what is shown, where.

What are the most commonly used layout handles?

If you are on a development machine and wish to see every layout handle that is utilized for a page, do the following:

- Open `vendor/magento/framework/View/Model/Layout/Merge.php`
- Locate the `addHandle` method.
- After the `foreach`, add: `echo $name . "<br/>";`
- After the `else`, add: `echo $handleName . "<br/>";`

For quick reference, here are some common layout handles.

Home Page:

- default
- cms_index_index
- cms_page_view
- cms_index_index
- cms_index_index_id_home
- 1column
- catalog_product_prices

Product Page:

- default
- catalog_product_view

- catalog_product_view_type_simple
- catalog_product_view_id_16
- catalog_product_view_sku_24-UG07
- 1column
- catalog_product_prices

In reviewing the above layout handle lists, you will see that there are some common entries. `default` is a layout handle that is available everywhere. If you add `view/[area]/layout/default.xml` to your module, every page in that area (`frontend` or `adminhtml`) will be affected.

Notice that the page layout is also included as a layout handle.

How can layout handles be used during theme customization?

Layout handles are a fantastic way to group updates together. For example, you can create a new layout handle to display specific information on a product page. Unless you instruct the Magento Layout model to include that handle, no changes appear. But, say, in a plugin, you tell the Layout model to include the handle, all changes associated with that layout handle will take effect on the website.

3.4 UNDERSTAND THE DIFFERENCES BETWEEN CONTAINERS AND BLOCKS

What is the purpose of blocks? What is the purpose of containers?

A block represents the end of the chain in rendering HTML for Magento. Containers contain blocks and can wrap them in an HTML tag. Also, containers will not render any output if there are no children assigned to them.

How can containers be used in theming?

As discussed above, a container holds blocks or containers. The container is helpful for eliminating useless HTML from PHTML files and instead putting that into the logical grouping where other blocks reside: a container.

How can blocks be used in theming?

Blocks handle all static-HTML rendering that containers do not handle.

What is the default block type?

\Magento\Framework\View\Element\Template

How can the order of rendered child blocks be influenced both in containers and in blocks?

The order is influenced through the use of the `before` and `after` attributes. You can specify a block or container's name or you can use the `-` (dash) symbol to denote at the beginning or the end.

HELPFUL LINKS:

- https://devdocs.magento.com/guides/v2.2/frontend-dev-guide/layouts/xml-instructions.html

3.5 DESCRIBE LAYOUT XML OVERRIDE TECHNIQUE

How can layout XML be overridden?

It is not ideal to override layout. However, as described by this article, there are some situations where overriding XML layout is inevitable.

For example, to completely erase the existing Magento product page layout in our SwiftOtter Flex theme, you would create a new XML file in: `app/design/SwiftOtter/Flex/Magento_Catalog/layout/override/frontend/catalog_product_view.xml`.

Note that these files are placed into the `override` folder instead of directly inside the `layout` folder. XML files that are directly placed in the `layout` directory are merged, while XML files that are placed in the appropriate subdirectory of `override` replace the original file.

All layout instructions from the Catalog module's original `catalog_product_view.xml` will be eliminated.

How can layout overriding be used in theming?

If an existing layout file contains an instruction that you cannot change by extending, then overriding that layout file might be the only recourse. In our experience with building Magento websites, we have yet to come across a time where this is necessary.

What are consequences of layout overrides during upgrades?

New blocks or containers could be added to the core layout file with a new version. If that file was overridden, other modules that depend on specific elements would not have access to them (and their functionality would be halted) unless you manually added them to your theme's override.

What is the effect of layout overrides on compatibility?

Overriding layout files circumvent any changes in core files and increase chances of trouble during upgrades. As such, it must be viewed as the ultimate last result.

3.6 UNDERSTAND LAYOUT MERGING

What is layout merging?

Layout merging is the process of assembling all of the layout XML files into one large XML document (https://github.com/magento/magento2/blob/2.2-develop/lib/internal/Magento/Framework/View/Model/Layout/Merge.php `_loadFileLayoutUpdatesXml` method). When rendering the page, Magento locates the layout instructions for the particular handles (see `_fetchPackageLayoutUpdates`) and merges them together. Magento then traverses this document instantiating PHP classes for each block and then rendering the output.

Layout is merged in order that the modules are loaded. To quickly see the loading order of modules, check the `app/etc/config.php` file. If two conflicting instructions are given, the module that loads last wins.

When you are building a new theme, you will need to make adjustments to the layout XML. While you can override layout XML files (see discussion above), this is less than ideal for upgradeability purposes as any updates that Magento ships will not be present in the overridden files.

Instead, utilize the layout XML merging directories. For example: `app/design/SwiftOtter/Flex/Magento_Catalog/layout`. We can create layout XML files that will be merged with layout XML files in the `Magento_Catalog` module. As such, we can use instructions like `<move element="element.to.move" destination="can.be.same.parent" after="element.to.be.placed.after"/>`

If you want to see the results of the layout merging on your developer machine, do the following:

- Navigate to `\Magento\Framework\View\Result\Page`
- Locate the `render()` method.
- After the `$output = $this->renderPage();` method, add:
- `$output .= $this->getLayout()->getXmlString();`

Go to a page on the website and view source. The XML will be appended after the HTML tag.

How do design areas influence merging?

Layout XML instructions are first merged in the `base` area and then by the area that applies to the current request (`frontend` or `adminhtml`).

Here is the order in which layout XML is merged:

- Module base files loaded
- Module area files loaded
- Sorted according to their module priority (array index of module's position in `app/etc/config.php`)
    - If their priorities are equal, they are sorted according to their alphabetical priority.
- Theme files (from furthest ancestor to current theme)
    - Layout files loaded
    - Override files replaced

- Theme override files replaced

Source: [vendor/magento/framework/View/File/Collector/Base.php]

How can merging remove elements added earlier?

The only way to remove elements added earlier is through the `remove` attribute on the `referenceBlock` or `referenceContainer` tags:

```
<referenceBlock name="info.product.sku" remove="true"/>
```

What are additive changes and what are overriding changes during layout merging?

Additive changes are when you create new XML elements that are in addition to what has already been written. These changes will likely affect existing elements (setting new arguments, for example).

Let's look at an example from `Magento_Catalog/view/frontend/layout/catalog_product_view.xml`. Locate the `product.attributes` block in this file:

```
<block class="Magento\Catalog\Block\Product\View\
Attributes" name="product.attributes" as="additional"
template="Magento_Catalog::product/view/attributes.phtml"
group="detailed_info">


  <arguments>


    <argument translate="true" name="title"
xsi:type="string">More Information</argument>
```

```
    </arguments>

</block>
```

Let's create our own, in `SwiftOtter/Flex/Magento_Catalog/layout/catalog_product_view.xml`:

```
<?xml version="1.0"?>

<page xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:View/
Layout/etc/page_configuration.xsd">

    <body>

        <referenceBlock name="product.attributes">

            <arguments>

                <argument translate="true" name="test"
xsi:type="string">Test Argument</argument>

            </arguments>

        </referenceBlock>

    </body>

</page>
```

When setting a breakpoint in `vendor/magento/module-catalog/view/frontend/templates/product/view/attributes.phtml`, you will see that the `$block->_data` array has a new key, `test`, with the value of `Test Argument`. *This is considered an additive change.*

You can change the argument name to `title` and notice the difference in the `$block->_data` variable. As such, you are overriding the existing value for `title` and are specifying your own. *This is considered an overriding change.*

3.7 UNDERSTAND PROCESSING ORDER OF LAYOUT HANDLES AND OTHER DIRECTIVES

In what order are layout handles processed?

They are processed in the order in which the handles were added to the Merge class. The `default` handle is loaded first. Then, in order of when they were added through the request lifecycle.

In the above class, see the `load()` method. In this method, the layout is merged as it loops through the associated layout handles.

As a side note: this can cause some trouble if you are relying on a custom-injected layout handle to adjust the design on a page. As such, there may be instances where you need to go to extra lengths to inject a layout handle earlier in the application's lifecycle.

In what order is layout XML merged within the same handle?

This would then be according to the order in which the layout files are loaded. See the in-depth topic on section 3.6.

How can the processing order be influenced?

The processing order and merging of layout XML is determined by the module load order. Adjusting the `<sequence/>` setting for modules will influence the processing order of layout XML.

For backend developers, you can use plugins to inject your layout handles earlier in the process. For example, if you want to add a layout handle before the `catalog_product_view` one, you need to create a `before` plugin for `\Magento\Catalog\Helper\Product\View::initProductLayout`.

What are common problems arising from the merge order of layout declarations?

The most common problems are:

- Accidental overrides: where you specify `<block name="...">` which happens to be the same name as another block.
- Wishful overrides: where you want to override another block, but you specify a different name and the block is injected twice.

3.8 SET VALUES ON BLOCK INSTANCES USING LAYOUT XML ARGUMENTS

How can arguments be set on blocks?

Through the use of the `<arguments><argument/></arguments>` nodes. See the examples above in 3.6.

Which data types are available?

- string
- boolean
- object
- number
- null
- array

HELPFUL LINKS:

- https://devdocs.magento.com/guides/v2.2/frontend-dev-guide/layouts/xml-instructions.html#argument

What are common arguments for blocks?

`template`: sets the template for the block

`translate_inline`: true = disable translation for this block

`module_name`: sets the module for the block. Usually this is automatically determined.

`cache_key`: custom-specific key for saving / retrieving cached information. This is helpful if the block needs to be cached, but with unique caches for unique sets of data (for example, loading customer data).

3.9 CUSTOMIZE A THEME'S APPEARANCE WITH ETC/VIEW.XML

What is the etc/view.xml file used for?

The etc/view.xml file is used to specify custom and updatable properties for the theme in Magento. The majority of the values apply to image sizes and the product gallery settings.

```
    <view xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="urn:magento:framework:Config/
    etc/view.xsd">


        <media>


            <images module="Magento_Catalog">


                <image id="bundled_product_customization_page"
    type="thumbnail">


                    <width>140</width>


                    <height>140</height>
```

```
            </image>

        </images>

    </media>

</view>
```

HELPFUL LINKS:

- https://devdocs.magento.com/guides/v2.2/frontend-dev-guide/themes/theme-images.html

How can it be used to customize a theme?

In a custom theme, you can create a new etc/view.xml file and specify the elements that you wish to change. Sorry if that seems obvious.

How can values from etc/view.xml be used during theming?

Every block is equipped with a `$block->getVar($name, $module = null)` method. Using this method, you can retrieve any value as specified in etc/view.xml.

How does theme inheritance influence values from etc/view.xml?

The closest value to the end of the rope wins. Any values declared in a child theme will override the parent. Also, values declared in a specific area (frontend, for instance), will take precedence over a base theme.

Magento 2 Certified Professional
Frontend Exam Preparation eBook

4. CREATE AND CUSTOMIZE TEMPLATE FILES

(5 QUESTIONS - 8%)

SWIFT OTTER SOLUTIONS

4.1 ASSIGN A CUSTOMIZED TEMPLATE FILE USING LAYOUT XML

How can a customized template file be assigned to a block using layout XML?

```
<block name="custom.block" template="SwiftOtter_
Module::test.phtml"/>
```

The provided template path can be broken down into two arguments, which are located either side of the scope separator `::`.

The first states the area to look in, in this case `SwiftOtter_Module`, which looks in the corresponding module:

```
app/code/SwiftOtter/Module
```

The second part `test.phtml` is the path within this module's template folder (view/[area]/template). In this case, that completes the path:

```
app/code/SwiftOtter/Module/view/frontend/template/test.phtml
```

If you would like to change the template and use another one:

```
<referenceBlock name="custom.block" template="SwiftOtter_
Module::product-attribute.phtml"/>
```

How does overriding a template affect upgradability?

When overriding a template file by assigning a new template via layout XML, you effectively break the fallback path for a template. This prevents themes from modifying the template.

What precautions can be taken to ease future upgrades when customizing templates?

- Backend developers: consider a plugin for the block (this would prevent using view models in such a case) that would only turn on the customized template when necessary.
- Avoid overrides where possible.

4.2 OVERRIDE A NATIVE TEMPLATE FILE WITH A CUSTOMIZED TEMPLATE FILE, USING THE DESIGN FALLBACK

How can the design fallback be used to render customized templates?

You can customize a template by adding it to your custom theme. For example, to modify vendor/magento/module-catalog/view/frontend/templates/product/list.phtml, use the following path:

```
app/design/frontend/SwiftOtter/Flow/Magento_Catalog/view/
frontend/templates/product/list.phtml
```

HELPFUL LINKS:

- https://devdocs.magento.com/guides/v2.2/frontend-dev-guide/templates/template-override.html

How does that influence upgradability?

While the chances are usually slim, there can be broken functionality that occurs when upgrading. The reason is that the template that was overridden may have been enhanced along with other aspects of the system that depended on the template. If the overridden template is not upgraded, things can break.

The most notorious example was from Magento 1 days. SUPEE-9767 enabled form key checking on the checkout. With the patch, Magento added the form key to the templates. But many checkouts had at least some customized template files which then did not include the form key. While this was easy to catch with testing, it added to the overall implementation time for this upgrade.

How can you determine which template a block renders?

The easiest way is to enable template hints, which can be activated using the command-line: `bin/magento dev:template-hints:enable` or in the admin area, in the follow location:

Frontend:

Stores > Settings > Configuration > Advanced > Developer > Debug > Enabled Template Path Hints for Storefront > Yes

Admin:

Stores > Settings > Configuration > Advanced > Developer > Debug > Enabled Template Path Hints for Admin > Yes

The next step is to look through the Magento code. The template is often set in layout XML, so this is the best place to start. Locate the block that renders the template in XML and see if the template is set. If so, it will contain the reference to the template. In some places, though, the template is defined in the PHP class itself. If this is the case, open the block class and look for the `$_template` property or some place where it is set.

HELPFUL LINKS:

- https://magento.stackexchange.com/a/173221/13

4.3 DESCRIBE CONVENTIONS USED IN TEMPLATE FILES

What conventions are used in PHP templates?

- `$this` no longer applies to the rendering block. Use `$block` or `$block->getData('view_model')` to obtain access to the instigating object or its data.
- Always type hint variables that are automatically imported (`$block`, `$viewModel`).
- Never use squiggly braces: this is a code smell that indicates your block or view model should be doing more work.

- If you need to use a loop, use the `foreach` > `endforeach` constructs.
- Keep templates to a reasonable minimum. Massive 500 line files are a code smell.
- Magento now uses <?= instead of <?php echo.
- Always translate strings in templates that will be displayed to user.

Why aren't the common PHP loop and block constructs used?

Magento templates do not use the `if(){ /* … */ }` constructs because squiggly braces are harder to discern amongst HTML.

HELPFUL LINKS:

- https://stackoverflow.com/questions/564130/difference-between-if-and-if-endif

Which common methods are available on the $block variable?

- `getRootDirectory()`
- `getMediaDirectory()`
- `getUrl()`
- `getBaseUrl()`
- `getChildBlock($alias)`
- `getChildHtml($alias, $useCache = true)`

- `getChildChildHtml($alias, $childChildAlias = '', $useCache = true)`: this method returns the HTML from a grandchild block.
- `getChildData($alias, $key = '')`: calls `getData` on a child block.
- `formatDate($date = null, $format = \IntlDateFormatter::SHORT, $showTime = false, $timezone = null)`
- `formatTime($time = null, $format = \IntlDateFormatter::SHORT, $showDate = false)`
- `getModuleName()`
- `escapeHtml($data, $allowedTags = null)`
- `escapeJs($string)`
- `escapeHtmlAttr($string, $escapeSingleQuote = true)`
- `escapeCss($string)`
- `stripTags($data, $allowableTags = null, $allowHtmlEntries = false)`
- `escapeUrl($string)`
- `getVar($name, $module = null):` locates a value from the theme's `etc/view.xml`

How can a child block be rendered?

```
$block->getChildHtml('child-name');
```

How can all child blocks be rendered?

```
$block->getChildHtml();
```

How can a group of child blocks be rendered?

Groups are a little-known part of the Magento layout system. These methods are found in `vendor/magento/framework/View/LayoutInterface.php`. The primary example of groups in the Magento core is on the product detail page's tabs and can be seen rendered at this link. Rendering them involves obtaining all of their names using `getGroupChildNames` and then rendering each block by name in a loop.

4.4 RENDER VALUES OF ARGUMENTS SET VIA LAYOUT XML

How can values set on a block in layout XML be accessed and rendered in a template?

In this example customization:

```
<referenceBlock name="product.info.sku">

    <arguments>

        <argument name="test_value"
xsi:type="string">11111</argument>

    </arguments>
```

```
</referenceBlock>
```

The value for `test_value` will be available in the template like: `$block->getData('test_value')` or `$block->getTestValue()`.

HELPFUL LINKS:

- https://alanstorm.com/magento_2_javascript_css_layout_woes/

4.5 DEMONSTRATE ABILITY TO ESCAPE CONTENT RENDERED AND TEMPLATE FILES

How can dynamic values be rendered securely in HTML, HTML attributes, JavaScript, and in URLs?

This is possible by utilizing the block's built-in methods.

- HTML: `$block->escapeHtml('value', $allowedTags);`
- HTML attributes: `$block->escapeHtmlAttr('value', $escapeSingleQuote);`
- JavaScript: `$block->escapeJs('value');`
- URLs: `$block->escapeUrl($url);`

Magento 2 Certified Professional
Frontend Exam Preparation eBook

5. STATIC ASSET DEPLOYMENT

SWIFT OTTER SOLUTIONS

(3 QUESTIONS - 5%)

5.1 DESCRIBE THE STATIC ASSET DEPLOYMENT PROCESS FOR DIFFERENT FILE TYPES

What commands must be executed to deploy static file types?

- `php bin/magento setup:static-content:deploy`
- `php bin/magento dev:source-theme:deploy` (on developer machine)

HELPFUL LINKS:

- https://devdocs.magento.com/guides/v2.1/config-guide/cli/config-cli-subcommands-static-view.html
- https://github.com/SwiftOtter/MagentoCI/blob/master/build/20-process/20-m2-compile.sh

What are common mistakes during the process?

- Not using a build tool (such as MagentoCI, by SwiftOtter; Capistrano, by David Alger). If no build tool is used, your site will likely be down for the duration of the compile which could be a minute or more. Additionally, if anything breaks, you will have to troubleshoot those problems on a live site.
- Not using the static file signing for cache busting. Users would see old content at this point.
- Not minifying the files.
- Running `bin/magento setup:static-content:deploy --force` in developer mode.
- Forgetting to deploy assets for some languages/themes.

5.2 DESCRIBE THE EFFECT OF DEPLOY MODES ON FRONTEND DEVELOPMENT

What are the differences between development and production mode in regard to frontend development?

According to DevDocs, `production` mode will not generate any missing static files whereas `development` and `default` will attempt to symlink any necessary assets into the corresponding folder in `pub/static`.

In developer mode, errors are rendered in the browser, while in production and default they are not shown and only written to a report file. This makes the feedback loop quicker in development mode.

5.3 DEMONSTRATE YOUR UNDERSTANDING OF LESS > CSS DEPLOYMENT AND ITS RESTRICTIONS IN DEVELOPMENT

Which LESS compilation options are available in Magento? How are they different?

Server-side and client-side are the two available options. This is configured in Store > Configuration > Advanced > Developer > Frontend Development Workflow.

Server-side: LESS files are compiled with a PHP LESS library. In developer mode, PHP will generate the CSS files on the fly provided there is not one already. Running `php bin/magento setup:static-content:deploy` will also compile the stylesheets.

Client-side: LESS files are compiled every page load on the client-side. This results in exceptionally slow response times and horrible flash-of-unstyled-text.

How do they influence the developer workflow during theming?

Client-side (Javascript based) compilation is hardly a consideration as developers should be more efficient than that. Magento provides a Grunt toolkit which will watch the source files and re-compile the output ones when changes are made. More information can be found on DevDocs. Further, we have had good success using Gulp from the Snowdog Frontools project and their Magento SASS port. SASS (and Gulp) compiles significantly faster than LESS (and Grunt) and Frontools provides additional features such as BabelJS support.

Magento 2 Certified Professional
Frontend Exam Preparation eBook

6. CUSTOMIZE AND CREATE JAVASCRIPT

SWIFT OTTER SOLUTIONS

(10 QUESTIONS - 17%)

6.1 INCLUDE CUSTOM JAVASCRIPT ON PAGES

What options exist to include custom JavaScript on a page?

Most methods involve including some Javascript or configuration directly in the page. It is then handled with Magento's special framework or RequireJS. This has the benefit of being able to apply it only to specific blocks or areas of the site. A unique option involves a RequireJS config file that is loaded from modules.

You can also include JS files into the `<head/>` tag of the rendered page. This is helpful to include external libraries.

Using the standard `<script type="text/javascript"></script>` tag. This is not recommended as Magento has no control over when this is executed. You do not have the ability to leverage any existing JS classes, and it can block rendering of the page. In some cases this is the right solution—for example if an external bundling solution is used.

For other code, use the `deferred="true"` or at least the `async="true"` attributes to the layout XML `<script>` tag, so the user experience is impacted as little as possible.

Using the `data-mage-init='{ "SwiftOtter_Module/js/modal": {"configuration-value":true} }'` attribute. This special Magento attribute requests and instantiates a Javascript module that takes two parameters: the element which hosts this attribute and the associated configuration (seen above in the ellipses).

The module would look like:

```
// app/code/SwiftOtter/Module/view/frontend/web/js/modal.
js

define([], function() {

  return function(element, config) {

    /* config: {configuration-value: true} */

  };

});
```

Using the Magento script tag:

```
<script type="script/x-magento-init">

  {

    ".element-selector": {

      "SwiftOtter_Module/js/modal": {

        "configuration-value": true

      }

    }
```

```
    }

</script>
```

In the above `.element-selector`, if multiple elements are found matching that selector, a new version of the module will be instantiated for each one (the module will still only be downloaded once). The selector can also be an asterisk (`*`). Contrary to the common use for the asterisk character in CSS selectors, here, it means that no elements are matched and `false` is passed to the script as the node argument.

Imperative notation:

While this is not usually the best way, it can be the easiest way to execute Javascript on a page that depends on other libraries (i.e. jQuery) or modules:

```
<script type="text/javascript">

require([

  "SwiftOtter_Module/js/modal"

], function(loader) {

  /* … */

});

</script>
```

This is not recommended by Magento.

RequireJS Config:

You can include Javascript on every page in an area using the `requirejs-config.js` file with a declaration like as shown below. As a side benefit, it usually runs sooner than the special Magento attribute or script tag because there is less Javascript that has to initialize before your module is loaded.

```
var config = {

  deps: ['SwiftOtter_Module/js/modal']

};
```

HELPFUL LINKS:

- https://devdocs.magento.com/guides/v2.2/frontend-dev-guide/layouts/xml-manage.html#layout_markup_css
- https://devdocs.magento.com/guides/v2.2/javascript-dev-guide/javascript/js_init.html
- https://swiftotter.com/technical/easily-add-custom-javascript-to-magento-2-knockout-frontend
- https://alanstorm.com/magento_2_javascript_init_scripts/
- https://www.mage2.tv/content/javascript/requirejs-fundamentals/

What are the advantages and disadvantages of inline JavaScript?

The advantage is that inline JavaScript is easy to include, and it does not send an additional request. The downside, though, is that the code becomes difficult to reuse. Additionally, depending on what type of Javascript is rendered, this code is not deferred and can block page loading.

If other places in the application need the same functionality, the script must be included there as well. This inhibits caching because the JavaScript cannot be cached separately from the main document and must be sent again every time. Also, as the complexity of the logic increases, it begins to be difficult to manage. At that point (or if you begin development knowing that it will soon get to this point), splitting your JavaScript into multiple files will ease development.

Additionally, inline JavaScript cannot be controlled with a Content-Security-Policy. This means that inline JavaScript is more likely to be exploited through Cross-Site-Scripting (XSS).

HELPFUL LINKS:

- https://developer.chrome.com/extensions/contentSecurityPolicy

How can JavaScript be loaded asynchronously on a page?

This occurs through placing functionality into `require` or `define` methods. What is the difference between these two methods? `require` executes immediately. When RequireJS is loading modules, and it comes across one that has this method, the contents are executed. `define` wraps a module that can be requested and used by other modules. As a result, it is only executed when called, such as

```
require(['SwiftOtter_Module/js/loader'], function(loader) {
/* … */ });
```

In the `<script/>` tag in layout XML, you can also apply common attributes that will be rendered on the `<script/>` tag in HTML, such as `defer` and `async`.

How can JavaScript on a page be configured using block arguments in layout XML?

The method `getJsLayout` exists in AbstractBlock. This returns the value of the `jsLayout` array as set on the block. On the frontend, `<?= $block->getJsLayout();?>` returns a JSON string, which was built from the `JsLayout` array (see example).

Using a hierarchy of `<arguments/>` in a block, we can assemble a JSON object for the frontend.

See this in action on the cart page:

- `checkout_cart_index.xml`
- `\Magento\Checkout\Block\Cart\Shipping`
- `shipping.phtml`

HELPFUL LINKS:

- https://www.mage2.tv/content/javascript/requirejs-fundamentals/loading-javascript-in-a-page-with-layout-xml/

How can it be done directly in a .phtml template?

This is completed by following the examples regarding including Javascript as described above.

6.2 DEMONSTRATE UNDERSTANDING OF USING JQUERY

Demonstrate understanding of jQuery and jQuery UI widgets.

jQuery is a library to assist with JavaScript development while smoothing over browser inconsistencies. It had its place several years ago while Magento 2 was in development. Here is an interesting discussion about the future and place of jQuery.

Magento uses jQuery v1.12: details and more details.

jQuery is famous for the `$('.element-selector').methodName(/* … */)` syntax. This snippet locates the HTML elements that match the selector and calls the fictitious `methodName` function on each. There are many other methods that can be called on the selector to modify its appearance or how it functions. jQuery is slightly less verbose than vanilla JavaScript, although this usually comes at a cost of browser performance.

jQuery UI widgets are an extension of jQuery that provides pre-built components for displaying common functionality. Here are a list of core widgets.

HELPFUL LINKS:

- https://learn.jquery.com/jquery-ui/widget-factory/how-to-use-the-widget-factory/

Demonstrate understanding of how to use Magento core jQuery widgets.

Magento includes a number of jQuery widgets that are available to customize the look of the frontend. The goal is to save development time by offering pre-built solutions to solve business requirements.

These are listed below in further reading.

DevDocs has good examples for how to utilize these widgets:

- https://devdocs.magento.com/guides/v2.2/javascript-dev-guide/widgets/widget_confirm.html
- https://devdocs.magento.com/guides/v2.2/javascript-dev-guide/widgets/widget_accordion.html

HELPFUL LINKS:

- https://devdocs.magento.com/guides/v2.2/javascript-dev-guide/widgets/jquery-widgets-about.html
- https://jqueryui.com/widget/

How can jQuery UI Widget methods be instantiated?

- Create a `require` method or a JS module with the `define` method.
- Specify the path of the module. For the confirmation widget, it would look like:

```
require(['Magento_Ui/js/modal/confirm'],
function(confirmWidget) { /* … */ });
```

- Utilize the widget by calling the function that was imported:
  - If the widget is independent of an HTML element: `confirmWidget(/* … */)`
  - If the widget affects an HTML element, you need to use the widget name as it was originally defined: `$('.element-selector').accordion({ /* … */ });`

How does variable naming / scope work with RequireJS and Magento widgets?

When a widget is defined, it will look something like this:

```
define([
  "jquery"
], function($) {
  $.widget("swiftotter.mywidget", {});
  return $.swiftotter.mywidget;
});
```

In the above code, we are exporting the newly-created widget from a RequireJS module. Here is where an important distinction is made. **When creating the widget, you assign its name to the jQuery object.** As such, that widget is callable from the jQuery selector.

However, the widget is **also** exported from the module. If you do not need to use the jQuery selector, you can import the widget and just use the name as imported.

```
define([

  "jquery",

  "SwiftOtter_Flex/js/mywidget"

], function($, importedMyWidget) {

   $(".element-selector").mywidget(/* … */);

   /**

    * The above works as it is referencing the same name
as specified when it was created.

    * This is because this widget needs to be attached to
HTML element(s).

    */
```

```
    importedMyWidget(/* … */);

    /**

     * Calling the variable importedMyWidget works because
  that is what is defined in this scope.

     */

  }
```

How can you call jQuery UI Widget methods?

This answer is for widgets who are attached to an element.

```
  $('.element-selector').accordion("activate");
```

The syntax is to execute the widget's name and pass, as a method argument, the name of the function you want to execute.

HELPFUL LINKS (AN EXAMPLE):

- https://devdocs.magento.com/guides/v2.2/javascript-dev-guide/widgets/widget_accordion.html

How can you add new methods to a jQuery UI Widget? How can a jQuery UI Widget method be wrapped with custom logic?

There are two ways to implement this. If the business requirements dictate modifying something already in use (like the product page tabs), a different approach must be used than if you are creating something new (and have the capacity to specify a new component).

We will address modifying an existing piece of functionality first. This is done by creating a mixin—a frontend parallel to interceptors (plugins).

To override an existing Magento jQuery UI widget:

```
//app/code/SwiftOtter/Test/view/frontend/requirejs-config.
js

var config = {

    "config": {

        "mixins": {

            "mage/tabs": {

                'SwiftOtter_Test/js/tabs-mixin': true

            }
```

```
        }

    }

};
```

```
// app/code/SwiftOtter/Test/view/frontend/web/js/tabs-
mixin.js

define(['jquery'], function(jQuery) {

    return function(original) {

        jQuery.widget(

            'mage.tabs',

            jQuery['mage']['tabs'],

            {

                activate: function() {

                    // your custom functionality here.

                    return this._super();

                }

            }
```

```
            );

            return jQuery['mage']['tabs'];

        }

    });
```

Many thanks to Alan Storm's article on this subject which makes it very clear on how to do this.

Here is an example on how to add a new method to a Magento jQuery UI widget by creating a new widget that extends another widget:

```
    // app/code/SwiftOtter/Test/view/frontend/web/js/custom-tabs.js

    define([

      'jquery',

      'jquery/ui',

      'mage/tabs'

    ], function($) {

        $.widget('swiftotter.customTabs', $.mage.tabs, {

            doSomething: function(input) {
```

```
            alert("Nothing like a good ol' fashioned
ALERT in your face. Input value: " + input);


        }

    });

});
```

Then, in your PHTML template:

```
// app/code/SwiftOtter/Test/view/frontend/templates/test.
phtml

<div class="element-selector"></div>

<script>

require([

  'jquery',

  'SwiftOtter_Test/js/custom-tabs'], function ($) {

    $(".element-selector").customTabs();

    $(".element-selector").customTabs("doSomething",
"hello");
```

```
    // this method demonstrates how to call a function on
a widget


});


</script>
```

HELPFUL LINKS:

- https://alanstorm.com/modifying-a-jquery-widget-in-magento-2/
- https://devdocs.magento.com/guides/v2.2/javascript-dev-guide/javascript/custom_js.html
- https://www.mage2.tv/content/javascript/jquery-ui-widgets/customizing-javascript-jquery-ui-widgets-with-requirejs-mixins/

6.3 DEMONSTRATE UNDERSTANDING OF REQUIREJS

The idea behind requireJS is fundamental to building a JavaScript application of any size. Instead of thinking of JavaScript as a “scripting” language, it has progressed to the point of being capable of solving business requirements on the frontend (and, with NodeJS, on the backend, too).

RequireJS gives the capacity to split JS functionality into separate files or **modules**. This makes each file easy to read and easier to test. JS is capable of being an Object-Oriented Language, although, similar to PHP, JS can be written either way.

How do you load a file with require.js?

For the most part, RequireJS follows a similar principle to Magento 2 backend development: a module never instantiates itself, and it does not directly instantiate another class. The dependencies that a module requests are provided to the requesting module.

There are already many examples above, but we will look at the specifics of directory paths and loading modules.

To instantiate a new application, do so in a rendered PHTML template. The advantage of this is that you can easily provide configuration details or translated string to the constructor.

```
<script type="text/x-magento-init">

{

  "*": {

    // this path maps to app/code/SwiftOtter/Test/view/
frontend/web/js/test.js

    "SwiftOtter_Test/js/test": {

      "details": "test"

    }
```

```
    }

}

</script>
```

POINTS TO REMEMBER

- Do not add the `.js` to the module file path. The extension is assumed.
- Do add `.babel` to the module file path if you are utilizing Frontool's Babel.

Note that you can find these module names in `requirejs-config.js` (example).

HELPFUL LINKS:

- https://alanstorm.com/magento_2_javascript_init_scripts/
- https://www.mage2.tv/content/javascript/requirejs-fundamentals/loading-javascript-in-a-phtml-template-with-requirejs-via-x-magento-init/

How do you define a require.js module?

```
define([], function() {

    return function(config, el) {
```

```
            console.log({el: el, config: config});

        }

    });
```

HELPFUL LINKS:

- https://www.mage2.tv/content/javascript/requirejs-fundamentals/defining-a-javascript-requirejs-amd-module/
- https://www.mage2.tv/content/javascript/requirejs-fundamentals/passing-arguments-from-php-to-simple-requirejs-javascript-modules/

How are require.js module dependencies specified?

They are an array specified in the first parameter of the `define` or `require` methods.

How are module aliases configured in requirejs-config.js?

An alias provides flexibility in directing requireJS to the correct (or new) URL for a module. This is set up in your module's `view/[area]/requirejs-config.js`.

Aliases are configured in the `map` element in `requirejs-config.js`.

HELPFUL LINKS:

- https://devdocs.magento.com/guides/v2.2/javascript-dev-guide/javascript/custom_js.html#js_replace
- https://www.mage2.tv/content/javascript/requirejs-fundamentals/aliasing-requirejs-module-files-with-requirejs-config/

How do you regenerate the compiled requirejs-config.js file after changes?

In developer mode, `requirejs-config.js` is regenerated on every page load. In default or production, `requirejs-config.js` is generated if it doesn't already exist.

See `vendor/magento/module-require-js/Model/FileManager.php ensureSourceFile($relPath)`.

HELPFUL LINKS:

- vendor/magento/framework/RequireJs/Config.php

How do you debug which file a requireJS alias refers to?

- Start by locating where the alias originates. This is found by a simple file search. Scope can be limited to `requirejs-config.js` files if desired, or search for the alias name in the merged requirejs-config.js file in the theme directory within pub/static/<area>. . .

- Determine if there are any overrides for the alias.
- Map these files to the directory path as described above.

Demonstrate that you understand how to create and configure Magento JavaScript mixins.

Let's look at an example of how to create a mixin and override an existing module's functionality.

We need to update functionality in: https://magento.test/static/version1/frontend/SwiftOtter/Flex/en_US/Magento_Catalog/js/gallery.js

This file is `pub/static/frontend/SwiftOtter/Flex/en_US/Magento_Catalog/js/gallery.js`

Its source file is vendor/magento/module-catalog/view/frontend/web/js/gallery.js

Add the following to your `requirejs-config.js` file:

```
// app/code/SwiftOtter/Flex/view/requirejs-config.js

var config = {

    "config": {

        "mixins": {

            "Magento_Catalog/js/price-utils": {
```

```
                "SwiftOtter_Test/js/price-utils-
override": true

            }

        }

    }

};
```

Then, create this file:

```
// app/code/SwiftOtter/Flex/view/frontend/js/price-utils-
override.js

define([], function() {

  var updates = {

    formatPrice: function(amount, format, isShowSign) {

      return '00000';

    }

  };
```

```
    return function(target) {

      return Object.assign(target, updates);

    }

});
```

HELPFUL LINKS:

- https://devdocs.magento.com/guides/v2.2/javascript-dev-guide/javascript/js_mixins.html
- https://inchoo.net/magento-2/custom-javascript-in-magento-2-with-requirejs/
- https://www.mage2.tv/content/javascript/jquery-ui-widgets/customizing-javascript-jquery-ui-widgets-with-requirejs-mixins/
- https://www.mage2.tv/content/javascript/requirejs-fundamentals/customizing-javascript-objects-with-requirejs-mixins/
- https://www.mage2.tv/content/javascript/frontend-ui-components/customizing-javascript-uicomponents-with-requirejs-mixins/

6.4 CONFIGURE JAVASCRIPT MERGING AND MINIFY IN THE ADMIN UI

What options are available to configure JavaScript minification and bundling?

The following settings are found in Store > Configuration > Advanced > Developer.

- **Merge JavaScript files:** concatenates source Javascript files from an area together into one file to download. The idea is that this boosts performance by reducing the total number of Javascript requests. This means a massive JS file is downloaded which can pose problems in its own way. HTTP/2 can be a better way to improve performance (or use Webpack).
- **Bundle JavaScript files:** groups the JS files into bundles. This is a similar idea to merging but supposed to be more flexible and downloads several files. The Inchoo article below has some interesting performance statistics and turning this on could negatively impact performance.
- **Minify JavaScript files:** reduces the JS file's size by doing things like stripping whitespace and shortening variable names.

The most important is to ensure that the server is properly configured with HTTP/2 and Gzip compression enabled.

HELPFUL LINKS:

- https://inchoo.net/magento-2/javascript-bundling-magento-2/

How does Magento minify JavaScript?

Magento's minification of JS is a two part system. The first part is to configure RequireJS to add a `.min` suffix to files. The second part is to minify the files.

Magento tells requireJS to download minified files through the minified resolver: `vendor/magento/module-require-js/Model/FileManager.php`, `ensureMinResolverFile()` method.

Then, Magento will minify the files as necessary. `vendor/magento/framework/App/StaticResource.php` is responsible for minifying each file.

This is where the hand-off to the JShrink library happens: `vendor/magento/framework/Code/Minifier/Adapter/Js/JShrink.php`

The minified files are saved with a `.min` suffix. As a result, all the file names are different in production. This poses a potential problem during deployment if those assets are built outside of the primary Magento database, or if the setting is changed at some point after the assets are built. If the static content is deployed with a different setting, none of the Javascript on the site will work.

What is the purpose of JavaScript bundling and minification?

To reduce download time and make the frontend more useful and faster.

6.5 UI COMPONENT CONFIGURATION

UI components are a complex and large framework. Unfortunately, their steep learning curve has given them a bad reputation in the community. While this section

does not represent an exhaustive study, hopefully this will shed some light and point you in the right direction for learning these.

Since this is a complex topic, and we can't cover everything, we suggest you to check out additional resources that can help guide you for further study:

- Mage2.tv (covering frontend UI components)
- Magento DevDocs UI Component Developer Guide

UI Components are available in the frontend and the backend. The way Magento uses these UI Components differs by the area. While it is generally better to follow precedent, both ways work in either place. The core UI component Javascript module is: `Magento_Ui/js/core/app`

In the frontend area, UI Components are configured through layout XML (`vendor/magento/module-checkout/view/frontend/layout/checkout_index_index.xml`). The `jsLayout` argument is used to specify information.

In the adminhtml area, UI Components are configured through a dedicated XML file. These files are placed in `app/code/SwiftOtter/Test/view/adminhtml/ui_component/[ui_component_name].xml`. They are then included in layout XML with the `uiComponent` tag:

```
<?xml version="1.0"?>

<page xmlns:xsi="http://www.
w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:View/
Layout/etc/page_configuration.xsd">

    <body>
```

```
        <referenceContainer name="content">

            <uiComponent name="cms_block_listing"/>

        </referenceContainer>

    </body>

</page>
```

(example from)

HELPFUL LINKS:

- https://devdocs.magento.com/guides/v2.2/ui_comp_guide/bk-ui_comps.html
- https://devdocs.magento.com/guides/v2.2/ui_comp_guide/concepts/ui_comp_config_flow_concept.html
- https://www.mage2.tv/content/javascript/frontend-ui-components/

How can you specify configuration options on a UiComponent widget in JSON and in Layout XML?

UI Components are initialized with JSON. This is seen when viewing the source of a Magento page that utilizes UI Components.

To use Layout XML:

```
// app/code/SwiftOtter_Test/web/js/

<!-- … -->

<arguments>

  <argument name=”jsLayout” xsi:type=”array”>

    <!-- -->

  </argument>

</arguments>
```

This layout XML is merged together, converted into JSON, and is available on the frontend.

A common system on the frontend is to use layout processors. In the checkout one page block, in the `getJsLayout` method, you will see that these layout processors have the opportunity to modify the UI component details before they are rendered. This is especially helpful for the checkout as customer details are likely to change should they refresh the page after filling in their information. In the case of the checkout layout processor, you can find their contract here.

HELPFUL LINKS:

- https://www.mage2.tv/content/javascript/frontend-ui-components/configuring-uicomponents-with-layout-xml/
- https://alanstorm.com/magento-2-understanding-ui-component-regions/

What configuration options are available on UiComponents?

Any values that are defined in the `defaults` property of the UI Component's Javascript module can be overridden with configuration loaded onto the page. When looking to update a UI Component, a good place to start is with the `defaults` property and checking if there is an item for the applicable value that should be changed. The module's parents likely also have `defaults` objects and all of those values can be overridden as well. When the modules are initialized, all of the `defaults` from a module and its ancestors are merged and then configuration from the page is applied over that. Magento DevDocs provides a list of common admin UI components and their available configuration options.

HELPFUL LINKS:

- https://devdocs.magento.com/guides/v2.2/ui_comp_guide/concepts/ui_comp_config_flow_concept.html
- https://devdocs.magento.com/guides/v2.2/ui_comp_guide/bk-ui_comps.html

How do you specify the ko (Knockout) template for a UiComponent?

For most UiComponents, you set the `template` option.

The template is specified as the module name, a slash, followed by the file path within the module's `view/<area>/web/template` directory, but without the .html file name suffix.

For example, the template `view/frontend/web/template/authentication-popup.html` in the module `Magento_Customer` is specified as `Magento_Customer/authentication-popup`.

Demonstrate an understanding of default tracks.

Tracks are specified in the UI Component's JS files. These convert the specified properties on the UI Component into a knockout-es5 observable.

HELPFUL LINKS:

- Implementation: `initObservable`
- Usage: `tracks` object.

6.6 UNDERSTANDING KNOCKOUT FRAMEWORK

KnockoutJS is the framework that powers much of Magento's frontend. It is structured around the idea of reactive data: instead of the developer tracking changes, and subscribing to events, data flows through the system and updates the pertinent bindings as it arrives. It has been built on top of and abstracted quite extensively in Magento.

HELPFUL LINKS:

- https://alanstorm.com/knockoutjs_primer_for_magento_developers/
- https://inchoo.net/magento-2/knockout-js-in-magento-2/

How do you use knockout.js bindings?

Bindings are a way to link reactive data with HTML. They are connected with the `data-bind` attribute like:

```
<input type="text" data-bind="value: firstName "/>
```

Assuming `firstName` is an observable in the Knockout model, the value of `this.firstName` will be automatically updated whenever the input's value changes.

There are many bindings available; here are some of the most common:

- `text`: sets the `innerText` of an element.
- `visible`: controls whether or not an element is visible.
- `if`: similar to `visible` but removes or adds the child nodes based on the evaluation of the property.
- `click`: calls the specified function when the element is clicked. It's a one-way binding (HTML to Javascript) which differs slightly from other bindings that are bi-directional.
- `value` / `checked` / `options`

Magento also provides some custom bindings. All of them can be set as regular bindings in a `data-bind=""` attribute, but some also may be specified as virtual knockout elements, custom attributes or custom elements.

All of the following are more or less equivalent if used within a knockout .html template:

```
<span data-bind="text: '1. Text output'"></span>
```

```
<!-- ko text: '2. Text output' --><!-- /ko →

<span text="'3. Text output'"></span>

<text args="'4. Text output'"></text>
```

Note that not all bindings are supported in all variations, and for some, the binding name is different (for example `data-bind="i18n: '…'"` and `translate="'…'"`).

HELPFUL LINKS:

- https://devdocs.magento.com/guides/v2.1/ui_comp_guide/concepts/knockout-bindings.html
- http://knockoutjs.com/documentation/visible-binding.html

How do you bind a ko view model to a section of the DOM with the scope binding?

The `scope` binding connects a UI component that is registered in the `uiRegistry` with an element that has already been configured. This changes that element's Knockout binding scope to use the class that represents the specified UI component.

```
<div data-bind="scope: 'estimation'">

    <p data-bind="text: description"></p>
```

```
</div>
```

In this example, the `div` tag will be bound directly to the UI component that is registered in the uiRegistry with the identifier `estimation`.

Due to the complexity of this, I highly recommend doing some experimenting and watching Mage2.tv's excellent series on Knockout and data binding.

HELPFUL LINKS:

- https://www.mage2.tv/content/javascript/
- https://devdocs.magento.com/guides/v2.1/ui_comp_guide/concepts/knockout-bindings.html

How do you render a ko template of a UiComponent?

The UI Components on the checkout page provide a good example regarding how to specify the template used to be rendered.

`vendor/magento/module-checkout/view/frontend/web/js/view/payment.js`

Here is a basic UiComponent and template:

```
// app/code/SwiftOtter/Flex/view/web/js/example-component.js

define([
```

```
    'uiComponent'

], function (

    Component

) {

    return Component.extend({

        defaults: {

            template: SwiftOtter_Flex/example-component',

            sampleText: "Here is some sample text."

        }

    });

});
```

```
// app/code/SwiftOtter/Flex/view/web/template/example-component

<div class="sample-component">
```

```
    <p data-bind="text: sampleText"></p>

</div>
```

HELPFUL LINKS:

- https://magento.stackexchange.com/a/89821/13

Demonstrate an understanding of the different types of knockout observables.

Observables allow a cascading effect of functionality to happen when data changes. Typically, when we build interfaces, we receive an `input`'s `change` event and then process the data. While that works, observables remove us having to write the code to link the data change to the events that transpire. As such, they are very powerful.

The basic observable is created with a call to `this.price = ko.observable(12.42)`. To get an observables value, it needs to be called as a function without arguments. (e.g. `this.price()`).

To update the value, it needs to be called as a function with an argument (e.g. `this.price(12.99)`.

Any update to the value triggers any dom elements that knockout rendered with that observable to be re-rendered.

This works well for any data type except arrays. If the observable contains an array or an object, a regular knockout observable does not trigger updates when values within the array change.

For this reason `ko.observableArray([])` should be used when working with arrays.

There also are observable functions, which are called “computed observables” and are created with a call to `ko.computed(callback)` or `ko.pureComputed(callback)`.

Magento also includes the knockout plugkn ko-es5. The benefit of ko-es5 is that it allows the creation of observables that can be read and set like regular properties, while knockout still tracks the changes under the hood.

You can use the observable’s `subscribe` method to listen to changes. The real value of the observable comes out when you bind that to an element in the template like:

```
<div data-bind="text: price"></div>
```

What common ko bindings are used?

See above section for a list of common ko bindings.

Demonstrate an understanding of ko virtual elements.

Virtual elements are a way to bind a child element without having to insert a node into the DOM just to specify the binding.

```
<ul>

  <li>Please choose</li>

  <!-- ko foreach: payments -->

  <li data-bind="text: $data.name"></li>

  <!-- /ko -->

</ul>
```

Supplying the default first list item can not be easily done when the `foreach` binding is specified on the `ul` element:

```
<ul data-bind="foreach: payments">

  <li data-bind="text: $data.name"></li>

</ul>
```

HELPFUL LINKS:

- http://knockoutjs.com/documentation/custom-bindings-for-virtual-elements.html

6.7 UNDERSTANDING DEPENDENCY BETWEEN COMPONENTS

Demonstrate an understanding of the links, imports, exports, and listens UiComponent configuration directives.

Links, imports, exports, and listens are how UI components share data between themselves. Having an understanding of each of these pieces will simplify your development responsibilities.

Some of the examples in this section have been adapted from Mage2.tv's discourse on the matter. These videos have greatly augmented my knowledge in this area.

HELPFUL LINKS:

- https://devdocs.magento.com/guides/v2.2/ui_comp_guide/concepts/ui_comp_linking_concept.html

`tracks:`

In order for a property to be updateable by the following methods, you must add this property to the `defaults.tracks` object. This converts each property specified in the object or array into an observable property.

`imports:`

`imports` is an object that resides in a UiComponent's `defaults` object. It retrieves data from another UI Component and assigns that to a property in the

requesting object. Whenever that data is updated, it flows through the properties as specified in the `imports` node.

```
define([
    'uiComponent'
], function(Component) {
    return Component.extend({
        defaults: {
            price: 11,
            tracks: {
                price: true
            },
            imports: {
                price: '${$.provider}:price'

                // we are linking the price variable on this with the price observable in $.provider.
            }
        }
    })
});
```

HELPFUL LINKS:

- https://www.mage2.tv/content/javascript/frontend-ui-components/javascript-uicomponent-imports/

`exports`

Exports sends an observable value to another module. It is the opposite of `import`.

`links`

This is a two-way binding system. If the value is updated on either component (the host or the linked component), both components' value will likewise be updated.

HELPFUL LINKS:

- https://www.mage2.tv/content/javascript/frontend-ui-components/javascript-uicomponent-links/

6.8 UNDERSTANDING STRING TEMPLATES

Demonstrate an understanding of ES5 string literal templates like ${$.provider}. What does $. inside of ${} resolve to?

`${}` evaluates the expression inside the curly braces. In ES5, this would typically be within backticks. However, because Magento has a system in place to process them

differently if browsers do not natively support template literals, quotes are used instead. If the browser does support template literals, they are processed with backticks.

Inside the `${}`, using `$`. resolves to `this`. As such, `$.provider` resolves to `this.provider`.

HELPFUL LINKS:

- https://www.mage2.tv/content/javascript/frontend-ui-components/javascript-uicomponent-es5-pseudo-template-strings/
- https://devdocs.magento.com/guides/v2.2/ui_comp_guide/concepts/ui_comp_template_literals.html

Magento 2 Certified Professional
Frontend Exam Preparation eBook

7.
USE LESS/ CSS TO CUSTOMIZE THE MAGENTO LOOK AND FEEL

(5 QUESTIONS - 8%)

SWIFT OTTER SOLUTIONS

Magento utilizes the LESS preprocessor to simplify theming. Its goal is to keep styles more concise by providing variables, handling nested selectors, and allowing common functions (mixins). Browsers cannot interpret LESS so it must be compiled into CSS.

The most common LESS files, that are compiled into CSS are: `styles-m.css`, `styles-l.css`, `print.css`. These files are included in the default_head_blocks.xml file. According to a convention, all LESS files that are directly included and compiled to CSS do not start with an underscore.

`styles-m.css`: mobile-friendly styles. In trying to reduce the download size for websites, `styles-m.css` contains any styles necessary for viewing on mobile.

`styles-l.css:` additions for desktop styles. This contains the additional styles for displaying on a screen that is 768px or wider.

`print.css`: additions for print styles.

There are other files that do not begin with an underscore and are individually output: see this link.

To insert an external stylesheet into the `<head/>` tag, you can use the `<css/>`, or the `<link/>` tag in layout XML:

```
<page xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:View/
Layout/etc/page_configuration.xsd">

    <head>

        <css src="css/local-stylesheet.css"/>
```

```
        <css src="https://swiftotter.com/external.
stylesheet.css" src_type="url" />


        <link rel="stylesheet" type="text/css"
src="https://swiftotter.com/another.external.stylesheet.
css" src_type="url" />


    </head>


</page>
```

The `<link/>` can also be used to add JavaScript resources. To make sure your code is clear, we would recommend adding CSS resources using the `<css/>` element.

HELPFUL LINKS:

- https://devdocs.magento.com/guides/v2.2/frontend-dev-guide/layouts/xml-manage.html#layout_markup_css
- https://www.classyllama.com/blog/a-look-at-css-and-less-in-magento-2

7.1 EXPLAIN CORE CONCEPTS OF LESS

Describe features like file import via @import directive, reusable code sections via mixins together with parameters and the usage of variables. Demonstrate your understanding of the special variable @arguments.

@import directive:

While Magento contains hundreds of `.less` files, it compiles those files back into a number of `.css` files. LESS provides an `@import` construct to import a file. All files can import other files. Eventually an entire tree is built of imported files. A LESS convention is that all files that are included by the `@import` construct starts with an underscore in its filename.

The `styles-l.less` file contains this line:

```
@import '_styles.less';
```

This is including `_styles.less`, which includes other files.

HELPFUL LINKS:

- https://devdocs.magento.com/guides/v2.0/frontend-dev-guide/css-topics/css-preprocess.html#fedg_css-import
- http://lesscss.org/features/#import-atrules-feature

Mixins

This is the equivalent to a method or a function in another programming language. Mixins return CSS instructions. The goal is that you write it once in a mixin and then reuse that throughout the application's theme (DRY principle).

The way it works is you write some LESS instructions (which, in this case, are verbatim CSS):

```
.element-selector {
  background-color: #ff0000;
}
```

And then call it:

```
.another-element-selector {
  .element-selector();
}
```

This results in:

```
.another-element-selector {
  background-color: #ff0000;
}
```

Mixins can also contain parameters. Using our above fictitious example:

```
.element-selector(@color) {
  background-color: @color;
}
```

This is called:

```
.another-element-selector {

  .element-selector(#0000ff);

}
```

Mixin parameters can be supplied in a format similar to a Javascript object and many in Magento are setup this way. When this is the case, one can pass in a subset of arguments by including the parameter's name along with the argument value.

HELPFUL LINKS:

- http://lesscss.org/features/#mixins-feature

Variables:

Variables allow you to specify a value in one place and then use it in other locations (just like with programming). Magento makes heavy use of LESS variables (hopefully, someday we will see them simplified).

HELPFUL LINKS:

- http://lesscss.org/features/#variables-feature

`@arguments`:

`@arguments` takes the arguments that were passed into a mixin and renders them in that order.

HELPFUL LINKS:

- http://lesscss.org/features/#mixins-feature-the-arguments-variable

Demonstrate how to use the nesting code formatting, and the understanding of media queries together with nesting.

In writing vanilla CSS, you can have a maximum of three levels of nested code:

- Media query
- @supports wrapper
- Selector

LESS gives you the power to nest CSS as deep as you wish. This LESS is then compiled back into the above two levels of nested code.

Here is an example:

```
// app/code/SwiftOtter/Test/view/web/css/source/_module.less


.element-selector {

  color: #000000;
```

```
    .child-selector {

      background-color: #ff0000;

    }

  }
```

This is compiled into:

```
  .element-selector { color: #000000; }

  .element-selector .child-selector { background-color:
  #ff0000; }
```

Media queries also follow this idea.

```
  .element-selector {

      background-color: #ff0000;

      @media screen (min-width: 750px) {

        background-color: #0000dd;

      }

  }
```

This is compiled into:

```
.element-selector {
  background-color: #ff0000;
}
@media screen (min-width: 750px) {
  .element-selector {
    background-color: #0000dd;
  }
}
```

While nesting can appear very helpful, it can introduce tremendous bloat into styling. It seems to fix specificity problems but adds many extra bytes. The question that we ask is: “are the additional selectors necessary?” In many cases they are not. Small changes to deeply nested selectors can have significant impact on output. For instance, adding a comma and second class name to a selector five levels deep doubles the entire chain. Deep nesting also leads to hard to manage selectors that are more specific than required and difficult to override. Instead, using a quality naming convention, such as BEM will eliminate many levels of nesting and simplify code reuse.

Describe how the & (Ampersand) works and its function. Describe how calculations are possible as well.

The `&` is a concatenation character and is particularly useful with writing BEM styles.

Here is an example:

```
.element-selector {
  color: #000000;
  &__text {
    font-size: 1rem;
  }
  &__call-to-action {
    background-color: #ff0000;
  }
}
```

This compiles to:

```
.element-selector { color: #000000; }
.element-selector__text { font-size: 1rem; }
```

```
.element-selector__call-to-action { background-color:
#ff0000; }
```

Calculations are also possible:

```
.element-selector {

  border-width: 1px + 1px;

}
```

This compiles to:

```
.element-selector { border-width: 2px; }
```

It is generally better to limit its use, though, because it can make finding the initial declaration of the style more tedious because searching is essentially useless.

HELPFUL LINKS:

- http://lesscss.org/#operations

7.2 EXPLAIN MAGENTO'S IMPLEMENTATION OF LESS (@MAGENTO_DIRECTIVE)

Demonstrate the process from magento-less files via php preprocessing into real LESS files with extracted @import directives. Where can the intermediate files be found?

Magento's LESS classes are found in the `\Magento\Framework\Css` namespace.

When loading the page, Magento looks for any `css` insertions into the `<head/>` tag in layout XML. Any `.css` file references are changed to `.less` and a search is made for a LESS file with that name. From there, Magento parses the `@imports` and `//@magento_import` instructions and assembles file paths based on the fallback directories.

Magento's Luma and blank themes load in two CSS files: `styles-l.css` (desktop) and `styles-m.css` (mobile and greater). These are the entry points into the LESS file structure. These are also the output files (except that they will have a `.css` extension). All LESS files that have been imported by another LESS file are included into one of these two files.

The intermediate files are found in the `var/view_preprocessed` directory. From there they are symlinked into the pub/static directory.

HELPFUL LINKS:

- https://devdocs.magento.com/guides/v2.0/frontend-dev-guide/css-topics/css-preprocess.html#server-side

What do you have to remember, when you change a less file? Which files will be re-processed on file changes? Are the original files copied or symlinked in developer environments?

When you create a new LESS file, you first need to run `bin/magento dev:source-theme:deploy`. Then (or if you just make a change to a new LESS file) you need to delete the pub/static/[area] directory. Refresh your browser to regenerate files (and get something to drink while you wait). Better yet, use the Grunt compiler.

TIP: Or for even faster response, use Snowdog's SASS port and Frontools.

If you use client-side compilation, most changes are seen immediately when refreshing the browser.

The original files are symlinked (see here and here).

7.3 DESCRIBE THE PURPOSE OF _MODULE.LESS, _EXTEND.LESS, _EXTENDS.LESS

Demonstrate LESS has no fallback capabilities and therefore Magento created @magento_import directives to enable FE devs to inject or replace parts of existing less structures of modules and themes.

LESS does not have the ability to dynamically find and include partials. As a result, every path must be included individually in the primary stylesheet. To work around this limitation, Magento provides a `//@magento_import` directive. This will search for files through many locations and include each version that it finds. Note the use

of the `//` as a comment to prevent any errors with LESS. Using the `//@magento_import` directive allows the inclusion of a specific file from each module with just one line.

HELPFUL LINKS:

- https://github.com/magento/magento2/blob/2.2-develop/app/design/frontend/Magento/blank/web/css/styles-l.less

Here are some of the primary uses of `//@magento_import`:

`_module.less`: This is the main entry point of a module's stylesheet. For a module, it can have partials of its own, if desired, and should place them in a `/module` directory next to this file. For a theme, this would override the core module's primary stylesheet. As a result, this would be done in cases where a significant portion of the styles are being updated.

`_extend.less`: This should mostly or entirely be empty in modules. For themes, if most of the core module's stylesheets are good and the goal is only to update a few things, the easiest way to do that is to create an `_extend.less` file. This file is loaded after `_module.less` and consequently will apply given the same selector.

`_extends.less`: this file contains a massive list of abstract selectors that can be "extended" from and mixins.

How to override a LESS theme file:

We need to modify the availability notice on the product page. The merchant has requested that this text is very large and prominent on the page. You have

already created the SwiftOtter/Flex theme. How do you update the font size for this element?

Since this is a small change, the best might be to leverage `_extend.less` by creating one in `app/design/SwiftOtter/Flex/Magento_CatalogInventory/web/css/source/_extend.less`.

However, if this is the first of a litany of changes, override the LESS file.

To do this, copy `vendor/magento/theme-frontend-luma/Magento_Catalog/web/css/source/_module.less` into `app/design/SwiftOtter/Flow/Magento_Catalog/web/css/source/_module.less`. Modify the necessary code. Clear `var/view_preprocessed` and `pub/static` and refresh the page.

7.4 SHOW CONFIGURATION AND USAGE OF CSS MERGING AND MINIFICATION

Demonstrate the primary use case for merging and minification. Determine how these options can be found in the backend. Understand the implications merging has in respect to folder traversal.

The goal of merging is to reduce the number of HTTP requests. The goal of minification is to reduce the number of bytes being transferred. Minification is valuable as it strips out whitespace which adds extra weight to the download request. Both of these provide performance boosts.

To enable / disable these settings, go into Store > Configuration > Advanced > Developer > CSS Settings.

7.5 MAGENTO UI LIBRARY USAGE

Demonstrate your understanding of Magento's UI library, a LESS-based library of mixins and variables for many different standard design elements on websites. How can you take advantage of the UI library? What do you have to do to enable it in your theme?

Magento has built a library of mixins to attempt to simplify the task of theming common layout components. These are found in lib/web/css/source/lib.

If you have built a custom theme, these components will need to be imported with a path like:

```
@import (reference) '../../css/source/lib/lib';
```

As an example, let's add a new font onto our website design (don't take the font names seriously):

```
.lib-font-face("Trebuchet MS", "@{baseDir}fonts/trebuchet-
ms", 500);
```

Another example:

```
.footer-breadcrumbs {

  .breadcrumbs();

}
```

Caution: some of the mixins are extraordinarily large. Instead of including a large mixin to accommodate your interface, we recommend exploring the option of matching Magento's selector structure in order to leverage their styles. This prevents introducing a large amount of nearly duplicate code.

HELPFUL LINKS:

- https://devdocs.magento.com/guides/v2.2/frontend-dev-guide/css-topics/theme-ui-lib.html
- https://devdocs.magento.com/guides/v2.2/frontend-dev-guide/css-topics/using-fonts.html

Which file is primarily used for basic setup of variables? Where can UI library files be found?

New variables should be placed in local theme `lib/` or local theme files. Overrides of existing variables should be declared in this file: `<theme_dir>/web/css/source/_theme.less`.

The source UI library files are located in `lib/web/css/source/lib`.

Source: https://github.com/magento/magento2/blob/2.2-develop/app/design/frontend/Magento/blank/web/css/source/_theme.less

How can it be extended?

There are multiple ways to extend a lib file, you can extend it by copying the `.less` file into your theme. For example, if you need to customize `_dropdowns.less`:

```
# Mac/Linux copy [from] [to]

cp lib/web/css/source/lib/_dropdowns.less app/design/
frontend/SwiftOtter/Flex/css/source/lib
```

The caveat here is that the less you copy core files, the more upgradeable the application will be.

Another approach is to implement your own custom version of the file, containing only your overrides. You can do this by adding the following file:

```
app/design/frontend/SwiftOtter/Flex/css/source/custom/lib/_
dropdowns.less
```

This will not be imported by default as it is an additional file, so you will need to add the import yourself. You can copy the core import file (`web/css/_styles.less`) to your theme at the following location:

```
app/design/frontend/SwiftOtter/Flex/css/_styles.less
```

Then add the additional import. The best method to use will be dependent on the level of extension required. For example, in a situation where you want to override a single mixin within a lib file that contains multiple, it may be beneficial to just add your own custom file containing the single override. This method also helps to clearly identify where edits have been made.

HELPFUL LINKS:

- https://www.classyllama.com/blog/a-look-at-css-and-less-in-magento-2

How can you change specific parts of the UI library?

The mixins usually have default parameters that are variables declared in the ./variables folder. As a result, many things can be changed by setting your own values on those variable names in the `_theme.less` file. You can also override the files and make further changes. Finally, when including a Magento UI library mixin in your own code, you can specify custom values for the many parameters that are available. The breadcrumbs UI widget has almost sixty variables that can be customized.

Magento 2 Certified Professional
Frontend Exam Preparation eBook

8. CUSTOMIZE THE LOOK AND FEEL OF SPECIFIC MAGENTO PAGES

(13 QUESTIONS - 22%)

SWIFT OTTER SOLUTIONS

8.1 UTILIZE GENERIC PAGE ELEMENTS

Demonstrate an understanding of customizing generic page elements that can be found on most pages: page header and footer, quick search, store view (language) switcher, mini cart, breadcrumbs, and sidebar menu.

Experience is the best teacher when it comes to performing various customizations here. If you have made updates to these various items, setup a Magento 2 playground and implement sample changes. Further, we recommend that you review every file listed in these sections. This will help boost your overall awareness of the layout and structure.

Page header / footer

- Layout XML
- Header block
- Header links
- Logo
- Menu bar
- Customer details UI component
- Header styles
- Footer container
- Footer block
- Copyright

Quick search

- Mini-search form
- Search styles

Store view switcher

- Block
- Styles

Mini-cart

- Minicart form
- Styles
- Layout XML
- UI Component

Breadcrumbs

- Template
- Block
- UI Widget
- Styles

Sidebar menu

- `sidebar.main` is the name of the container that represents the sidebar.
- Styles

8.2 CUSTOMIZING PRODUCT DETAIL PAGES

How can design changes (page layout) be configured on product detail pages?

The Design tab on a product edit page provides the ability to affect how a product is rendered on the frontend.

- Theme: change the applied theme.
- Layout: choose from layouts as specified in `page_layouts.xml` files.
- Display Product Options in: determines where to place the product's options
- Layout Update XML: the capability to inject Layout XML updates into a product's page. Here you can create / remove blocks, change templates, etc.

One thing to reiterate is that layout XML changes are possible within the Layout Update XML field. Here is an example of adding a customer block:

```
<referenceContainer name="product.info.main">

    <block template="SwiftOtter_Test::test.phtml"/>

</referenceContainer>
```

How can design changes be configured for specific product types?

Using layout handles. When a product is rendered, multiple layout handles are assigned:

- `catalog_product_view`

- `catalog_product_view_type_[product type]:`
  - `catalog_product_view_type_simple`
  - `catalog_product_view_type_configurable`
  - `catalog_product_view_type_bundle`
- `catalog_product_view_id_123` (the product's ID)
- `catalog_product_view_sku_MJ01` (the product's SKU attribute)
- `catalog_product_prices`

How can you use custom layout updates for specific product pages?

You can use either the layout handle for the particular SKU or product ID or the Layout Update XML field in the admin panel for that product.

Demonstrate an understanding of how to use the container blocks provided by Magento to display additional information on product pages.

Magento provides many containers on the product page, making it much easier than Magento 1 to modify the output on a product page. All you need to do is reference the container and add blocks to it.

Here is a list of all the containers available (as of Magento 2.2, reference):

- `content`
- `product.info.main`

- `product.info.price`
- `product.info.stock.sku`
- `product.info.type`
- `alert.urls`
- `product.info.form.content`
- `product.info.extrahint`
- `product.info.social`
- `product.info.media`
- `skip_gallery_before.wrapper`
- `skip_gallery_after.wrapper`

8.3 CUSTOMIZING CATEGORY PAGES

How can design changes (page layout) be configured on category pages? How can the layered navigation be configured?

There are two sections within a category page that affect how the page displays: Design and Display Settings.

Design

Use Parent Category Settings: if checked, the rest of the settings in this tab are loaded from the parent category.

Theme / Layout: similar to product section above.

Apply Design to Products: if this is selected the theme, layout settings, and layout XML updates are merged with the product (in the event of a conflict, the product will win). Reference, `getDesignSettings` method.

Display Settings

Display mode: available options are: product, static block only, or products and static block.

Anchor: determines whether or not to include the products of child categories.

Layered Navigation Price Step: configures the price filter.

Layered Navigation templates are found in several files:

- https://github.com/magento/magento2/blob/2.2-develop/app/code/Magento/Catalog/view/frontend/templates/navigation/left.phtml
- https://github.com/magento/magento2/blob/2.2-develop/app/code/Magento/LayeredNavigation/view/frontend/templates/layer/filter.phtml
- https://github.com/magento/magento2/blob/2.2-develop/app/code/Magento/LayeredNavigation/view/frontend/templates/layer/state.phtml
- https://github.com/magento/magento2/blob/2.2-develop/app/code/Magento/LayeredNavigation/view/frontend/templates/layer/view.phtml

Demonstrate an understanding of configuring design inheritance for category pages.

This is configured in a category's design tab, "Use Parent Category Settings" option.

How can a CMS block be configured as a category landing page?

By setting the category's Display Mode to be Static Block only.

8.4 CUSTOMIZING CMS PAGES

CMS pages are configured in Content > Pages.

How can design changes (page layout) be configured on CMS pages?

These updates are found on the CMS page's design tab. You can adjust the layout, theme, and the page's layout XML.

Demonstrate an understanding of static variables in CMS blocks and pages. Demonstrate an understanding of the use of CMS template directives (var, store, block ...).

A content manager can utilize static variables to store small pieces of information and make them easily updatable. There are two type of variables: contact information and custom variables. The list of contact information variables comes from these files (https://github.com/magento/magento2/blob/2.2-develop/app/code/Magento/Email/Model/Template/Filter.php and https://github.com/magento/

magento2/blob/2.2-develop/app/code/Magento/Email/Model/Source/Variables.php). The list of custom variables are set up in System > Custom Variables.

Template directives can be found in this file. Search for the following: `Directive(`

`var`: loads a variable that is present in the current scope. This is found in the `$this->templateVars` array, which is set when `setVariables` is called. As such, it is not available for use in CMS pages.

`store`: formulates a URL. See the `storeDirective` method for all available options:

```
{{store url="about-us"}}

// renders: https://swiftotter.com/about-us
```

`block`: renders a block. Specifying the `class` attribute allows you to instantiate a block of a particular type. All of the attributes specified in this directive are passed to the block with magic setters. If no `class` attribute is specified and an integer `id` attribute is, a CMS block is rendered for the ID specified.

```
{{block class="\SwiftOtter\Test\Block\TestBlock" test="1"}}

{{block id="id_from_block_id_column"}}

<!-- OR -->

{{block id="id_from_identifier_column"}}
```

`if`: if a variable has a value, then the value will be printed.

```
{{if customer_name}}

Hello {{var customer_name}},

{{else}}

Hello no name,

{{/if}}
```

HELPFUL LINKS:

- https://www.atwix.com/magento-2/directives/
- https://gordonlesti.com/magento-2-email-template-template-directives/
- https://gordonlesti.com/magento-2-email-template-config-directives/

8.5 CUSTOMIZING WIDGETS

How is a widget instance created? Where can widgets be used? Demonstrate an understanding of configuring a widget instance.

Block widgets (as specified in `widgets.xml`):

Widgets are a tool for placing complex components into CMS output. They provide a centralized mechanism for rendering and retrieving information.

A widget is configured in a module's `etc/widget.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>

<widgets xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"

xsi:noNamespaceSchemaLocation="urn:magento:module:Magento_
Widget:etc/widget.xsd">

    <widget id="sample_widget" class="SwiftOtter\Test\
Block\TestWidget"

            placeholder_image="SwiftOtter_Test::images/
test_widget.jpg">

        <label translate="true">Sample Widget</label>

        <description translate="true">Incredible and
functional widget</description>

        <parameters>

            <parameter name="your_name" xsi:type="text"
visible="true" required="true" sort_order="10">

                <label translate="true">Your Name</label>

            </parameter>
```

```
            </parameters>

        </widget>

    </widgets>
```

Each widget must implement `\Magento\Widget\Block\BlockInterface`. Many widgets extend `\Magento\Framework\View\Element\Template`. In your widget's block, you can set up the template or execute any custom code for that widget.

Widgets can be used anywhere that the content is filtered by a template filter. This includes:

- CMS pages
- CMS static blocks (beware of recursion)
- Email templates

Content widgets (as created in Content > Widgets):

Widgets allow you to inject a CMS block into a location in layout XML configuration. This is a great option for content managers adding promotional material to the website. You can select from a number of different layout handles and then choose a container in which to inject the CMS block.

How can a custom widget target be created?

New targets can be created to facilitate widgets being injected into. Do this with the `<containers/>` tag found inside the `<widget/>` configuration:

```
<widgets>

    <widget id="test" class="SwiftOtter\Test\Block\
TestWidget">

        <!-- … -->

        <containers>

            <container name="content">

                <template name="grid" value="default" />

            </container>

        </containers>

    </widget>

</widgets>
```

In the above, each `container` node is mapped to a `container` node in layout XML (as defined by the `name` attribute).

When a widget is displayed in a container, a content manager needs to be able to control the template specified. The `<template/>` nodes allow you to specify options. These take a `name` and `value` attribute. We aren't seeing anything that happens with the `name` attribute other than the name must be unique. The `value` specifies which template is applicable inside this container. Specify multiple

`<template/>`s and the content manager will be able to select from a list of templates.

HELPFUL LINKS:

- https://magento.stackexchange.com/a/147982/13
- https://inchoo.net/magento-2/magento-2-custom-widget/

8.6 CUSTOMIZING CMS BLOCKS

How do you create and insert CMS blocks? Demonstrate an understanding of the use of CMS template directives (var, store, block ...).

CMS blocks are managed in Content > Blocks. CMS blocks can be inserted into other parsed variables with the `{{block id=""}}` directive. Note that the `id` parameter can either be the block's numeric ID (from the `block_id` column) or identifier (source).

You can also add a CMS block with layout XML:

```
<block type="Magento\Cms\Block\Block" name="custom.cms.
block">

  <arguments>

    <argument name="block_id" xsi:type="string">custom_
```

```
    block_identifier</argument>

      </arguments>

    </block>
```

See above discussion for information about CMS template directives.

8.7 CUSTOMIZING CUSTOMER ACCOUNT PAGES

All customer account pages include the `customer_account` layout handle (see https://github.com/magento/magento2/blob/2.2-develop/app/code/Magento/Customer/view/frontend/layout/customer_account.xml). Utilizing this will allow you to add common elements to all pages.

How do you remove or add an item from the customer account navigation using layout XML?

```
    <!-- app/code/SwiftOtter/Test/view/frontend/layout/
    customer_account.xml -->

    <?xml version="1.0"?>

    <page xmlns:xsi="http://www.
    w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="urn:magento:framework:View/
    Layout/etc/page_configuration.xsd">
```

```
    <body>

        <referenceBlock name="customer-account-navigation-
wish-list-link" remove="true"/>

        <referenceBlock name="customer-account-navigation-
billing-agreements-link" remove="true"/>

    </body>

</page>
```

Source: https://magento.stackexchange.com/a/91666/13

Demonstrate an understanding of formatting customer addresses.

Customer addresses are configured in Stores > Configuration > Customers > Customer Configuration > Address templates. These templates use the directives discussed above in 8.4 Customizing CMS pages.

8.8 CUSTOMIZING ONE-PAGE CHECKOUT

Demonstrate an understanding of the container blocks provided in the Magento checkout to display additional information.

Container blocks provide easy places to add elements to the checkout. The primary file for the checkout is this.

There are three layout containers specified: `shipping-step`, `billing-step` and `sidebar`. With each of the names listed below, new components are merged into the `children` node below the name specified. For example:

```
<!-- … -->

<item name="itemsAfter" xsi:type="array">

  <item name="component" xsi:type="string">uiComponent</
item>

    <item name="children" xsi:type="array">

      <!-- merge your components here -->

    </item>

  </item>

</item>

<!-- … -->
```

The `shipping-step` contains the following extension points (uiComponent, template):

- `before-form`
- `before-fields`
- `address-list-additional-addresses`
- `before-shipping-method-form`

The `billing-step` is used for collecting the customer's payment. (uiComponent, template)

- `payment` > `renders`
- `payment` > `beforeMethods`
- `payment` > `afterMethods`

The `sidebar` provides an overview of order details. (uiComponent, template)

- `summary`
- `summary` > `totals`
- `summary` > `itemsBefore`
- `summary` > `itemsAfter`

HELPFUL LINKS:

- https://www.mage2.tv/content/javascript/

8.9 UNDERSTAND CUSTOMIZATION OF TRANSACTIONAL EMAIL TEMPLATES

How do you create and assign custom transactional email templates? How do you use template variables available in all

emails? How do you access properties of variable objects (for example, var order.getCustomer.getName)?

Custom transactional email templates are configured in Marketing > Email Templates. These extend or inherit an existing email template. See here for a simplified lesson on creating a new transactional email template.

Once you have created a new transactional email template, you need to instruct Magento which email template to use. This is done in Store > Configuration. For order emails, go to Sales > Sales Emails. For customer emails, go to Customer > Customer Configuration.

Transactional email templates use the same directives as was discussed in 8.4. You can utilize the {{var ...}} attribute in these email templates for any variable that was specified when initializing the email. For example, for the order notification template, the following variables are available (source):

- `order`: `{{var order.getCustomerEmail}}`
- `billing`: `{{var billing.getFirstname}}`
- `payment_html`: `{{var payment_html}}`, renders the HTML output from the payment block
- `store`: `{{var store.getName}}`
- `formattedBillingAddress`: `{{var formattedBillingAddress}}`
- `formattedShippingAddress`: `{{var formattedShippingAddress}}`

Additionally, in the above example, the `email_order_set_template_vars_before` is dispatched so you can modify the parameters or add more to the list.

How can you create a link to custom images from transactional email templates? How do you create links to store pages in transactional email templates?

There is not a WYSIWYG image uploader for transactional email templates. To add an image, you need to place the image in a module or a theme's `web/images` directory. You can load the image like (see, `viewDirective` method):

```
<img src="{{view url="SwiftOtter_Test/images/your-file-name.
png"}}" title="SwiftOtter.com"/>
```

To create links to store pages, use the `{{store url="..."}}` directive. The value in the `url` parameter is relative to the store's URL. If your store URL is `https://swiftotter.com/` and you want to link to `https://swiftotter.com/test-url`, you would add the following directive:

```
<a href="{{store url="test-url"}}" title="Test URL">Go to
our test url</a>
```

HELPFUL LINKS:

- https://devdocs.magento.com/guides/v2.2/frontend-dev-guide/templates/template-email.html
- https://community.magento.com/t5/Magento-2-x-Admin-Configuration/transactional-emails-where-do-images-go/m-p/91304/highlight/true#M2388

Magento 2 Certified Professional
Frontend Exam Preparation eBook

9.
IMPLEMENT INTERNATIONALIZATION OF FRONTEND PAGES

SWIFT OTTER SOLUTIONS

(3 QUESTIONS - 5%)

9.1 CREATE AND CHANGE TRANSLATIONS

Demonstrate an understanding of internationalization (i18n) in Magento. What is the role of the theme translation dictionary, language packs, and database translations?

Internationalization has been a core Magento feature since its early days. Magento 2 maintains strong support across the entire platform.

Magento includes a feature to locate all translatable strings within a particular path. You can utilize this for an entire Magento installation or just for a module or a theme.

To find all translatable strings for a module (or modules):

```
bin/magento i18n:collect-phrases app/code/SwiftOtter/Test
```

To assist in building a language package, you need to locate all strings within the Magento application. You can run this command to obtain this information:

```
bin/magento i18n:collect-phrases -m
```

When run with the `-m` flag, two additional columns are added: `type` and `module`. `type` is either theme or `module`. The `module` column represents the module that utilizes this translation.

Theme translation dictionary

A theme translation dictionary allows you to specify translations for words used in a theme or a module. These phrases are placed in a `.csv` inside your module or theme's `i18n` directory (`app/code/SwiftOtter/Test/i18n/de_DE.csv`).

Translations specified for the theme or module are the first two sources of translation data. These translations can be overridden by a language package or in the database.

HELPFUL LINKS:

- https://devdocs.magento.com/guides/v2.2/config-guide/cli/config-cli-subcommands-i18n.html#config-cli-subcommands-xlate-dict
- https://devdocs.magento.com/guides/v2.2/frontend-dev-guide/translations/theme_dictionary.html

Language packs

A language pack allows you to translate words used anywhere in Magento. The source for this is `bin/magento i18n:collect-phrases` command with the `-m` flag. This searches the entire Magento application (including modules and themes) for all translatable strings.

The output from this command is the fundamental ingredient to a language pack.

Once you have translated the strings, you then execute this command (substituting the path to the CSV file and specifying the target language):

```
bin/magento i18n:pack /absolute/path/to/file.csv de_DE
```

You must then create a new language module within the `app/i18n/SwiftOtter_FR/` (or something similar). This contains the usual module files (`registration.php` uses the `\Magento\Framework\Component\ComponentRegistrar::LANGUAGE` component type). It also includes a `language.xml` file:

```
<?xml version="1.0"?>

<language xmlns:xsi="http://www.
w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:App/
Language/package.xsd">

  <code>fr_FR</code>

  <vendor>SwiftOtter</vendor>

  <package>fr_fr</package>

  <use vendor="Magento" package="fr_FR"/>

</language>
```

You can include multiple `<use/>` nodes in the `language.xml` file. If Magento does not find a string in the included language package, it will search through each `<use/>` (and subsequently those language package's `<use/>` nodes) until it finds an applicable translation.

If no translation is found, module or theme's own translation is used. If none is found, the original, untranslated text is returned.

Inside the language package, include the CSV generated above named as the contents of the `<code/>` node with a `.csv` suffix.

HELPFUL LINKS:

- https://devdocs.magento.com/guides/v2.2/config-guide/cli/config-cli-subcommands-i18n.html#config-cli-subcommands-xlate-pack
- https://github.com/splendidinternet/Magento2_German_LocalePack_de_DE
- https://github.com/magento/magento2/tree/2.2/app/i18n/Magento/fr_FR

Database translations

Database translations are the easiest to implement, but the most difficult to transfer from installation to installation. They are found in the `translation` table.

To create a new translation, the easiest is to turn on Inline Translation (Store > Configuration > Developer > Translate Inline). You could also insert new rows in the the `translation` table manually with the limitation being the need to determine the module to associate the translation to.

While translating with Inline Translation, disable the `Translations`, `Block HTML` and `Full Page` caches.

Understand the pros and cons of applying translations via the translate.csv file versus the core_translate table. In what priority are translations applied?

Any translations found in the `translate.csv` are easily replicated to other instances where this file is utilized. However updating a file is usually less convenient than using the translate inline feature for the Magento frontend.

Translations are applied in this order:

1. Module translations
2. Theme translations
3. Translation package
4. Database translations

As a result, database translation is good for overriding other translations where necessary.

9.2 TRANSLATE THEME STRINGS FOR .PHTML, EMAILS, UI COMPONENTS, .JS FILES

PHTML:

```
<?= __('Shopping cart');?>

<?= __("There are %s items in your cart", $count); ?>
```

Email templates:

```
{{trans "Shopping Cart"}}

{{trans "%items are shipping today." items=shipment.
getItemCount}}
```

Demonstrate an understanding of string translation in JavaScript.

The first example under each heading below shows rendering a plain string. The second example shows how to use substitutions.

UI Component templates:

```
<span data-bind="i18n: 'Shopping Cart'"></span>

<input type="text" data-bind="attr: {placeholder:
$t("Email")}"/>

<translate args="'Shopping Cart'"></translate>

<span translate="'Shopping Cart'"></span>
```

UI Component configuration files:

specify `translate="true"` on the element that contains text.

JS files:

require the `jquery` and `mage/translate` modules, then

```
$.mage.__('Shopping Cart')

$.mage.__("%1 items in your cart").replace("%1",
numberOfItems);
```

Alternatively require only `mage/translate` and assign it to a variable named `$t`.

```
$t('Shopping Cart')
```

The method names `$t` or `$.mage.__` are mandatory, otherwise the JavaScript translation string will not be included in the generated js-translation.json file.

HELPFUL LINKS:

- https://devdocs.magento.com/guides/v2.2/frontend-dev-guide/translations/translate_theory.html
- https://www.mage2.tv/content/javascript/localization-in-javascript/

Magento 2 Certified Professional
Frontend Exam Preparation eBook

10. MAGENTO DEVELOPMENT PROCESS

(2 QUESTIONS - 3%)

SWIFT OTTER SOLUTIONS

10.1 DETERMINE ABILITY TO MANAGE CACHE

Demonstrate an understanding of configuring the Magento cache types for development and production.

Cache types can be configured in the admin area, by navigating to:

System > Tools > Cache Management

Note that you cannot disable caches in the Magento admin when in production mode.

Cache types can also be configured on the command line:

```
bin/magento cache:enable

# OR

bin/magento cache:disable

# OR

bin/magento cache:flush

# OR

bin/magento cache:clear full_page
```

You can see which caches are enabled and disabled, in the admin area, or with the command

```
bin/magento cache:status
```

You can abbreviate any of these commands within each namespace so long as that namespace is still unique.

One thing to note is that you cannot disable caches in the admin panel while in production mode.

HELPFUL LINKS:

- https://devdocs.magento.com/guides/v2.2/config-guide/cli/config-cli-subcommands-cache.html

10.2 UNDERSTAND MAGENTO CONSOLE COMMANDS

How do you switch between deploy modes? What bin/magento commands are commonly run during frontend development?

To switch between deployment modes:

```
bin/magento deploy:mode:set production

OR

bin/magento deploy:mode:set developer
```

To show the current deployment mode:

```
bin/magento deploy:mode:show
```

To clear all caches:

```
bin/magento cache:flush
```

To disable a cache:

```
bin/magento cache:disable full_page
```

To see the enabled or disabled state of caches:

```
bin/magento cache:status
```

To symlink the JS, LESS, and image files into the `pub/static` folder during development:

```
bin/magento dev:source-theme:deploy
```

When pushing code to production, to compile the frontend assets (note that this does not work in developer mode):

```
bin/magento setup:static-content:deploy
```