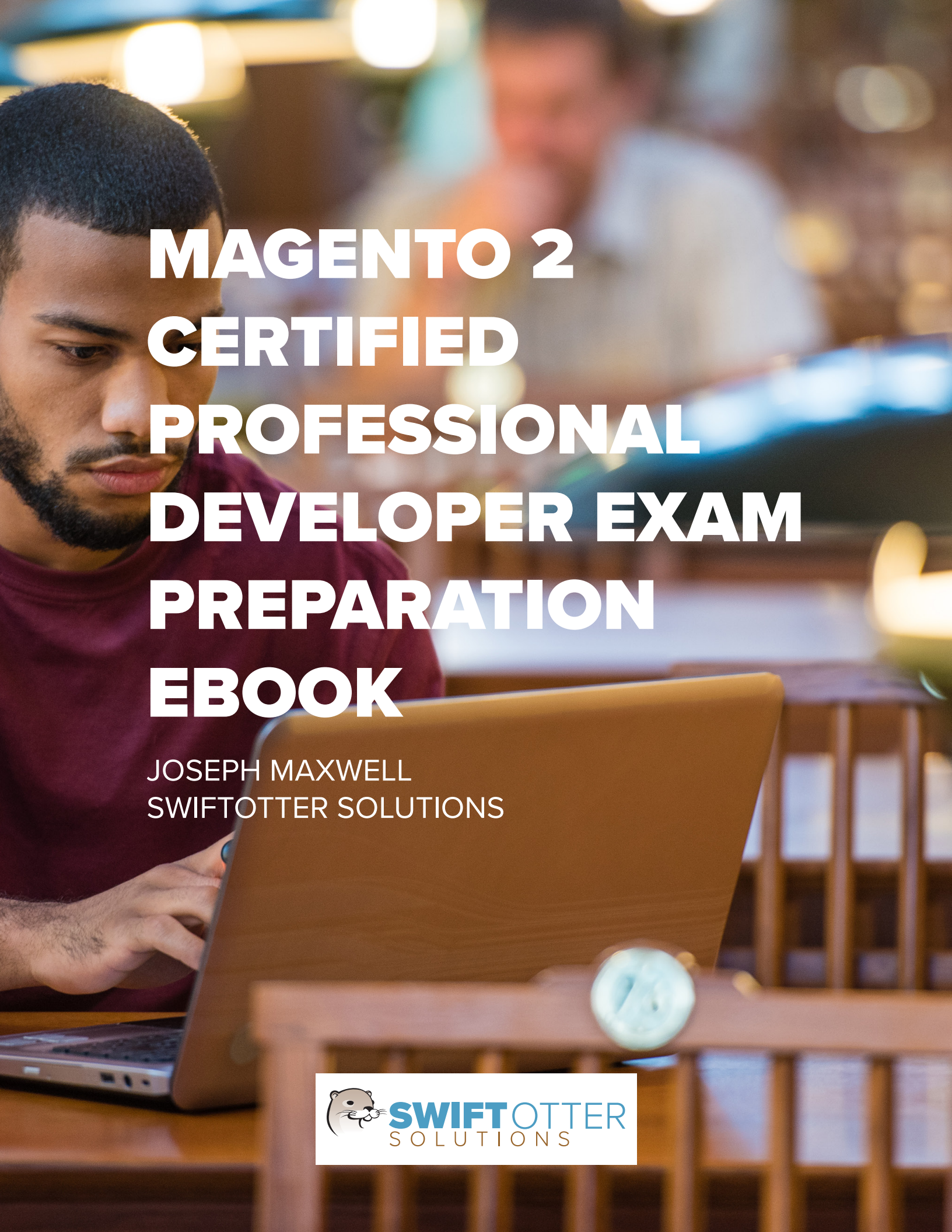


MAGENTO 2 CERTIFIED PROFESSIONAL DEVELOPER EXAM PREPARATION EBOOK

JOSEPH MAXWELL
SWIFTOTTER SOLUTIONS

SWIFTOTTER SOLUTIONS

INTRODUCTION

You have taken the first step toward becoming a Magento 2 Certified Professional Developer by downloading SWIFTotter's study guide. We've worked hard to produce a quality annotated eBook. Now it's your turn to get to work improving your Magento 2 knowledge and your future.

Detailed explanations are throughout, but if you are advanced, you can skim and refresh as you prepare for the exam. Novices can deep dive for a true understanding of Magento 2, not just cramming soon-forgotten facts for a test.

Magento is a world-case platform and highly skilled professional developers elevate the whole community. The better you are, the more everyone benefits.

The best way to pass the test is to know Magento 2.

- The test is 60 questions and 90 minutes — 1.5 minutes per question.
- The test questions are scenario-based. You are provided information and a relevant question. You then choose the appropriate answer(s).
- If you know Magento 2, 1.5 minutes is long enough to answer. Your knowledge of Magento 2 is what will carry you through.

Here are some other interesting reads about the exam preparation:

- https://www.integer-net.com/magento-2-certified-professional-developer-about-the-exam/

I would also like to sincerely thank my friend, Vinai Kopp, for his excellent and detailed review of the study guide. His expert thoughts were greatly appreciated.

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.

DRIVER

Driver—transforming your production database to staging and local.

There are a number of tools to assist in getting production data back to staging. One of these is Sonassi's. They work well.

But, there are some inherent shortcomings:

- These are limited by your skill level with `sed`.
- No way to run SQL commands.
- Transformations happen in the production environment.

What if you could:

- Run custom SQL commands to generate custom output for multiple environments?
- Automatically upload to S3?
- Store configuration in git, making it easy to replicate and adjust?

We have built a tool just for that: Driver.

Driver allows you to do all the above. For example, we generate data for three environments: staging, local initialization and local refresh. For the latter two environments, the customer and order data has been so sanitized that we have Driver set a standard admin login and password. The only thing that touches the production database is the initial `mysqldump` command.

This process has become so automated that we can run one command on our development machine to refresh the data: `reload_customer_name`.

How does it work?

The way Driver works is the production data is dumped and uploaded to a fresh Amazon RDS (relational data storage) instance. The transformations are run for each environment; that environment is dumped and loaded into S3. You can easily automate the synchronizations back to your staging and local environments.

https://github.com/SwiftOtter/Driver

CI

Are you familiar with Jenkins automation? Maybe you are having trouble getting code to production. We have developed an open-source continuous integration and deployment system built on Jenkins to make building and deploying Magento 1 and 2 websites easy.

Additionally, our system enables deploying Magento 2 code with zero downtime (or very little—if database updates are necessary).

Go to our website for more information.

CONTENTS

Magento 2 Certified Professional Developer Exam Preparation eBook

1. MAGENTO ARCHITECTURE

AND CUSTOMIZATION TECHNIQUES

SWIFT OTTER SOLUTIONS

Magento 2's architecture is much more condensed than Magento 1. As such, for developers who are making the switch, there are some nuances to be aware of.

In this study guide, we will be working with a fictitious module for a company called AcmeWidgets. The module we are working on is named ProductPromoter. It will show or hide products based on specified parameters.

1.1 DESCRIBE MAGENTO'S MODULE-BASED ARCHITECTURE

POINTS TO REMEMBER

- There are five areas: `adminhtml`, `frontend`, `base`, `webapi_rest`, `webapi_soap` and `cron`. Not all areas are always available. For instance, the cron area is only used when running cron jobs.
- The three necessary files to bootstrap a module are `registration.php`, `etc/module.xml` and `composer.json`.

Overview

Magento 2's module-based architecture keeps that module's files in one folder. This makes discovery of the functionality pertaining to that module easier.

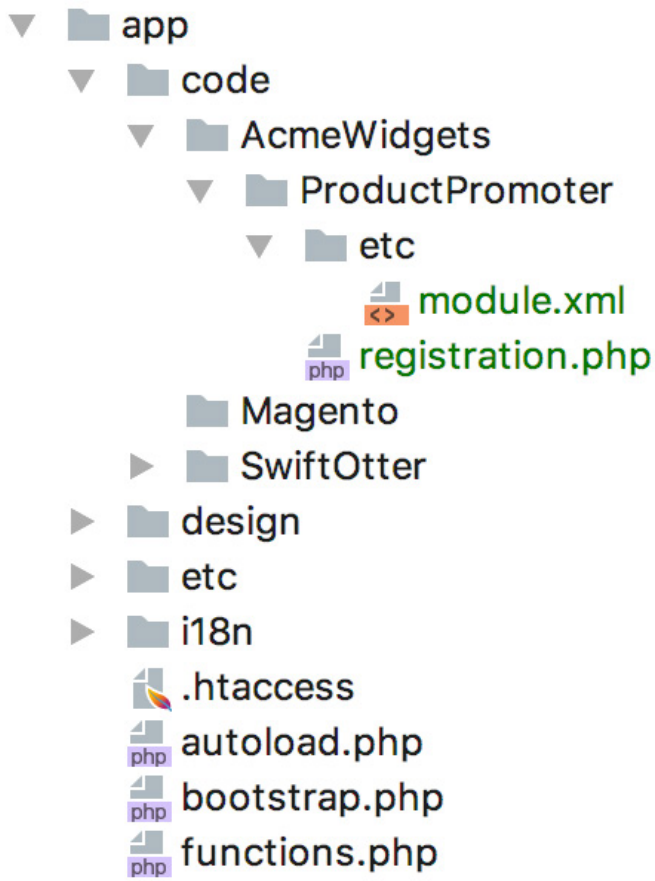
Modules live in one of two places:

- `app/code/CompanyName/ModuleName`
- `vendor/vendor-name/module-name`

With Magento 2, modules can be installed with Composer, making for easier upgrades and version management. These modules are placed in `/vendor`. Any file in `/vendor` is considered “untouchable” (not directly editable) as those files will be removed when new versions are released. Installing modules via Composer into the `/vendor` folder reduces technical debt as you no longer have to directly manage the versions of these modules: Composer does it for you.

In most cases, the modules that you develop go into `app/code/CompanyName`. This is the namespace used for development. Modules are placed in the folder directly below. With Magento’s dependency on Composer, you can theoretically import modules from almost any location in a project.

Example:



The only required files to initiate a module is:

- `registration.php`
- `etc/module.xml`

`registration.php`

This file is included by the Composer autoloader (`app/etc/NonComposerComponentRegistration.php`).

This adds the module (component) to the static list of components in `\Magento\Framework\Component\ComponentRegistrar`. Once it is there, Magento will look for `etc/module.xml`.

Example:

```php
<?php

\Magento\Framework\Component\ComponentRegistrar::register(
    type: \Magento\Framework\Component\ComponentRegistrar::MODULE,
    componentName: 'AcmeWidgets_ProductPromoter',
    path: __DIR__
);
```

`etc/module.xml`

This file specifies the setup version and loading sequence for the module. The setup version is available in the module's Setup classes to determine what upgrade activities should occur.

The sequence tells Magento (`vendor/magento/framework/Module/ModuleList/Loader.php`) in which order to load modules. This adjustment

happens only when the module is installed. The list of modules is stored in `app/etc/config.php`.

The sequence is useful for events, plugins and preferences, (although events' listeners should never be dependent on the order in which they are called) and layouts. If you modify the layout of another module and your module is initialized first, the other module will overwrite your adjustments.

Example:

```xml
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="urn:magento:framework:Module/etc/module.xsd">
    <module name="AcmeWidgets_ProductPromoter" setup_version="1.0.0">
        <sequence>
            <module name="Magento_Catalog"/>
            <module name="Magento_LayeredNavigation"/>
        </sequence>
    </module>
</config>
```

Describe module limitations.

Magento 2 modules are to be completely contained within the module's directory (ie. `app/code/AcmeWidgets/ProductPromoter`). All customizations and extensions are made within that folder.

If it is disabled, the module is not functional. Keep in mind that disabling a module does not automatically remove its tables or entries in the database.

If you are having trouble with modules not being enabled, check:

- The required files are present (see above).
- The module is enabled (`bin/magento module:enable AcmeWidgets_ProductPromotor`).

How do different modules interact with each other?

Magento 2 provides a much-improved system for inter-module interaction. At its core, Magento 2 is PSR-4 compliant. This standard declares that the namespace path will match the file path to the class.

While the PSR-0 standard has been deprecated, Composer still uses this idea to assist in autoloading classes. In `/composer.json` (in the root folder), there should be a “psr-0” node. Inside it, will be an “app/code” value. This has the same effect of adding an include directory for PHP to search.

Magento 2 does not include the Mage “god” class like its earlier counterpart. Magento 2 relies on service contracts and dependency injection to determine what classes to use. This will be discussed more in the Dependency Injection section.

These service contracts are stored in the `app/code/AcmeWidgets/ProductPromoter/Api/` directory. Data representations are in the `app/code/AcmeWidgets/ProductPromoter/Api/Data` folder.

When building modules that are dependent on other modules, whenever possible, use interfaces as defined by the service contracts. That way, your module works with a blueprint and it does not matter what class or module fulfills that blueprint.

HELPFUL LINKS:

- http://devdocs.magento.com/guides/v2.0/architecture/archi_perspectives/components/modules/mod_and_areas.html
- Service contracts introduction: https://alankent.me/2014/10/31/magento-2-service-contract-patterns/

What side effects can come from this interaction?

Any time you are working with a system where any installed module can extend core functionality, fallout can happen.

When implementing according to Magento 2's guidelines, the fallout is limited. When breaking recommendations, problems happen.

It is good to assume that other modules will change things in the system. For example, let's say you install a module that introduces a massive change for a product. This change is so significant that the developer wrote an entirely new class to represent the product model. This class was written against the service contracts that Magento specifies for a product:

```
namespace \SwiftOtter\NewProduct\Model

use Magento\Framework\DataObject\IdentityInterface;

use Magento\Framework\Pricing\SaleableInterface;

use Magento\Catalog\Api\Data\ProductInterface;

class Product implements IdentityInterface,
SaleableInterface, ProductInterface
{
    // …
}
```

But what if your custom product code uses a method that is only defined in `\Magento\Catalog\Model\Product`? You (or your customer or a website

visitor) might get a PHP Fatal Error when it attempts to call a method that does not exist in the other class. This flexibility is incredible but also must be used responsibly.

When working with Magento-defined classes that have service contracts (`Api/` and `Api/Data` folders), use those interfaces instead of the concrete classes that implement the service contracts. If you must explicitly use a Magento-defined class (such as `\Magento\Catalog\Model\Product`), make the implicit dependency explicit by depending on the concrete implementation instead of the service contract interface.

HELPFUL FILES TO STUDY:

- Module loading: `vendor/magento/framework/Module/ModuleList/Loader.php`
- CLI command to enable a module: `vendor/magento/magento2-base/setup/src/Magento/Setup/Console/Command/AbstractModuleManageCommand.php`

1.2 DESCRIBE MAGENTO'S DIRECTORY STRUCTURE

POINTS TO REMEMBER

- Magento modules are found in `/app/code/Magento` or `/vendor/Magento`
- Frontend-related files are found in `view/`

Determine how to locate different types of files in Magento.

Magento 2 brings a very flexible system for building and structuring modules. The below answers will be a hybrid of Magento's practices and the freedom that you now have with naming files and folders.

Magento 2's core files are found in `/vendor/magento` or `/app/code/Magento` (only to be used for Magento core code contributions). Some supporting JS and CSS files are found in `/lib`.

File reference (in the module's home: `app/code/AcmeWidgets/ProductPromoter`)

`/Api`: Service Contracts

The `/Api` folder stores the contracts for modules that contain specific actions that can be reliably utilized from various places in the app. An example of this would be `\Magento\Catalog\Api\CategoryListInterface`.

`/Api/Data`: Data Service Contacts

This folder contains interfaces that represent data. Examples of this would be a Product interface, a Category interface or a Customer interface. The concrete implementations of these interfaces usually do little more than provide getters and setters for data (a.k.a Data Transfer Objects).

`/Block`: View Models (or template "assistants")

Very little PHP should be done in templates (separation of responsibilities: see here and here). As a result, Magento's blocks perform or provide business logic for the templates. In the MVVM (model, view, view-model) methodology Magento utilizes, Blocks are "View Models."

One problem with inheriting the template block is the constructor is very large. This makes testing and reusability difficult. The answer to this is using a View Model. While View Models live in the `Block/` directory, they are very different. A view Model provides the business logic of a block, and not the rendering functionality. This makes another distinction in code where the block is responsible for outputting its contents and the view model handles the logic.

To create a block with a view model, you would do something like (adapted from `vendor/magento/module-backend/Block/Template.php`):

```
 <block name="product.promotions" template="AcmeWidgets_
ProductPromoter::promotions.phtml" >
     <arguments>
         <argument name="viewModel"
xsi:type="object">AcmeWidgets\ProductPromoter\Block\
Promotions</argument>
     </arguments>
</block>
```

HELPFUL LINKS:

- Vinai Kopp wrote a must-read article on how to use these view models.

`/Console`: Console Commands

When running `bin/magento` on the command line, a list of available commands to run is output. The code for commands should reside in the `/Console` folder. An example of a console command is for the `bin/magento catalog:product-attributes:cleanup` command: `/vendor/magento/module-catalog/Console/Command/ProductAttributesCleanUp.php`.

`/Controller`: Web Request Handlers

When a page is requested from a Magento website, the path is parsed out and matched to parameters found in `routes.xml` and files found in the `/Controller` folder.

HELPFUL FILES:

- Route finder: `vendor/magento/framework/App/FrontController.php`
- Default router (for most frontend requests): `vendor/magento/framework/App/Router/Base.php`

This is a magic path. Magento expects admin controllers to be found in this folder.

`/Controller/Adminhtml`: Admin controllers

M1

MAGENTO 1

Remember back in the Magento 1 "good-old-days" where all of the Magento adminhtml files were in `app/code/core/Mage/Adminhtml` (Mage_Adminhtml module)? Thankfully, those days are over with Magento 2. Admin controllers now are spread through their respective and applicable modules.

`/Cron`: Cron Job Classes

This folder is the standard place for storing actions that are executed on a schedule (cron job).

`/etc`: Configuration files

Configuration files are placed here. Examples include: `module.xml`, `crontab.xml`, `config.xml`, `webapi.xml`, `di.xml`.

Any files directly within the `/etc` folder are global. The area of those files can be controlled by putting it within a folder with the name of the target area. For example, a `di.xml` within `/etc/frontend` would only apply to the front end.

Some files have to be within an area folder (e.g. `routes.xml` and `sections.xml`), some have to be global (e.g. `acl.xml`) and some can be global or area specific (e.g. `di.xml`).

`/Helper`: Occasionally useful for small, reusable code

With Magento 2, this folder is no longer necessary. However, the core still utilizes this folder for storing actions. In my experience, here are good rules to apply for functionality:

- Every method should be capable of being declared static (whether or not it is). No transient data (other than injected classes) is stored in a helper class.

> M1 **MAGENTO 1**
>
> Consider whether there is a better place to put the code before adding it to such a generic namespace. Since we are no longer limited by the Magento 1 folder structure, I have begun creating other folders that are more defined. For example, `/Query/TestQuery` instead of `/Helper/Query/TestQuery`.

`/i18n`: Translation CSV Files

This is where all a module's translation CSV files are located. In Magento 1, these were in `app/locale/de_DE/`.

There are CSV files with two columns: From, To

`/Model`: Data Handling and Structures

In the MVVM acronym, this directory houses the Models. I have found that many classes that are placed in the `/Helper` folder out to be in `/Model`. This folder stores anything that is related to the data structure (`vendor/magento/module-catalog/Model/Product.php`) to loading that data (`vendor/magento/module-catalog/Model/ProductRepository.php`).

`/Model/ResourceModel`: Database Interactions

These represent how data is retrieved and saved in the database. Any direct database interaction should happen in these files. Here is an example of the product's Resource model (`vendor/magento/module-catalog/Model/ResourceModel/Product.php`).

`/Observer`: Event Listeners

When Magento fires an event, listeners that are attached to it are called. This decouples the system. Magento Commerce integrates with RabbitMQ which allows even more control and reliability to this process. Event data should be able to be stored and then run in a queue at a later time.

Event listeners must implement the `ObserverInterface` (`\Magento\Framework\Event\ObserverInterface`). The observer class should be understandable as the intent of its function. The PHP class should follow the stand of using TitleCase while the event name should be `snake_case`. Avoid putting business logic into an observer. Put the logic in another location and inject that class into your `Observer`. When you need to use the logic elsewhere in the system, you will thank yourself.

`/Plugin`: Function Modification

Plugins are one of the most powerful features in Magento 2. They allow you to modify or change the functionality for almost any class or interface. Plugins only work on objects that are instantiated by the ObjectManager.

Plugins can be placed anywhere. Magento puts them in the `/Plugins` folder which makes them easy to find. I prefer to name the plugin file after the class that I am customizing followed by "Plugin." If I am modifying a method in `\Magento\Catalog\Api\Data\ProductInterface`, I would create the following `class: /Plugins/Magento/Catalog/ProductPlugin.php`. This reduces confusion when you import classes. Another approach is to name Plugins after what they do instead of the class. This makes it easier to locate the correct one when working with the PHPStorm Magento 2 plugin (e.g. MinimumPricePlugin instead of ProductPlugin on a Plugin into `Model\Product`)

`/Setup`: Database Modification

File list:

- `InstallSchema.php`: sets up table / column schema when the module is installed.
- `UpgradeSchema.php`: modifies table / column schema when the module version is upgraded.
- `Recurring.php`: runs after every install or upgrade.
- `InstallData.php`: sets up data when the module is installed. An example would be adding a custom CMS block.
- `UpgradeData.php`: modifies data after the module is installed, when the module version is upgraded.
- `RecurringData.php`: applies to data after every install or upgrade.

HELPFUL FILES:

- `setup/src/Magento/Setup/Model/Installer.php`

`/Test`:

This houses the module's tests. `/Test/Unit` stores unit tests. You can run tests via the command line: `bin/magento dev:tests:run [all, unit, integration, integration-all, static, static-all, integrity, legacy, default]`

`/Ui`: UI Component Data Providers

This folder stores the data providers and modifiers for UI components.

`/view/[area]/layout`: Layout XML Directives

Layout XML links blocks (/Blocks) with templates. It is a very flexible means to control what is displayed where.

`/view/[area]/templates`: Block Templates

The counterpart to a Block is a template to render the HTML for the block. While the block (in `/Block`) represents the business logic, the template represents how the results of the business logic are shown to the user. Again, the best is for as little PHP as possible to go in the templates.

When building modules, keep in mind that this folder here is plural "templates."

`/view/adminhtml/ui_component`: UI Components

The checkout on the frontend is also a UI component, but this is configured with layout XML and not UI Component XML. You can also find a `/templates` folder inside the `ui_component` folder that contains some templating for UI components

This folder contains the XML configuration for UI Components. They are used to represent distinct UI elements, such as grids and forms, and were designed for flexible user interface rendering. Most Magento admin grids (such as the Catalog Product grid and the Customer grid) are composed with UI Components. The checkout on the frontend is also a UI component.

`/view/[area]/web`: Web Assets

This is where the web assets are stored. This includes JS, CSS (or LESS / SCSS), and images.

`/view/[area]/web/template`: JS Templates

HTML templates which can be asynchronously requested with Javascript (possible using Magento's customization of KnockoutJS) are placed here. These files often contain KnockoutJS declarative bindings. This folder is singular "template"—which can be confusing as the PHTML and the XHTML template directories are plural.

`/view/[area]/requirejs-config.js`

The module's RequireJS configuration. This configuration is used to control Javascript module dependencies, create aliases, and declare mixins.

Where are the files containing JavaScript, HTML, and PHP located?

JavaScript: in the `/view/[area]/web/` folder

HTML: in the `/view/[area]/templates` folder (with .phtml file extensions)

PHP: any folder except for `/view/[area]/web/`

How do you find the files responsible for certain functionality?

The files and folder list above detail out what files are stored where.

1.3 UTILIZE CONFIGURATION XML AND VARIABLES SCOPE

Magento's XML configuration is split across multiple files, depending on the purpose. This helps avoid having one very large configuration file.

M1

MAGENTO 1

Magento 2's XML configuration looks much different than Magento 1. Because Magento 2 has adopted the PSR-4 convention, much of the boilerplate that was in `etc/config.xml` in Magento 1 has been eliminated. There are a few assumed paths (such as `/Setup`), but almost everything else is left for you to arrange for the optimal layout of the module.

Additionally, Magento 2 supports configuration based on the area. This can be seen in `vendor/magento/magento-catalog/etc/` where there are the following folders: `adminhtml`, `frontend`, `webapi_rest`, `webapi_soap`. The configuration that is in these folders is only loaded if Magento is initialized in that area. For example, when browsing the admin panel, the configuration found in frontend, `webapi_rest` or `webapi_soap` is not loaded.

We will discuss the more important XML files found in the `/etc` folder.

`module.xml`

This is the only required configuration file. It specifies the current module's version and the module loading order. Example: `vendor/magento/module-catalog/etc/config.xml`

`acl.xml`

This defines the permissions for accessing protected resources. Example: `vendor/magento/module-catalog/etc/acl.xml`

`config.xml`

This loads in default configuration in to Store > Configuration. This is also where configuration entries can be marked as encrypted (password). See `vendor/magento/module-fedex/etc/config.xml`

`crontab.xml`

This identifies actions that are to occur on a schedule. Example: `vendor/magento/module-catalog/etc/crontab.xml`

`di.xml`

This configures dependency injection for your module. This is perhaps the most frequently used file when customizing Magento. Here plugins are defined, class substitutions performed, concrete classes are specified for interfaces, virtual types setup, and constructor arguments can be specified or modified.

It is very important to familiarize yourself with the capabilities of this file.

Example: `vendor/magento/module-catalog/etc/di.xml`

`email_templates.xml`

Specifies email templates that are used in Magento. The template id is the concatenated XML-style path to where in system configuration template is specified.

Example: `vendor/magento/module-customer/etc/email_templates.xml`

`events.xml`

This file registers event listeners. This file can often be put into a specific area.

Example: `vendor/magento/module-catalog/etc/frontend/events.xml`

`indexer.xml`

Configures Magento indexers. Example: `vendor/magento/module-catalog/etc/indexer.xml`

`adminhtml/menu.xml`

Configures the menu in the adminhtml area.

Example: `vendor/magento/module-customer/etc/adminhtml/menu.xml`

`mview.xml`

Triggers a type of event when data is modified in a database column (materialized views). This is most often used for indexing.

Example: `vendor/magento/module-catalog/etc/mview.xml`

`[area]/routes.xml`

Tells Magento that this area accepts web requests. The route node configures the first part of the layout handle (route ID) and the front name (first segment in the URL after the domain name).

`adminhtml/system.xml`

Specifies configuration tabs, sections, groups and fields found in Store Configuration.

Example: `vendor/magento/module-customer/etc/adminhtml/system.xml`

`view.xml`

Similar to `config.xml` but used for specifying default values for design configuration.

Example: `vendor/magento/theme-frontend-luma/etc/view.xml`

`webapi.xml`

Configures API access and routes. Example: `vendor/magento/module-catalog/etc/webapi.xml`

`widget.xml`

Configures widgets to be used in products, CMS pages, and CMS blocks.

Example: `vendor/magento/module-catalog/etc/widget.xml`

Determine how to use configuration files in Magento.

Magento's XML configuration is loaded on an as-needed basis. When `di.xml` is needed, Magento finds all `di.xml` files and merges them together.

Using a configuration file is easy: create that file in the `/etc/` folder. Ideally, the file can be limited to a specific area, such as the frontend or adminhtml. Copy an existing file from another module to start with a boilerplate that works.

You can also create custom configuration files. Several updates are required to do this:

- A XSD defining the schema to validate configuration files.
- VirtualType created that extends `\Magento\Framework\Config\Reader\Filesystem`, specifies the `converter`, `schemaLocator`, and `fileName`.
- A `Converter` class that implements `\Magento\Framework\Config\ConverterInterface`
- A `SchemaLocater` class that implements `\Magento\Framework\Config\SchemaLocatorInterface`
- A `Data` class that extends `Magento\Framework\Config\Data`

HELPFUL LINKS:

- http://devdocs.magento.com/guides/v2.2/config-guide/config/config-files.html#config-files-classes-int
- https://www.atwix.com/magento-2/working-with-custom-configuration-files/

Which configuration files correspond to different features and functionality?

Mostly detailed above. Some less popular files:

- `address_formats.xml`: the output types for an address
- `extension_attributes.xml`: generic entity extensibility system.
- `product_options.xml`: types of product options (text, file, select, etc.)
- `product_types.xml`: stores product types (simple, configurable, etc.)

HELPFUL LINKS:

- http://devdocs.magento.com/guides/v2.2/config-guide/config/config-files.html

1.4 DEMONSTRATE HOW TO USE DEPENDENCY INJECTION

POINTS TO REMEMBER

- Dependency injection is a means of giving a class what it needs to function.
- `ObjectManager` is Magento's internal object storage unit and should rarely be directly accessed.
- The `ObjectManager` makes implementing the Composition over Inheritance principle possible.
- Dependency injection makes testing easier, application more configurable and provides options for powerful features such as plugins.

Magento is very customizable. Magento's Dependency Injection concept embraces that and allows a great deal of control.

Dependency Injection is literally injecting what a class' dependencies are into the constructor or setter methods.

Alan Kent has a great article about dependency injection and its benefits.

The beauty of dependency injection is that it is very easy to see what your class needs are at a glance. You build your class around the class or interface that you inject. You do not care what class or interface is injected.

Describe Magento's dependency injection approach and architecture.

Magento's dependency injection framework is unique in that it is very automatic. Many other frameworks require at least some level of configuration to get going. Magento provides ways to customize and adjust dependency injection on the fly as well.

Magento uses constructor injection: that is, all of the dependencies are specified as arguments in the `__construct()` function.

Before we continue, I should note that it is very poor practice to directly use the `ObjectManager`—the primary class that handles dependency inject. It is against Magento standards to do this (exception for only a few cases).

Here is a sample constructor:

```
<?php
namespace AcmeWidgets\ProductPromoter\Component;
class Details
{
    private $registry;
    public function __construct(\Magento\Framework\Registry
$registry)
    {
        $this->registry = $registry;
    }
}
```

We specified a constructor for the above class. We are loading a reference entry to the `Registry` class. This happens by setting the class type and an argument name on the constructor.

Magento's DI container holds a list of objects. Each time one is created, it is added into this container. Whenever an object is requested, it is loaded from this container. In PHP, objects are references. Unless you clone the object, anywhere you pass the object (as an argument) that same object is referenced.

HELPFUL LINKS:

- http://devdocs.magento.com/guides/v2.2/coding-standards/technical-guidelines.html
- http://devdocs.magento.com/guides/v2.2/extension-dev-guide/proxies.html

For objects that might be time intensive to create, Magento provides proxies. A proxy lazy loads the class. To utilize this, specify a class like `\Magento\Catalog\Model\Product\Proxy` in `di.xml`. Proxies must NOT be specified in the `__constructor` method.

For objects that need to be new every time they are used, Magento uses factories. To use a factory, specify the class or interface name and append Factory to the end. Like: `\Magento\Catalog\Api\Data\ProductInterfaceFactory`. Magento will find the preference for the ProductInterface (default is `\Magento\Catalog\Model\Product`) and create a factory for that class. The factory has one public method, `create`. Calling this method creates a new instance of the desired class. Magento creates these classes automatically. If you want, you can create factory classes. Here is an example: `vendor/magento/module-paypal/Model/IpnFactory.php`.

M1

MAGENTO 1 VS MAGENTO 2

A key concept to understand when comparing Magento 1 and Magento 2 is this:

- In Magento 1, when you needed a product object, you **went and got it** (`Mage::getModel('catalog/product');`).
- In Magento 2, you are **given** the object in the constructor. If you need a new / clean object, you choose a Factory class.

For objects that might be time intensive to load, Magento provides proxies. A proxy lazy loads the class. To utilize this, specify a class like `\Magento\Catalog\Model\Product\Proxy` in the constructor.

How are objects realized in Magento?

Since dependency injection happens automatically in the constructor, Magento must handle creating classes. As such, class creation either happens at the time of injection or with a factory.

Class creation at the time of injection

A great way to watch this step-by-step is to set a breakpoint in `vendor/magento/framework/App/Router/Base.php::matchAction,` on the line that contains `$this->actionFactory->create()`.

The first step in the process is the object manager locating the proper class type. If an interface is requested, hopefully an entry in `di.xml` will provide a concrete class for the interface (if not, an exception will be thrown).

The deploy mode (`bin/magento deploy:mode:show`) determines which class loader is used.

- Developer: `vendor/magento/framework/ObjectManager/Factory/Dynamic/Developer.php`
- Production: `vendor/magento/framework/ObjectManager/Factory/Dynamic/Production.php`

The parameters for the constructor are loaded. Then those parameters are recursively parsed. Not only are the dependencies for the initially requested class loaded, but dependencies of dependencies as well.

A metaphor of this would be a tree. In a tree, you have the trunk and then the branches. The trunk would represent an object type. But that object has

dependencies, which continue splitting and going up the tree. Eventually, you have all the branches and all the leaves representing all of the classes (dependencies) that your class needs to perform its functions.

Class creation with a factory

There are some instances where you do not want a shared object. A good example would be when creating new product and persisting that to the database. If you requested an instance of `\Magento\Catalog\Api\Data\ProductInterface`, that object will be shared with all other requests for that instance. As such, calling any setter methods on this object result in that value being change in all other requests for that instance.

If an inject object does not store data in the class (ie. all data is transient and only passes through the class), you likely do not need to load an instance of the class with a factory. If you need to use a class that stores data in the class (for example in the `_data` array), you likely need to use a factory.

Another way of looking at it is instead of calling `new ClassName()` in the code, inject an instance of `ClassNameFactory` into the constructor and call `$classNameFactory->create()` instead.

To see Magento auto-created factories, look in the `/generated` folder. If you need to create a custom factory, feel free to copy one of these as boilerplate, changing the namespace, class name and likely the parameters of the `create` function.

HELPFUL LINKS:

- http://devdocs.magento.com/guides/v2.2/extension-dev-guide/depend-inj.html

Why is it important to have a centralized process creating object instances?

Having a centralized process to create objects makes testing much easier. It also provides a simple interface to substitute objects as well as modify existing ones.

Identify how to use DI configuration files for customizing Magento.

You should be very familiar with `di.xml` and how to use it.

Plugins

Plugins allow you to wrap another class' public functions, add a before method to modify the input arguments, or add an after method to modify the output.

These will be discussed more in the next section.

Example: `vendor/magento/module-catalog/etc/di.xml` (search for "plugins")

Preferences

Preferences are used to substitute entire classes. They can also be used to specify a concrete class for an interface. If you create a service contract for a repository in your `/Api` folder and a concrete class in `/Model`, you can create a preference like:

```
<preference
    for="AcmeWidgets\ProductPromotor\Api\
    PromotionRepositoryInterface"
    type="AcmeWidgets\ProductPromotor\Model\
    PromotionRepository" />
```

Virtual Types

A virtual type allows the developer to create an instance of an existing class that has custom constructor arguments. This is useful in cases where you need a “new” class only because the constructor arguments need to be changed.

This is used frequently in Magento to reduce redundant PHP classes.

HELPFUL LINKS:

- http://devdocs.magento.com/guides/v2.1/extension-dev-guide/build/di-xml-file.html#type-configuration
- https://alanstorm.com/magento_2_object_manager_virtual_types/

Argument Preferences / Constructor Arguments

It is possible to modify what objects are injected into specific classes by targeting the name of the argument to associate it with the new class.

Now, with Magento 2, you can inject your custom class into any other classes constructor in `di.xml`

M1

MAGENTO 1

With Magento 1, if you wanted to make a modification to a class, you rewrote that class, and that change was permanent. All classes that interacted with the modified class had to use that modified class. That was because Magento 1 lacked dependency injection.

Example:

```
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:noNamespaceSchemaLocation="urn:magento:fram
ework:ObjectManager/etc/config.xsd">
    <type name="AcmeWidgets\ProductPromoter\Algorithms\
    Primary">
        <arguments>
            <argument name="basedOn"
            xsi:type="string">sort_order</argument>
        </arguments>
    </type>
</config>
```

HELPFUL LINKS:

- http://devdocs.magento.com/guides/v2.1/extension-dev-guide/build/di-xml-file.html#constructor-arguments

How can you override a native class, inject your class into another object, and use other techniques available in `di.xml` (such as virtualTypes)?

Overriding a native class: use a `<preference/>` entry to specify the existing class name (preceding backslash \ is optional) and the class to be overridden.

Inject your class into another object: use a `<type/>` entry with a `<argument xsi:type="object">\Path\To\Your\Class</argument>` entry in the `<arguments/>` node.

Other uses of `di.xml` are found above.

1.5 DEMONSTRATE ABILITY TO USE PLUGINS

POINTS TO REMEMBER

- There are three types of plugins: around, before and after.
- Plugins only work on `public` methods.
- They do not work on `final` methods, `final` classes.
- They must be configured in `di.xml`.
- **Important:** plugins can be used on interfaces, abstract classes or parent classes. The plugin methods will be called for any implementation of those abstractions.

Demonstrate how to design complex solutions using the plugin's life cycle.

Keeping methods to a small number of lines of code is sometimes challenging. There are those methods that seem to have to do everything.

Magento 2's idea of plugins brings a completely new idea to the table. Every public method can be intercepted, changed, or even circumvented.

There are three types of plugins: **around**, **before**, and **after**. There are some complications with the around plugin, so it is advised to use it sparingly and only when the others will not do. Before plugins modify the input arguments to a method. You can change them to any value. After plugins are used to modify the return value.

Let's say you have a complex saving operation. In this operation, you also need to validate the input data and return an error when there is a problem. You also need to respond with something more than true or false.

You can use an around plugin to validate the incoming request and cancel the save if the result is invalid. Doing this, you are applying the single responsibility principle, making your code easier to understand, debug, and test.

While plugins are often thought of as modifying core functionality, that example demonstrates that they can be useful for a broad range of applications.

How plugins work

When you create a plugin entry, Magento automatically generates a class wrapper for the plugin target. For example, if you want to modify

`\Magento\Catalog\Model\Product`, Magento will auto-generate the `\Magento\Catalog\Model\Product\Interceptor` class. Every function inside the target class will be represented in the auto-generated interceptor class (if you add new functions to a target class, you may need to delete the auto-generated interceptor class from the /generated folder).

Magento then handles locating the plugins and executing them in the Interceptor class: `vendor/magento/framework/Interception/Interceptor.php`

For more information, consult the plugin reference in Magento DevDocs.

Before Plugin

Example from: `\Magento\Catalog\Block\Product\ListProduct`

If you want to modify the input arguments of a method, create a before plugin. To modify the `prepareSortableFieldsByCategory($category)` method, add a method to the plugin class:

```
public function beforePrepareSortableFieldsByCategory(
    \Magento\Catalog\Block\Product\ListProduct $context,
    $category
) {
    // …
    return [$category];
}
```

The method above is run before `\Magento\Catalog\Block\Product\ListProduct::prepareSortableFieldsByCategory`. The return value for

the before plugin determines the arguments going into the next plugin or the final targeted method.

After Plugin

Example from: `\Magento\Catalog\Block\Product\ListProduct`

If you need to modify the output from a public method, use an after plugin. In our example class, let's modify the `getProductPrice($product)`. As such in our plugin class, we would create:

```
public function afterGetProductPrice(
    \Magento\Catalog\Block\Product\ListProduct $context,
    $result,
    \Magento\Catalog\Model\Product $product
) {
    // …
    return $result;
}
```

The after plugin (as of 2.2) includes the input parameters in addition to the return result.

HELPFUL LINKS:

- http://devdocs.magento.com/guides/v2.1/extension-dev-guide/plugins.html

Around Plugin

The around plugin provides full control over the input and output of a function. The original function is passed in as a callback and, by standard, is named `$proceed`. Magento recommends against using these plugins whenever possible. This is because it is easy to accidentally alter major functions in the system by omitting a call to `$proceed()`. It also adds frames to the call stack making debugging more cumbersome.

How do multiple plugins interact, and how can their execution order be controlled?

With every good idea come potential downsides. Controlling how multiple plugins interact would be the problem with plugins. When a plugin is declared, the `sortOrder` attribute can be set. The lower the sort order, the sooner it will be executed in the list. The greater the sort order, the later it will be executed. This allows a degree of control over how one plugin will interact with others.

Additionally, if you need to disable an existing plugin, you can reference it by the name attribute and add the `disabled` attribute.

HELPFUL LINKS:

- http://devdocs.magento.com/guides/v2.1/extension-dev-guide/plugins.html#prioritizing-plugins

How do you debug a plugin if it doesn't work?

First unplug it, then remove the bug.

There are a number of things that can go wrong with plugins. Here are some things to check:

- Is the `di.xml` configuration correct? Are there any syntax errors?
- Is the plugin marked as disabled?
- Do you have the correct class specified in the `<type name="...">` node? Is it the target class?
- Do you have the correct plugin class specified in the `<plugin type="...">` node?
- Is the class or method you are modifying marked as `final`? If so, plugins will not work.
- Does your plugin class have a method to modify a method on the target class?
  - `beforeMethodName` or `afterMethodName` or `aroundMethodName`
  - NOT `methodName`
- Do any of the limitations mentioned in the DevDocs apply?

One technique that has been helpful for us is to set a breakpoint in the method you want to debug. When that breakpoint has been encountered, look at the call stack to see if there are any references to an `Interceptor` class in the recent call stack.

HELPFUL LINKS:

- http://devdocs.magento.com/guides/v2.2/extension-dev-guide/plugins.html

Identify strengths and weaknesses of plugins.

Plugins are very powerful to discreetly modify functionality of existing code. They can also be used to follow the single responsibility principle (SRP) by segregating each piece of functionality to their own areas.

The greatest weakness is exploited in the hands of a developer who is either not experienced or not willing to take the time to evaluate the fallout. For example, used improperly, an `around` plugin can prevent the system from functioning. They can also make understanding what is going on by reading source code hard (spooky action at a distance).

What are the limitations of using plugins for customization?

- Plugins only work on public functions (not protected or private).
- Plugins do not work on final classes or final methods.
- Plugins do not work on static methods.
- Read more: http://devdocs.magento.com/guides/v2.1/extension-dev-guide/plugins.html

In which cases should plugins be avoided?

Plugins are useful to modify the input, output, or execution of an existing method.

Plugins are also best to be avoided in situations where an event observer will work. Events work well when the flow of data does not have to be modified.

1.6 CONFIGURE EVENT OBSERVERS AND SCHEDULED JOBS

POINTS TO REMEMBER

- Event observers listen to events that are triggered within Magento.
- Event observers should not modify the sent data (what plugins are for).

Event observers and scheduled jobs are used to carry out tasks on data. They are an ideal way to extend Magento functionality.

Event observers and scheduled jobs carry a similar characteristic: both do not (should not) modify data as it traverses event observers. A scheduled job makes modifying the flow of data impossible while event observers still do allow it (even though it is against Magento development guidelines).

When an action occurs, an event can be triggered. Event observers listen to these events and act as a notification system. Observers implement `\Magento\Framework\Event\ObserverInterface`.

If you need to modify the data in a method, it is best to use a before or after plugin.

Demonstrate how to configure observers.

To create an event observer, create the file `events.xml` in the `etc` directory. If the event only needs to be listened to in a specific area, create an `events.xml` in that directory.

Create a class that will receive the payload from the event dispatcher. This class must implement `\Magento\Framework\Event\ObserverInterface`.

How do you make your observer only be active on the frontend or backend?

Place it in the `/etc/[area]/events.xml` folder.

Demonstrate how to configure a scheduled job.

To execute a specific action on a schedule, you need to setup the `crontab.xml` file. This file always resides in the `/etc` folder (not in `/etc/[area]`).

The basic syntax looks like:

```
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:noNamespaceSchemaLocation="urn:magento:modu
le:Magento_Cron:etc/crontab.xsd">
    <group id="default">
        <job name="job_name_goes_here"
         instance="AcmeWidgets\ProductPromoter"
         method="execute">
            <schedule>*/10 * * * *</schedule>
        </job>
    </group>
</config>
```

Group

Magento allows you to group cron activity together, making logical groups of functionality. For most scheduled activity, use the `default` group. Magento also does provide `index` group. You can set up configuration options for groups in `cron_groups.xml`.

Job

Configuring the job is simple:

- assign a unique name
- specify the class
- define a method
- set a schedule (using regular crontab schedule notation)

Which parameters are used in configuration, and how can configuration interact with server configuration?

System environment variables can be used to change store configuration information. To change a store configuration value with environment variables:

1. Find the store configuration path found in the database, `etc/adminhtml/system.xml` or the admin panel.
2. Change "`/`" to "`__`" (double underscores) in the path.
3. If this configuration will be set on a global basis (not per store), prepend the path with `CONFIG__DEFAULT__`.
4. If this configuration will be set for a particular website, prepend the path with `CONFIG__WEBSITES__[WEBSITE_CODE]__`

HELPFUL LINKS:

- http://devdocs.magento.com/guides/v2.1/config-guide/prod/config-reference-var-name.html

Identify the function and proper use of automatically available events, for example *`_load_after`, etc.

Magento provides a number of pre-built events. These cover functionality for request routing, data loading, caching, and HTML output.

One of the most common uses for event listeners / observers is with data loading. For your reference, here is a list of all events triggered in `\Magento\Framework\Model\AbstractModel` (the parent class of data that is loaded from the database):

- `$this->_eventPrefix . _load_before`
- `$this->_eventPrefix . _load_after`
- `$this->_eventPrefix . _save_before`
- `$this->_eventPrefix . _save_after`
- `$this->_eventPrefix . _save_commit_after`
- `$this->_eventPrefix . _delete_before`
- `$this->_eventPrefix . _delete_after`
- `$this->_eventPrefix . _delete_commit_after`
- `$this->_eventPrefix . _clear`

`$this->_eventPrefix` is a protected variable that is set in a model (see `vendor/magento/module-catalog/Model/Product.php` for an example).

> M1 **MAGENTO 1**
>
> The case for events is much smaller than it was with Magento 1. Because of the availability of plugins, gaining access to data at the same time is much easier.

When to use events or plugins?

At a fundamental level, the data that is sent to events should not be transformed. Events should be able to be run completely asynchronously:

- You can use the `catalog_product_save_commit_after` to refresh a static file that you have generated. You could use `save_after`—the downside is if an uncaught exception is thrown in the observer, the database transaction will be rolled back and data will not be saved. As long as you have product data, this operation can run at any future date.

- You can send a notification email when an action has occurred (similar to sending an email when an order is placed).

If you want to transform data, use a plugin:

- If you want to transform data, the Magento technical requirements say to use a plugin. The reason for this is that plugins are designed to modify the flow of data into and out of a method. They wrap and control the data flow. Events could be viewed as the spoke of a bicycle. The data is dispatched to the event listeners. There is no expectation of an order of processing and the data could be handled in different orders at different times.
- That said, events are still widely used even for data mutations.
- If you want to modify data before a product is saved, create an `after` plugin for the `beforeSave` method: `afterBeforeSave` (a little hard to read).
- If you need to add some additional data when a customer has been loaded, you can do this as an `after` plugin in the `afterLoad` method: `afterAfterLoad`. Ideally, you would be using extension attributes for this purpose as well.

HELPFUL LINKS:

- https://cyrillschumacher.com/magento2-list-of-all-dispatched-events/
- http://devdocs.magento.com/guides/v2.1/ext-best-practices/extension-coding/observers-bp.html

To dispatch an event, inject an instance of `\Magento\Framework\Event\ManagerInterface` into the constructor. Then call:

```
$this->eventManager->dispatch('event_name_goes_here',
['parameter' => 'array']);
```

1.7 UTILIZE THE CLI

POINTS TO REMEMBER

- The Magento CLI is based on the Symfony Console component.
- It is important to familiarize yourself with these commands.

Describe the usage of `bin/magento` commands in the development cycle.

The Magento CLI is the Magento developer's good friend. It contains many commands to make your life easier.

Hint: create an alias in `.bash_profile` like: `alias mage="php -d memory_limit=1024M bin/magento"`. With this, you can enter a Magento 2 directory and type "`mage setup-upgrade`" instead of typing "`bin/magento setup-upgrade`".

The Magento CLI is based on the Symfony Console component. If you are interested in building CLI commands, reference this document for helpful information. Additionally, Magento DevDocs provides a tutorial on how to add commands.

These next topics will discuss some of the most useful CLI commands in the development process. Note the abbreviations next to some of the commands: this is a feature in Symfony Console.

To see the available options for a command, run the command and add `--help` to the end.

`bin/magento cache:flush` (abbreviated `bin/magento c:f`)

Flushes, or destroys, the cache. It is different than `cache:clean` in that flush actually deletes the keys. The end result is typically the same.

`bin/magento cache:status`

Lists the cache types and their status (enabled / disabled). This is helpful for toggling caches with the `cache:enable` / `cache:disable` command.

`bin/magento deploy:mode:show`

This command shows the current deploy mode: developer, default, or production. When developing Magento, your life will be much easier if you turn the deploy mode to “developer.”

`bin/magento dev:query-log:enable`

Turns on logging of database queries. Can be helpful in tracking down an obscure bug. My preference is to use xdebug and set a breakpoint in `vendor/magento/`

`framework/Model/ResourceModel/Db/AbstractDb.php` as logging can be verbose.

`bin/magento indexer:info` AND `bin/magento indexer:reindex`

These commands show indexer details or initiate a reindex respectively.

`bin/magento module:enable`

After creating the required module files, you need to enable the module in Magento 2 (Magento 1 did this automatically for you). This is done by running the above command with either the `--all` flag (enabling all modules) or with appending the name of the module to enable to the above command.

`bin/magento module:status`

Lists the enabled and disabled modules. If you are having trouble figuring out why a module is not working, this is a good command to put into your troubleshooting repertoire. This has been very helpful to me because it is easy to forget to enable modules.

`bin/magento setup:upgrade`

When the version of a module (in `etc/module.xml`) is incremented, you need to run this command to synchronize those changes to the database. Until you do so, your entire frontend and admin panel will show an error message.

You can run this command with a `--keep-generated` flag to prevent the default behavior of removing all generated assets. You can use this flag in a code deploy situation where your assets have already been compiled for the new version.

`bin/magento setup:db:status`

To check the upgrade status for Magento, use this command. Check the output of this command to determine whether or not you need to upgrade the database (using `setup:upgrade`).

Which commands are available?

To see what commands are available, execute `bin/magento` in your command line. This provides a list of all commands available.

As of Magento 2.2.3 (March 2018), here is the current list of commands for Magento Open Source:

MAGENTO CLI VERSION 2.2.3

Usage:	command [options] [arguments]
Options:	
`-h, --help`	Display this help message
`-q, --quiet`	Do not output any message
`-V, --version`	Display this application version
`--ansi`	Force ANSI output
`--no-ansi`	Disable ANSI output
`-n, --no-interaction`	Do not ask any interactive question
`-v|vv|vvv, --verbose`	Increase the verbosity of messages: 1 for normal output, 2 for more verbose output, and 3 for debug
Available commands:	

help	Displays help for a command
list	Lists commands
admin	
admin:user:create	Creates an administrator
admin:user:unlock	Unlock Admin Account
app	
app:config:dump	Create dump of application
app:config:import	Import data from shared configuration files to appropriate data storage
cache	
cache:clean	Cleans cache type(s)
cache:disable	Disables cache type(s)
cache:enable	Enables cache type(s)
cache:flush	Flushes cache storage used by cache type(s)
cache:status	Checks cache status
catalog	
catalog:images:resize	Creates resized product images
catalog:product:attributes:cleanup	Removes unused product attributes
config	
config:sensitive:set	Sets sensitive configuration
values	
config:set	Change system configuration
config:show	Shows configuration value for given path. If path is not specified, all saved values will be shown.
cron	
cron:install	Generates and installs crontab for current user
cron:remove	Removes tasks from crontab
cron:run	Runs jobs by schedule
customer	
customer:hash:upgrade	Upgrade customer's hash according to the latest algorithm
deploy	
deploy:mode:set	Set application mode
deploy:mode:show	Displays current application mode

dev	
dev:di:info	Provides information on Dependency Injection configuration for the Command.
dev:query-log:disable	Disable DB query logging
dev:query-log:enable	Enable DB query logging
dev:source-theme:deploy	Collects and publishes source files for theme
dev:template-hints:disable	Disable frontend template hints. A cache flush might be required.
dev:template-hints:enable	Enable frontend template hints. A cache flush might be required.
dev:tests:run	Runs tests
dev:urn-catalog:generate	Generates the catalog of URNs to *.xsd mappings for the IDE to highlight xml.
dev:xml:convert	Converts XML file using XSL style sheets
i18n	
i18n:collect-phrases	Discovers phrases in the codebase
i18n:pack	Saves language package
i18n:uninstall	Uninstalls language packages
indexer	
indexer:info	Shows allowed Indexers
indexer:reindex	Reindexes Data
indexer:reset	Resets indexer status to invalid
indexer:set-mode	Sets index mode type
indexer:show-mode	Shows index mode
indexer:status	Shows status of Indexer
info	
info:adminuri	Displays the Magento Admin URI
info:backups:list	Prints list of available backup files
info:currency:list	Displays the list of available currencies
info:dependencies:show-framework	Shows number of dependencies on Magento framework
info:dependencies:show-modules	Shows number of dependencies between modules
info:dependencies:show-modules-circular	Shows number of circular dependencies between modules
info:language:list	Displays the list of available language locales
info:timezone:list	Displays the list of available time zones

maintenance	
maintenance:allow-ips	Sets maintenance mode exempt IPs
maintenance:disable	Disables maintenance mode
maintenance:enable	Enables maintenance mode
maintenance:status	Displays maintenance mode status
module	
module:disable	Disables specified modules
module:enable	Enables specified modules
module:status	Displays status of modules
module:uninstall	Uninstalls modules installed by composer
sampledata	
sampledata:deploy	Deploy sample data modules
sampledata:remove	Removes all sample data packages from composer.json
sampledata:reset	Resets all sample data modules for re-installation
setup	
setup:backup	Takes backup of Magento Application code base, media, and database
setup:config:set	Creates or modifies the deployment configuration
setup:cron:run	Runs cron job scheduled for setup application
setup:db-data:upgrade	Installs and upgrades data in the DB
setup:db-schema:upgrade	Installs and upgrades the DB schema
setup:db:status	Checks if DB schema or data requires upgrade
setup:di:compile	Generates DI configuration and all missing classes that can be auto-generated
setup:install	Installs the Magento application
setup:performance:generate-fixtures	Generates fixtures
setup:rollback	Rolls back Magento Application codebase, media, and database
setup:static-content:deploy	Deploys static view files
setup:store-config:set	Installs the store configuration. Deprecated since 2.2.0. Use config:set instead

`setup:uninstall`	Uninstalls the Magento application
`setup:upgrade`	Upgrades the Magento application, DB data, and schema
`store`	
`store:list`	Displays the list of stores
`store:website:list`	Displays the list of websites
`theme`	
`theme:uninstall`	Uninstalls theme
`varnish`	
`varnish:vcl:generate`	Generates Varnish VCL and echoes it to the command line

How are commands used in the development cycle?

CLI commands provide a secure entry point for conducting operations that could be insecure to run in the Magento admin panel. SSH access should be a secure way to vet a user's authorization status.

Magento provides commands to run tests (`dev:tests:run` although using PHPUnit in PHPStorm is faster and provides xdebug capabilities), compile dependency injection (`setup:di:compile`), and deploy static assets (`setup:static-content:deploy`).

Enabling modules and running setup scripts has to be done using the command line during development. It is often quicker to toggle caching of sections or to flush the cache during development using the command line compared to using the admin interface.

Demonstrate an ability to create a deployment process.

Deploying Magento 2 involves several steps.

Publisher's note: we would like to eventually publish some articles on a build > deploy system that we use at SWIFTotter. Look on our website to see these articles as we are able to release them.

The basic steps for deploying Magento 2 code:

- Enable maintenance mode
- Copy files to the deploy target
- Enable modules / apply deploy configuration
- Run dependency compilation
- Build static assets
- Upgrade the database
- Disable maintenance mode

As you can see, this process involves downtime for production. For many companies, up to 30 minutes of downtime each deploy is not acceptable. Magento has been actively working to mitigate this. In addition, several community systems are available. Look in the further reading section below.

HELPFUL LINKS:

- https://github.com/SwiftOtter/MagentoCI (look at the deploy folder)
- http://devdocs.magento.com/community/resources/#installdeploy
- https://github.com/davidalger/capistrano-magento2

How does the application behave in different deployment modes, and how do these behaviors impact the deployment approach for PHP code, frontend assets, etc.?

Default (not ideal for production or development, but as a hybrid)

- Symlinks are established in the `pub/static` folder. These are linked from files in the `app/code` or `app/design` folders.
- Exceptions are not displayed to the user (making development very difficult). They are logged in `var/log`.
- Static files are generated on the fly and are symlinked into the `var/view_preprocessed` folder.

Developer mode (security risk if used in production)

- Symlinks are established in the `pub/static` folder. Use `dev:static-content:deploy` to refresh these links or simply remove the stale link manually (much faster).
- Errors are shown to the user and logging is verbose. Caution: debug logging is disabled by default even in developer mode.
- Magento automatically builds code for plugins (interceptors), factories, etc. as it does in the other modes.
- Slow performance.

HELPFUL LINKS:

- http://devdocs.magento.com/guides/v2.1/config-guide/bootstrap/magento-modes.html

Production mode (difficult to use for development)

- Static files must be pre-compiled as no compilation will happen on the fly.
- Errors are only logged.

1.8 DEMONSTRATE THE ABILITY TO MANAGE THE CACHE

POINTS TO REMEMBER

- Understanding the `config`, `layout`, `block_html`, `config_webservice` and `full_page` caches and their functions is important.
- You can disable full page caching on a particular page by marking a block as `cacheable="false"`

Describe cache types and the tools used to manage caches.

Magento includes multiple types of caching to speed retrieval of CPU-consuming calculations and operations. To see a list of all cache types, run `bin/magento cache:status`.

Magento 2 includes two means of caching: server caching, Varnish caching and browser caching.

Cache configuration is stored in `/etc/cache.xml`. Here is a list of all the `cache.xml` files in Magento 2.2:

- `module-eav/etc/cache.xml`
- `module-translation/etc/cache.xml`

- `module-customer/etc/cache.xml`
- `module-webapi/etc/cache.xml`
- `module-page-cache/etc/cache.xml`
- `module-store/etc/cache.xml`
- `module-integration/etc/cache.xml`

You can clear a specific cache by using the `bin/magento cache:flush` command. For example, the following clears the `config` and `layout` caches: `bin/magento cache:flush config layout`. Here is a list of some of the more important caches:

HELPFUL LINKS:

- http://devdocs.magento.com/guides/v2.1/frontend-dev-guide/cache_for_frontdevs.html

`config`: Magento Configuration

The `config` cache stores configuration from the XML files along with entries in the `core_config_data` table. This cache needs to be refreshed when you add system configuration entries (`/etc/adminhtml/system.xml`) and make XML configuration modifications.

`layout`: Layout XML Updates

With Magento's extensive layout configuration, a lot of CPU cycles are used in combining and building these rules. This cache needs to be refreshed when making changes to files in the `app/design` and the `app/code/`

`AcmeWidgets/ProductPromoter/view/[area]/layout` folders. For frontend development, we usually disable this cache. This cache type is also used to cache `ui_component` XML.

`block_html`: Output from the `toHtml` method on a block

Obtaining the HTML from a block can also be expensive. Caching at this level allows some of this HTML output to be reused in other locations or pages in the system. For frontend development, we usually disable this cache.

`collections`: Multi-row results from database queries

This cache stores results from database queries.

`db_ddl`: Database table structure

See this file: `vendor/magento/framework/DB/Adapter/Pdo/Mysql.php`

`config_webservice`:

This stores the configuration for the REST and SOAP APIs. When adding methods to the API service contracts, you will need to flush this one frequently.

`full_page`: Full page cache (FPC)

This HTML page output can be stored on the file system (default), database or Redis (fastest). When doing any type of frontend development, it is best to leave the FPC

off. Before deploying new frontend updates, though, it is important turn it back on and ensure that the updates do not cause problems with the cache.

How do you add dynamic content to pages served from the full page cache?

In Magento 2 FPC, pages are either cached or they are not. To make a page not cacheable, you can add the `cacheable="false"` attribute to any block on the page. Keep in mind that this affects website performance and should be used sparingly.

Combining speed and personalization involves serving a cached page and then substituting or adding content with an AJAX request.

The DevDocs contains an article on how to do this with UI Components. This is considered the only “proper” way of hole punching as Magento leverages the browser’s local cache storage system.

Describe how to operate with cache clearing.

When working with the cache, assume that the requested cache entry is not present. If it is not present, run your CPU-intensive operation, and save the results into the cache. Subsequent visits to the cache should be populated and save precious computation time.

How would you clean the cache?

`bin/magento cache:clean` OR `bin/magento` cache/flush

HELPFUL LINKS:

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

In which case would you refresh cache/flush cache storage?

Magento recommends running cache cleaning (`cache:clean`) operations first as this does not affect other applications that might use the same cache storage. If this does not solve the problem, they recommend flushing the cache storage (`cache:flush`).

In reality, if the file system cache storage is used, you should never have multiple applications' cache storage combined. Sessions and content caching should never share a database in Redis. Ideally, they are stored in altogether separate Redis instances. As such, flushing the cache should not have any consequences.

Describe how to clear the cache programmatically.

You can clear the cache programmatically by calling the `\Magento\Framework\App\CacheInterface::remove()` method.

What mechanisms are available for clearing all or part of the cache?

`clean_cache_by_tags` event

You can dispatch a `clean_cache_by_tags` event with an `object` parameter of the object you want to clear from the cache.

From: `\Magento\PageCache\Observer\FlushCacheByTags::execute`

`\Magento\Framework\App\CacheInterface->clean()`

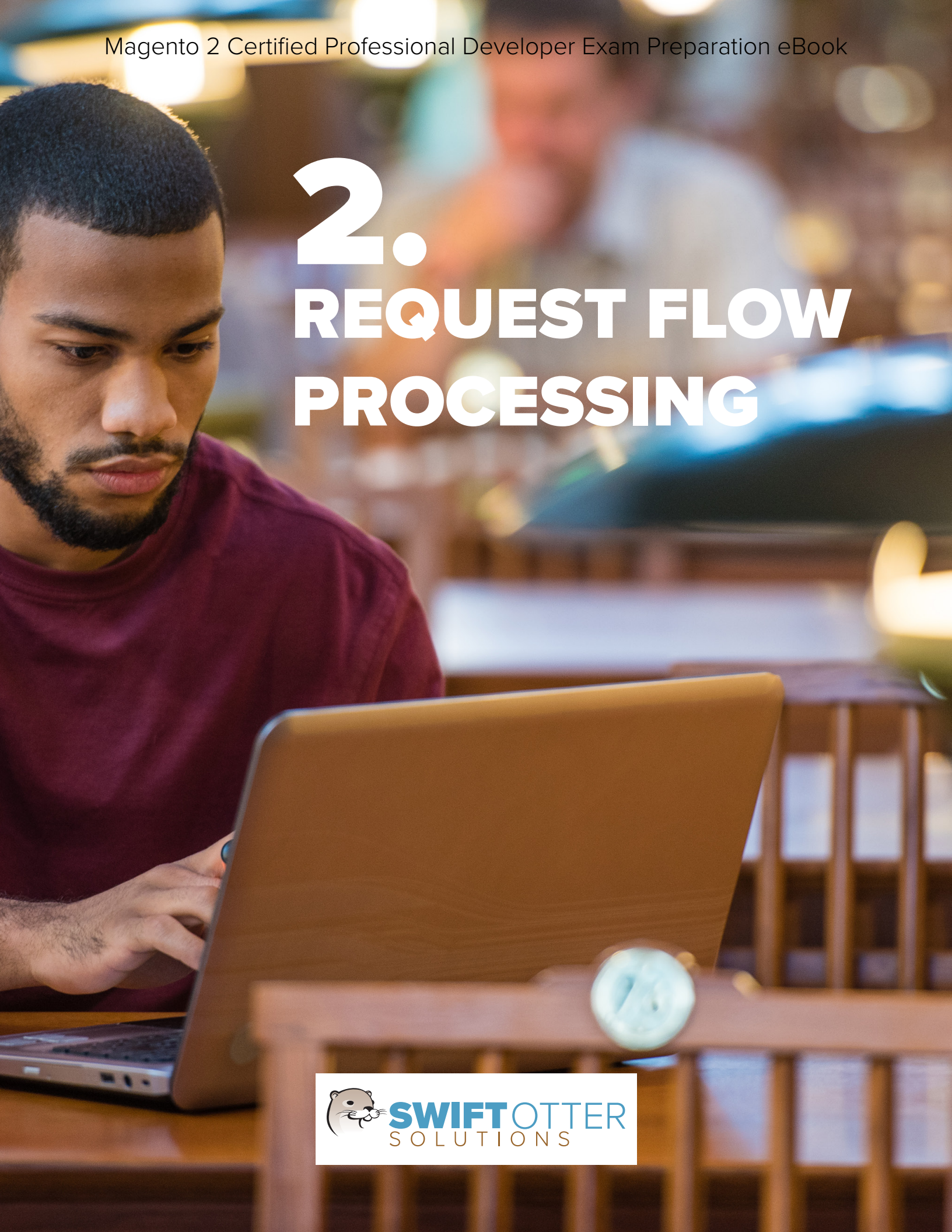
As seen above in “Describe how to clear the cache programmatically.”

Magento CLI console

`bin/magento` cache/flush or `bin/magento cache:clean`

Manually

You can `rm -rf var/cache/` or use the `redis-cli`, select a database index, and run `flushdb`.



2. REQUEST FLOW PROCESSING

SWIFT OTTER SOLUTIONS

2.1 UTILIZE MODES AND APPLICATION INITIALIZATION

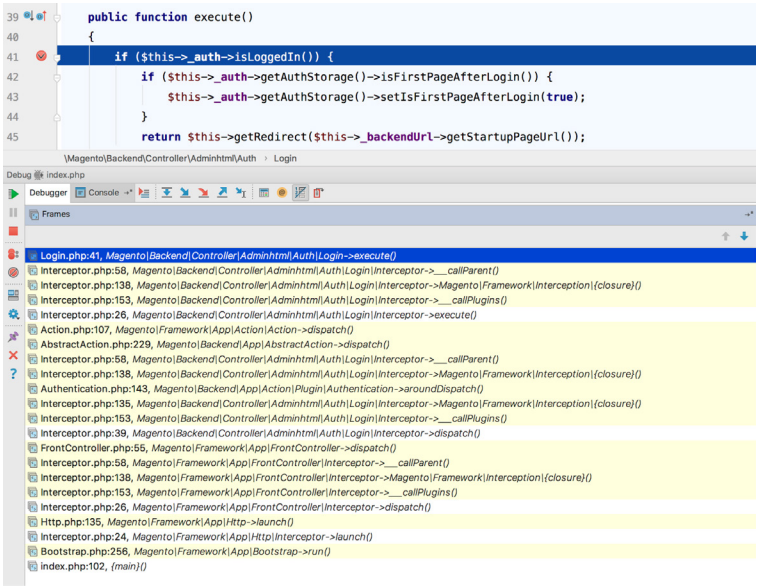
POINTS TO REMEMBER

- The recommended Magento entry point is `pub/index.php`
- The `default` deploy mode is the default for Magento's deploy mode feature.

Identify the steps for application initialization.

To see for yourself the path of execution, open `vendor/magento/module-backend/Controller/Adminhtml/Auth/Login.php` and set a breakpoint in the `execute()` method. Clear cookies in your development website and navigate to the admin panel.

Look at the call-stack for the `execute()` method:



- The recommended application entry point is `pub/index.php`. Nginx or Apache should use `/pub` as the website's primary directory.
- `pub/index.php`
  - A bootstrap instance is initialized (which creates the object manager).
  - An HTTP (`\Magento\Framework\App\Http`) application is created. See concrete classes that implement `\Magento\Framework\AppInterface` to find other application types.
  - The application is run.
- `vendor/magento/framework/App/Bootstrap.php::run`
  - Checks are completed (is installed, is not in maintenance mode).
  - Application is launched.
- `vendor/magento/framework/App/Http.php::launch`
  - Area code (`frontend`, `adminhtml`, etc.) is determined.
  - Object manager is configured for that area.
  - Front controller is created. This is based on the area.
    - Http: `vendor/magento/framework/App/Http.php`
    - Rest: `vendor/magento/module-webapi/Controller/Rest.php`
    - Soap: `vendor/magento/module-webapi/Controller/Soap.php`
  - Front controller is tasked with figuring out where to direct the request.
- `vendor/magento/framework/App/FrontController.php`
  - The list of routers (`\Magento\Framework\App\RouterListInterface`) is traversed.

    - Each router (`\Magento\Framework\App\RouterInterface`) is asked if it can match the route.
    - If it can, a `\Magento\Framework\App\ActionInterface` is returned. This action is the controller that will be executed. Controllers must implement this interface.
    - The `execute` method is run on the controller action.
    - The `Result` from this is returned to the FrontController.
    - The `Result` is rendered and set on the `Response`
    - The `Response` is output
- `vendor/magento/framework/App/Http.php::launch`

How would you design a customization that should act on every request and capture output data regardless of the controller?

This is an excellent use-case for events. Create an event observer for the `controller_action_postdispatch` event.

Describe how to use Magento modes.

See answer in Chapter 1 for: “How does the application behave in different deployment modes, and how do these behaviors impact the deployment approach for PHP code, frontend assets, etc.?”

What are pros and cons of using developer mode/production mode? When do you use default mode?

See answer in Chapter 1 for: “How does the application behave in different deployment modes, and how do these behaviors impact the deployment approach for PHP code, frontend assets, etc.?”

Default mode is enabled out of the box. It is a hybrid of both production and development modes designed to be secure but also allow for some development capabilities.

How do you enable/disable maintenance mode?

```
bin/magento maintenance:enable
```

```
bin/magento maintenance:disable
```

Describe front controller responsibilities.

The front controller (implementing `\Magento\Framework\App\FrontControllerInterface`) is responsible for running business logic and returning the result of that. It could be as simple as returning an HTML layout (`/vendor/magento/module-backend/Controller/Adminhtml/System/Index.php`). It can also handle more complex processing, such as loading an object for editing (`/vendor/magento/module-catalog/Controller/Adminhtml/Product/Edit.php`).

In which situations will the front controller be involved in execution, and how can it be used in the scope of customizations?

The FrontController is involved in every non-API request. It is the entry point for the request flow processing. It transforms a request into a Result (not necessarily HTML).

The front controller is used for transforming the input of a url (and parameters) into an HTML response. It is not used in the API or the console.

2.2 DEMONSTRATE ABILITY TO PROCESS URLS IN MAGENTO

POINTS TO REMEMBER

- Many Magento urls consist of three segments:
  - Front name
  - Path to the action
  - Name of the action
- Magento makes it easy to add a new routing system.

Describe how Magento processes a given URL. How do you identify which module and controller corresponds to a given URL?

Magento determines the area based on the front name (`\Magento\Framework\App\AreaList::getCodeByFrontName`). If no front name matches, the default frontend area is used.

If the request is not for the API, Magento parses the url. `\Magento\Framework\App\Router\Base::parseRequest` handles this operation. The path (the segment after the domain) is exploded with the slash as a delimiter.

Example URL: https://storeurl.com/catalog/product/view/id/15

- The first parameter is the module's front name. This is configured in `etc/[area]/routes.xml`. In this case, the router in `vendor/magento/module-catalog/etc/frontend/routes.xml` matches the `catalog` front name. The base router will look for a controller in the `vendor/magento/module-catalog/Controller` folder.
- The second parameter is the path to the folder that contains the action. Slashes on the file system are exchanged with underscores in the url. Magento chooses the `Product/` inside the folder listed in the previous point.
- The third parameter is the action. Magento finds the `View.php` here and will run the `execute()` action.

What is necessary to create a custom URL structure?

- Create a new router (example: `vendor/magento/module-cms/Controller/Router.php`) which implements the `\Magento\Framework\App\RouterInterface` interface. It is easiest to extend an existing router.
- Register the router (example: `vendor/magento/module-cms/etc/frontend/di.xml`)
- Return an instance of an `\Magento\Framework\App\ActionInterface`

- In most cases a custom router will forward the request to the base router by mutating the request (like the CMS router or the default routers do), instead of directly returning an Action instance.

Describe the URL rewrite process and its role in creating user-friendly URLs.

URL rewrites are a pretty face to an ugly url. The url `catalog/product/view/id/1234` is neither search engine nor user friendly. As such, Magento uses the `url_rewrite` database table to provide a means to map from a pretty url to a system url.

For example with the above url, say “super-blue-widget” has the ID `1234` in the database. In the `url_rewrite` table, you will find a row that has the following data:

`request_path`: “`super-blue-widget`”

`target_path`: “`catalog/product/view/id/1234`”

Magento looks for a match with the `request_path` and redirects the *internal* router to the target path.

The URL rewrite module contains a router which checks to see if the given route can be matched in the `url_rewrite` table. If it can, the router redirects to the route that is given in the table.

This functionality is found in the `module-catalog-url-rewrite` and the `module-url-rewrite` modules.

The URL Rewrite module also contains a router (`\Magento\UrlRewrite\Controller\Router`) that handles the conversion of the pretty url into something that Magento better understands.

How are user-friendly URLs established, and how are they customized?

The user-friendly urls come from the `url_key` attributes. These attributes apply to the product and category entities.

For every category that the product is assigned to, Magento generates user-friendly urls by creating a path based on the category tree for that category. The product's url identifier is appended to this.

These values are then stored in the `url_rewrite` table.

Describe how action controllers and results function.

Action controllers implement the `\Magento\Framework\App\ActionInterface` interface. Usually, they extend the `\Magento\Framework\App\Action\Action` class.

M1

MAGENTO 1

Every action controller file only handles one action. This marks a difference from Magento 1 where an action controller file could handle many actions (the method name used the action url followed by "Action").

The `execute` method must return one of the following types:

- `\Magento\Framework\Controller\ResultInterface`: this one assists in rendering HTML, JSON, or any other type of output.
- `\Magento\Framework\App\ResponseInterface`: this is for returning raw output. This is deprecated and should only be used when working with legacy code.

How do controllers interact with another?

- A controller can forward to another route (returning `\Magento\Framework\Controller\Result\Forward`). The user sees no change in URL, but a new controller takes over processing the URL. This restarts the router matching loop with the new arguments.
- A controller can redirect to another URL address (`\Magento\Framework\Controller\Result\Redirect`) this returns a `\Magento\Framework\App\ResponseInterface`. It returns a 30x HTTP code with the URL for the browser to redirect to.

How are different response types generated?

Types of results with default Magento 2 (must implement `\Magento\Framework\Controller\ResultInterface`):

- `vendor/magento/framework/Controller/Result/Json.php`
- `vendor/magento/framework/Controller/Result/Forward.php`
- `vendor/magento/framework/Controller/Result/Raw.php`
- `vendor/magento/framework/View/Result/Page.php`
- `vendor/magento/framework/Controller/Result/Redirect.php`

Inject the auto-generated Factory class into your controller. Create an instance of the class, set the values needed, and return it.

2.3 DEMONSTRATE ABILITY TO CUSTOMIZE REQUEST ROUTING

Describe request routing and flow in Magento. When is it necessary to create a new router or to customize existing routers?

Many times, business requirements accommodate Magento's preferred URL structure. This is especially true of AJAX or adminhtml requests.

If you have a small amount of customization necessary, the URL rewrite functionality can be a consideration.

Beyond that, any custom URL structure will need a custom router. Magento 2 makes it very easy to configure a custom router for your application.

How do you handle custom 404 pages?

The default Magento 404 page is a CMS page in the database. Modifications can easily be made to the CMS page.

If custom functionality is required around the generation of the 404 page, inject a `NoRouteHandler` into the `\Magento\Framework\App\Router\NoRouteHandlerList`. Then forward to the desired action that processes the not found request.

2.4 DETERMINE THE LAYOUT INITIALIZATION PROCESS

POINTS TO REMEMBER

- Magento locates all layout XML files and merges them into one large XML document that is stored in the `layout` cache.
- These are loaded based on the sequence element in `etc/module.xml`.

Determine how layout is compiled.

There are three types of layout instructions: page layout (define the structure for a page), page configuration (the details/meta for a particular page) and the generic layout (declare blocks for a particular page). In most cases, little needs to be done with the page layout or the page configuration and most of the work happens with the generic layout instructions. See this page for more details.

Magento generic layout instructions are found in the `app/code/AcmeWidgets/ProductPromoter/web/[area]/layout` or the `app/design/[area]/Package/Theme/[Module_Name]/layout` directory.

The layout XML files are merged into a large XML document and saved to the cache. See `vendor/magento/framework/View/Model/Layout/Merge.php` for how this merging process works.

How would you debug your `layout.xml` files and verify that the right layout instructions are used?

Here are a few suggestions on debugging layout XML instructions (all of these require an IDE such as PHPStorm and Xdebug):

- Determine that your layout XML file is being processed. `\Magento\Framework\View\Model\Layout\Merge::_loadFileLayoutUpdatesXml`. Set a breakpoint in the `foreach` loop and add a condition to the breakpoint to watch for the following condition: `stripos($file->getFilename(), "[YOUR FILENAME]") > -1`
- Verify that the XML in your layout file is accurate (use an online XML validator). This includes using the proper tag and attribute names.
- Verify that no errors are being displayed on the frontend (in Developer mode).
- Determine the handles that are being applied to the page and check if the desired declarations are within one of them.
- Check that the XML file name is correct (check the class attribute in the `<body/>` tag for most of the handles).
- If you are adding a block instruction, verify that the class exists and is being autoloaded. Do this by setting a breakpoint on the class declaration.
- Verify that the HTML output is being processed correctly. In the block you are utilizing, temporarily create a `_toHtml` function that returns the value from the `parent::_ toHtml()` method. Use your debugger to step into the return `$this->fetchView($this->getTemplateFile());` method call.

M1

MAGENTO 1

This method is no longer final so you can create a plugin for this. Modifying block HTML is much easier now.

Determine how HTML output is rendered. How does Magento flush output, and what mechanisms exist to access and customize output?

Block renderer:

`\Magento\Framework\View\Element\AbstractBlock::toHtml()`

The block that renders HTML from a template is: `\Magento\Framework\View\TemplateEngine\Php::render`.

> M1 **MAGENTO 1**
>
> Block template rendering has changed. A `$block` variable is exposed to your template. A PHTML template can only access public functions on the block class.

Magento uses the PHP output buffering `functions` to render the HTML. This allows Magento to use the `include` construct to directly render the HTML. Once complete rendering is complete, Magento flushes the output buffer which collects everything that was rendered and returns it as a variable.

Determine module layout XML schema. How do you add new elements to the pages introduced by a given module?

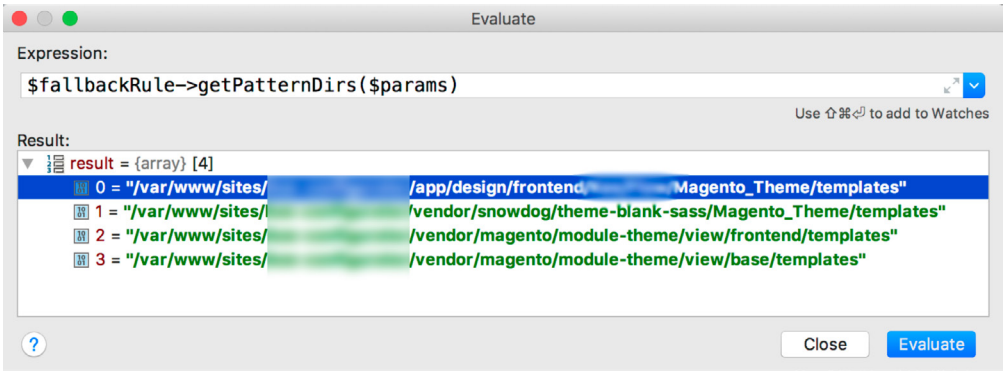
- Locate the XML layout handle for the page to be updated. The layout handle in the `<body/>` tag on that page can be extrapolated from the `class` attribute. There any many more layout handles that are not listed in `<body>`. Use `\Magento\Framework\View\Layout\ProcessorInterface::getHandles()` to see which are used.

- You can also find the layout handle by setting a breakpoint `vendor/magento/framework/App/Action/Action.php` in the dispatch method. When that breakpoint is reached, use the immediate console window to load the value for `$request->getFullActionName()`.

- Use the `referenceBlock` or `referenceContainer` node to target the block in which to place the new node. Note that you could also place this node inside the root `<body/>` node.

Demonstrate the ability to use layout fallback for customizations and debugging.

When trying to determine why a particular PHTML file is displayed (or not displayed), look at `vendor/magento/framework/View/Design/FileResolution/Fallback/Resolver/Simple.php::resolveFile`. In your immediate evaluator, execute the array provider: `$fallbackRule->getPatternDirs($params)`. Here is an example of the output:



The above gives a quick way to determine which paths are being looked at for fallbacks.

How do you identify which exact `layout.xml` file is processed in a given scope?

- Determine which area is loaded (`frontend` or `adminhtml`).
- Locate the layout handle.
- Search through the project to find all XML files with that name.

How does Magento treat layout XML files with the same names in different modules?

The XML configuration is merged together. Magento uses the module's `sequence` node to determine the order of the files to load.

Identify the differences between admin and frontend scopes. What differences exist for layout initialization for the admin scope?

The primary difference is that the admin scope uses the Magento ACL to determine whether or not it should be displayed.

HELPFUL FILES:

- `vendor/magento/module-backend/Block/AbstractBlock.php`

2.5 DETERMINE THE STRUCTURE OF BLOCK TEMPLATES

POINTS TO REMEMBER

- Root template is the basic building block for Magento HTML output.
- New page structures are stored in `view/frontend/page_layout`
- Identify and understand root templates, **`empty.xml`**, and **`page_layout`**.

Page structures are defined by the `layout` in the `page` layout nodes. An example can be seen in the `catalog_product_view.xml` layout handle.

Root template

The root template is the most basic piece and common conglomeration of HTML that is possible in Magento. It is where the result of the layout instructions are put.

Magento only has one root template: `vendor/magento/module-theme/view/base/templates/root.phtml`.

This file is injected into the controller's `Page` result (`vendor/magento/module-theme/view/base/templates/root.phtml`) with dependency injection:

- `vendor/magento/module-backend/etc/adminhtml/di.xml`
- `vendor/magento/magento2-base/app/etc/di.xml`

`empty.xml`

This is the parent page layout XML file. All page layout files derive and extend this.

`page_layout` (found in `app/code/AcmeWidgets/ProductPromoter/view/frontend/page_layout.xml`)

Files in these directories represent different page layout ideas. For example, Magento comes with a 1 column layout, a 2 column (right or left) layout, and a 3 column layout (found in `vendor/magento/module-theme/view/frontend/page_layout`).

You can create new layout types by simply adding a new file in a `view/frontend/page_layout` directory.

How are page structures defined, including number of columns, which basic containers are present, etc.?

To specify a layout for a page, wrap the XML file in a `<page/>` node and set the `layout=""`.

By adding a container to the `columns` container. Containers are powerful in that you can specify the HTML tag name and associate classes.

Examples:

- `vendor/magento/module-theme/view/frontend/page_layout/1column.xml`
- `vendor/magento/module-theme/view/frontend/page_layout/3columns.xml`

Describe the role of blocks and templates in the request flow. In which situations would you create a new block or a new template?

A view model can be used to provide data to a template (`.phtml` file). In most cases, creating a custom block type is no longer necessary. The block includes the template, and shares the view model for data retrieval.

You do not need to specify a block's type in layout XML anymore. The default `\Magento\Framework\View\Element\Template` is specified in the absence of the `type` attribute. Instead, specifying the `view_model` argument is the way that Magento now recommends.

Two cases where creating a block is necessary would be if the template is determined at runtime (cart item renderers) or non-default block caching needs to be used.

Templates are created to display information. For module development, templates and blocks represent an almost 1:1 ratio. For theme development, many more templates will be created or overridden than blocks.

3. CUSTOMIZING THE MAGENTO UI

SWIFT OTTER SOLUTIONS

3.1 DEMONSTRATE ABILITY TO UTILIZE THEMES AND THE TEMPLATE STRUCTURE

POINTS TO REMEMBER

- Themes are stored in `app/design/[area]`
- `theme.xml` and `registration.php` are the only two required files.
- Overriding existing Magento or 3rd-party module template files is as simple as replicating the directory structure in your theme.

Demonstrate the ability to customize the Magento UI using themes. When would you create a new theme?

Themes encapsulate design changes. You can extend and customize an available theme or even create one from scratch, although that is not recommended. Creating a theme usually involves copying and modifying any necessary templates, creating XML files to adjust layouts, and creating the necessary styles to make the frontend (or adminhtml area) match the design specifications.

Most merchants will have a custom theme installed. This theme could be a purchased theme or built specifically for the website.

Publisher's note: we have used the Snowdog Blank SASS theme. It is a straight transformation of LESS to SASS and thus inherits the problems of the Magento themes. However, its compatibility with Webpack, Gulp, and Babel are worth the switch.

Note: the certification exam only refers to the Magento/blank and Magnto/luma LESS based themes.

Theme directory structure

Custom themes are located in the `app/design/frontend` directory. They continue the notation of `ThemePackage_ThemeName`.

Inside this directory, the only required files are `registration.php` and `theme.xml` (`etc/view.xml` is required if there is no parent theme). The registration file is identical to a module's (in `app/code/MyCompany/MyModule`) `registration.php`. `theme.xml` specifies the parent (fallback) theme, the theme name, and a preview image.

HELPFUL LINKS:

- http://devdocs.magento.com/guides/v2.1/frontend-dev-guide/themes/theme-structure.html

How do you define theme hierarchy for your project?

Specify the parent theme in the `parent` node of `theme.xml`.

Demonstrate the ability to customize/debug templates using the template fallback process. How do you identify which exact theme file is used in different situations?

The first step in customizing a template is to locate it. This is done by either finding the path in layout XML, using the debugger, or by enabling template hints (`bin/magento dev:template-hints:enable`). Another way is to find a unique string in the HTML (translation, tag or class). Searching the entire project directory

often yields the location of the file. Be aware that there are not only `.phtml` templates, but also `.html` knockout.js templates or `.xhtml` template files.

Once you find the file, create a folder in your theme for the module that you copied it from. For example, if you are modifying a template from the `Magento_Catalog` module, create a `Magento_Catalog` folder inside your theme directory.

Recreate the path (including the `templates` directory) to and including the filename inside this directory that was just created.

If you are extending an existing theme, check that theme to see if the file exists there. If so, copy it from there.

How can you override native files?

Example: recently, we have been working on a project that made some extensive modifications to the customer account area. In this case, we need to modify some details shown in the Info template (account summary and newsletter subscription). Our theme is `SwiftOtter_Flow` and is found in `app/design/frontend/SwiftOtter_Flow/`.

- Locate the existing template path: `vendor/magento/module-customer/view/frontend/templates/account/dashboard/info.phtml`
- Because this file is coming from the `Magento_Customer` module, create the `Magento_Customer` folder: `app/design/frontend/SwiftOtter_Flow/Magento_Customer`
- Recreate the folder structure after `view/frontend/` in the existing template path: `app/design/frontend/SwiftOtter_Flow/Magento_Customer/templates/account/dashboard`

- Copy `info.phtml` from the existing template path into the new folder that was created.

3.2 DETERMINE HOW TO USE BLOCKS

POINTS TO REMEMBER

- Blocks can be referenced with the `$block` variable into a template.
- As such, only public methods can be accessed.

Demonstrate an understanding of block architecture and its use in development. Which objects are accessible from the block? What is the typical block's role?

Blocks are PHP classes that provide information to the `.phtml` templates. They are the lifeblood of normally static `.phtml` files. The block allows for a separation of duties from the template file: blocks handle data access and retrieval (dependency injection works here), whereas the templates display information.

IMPORTANT

It is a major code smell when the `ObjectManager` is found in a template file. This means that the effort was not extended (or knowledge imparted) to create a block to locate the information. Using `$this->helper()` is almost as bad.

By default, template blocks are injected with a `\Magento\Framework\View\Element\Template\Context` object. This object contains a number of other objects that are loaded into the instance:

- `$this->validator: \Magento\Framework\View\Element\Template\File\Validator`
- `$this->resolver: \Magento\Framework\View\Element\Template\File\Resolver`
  - Resolves template file paths
- `$this->_filesystem: \Magento\Framework\Filesystem`
- `$this->templateEnginePool: \Magento\Framework\View\TemplateEnginePool`
- `$this->_storeManager: \Magento\Store\Model\StoreManagerInterface`
- `$this->_appState: \Magento\Framework\App\State`
- `$this->pageConfig: \Magento\Framework\View\Page\Config`

Identify the stages in the lifecycle of a block. In what cases would you put your code in the `_prepareLayout()`, `_beforeToHtml()`, and `_toHtml()` methods?

The abstract block is found in `vendor/magento/framework/View/Element/AbstractBlock.php`.

Stages:

- Block is created (`__construct`).

- The `setLayout` method is called, assigning `_layout` to be the current layout object.
- When Magento is ready to obtain the HTML:
    - `toHtml` is called
        - The `view_block_abstract_to_html_before` event is triggered.
        - If a cache entry is present, the template will not be loaded.
        - `_beforeToHtml`
        - `_toHtml`
        - `_afterToHtml` (regardless of whether or not the cached value was used)
        - The `view_block_abstract_to_html_after` event is triggered
    - HTML is returned

Remember that plugins can only modify the flow of data from public methods. Blocks do not expose many public methods and so the best choice for globally modifying data is often a plugin for `toHtml`.

How would you use events fired in the abstract block?

The original idea was to use events to modify the flow of data. However, with the rule in the Magento Technical Guidelines, plugins should be used for modifying data.

As such, the use cases for events triggered from blocks is limited.

Describe how blocks are rendered and cached.

As discussed above, the `toHtml` is responsible for rendering the block as HTML. When using a template, this involves loading a template. If a cache entry exists, that is returned. Otherwise, the `toHtml` method passes off rendering to the `_toHtml` method for the actual rendering.

Caching

A block can be cached when:

- The cache lifetime is not `null`.
- The `block_html` cache type is enabled.

By default, the cache lifetime is not set, so this needs to be specified to enable caching the block.

If caching can happen, a cache key is generated (`$this->getCacheKey()`). The rendered HTML is then searched for the presence of SID parameters (session IDs), which are then replaced with a placeholder before the string is saved.

HELPFUL FILES:

- `vendor/magento/framework/View/Element/AbstractBlock.php`

Identify the uses of different types of blocks. When would you use non-template block types?

- `Template` (`vendor/magento/framework/View/Element/Template.php`): renders HTML to the user.

- `Text` (`vendor/magento/framework/View/Element/Text.php`): prints a text string.
- `ListText` (`vendor/magento/framework/View/Element/Text/ListText.php`): similar to a container as it returns the output of each of the child blocks.

With Magento's layout containers, the use cases for non-template blocks is quite few. The Text block could be used to show a text string that is set in the layout XML.

In what situation should you use a template block or other block types?

Templates are used to display HTML. There are many blocks that extend the basic template-type block. You can find a number of them in the `vendor/magento/framework/View/Element` directory.

3.3 DEMONSTRATE ABILITY TO USE LAYOUT AND XML SCHEMA

POINTS TO REMEMBER

- Familiarize yourself with the XML directives.

Describe the elements of the Magento layout XML schema, including the major XML directives. How do you use layout XML directives in your customizations?

Layout XML is the powerful link between blocks and templates. This continues the idea of separation as it ensures that neither the blocks nor the templates carry responsibility for the binding details.

Below is a brief description of the instructions that are present with Magento 2.2's layout XML in the `<body/>` tag:

- `<block/>`: creates a new block. Expects the full namespace and class name for the block (do not put a "\" at the beginning of the type).
- `<container/>`: a grouping of blocks. Specify an HTML tag and class to wrap the output of the child blocks.
- `<arguments>` and `<argument>`: sets a value in the block's `$data` array. (see this link)
- `<referenceBlock/>`: targets an existing block to make modifications to it.
- `<referenceContainer/>`: targets an existing container to make modifications to it.
- `<move/>`: enables moving of an existing block to another block or container.
- `<remove/>`: removes a block.
- `<update/>`: adds a layout handle to be displayed.

HELPFUL LINKS:

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

Describe how to create a new layout XML file.

- Open your favorite text editor.
- Ensure the following directory exists: `app/code/AcmeWidgets/ProductPromotor/view/frontend/layout`

- Create a file for the handle to modify and append `.xml` to it. Example: "`catalog_product_view.xml`"
- Add the `<page/>` XML node at the root.

HELPFUL LINKS:

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

Describe how to pass variables from layout to block.

Use the `arguments` and `argument` nodes. These call `setData` on the block. As such, you can retrieve these values later by calling the `getData()` method. This is also how the ViewModel is specified.

HELPFUL LINKS:

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

Example:

```
<?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="block.name">
            <arguments>
                <argument name="test_details"
xsi:type="string">Information here</argument>
            </arguments>
        </referenceBlock>
    </body>
</page>
```

3.4 UTILIZE JAVASCRIPT IN MAGENTO

POINTS TO REMEMBER

- `<script type="text/x-magento-init">` used with a JSON object to load modules and initialize them with server-generated configuration
- `data-mage-init` attribute instantiates a Javascript module with any provided configuration for the current tag.
- Using JavaScript and UI Components to load customer-specific data allows Magento to keep a page fully-cached.

Describe different types and uses of JavaScript modules. Which JavaScript modules are suited for which tasks?

Magento 2 utilizes RequireJS to create a modular and extensible system in JavaScript.

Javascript modules can be completely custom, can extend a UI Component module, or be a jQuery widget. They can be asynchronously loaded or embedded directly in a page using a plain `<script/>` tag. Magento's `<script type="text/x-magento-init"/>` allows for loading modules and passing even complex configuration info into them.

These modules handle much interaction between the website user and the Magento PHP application. Using Javascript to load aspects of customer data is ideal because it allows Magento to cache the full page. You can keep all blocks cacheable and load in the customer-specific information in an AJAX request.

For work with grids and forms in the admin panel or the checkout, a UI Component Javascript module would be a good approach. For using KnockoutJS for a customer area, a simple module that is initialized through `<script type="text/x-magento-init"/>` would work well. For handling smaller DOM manipulation tasks, a module that creates a jQuery widget would be good.

Describe UI components. In which situation would you use UiComponent versus a regular JavaScript module? Describe the use of `requirejs-config.js`, `x-magento-init`, and `data-mage-init`.

The idea behind UI components was to create a modular and reusable system for rendering layout on the frontend and adminhtml areas. A new UI component can be created and reused as many times as necessary.

UI components work well in the admin panel for grids and forms. Customizing these pages involves making updates to XML instead of repeating many lines of boilerplate PHP.

`requirejs-config.js`

This file (`app/code/AcmeWidgets/ProductPromoter/view/frontend/requirejs-config.js`) manages the dependency injection preferences for the Magento JavaScript application. RequireJS config can be used to load your module instead of an existing resource. It can also create aliases for module paths.

Example:

```
var config = {
    "map": {
        "*": {
            "mage/adminhtml/wysiwyg/tiny_mce/setup":
"AcmeWidgets_ProductPromoter/js/WysiwygSetupOverride"
        }
    }
};
```

`x-magento-init`

Initializes a JavaScript module with specific parameters.

Example:

```
<script type="text/x-magento-init">
    {
        ".hero": { // creates a new JavaScript object
instance for each selected item
            "AcmeWidgets_ProductPromoter/js/hero": {
                imageCount: 1
                // JSON object passed (from PHP) to
JavaScript
            }
        }
    }
</script>
```

`data-mage-init`

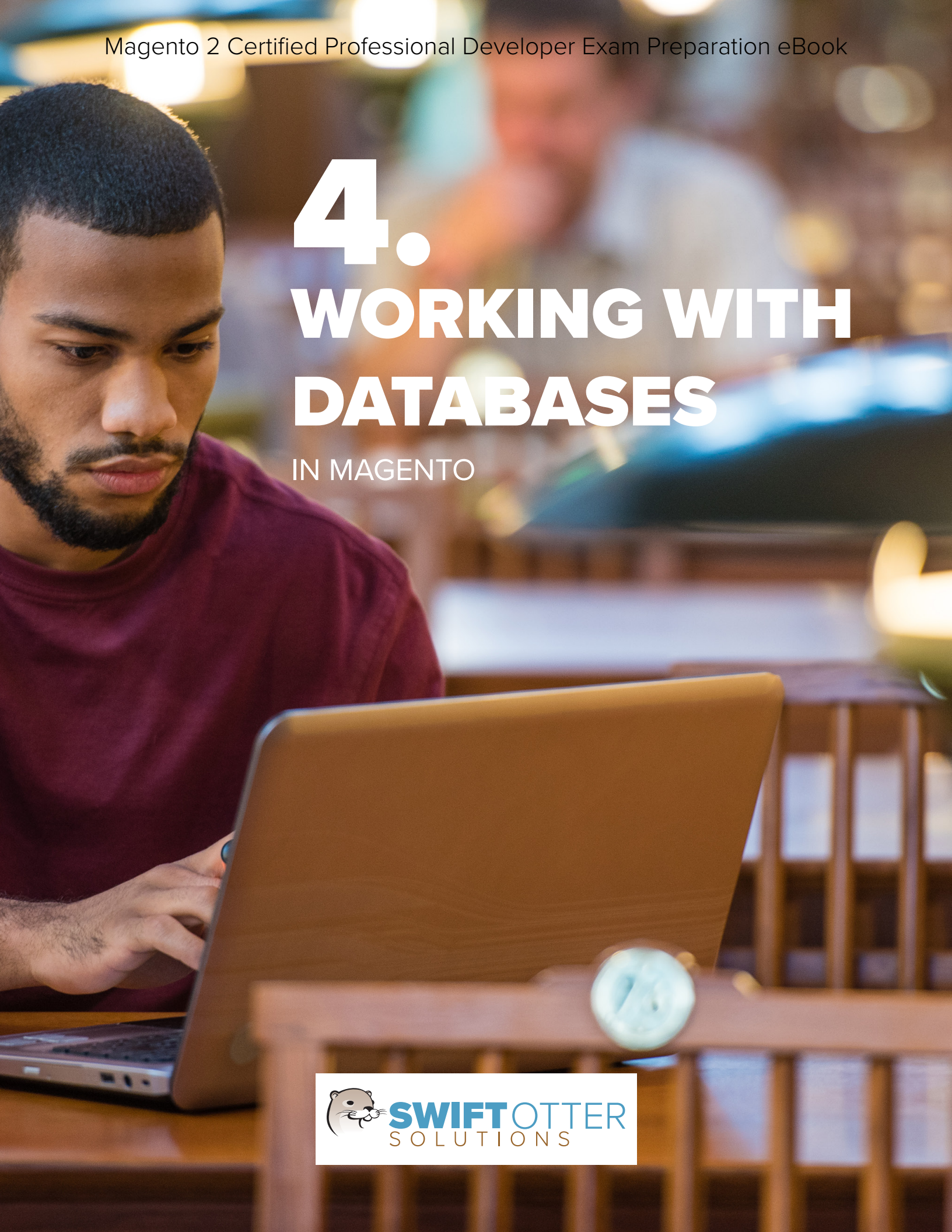
This is similar to `x-magento-init` except that it is an attribute for an HTML tag.

Example (identical in functionality to the above example):

```
<div class="hero" data-mage-init='{ "AcmeWidgets_
ProductPromoter/js/hero": { imageCount: 1} }'>
</div>
```

HELPFUL LINKS:

- http://devdocs.magento.com/guides/v2.2/javascript-dev-guide/javascript/js_init.html



4. WORKING WITH DATABASES IN MAGENTO

SWIFT OTTER SOLUTIONS

4.1 DEMONSTRATE ABILITY TO USE DATA-RELATED CLASSES

POINTS TO REMEMBER

- Familiarize yourself on the methods in repositories (`getById`, `getList`, `save` and `delete`).

Describe repositories and data API classes. How do you obtain an object or set of objects from the database using a repository?

Repositories represent a mechanism that handles data operations. They are designed to be used in the API or backend code. The purpose of Repositories is to encapsulate and hide away the persistence layer (i.e. the ORM). If used consistently, the ORM can be replaced without breaking custom code.

M1

MAGENTO 1

Repositories are often used instead of collections.

Since repositories are written against an interface, a developer is able to change out complete implementations of accessing and modifying data with little effort.

For example, the `ProductRepository` (`vendor/magento/module-catalog/Model/ProductRepository.php`) class is written to implement the `\Magento\Catalog\Api\ProductRepositoryInterface` interface. By satisfying this interface, the entire `ProductRepository` can be changed.

To get an object from the database, use the `getById` method.

To get multiple objects from the database, use the `getList` method.

Adding the table name prefix is handled on the setup resource layer with the method `$setup->getTable('tablename')`. If the methods on the database abstraction layer are called directly, the table name must be passed through `getTable()` first (e.g. `$setup->getConnection()->isTableExists($setup->getTable('tablename'))`. Methods on the setup resource layer (e.g. `$setup->tableExists('tablename')` take care or qualifying the table name themselves.

How do you configure and create a `SearchCriteria` instance using the builder? How do you use `Data/Api` classes?

- Inject an instance of the `\Magento\Framework\Api\SearchCriteriaBuilder` class.
- Call the necessary methods on the `SearchCriteriaBuilder` instance to filter (`addFilter`) or sort (`addSortOrder`) the final result.
- Call `getList` on the repository and pass in the created `SearchCriteria`.

HELPFUL LINKS:

- http://devdocs.magento.com/guides/v2.2/extension-dev-guide/searching-with-repositories.html

Describe how to create and register new entities. How do you add a new table to the database? Describe the entity load and save process.

This answer covers simple entities and not EAV, which is covered in the next segment.

Adding a new table happens in `app/code/AcmeWidgets/ProductPromoter/Setup/InstallSchema` or `app/code/AcmeWidgets/ProductPromoter/Setup/UpgradeSchema`. The `install` or `upgrade` method is called when `bin/magento setup:upgrade` is run in the CLI. The first parameter is an object that contains a connection to the database. As such, it is easy to create tables.

> M1 **MAGENTO 1**
>
> You do not have to register table names anymore. For that matter, you don't have to register blocks, helpers, models, resource models, or setup details.

Entity load process

Entity resource models extend `vendor/magento/framework/Model/ResourceModel/Db/AbstractDb.php`. This class contains the functionality that is used to load and save objects to and from the database. Look at the `load` method in this class:

- The resource model's `beforeLoad` method is called. This is a great plugin point.
- The load select is generated.
- The row is fetched.
- The object is hydrated.
- The object is returned.

Entity `save` process

The action happens in the `save` method:

- A transaction is started.
- If nothing has been modified in the entity, the transaction is commit, and the method returned.
- The resource model's `beforeSave` method is called. This is a great plugin point.
- If saving is allowed, the database representation is updated or created (based on whether the ID returns an integer or not).
- While there appears to be a bug whereby the resource model's `afterSave` is never called, this would be the point where `afterSave` is executed on the model. Any exceptions that are thrown in the execution of this method will cancel the transaction.
- The save after commit event is dispatched after a successful commit. This event is sent with the newly saved object. This event occurs after the commit and any exceptions thrown during the execution of the event will not affect the transaction.

HELPFUL FILES:

- `vendor/magento/framework/Model/ResourceModel/Db/AbstractDb.php`

Describe how to extend existing entities. What mechanisms are available to extend existing classes, for example by adding a new attribute, a new field in the database, or a new related entity?

- You can substitute the class for another class that extends the original class. This can be risky if you get into a rewrite conflict.

- Use the magic getters and setters (not good for testing) or `getData` and `setData` (not as “pretty”).
- Create a new class that is a relation to the original class. With this, create a new table that has a one-to-one relation to the original class and data structure.

Make sure that the new table entities have been created.

Extension Attributes

Extension attributes are a new phenomenon in Magento, and one that is very welcome. Prior to Magento 2, adding data to an existing entity, especially if that data came from another table or was calculated was difficult and error-prone.

At this point, extension attributes exert little control over how you implement, which gives you great power. An ACL resource can be also utilized. If a resource is set, the extension attribute will only be available via the web API to admin users with access to the given resource.

> **!** **IMPORTANT**
>
> When you implement extension attributes, be aware that you must persist the data yourself. This is different than with custom attributes where the saving operations are done automatically. With extension attributes, you save the data into the database, you load it back out and assign it to the object.

Extension attributes work with any entity that extends `\Magento\Framework\Model\AbstractExtensibleModel`.

The entity's `getExtensionAttributes` method returns an auto-generated interface that contains the Camel cased getters and setters for the attribute codes specified in `extension_attributes.xml`. The setter's argument type is an interface that you create. The getter returns an instance of the interface that you created (or `null`). This interface (and thus concrete class) stores the value(s) in the extension attribute.

Making an entity extensible:

- Change the model to extend `\Magento\Framework\Model\AbstractExtensibleModel`
- Add two methods to the entity's service contract (interface) and model:
  - `getExtensionAttributes()`
  - `setExtensionAttributes($attributes)`

The getter return type and the setter argument type is a Magento auto-generated interface. To determine what to place here, we will use the following interface for the entity:

`AcmeWidgets\ProductPromoter\Api\Data\PromotionInterface`

To determine the interface for the extension interface, insert `Extension` between `Promotion` (the entity type) and `Interface`: `PromotionExtensionInterface`.

Adding an extension attribute:

The first step to adding an extension attribute is to create `etc/extension_attributes.xml`:

```
<?xml version="1.0"?>
<config xmlns:xsi="http://www.
w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:Api/
etc/extension_attributes.xsd">
    <extension_attributes for="Magento\Catalog\Api\Data\
ProductInterface">
        <attribute code="promotions" type="AcmeWidgets\
ProductPromoter\Api\Data\PromotionLinkInterface" />
    </extension_attributes>
</config>
```

- Create an interface for the type specified in the `type` parameter.
- Add getter and setter functions to interface.
- Create an object that provides a concrete implementation to the interface.
- Add the preference into `di.xml` for the interface and concrete implementation of it.
- Create plugins for the following: `afterSave` (on the entity's repository), `afterGet` (on the entity's repository), `afterGetList` (on the entity's repository). There is no magic place to set the values for the extensions as it is still a very manual process.
- The plugin should do the following:

```
/**
  * @var \Magento\Catalog\Api\Data\
ProductExtensionInterfaceFactory
private $productExtensionFactory;

private function injectExtensionAttributes($product)
{
    $extensions = $product->getExtensionAttributes();
    /** @var \Magento\Catalog\Api\Data\
ProductExtensionInterface $extensions */
    $extensions = $extensions ?? $this-
>productExtensionFactory->create();

    /** @var \AcmeWidgets\ProductPromoter\Api\Data\
      PromotionLinkInterface $details */
    $details = $this->promotionsLinkFactory->create();
    $details->setValue(rand(0, 50000));
    $extensions->setPromotions($product);
    $product->setExtensionAttributes($extensions);

    return $customer;
}
```

Saving the extension attribute:

The opposite of the above needs to happen. You are responsible for saving the data.

HELPFUL LINKS:

- https://store.fooman.co.nz/blog/an-introduction-to-extension-attributes.html
- http://devdocs.magento.com/guides/v2.2/extension-dev-guide/extension_attributes/adding-attributes.html

Describe how to filter, sort, and specify the selected values for collections and repositories. How do you select a subset of records from the database?

M1 MAGENTO 1

The public methods on a collection has changed very little in Magento 2.

Collections

- Filter: `$collection->addFieldToFilter()`
- Sort: `$collection->addOrder()`
- Choose column: `$collection->addFieldToSelect()`
- Pagination: `$collection->setPageSize()` and `$collection->setCurPage()`

Repositories

These are a powerful feature in Magento 2. However, the simplicity of collections cannot be matched as repositories are still complex. We suggest you read through this DevDocs article to familiarize yourself with this concept.

Describe the database abstraction layer for Magento. What type of exceptions does the database layer throw? What additional functionality does Magento provide over Zend_Adapter?

See: `vendor/magento/framework/Model/ResourceModel/Db/AbstractDb.php`

Exceptions:

- “Empty identifier field name” when no ID Field Name specified
- “Empty main table name” when no main table specified
- “Unique constraint violation found” when trying to insert a row with a primary key that already exists

Additional functionality:

The resource model provides methods that make retrieving and saving rows to and from the database table very easy.

The Zend adapter class for MySQL is: `vendor/magento/zendframework1/library/Zend/Db/Adapter/Mysqli.php`.

4.2 DEMONSTRATE ABILITY TO USE, INSTALL, AND UPGRADE SCRIPTS

Describe the install/upgrade workflow. Where are setup scripts located, and how are they executed?

Upgrade and install scripts are located in a module’s Setup directory.

Example:

- `app/code/AcmeWidgets/ProductPromoter/Setup/InstallSchema.php`
  - Implements `\Magento\Framework\Setup\InstallSchemaInterface`
- `app/code/AcmeWidgets/ProductPromoter/Setup/UpgradeSchema.php`
  - Implements `\Magento\Framework\Setup\UpgradeSchemaInterface`
- `app/code/AcmeWidgets/ProductPromoter/Setup/Recurring.php`
  - Implements `\Magento\Framework\Setup\InstallSchemaInterface`
- `app/code/AcmeWidgets/ProductPromoter/Setup/InstallSchema.php`
  - Implements `\Magento\Framework\Setup\InstallDataInterface`
- `app/code/AcmeWidgets/ProductPromoter/Setup/UpgradeData.php`
  - Implements `\Magento\Framework\Setup\UpgradeDataInterface`
- `app/code/AcmeWidgets/ProductPromoter/Setup/RecurringData.php`
  - Implements `\Magento\Framework\Setup\InstallDataInterface`

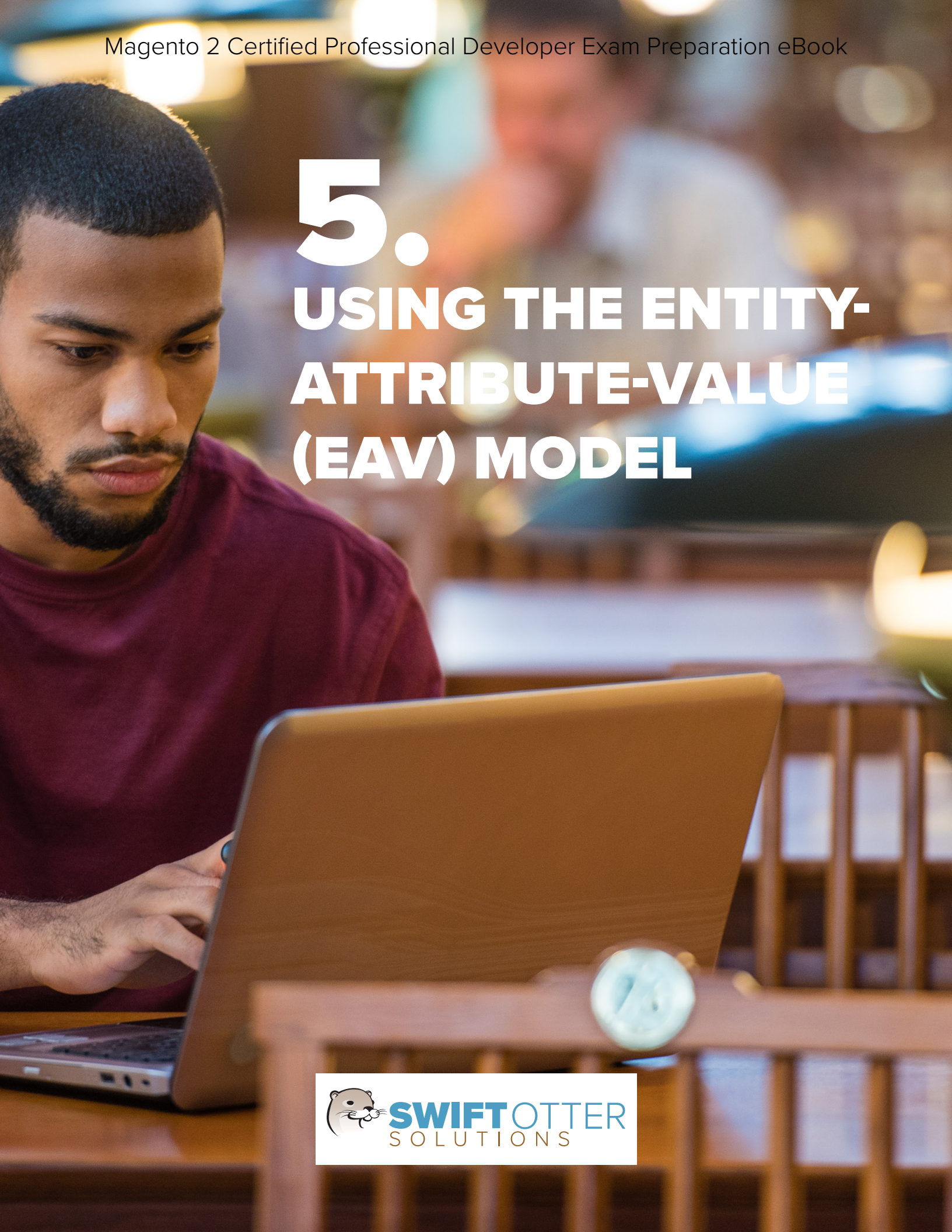
The source for the `bin/magento setup:upgrade` command is found in `setup/src/Magento/Setup/Console/Command/UpgradeCommand.php`.

The CLI command then calls `setup/src/Magento/Setup/Model/Installer.php::installSchema` which calls `handleDBSchemaData`. This function is good to review to learn how Magento handles the upgrade scripts.

Which types of functionality correspond to each type of setup script?

- Schema: database tables, columns, keys, and foreign keys.

- Recurring: triggered after every time that `setup:upgrade` is run whether or not an install or upgrade happened.
- Data: fields and row in a table.



5. USING THE ENTITY-ATTRIBUTE-VALUE (EAV) MODEL

SWIFT OTTER SOLUTIONS

5.1 DEMONSTRATE ABILITY TO USE EAV MODEL CONCEPTS

POINTS TO REMEMBER

- EAV means Entity Attribute (Attribute) Value.
- Data is stored vertically instead of horizontally (standard relational database).
- Performance is much more of a concern.

Describe the EAV hierarchy structure.

EAV is a flexible solution to storing data in a database. Relational databases are ideal for situations where there is a fixed number of descriptors (columns) for an entity row. This works well for, say, storing CMS pages or admin user details. But it does not work well for products as they can have a variable (and semi-infinite) number of descriptors (columns). While the data is split up across five or more tables, the end result in the Magento application is that a product is loaded with the set data as if it was one row (Magento handles the saving and loading operation from multiple tables).

In other words, EAV stores its relational-columnar information in rows. Instead of data representing the same entity being stored horizontally, it is stored vertically.

EAV is made up of three components. The entity, the attribute, and the attribute value.

Entity: EAV stores the primary key in the `entity` table (for example, `catalog_product_entity`). Columns in this table are considered static (see the

`backend_type` column in the `eav_attribute` table). You can find the entity types in `eav_entity_type`.

Attribute: Attributes are stored in `eav_attribute`. Each attribute represents a "column" in the loaded Magento product.

Value: Each attribute has a `backend_type`. This value determines which table the values for that attribute are stored in. `static` means those values are stored in the parent `_entity` table. Otherwise, look for a table with `_entity_[backend_type]`. The value is the last column. The first columns relate the value to the entity row, store, and the attribute ID.

What happens when a new attribute is added to the system?

- Magento adds a new row to the `eav_entity_attribute` table.
- If the new attribute is added programatically, the attribute will be automatically added to all attribute sets. The exception is if an attribute is marked as `is_user_defined = 0` (which will result in non-functioning custom attributes) or a `group` is specified.

What is the role of attribute sets and attribute groups?

Attribute sets: a set of attributes that are associated with a product. This allows a content manager to create a set of attributes that apply to a T-Shirt, which would vary from a book.

Attribute group: groups attributes on an edit page under an accordian to make attributes easier to find.

How are attributes presented in the admin?

- Stores > Product Attributes

Describe how EAV data storage works in Magento. Which additional options do you have when saving EAV entities?

EAV stores the attribute data in tables specific to the attribute's type (`backend_type` column). The exception is when the attribute type is `static`: this means the attribute value will be stored in the entity's primary table.

How do you create customizations based on changes to attribute values?

Create a plugin for the `afterSave` on the entity's resource model.

If the attribute is programmatically installed, use a `backend` model.

Setup `mview.xml` for your module and listen to changes to an attribute's table (Example: `vendor/magento/module-catalog/etc/mview.xml`).

Describe the key differences between EAV and flat table collections. In which situations would you use EAV for a new entity?

EAV data is stored vertically. Flat tables data is stored horizontally.

EAV is ideal when the content manager needs to customize or add new details. EAV is extremely flexible. Unfortunately, MySQL is not designed as such for EAV and this causes performance concerns.

In almost every case, flat tables are better.

What are the pros and cons of EAV architecture?

EAV is flexible. But, EAV is slow and harder to develop with. Because of these concerns, the use of flat tables is still prominent.

5.2 DEMONSTRATE ABILITY TO USE EAV ENTITY LOAD AND SAVE

Describe the EAV load and save process and differences from the flat table load and save process.

Source: `vendor/magento/module-eav/Model/Entity/AbstractEntity.php`

The interface for loading and saving EAV entities is identical to simple models. The process for saving and loading the initial row is similar. Once the initial entity row is loaded for a flat table, the load operation is complete. EAV entities require loading attributes from the entity attribute tables. The same is true for saving.

See `_loadModelAttributes()` in the above file for details about the loading process. Magento creates a UNION select to load attributes from each of the entity type tables to locate the applicable attribute values.

See `_collectSaveData()` in the above file for how the saving process works. EAV collections offer other methods such as `addAttributeToSelect` (note that the `field` methods, such as `addFieldToFilter` have been mapped to their `attribute` counterparts).

What happens when an EAV entity has too many attributes? How does the number of websites/stores affect the EAV load/save process? How would you customize the load and save process for an EAV entity in the situations described here?

If too many attributes are created, the website will begin to slow down. That is because of the large select statements that are needed to request the attribute values. It is recommended to enable flat tables (for products and categories). However, these flat tables are limited by the number of columns available.

Depending on the MySQL version, the maximum number of columns is fixed at 4096. This is less for prior versions of MySQL.

You can limit the number of columns in a flat table by making attributes not “Visible in Product Listing.”

The number of websites and stores can dramatically increase the loading and saving process times. The reason for this is that every variation needs to be stored in its own row. Additionally, loading the value involves selecting at least two rows. It also helps to reduce the number of options for configurable attributes helps a lot. Finally, you can split attributes into smaller attribute sets.

5.3 DEMONSTRATE ABILITY TO MANAGE ATTRIBUTES

Describe EAV attributes, including the frontend/source/backend structure. How would you add dropdown/multiselect attributes?

Frontend: formats or adjusts the value of the attribute on the frontend.

Source: provides a list of acceptable options for an attribute. The most basic example would be boolean options. See the `visibility` attribute for an example.

Backend: controls how the attribute's value is saved to the database. They are also good for validation, indexation, cache operations. For a basic example, see `vendor/magento/module-customer/Model/Attribute/Backend/Data/Boolean.php`

What other possibilities do you have when adding an attribute (to a product, for example)?

See this answer for available configuration details.

Describe how to implement the interface for attribute frontend models. What is the purpose of this interface? How can you render your attribute value on the frontend?

Example: `vendor/magento/module-eav/Model/Entity/Attribute/Frontend/Datetime.php`

- Create the new frontend class that extends `\Magento\Eav\Model\Entity\Attribute\Frontend\AbstractFrontend`.
- Set the `frontend_model` for the attribute needed to customize.
- Build the needed functionality in the `getValue` method.

The purpose for the interface is to provide dependency inversion.

To render a formatted attribute value on the frontend:

```
    /**
       *@var Magento\Catalog\Helper\Output
       */

    private $productHelper;



    public function getAttributeValue($attributeCode)
    {
        return $this->productHelper->productAttribute(
            $this->product,
            $this->product->getCustomAttribute($attributeCode)-
>getValue(),
            $attributeCode
        );
    }
```

Identify the purpose and describe how to implement the interface for attribute source models. For a given dropdown/multiselect attribute, how can you specify and manipulate its list of options?

Example: `vendor/magento/module-eav/Model/Entity/Attribute/Source/Boolean.php`

- Extend `\Magento\Eav\Model\Entity\Attribute\Source\AbstractSource` (or implement the interface that the abstract source class implements)

- If the attribute is to be included in the `catalog_product` flat tables index, make sure to build the `getFlatColumns()` and `addValueSortToCollection()` methods.
- Return results in the `getAllOptions` method.

If you need to manipulate an existing source (that does not extend the `Table` source model), create an `after` plugin for the `getAllOptions` method.

Identify the purpose and describe how to implement the interface for attribute backend models. How (and why) would you create a backend model for an attribute?

Example: `vendor/magento/module-catalog/Model/Product/Attribute/Backend/Boolean.php`

- Create a new class that implements `\Magento\Eav\Model\Entity\Attribute\Backend\BackendInterface` or extends `\Magento\Eav\Model\Entity\Attribute\Backend\AbstractBackend` (easier)

Describe how to create and customize attributes. How would you add a new attribute to the product, category, or customer entities? What is the difference between adding a new attribute and modifying an existing one?

Because attribute information is data-related and not schema-related, use the `Setup/InstallData.php` or `Setup/UpgradeData.php` classes. Inject an instance of `\Magento\Eav\Setup\EavSetupFactory` to create these attributes.

A few classes extend `\Magento\Eav\Setup\EavSetupFactory` that provide additional features for specific entity types:

- Categories: `\Magento\Catalog\Setup\CategorySetup`
- Customers: `\Magento\Customer\Setup\CustomerSetup`
- Orders, Invoices, Shipments, and Credit memos: `\Magento\Sales\Setup\SalesSetup`
- Quotes: `\Magento\Quote\Setup\QuoteSetup`

6. DEVELOPING WITH ADMINHTML

SWIFT OTTER SOLUTIONS

6.1 DESCRIBE COMMON STRUCTURE/ ARCHITECTURE

Describe the difference between Adminhtml and frontend. What additional tools and requirements exist in the admin?

The Magento admin panel provides a content manager or admin access to information that is displayed on the frontend. As such, controlling this access is very important. This is the greatest difference between adminhtml and the frontend.

The resources defined in `acl.xml` do not only apply to the admin user interface but also can be used to restrict access to REST and SOAP APIs and extension attributes.

Another difference is secret keys in admin urls. This prevents cross-site request forgery by ensuring that Magento generates links to every page in Magento admin.

The admin area allows administrators to enforce password longevity restrictions: you can be forced or encouraged to change your password every so often.

Building modules that operate in the adminhtml sphere requires taking several things into account:

- Utilizing authorization / ACL controls
- Extending different classes / implementing different interfaces
- Knowledge of specific adminhtml styles and components

Utilizing authorization / ACL controls

For many companies, granting carte blanche access to all employees who access Magento is a bad thing. Keeping this locked down to as few people as possible helps reduce mistakes and other unintentional consequences.

`etc/acl.xml` gives the developer access to creating new policy items that can be controlled by administrators. Creating these policy entries is only part of the puzzle: you must also check to ensure they are enabled before allowing an action to proceed. The resources defined in `acl.xml` do not only apply to the admin user interface but also can be used to restrict access to REST and SOAP APIs and extension attributes.

That is where Magento adminhtml has provided specific classes and interfaces to make this easy.

Extending different classes / implementing different interfaces

Blocks: see above section for discussion on how to utilize view models with blocks. The default Magento backend block is `\Magento\Backend\Block\Template`. This backend template checks the ACL to ensure that a block can be displayed.

Controllers: every action controller should extend `\Magento\Backend\App\AbstractAction`. This abstract controller provides basic authentication and authorization verification for each admin controller.

`_isAllowed` should be overridden to insert your custom resource path for verifications.

For further details, browse through the files in `vendor/magento/module-backend`.

6.2 DEFINE FORM AND GRID WIDGETS

Define form structure, form templates, grids, grid containers, and elements. What steps are needed to display a grid or form?

M1 **MAGENTO 1**

Magento 2 is very different with regards to how admin grids and forms are rendered. This section is an important read when learning the new version.

UI Components comprise a complex system spanning backend configuration and frontend functionality. They are assembled with declarative XML that allows deeply nested components. The configuration is output as JSON on the page which is then used to bootstrap a network of Javascript modules. These are small, reusable pieces of dynamic functionality which allow for building complex user interfaces.

This section will not go into great detail on how to build a UI component. It will provide you with sample files that you need to study. The best way to learn is to actually implement a UI component. They appear scary on the surface but after some trial and error, they become more approachable.

Forms

Forms are a list of labels and inputs laid out in a vertical format. For a form to work, you must provide it the following ingredients: a data source and fieldset / field configuration. The most simple form component configuration in Magento is: `vendor/magento/module-cms/view/adminhtml/ui_component/cms_block_form.xml`

As you review this field's configuration, you will see that while it is somewhat verbose, Magento gives tremendous flexibility in this customization.

Grids

In UI Component speak, grids are listings. These represent data stored in a tabular format.

The simplest version of a listing UI component is: `vendor/magento/module-cms/view/adminhtml/ui_component/cms_block_listing.xml`

Implementing a grid or form UI component:

When you build functionality, if you can find where Magento already has something similar, you just saved yourself a lot of time. Above is listed two UI components that will provide the basic foundation for getting started.

Copy the component into `app/code/AcmeWidgets/ProductPromoter/view/adminhtml/ui_component`.

Rename it to something that is applicable for you.

Search the document for all references to the previous UI component (and all derivatives of that) and replace with the comparable updated string.

Add this UI component to your layout XML. Example: `vendor/magento/module-cms/view/adminhtml/layout/cms_block_edit.xml`

Find all references to Magento PHP classes within the UI component and build comparable classes within your module.

Describe the grid and form workflow. How is data provided to the grid or form? How can this process be customized or extended?

Grid

Magento automatically wires up the data and inserts it through the `mui/index/render` controller action. This action loads data from the CMS data provider: `vendor/magento/module-cms/Ui/Component/DataProvider.php`

You can alternatively configure the grid to fetch and return the data. See an example here (`vendor/magento/module-cms/etc/di.xml`) looking at the `<type name="Magento\Framework\View\Element\UiComponent\DataProvider\CollectionFactory">` node.

Form

Using the `dataSource` nodes, Magento needs to know what `dataProvider` to retrieve data from. See this example: `vendor/magento/module-cms/Model/Page/DataProvider.php`.

Customizing the process

Plugins are ideal as they allow you to modify the output from the data provider methods (`getData()`). You can also create a preference for an entirely different data provider.

HELPFUL LINKS:

- http://devdocs.magento.com/guides/v2.2/ui_comp_guide/bk-ui_comps.html
- https://alanstorm.com/magento_2_introducing_ui_components/
- https://magento.stackexchange.com/questions/124817/debugging-ui-components

Describe how to create a simple form and grid for a custom entity. Given a specific entity with different types of fields (text, dropdown, image, file, date, and so on) how would you create a grid and a form?

This is described above in the first topic for this section.

6.3 DEFINE SYSTEM CONFIGURATION XML AND CONFIGURATION SCOPE.

Define basic terms and elements of system configuration XML, including scopes. How would you add a new system configuration option? What is the difference in this process for different option types (secret, file)?

Store configuration XML is stored in `etc/adminhtml/system.xml`. The resulting values that the admin specifies are stored in `core_config_data`. The following will cover the primary elements found in the `system.xml` file.

All configuration lives within the root node (`<config/>`) > `system`.

Tabs `<tab/>`

Example: `vendor/magento/module-backend/etc/adminhtml/system.xml`

```
    <tab id="general" translate="label" sortOrder="100">
        <label>General</label>
    </tab>
```

Sections `<section/>`

These are the items within the tab accordion on the left sidebar.

Example: `vendor/magento/module-backend/etc/adminhtml/system.xml`

Important attributes

- `id`: the name of this group. This will be used to formulate the store configuration path in `core_config_data` and in retrieving this value.
- `showInDefault`: whether this section is visible or not in the default scope (no store is selected)
- `showInWebsite`: whether this section is visible or not in the website scope
- `showInStore`: whether this section is visible or not in the store scope

Available children

- `class`: CSS classes to apply

- `label`: Title of the section (notice the attribute `translate="label"` in the parent `section` tag)
- `tab`: which tab ID this section belongs to
- `resource`: ACL path for this particular section (not specifying a resource means that all admins can access this)
- `group` (1+ entries): the groups / accordions presented on the right side after clicking into this section.

Groups `<group/>`

Collapses a list of fields into one row (showing the group's label) to make browsing easier.

Important attributes

- `id`: the name of this group. This will be used to formulate the store configuration path in `core_config_data` and in retrieving this value.
- `showInDefault`: whether this section is visible or not in the default scope (no store is selected)
- `showInWebsite`: whether this section is visible or not in the website scope
- `showInStore`: whether this section is visible or not in the store scope

Available children

- `label`: title of the section
- `field` (1+ entries): describes a field's configuration

Fields `<field/>`

This is the destination for system configuration. A field allows input by an administrator that will be saved into the database.

Important attributes:

- `id`: the name of this group. This will be used to formulate the store configuration path in `core_config_data` and in retrieving this value.
- `type`: one of `text`, `wysiwyg`, `textarea`, `select`, `multiselect`, `obscure`. This can be blank if you specify the `frontend_model` node.
- `showInDefault`: whether this section is visible or not in the default scope (no store is selected)
- `showInWebsite`: whether this section is visible or not in the website scope
- `showInStore`: whether this section is visible or not in the store scope

Available children

- `label`: the field's title
- `comment`: an explanatory note to describe the field.
- `source_model`: specifies a class that implements `\Magento\Framework\Option\ArrayInterface`. This provides a list of options to select or multiselect.
- `frontend_model`: specifies a class that extends `\Magento\Config\Block\System\Config\Form\Field`. This is how you display a custom input field in store configuration.

- `backend_model`: Changes or adjusts data coming to / from the system (example: `vendor/magento/module-authorizenet/etc/adminhtml/system.xml`). The `obscure` field type commonly uses this.
- `depends`: makes the visibility of this field dependent on the setting of another field.
- `validate`: a CSS validation class (example: `validate-email`)

Describe system configuration data retrieval. How do you access system configuration options programmatically?

Magento provides two layers for store configuration: the defaults (specified in `etc/config.xml`) and the options stored in the database.

The data is retrieved in `\Magento\Framework\App\Config` by the `\Magento\Config\App\Config\Type\System` configuration type.

Accessing store configuration

Inject `\Magento\Framework\App\Config\ScopeConfigInterface` into a class needing this configuration information. Call the `getValue` (for raw value) or `isSetFlag` (for boolean value) method to retrieve the value needed.

6.4 UTILIZE ACL TO SET MENU ITEMS AND PERMISSIONS

Describe how to set up a menu item and permissions. How would you add a new menu item in a given tab? How would you add a new tab in the Admin menu? How do menu items relate to ACL permissions?

Menu items are configured in the `etc/adminhtml/menu.xml` XML configuration file.

Here is a very basic example:

```
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:module:Magento_
Backend:etc/menu.xsd">
    <menu>
        <add id="AcmeWidgets_ProductPromoter::PromotionBuilder"
            title="Promotion Builder"
            translate="title"
            module="AcmeWidgets_ProductPromoter"
            sortOrder="50"
            parent="Magento_Catalog::inventory"
            action="promotion/builder"
            resource="AcmeWidgets_ProductPromoter::PromotionBuilder"
            />
    </menu>
</config>
```

The primary node here is `<add/>`. The following will discuss the attributes listed above.

- `id`: this value is used when creating a hierarchy of menu items. The `parent` module refers to another `<add/>` node's `id` parameter.
- `title`: the title of the menu item
- `translate`: which parameters to translate. This is usually just `title`.
- `module`: the module that is associated with the menu item
- `sortOrder`: determines the order in which this item appears
- `parent`: menu item that will host the current menu item. If no `parent` is specified, this item will appear on the sidebar.
- `action`: the controller action to send the click of the menu item. If no `action` is specified, this item will act as a header.
- `resource`: the ID of the ACL entry to validate the user's permissions.

The other nodes that are available are `update` and `remove`. Both take the `id` attribute relating to another menu item's `id` attribute.

Describe how to check for permissions in the permissions management tree structures. How would you add a new user with a given set of permissions? How can you do that programmatically?

Every item in the ACL has an ID. This ID is used as the lookup field for determining whether or not the action is allowed.

The tree structure is useful for allowing or disallowing a particular group. As a result, that particular resource ID might not have an entry in the database. Magento does a “depth-first search” (see: `vendor/magento/zendframework1/library/Zend/Acl.php`) starting at the deepest node (the one with the ID you are looking up) and works up until it finds an Allow or Deny for the current user.

To create a user with a particular set of permissions (role), create the user and specify its role.

Programatically, you could obtain an instance of `\Magento\User\Model\UserFactory`, set the details for the user, and specify `role_id` for this instance `$user->setData('role_id', $roleId);`.

7. CUSTOMIZING THE CATALOG

SWIFT OTTER SOLUTIONS

7.1 DEMONSTRATE ABILITY TO USE PRODUCTS AND PRODUCT TYPES

Identify/describe standard product types (simple, configurable, bundled, etc.). How would you obtain a product of a specific type?

Magento provides the capability for various product types. The default products are:

- `simple`: represents the basic unit of inventory on the shelf.
- `configurable`: provides a select list of options based on chosen attributes.
- `grouped`: customer selects how many they want to order of each product in a list.
- `bundled`: a list of multiple configuration options.
- `virtual`: an intangible product or digital goods.
- `downloadable`: virtual product that can be downloaded.

Retrieving product(s) of a specific type:

See example on next page.

```
namespace AmceWidgets\ProductPromoter\Model;

class ProductLoader
{
    private $productRepository;
    private $searchCriteriaBuilder;
```

```
    public function __construct(
        \Magento\Catalog\Api\ProductRepositoryInterface
$productRepository,
        \Magento\Framework\Api\SearchCriteriaBuilder
$searchCriteriaBuilder
    ) {
        $this->productRepository = $productRepository;
        $this->searchCriteriaBuilder =
$searchCriteriaBuilder;
    }

    public function getBundleProducts()
    {
        $searchCriteria = $this->searchCriteriaBuilder
            ->addFilter(\Magento\Catalog\Api\Data\
ProductInterface::TYPE_ID, \Magento\Bundle\Model\Product\
Type::TYPE_CODE)
            ->create();

        return $this->productRepository-
>getList($searchCriteria)->getItems();
    }
}
```

What tools (in general) does a product type model provide?

The product type model:

- is responsible for handling data as it is passed to and from the database.
- handles loading child products (if applicable).
- loads and configures product options.
- checks whether the item is saleable.
- prepares the product to be added to the cart.

What additional functionality is available for each of the different product types?

- Simple product represents the basic unit of inventory.
- Configurable products load the list of their children and the attribute associated with them.
- Downloadable products are virtual at heart, but after the sale allows the user to download content. It provides the ability to limit the download to the logged in user.
- Grouped products are similar to configurable in that it stores a list of children, but no attributes are used in loading the children.
- Bundled products contain functionality to load the bundle options, configure the final products, and display the price.

7.2 DESCRIBE PRICE FUNCTIONALITY

POINTS TO REMEMBER

- Base price, special pricing, and catalog rules apply to the price visible on a product page.
- Tiered pricing, options price, tax / VAT (depending on the point in the checkout process), and shopping cart rules determine the price in the shopping cart.

Identify the basic concepts of price generation in Magento. How would you identify what is composing the final price of the product? How can you customize the price calculation process?

Magento offers many layers of pricing calculation in the application. Here are the primary calculations that take place for the price shown on a product page: (`\Magento\Catalog\Model\Product\Type\Price::calculatePrice`)

- Base price (`price` attribute) or existing price
- Special pricing
- Catalog rules

Once a quantity has been determined for a product (ie. added to the cart), several other options apply:

- Tiered pricing (applicable to quantity, customer group, and website)
- Options price
- Tax / VAT

When determining a product's final price (what is shown on the product's page), Magento works through each one of these.

`vendor/magento/module-catalog/Model/Product/Type/Price.php` is where these calculations take place. Customization can happen with plugins (`afterCalculatePrice`, for example) or replacing the entire price calculation class.

Describe how price is rendered in Magento. How would you render price in a given place on the page, and how would you modify how the price is rendered?

Pricing renderers are setup in `vendor/magento/module-catalog/view/base/layout/default.xml`. In this file, a block is created with the name `product.price.render.default`. You can use this block to render pricing elsewhere in the application (example: `vendor/magento/module-downloadable/view/frontend/layout/catalog_product_view_type_downloadable.xml`).

The templates for these renderers are found here: `vendor/magento/module-catalog/view/base/templates/product`.

Additionally, there is a JS UI `component` to display prices. These templates can be found in `vendor/magento/module-catalog/view/base/web/template`.

7.3 DEMONSTRATE ABILITY TO USE AND CUSTOMIZE CATEGORIES

POINTS TO REMEMBER

- Root category is used to group categories within a store. It is never shown to the customer.
- `parent_id = 0` is the identifier for the grandparent of root categories. ID `0` does not exist in the database and is used as a placeholder.
- Category name is the only required category attribute that is not automatically specified.

Describe category properties and features. How do you create and manage categories?

Categories are easily created in the Magento Admin panel under the Catalog menu item. Categories represent a tree-view hierarchy in that there is one parent category and everything descends from that parent.

You can also create categories with code using the `\Magento\Catalog\Api\Data\CategoryInterfaceFactory` object and saving with `\Magento\Catalog\Api\CategoryRepositoryInterface`.

Categories are an EAV-type and are stored in the `catalog_category_entity` table.

Describe the category hierarchy tree structure implementation (the internal structure inside the database).

The hierarchy is stored in the `path` column in the `catalog_category_entity` table. The first number in the path is the root category; the last number is the `entity_id` of the current row. By exploding the `path` column by the `/` character, you can determine the path to the current category. Additionally, the `level` column gives insight as to how deep this category is within the tree.

Please note that the root category is never shown to the frontend user. This is used to group categories within a store.

HELPFUL FILES:

- `vendor/magento/module-catalog/Model/Category/Tree.php`

What is the meaning of parent_id 0?

Parent ID `0` serves as the grandparent to all root categories (ID `0` only exists in code and not in the database). Magento automatically creates the category with ID `1`, which is the parent to all root categories.

`Category::ROOT_CATEGORY_ID = 0;` is specified in the `Category` model but is not referenced or used anywhere else.

`Category::TREE_ROOT_ID = 1;` is specified in the `Category` model and is used extensively in determining the tree path in the `path` column.

Here is a screenshot of the usage in the database:

row_id	entity_id	created_in	updated_in	attribute_set_id	parent_id	created_at	updated_at	path	position	level	children_count
1	1	1	2147483647	3	0	2016-12-30 18:42:23	2018-03-06 13:30:17	1	0	0	107
2	2	1	2147483647	3	1	2016-12-30 18:42:23	2017-09-07 17:41:55	1/2	1	1	105
3	3	1	2147483647	3	2	2016-12-30 18:42:49	2018-01-24 18:11:05	1/2/3	4	2	4

How are paths constructed?

Paths are constructed by taking the parent ID of the current category and ascending up the tree until the root category's parent (ID: `1`) is reached. This is then imploded separated by a `/`.

A good way to see how paths are constructed is by inspecting the code that constructs the paths: `\Magento\Catalog\Model\ResourceModel\Category::changeParent`.

Which attribute values are required to display a new category in the store?

- Category name
- Available Product Listing Sort By (defaults to Use All)
- Default Product Listing Sort By (defaults to Use Config Settings)

What kind of strategies can you suggest for organizing products into categories?

The most important aspect of this is how a user wants to browse the website. Performing user studies and analysis is better than a theoretical strategy that can be contrived and then presented.

Beyond that, products should be organized into logical groups. For example, this might be “Computers” or “Smartphones” or “Carrying cases.” Under “Computers” would likely be “Laptops,” “Desktops” and “All-in-One.”

Content managers can use the sort order column to show more popular or strategic products first.

7.4 DETERMINE AND MANAGE CATALOG RULES

POINTS TO REMEMBER

- Catalog rules are indexed and the data is stored in `catalogrule_product_price`.

Identify how to implement catalog price rules. When would you use catalog price rules? How do they impact performance? How would you debug problems with catalog price rules?

Setting up catalog price rules is done in the admin panel under Marketing > Catalog Price Rule. Here, the content manager can easily filter for product(s) to apply this rule to, allow only specific customer groups to utilize it, and apply discounts. This is not to be confused with Shopping Cart Rules.

Catalog price rules are a great way to set up sales on a more global basis than special pricing allows for. A product’s special price is easy to set up for a one-off product or simple group of products. Catalog price rules can be used to apply a

discount to a set of (or all) products. Unlike special pricing, catalog price rules can apply to certain customer groups.

Catalog price rules will slightly affect performance. These rules are not indexed by the price indexer. They are indexed, however, by the Catalog Product Rule index and the applicable rule price resides in the `catalogrule_product_price` table.

Debugging rules

Debugging rules for a merchant is much easier if you have access to a copy of their production database (we have built a tool that we use to always keep secure copies of production data with no custom information: Driver).

Catalog rules are indexed. This means that debugging has to isolate the problem in two places:

- is the problem the data going into the indexed table?
- is the problem the data coming out of the indexed table and not being applied to the product?

The catalog rule indexes are built in: `vendor/magento/module-catalog-rule/Model/Indexer/IndexBuilder.php`

8. CUSTOMIZING THE CHECKOUT PROCESS

SWIFT OTTER SOLUTIONS

8.1 DEMONSTRATE ABILITY TO USE QUOTE, QUOTE ITEM, ADDRESS, AND SHOPPING CART RULES IN CHECKOUT

Describe how to modify these models and effectively use them in customizations.

Each of the following entities extends `\Magento\Framework\Model\AbstractExtensibleModel` so you can use extensible attributes as discussed in Chapter 4.

Quote

Quotes are the "storage container" for a customer's shopping cart session. They contain details about what is being purchased, the current totals, and customer information. A quote is associated with the current session in `\Magento\Checkout\Model\Session::getQuote()`.

Quotes are stored in the `quote` table.

This is a frequent place to make customizations as this is integral to the customer order experience. The model containing the quote's data is: `vendor/magento/module-quote/Model/Quote.php`

Quote Item

Quote items contain the contents of a visitor's shopping cart. They link the quote (shopping cart) to products. These items are stored in `quote_item`. This table

is a good place to store additional information about the item (such as custom information about taxes).

A quote item is represented by `vendor/magento/module-quote/Model/Quote/Item.php`.

Address

Every quote has at least a billing address and a shipping address (unless the quote only contains virtual items). A quote address is represented by `\Magento\Quote\Model\Quote\Address`. Quote addresses also store a list of items that are related to this address.

Shopping cart rules

Shopping cart rules apply discounts to items in the cart. The `Magento\SalesRule` module handles this logic.

Describe how to customize the process of adding a product to the cart. Which different scenarios should you take into account?

There are a number of ways to add items to the cart in Magento 2:

- Frontend, through Magento application
- Backend, through Magento application
- From the wishlist
- Reordering a product

- During quotes merge (visitor has items in their cart, logs in, the current quote is merged with the quote associated with the logged in customer)
- REST API

If customization needs to happen when adding items to the cart, it is important to not hook into a specific controller, but rather use plugins on functions that are used in every situation.

8.2 DEMONSTRATE ABILITY TO USE TOTALS MODELS

Describe how to modify the price calculation process in the shopping cart. How can you add a custom totals model or modify existing totals models?

To create a custom total modal, you will need to wire it up in `etc/sales.xml` and create the model that will handle the processing.

`vendor/magento/module-tax/etc/sales.xml` provides an example of how the tax module links this up:

See next page for example.

```
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:module:Magento_
Sales:etc/sales.xsd">
    <section name="quote">
        <group name="totals">
            <item name="tax" instance="Magento\Tax\Model\
Sales\Total\Quote\Tax" sort_order="450">
                <renderer name="adminhtml"
instance="Magento\Sales\Block\Adminhtml\Order\Create\
Totals\Tax"/>
                <renderer name="frontend"
instance="Magento\Tax\Block\Checkout\Tax"/>
            </item>
        </group>
    </section>
</config>
```

Every total model extends `\Magento\Quote\Model\Quote\Address\Total\AbstractTotal`.

8.3 DEMONSTRATE ABILITY TO CUSTOMIZE THE SHOPPING CART

Describe how to implement shopping cart rules.

POINTS TO REMEMBER

- The only method that is common means of adding an item to the cart is `\Magento\Sales\Model\Quote::addItem`.
- Only one shopping cart rule as applied with a coupon code can be used at a time. Multiple rules can be applied at once.

Shopping cart rules are primarily built through the admin panel (the other option is adding it programmatically through `Setup/UpgradeData.php`). This is done in Marketing > Cart Price Rules.

What is the difference between sales rules and catalog rules?

Sales rules apply to products that are in the cart. Catalog rules apply to products before being added to the cart.

As a result of being added to the cart, sales rules have more options for customizing as they can offer free shipping, discounts on the entire cart, and specific coupon codes.

How do sales rules affect performance? What are the limitations of the native sales rules engine?

Sales rules affect performance negatively in the shopping cart and order creation because processing the rules can be a significant task (especially if there are complex filtering conditions). Performance will slow down the more global (not limited to a customer group or website) the rule, and the more rules that are available, and the fewer that require a coupon code.

Limitations: only one coupon code can be used at a time. You cannot have one product in your cart and another appear for free (or a discount). Rules cannot add other products to the cart. Rules only apply to the item they were designated to work with.

Describe add-to-cart logic in different scenarios. What is the difference in adding a product to the cart from the product page, from the wishlist, by clicking Reorder, and during quotes merge?

Product page: `vendor/magento/module-checkout/Controller/Cart/Add.php`

Events triggered (object provided):

- `checkout_cart_add_product_complete`
- `checkout_cart_product_add_after` (in cart entity, applicable to frontend, backend, and REST)
- `sales_quote_product_add_after` (in quote entity)
- `sales_quote_add_item` (in quote entity)

A buy request is created with the incoming quantity and other product options. This is passed into the cart model which interacts with the quote model to generate items to add.

Wishlist page: `vendor/magento/module-wishlist/Controller/Index/Cart.php`

Events triggered:

- `checkout_cart_product_add_after` (in cart entity, applicable to frontend, backend, and REST)
- `sales_quote_product_add_after` (in quote entity)
- `sales_quote_add_item` (in quote entity)

The buy request is used to check that it still is correct. The wishlist item is then told to add itself to the cart. This calls the cart `addProduct` method which is what the above path calls.

Reorder: `vendor/magento/module-sales/Controller/AbstractController/Reorder.php`

Events triggered:

- `checkout_cart_product_add_after` (in cart entity, applicable to frontend, backend, and REST)
- `sales_quote_product_add_after` (in quote entity)
- `sales_quote_add_item` (in quote entity)

The reorder controller calls the cart's `addOrderItem` method. This loads the buy request and calls the cart's `addProduct` method.

Merge: `vendor/magento/module-quote/Model/Quote.php::merge`

Events triggered:

- `sales_quote_add_item` (in quote entity)

Describe the difference in behavior of different product types in the shopping cart. How are configurable and bundle products rendered? How can you create a custom shopping cart renderer?

- **Simple**: appear as one line item in the shopping cart.
- **Configurable**: appear as one line item in the cart. The parent product's title is shown with the chosen option visible.
- **Bundle**: appear as one line item in the cart with all the options displayed below the title.
- **Grouped**: appear as multiple line items: one for each item chosen.

Rendering configurable and bundle products

These products are rendered as specified in `view/frontend/layout/checkout_cart_item_renderers.xml` (example: `vendor/magento/module-bundle/view/frontend/layout/checkout_cart_item_renderers.xml`). The `renderer` block specified the `as` parameter to determine which child to render for the given input.

Describe the available shopping cart operations. Which operations are available to the customer on the cart page?

- Change item quantity
- Delete item
- Edit item
- Move to wishlist (if enabled)
- Add gift message (if enabled)
- Update cart (after changing item quantities)

- Apply / remove coupon
- Go to checkout

How can you customize cart edit functionality?

Clicking `edit` on a product returns the visitor to what looks like the product page, except that the URL starts with `checkout/cart/configure`.

The configuration controller is in `vendor/magento/module-checkout/Controller/Cart/Configure.php`. This action simply loads the quote item and sends it to the product helper to render.

How would you create an extension that deletes one item if another was deleted?

Create an event observer for `sales_quote_remove_item` or a plugin for the `\Magento\Quote\Model\Quote::removeItem` OR `\Magento\Quote\Api\CartItemRepositoryInterface::deleteById()` method.

How do you add a field to the shipping address?

- Add a new column to the `sales_order_address` table.
- Create a `view/frontend/layout/checkout_index_index.xml` file.
- Replicate the path to `<item name="shipping-address-fieldset" xsi:type="array">` that is found in: `vendor/magento/module-checkout/view/frontend/layout/checkout_index_index.xml`
- Add the `children` node and specify the child node.

8.4 DEMONSTRATE ABILITY TO CUSTOMIZE SHIPPING AND PAYMENT METHODS

POINTS TO REMEMBER

- Shipping methods implement `\Magento\Shipping\Model\Carrier\CarrierInterface`
- Payment methods implement `\Magento\Quote\Api\Data\PaymentMethodInterface` and `\Magento\Payment\Model\MethodInterface`

Describe shipping methods architecture. How can you create a new shipping method?

Shipping methods are configured with XML in the `etc/` folder.

Creating a new shipping method:

- Create a new group in `etc/adminhtml/system.xml` for `carriers/[shipping_code]`.
- Add `etc/config.xml` and create the path for `default/carriers[shipping_code]` with default values (if necessary) for what was specified in `system.xml`.
- Add a `<model/>` node to point to the class that contains the shipping method.
- Create a new class that implements `\Magento\Shipping\Model\Carrier\CarrierInterface` and possibly extends `\Magento\Shipping\Model\Carrier\AbstractCarrierOnline`.

What is the relationship between carriers and rates?

Carriers provide a list of available rates. For example, for UPS, the available rates might be Overnight, Overnight AM, and Ground.

Describe how to troubleshoot shipping methods and rate results.

- Find the shipping method's class.
- Set a breakpoint in `\Magento\Shipping\Model\Carrier\AbstractCarrierOnline::canCollectRates` to make sure that the carrier is enabled. Remember that plugins can change the returned value.
- Set a breakpoint in the implementation for `\Magento\Shipping\Model\Carrier\AbstractCarrierInterface::collectRates` and step through the request formulation and the response parsing.

Where do shipping rates come from?

Shipping rates can come from an API, calculations, or be fixed.

Examples:

- API: `\Magento\Fedex\Model\Carrier`
- Calculations: `\Magento\OfflineShipping\Model\Carrier\Tablerate`
- Fixed: `\Magento\OfflineShipping\Model\Carrier\Flatrate`

How can you debug the wrong shipping rate being returned?

See the answer above for "Describe how to troubleshoot shipping methods and rate results."

Describe how to troubleshoot payment methods.

Similar to the above methods for troubleshooting shipping methods. For simpler payment methods, check to ensure that the method's `isAvailable()` is returning `true`. This is originally declared in `\Magento\Payment\Model\Method\AbstractMethod`.

More complex implementations such as Braintree, Paypal, and Authorize.net are more difficult to troubleshoot. To enable better security, these solutions have transitioned to iframes or a JavaScript token solution. The IPN (instant payment notification), the route where PayPal sends updates about the payment status, is not easily debugged.

Braintree also uses a relatively new Magento module called the Vault. The Vault provides store customers with the ability to save credit cards in a PCI compliant way. From a developers perspective, it provides a common set of functionality around loading and saving sensitive credit card information.

The good news is that many of these payment methods have a `Debug` configuration setting. Turning this on enables verbose logging in `var/log` and can help point to where problems are occuring.

HELPFUL LINKS:

- http://devdocs.magento.com/guides/v2.1/payments-integrations/vault/vault-intro.html

What types of payment methods exist?

There are two primary types of payment methods: offline and online. Offline is represented by check / money order, purchase order, or bank transfer. Online is any payment method that communicates to an external webservice. This would be methods like Paypal, Braintree, Authorize.net, and Cybersource (Magento Commerce).

What are the different payment flows?

Offline and online. While both utilize the same implementation, offline payment methods usually do very little with the authorize and capture steps. On the other hand, online payments utilize these steps to communicate with the external API to ensure funds are available, capture them, and refund the funds.

9. SALES OPERATIONS

SWIFT OTTER SOLUTIONS

9.1 DEMONSTRATE ABILITY TO CUSTOMIZE SALES OPERATIONS

POINTS TO REMEMBER

- Going into Magento admin > Stores > Order Status > Assign Order Status to State allows you to change the status associated with the default order state.
- There are two types of credit memos (as associated with the payment method): online and offline.

Describe how to modify order processing and integrate it with a third-party ERP system.

There are several readily accessible touchpoints to integrate Magento orders with an external system.

REST API

Magento 2 includes an extensive API that provides access to orders. Specifically, the sales order repository has been made available through the API: `/rest/V1/orders`. See: `http://devdocs.magento.com/swagger/index_22.html`. The documentation provides information on how to filter for a specific subset of orders.

The use case for this would be an external service that queries Magento for the latest orders. This service operates as a pull operation, retrieving orders from Magento, and pushing into the ERP.

Events

Magento provides a number of events that indicate when an order is complete. For example, `sales_order_place_after` is triggered when the order has been placed.

You can also use RabbitMQ (available in Magento Open Source as a module). If consistency is important, using a messaging system might not be the best approach. If the operation fails after being sent to the external system, the data in the two systems becomes inconsistent.

Describe how to modify order processing flow. How would you add new states and statuses for an order? How do you change the behavior of existing states and statuses?

By default, an order is placed, then invoiced, then shipped (marking the order as complete).

There are a number of ways to change this. First, invoices can be automatically created. An event such as `sales_order_place_after` is an ideal time for this. Hooking into a 3rd-party system that will automatically generate orders also could modify this flow.

Adding new states and statuses for an order.

You can create new order statuses in the admin panel, Stores > Order Statuses. This can be also updated in the database.

Clicking on the Assign Status to State button in this panel allows you to change the order status associated with the default state.

States are hardwired into the `\Magento\Sales\Model\Order` class. While you can create plugins to hook into and change these states, unintended consequences might arise.

Describe how to customize invoices. How would you customize invoice generation, capturing, and management?

Invoices are created when payment has been captured. There are, however, a number of cases where payment capture doesn't matter. An excellent example is a terms payment method (where a customer is invoiced at the end of the month) when an ERP is involved. In these cases, the order needs to be marked as complete because the ERP handles invoicing at the end of the month. Automatically creating the invoice is necessary here.

Invoices are built against the `\Magento\Sales\Api\Data\InvoiceInterface` service contract. Additionally, invoices are extendable, so you can create extension attributes for them.

Invoices are also accessible through the Magento REST API.

Describe refund functionality in Magento. Which refund types are available, and how are they used?

Refunds (credit memos) in Magento are used to return money back to the purchaser. Often this is a result of the customer returning part of (or all of) their order.

There are two refund types: online and offline. In the admin order view, you will see a button called "Credit Memo." Creating a credit memo here will generate an offline credit memo. This means that if an online payment method was used (Braintree,

Paypal, Authorize.net), the payment provider will not be contacted to issue the refund: it happens offline. This is a point of confusion for many merchants.

An online credit memo (when the external payment's provider is contacted to create the refund) is created from an invoice. To create an online memo, go to an order, click on the Invoices tab, choose an invoice, and then click Credit Memo. Please note that this option is only available if the payment method is online.

10. CUSTOMER MANAGEMENT

SWIFT OTTER SOLUTIONS

10.1 DEMONSTRATE ABILITY TO CUSTOMIZE MY ACCOUNT

Describe how to customize the “My Account” section. How do you add a menu item?

The My Account area of the customer control panel is controlled with the `customer_account` layout handle (`view/frontend/layout/customer_account.xml`).

To add a menu item, create the above layout file and add the following code:

```
<?xml version="1.0"?>
<body>
    <referenceContainer name="content">
        <!-- removing the wish list link -->
        <referenceBlock name="customer-account-navigation-
wish-list-link" remove="true" />

        <referenceBlock name="customer_account_navigation">
            <block class="Magento\Customer\Block\
Account\SortLinkInterface" name="product-promotions"
after="customer-account-navigation-address-link">
                <arguments>
                    <argument name="label" xsi:type="string"
translate="true">Promoted Products</argument>
                    <argument name="path"
xsi:type="string">promotions/view</argument>
                    <argument name="sortOrder"
xsi:type="number">1</argument>
                </arguments>
            </block>
        </referenceBlock>
    </referenceContainer>
</body>
```

How would you customize the “Order History” page?

Changes here are made in the `sales_order_history` layout handle (`view/frontend/layout/sales_order_history.xml`). The origin for this layout handle is a good starting place for modifications: `vendor/magento/module-sales/view/frontend/layout/sales_order_history.xml`

An easy place to insert additional content is in the `sales.order.history.info` container on this page.

10.2 DEMONSTRATE ABILITY TO CUSTOMIZE CUSTOMER FUNCTIONALITY

POINTS TO REMEMBER

- New customer attributes must be associated with a form in the DB table `customer_form_attribute`.
- Customer groups are an excellent way to broadly apply pricing changes (for example, across an entire category).

Describe how to add or modify customer attributes.

Adding customer attributes is similar to creating any other type of EAV attribute with one exception. Once the attribute is created, it must be added to a form to be editable.

```
$attribute = $this->eavConfig->getAttribute(\Magento\
Customer\Model\Customer::ENTITY, Attribute::CUSTOMER_
PROMOTION_PREFERENCE);
$attribute->setData('used_in_forms', ['adminhtml_customer',
'customer_account_edit']);

$attribute->save();
```

Running the above code in a `UpgradeData` setup script will add this attribute to admin forms specified in `customer_form_attribute`. The frontend is more manual and will require you adding the fields to templates in the customer account area.

IMPORTANT

If you have created the attribute, and added it to the frontend customer account HTML, but the attribute is not saving, check to make sure that the attribute is in the `customer_account_edit` form. Additionally, check that the `is_system` attribute value is set to `0`.

Describe how to extend the customer entity. How would you extend the customer entity using the extension attributes mechanism?

Extending the customer entity through extension attributes is detailed in chapter 4. Magento makes including this extra information very easy.

You can also create a custom EAV attribute (discussed above).

You can use an altogether different class preference (substituting the original class).

Describe how to customize the customer address. How would you add another field into the customer address?

Adding a field to the customer address involves creating a new customer address attribute (see `eav_entity_type` table), assigning that attribute to a form, and adding the field to the frontend.

Magento 2 does not automatically update the frontend customer fields. Again, keep in mind that customer and customer address attributes must be included in a form (`customer_form_attribute` table) so they can be saved.

HELPFUL LINKS:

- https://magento.stackexchange.com/a/125367/13

Describe customer groups and their role in different business processes. What is the role of customer groups? What functionality do they affect?

Customer groups are a way for customers to be categorized. Many businesses offer wholesale pricing. When a customer agrees with a merchant's reseller policy, the store manager can assign the "Wholesale" group to the customer. Some merchants might have differing levels of discounts which would be associated with different customer groups.

The advantage of utilizing customer groups is that:

- using tiered pricing, products can be discounted by customer group based on the quantity purchased.
- tax rules can be specified per customer group.
- using catalog price rules, groups of products can be discounted by customer group, category, and websites.
- using shopping cart rules, promotions can be applied to products in the shopping cart.

Describe Magento functionality related to VAT. How do you customize VAT functionality?

VAT is used in many countries in the world whereas sales tax is used in the United States. VAT is usually included in the price whereas sales tax is usually added to the final price.

Magento accommodates VAT calculations. This is setup by first creating tax classes for VAT and then customizing the store configuration to match VAT calculation requirements.

Read the following articles to understand how to configure and setup VAT.

HELPFUL LINKS:

- https://www.mageplaza.com/kb/how-to-configure-eu-tax-magento-2.html
- http://docs.magento.com/m2/ce/user_guide/tax/vat-validation.html

ACKNOWLEDGEMENTS

Author: Joseph Maxwell

Editor: Jesse Maxwell

Special thanks to my awesome team: Sarah, Anna, and Mary who took care of copy editing, layout / design, and video edits.