SwiftOtter's

Professional DEVELOPER PLUS Study Guide

A detailed guide to advanced Magento development.

Joseph Maxwell

INTRODUCTION

You made a great decision by purchasing the most comprehensive study guide for the Magento 2 Certified Professional Developer Plus certification (testing your knowledge of Magento Commerce 2.2)!

Our goal is that this guide serves two purposes:

1. For those who are new to Magento development, this guide will give concrete information on getting started. This is not a "read and you will pass" guide but rather a launching point for obtaining the experience necessary to pass. The paid version contains nine chapters and many illustrations on how to complete tasks. Each illustration has been stripped back to the minimum allowing you to easily build and extend.

2. For those who are experienced and are ready to go for this certification, this will be both a refresher on subjects you haven't worked on lately and will point out some new areas that you haven't utilized yet (message queueing anyone?).

The study guide answers the questions presented in Magento's study guide for the Professional Developer Plus test. Just reading this study guide is part of solid preparation for passing the test but does not guarantee a passing grade. That is why we have included a "Practical Experience" section on most points. This will guide you to specific launching points where you can dive in and learn.

Requirements

Please understand that the code examples are designed to work with Magento Commerce 2.2. If you do not have access to Magento Commerce code, you will run into parts of the code that do not function properly. In addition, you might pass the test but will have a very difficult time with the staging and Commerce questions.

Acknowledgments

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

- Andreas von Studnitz (Integer Net)
- Joshua Romero (MedioType)
- Navarr Barnier (MedioType)
- Artem Klimov
- Itonomy Commerce
- Richard Feraro (Acid Green)
- Yuchen Liu (Acid Green)
- Edward Sackett (Acid Green)
- Edgar Dela Cruz (Acid Green)
- Jan Ramos (Acid Green)
- Roman Vovnenko (Acid Green)
- Tony Trinh (Acid Green)
- Ferdinand Lipa (Acid Green)
- Harold Bataller (Acid Green)

- Armin A (Acid Green)
- Jack S (Web Solutions NYC)
- Kevin Bodwell
- James Cowie (SheroCommerce)
- Eren Karayigit (Improove)
- Gregor Pollak (Rocket Web)
- Евген Швец
- Nadezhda Glonyagina

All the best!

Joseph Maxwell

ADDITIONAL RESOURCES

Throughout this guide, I will reference many external websites and resources. Here are a few of my favorites:

- **Mage2.tv**: by Vinai Kopp. I highly recommend taking advantage of his training. It is well-thought-out material covering advanced subjects. To get 2 free weeks of access, utilize the promo code: STUDY-WELL. You will not regret subscribing! Besides, you are helping a fellow community member create more amazing materials.
- **IDE**: PHPStorm (by JetBrains). I use this every day that I am writing Magento code.

INTRODUCTION AND SETUP

This study guide is one of the most (or the most) concentrated source of information for advanced backend Magento development. If you hope to achieve this certification or just want to learn, you won't be disappointed.

Our goal is not to just tell you how to write code, but also point you to where you can dive in yourself AND show you how to do it. With this three-pronged approach, your willingness to learn is the only thing stopping you from hitting your goals.

Setup of your development website

We utilize the domain lc.commerce.site for all code examples in this book. Feel free to place the website at whatever development site domain you wish. Note there are some examples that you will need to change out the URL. Please download the practice code from your account area on SwiftOtter.com.

Setup of your development environment

I sincerely hope that you have a quality IDE (like PHPStorm) and your environment configured with Xdebug. The latter tool allows you to set breakpoints and step through the code, line-by-line, read variables and execute commands at run-time. Consider this a requirement to effectively completing this book.

Magento code

As stated above, you need to have license keys for Magento Commerce (formerly, Enterprise). If you work for a partner or are a Community Insider, you will have these keys available to you.

The test

This Professional Developer Plus test consists of 60 questions, requires a 62% score to pass, and has a time limit of 90 minutes. The questions are scenario-based, which ensures you have proper knowledge of the subject (instead of simply memorizing many subjects).

The test ensures you are extremely knowledgeable about Magento 2.2.

CONTENTS

1 MAGENTO ARCHITECTURE

4 questions

Professional Developer Plus

SWIFT OTTER SOLUTIONS

1.1 DETERMINE ADVANCED USES OF THE MAGENTO CONFIGURATION SYSTEM

Understand how to create a custom config file. Demonstrate an understanding of the Magento Configuration files framework

A core feature of Magento is its customizability (configuration). Magento's configuration is divided into areas: global (base), frontend, adminhtml, webapi_rest, webapi_soap and crontab. Configuration is loaded depending on the area chosen. By default, the global configuration is loaded and then the specific area is merged (or loaded) on top of the global configuration.

Magento includes a number of XML files that cover the majority of use cases in normal development. Module configuration is located in the `app/code/MyCompany/MyModule/etc/` directory.

Here are a few:

- `module.xml`: the initialization / configuration for a module. This file contains the `sequence` element that determines the module load order.
- `events.xml`: where event observers are declared (for example, executing code when the `sales_order_load_before` event is triggered).
- `di.xml`: where dependency injection is configured. This is where virtual types, plugins, and argument modifications are set.
- `config.xml`: where default configuration values are specified.
- `acl.xml`: where ACL nodes are configured.
- `[AREA]/routes.xml`: where the controller routes are initialized.
- `adminhtml/system.xml`: where new Store configuration fields are set up.

A core feature of Magento is the ability to override literally every configuration file.

Example:

- `app/code/Chapter1/ConfigLoadOrder` and `app/code/Chapter1/ConfigLoadOrder2`
- https://lc.commerce.site/chapter11/

Notice `app/etc/config.php`:

```
<?php
return [
    'modules' => [
        'Magento_Store' => 1,
        'Magento_Directory' => 1,
        'Magento_Eav' => 1,
        'Magento_Customer' => 1,
        'Chapter1_ConfigLoadOrder2' => 1,
        'Chapter1_ConfigLoadOrder' => 1,
…
```

`ConfigLoadOrder` contains a `sequence` node for `ConfigLoadOrder2`. This means that `ConfigLoadOrder` will be loaded after `ConfigLoadOrder2`.

Navigate to https://lc.commerce.site/chapter11/. Observe the difference as specified. Later modules override earlier modules regardless of the area specified.

Information:

- Create or extend configuration types
- Module configuration files

Understand how to create a custom config file with validation and a unique node that is overridden on merging

Magento also provides the capability to create custom configuration files with relative ease. This requires a number of pieces:

- A custom XML configuration file.
- A custom XSD schema file.
- A class for exposing the configuration data. While this isn't 100% necessary, it makes easier maintainability in retrieving the data.
- The above utilizes a class that implements `\Magento\Framework\Config\DataInterface` or extends `\Magento\Framework\Config\Data`. This provides the caching wrapper for the configuration data. (You can use a `virtualType` in `di.xml`.)
- A class that implements `\Magento\Framework\Config\ReaderInterface` or extends `Magento\Framework\Config\Reader\Filesystem`. (You can use a `virtualType` in `di.xml`.)
- A class that implements `\Magento\Framework\Config\ConverterInterface`. This class converts XML nodes into a PHP array.

Using an attribute as the unique identifier enables specifying the `idAttribute` in the `ReaderInterface`. The `idAttribute` is used to identify similar nodes for merging.

Example:

- `app/code/Chapter1/CustomConfig`
- https://lc.commerce.site/chapter11custom

We create a custom `discounts.xml` file in which we can create new `<discount/>` entities. Note the `name` attribute (this is specified in the `idAttributes` parameter in `di.xml`). This allows us to merge discounts for the same name.

Note that for the sake of simplicity, we are using `virtualTypes` as defined in `di.xml`. Also note that `virtualTypes` do not generate any classes in the `generated/` directory.

Practice:

- Create a new configuration type. Do not copy any code examples in this section but rather hand type. Pay special attention to the relationship between classes.

Examples:

- vendor/magento/module-email/Model/Template/Config.php
- vendor/magento/module-email/Model/Template/Config/Data.php
- vendor/magento/module-email/Model/Template/Config/Reader.php
- vendor/magento/module-email/Model/Template/Config/Converter.php
- vendor/magento/module-email/Model/Template/Config/FileResolver.php

Information:

- Create or extend configuration types
- Working with custom configuration files in Magento 2
- Magento 2 create custom xml config reader php class

Understand how to create a config file with a remote schema

Believe it or not, you can create a configuration file with a remote schema. Here is an example:

```
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://raw.githubusercontent.com/
magento/magento2/2.3-develop/lib/internal/Magento/Framework/App/
etc/routes.xsd">
    <router id="standard">
        <route id="chapter11remote" frontName="chapter11remote">
            <module name="Chapter1_RemoteXsd" />
        </route>
    </router>
</config>
```

Note that Magento may throw an error if an `https` URL is utilized because it can't properly negotiate SSL.

Example:

- `app/code/Chapter1/RemoteXsd`
- https://lc.commerce.site/chapter11remote

Practice:

- What happens when you change the noNamespaceSchemaLocation in `app/code/Chapter1/RemoteXsd/frontend/routes.xml` to be an invalid URL?

1.2 DEMONSTRATE AN ABILITY TO DESIGN COMPLEX CUSTOMIZATIONS USING PLUGINS AND DI.XML

Plugins sort order, plugin on plugin scenario, plugin debugging techniques.

Plugins are part of the foundational bedrock of customizing Magento. Plugins attach to `public` methods and can alter the input parameters, determine whether or not the method is executed or alter the output of a method. For classes that are built correctly and expose a reasonable number of `public` methods, this enables other developers (or even yourself) to build customizations in a modular format.

Note: on a complex application we recently built, we created a model that needed some complex validation before and after a calculation was run. The calculation was run in a class, and then, in the same module, we utilized `before` and `after` plugins to validate the input and output from the calculations.

There are three types of plugins:

`before` changes input parameters

These are returned as an array, in the order specified in the original method.

Example:

```
// original:

namespace MyCompany\MyModule\Model;

class MyRepository
{
    public function save(MyInterface $model, bool $isFlatTable)
    {
        // ...
    }
}

// plugin:
public function beforeSave(
    \MyCompany\MyModule\Model\MyRepository $subject,
    $model,
    $isFlatTable
) {
    return [$model, $isFlatTable];
}
```

`around` determines whether or not the original method is executed.

Magento documentation warns that `around` plugins can cause unintended consequences if the original method has more functionality than the developer anticipated, or if other plugins are required to successfully complete the task, or if the functionality breaks as Magento is upgraded in the future.

```
// around plugin for above MyRepository
public function aroundSave(
    \MyCompany\MyModule\Model\MyRepository $subject,
    callable $proceed,
    $model,
    $isFlatTable
) {
    return $proceed($model, $isFlatTable);
}
```

`after` alters the output of a method.

`after` also provides the parameters that were sent to the method itself, allowing for differing alterations based on these parameters.

```
// after plugin for above MyRepository
public function afterSave(
    \MyCompany\MyModule\Model\MyRepository $subject,
    $result,
    $model,
    $isFlatTable
) {
    // do something to $result
    return $result;
}
```

Sort order

Magento provides the capability to specify a `sortOrder` for each plugin. For most plugins it is unnecessary or even discouraged to set the `sortOrder`. However, if you, for example, need to change the input sent to another plugin (customizing a module from the Magento Marketplace), then `sortOrder` would have a valid use-case.

Typically, plugins are executed in a lowest-to-highest sort order. The complicating factor is when there are multiple plugins for the same method in a class.

When there are multiple plugins for the same method in a class, here is how it works: `before` plugins are executed lowest sort order, first, to highest sort order, last. `after` plugins are executed high sort order, first, to lowest sort order, last. Please read and study the example in DevDocs below and our example for this section.

Example:

- `app/code/Chapter1/Plugins`
- https://lc.commerce.site/chapter12plugins/example1

Further Reading:

- Prioritizing plugins

Plugin on Plugin

It is possible to attach a plugin to another plugin. This is done with normal configuration. One thing to note is that the modifying plugin has a different method signature than the original plugin.

Debugging

Here is a sample interceptor:

```
public function format(string $name)
{
    $pluginInfo = $this->pluginList->getNext($this->subjectType,'format');
    if (!$pluginInfo) {
        return parent::format($name);
    } else {
        return $this->___callPlugins('format', func_get_args(), $pluginInfo);
    }
}
```

To debug a plugin, the easiest place to start is by setting a breakpoint on the method that you are adding a plugin to. Check the call stack to see if an interceptor is there. If there is no interceptor, ensure you have properly configured the plugin in `di.xml` (remember `type` then `plugin`). If the interceptor is present, set a breakpoint in the interceptor to ensure that your plugin is being located. Step through the `___callPlugins` method in `\Magento\Framework\Interception\Interceptor` to follow the chain of commands.

Further Reading:

- Note: you can use the `n98-magerun2.phar config:data:di` tool.
- Plugins (Interceptors)

Example #1: app/code/Chapter1/Plugins

https://lc.commerce.site/chapter12plugins/example1
(remember, this is a development machine URL: please copy the URL and paste it into your URL bar and change the domain to whatever your development machine runs)

Study the example. Notice how the `before` in `Chapter1\Plugins\Plugins\Example1Both1` is executed after the `\Chapter1\Plugins\Plugins\Example1Around1`. This is because of the sort order. What happens when you change the execution order around?

Example #2: app/code/Chapter1/Plugins

https://lc.commerce.site/chapter12plugins/example2

This example proves how you can use a plugin to modify another plugin. The best use case for this is a third party module that contains a plugin that you need to modify. Attach your plugin to the other plugin (confusing, I know).

File: `app/code/Chapter1/Plugins/Model/Plugins/Example2Plugin2.php`

Practical experience:

- Locate the interceptor for `MyModel` class and set a breakpoint. Follow the execution through.

Demonstrate an understanding of virtual types, shared objects, object instantiation process, proxies, factories

Virtual types:

These are classes that are instantiated by the `ObjectManager` but have no concrete class located in `app/code` or in `generated`. Confusingly, a `type` is different in that the `type` adjusts existing classes whereas a `virtualType` is "creating" a new type of class. A `virtualType` must extend a concrete class type.

More concretely:

- If you wish to adjust the constructor parameters for an existing class, use the `type` declaration. These changes take effect for all class instantiations for that object in that area.

- If you wish to create an entirely new type of object that can be injected elsewhere, use the `virtualType` declaration. You need to then inject or utilize this class in XML configuration (in `di.xml` or as a block).

These are useful if the only necessary modifications to a class are the classes that are injected. Additionally, virtual types do generate code (in `generated`). As such, don't use them anywhere that does not use the Object Manager (for example, async bulk consumers).

Note that you cannot reference the specific `virtualType` in a class' constructor. Rather, your constructor should utilize the concrete class type that the `virtualType` extends. You also cannot write a class that inherits a virtual type.

Example for how to inject a virtual type:

```
<!-- app/code/MyCompany/MyModule/etc/di.xml -->
<virtualType name="MyVirtualType" type="Magento\Sales\
  OrderRepository">
    <!-- … -->
</virtualType>
<type name="MyCompany\MyModule\DoSomething">
    <arguments>
        <argument name="repository" xsi:type="object">
         MyVirtualType</argument>
    </arguments>
</type>
```

```
/** app/code/MyCompany/MyModule/DoSomething.php **/
public function __construct(\Magento\Sales\OrderRepository
$repository)
{
    /** … **./
}
```

Example:

- see `app/code/Chapter1/CustomConfig`.

Shared Objects:

The `ObjectManager` has two methods for getting an object: `create` and `get`.

`create`: loads a new version of the object every time. This is the method that factories call. Note that arguments can be passed to the object's constructor with this method.

`get`: loads a "cached" version of the object (you cannot pass arguments to the constructor because the object might have been already instantiated). The cache is a `private` variable in the `ObjectManager`. If the object is not saved, it is created and then saved. This is the method that is used for injecting objects into the constructor. Arguments cannot be passed to the object's constructor, as Magento has no way of knowing, beforehand, whether or not the object exists in the `ObjectManager` repository.

See: `generated/code/Magento/Catalog/Model/ProductFactory.php`

Practical Experience:

- Set a breakpoint on both the `create` and `get` methods. How is the actual object type determined?

Object Instantiation Process:

The interface that controls the instantiation of objects is: `\Magento\Framework\ObjectManager\FactoryInterface`. Inspecting this shows that there are several classes that extend `FactoryInterface`:

- `\Magento\Framework\ObjectManager\Factory\Dynamic\Production`
- `\Magento\Framework\ObjectManager\Factory\Dynamic\Developer`

Practical Experience:

- What are the differences between the production and developer factories? How does the ObjectManager get created in the first place (hint: `vendor/magento/framework/App/ObjectManagerFactory.php`)?

Proxies:

A proxy allows a class to be loaded at a later point in time. This is useful in the case of needing to import a class with a resource-intensive function, but it is not executed until it is needed.

```
/** generated/code/Magento/Customer/Model/Session/Proxy.php **/

public function loginById($customerId)
{
    return $this->_getSubject()->loginById($customerId);
}

protected function _getSubject()
{
    if (!$this->_subject) {
        $this->_subject = true === $this->_isShared
            ? $this->_objectManager->get($this->_instanceName)
            : $this->_objectManager->create($this->_instanceName);
    }
    return $this->_subject;
}
```

Additionally, proxies can be used to break a circular dependency loop. Note that you should never directly reference the `proxy` class in your constructor. Rather, use `di.xml`. With `di.xml`, you set the argument to be the `\Proxy` of the class to inject. `\Proxy` is automatically generated and extends the base class. The reason you should use `di.xml` is because PHP will throw errors if the class is not found (hasn't been generated yet).

Example:

```
// --- MyCompany\MyModule\Model\ThisClass --- //
public function __construct(AnotherClass $anotherClass) {}

// --- MyCompany/MyModule/etc/di.xml
<type name="MyCompany\MyModule\Model\ThisClass">
    <arguments>
        <argument name="anotherClass" xsi:type="object">MyCompany\
         MyModule\Model\AnotherClass\Proxy</argument>
    <arguments>
</type>
```

Further Reading:

- Proxies

Example:

- `app/code/Chapter1/Proxies`
- https://lc.commerce.site/chapter12proxies

The example is a contrived demonstration of how to resolve a circular dependency using proxies.

Practical experience:

- What error message do you get when you remove the `proxy` from `di.xml`?
- What class triggers that exception?

Factories:

Because all injected objects are shared, by default, there has to be a way to create a new object. As you know, the `new` keyword is pretty much forbidden in Magento development (except for throwing exceptions). If we need to create a new `Product` model, importing the `Product` model means that any other files that likewise import the `Product` model will be shared: updates made in one class `Product` model will be seen in another class' `Product` model—not good.

Here is an example of the `Product` factory:

```
public function create(array $data = array())
{
    return $this->_objectManager->create($this->_instanceName,
$data);
}
```

Note that you can send an array of arguments in the `$data` parameter. These arguments are sent to the constructor of the class to create:

```
$this->productFactory->create([
    'categoryRepository' => $this->getCategoryRepository()
]);
```

In the above example, the `$categoryRepository` parameter is specified by the input array instead of loaded through dependency injection.

Practical experience:

- How do parameters passed in through the `$data` parameter affect parameters specified in `di.xml`?
- Let's say you have a calculator class. You need to instantiate this calculator class from several locations. The calculator's constructor contains several scalar parameters. Create a factory to make instantiation of this class easier and typed.

How does an around plugin modify the plugin execution order?

This can be pretty tricky. If the `sortOrder` is specified, the `around` plugin is executed according to its assigned sort order. This is easy to understand for the "before" part of the `around` (inside the `around`, you should call the `proceed` callable—your code can be executed on either side).

If there are plugins with a higher sort order than the `around` plugin's sort order, these plugins are run **before** the "after" part of the `around` is executed. If you are not able to follow this, please reference the additional instruction available from SwiftOtter.com.

Practical experience:

- See example #1 from 1.2. Change the sort order of the `around` plugin. How does this affect the output in https://lc.commerce.site/chapter12plugins/example1?
- `vendor/magento/module-customer/etc/di.xml` has a plugin named `transactionWrapper`. Create a plugin to run some code after the target method's code is run but before the transaction is committed to the database.

How do you debug a plugin that is not executed?

- Ensure that the plugin exists `di.xml` and:
  - That it is specified for the correct area.
  - That it is not `disabled` by another module.
- That the method fits within the requirements as specified in the documentation.
- Set a breakpoint in the targeted method and see if an `Interceptor` class has been generated.

Practical experience:

- Module: `app/code/Chapter1/Challenge12PluginDebug`
- Url: https://lc.commerce.site/chapter12pluginchallenge/
- Get the value to match the expected result.

Demonstrate Plugin on Plugin examples

See the SwiftOtter additional training for this study guide for demonstrations on how this works.

Which classes are instantiated outside of the ObjectManager so they cannot be customizing using di.xml?

- Any classes for Composer dependencies.
- Bootstrapping classes, for example `Magento\Framework\App\Http`

Further Reading:

- Magento application initialization and bootstrap

Demonstrate a use case for a virtual type (different instances of a class with a different set of arguments)

See above for examples. The companion pack for this study guide also has additional information.

Further Reading:

- `\Magento\Framework\Event\Manager::dispatch`
- `\Magento\Framework\Event\Invoker\InvokerDefault::dispatch`
- `\Magento\Framework\Event\Invoker\InvokerDefault::_callObserverMethod`

1.3 DEMONSTRATE UNDERSTANDING OF MAGENTO EVENTS PROCESSING

Demonstrate an understanding of the events processing flow.

- `\Magento\Framework\Event\Manager` is injected.
- `dispatch` is called.
- Observers are iterated
  - A new event and observer are created for each event triggered.
    - If you set a new value on the event's `data` property, that value will be discarded. My understanding is that the goal is to discourage leveraging events for what plugins are intended to do.
  - An invoker is called (`\Magento\Framework\Event\Invoker\InvokerDefault`) which ensures that the observer is not disabled, and determines whether or not to use a shared instance.
- Observers are called
  - Observers must implement `\Magento\Framework\Event\ObserverInterface` (developer mode triggers an exception, production mode skips execution).

```
<event name="catalog_product_save_after">
    <observer name="RefreshCaches" instance="MyCompany\MyModule\
      Observers\RefreshCaches"/>
</event>
```

Practical experience:

- Create a new observer for the `catalog_controller_category_init_after` event.
- If an event is triggered with a string, are you able to modify it?
- Now, disable the event from another module.

Influence of Staging on the event processing. What is a modification of the event processing mechanism introduced by the staging module? (Commerce)

The Staging module changes the default implementation for `Magento\Framework\Event\ManagerInterface` to `Magento\Staging\Model\Event\Manager` (actually, this module's proxy). The Staging event manager introduces new functionality to prevent events or observers from firing when the page that is being viewed is in preview mode.

Example:

- `vendor/magento/module-catalog-url-rewrite-staging/etc/adminhtml/di.xml`

At this time, there are no banned observers or events for the frontend.

Practical experience:

- Ban an event and an observer on the frontend.
- Review the `VersionManager` in `vendor/magento/module-staging/Model/VersionManager.php`.
    - What happens if there is no database connected (in a build environment)?

1.4 DEMONSTRATE AN ABILITY TO USE THE MAGENTO COMMAND-LINE INTERFACE

Create a new CLI command, emulate different areas within it. Create a new CLI-command, configure it in di.xml, add optional/required options/keys.

Creating a new CLI command is quite simple:

- Add CLI command to `Magento\Framework\Console\CommandListInterface` via `di.xml`. (example)
- Create a class (generally in your modules' `Console` directory).
- Make the class extend `Symfony\Component\Console\Command\Command`.

The `\Magento\Framework\App\State` controls the state for the application. By default, the app state is not set and your application will throw an error if it accesses any functionality that requires a state to be set. It is up to the developer to change this to whatever area is applicable for the actions that need to be undertaken.

Options vs Arguments:

Remember this:

- Options have a flag (`bin/magento do:something --admin-email="myemail@swiftotter.com"`)
- Arguments have no flag (`bin/magento do:something myemail@swiftotter.com`)

Most console commands accept options. A few offer arguments. In the example listed below, the argument is an array (separated by spaces) of languages:

```
bin/magento setup:static-content:deploy --content-version=31 en_US de_DE
```

Example:

- `app/code/Chapter1/CLI`
- `bin/magento customer:create`

Run:

- `bin/magento customer:create --email=myemail@swiftotter.com`
- `bin/magento customer:create "Joseph Maxwell"`

Practical Experience:

- Create a new CLI command to execute:
  - `bin/magento mycommand:run --setup=1 --area=frontend` (or `adminhtml`)
  - Command must automatically switch areas.

Environment specification using Area class

- Inject `\Magento\Framework\App\State`.
- Call `$this->state->setAreaCode()`.

Environment emulation for a section of code

If you are in an area, say the `adminhtml` area, and you need to switch to the `frontend` area, as we see above, you cannot just "switch" the area. A time when this is used is in the product alerts (`vendor/magento/module-product-alert/Model/Email.php`).

Important Notes

- Calling `getAreaCode()` when no area code is set throws an exception.
- Calling `setAreaCode()` when an area code is already set throws an exception.
- **Example:** app/code/Chapter1/CLI
  - `bin/magento trigger:event`

The "magic" method is `emulateAreaCode()`. This method accepts two parameters, the new area and a callback that contains the code to execute while in emulation mode. Note that emulation does not change all aspects of the system: the Magento scope remains the same. When calling `setAreaCode` the scope is initially set, but emulation does not change it.

The other way to emulate is through the `\Magento\Store\Model\App\Emulation` class. This class does not use a callback but rather the `startEnvironmentEmulation()` and `stopEnvironmentEmulation()`.

Example:

- `app/code/Chapter1/CLI`
- `bin/magento trigger:event:area`

Practical experience:

- Determine what features do change when emulating an environment.

Further Reading:

- Emulating Areas in Magento 2
- Magento2 Store Emulation

2 MAGENTO UI

4 questions

Professional Developer Plus

SWIFT OTTER SOLUTIONS

2.1 DEMONSTRATE UNDERSTANDING UICOMPONENTS ARCHITECTURE

Warning:

I am barely able to scratch the surface of uiComponents. It is imperative that you study this section, work through the practical examples, and understand the sample code before attempting the test.

Note:

Note: with uiComponents, everything is a component. That means every uiComponent extends `uiElement` (`vendor/magento/module-ui/view/base/web/js/lib/core/element/element.js`). Every uiComponent is a child of another uiComponent. The custom uiComponents you create are children of uiComponents. This is fundamental knowledge.

UiComponent workflow, initialization, execution, configuration structure, data loading process

Workflow:

UiComponents represent a semi-intuitive means of creating admin interfaces. While they appear complex on the surface, one can create an extensible interface with only a few files.

Beyond the standard requirements to serve an HTML page, these files are required for a UiComponent:

- UiComponent XML definition: `view/adminhtml/ui_component/ui_component_name.xml`
- (nothing else)

As you can see, there are very few files required. While some of the uiComponent XML structure makes little sense, drawing up even basic uiComponents is quite fast.

Beyond the basics, here are some helpful ancillary files that you will probably create:

- Actions column which will provide the actions drop down menu column. Extend `\Magento\Ui\Component\Listing\Columns\Column`.
- Data provider to format, customize, or retrieve data. Extend `\Magento\Framework\View\Element\UiComponent\DataProvider\DataProvider`.

Note that you also must tell the DataProvider collection factory that there are more collections available. This usually happens in `di.xml`, like:

```
<type name="Magento\Framework\View\Element\UiComponent\
DataProvider\CollectionFactory">
    <arguments>
        <argument name="collections" xsi:type="array">
            <item name="custom_product_grid_data_source"
               xsi:type="string">CustomProductGridCollection</item>
</argument>
    </arguments>
</type>
```

The `name` above is originally specified in the UiComponent `data.js_config.provider` and `data.js_config.deps` nodes.

To load a UiComponent, add the `<uiComponent name="sample_ui_component"/>` into your layout XML. Note that this must be placed within another block or container, like:

```
<referenceContainer name="content">
    <uiComponent name="custom_product_form"/>
</referenceContainer>
```

Often, the host layout XML files are quite empty, as they just contain the uiComponent instruction.

Execution:

The uiComponent reader model (`vendor/magento/framework/View/Layout/Reader/UiComponent.php`) is responsible for instantiating PHP classes for the layout XML instructions. Specifically, the `interpret` method:

- Schedules the addition into the layout structure.
    - Formulates a path (ex. `root/backend.page/page.wrapper/page.content/page.main.container/admin.scope.col.wrap/main.col/content/custom_product_grid`) for the new component.
- Standardizes visibility conditions for:
    - `ifconfig` and `aclResource` attributes.
    - `visibilityCondition` child node.
- Up until this point, no uiComponent configuration has been loaded.

- `$this->uiConfigFactory` loads the uiComponent.

Next, the uiComponent is converted into a block. This happens in `vendor/magento/framework/View/Layout/Generator/UiComponent.php`.
It is important to note that the interface for a uiComponent is `\Magento\Framework\View\Element\UiComponent\ContainerInterface`.
The concrete implementation is `\Magento\Ui\Component\Wrapper\UiComponent`.

- All child blocks are merged into the uiComponent to be rendered.
- The base `.xhtml` template that is rendered is `vendor/magento/module-ui/view/base/ui_component/templates/[component type]/default.xhtml`.
- The template engine is `\Magento\Framework\View\TemplateEngine\Xhtml`.
- `\Magento\Ui\TemplateEngine\Xhtml\Result::__toString()` is called with the express purpose of rendering the uiComponent.
- The compiler builds the representation of the `.xhtml` file.
- `appendLayoutConfiguration()` is called which converts the uiComponent into a JSON representation (see `wrapContent()`).

The original uiComponent rendering contains data to be displayed. Yet, that data is not displayed until a web request is triggered. Keep reading…

Data loading:

Once the Javascript files on the page have loaded, the uiComponent triggers a request to: `vendor/magento/module-ui/Controller/Adminhtml/Index/Render.php`. This controller is specified with the `listing/dataSource/dataProvider/data/config/update_url` path (note that the latter items are in `argument` and `item` nodes).

So, how does the render class know where the data is? Easy: the request must specify a `namespace` parameter. The `namespace` is really the name of the uiComponent. Add `.xml` to it and search through your codebase for a file with that name (ex. `custom_product_grid.xml`), and you will find the uiComponent.

In the `render` controller:

- The ACL resource is validated.
- All child components are prepared (`\Magento\Ui\Component\AbstractComponent::prepare`, a `public` method).
- If `componentJson` is a parameter in the URL, the entire component will be returned. Otherwise, just the data. Note that the entire component is still loaded.

Forms data:

The default data wrapper name is `general`. Inside the `general` array, you will find the data for your form.

In the example, our form has multiple tabs. Please note that the form will not save as this is simply a demonstration of uiComponent forms. Pay special attention to how the data is mapped.

To better understand how this works, run these commands in your browser's developer tools:

```
require('uiRegistry').get('index = name');
```

Notice the name field's `links`:

```
links:
    value: "custom_product_form.custom_product_form_data
         source:data.general.name"
```

You can then call:

```
require('uiRegistry').get('custom_product_form.custom_product_
form_data_source').data
```

to see the data object that your component is hydrated with.

It is important to note that you can insert regular layout instructions into your uiComponent. For example, in the Customer Form uiComponent (`vendor/magento/module-customer/view/base/ui_component/customer_form.xml`), you will find the following code. Notice that the **`htmlContent`** is wrapping the regular layout XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<form xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:module:Magento_Ui:etc/
ui_configuration.xsd">
    <!-- … -->
    <htmlContent name="customer_edit_tab_view_content">
        <block class="Magento\Customer\Block\Adminhtml\Edit\
         Tab\View" name="customer_edit_tab_view"
         template="Magento_Customer::tab/view.phtml">
            <arguments>
                <argument name="sort_order" xsi:type="number">10
                    </argument>
                <argument name="tab_label" xsi:type="string"
                    translate="true">Customer View</argument>
            </arguments>
            <block class="Magento\Customer\Block\Adminhtml\Edit\
               Tab\View\PersonalInfo" name="personal_info"
               template="Magento_Customer::tab/view/personal_info.
               phtml"/>
        </block>
    </htmlContent>
    <!-- … -->
</form>
```

Grid examples:

- `vendor/magento/module-customer/view/adminhtml/ui_component/customer_listing.xml`
- `vendor/magento/module-catalog/view/adminhtml/ui_component/product_listing.xml`

Form examples:

- `vendor/magento/module-customer/view/base/ui_component/customer_form.xml`
- `vendor/magento/module-catalog/view/adminhtml/ui_component/product_form.xml`

Note:

Note that an example and more details of this are available in the SwiftOtter learning kit.

Practical experience:

- Set a breakpoint in `__construct` in `vendor/magento/framework/View/Element/UiComponent/DataProvider/DataProvider.php` (or your custom `DataProvider`) and observe how the `dataProvider` node in the uiComponent configuration matches the parameters. Doesn't this open further configuration possibilities?
- In a DataProvider, what public methods are available for plugging into?
- Set a breakpoint in `vendor/magento/framework/View/Layout/Reader/UiComponent.php`, and look at the call stack.

- Navigate to the catalog product page and set a breakpoint in `vendor/magento/framework/View/Layout.php:renderNonCachedElement`. Follow the chain of instructions.
- Prevent the `prefix` attribute from appearing in the `customer_form.xml` uiComponent.
- Now, prevent the `disable_auto_group_change` in the `customer_form.xml` uiComponent.
    - You will need to create a plugin in `\Magento\Customer\Model\Customer\DataProvider`. The reason for this is that the uiComponent configuration is overridden by the metadata coming from the database.
    - One important note is that the uiComponent configuration is cached in the Configuration XML. However, the overrides are not cached and are loaded through the above DataProvider.
    - Should you run into issues with changing a uiComponent, feel free to modify the DataProvider.
- Become familiar with the classes in `module-ui`.
    - What is the purpose of the classes in `Component/`?
    - Where are some places that the `\Magento\Ui\DataProvider\Modifier\ModifierInterface` is used?
- Set a breakpoint in `vendor/magento/module-ui/Controller/Adminhtml/Index/Render.php:execute` and navigate either to our custom uiComponent or to any page that has a uiComponent (product listing, customer listing).

Further Reading:

- https://www.mage2.tv/ has a large amount of detailed instruction on uiComponents.

Retrieving a UiComponent's instance from the uiRegistry

To access a uiComponent's Javascript implementation, navigate to Catalog > Products (custom), open your browser's developer tools and run:

```
require('uiRegistry').get('custom_product_grid.custom_product_grid')
```

You can access the base component by first calling the name of the uiComponent (`custom_product_grid`) and then the name of the columns specified (`custom_product_grid`), separated by a `.`.

The `dataSource` is also a uiComponent. This is accessed in the same way. In our example grid uiComponent (`app/code/Chapter2/UiDemonstration/view/adminhtml/ui_component/custom_product_grid.xml`), you can access the `dataSource` by calling:

```
require('uiRegistry').get('custom_product_grid.custom_product_grid_data_source')
```

You can also get an item with the `index` command, instead of using the full component name:

```
require('uiRegistry').get('index = name');
```

Further Reading:

- uiRegistry

Understand the difference between executing a data provider component and loading data

The big question is how does data get hydrated into our components? The two most-used containers are forms and grids, so we will talk about those separately.

By default, data providers are responsible for storing and managing the data saving process. Confusingly, a data storage component loads the data from the server.

Forms:

Provider: `vendor/magento/module-ui/view/base/web/js/form/provider.js`. The data is specified in the initialization JSON. It is then accessible in the data source component.

Grids:

Provider: `vendor/magento/module-ui/view/base/web/js/grid/provider.js`

Data Storage: `vendor/magento/module-ui/view/base/web/js/grid/data-storage.js`

The data storage component is responsible for loading data. Note that the initialization JSON does have data specified, but that data is quickly discarded.

Practical experience:

- Set a breakpoint in `vendor/magento/module-ui/view/base/web/js/grid/data-storage.js` in Admin > Catalog > Products (custom) to see how the data is loaded in.
- Set a breakpoint in `\Magento\Ui\Controller\Index\Render::execute` to understand how data is loaded in the renderer path.

Describe the role of UiComponent PHP classes

The purpose of the UiComponent PHP classes is to interface between XML configuration files and the JSON output. These classes are why you can utilize a `fieldset` tag in your uiComponent.

The `prepare` method in (`vendor/magento/module-ui/Component/AbstractComponent.php`) contains the majority of global customizations.

Further Reading:

- `vendor/magento/module-ui`

Practical Experience:

- Set a breakpoint in `\Magento\Ui\Component\AbstractComponent::__construct`. Understand how the values that come into the `$data` argument originate in the XML uiComponent definition.
- With this understanding, examine `\Magento\Ui\Component\AbstractComponent` to determine how you can extend a uiComponent.
- Determine what is the purpose of `observers` (as seen in `\Magento\Ui\Component\AbstractComponent`).

Understand the uiClass instance, extending uiComponent

First, `uiClass` is not an actual Javascript file in the Magento codebase. Instead, it is an alias with the association being found in `vendor/magento/module-ui/view/base/requirejs-config.js`.

uiClass: `vendor/magento/module-ui/view/base/web/js/lib/core/class.js`

\/

uiElement: `vendor/magento/module-ui/view/base/web/js/lib/core/element/element.js`

\/

uiComponent: `vendor/magento/module-ui/view/base/web/js/lib/core/collection.js`

uiClass:

This is the basic building block for any uiComponent or related element. This provides the basic functions necessary to propagate and extend.

uiElement:

This represents a basic element—one that does not need the ability to manage children.

Example:

- `vendor/magento/module-ui/view/base/web/js/grid/resize.js`

uiComponent:

Most uiComponents extend this class. This has the capability of storing and rendering multiple children (similar to `blocks` in Magento layout).

Example:

- `vendor/magento/module-ui/view/base/web/js/grid/toolbar.js`

2.2 DEMONSTRATE ADVANCED USE OF MAGENTO LAYOUTS

Non-standard layouts, custom handles, debugging layouts

Magento layout is powerful, and we will delve into the more subtle intricacies of this part of the system.

A layout must be defined in two places in a module:

- `view/frontend/layouts.xml`: this specifies the `id` and the human-readable name of the layout. The example in our sample module is `my-custom-layout`.

- `view/frontend/page_layout/[layout id].xml`: this sets up the initial layout instructions. The minimum is likely to be `<update handle="1column"/>` or something similar.

Further Reading:

- Layout File Types
- `vendor/magento/magento-theme`

Custom handles are usually easy to add (an exception being a category page). Our controller (`app/code/Chapter2/Layout/Controller/Index/Index.php`) contains an example of how to do this.

There are a few techniques to debugging problems with layouts.

- Are you sure that your module is enabled?
- Set a breakpoint in `vendor/magento/framework/View/Model/Layout/Merge.php:_loadFileLayoutUpdatesXml` to see if your module is loaded.
- Are you sure that you have the correct layout XML file? It is easy to confuse the router id and the router front name.
- Set a breakpoint on your block or view model's `class` line. This will tell you when the autoloader includes your file, and you can continue until you locate where Magento skips over your block.

Example:

- https://lc.commerce.site/index.php/chapter22layout/ (handles)
- `app/code/Chapter2/Layout`
- https://lc.commerce.site/index.php/chapter22layout/index/index/xml/1 (displaying XML layout)

Further Reading:

- Adding Custom Layout Handles in Magento 2
- How to add custom layout handles programatically for Category View in Magento 2

Add a custom handle, obtain a list of handles loaded for a page

See above example.

Obtain the layout XML for a page

See above example.

Containers elements with a wrapping DIV tag

In your layout XML file, add:

```
<container name="my.container" htmlTag="div">
    <!-- … -->
</container>
```

Note that you can also specify several other attributes:

- `htmlId`
- `htmlClass`

Further Reading:

- Container Structure

Dynamically modify the layout tree

The `\Magento\Framework\View\LayoutInterface` class contains the majority of methods to modify the layout tree. With this, you can create, access, and delete elements.

Because `LayoutInterface` extends `\Magento\Framework\Simplexml\Config`, its core is literally an XML node storage system. In the `Config` class, you will find methods like `getXmlString()`, `loadFile()`, `extend`, etc.

One thing to note is that many methods in `\Magento\Framework\View\Layout` (the implementation for `LayoutInterface`) call the protected `build()` method, which ultimately calls `\Magento\Framework\View\Layout\Builder`. This method loads layout XML files and uses that information to spawn the PHP classes that power the rendering of the frontend.

A side effect of this is that once `build()` has been called, any additional layout handles added will have no effect on the Layout XML that is loaded. As such, it is imperative to add layout handles as early as possible.

See the example for one way to do this.

Example:

- https://lc.commerce.site/index.php/chapter22layout/index/addblock
- `app/code/Chapter2/Layout/Controller/Index/AddBlock.php`

2.3 DEMONSTRATE AN ABILITY TO OPERATE WITH MAGENTO BLOCKS AND TEMPLATES

Block caching, fallback debugging, email templates, translations

Block caching

By default, blocks are not cached (unless a block is included in a parent block's cache). To cache a block, you must specify the `cache_lifetime` key in the block's `_data` variable and have the `block_html` cache enabled. Because Magento 2 has integrated full page caching so well, I believe that block caching is less relevant than it used to be.

Note that you can disable the entire page's caching by specifying `cacheable="false"` in a block, like:

```
<block name="my.test.block" template="MyCompany_MyModule::test-
block.phtml" cacheable="false">
    <arguments>
        <argument name="customer_info" xsi:type="object">MyCompany\
         MyModule\ViewModel\CustomerInfo</argument>
    </arguments>
</block>
```

There are few use cases where this makes sense. Unfortunately, there are modules in circulation out there that misuse this directive and end up excluding half or more of a website from being full-page cached.

You can use it to prevent customer data from being cached. However, in these cases, it is usually better to use AJAX requests (and the API) to load the respective data. You can also utilize Magento's ESI implementation called Private Content. You can also search the Magento codebase to find the cases where this is used.

Further Reading:

- Page Caching

This directive is used in `\Magento\Framework\View\Layout::isCacheable`. This method searches the XML layout tree to find any instances of `cacheable="false"`. If any are found, the method returns false.

If you are instantiating a `LayoutInterface`, you can pass `['cacheable' => false]` array as a constructor argument to prevent caching.

Example:

- `\Magento\Email\Model\Template\Filter::emulateAreaCallback`

Practical experience:

- Step through `\Magento\Framework\View\Element\AbstractBlock::_loadCache`

Further Reading:

- Private Content

Fallback debugging

The core class that renders templates is: `\Magento\Framework\View\Element\Template`. The `getTemplateFile()` method is responsible for locating the specific file to be used in this situation.

Remember that as of Magento 2.2.1, the aforementioned class is the default class for a block. Specifying a block's class is no longer required as Magento assumes this default.

Further Reading:

- Layout Instructions

To debug what is happening with the template override:

- Check to ensure that the path of the template override is correct:

```
app/design/frontend/MyCompany/MyTheme/Module_Name/templates/path/to/template.phtml
```

- Make sure your theme is enabled.
- Set a breakpoint in `\Magento\Framework\View\Element\Template::getTemplateFile` and step into `$this->resolver`.

Example:

- https://lc.commerce.site/index.php/chapter23blocks
- `app/code/Chapter2/Blocks`

Practical experience:

- Set a breakpoint in `\Chapter2\Blocks\Block\ExampleBlock`. Step through line-by-line to understand how the template resolver works.
- Create a module to change the template for the Luma theme's menu.
- Examine the `\Magento\Framework\View\Design\Fallback\RulePool::getRule` method (and associated methods).
- Step through `\Magento\Framework\View\Design\Fallback\Rule\Theme::getPatternDirs`.

Email templates

Email template fallbacks are supported in the same way that regular block templates are supported. Instead of email templates being located in the `view/[area]/templates/` directory, they are found in `view/[area]/email`.

Further Reading:

- Customize Email Templates

Translations

A fundamental and core feature of Magento is the ability to provide translations for every string that is rendered on the website.

Translations are stored in several places:

- In the database: `translation` table.
- In CSV files in your module's `/i18n` directory.
- In CSV files in your theme's `/i18n` directory.
- In CSV files inside the `app/i18n` directory.

Magento can generate a language package for your module:

```
bin/magento i18n:collect-phrases app/code/Chapter2/Layout
```

The above command will generate a translation file for your module that you can then send to someone to translate.

Further Reading:

- Translations Overview
- Generate a translation dictionary

Cache all instances of the block, specific instance

This is accomplished by setting the `cache_lifetime` argument for a block. You can specify this for each block in an argument. However, to make all instances of a block cached, you will have to extend `\Magento\Framework\View\Element\Template` and utilize this block.

Keep in mind that a `virtualType` would be a perfect fit here: define the virtual type and set the cache_lifetime. Then, utilize that virtual type in layout XML. Note that you cannot implement plugins on a virtual type, but instances, where this is necessary, are very rare.

Example:

- https://lc.commerce.site/index.php/chapter23blocks
- `app/code/Chapter2/Blocks`
- Ensure `full_page` cache is off, but `block_html` cache is on. Then, refresh the page several times and note how the upper time does not change, but the lower time does.

Assign an object to the email template. Render different images depending on a locale in the email template

```
$transport = $this->transportBuilder->setTemplateIdentifier($templateId)
    ->setTemplateOptions(['area' => 'frontend', 'store' => $storeId])
    ->setTemplateVars(['parameters' => 'for', 'the' => 'template'])
    ->setFrom($from)
    ->addTo($email, ''Joseph Maxwell')
    ->getTransport();

$transport->sendMessage();
```

The `setTemplateVars` method applies objects to the email template. In the above example, calling `getTransport` generates the email message body (see `\Magento\Framework\Mail\Template\TransportBuilder::prepareMessage`).

If you want to intercept the variables before they are applied to the email message, the easiest place is likely: `\Magento\Email\Model\AbstractTemplate::getProcessedTemplate`.

Render different images depending on a locale in the email template

See `\Magento\Newsletter\Model\TemplateTest::testGetProcessedTemplateArea` for an example of how this works.

```
{{view url="Magento_Theme::favicon.ico"}}
```

Further Reading:

- `\Magento\Customer\Model\EmailNotification::passwordReset`
- `\Magento\Email\Model\Template\Filter::viewDirective`
- `\Magento\Framework\View\Asset\Repository::getUrlWithParams`

Identify the location of block instantiation

Practical experience:

- Hopefully, you easily know **how** to find this by now: if you don't, here is the answer. Set a breakpoint on the `class Template` line in `\Magento\Framework\View\Element\Template` and reload a Magento frontend page.
- Look back through the call stack to determine where the block is instantiated. Note that everything down to the ObjectManager is standard Magento instantiation. The next class or two would be considered the answer.

Answer:

`\Magento\Framework\View\Element\BlockFactory::createBlock`

`\Magento\Framework\View\Layout\Generator\Block::getBlockInstance`

`\Magento\Framework\View\Layout\Generator\Block::createBlock`

Print out all places where Magento looks for a template

Example for template `Chapter2_Blocks::override.phtml`:

- `app/design/frontend/SwiftOtter/Flex/Chapter2_Blocks/templates`
- `vendor/magento/theme-frontend-blank/Chapter2_Blocks/templates`
- `app/code/Chapter2/Blocks/view/frontend/templates`
- `app/code/Chapter2/Blocks/view/base/templates`

Source: `\Magento\Framework\View\Design\FileResolution\Fallback\Resolver\Simple::resolveFile`

Demonstrate an understanding of different types of translations working together (inline, phrase in JavaScript code, CSV file)

Inline

This happens with the now-famous `__()` method. This method is available anywhere (from `vendor/magento/framework/Phrase/__.php`).

```
__('My %1 phrase (of %2 phrases) to translate.', 'first', 'many');
```

The above method returns a `\Magento\Framework\Phrase` object which translates the input phrase when the class is cast to a `string`.

By default, there are three translation renders. Each one receives the output from the previous renderer:

- `\Magento\Framework\Phrase\Renderer\Translate`: this class does what we would expect from a translation renderer. It translates the input text. Ultimately, the translation is loaded from `\Magento\Framework\Translate`.
- `\Magento\Framework\Phrase\Renderer\Placeholder` This one parses the placeholders (above, `%1`) and replaces them in the translation.
- `\Magento\Framework\Phrase\Renderer\Inline` If inline translation is enabled, this returns output such that this phrase is capable of being translated on the frontend.

Phrase in JavaScript code

The `mage/translate` class is responsible for providing translations. Step through `lib/web/mage/translate.js` to observe how this works.

Example:

- `app/code/Chapter2/Blocks/view/frontend/templates/translation.phtml`

CSV file

These are the heart of developer-supplied translations—CSV files that are easily transferred from translator to developer allow for quick translation updates.

Practical experience:

- Create a new translation file in your module. Using `en_US.csv` as the default works fine.
- Manually add a new translation into the `translation` table.
- Create a template that utilizes the `__()` translation method and set a breakpoint there to understand how translation works.
- Set breakpoints in `app/code/Chapter2/Blocks/view/frontend/templates/translation.phtml` to understand how this logic works.

3 WORKING WITH DATABASES

8 questions

Professional Developer Plus

SWIFT OTTER SOLUTIONS

3.1 DEMONSTRATE UNDERSTANDING OF THE ARCHITECTURAL LAYERS OF THE DATABASE ACCESS CLASSES, INCLUDING MODELS, REPOSITORIES, AND DATA MAPPERS

Models, resource models, and collections in Magento, their impact on performance. Repositories, SearchCriteria, WebAPI, WebAPI access, extension attributes

Example website:

- https://lc.commerce.site/index.php/chapter3database

Models

Models represent a row of data from the database. Each row, whether it is loaded through a collection or a resource model, is represented by the model.

Custom models typically extend `\Magento\Framework\Model\AbstractModel`.

Models that need extension attributes (most Magento modules, 3rd-party extensions) extend `\Magento\Framework\Model\AbstractExtensibleModel`.

Few, if any, true Magento data models will utilize the constructor for dependency injection. Best practice is to have data models as a storage container for storing data so that it is easily utilized.

Note:

When creating an API that returns a model, you do not have to return "the" data model that was loaded from the database. Feel free to create a new model and inject "the" data model that was loaded. The new model acts as a proxy and modifies output values. Another technique is to utilize two interfaces: one for the internal application and one for the external application. We will discuss this more later.

Example:

- `app/code/Chapter3/Database/Model/Discount.php`

Resource models

This is where stuff gets done. The resource model is what communicates with the database. This is where you should put most custom queries. The resource model is hooked up to the model, collection, and repository.

Resource models extend:
`\Magento\Framework\Model\ResourceModel\Db\AbstractDb`

Example:

- `app/code/Chapter3/Database/Model/ResourceModel/Discount.php`
- Notice the method to load all the discount amounts.

Collections

A collection loads multiple entities from the database. Their use is greatly diminished with the advent of repositories (although both share very similar

purposes). One advantage of a collection over a repository for EAV entities (products, categories, customers) is that you can specify what data is loaded in a collection. You have little to no control over this in a repository—all attributes are loaded. As such, for EAV entities, a collection represents a much faster way to load data. For the API, this really doesn't matter, and a repository is better suited.

Collections extend:

`\Magento\Framework\Model\ResourceModel\Db\Collection\AbstractCollection`

Example:

- `app/code/Chapter3/Database/Model/ResourceModel/Discount/Collection.php`
- Notice how the `getAllDiscountsOver10()` method filters a custom set of models.

Repositories

At a basic level, repositories are a combination of resource models and collections. They save individual models but also have a defined way to load a list of models (`SearchCriteriaInterface`). Their original purpose, in my understanding, was to facilitate API interaction, but they have grown to be useful in every part of Magento.

There are no abstract repository classes. To be available on the frontend, repositories must implement a service contract (stored in the `/Api` directory). While it would be worthless, you could build a repository that exposes no public methods. The repository only has the number of methods and the amount of functionality that you give to it. While many Magento repositories are full-

featured, you do not have to write a repository that uses all these features. The repository we are creating for our discount model is based on the `\Magento\Cms\Model\BlockRepository`.

Example:

- `app/code/Chapter3/Database/Model/DiscountRepository.php`

To create a repository, you must:

- Build a service contract interface for the repository.
- Create a repository model that implements this service contract.
- Create an interface to make the `\Magento\Framework\Api\SearchResultsInterface` type-friendly to your class (if you wish to build a `getList` method).
- Inject:
    - Resource model (to perform save / delete operations)
    - Model factory (initialize before the resource model hydrates)
    - Collection factory (for returning a list of results)
    - `\Magento\Framework\Api\SearchCriteria\CollectionProcessorInterface` to convert a `\Magento\Framework\Api\SearchCriteriaInterface` into filters that a collection understands.
- Write it up. See the example for more details.

To enable the repository to be accessed via an API, you must:

- Create `etc/webapi.xml`
- Create `etc/acl.xml` (if it is necessary to restrict access to the API calls)

SearchCriteria

This is an ingenuitive and new way to locate records. You attach a long url parameter to a request (that accepts `SearchCriteriaInterface`, such as a `getList` method).

In our sample application, note that to get all items, simply pass a `null` value for the `criteria` parameter.

```
const url = '/rest/V1/discount/?' +
  'searchCriteria[filter_groups][0][filters][0][field]=' + field + '&' +
  'searchCriteria[filter_groups][0][filters][0][value]=' + value + '&' +
  'searchCriteria[filter_groups][0][filters][0][condition_type]=' + conditionType;


return fetch(url, {
  credentials: 'include',
  headers: {
    'Accept': 'application/json, text/plain, */*',
    'Content-Type': 'application/json'
  }
}).then(function(response) { return response.json() });
```

Example:

- `app/code/Chapter3/Database/view/frontend/web/js/api.js`

Practical experience:

- Create an API request (curl, fetch, or PHP) to request `/rest/V1/products` (in the admin panel, so keep sessions in mind). Ensure you are familiar with creating filter groups and filters.

Further Reading:

- Search using REST endpoints

WebAPI, WebAPI access

The web API is a powerful system for allowing machine interaction with the website.

Here are a couple of concepts:

- API routes and endpoints are defined in `etc/webapi.xml`.
- APIs rely on service contracts. The types must be clearly specified in the docblock.
- API configuration is saved in the cache: any updates to `etc/webapi.xml` require a refresh of the `config_webservice` cache.
- APIs do not allow you to set cookies. If this is required (like logging a user in), you must use a controller, which can be accompanied by a JSON response.
- You can restrict access with the `<resources/>` node:
  - `<resource ref="anonymous"/>`: anyone can access.
  - `<resource ref="self"/>`: authenticated by the session. At this point, only the frontend session works with this.

- `<resource ref="Chapter3_Database::discounts"/>`: authenticated with the admin ACL.

Examples:

- `app/code/Chapter3/Database/etc/webapi.xml`
- `app/code/Chapter3/Database/etc/acl.xml`

Extension attributes

Magento is built on a foundation of flexibility. Yet, in this age of strong-typing everything, how is it possible to extend even interfaces?

Meet the mighty extension attribute. It is a brilliant solution to an age-old problem such as this.

Extension attributes are configured in a module's `etc/extension_attributes.xml` file. Extension attributes can be scalar values or objects. These attributes are compiled into an `Extension` class and interface.

Here is an example of extension attribute configuration:

```
<extension_attributes for="Magento\Directory\Api\Data\
  CountryInformationInterface">
    <attribute code="currency_information" type="Chapter3\
     Database\Api\Data\CurrencyInformationInterface" />
</extension_attributes>
```

Further Reading:

- Configure services as web API

By adding the above, Magento will add `getCurrencyInformation` and `setCurrencyInformation` to three generated files (found in `generated/code/Magento/Directory/Api/Data/`):

- `CountryInformationExtension.php`: the concrete class providing getters and setters for each declared extension attribute entry.
- `CountryInformationExtensionInterface.php`: the contract defining the getters and setters.
- `CountryInformationExtensionInterfaceFactory.php` the factory for generating `CountryInformationExtension`. Because the extension attribute classes are not generated automatically, you will utilize this class to create an instance if necessary.

For the above extension attribute declaration, let's make some observations:

- The generated files are in the same namespace as the `for` class or interface.
- The generated files replace `Interface` (if it exists) and add `Extension`, `ExtensionInterface` and `ExtensionInterfaceFactory` to the class or interface name.

If you are adding a new extension attribute to a class whose extension files have already been generated, you will need to delete the generated files for that class.

Our example will inject a dummy class into a list of directory countries.

Practical experience:

- Set a breakpoint in `\Chapter3\Database\Plugin\SetCurrencyInformation::afterGetCountriesInfo`.
- Navigate to https://lc.commerce.site/rest/V1/directory/countries and notice the `currency_information` coming through in the `extension_attributes`.
- Based on our example, create another extension attribute.

How to create an entity that supports extension attributes

Extension attributes depend on one method being specified in a class: `getExtensionAttributes`. This method is automatically specified in `\Magento\Framework\Api\AbstractExtensibleObject` and `\Magento\Framework\Model\AbstractExtensibleModel` (for EAV models). However, you are required to override this method in your implementation to set the correct return type in the PHP doc block.

Further Reading:

- An Introduction to Extension Attributes
- EAV and extension attributes

Here is an example from the order model:

```
// vendor/magento/module-sales/Model/Order.php
/**
 * @return \Magento\Sales\Api\Data\OrderExtensionInterface|null
 */
public function getExtensionAttributes()
{
    return $this->_getExtensionAttributes();
}
```

I suggest you review the `ExtensionAttributesFactory` below to observe how this generates the applicable files.

Practical experience:

- Set a breakpoint in `\Magento\Framework\Api\Code\Generator\ExtensionAttributesGenerator::_generateCode`.
- Delete the files in `generated/code/Magento/Directory/Api/Data`.
- Navigate to: https://lc.commerce.site/rest/V1/directory/countries

How to implement SearchCriteria processing in the repository::getList method

See the example in `app/code/Chapter3/Database/DiscountRepository.php`.

Further Reading:

- `vendor/magento/framework/Api/ExtensionAttributesFactory.php`

How to perform bulk save operations in Magento

Note:

That this is a Commerce-only feature until Magento 2.3, when it is available for Open Source, too.

Bulk save operations are an extension of message queues (discussed later). This type of operation is generally bolted onto a cron job or, worse, just done in the admin, or even worse, when an order is placed. Bulk operations defer operations, making a better Admin user experience.

This is a tremendous feature and our example should make the learning curve of this much easier.

To build a bulk save operation, two components are required:

- A publisher, to format the data such that it can be stored and transported to the queueing system.
- A consumer, to load the data and perform the necessary actions. This operation is manually started (think supervisor) to receive the messages coming from RabbitMQ.

In reality, there are more components that are required:

- Scheduler (`\Queueing\BulkSave\Operations\BulkDiscountScheduler::execute`). This publishes the operation data to the message queue.
- Merger (`\Queueing\BulkSave\Operations\BulkDiscountMerger`). This merges messages together before being published.
- Consumer (`\Queueing\BulkSave\Operations\BulkDiscountConsumer`). This receives messages from the queue system and takes action on them.

As you review the above code examples, you will see that a large portion of the code is gracefully handling error cases.

Example:

- See `\Queueing\BulkSave\Controller\Index\Index` (this is in a different namespace as the message queue system and does not use the Magento Object Manager, thus no virtual types, and cannot have numbers in the class path).
- Install `RabbitMQ` on your development machine.
    - Note that you can utilize MySQL, but I wrote this in RabbitMQ so you get the full picture.
- Configure your `app/etc/env.php` file to connect to RabbitMQ (example below)
- Run `bin/magento setup:upgrade`.
- Navigate to: https://lc.commerce.site/index.php/chapter3bulksave
- Then, go to the console and run `bin/magento queue:consumers:start save_discount_price_consumer`. This will begin the consumer process.

- Check the values in `swiftotter_discounts`.
- Refresh https://lc.commerce.site/index.php/chapter3bulksave and recheck `swiftotter_discounts`. The values should update almost immediately.

RabbitMQ configuration in `app/etc/env.php`:

```
'queue' => array (
    'amqp' =>
        array (
            'host' => 'localhost',
            'port' => '5672',
            'user' => 'guest',
            'password' => 'guest',
            'virtualhost' => '/'
        ),
)
```

I also had to enable permissions for the guest user:

```
sudo rabbitmqctl set_permissions -p / guest ".*" ".*" ".*"
```

How to extend the Magento data object (Data API class) with an attribute that has values in a remote system

Extension attributes. Ideally, the values for the extension attributes will be cached (in the Magento cache or in an intermediary API). Loading values from a

Further Reading:

- Bulk Operations
- How to Use RabbitMQ Queue in Extensions For Magento 2.0 EE
- Configure message queues
- Magento 2 tutorial: message queues
- How to integrate RabbitMQ with Magento 2.2.* EE

remote system could be time intensive and make an API request come to a grinding halt.

Further Reading:

- How to Use Data-related Classes, Repositories and Data API in Magento 2

How to extend existing WebAPI calls with a new parameter

Adding a parameter is fairly simple but can have some consequences. If this is a widely-used REST URL (such as saving a product), ensure that you have the proper safeguards to prevent unintended side effects.

If you wish to add a parameter to the URL, from:

```
https://lc.commerce.site/rest/V1/testOverride/joseph/
```

to

```
https://lc.commerce.site/rest/V1/testOverride/joseph/maxwell/
```

This is essentially a new API definition and should be completed accordingly. Magento sees these as two separate URLs. If you can simply add a parameter to the URL, this is backward compatible and should be preferred.

Practical experience:

- In your browser, visit: https://lc.commerce.site/rest/V1/testOverride/Joseph%20Maxwell
    - What is the response? What class is handling this response?
- Now, visit: https://lc.commerce.site/rest/V1/testOverride/Joseph/Maxwell
    - What is the response for this?

You can also override the existing API definition by simply specifying the same URL and method but with a different interface. You must also ensure that your module's sequence is after the module you are overriding.

Practical experience:

In your web browser's developer tools, first run:

```
fetch('https://lc.commerce.site/rest/V1/postOverride', {
    method: 'post',
    headers: {
            "Content-Type": "application/json"
    },
    body: JSON.stringify({name: 'Joseph Maxwell'}),
}).then(response => response.json()).then(response => console
log(response));
```

What is the response? Now, run:

```
fetch('https://lc.commerce.site/rest/V1/postOverride', {
    method: 'post',
    headers: {
            "Content-Type": "application/json"
    },
    body: JSON.stringify({firstName: 'Joseph', lastName:
'Maxwell'}),
}).then(response => response.json()).then(response => console.
log(response));
```

How to create a dynamic WebAPI ACL

Create a plugin for: `\Magento\Framework\Webapi\Authorization::isAllowed`

Example:

```
<!-- vendor/magento/module-webapi/etc/webapi_rest/di.xml -->
<type name="Magento\Framework\Authorization">
    <plugin name="guestAuthorization" type="Magento\Webapi\Model\
     Plugin\GuestAuthorization" />
</type>
```

PHP: `\Magento\Webapi\Model\Plugin\GuestAuthorization::aroundIsAllowed`

The difference between extension attributes and custom attributes

Custom attributes are loaded from the model's EAV entity tables (ex: `catalog_product_entity_varchar`). Extension attributes could be this, but rather, they are other information structures loaded from elsewhere (database, cache, filesystem, even remote APIs).

3.2 Demonstrate understanding of the staging workflow

Staging modification to the Magento database operations (row_id, entity managers)

Commerce has a feature called content staging. This allows content managers to test and then schedule updates in the future.

One of the fundamental changes that arise as a result is how Magento uses `row_id` as an entity table's primary key (instead of `entity_id`).

`entity_id` is present in Open Source and represents an item (ex. product). `row_id` is the version identifier that defines the version of the current row and is the unique, primary key. `created_in` and `updated_in` are Unix timestamps (`bigint`) that correspond with the `id` column in `staging_update`.

Once the `*-staging` modules are installed (Commerce-only), a significant change takes place in the attribute value tables. Instead of these tables referencing the `entity_id` (or having an `entity_id` column) they reference the `row_id` column.

`catalog_product_entity` table:

row_id	entity_id	created_in	updated_in	attribute_set_id	type_id	sku	has_options	required_options	created_at	updated_at
41	41	1	1547914549	37	simple	24-WG09	0	0	2019-01-19 16:14:42	2019-01-19 16:14:46
42	42	1	1547914550	37	simple	24-WG01	0	0	2019-01-19 16:14:42	2019-01-19 16:14:47
43	43	1	2147483647	37	simple	24-WG03	0	0	2019-01-19 16:14:43	2019-01-19 16:14:43
44	44	1	2147483647	37	simple	24-WG02	0	0	2019-01-19 16:14:43	2019-01-19 16:14:43
45	2	1547914545	2147483647	41	simple	24-MB04	0	0	2019-01-19 16:14:29	2019-01-25 08:06:30
46	10	1547914546	2147483647	41	simple	24-WB05	0	0	2019-01-19 16:14:31	2019-01-25 08:06:32
47	11	1547914547	2147483647	41	simple	24-WB06	0	0	2019-01-19 16:14:31	2019-01-25 08:06:36
48	16	1547914548	2147483647	37	simple	24-UG07	0	0	2019-01-19 16:14:33	2019-01-25 08:06:44
49	41	1547914549	2147483647	37	simple	24-WG09	0	0	2019-01-19 16:14:42	2019-01-25 08:06:50
50	42	1547914550	2147483647	37	simple	24-WG01	0	0	2019-01-19 16:14:42	2019-01-25 08:06:56

`catalog_product_entity_varchar` table:

value_id	attribute_id	store_id	row_id	value
1	73	0	1	Joust Duffle Bag
2	106	0	1	container2
3	124	0	1	joust-duffle-bag
4	216	0	1	20,31,30,28
5	217	0	1	34,35,38
6	218	0	1	46,47
7	219	0	1	70,71,72,73,74,75
8	220	0	1	83,85,88
9	87	0	1	/m/b/mb01-blue-0.jpg

`staging_update` table:

id	start_time	name	description	rollback_id	is_campaign	is_rollback	moved_to
1547914545	2019-01-19 16:15:45	24-MB04	NULL	NULL	0	NULL	NULL
1547914546	2019-01-19 16:15:45	24-WB05	NULL	NULL	0	NULL	NULL
1547914547	2019-01-19 16:15:45	24-WB06	NULL	NULL	0	NULL	NULL
1547914548	2019-01-19 16:15:46	24-UG07	NULL	NULL	0	NULL	NULL
1547914549	2019-01-19 16:15:46	24-WG09	NULL	NULL	0	NULL	NULL
1547914550	2019-01-19 16:15:46	24-WG01	NULL	NULL	0	NULL	NULL
1547914645	2019-01-19 16:17:25	20% off all Women's and Men's Pants	NULL	NULL	0	NULL	NULL
1548717840	2019-01-28 23:24:00	Test update	NULL	NULL	0	NULL	NULL
1548986400	2019-02-01 02:00:00	Test 2	NULL	NULL	0	NULL	NULL

Staging updates are based on timestamps (`bigint type`). In the example below, "1547914545" converts to Sat, 19 Jan 2019 16:15:45 +0000. By default, `created_in = 1` and `updated_in = 2147483647` (`\Magento\Staging\Model\VersionManager::MAX_VERSION`). These columns will never have the same value.

Important points to remember:

- `staging_update` is the master table for new scheduled updates.
- `row_id` becomes the primary key.
- All attribute value tables reference the entity table's `row_id` and not the `entity_id`.
- There can be multiple of the same `entity_id` in an EAV + staging enabled entity.
- `created_in` is always `!==` to `updated_in`.
- `created_in` and `updated_in` are timestamps.

Practical experience:

Locate the table that is keyed to the `row_id` for the products (hint: `sequence_…`).

Further Reading:

- `vendor/magento/module-catalog-staging`
- Magento 2: Schema Changes for EE Catalog Staging
- Magento 2.1 ee Content Staging

Entity Manager

The entity manager is a framework that Magento built to provide database interaction by configuration. They achieved the goal—but have abandoned it due to too much configuration.

Information:

Study `vendor/magento/module-staging/Model/VersionInfoProvider.php`.

If you wish to learn more about the entity manager, see the appendix at the end of the book.

How does data versioning work?

Note: if you want to see what edition and version of Magento a site is running, go to `/magento_version`, like: https://swiftotter.com/magento_version/

This is the deepest dive that I have taken into this mechanism in Magento. However, I wasn't seeing any code related to this in Magento. Doing some research, I found that Magento released CMS version control in Magento 2.0 but then removed in Magento 2.0.1 (`vendor/magento/module-versions-cms/Setup/UpgradeSchema.php`).

As such, we will explore Scheduled Changes.

Important points:

- The `created_in` and `updated_in` columns reference the `id` column in `staging_update`. All are Unix timestamps.
- The core entity model for Scheduled Changes is `\Magento\Staging\Model\Update`.
- The `\Magento\Staging\Model\VersionManager` stores information about the active version (whether with saving, or preview or active).

- `staging_update` is the primary location for versioning information.
- `\Magento\Staging\Model\EntityStaging` is the central point for scheduling or unscheduling changes.
- `vendor/magento/module-versions-cms` is actually the module for a CMS hierarchy.

How does it work when creating a new scheduled update?

- `\Magento\Staging\Model\Entity\Update\Save::execute` is the entry point for creating or updating a scheduled update.
- The specific action class is loaded from `Magento\Staging\Model\Entity\Update\Action\Pool`. An example of this configuration is in `vendor/magento/module-cms-staging/etc/adminhtml/di.xml`.
- The save action model (inheriting `Magento\Staging\Model\Entity\Update\Action\Save\SaveAction`) is created.
- The executor action model (`\Magento\Staging\Model\Entity\Update\Action\TransactionExecutor`) is loaded.
- A transaction is created.
- The action is executed:
    - Update is either created or loaded.
    - The version manager has the new scheduled update version set.
    - The entity is scheduled (i.e. saved).
    - See: `\Magento\CmsStaging\Model\PageStaging::schedule`

Practical experience:

- Set a breakpoint in `\Magento\Staging\Model\Entity\Update\Save::execute`.
- Open a CMS page, and create a Scheduled Update for it.
- Save the scheduled update and step through the process.
- Then, change the scheduled update and save it again.

Different possibilities of data versioning (row/table/database level) and how this is implemented in Magento

At the row level, data versioning uses the `row_id` (multiple `row_id`'s per `entity_id`). These rows are associated with the table `sequence_[entity type]` to ensure a unique value. In addition, the `created_in` and `updated_in` columns for a table that is managed in content staging, references the `staging_update` table.

To load a product from the database, the Magento Open Source query would look something like:

```
SELECT * FROM catalog_product_entity WHERE entity_id = 1;
```

However, this changes significantly with Magento Commerce. Consider this dataset with the above query. This is the result:

row_id	entity_id	created_in	updated_in	attribute_set_id	type_id	sku
1	1	1	1549763040	41	simple	24-MB01
2055	1	1	549763040	2147483647	41	simple	24-MB01

As such, you need to load a specific row for an entity_id. Here is an example SQL query:

```
SELECT `catalog_product_entity`.* FROM `catalog_product_entity`
WHERE (entity_id =1) AND (catalog_product_entity.created_in
<= '1549763040') AND (catalog_product_entity.updated_in >
'1549763040')
```

How do you load a product that is the current version?

You could write something yourself, or you can take the better road and let Magento do it for you—automatically! Simply load a product from a staging-powered repository or resource model, and Magento will add the `where` clause for determining the correct staging version. Actually, you can even create a custom `Select`, with the `from` being a staging-enabled entity table, and Magento will add the conditions for you.

Here's how the SQL statement renderers work:

- One of the renderers is supplied by the staging module (`\Magento\Staging\Model\Select\FromRenderer`).
    - This class has a `stagedTables` property that contains a list of all staging-enabled tables. To add a table to this list, add your table name to the `Magento\Staging\Model\StagingList`'s `entities` argument.
    - Then, this class will load metadata from the metadata pool for this table.
- When rendering the table name, the version (`vendor/magento/module-staging/Model/VersionManager.php::getVersion`) is loaded.

- This takes into account the `__version` request parameter, to specify a version.
- `\Magento\Staging\Model\ResourceModel\Update::getMaxIdByTime` is called. This method converts the requested timestamp into a `\DateTime` object. (`SELECT * FROM staging_update WHERE start_time <= [TIMESTAMP] ORDER BY `id` DESC LIMIT 1`). The current version or the next most recent (but not in the future) will be selected.
- If no `__version` request parameter is specified, then Magento looks up `staging` in the `flag` table. The `flag_data` column contains the current staging version.

Note:

The `flag` table stores information that isn't configuration or cacheable but needs to be updated or used on a regular basis.

flag_id	flag_code	state	flag_data	last_update
1	staging	0	{"current_version":"1534546801"}	2018-08-17 23:00:02
2	report_order_aggregated	0	NULL	2019-02-09 06:00:02
3	report_tax_aggregated	0	NULL	2019-02-09 06:00:03
4	report_shipping_aggregated	0	NULL	2019-02-09 06:00:03
5	report_invoiced_aggregated	0	NULL	2019-02-09 06:00:02
6	report_refunded_aggregated	0	NULL	2019-02-09 06:00:02
7	report_coupons_aggregated	0	NULL	2019-02-09 06:00:03
8	report_bestsellers_aggregated	0	NULL	2019-02-09 06:00:03
9	report_product_viewed_aggregated	0	NULL	2017-07-25 16:48:06
10	log_rotation	0	1549699262	2019-02-09 08:01:02
12	config_hash	0	{"system":"2be88ca4242c76e8253ac62474851065032d6833"}	2018-12-03 22:11:07

Practical experience:

- Schedule a product modification.
- Preview it.
- Set a breakpoint in `\Magento\Catalog\Helper\Product::initProduct`.

- Follow the path through the stack trace to find where the specific version is set. Observe how this is set and where the versions are pulled from.

Further Reading:

- `vendor/magento/module-staging/Model/Select/FromRenderer.php`
- `vendor/magento/module-staging/Model/Update/VersionHistory.php`

How do you load a product or entity that is in a specified version?

First, read the above as it will provide additional context for this example.

```
/** @var $versionManager \Magento\Staging\Model\VersionManager **/
$versionManager->setCurrentVersionId($versionId);
$this->productRepository->getById($productId);
```

If you wish to disable the additions for staging, and you have access to the `Select` object, add a part to the `Select`:

```
$select->setPart('disable_staging_preview', true);
```

Practical example:

- Log into your Magento admin and create some staging versions for product ID #1 (24-MB01)
- Navigate to Catalog > Example 1 (select products)
- Code found in: `app/code/Chapter3/Staging`

How do you create a join for a staging entity?

You join on the `entity_id` (`$productResource->getEntityId()`) column. Here is an example of SQL:

```
SELECT `main_table`.*, `product`.`sku`, `product`.`created_in`,
`product`.`updated_in`
FROM `sales_order_item` AS `main_table`
INNER JOIN `catalog_product_entity` AS `product` ON main_table.
product_id = product.entity_id AND (product.created_in <=
'1547914645' AND product.updated_in > '1547914645')
```

Notice that joining to a staging entity involves using the `$resource->getEntityId()`. Joining from a staging entity to staging entity's attribute values requires using the `$resource->getLinkField()`.

Practical example:

- Log into your Magento admin and create some staging versions for product ID #1429 (WS03-M-Red)
- Navigate to Catalog > Example 2 (join products)
- Code found in: `app/code/Chapter3/Staging`

Further Reading:

- `vendor/magento/module-catalog-staging/Model/Product/Locator/StagingLocator.php`

What happens when a staging update "goes live"?

Every minute the Magento cron executes the `\Magento\Staging\Model\StagingApplier::execute`. This class:

- Determines the current timestamp.
- Iterates through each staging-enabled entity type.
- Finds all rows that have the `created_in` between the last-enabled version and the current version (timestamp).
- Runs the applicable applier (if available) for each row.
- Then, loads up the repository and saves that item (`save`).

The performance implications are that each item is being saved. If there are many items affected by the scheduled update, this could have a major impact on the website, and possibly make the frontend unresponsive, unless varnish has been properly configured.

The role of the entity manager

As described above, the entity manager is (was) a replacement for the abstract resource model. It handles all CRUD operations for the more complex of Magento's entities. It provides tremendous flexibility and capability over how these operations are handled. However, it was a bit too complex for most usages.

High level staging implementation overview

In the PHP code, Staging-enabled entities are not modified from the Open Source not-enabled entities. However, staging brings additional configuration and classes to support the extra functionality.

Staging is enabled through the entity manager.

Practical investigation:

- Compare `vendor/magento/module-cms` and `vendor/magento/module-cms-staging`. The CMS module is the most simple staging-enabled module.
    - `vendor/magento/module-cms-staging/etc/di.xml`

Comparison of metadata pool.

Original (`vendor/magento/module-cms/etc/di.xml`):

```
<!-- … -->
<type name="Magento\Framework\EntityManager\MetadataPool">
    <arguments>
        <argument name="metadata" xsi:type="array">
            <item name="Magento\Cms\Api\Data\PageInterface" xsi:type="array">
                <item name="entityTableName" xsi:type="string">cms_page</item>
                <item name="identifierField" xsi:type="string">page_id</item>
            </item>
            <!-- … -->
        </argument>
    </arguments>
</type>
<!-- … -->
```

Modified (`vendor/magento/module-cms-staging/etc/di.xml`):

```
<type name="Magento\Framework\EntityManager\MetadataPool">
    <arguments>
        <argument name="metadata" xsi:type="array">
            <item name="Magento\Cms\Api\Data\PageInterface" xsi:type="array">
                <item name="entityTableName" xsi:type="string">cms_page</item>
                <item name="sequenceTable" xsi:type="string">sequence_cms_page</item>
                <item name="identifierField" xsi:type="string">page_id</item>
            </item>
            <!-- … -->
        </argument>
    </arguments>
</type>
```

Also, note the update in the OperationPool.

Original (`app/etc/di.xml`):

```
<type name="Magento\Framework\EntityManager\OperationPool">
    <arguments>
        <argument name="operations" xsi:type="array">
            <item name="default" xsi:type="array">
                <item name="checkIfExists" xsi:type="string">
                    Magento\Framework\EntityManager\Operation\
                    CheckIfExists</item>
                <item name="read" xsi:type="string">Magento\
                    Framework\EntityManager\Operation\Read</item>
                <item name="create" xsi:type="string">Magento\
                    Framework\EntityManager\Operation\Create</item>
                <item name="update" xsi:type="string">Magento\
                    Framework\EntityManager\Operation\Update</item>
                <item name="delete" xsi:type="string">Magento\
                    Framework\EntityManager\Operation\Delete</item>
            </item>
        </argument>
    </arguments>
</type>
```

Modified:

```
<type name="Magento\Framework\EntityManager\OperationPool">
    <arguments>
        <argument name="operations" xsi:type="array">
            <item name="Magento\Cms\Api\Data\PageInterface"
               xsi:type="array">
                <item name="create" xsi:type="string">Magento\
                    Staging\Model\Operation\Create</item>
                <item name="update" xsi:type="string">Magento\
                    Staging\Model\Operation\Update</item>
                <item name="delete" xsi:type="string">Magento\
                    Staging\Model\Operation\Delete</item>
            </item>
            <!-- ... -->
        </argument>
    </arguments>
</type>
```

3.3 DEMONSTRATE AN ABILITY TO USE DIFFERENT TYPES OF SETUP SCRIPTS IN MAGENTO

Schema and data setup scripts, uninstall scripts, recurring scripts, uninstall schema vs. uninstall data

Setup scripts are Magento 2.2's way of configuring the database. While they are primitive, in functionality, they give fairly unfettered access to the database and can pose a danger if not treated with respect.

Setup scripts are quite similar but categorized for specific purposes (data, schema, etc.).

Practical examples:

- Run `bin/magento module:enable Chapter3_SetupScripts`
- `app/code/Chapter3/SetupScripts`
- In PHPStorm, create a new PHP Script configuration in Run > Run/Debug Configurations.
- Set the file to be the path to your `bin/magento` file.
- Set the arguments to be: `setup:upgrade --keep-generated`

Install / upgrade scripts

Install scripts run when the module is not found in `setup_module` table, and upgrade scripts run when the module is out of date.

You might notice the availability of `ModuleDataSetupInterface::startSetup()` and `ModuleDataSetupInterface::endSetup()`. These are not necessary to include in your setup scripts. If you step into the code (`\Chapter3\SetupScripts\Setup\UpgradeData::upgrade`), you will see that the execution for this command happens in `\Magento\Framework\DB\Adapter\Pdo\Mysql::startSetup`. This method only runs the following SQL queries:

```
SET SQL_MODE=''"
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0
SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='NO_AUTO_VALUE_ON_ZERO'
```

As you can see, calling `startSetup` disables foreign key checks and allows `0` not to trigger, replacing itself with the next auto increment ID.

Further Reading:

- Setup scripts in Magento 2

Recurring scripts

Recurring scripts run anytime `setup:upgrade` is run. These scripts run after the schema or data scripts are executed.

Recurring schema: `MyCompany\MyModule\Setup\Recurring` implements `\Magento\Framework\Setup\InstallSchemaInterface`

Recurring data: `MyCompany\MyModule\Setup\RecurringData` implements `\Magento\Framework\Setup\InstallDataInterface`

Practical examples:

- `\Chapter3\SetupScripts\Setup\Recurring`
- `\Chapter3\SetupScripts\Setup\RecurringData`

Uninstall scripts

There is one uninstall script (combined for data and schema). The only example in the Magento code is: `\Temando\Shipping\Setup\Uninstall`. If you wish to include an `Uninstall` script, place it in your module's `Setup/Uninstall.php` file.

These scripts are executed whenever `bin/magento module:uninstall [MODULE NAME]` is run.

What happens when an uninstall script is executed: data version change, deleted tables, etc.

What happens is whatever the developer writes in the code. There is no magic to running these methods.

Recurring scripts and their order in the setup:upgrade process

- Schema install / upgrade
- Schema recurring
- Data install / upgrade
- Data recurring

Practical experience:

- Create a breakpoint in:
  `\Magento\Setup\Model\Installer::handleDBSchemaData`

Accessing areas and system configuration values in setup scripts

Areas: `\Magento\Framework\App\State` (Make sure to catch exceptions as the area code is not set by default in setup scripts.)

Store configuration values:
`\Magento\Framework\App\Config\ScopeConfigInterface`

Further Reading:

- Setup scripts

Practical experience:

- Set a breakpoint in:
  `\Chapter3\SetupScripts\Setup\Recurring::install`
- When running `bin/magento setup:upgrade --keep-generated`
- Observe how `getValue` behaves.

4 USING THE ENTITY-ATTRIBUTE-VALUE (EAV) MODEL

6 questions

Professional Developer Plus

SWIFT OTTER SOLUTIONS

4.1 DESCRIBE THE EAV DATA ACCESS PROCESS IN MAGENTO

Getting an attribute instance

Inject an attribute repository instance (`\Magento\Eav\Model\AttributeRepository`) into your class and call `get`.

Practical example:

- `\Chapter3\Staging\ViewModel\JoinProducts::getOrderItems`

Impact of attribute sets. What is the impact of 10,000 attribute sets?

This will slow down response times for loading product pages (edit and listing) in the admin panel. Each attribute set is written into the edit product response (uiComponent). There is no limit for how many attribute sets are available for selection.

Additionally, the user interface will also slow down as KnockoutJS is not optimized for handling large amounts of data.

Further Reading:

- `\Magento\Catalog\Ui\DataProvider\Product\Form\Modifier\AttributeSet`

Large number of attributes and attribute sets. 1,000 attributes in a set?

This can have a serious impact on the rendering of products on the frontend. Remember that Magento loads the attribute metadata and then the value for that attribute. Entities will have to store this information in memory. However, because attribute metadata is stored in the Magento cache, the worst effects would be seen on the first page load.

This can also cause problems with indexing in the flat entity tables. Flat entity tables are not per attribute set and thus are an aggregation of all attributes. You could run into MySQL maximum column limits. As of MySQL 5.6.9, the maximum number of columns is 1017. It should be quite simple to pare back the number of attributes that are marked as "Used in Product Listing."

Finally, so many attributes could bring the admin product edit page to an unusable crawl. Again, KnockoutJS is not designed for high-performance.

Further Reading:

- Error Code 1117 Too many columns; MySQL column-limit on table
- 14.6.1.6 Limits on InnoDB Tables

How to get information about an attribute

The safest is to utilize methods in `\Magento\Eav\Api\Data\AttributeInterface` to obtain information. However, the type returned (`\Magento\Eav\Model\Entity\Attribute`) from the attribute repository

contains many more methods. One helpful method is the `getBackendTable()` which provides access to the table that stores this attribute's values.

Note:

Note: `\Magento\Catalog\Api\Data\ProductAttributeInterface` has many default product attribute codes already defined in an `@api`. Anytime you can use methods or constants specified in the API, your code has less chances of breakage.

How to perform attribute operations programmatically: assign it to a set/group, update properties, etc.

Creating attributes is quite simple: inject the `\Magento\Eav\Setup\EavSetupFactory` class and call `addAttribute`. We will not go into all of the options that are available.

Here are a few key points:

- If you specify a `group` key in the attribute settings, Magento will either create a group or add this attribute to an existing group. Remember, an attribute set has attribute groups which have attributes. Additionally, if the `group` key is specified, the attribute will be added into all attribute sets. Do not specify the `group` if you want to control which attribute sets the attribute is added to.
    - Reference: `\Magento\Eav\Setup\EavSetup::addAttribute`
- To update an attribute, call the `updateAttribute` method on the `EavSetupFactory`.
- EAV entities can have an `addition_attribute_table` specified (see

`eav_entity_type`). These tables (`catalog_eav_attribute` and `customer_eav_attribute`) contain extra information about an attribute. To specify values for these tables, simply include them in the attribute definition when creating or updating.

Customer attributes require adding the attribute to forms, after the attribute is created. See the practical example for how to do this.

Practical example:

- `\Chapter4\EAV\Setup\InstallData::install` Feel free to execute this as a PHP script in PHPStorm. Set a breakpoint on the `addAttribute` line.
- `\Chapter4\EAV\Setup\UpgradeData::updateProductAttribute`
- `\Chapter4\EAV\Setup\UpgradeData::createCustomerAttribute`

Further Reading:

- How to Add a New Product Attribute
- Magento 2 Add Customer Attribute Programmatically

4.2 DESCRIBE THE DATABASE TABLES FOR EAV ENTITIES AND HOW TO CREATE THEM

The EAV database structure, performance considerations, entity-level attribute properties (catalog_eav_attribute)

There are quite a number of tables for the EAV configuration:

- `eav_attribute`: where the attributes are configured and stored.
- `eav_attribute_label`: provides translations from the admin panel for attribute labels.
- `eav_attribute_option`: definition for admin-created attribute options. Specifies sort order and attribute ID.
- `eav_attribute_option_swatch`: specifies swatch information for an option value.
- `eav_attribute_option_value`: sets the name per store for an option.
- `eav_attribute_group`: groups attributes together (the collapsible items on the product edit page).
- `eav_attribute_set`: defines sets or groupings or lists of attributes.
- `eav_entity_attribute`: associates an attribute with a group and a set.
- `eav_entity_type`: defines the type of entity (product, category, customer, etc.)

Then, for each entity type, there are several tables that store values (using `catalog_product` as the example):

- `catalog_product_entity`: defines information for the product. Note that this table contains the fastest lookups. It is quite simple to add columns to this table (so long as the attribute's scope is Global, more later).
- `catalog_product_entity_[datetime, decimal, int, text, varchar]`: stores values for the entity.

Note that the entity-level attribute properties are discussed above in 4.1.

Where are catalog-specific attribute properties stored and what are they used for?

They are stored in `catalog_eav_attribute`. They provide supplemental information about each attribute. For example, the `apply_to` column defines which product types that this attribute works for. Note that if a row's value for this column is blank, then the attribute applies to everything. However, you can limit back an attribute's application by setting a comma-separated list in this column.

How does Magento store the attribute to attribute group association?

This is stored in `eav_entity_attribute`. Here are the columns in the table:

- `entity_attribute_id`: primary key
- `entity_type_id`: the type of attribute. Technically, this is a repeat from what is found in the definition in `eav_attribute`.
- `attribute_set_id`: the attribute set
- `attribute_group_id`: the attribute group (inside the attribute set). Note that troubleshooting a missing attribute can lead you to an attribute group that is assigned to the wrong attribute set.
- `attribute_id`: the attribute
- `sort_order`: in what order this attribute should appear.

What backend types are available? How do you add a new backend type?

By default:

- `datetime`
- `decimal`
- `int`
- `text`
- `varchar`

To create a new backend type, create an attribute with the custom backend type specified. You will need to either specify the value for the attribute's `backend_table` or you need to create a new table for that entity with the type. For example, if you wanted to create a new JSON backend type for the `catalog_product` entity type, you would create a new table: `catalog_product_entity_json` to host the data.

Specifics around static attributes

Static attributes are an easy way to store data where it is quickly retrievable (i.e. minimal processing overhead). ***The disadvantage with static attributes is that they can only be utilized for the global scope.***

You must manually create the column for the static attribute.

4.3 DEMONSTRATE AN ABILITY TO OPERATE WITH ATTRIBUTE OPTIONS

Different ways to store attribute options. Using eav_attribute_option_* tables

Attribute options are loaded via the attribute's `source` class. The `source` class must extend `\Magento\Eav\Model\Entity\Attribute\Source\AbstractSource`. Note that if you have a custom `source` class and you are utilizing flat tables and your attribute is not added to the flat tables, you need to specify some additional information. Look at the `getFlatColumns()` method in the aforementioned abstract class.

If you wish to initially populate options for an attribute (and let the admin control the values henceforth), make sure to set the `input` (create attribute) or `frontend_input` (update attribute) to be `multiselect` or `select`. Then, when no custom `source_model` is specified `\Magento\Eav\Model\Entity\Attribute\Source\Table` is automatically assumed.

You can call `addAttributeOption` on the `EavSetup` class for each option you want to add.

Further Reading:

- `\Magento\Customer\Setup\UpgradeData::upgradeVersionTwoZeroTwo`

The `eav_attribute_option_ table[sic]`: tables that contain shared options between different entities, pros and cons of using the table

I am unable to see how the options as specified in `eav_attribute_option` are able to be shared with other attributes. Each row is associated with one specific `attribute_id`.

If you need to create shared options between entities, you must create a custom `source_model` for your attribute. This will load rows from the pertinent table and make those available in the `getAllOptions` method.

Why would you, as a Magento developer, add items to this table? You would do so if you need to give the store's administrator an easy place to manage these values or to allow easy customization per store. However, it is very inflexible for future updates to this information. As such, we do not use the `eav_attribute_option` tables.

4.4 DEMONSTRATE AN ABILITY TO USE NON-CATALOG EAV ENTITIES

Adding an attribute to Customer, Customer Address and Sales entities. Making an attribute visible in the Admin or the storefront. Pitfalls in attributes operations in non-catalog EAV attributes

See the examples for how to create these attribute types.

To make a customer or customer address attribute save, you must add the attribute to the correct form. Also, you must specify the `group` = `General` for non-catalog entities.

To make the inputs appear on the frontend, you must update the customer form or customer address form templates to add the appropriate inputs.

To simplify creating an order attribute, Magento has provided a setup helper class (`Magento\Sales\Setup\SalesSetup`) to streamline the process of adding order attributes. Calling the standard `addAttribute` method on

the `SalesSetup` class adds a column to the `sales_order` table and the `sales_order_grid` table (actually, for any table referenced in that class' `$_flatEntityTables` property).

Practical examples:

- `\Chapter4\EAV\Setup\UpgradeData::createCustomerAttribute`
- `\Chapter4\EAV\Setup\UpgradeData::createCustomerAddressAttribute`
- `\Chapter4\EAV\Setup\UpgradeData::createOrderAttribute`

Further Reading:

`\Magento\Eav\Model\ResourceModel\Attribute::_afterSave` is where the attributes are added to the forms. In my tests, saving an attribute via the AttributeRepository did not add the attribute to the forms. Rather, I had to call `save` directly on the attribute.

Adding an attribute to customers, saving and loading the attribute, problems related to the save process. What is the role of attribute sets and groups for customer attributes?

See examples above. If you do not add the attribute to the `customer_form_attributes` table, the value will not save on the frontend.

There is one attribute set for customers.

Adding an attribute to customer addresses, the role of the `is_system` property and why it only works for the Customer Address entity

The `is_system` switch tells Magento whether or not an attribute's value should go into the custom attributes list (`\Magento\Customer\Model\Metadata\AddressMetadata::getCustomAttributesMetadata`).

This switch also tells the Customer Attribute management module whether or not the attribute can be deleted (see `\Magento\CustomerCustomAttributes\Block\Adminhtml\Customer\Formtype\Edit::_construct`).

How to make a customer or customer address attribute visible in the My Account, Checkout, and Admin pages

My Account

Override `Magento_Customer::form/edit.phtml` and add the HTML to make your new input appear.

Checkout

Customer addresses are not directly visible in the checkout. You must configure the frontend uiComponent.

Further Reading:

- Add a new field in address form

Admin pages

As discussed above, you must ensure that the attribute is present in the `customer_eav_attributes` table (and, of course, clear the cache).

What is the purpose of the SalesSetup class and why do you use the addAttribute method for sales entities?

See above under "Adding an attribute to Customer, Customer Address and Sales entities."

5 DEVELOPING WITH ADMINHTML

3 questions

Professional Developer Plus

SWIFTOTTER SOLUTIONS

5.1 DEMONSTRATE ABILITY TO USE ACL

Complex cases of ACL setup

Magento ACL lookups are comprised of several components:

- The admin user's role.
- The resource needing to be accessed.
- (there is a `$privileges` parameter, but this is never used in Magento)

Further Reading:

- `\Magento\Authorization\Model\Acl\Loader\Rule::populateAcl`

Configuration for ACL is stored in each module's `etc/acl.xml` file. Here is a sample:

```
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:Acl/etc/acl.xsd">
    <acl>
        <resources>
            <resource id="Magento_Backend::admin" title="Magento
               Admin" translate="title" sortOrder="20">
                <resource id="Chapter5_ACL::all_actions"
                 title="All Actions" translate="title" sortOrder="0" />
            </resource>
        </resources>
    </acl>
</config>
```

Note that the admin ACL entries are always found in the `Magento_Backend::admin` resource (otherwise, you will not be able to administer these entries). Below this node, you can create as complex of a structure as necessary.

WebAPI ACL

The API is tightly integrated with the ACL. Every API route declaration can be associated with a ACL resource (notice how the `ref` matches):

```
<route url="/V1/acl" method="GET">
    <service class="Chapter5\ACL\Api\AclInterface" method="getById"/>
    <resources>
        <resource ref="Chapter5_ACL::all_actions"/>
    </resources>
</route>
```

Note that the entry point for the WebAPI ACL is: `\Magento\Framework\Webapi\Authorization::isAllowed`. This allows a separation between the Magento Admin ACL and the API ACL. The API authorization mechanism also allows for `anonymous` and `self` (customer session). It does not work for the admin session as the API always initializes the frontend session.

Note that you must use a token to access admin resources.

`\Magento\Authorization\Model\Acl\AclRetriever::_canRoleBeCreatedForUserType`

Further Reading:

- Authentication
- Order Admin Token

Practical experience:

- Enable Chapter5_ACL.
- First, go to: https://lc.commerce.site/rest/V1/acl. Step through to the code where this is rejected.
    - `\Magento\Webapi\Model\WebapiRoleLocator::getAclRoleId`

- Now, modify `app/code/Chapter5/api-test.php` to match your username and password.
- Then run the file (PHPStorm's run PHP scripts works well), or:
  - `php -f app/code/Chapter5/api-test.php`
  - You will need to change the username and password to ones that work in your test environment.
  - Note that if you do not use the testing URL (https://lc.commerce.site), you will need to update this file.

ACL process customization

Customization is done relatively easy through plugins. See below for more examples.

Admin and API common point: `\Magento\Framework\Authorization::isAllowed`

API only: `\Magento\Framework\Webapi\Authorization::isAllowed`

How to debug an ACL record

First, ensure that your record is in `Magento_Backend::admin` (for admin and API entries) and `Magento_Backend::admin/Magento_Backend::stores/Magento_Backend::stores_settings` for store configuration entries (so that it matches the configuration tree in role configuration).

- Set a breakpoint in `\Magento\Framework\Authorization::isAllowed`. This is common to both the admin and API paths.
- Ensure that a user is set and that a role is specified.
- Step through `\Zend_Acl::isAllowed` to determine what resources are allowed for this user.

Further Reading:

- Relation between acl.xml and webapi.xml

How does a row-based ACL or IP-based ACL work in Magento?

Because you can create a plugin for the `isAllowed` method, you have complete flexibility over what is allowed.

For example, to authenticate a user's IP address, inject `Magento\Framework\HTTP\PhpEnvironment\RemoteAddress` into your plugin's `__construct()` method. Then, create `afterIsAllowed` and check that the IP address matches.

Row-based ACL is authorizing access by row. In other words, the concept is that specific rows in the database are authorized or denied. For example, this would be specific products are allowed and others are denied.

Magento provides a glimpse into how this works in the AdminGws module. Instead of using the `isAllowed` method, they interact with each touchpoint throughout the Magento admin (and there are a lot). This allows more fine-grained control to prevent users from even seeing objects that they are not allowed to see.

Further Reading:

Study the approach that Magento takes in the `Magento\AdminGws` module.

The connection between admin ACL and WebAPI ACL

The connection is found in the `webapi.xml` file, as discussed above. ACL resources are setup in `acl.xml` and then associated with specific routes in `webapi.xml`.

Different ways to access WebAPI resources including admin access

To access protected API endpoints, you must be authenticated. One way to do this is through the `/rest/V1/integration/admin/token` endpoint (token authentication). This is good for machine-to-machine interaction.

If no authentication is provided, Magento assumes `guest` access. If a valid session is passed, Magento initializes the `frontend` session and attempts to authenticate as a customer (note that this does not work for an admin session).

Finally, Magento allows Oauth 1.0 (important that this is NOT Oauth 2.0). This allows a user to configure access.

Further Reading:

- Authentication

5.2 DEMONSTRATE UNDERSTANDING OF THE ADMIN LOGIN PROCESS AND ADMIN ACTIONS PROCESSING

Admin login, customizing and debugging issues related to admins logging into Magento, the Admin Action class

Magento uses an `around` plugin for the `Magento\Backend\App\AbstractAction::dispatch` method to either log in an admin user or ensure they are logged in.

Here is an overview of what happens:

- `aroundDispatch` checks:
  - If the user is logged in, the ACL permissions are refreshed for the session.
  - If the user is NOT logged in:
    - If the form key matches and there is a `login` post parameter: `\Magento\Backend\Model\Auth::login` is called with the username and password specified in login.
    - `\Magento\User\Model\User::login()`, then `\Magento\User\Model\User::authenticate()` is called.
    - Inside this class, the public `verifyIdentity()` is called. This ensures that the password matches. It also ensures that this user is active and is assigned to a role.
    - You can use either events (`admin_user_authenticate_after`) or plugins to modify the functionality of this method.
    - The login is recorded to the database.

As you can see, there are many access points to customizing the admin login process.

All adminhtml controllers should extend the venerable `\Magento\Backend\App\AbstractAction` class. This class ensures that users are properly logged in and the `formKey` or URL `secretKey` is validated.

Practical experience:

- Set a breakpoint in: `\Magento\Backend\App\Action\Plugin\Authentication::aroundDispatch`
- Log into the admin panel.

Other examples:

- Google Backend Login
- MSP TwoFactorAuth

Debugging the login process

See above for information on how to do this.

Logging in an admin user programmatically

The process is as follows:

- Obtain a `\Magento\User\Model\User` class with a valid user.
- Call `\Magento\Backend\Model\Auth\StorageInterface::setUser($user)` to assign the user with the current admin session.

- Call `\Magento\Backend\Model\Auth\StorageInterface::processLogin()` to setup the session and refresh the ACL.

As long as proper security measures are followed, a developer can completely bypass the default Magento authentication system and easily provide login access through a different mechanism. One example would be using Google Oauth to provide single sign-on functionality.

Customizing the login process: for example, adding 2-factor authentication

The login process can easily be customized. Above I detail the numerous places where you can attach functionality to the login process.

In the example of 2-factor authentication, Riccardo Tempesta has created an outstanding module that adds 2FA for any Magento adminhtml area. I recommend using this module on every Magento 2 website. However, at least, review the code as it is a solid implementation of this concept.

Further Reading:

- MSP TwoFactorAuth
- Specifically: ControllerActionPredispatch.php

Operations performed by the `Magento\Backend\App\Action` class, for example, the secret key

If you look at this class, you will see it is blank. This class extends `\Magento\Backend\App\AbstractAction`.

Here are some of the adminhtml-only functionalities available:

- `dispatch()`: ensures that the user is logged in and that they have access to the `const ADMIN_RESOURCE` method. Note that descendant controller actions can override this `const` instead of specifying the `_isAllowed` method. Attaching a plugin to this method ensures that your functionality will be executed on every admin page.
- `_validateSecretKey`: every URL in the Magento admin has a secret key. This prevents cross-site request forgery where a malicious script takes advantage of the fact that Magento trusts you.
- There are quite a few view helper methods (`_addContent`, `_addLeft`, `_addJs`, etc.) that give the controller better access to the view layer.

Further Reading:

- `\Magento\Backend\App\AbstractAction`

5.3 DEMONSTRATE AN ABILITY TO CREATE COMPLEX FORMS AND GRIDS

In this section, we will focus on a high-level overview on how to complete each action with some demonstration and further reference points.

Complex forms with custom elements and with tabs.

This will be discussed later in this section.

Complex grids with custom columns and inline editing customizations

Custom columns are created:

- Use a `column` tag in the uiComponent `xml` file.
- If you want to customize the data that is going to the column, you can specify a `class` attribute that extends the `\Magento\Ui\Component\Listing\Columns\Column` column.
- If you wish to utilize a custom renderer, specify the `component` attribute. Your component will likely extend `Magento_Ui/js/grid/columns/column`.
- You can also set a `settings` node, or use `<argument name="data" .../>` for a custom settings tree.

Inline editing is enabled:

- `app/code/Chapter5/BackendCustomization/view/adminhtml/ui_component/custom_customer_grid.xml`
- Create the `columns/settings/editorConfig` area.
- Add the `editor/editorType` configuration to each column that is editable.

Practical example:

- In `app/code/Chapter5/BackendCustomization`
- Run `bin/magento module:enable Chapter5_BackendCustomization`
- Go to the Magento admin > Customers > Chapter 5 Backend Customization

Further Reading:

- How inline edit works in admin ui-components grid Magento2?

Create custom elements for forms

To create a custom form element:

- Inside your `field` declaration, add a `formElements/component` tag.
- If you want to update functionality, create a Javascript file to extend the original uiComponent (the default is `vendor/magento/module-ui/view/base/web/js/form/element/abstract.js`).
- If you want to change the HTML template, create a new HTML template and reference that in the `template` property.

You can also add a button to any existing uiComponent like:

```
<listing>
    <settings>
        <!-- adding a new button is as easy as: -->
        <buttons>
            <button name="add">
                <url path="*/*/edit"/>
                <class>primary</class>
                <label translate="true">Edit Customer</label>
            </button>
        </buttons>
    </settings>
</listing>
```

Practical example:

- `app/code/Chapter5/BackendCustomization/view/adminhtml/ui_component/custom_customer_form.xml`
- See `firstname` field.

Further Reading:

- Declare a custom UI Component

Create a form with tabs

This is created with the `fieldset` component.

Note that, by default, Magento puts fields in different fieldsets into an array with that fieldset's name. For example, `general` is the default fieldset name. All fields inside `general` will be POSTed in the `$_POST['general']` array, like `$_POST['general']['my_field_name']`. If you create another fieldset named `customer`, all fields in that fieldset would be accessed in `$_POST['customer']['customer_name']`. The use of `$_POST` in the example is for the sake of simplicity.

If you wish to keep all POST data in the same array, specify `fieldset/settings/dataScope`, like:

```
<fieldset name="customer">
    <settings>
        <dataScope>data.general</dataScope>
    </settings>
</fieldset>
```

Further Reading:

- How to add Tab in Form Ui Component
- Fieldset Component

Create a form with a grid inside of a tab

To add a grid inside of a form's tab:

Add a `htmlContent` element. This element will somewhat switch the "mode" from uiComponents to layout XML.

Add a `block` element with a class that implements `\Magento\Ui\Component\Layout\Tabs\TabInterface` and extends `\Magento\Backend\Block\Widget\Grid\Extended`. This is the "old" style grid (think Magento 1) but is often used for embedding grids.

Further Reading:

- `vendor/magento/module-rma/view/base/ui_component/customer_form.xml`

Forms for editing related/nested data

This is the same concept as above. Unfortunately uiComponents operate on a fairly linear level and are not extensible beyond that.

Customization of inline editing in a grid; for example, a file uploader

The field-type to editing field type is defined in the file below in further reading. You can use a mixin to create a new mapping in `defaults/templates/fields/[FIELD TYPE]`.

Further Reading:

- `vendor/magento/module-ui/view/base/web/js/grid/editing/record.js`

Bookmark filters selection for a grid

To include bookmark functionality, add this to your grid uiComponent:

```
<bookmark name="bookmarks"/>
```

You can also specify the namespace in which to store the bookmark data:

```
<bookmark name="bookmarks">
    <settings>
        <storageConfig>
            <namespace>my_namespace</namespace>
        </storageConfig>
    </settings>
</bookmark>
```

Further Reading:

- How to use Magento 2 UI Components

Grid meta information: adding a new column that requires a join to another table

This happens in the `DataSource`. See the practical example below for a sample on how to complete this.

Practical example:

- `app/code/Chapter2/UiDemonstration/Plugin/AddProductDetailsToDataProvider.php`

6 CUSTOMIZING THE CATALOG

14 questions

Professional Developer Plus

SWIFTOTTER SOLUTIONS

6.1 DEMONSTRATE AN ABILITY TO UNDERSTAND AND CUSTOMIZE MAGENTO PRODUCTS

Selecting the right product type for a given requirement (configurable with custom options, bundle with grouped).

Here are the product types which are included with Magento:

- Simple (basic unit of inventory)
- Configurable
- Group
- Virtual (base building block for non-inventory items)
- Bundle
- Downloadable
- Gift Card (Commerce)

I find the major delimiting factors are these:

- Does the item require inventory? Simple product (yes), virtual or downloadable (no).
- Is this item a container for other items? Group, configurable or bundle.

You can create a configurable product that has custom options. Note that you are unable to specify a percent price type for a custom option.

However, a grouped product only allows simple and virtual product types as participants in its selection choice (see `vendor/magento/module-bundle/etc/product_types.xml`).

Practical experience:

- Create a configurable product and assign some custom options to it. How does it appear on the frontend?
- While we **assume** we are experts in the Magento admin panel, are we? Ensure you have tested combinations and somewhat memorized the admin panel.

To create a new product type (overview):

1. Configure the new product type in your module's `etc/product_types.xml`.
2. If the product will be saleable, add the product type to your module's `etc/sales.xml` in the `order/available_product_type` node (see `vendor/magento/module-bundle/etc/sales.xml`).
3. For any attributes that you wish to have applied to this new product type, you may need to update the `catalog_eav_attribute` table. By default, `apply_to` is blank and a blank value means that this attribute applies to all product types. However, if `apply_to` is not blank, you need to append the new product type to this list.
4. Configure your price renderer in the `catalog_product_prices` layout handle (see `vendor/magento/module-bundle/view/base/layout/catalog_product_prices.xml`).
5. Add your product type to the `checkout_cart_item_renders` layout handle to configure how your item will be rendered in the cart.
6. Add your product type to the admin and frontend renderers (see the `view/[area]/layout` directory in `vendor/magento/module-bundle`).

Example product declaration from the bundle product:

```
<!-- vendor/magento/module-bundle/etc/product_types.xml -->
<type name="bundle" label="Bundle Product" modelInstance="Magento\
Bundle\Model\Product\Type" composite='true' indexPriority="40"
sortOrder="50">
    <priceModel instance="Magento\Bundle\Model\Product\Price" />
    <indexerModel instance="Magento\Bundle\Model\ResourceModel\
     Indexer\Price" />
    <stockIndexerModel instance="Magento\Bundle\Model\
     ResourceModel\Indexer\Stock" />
    <allowedSelectionTypes>
        <type name="simple" />
        <type name="virtual" />
    </allowedSelectionTypes>
    <customAttributes>
        <attribute name="refundable" value="true"/>
    </customAttributes>
</type>
```

Important notes:

- Note the separate price and indexer models. Price calculates the product price on the product page and thereafter. The indexer calculates the price before the product page (search and category).
- The `allowedSelectionTypes` node determines which products can be included in this product (see that the `composite` attribute is `true`).

- The `customAttributes` node allows you to configure specifications about a product that can be used in filtering product types. As far as I can see, this has nothing to do with a product model's custom attributes.

Further study:

- `vendor/magento/module-bundle/`
- `vendor/magento/module-grouped/`

Deciding to use non-standard products: for example, for licenses, subscriptions, courseware, or glasses. Product relations (related, upsells).

In deciding which product type to utilize, the first thing to determine is whether or not you can use something that already exists.

Magento product types solve for the majority of use cases but not everything.

We solved for this on SwiftOtter.com by creating a new product type: Test. This allows us to build custom functionality surrounding the handling and layout of this product type.

Select a product type for a subscription/subscription bundled with a physical product

Unfortunately, the intent of this objective is unclear. Ultimately, we have two products in use: a subscription and a physical product. The subscription will likely be an extension of a virtual product. The grouping of the two will likely

be either a grouped product or a customized version (kit) where users cannot select how many of each product they want.

Select a product type for courseware

This objective is also unclear. If the courseware will be built into Magento, you will begin building with a virtual product (remember, no inventory, so a simple product is not necessary). If the courseware is downloadable, then use the product type with that same name.

Select a product type for glasses with a prescription

This would be a simple product. You can upload the prescription with a custom option.

Compare custom options with configurable products

Custom options are a great way to customize a specific product (unit of inventory). This would include engraving a name on a coffee mug or powder coating a steel fitting for a surcharge.

Configurable products are a way to choose a specific unit of inventory (simple product). The product is not being customized but rather mapped (1-1) to the unit of inventory.

Configurable product with custom options for the associated simple products

Simple products with associated options are not allowed in a configurable product. If custom options are needed (by default), they must be applied to the configurable product and not the child.

Bundled product with custom options for its associated simple products

Bundled products do not allow simple products with custom options.

What is the related products database structure? Understand the performance impact of having many related products

Related products are stored in `catalog_product_link`. The types of product relations are stored in `catalog_product_link_type`. The definition for relationship attributes is stored in `catalog_product_link_attribute`. The values for these attributes are stored in `catalog_product_link_attribute_[decimal, int, varchar]`.

Having a large number of product relationships will slow down both editing the product as well as the product view page (possibly the shopping cart page, too).

Compare related with upsells

Upsells are a bigger and better product that (hopefully) the merchant will make more money selling.

Related products are an add-on product that complements the product being viewed. These products are easily added to the cart with a checkbox.

Further Reading:

- Related Products
- Up-sells

Programmatically access related products, custom options, configurable parameters

Related products:

The most performant way to get the list of related products is to go to the Link resource model: `Magento\Catalog\Model\ResourceModel\Product\Link::getChildrenIds($parentId, $typeId)`. You can also load specific relation types through the `Magento\Catalog\Model\Product` class (for example, `getUpSellProductCollection()`).

Custom options:

The `Magento\Catalog\Model\Product` class offers the `getCustomOptions` and `getOptions` methods. Additionally, you can use `\Magento\Catalog\Model\Product\Option::getProductOptionCollection()` to obtain the custom options for a specific product.

Configurable parameters:

A very performant way to retrieve a configurable product's children is with the `getChildrenIds()` method in `\Magento\ConfigurableProduct\Model\ResourceModel\Product\Type\Configurable`.

The `\Magento\ConfigurableProduct\Model\Product\Type\Configurable` class contains easy ways to obtain information about the attributes and product that are associated with a configurable product. Note that this class can be accessed from a product's `$product->getTypeInstance()` method. However, blindly assuming that you are working with a `Configurable` class may yield some PHP fatal errors, so ensure you do proper type checking.

Here are some of the most important methods:

- `getUsedProductAttributes`: this method returns the associated attributes for a configurable product.
- `getUsedProducts`: This returns a list of all associated products. The intersection of products and selected attributes is what determines the final select product.

Dynamic related products

To make a related product list truly dynamic, the best solution is to use plugins for the applicable method in the `Magento\Catalog\Model\Product` class (`getCrossSellProductCollection`, `getUpSellProductCollection` and `getRelatedProductCollection`). With these methods, you can entirely replace the collection that is being used.

6.2 DEMONSTRATE AN ABILITY TO PERFORM COMPLEX OPERATIONS WITH THE MAGENTO PRICING FRAMEWORK

Understand the pricing calculation and rendering framework. Which classes are involved in rendering/ calculation?

Prices are figured in the `\Magento\Catalog\Model\Product\Type\Price` class, or a derivative thereof. There is no interface to define a standard structure for these classes.

There are several types of prices (simple product):

- Price: the basic value that is loaded from the database.
- Tiered price: a discount that is available based on the quantity ordered, website, and customer group.
- Special price: a price reduction that is in effect for a specified period of time.
- Base price (accepts qty parameter): minimum of the tiered price and special price.
- Final price: base price plus options amount.

Practical example:

- Set a breakpoint on line ~33 (`renderAmount`), and step through the chain for each product type.
  - `vendor/magento/module-configurable-product/view/base/templates/product/price/final_price.phtml`
- Product price types:
  - `\Magento\Catalog\Model\Product\Type\Price`

- `\Magento\ConfigurableProduct\Model\Product\Type\Configurable\Price`
- `\Magento\Bundle\Model\Product\Price`
- `\Magento\Downloadable\Model\Product\Price`
- `\Magento\GiftCard\Model\Catalog\Product\Price\Giftcard`
- `\Magento\GroupedProduct\Model\Product\Type\Grouped\Price`

Configurable product

Calling `getPrice` ("Price" above) checks to see if a child product has been set. If it has, the price is calculated based on the child product. Otherwise, `0` is returned.

Bundle product

Overrides the Final Price calculation to add the total bundle items price to the base price. Remember that bundle products still have a base price.

Downloadable product

Overrides the Final Price calculation to allow downloadable item links to be purchased separately.

Giftcard

Overrides Price and Final Price to specify the custom option value for the amount that was specified.

Grouped

Overrides the Final Price calculation to add the totals for the associated (child) products found in the `associated_product_[child product id]` custom option.

Special price

Special pricing has two extra fields: start and stop dates for the new price. You might notice that these fields are missing in Commerce. The reason is that Commerce uses the staging start and stop dates. As such, the `special_from_date` and the `special_to_date` fields are loaded the same as normal attributes.

Further Reading:

- `Magento\CatalogStaging\Observer\UpdateProductDateAttributes::execute`

Price rendering

To render a price, Magento has created the `catalog_product_prices` layout handle (`vendor/magento/module-catalog/view/base/layout/catalog_product_prices.xml`). A good example of how to take advantage of this functionality is found in the `\Magento\Catalog\Block\Product\ListProduct::getProductPrice` method:

```
$this->getLayout()->getBlock('product.price.render.default')->render(
    \Magento\Catalog\Pricing\Price\FinalPrice::PRICE_CODE,
    $product,
    [
        'include_container' => true,
        'display_minimal_price' => true
    ]
);
```

Through a roundabout way (`vendor/magento/module-catalog/view/base/layout/empty.xml`) this handle is included on every page in the frontend.

Magento then allows you to specify specific renderers and templates for each product type.

Default price renderers:

- Default: `Magento\Catalog\Pricing\Render\PriceBox`
- Amount: `Magento\Framework\Pricing\Render\Amount` Used to render an amount. For example, on the product list page (category), this renders the "As low as: $12" for configurable products.
- Final Price: `Magento\Catalog\Pricing\Render\FinalPriceBox` As guessed, this ensures a product is salable, then it prints the final price for a product.
- Configured Price: `Magento\Catalog\Pricing\Render\ConfiguredPriceBox`

There is a specific order for loading these renderers (`simple` product type as an example):

- `simple/prices/final_price/render_class`
- `simple/default_render_class`
- `default/prices/final_price/render_class`
- `default/default_render_class`

What is the role of indexing? How do different price modifiers work together?

Indexing calculates the prices, and range of prices, for products. The goal of this is to eliminate redundant pricing calculations and, instead, store these values in the database.

The product indexer inserts prices (originating in `\Magento\Catalog\Model\ResourceModel\Product\Indexer\Price\Query\BaseFinalPrice::getQuery`) into a temporary table. Then, the pricing modifiers take over and apply their logic and adjustments to the values in this table. Once the pricing modifiers are complete, all items are moved from the temporary table, ultimately, to the final price index table.

Pricing modifiers:

These modify and adjust prices that are visible (see `\Magento\Catalog\Model\ResourceModel\Product\Indexer\Price\BasePriceModifier::modifyPrice`). For example, if products are to be hidden when they are out of stock, the `CatalogInventory` adjuster deletes indexed prices for all out of stock items.

- `Magento\CatalogInventory\Model\Indexer\ProductPriceIndexFilter`
- `Magento\CatalogRule\Model\Indexer\ProductPriceIndexModifier`
- `Magento\Catalog\Model\ResourceModel\Product\Indexer\Price\CustomOptionPriceModifier`

Practical experience:

Create a PHP Script runner in PHPStorm to execute `bin/magento indexer:reindex catalog_product_price`.

Set a breakpoint in `Magento\Catalog\Model\ResourceModel\Product\Indexer\Price\SimpleProductPrice::executeByDimensions`.

You can also set breakpoints in each class that implements `Magento\Framework\Indexer\DimensionalIndexerInterface`.

How is a price calculated on the product detail page, the product listing page? Price calculation classes. Relation to the price indexer

Pricing on the product detail page (and all pages thereafter in the customer journey) is calculated on-the-fly.

Pricing on the product listing page is loaded from the index.

As far as I can see, there is no connection of price calculation classes to price indexing. The reason for this is they approach the price calculation problem from two angles and are used to solve two different problems.

Price calculation classes operate on additional information. For example, they utilize the quantity to determine the product's final price. They use custom options which detail the child products included to figure how much the children cost.

Price indexing is used to display an initial price, very fast, on lists of products. There, the extra information does not matter.

Exception: when previewing category pages in Commerce staging, note that the indexers are bypassed.

The Magento\Framework\Price[sic]* component and its extension in the Catalog module

All Magento pricing entities extend the `\Magento\Framework\Pricing\Price\AbstractPrice` (which implements `\Magento\Framework\Pricing\Price\PriceInterface`).

In the Catalog module, `TierPrice`, `CustomOptionPrice`, `SpecialPrice`, `RegularPrice`, `FinalPrice` and `ConfiguredPrice` all extend this class.

Practical experience:

- Set breakpoints in the following, and browse the frontend:
  - `Magento\Framework\Pricing\Render::render`
  - `Magento\Framework\Pricing\Render::renderAmount`
  - `Magento\Framework\Pricing\Render\Amount::_toHtml`

How to render a product price: which class to use, which data needs to be provided

See above discussion on "Price rendering" that details how to accomplish rendering a price.

Indexed price vs. prices calculated on the fly

See above discussion for the differences that these offer.

Price configuration: tier prices, special price, custom option, configurable adjustment, catalog rules, cart rules

Tier prices

Tier prices are a way to provide discounts at quantity. Prices are configured for combination websites and customer groups (or neither).

Special prices

Special prices is another way to say "sale." In Open Source, there are three fields: Special Price, Special Price From (start date), and Special Price To (end date). With Commerce, there is only Special Price as Scheduled Updates are used to configure the start and end dates.

Custom options

These allow a visitor to change information about a product. For example, customers can specify initials to be engraved on a widget using a custom option. These can appear similar to a configurable product in that they both offer a drop-down menu selection (in addition to text and image upload). The general rule is that custom options make modifications to a specific unit of inventory.

Configurable adjustment

Configurable products are a way to aggregate simple (child) products each of which represents a specific unit of inventory. Once the options on a configurable product are selected, the configurable product is simply a wrapper for the chosen child product. These are different from custom options because the configurable product determines the unit of inventory to ship. Custom options modify the unit of inventory to be shipped.

Catalog rules

Catalog rules adjust pricing before items are added to the cart. User-defined attributes, categories and attribute sets are available for selection as a condition. Action is the discount.

Cart rules

Cart rules adjust pricing after items are added to the cart. These are applied by website and customer group. You can offer a coupon code.

If you wish to apply a cart rule by customer segment, you can specify this as a condition. In addition to product attributes—total, weight, and shipping destination are available as conditions.

One confusing point is that both conditions and actions have conditions. The conditions tab determines whether or not to activate this price rule. The actions tab defines which products to activate the rule for. A good example is free shipping for specific products. Maybe some products are too heavy to economically ship. You want to calculate live rates for the heavy products, but exclude the tally for the light products.

Practical experience:

- Create a product with a custom option.
- Create a configurable product: can you add custom options to a configurable product?
- Create a catalog rule. What conditions are present? How can these conditions be modified?
- Create a cart price rule. Offer free shipping on just women's jackets. Shipping should calculate for all other products added to the cart.

How to change the price rendering process for a given product type/product instance

Because Magento utilizes a layout handle for product pricing, changing the base block to render prices is easy. Additionally, you can use plugins and override templates.

Further Reading:

- Default price renderer: `Magento\Catalog\Pricing\Render\PriceBox`
- Default final price renderer: `Magento\Catalog\Pricing\Render\FinalPriceBox`
- Default configurable price renderer: `Magento\Catalog\Pricing\Render\ConfiguredPriceBox`
- `vendor/magento/module-bundle/view/base/layout/catalog_product_prices.xml`
- `Magento\Bundle\Pricing\Render\FinalPriceBox`

How to add a custom price adjustment to modify the calculation process. What happens if the price index is not aligned with this change?

Plugins. They are one of the best inventions since the invention of sliced bread (at least in Magento-land). Or, if you need access to protected methods, you can create a preference to entirely replace the calculation.

Remember, calculations are used for the product page and beyond in the customer journey. If these changes are not synchronized, customers will see one price on the category page and another on the product page.

6.3 CUSTOMIZE CATALOG PRICE RULES

Programmatically create a catalog price rule, the impact of catalog price rules on performance, extending catalog price rule conditions with custom entities

Programmatically create a catalog price rule

To create a catalog price rule, you follow these steps:

- Replicate the data structure from the `$_POST` for creating a price rule in admin (see practice example).
- Feed this information into an empty price rule model and call `loadPost`.
- Save it.

Practical example:

Create a catalog price rule programmatically. The basic Magento example is found in: `\Magento\CatalogRule\Controller\Adminhtml\Promo\Catalog\Save::execute`. Set a breakpoint here, and create a rule in the admin panel. This will give you the data structure to pattern in creating your own rule.

Impact of catalog price rules on performance

Catalog rules are indexed. They are also staging enabled. As such, there can be a very large number of rows in the `catalogrule*` tables. The indexers combine the results in the `catalogrule_product_price` with the other indexed prices.

The greatest performance impact would be when calculating product prices on the product page. If, for some reason, thousands of price rules are used, this could cause all uncached product pages and shopping cart / checkout pages to load slow.

Further Reading:

- `\Magento\CatalogRule\Model\Rule::calcProductPriceRule`

Extending catalog price rule conditions with custom entities

Conditions are found in `\Magento\CatalogRule\Model\Rule\Condition` namespace. To add or adjust price rule conditions, you would tap into `\Magento\CatalogRule\Model\Rule\Condition\Combine::getNewChildSelectOptions`.

6.4 DETERMINE HOW TO USE MAGENTO CATEGORIES

Advanced category features: hierarchy, custom attributes, impact on performance

Hierarchy

Categories are able to be ordered in a parent / child hierarchy (you are probably surprised to hear this). The hierarchy information is stored in the `catalog_category_entity` table. The pertinent columns are `path`, `position` and `level`. The base category is always ID #1 and is not visible in the admin panel: all categories inherit from ID #1. The path is the path of the entity IDs (not row IDs).

Custom attributes

Categories like other EAV entities use an attribute set and have attributes assigned to them. When creating most attributes, category attributes included, the attribute is automatically added to every applicable attribute set.

The big difference for category attributes is you need to create the uiComponent XML to make the attribute visible on the category edit page.

Further Reading:

- Add a Category Attribute

Practical example:

- `app/code/Chapter4/EAV/view/adminhtml/ui_component/category_form.xml`

What happens if a project has many categories?

Having many categories will affect the performance of the edit category page in the admin. On the frontend, navigating from one to another category should have little performance impacts unless a category has a very large number of sub-categories and filtered navigation is active.

One of the best ways to improve performance in categories is to enable the flat tables (Store > Configuration > Catalog > Catalog). Unlike products, flat category tables contain all attributes. Additionally, the source model does not define the column structure. This is determined in `\Magento\Catalog\Model\Indexer\Category\Flat\AbstractAction::getEavColumns`. Also note that having a massive number of attributes can bump into MySQL column limits, causing unintended errors or functionality (Magento has no limiting mechanism).

Dynamic rules for the order of products in category

Dynamic rules for the order of product in a category are found in the `\Magento\VisualMerchandiser\Model\Sorting` namespace and inherit `\Magento\VisualMerchandiser\Model\Sorting\SortInterface`.

Unfortunately, the list of dynamic sorting attributes is hardcoded so you will need to use plugins to adjust the items in the list of sorting attributes.

The category `is_anchor` attribute and its effect on performance

`is_anchor` does two things:

- Turns on layered navigation. This helps customers find products easier.
- Displays all products in child categories in this category. The benefit of this is that product lists can be "drilled-down" by category or attribute. The downside is that effectively managing the products displayed can become unwieldy. This can contribute to poor user experience as products are not curated and organized.

Further Reading:

- Anchor categories and position sorting explained
- What is 'best practice' when setting anchor on multi-level categories?

6.5 DEMONSTRATE AN UNDERSTANDING OF CATALOG INDEXERS

The indexing framework, the price indexer, the inventory indexer, the EAV indexer. How does Magento use indexed data?

Indexers are declared in the `etc/indexer.xml` file. Here is an example declaration:

```
<indexer id="catalog_product_price" view_id="catalog_product_price"
  class="Magento\Catalog\Model\Indexer\Product\Price">
    <title translate="true">Product Price</title>
    <description translate="true">Index product prices</description>
</indexer>
```

You can configure an indexer to depend on another indexer using the `dependencies` node (see `vendor/magento/module-catalog-rule/etc/indexer.xml`). For example, the `catalogrule` index is reindexed before the `catalog_product_price` index.

The list of indexers is available in the indexer collection `\Magento\Indexer\Model\Indexer\Collection`. If you need to add an indexer but only enable it in specific conditions (like enabling a module), you can put a plugin on the `\Magento\Indexer\Model\Indexer\Collection::getItems` method.

Practical experience:

- Create a PHPStorm PHP Script debug configuration to execute this command: `bin/magento indexer:reindex catalog_product_price`.
- Set a breakpoint on the first line of `\Magento\Indexer\Console\Command\IndexerReindexCommand::execute`.

The big picture is that the indexers are represented by the `\Magento\Indexer\Model\Indexer`. There are three methods relating to updating index rows: `reindexAll`, `reindexRow` and `reindexList`.

The action instance (what does the work) must implement `\Magento\Framework\Indexer\ActionInterface`.

Important Note:

Running the indexer too often can impact the user experience. Prices might be $0, for example, if the page is loaded while the new price indexes are being inserted into the price index table.

Price Indexer

The price indexer determines the lowest price from all of the pricing sources (price, special price, tier price, catalog price rules). This data is stored in `catalog_product_index_price`.

The price indexer iterates through each product type's `indexerModel` entry. These instances should implement `\Magento\Framework\Indexer\DimensionalIndexerInterface` (although there is backward compatibility not to use this method).

What are dimensions? Dimensions are a way to execute the indexer in multiple threads (processes) so indexing is run in parallel for each product type. Since Magento 2.2.6, indexers can be executed in multiple threads separated by Website and Customer Group (see `\Magento\Catalog\Model\Indexer\Product\Price\DimensionModeConfiguration`). The PHP process is forked into as many child processes as necessary per indexer.

To run a reindex using parallel dimensions, set the `MAGE_INDEXER_THREADS_COUNT=3` before running the reindex process.

This is especially helpful to ensure that large catalogs with many tiered prices are indexed efficiently without taking down the entire site.

Indexers are then run in batches (see `\Magento\Framework\Indexer\BatchProvider::getBatches`) to prevent memory errors.

Pricing data is utilized in the category view.

Further Reading:

- View indexer status

Practical experience:

- Set a breakpoint in each method in `\Magento\Catalog\Model\Indexer\Product\Price`. How can you trigger each method (`reindexAll`, `reindexRow` and `reindexList`)?
- Step through the creation of the price indexes.
- How does setting the `MAGE_INDEXER_THREADS_COUNT` environment variable make a difference in the reindex process?
- Set a breakpoint at the end of `\Magento\Catalog\Block\Product\ListProduct::initializeProductCollection`.
  - Then, run `$collection->getSelect()->__toString()`. You can observe the SQL joins that include the indexed tables.

Inventory indexer

The inventory indexer determines the number of items in stock and whether or not the item is available to be ordered.

The primary indexer class is `\Magento\CatalogInventory\Model\Indexer\Stock`. Each type of index operation (full, list, ID) has their own indexing class. Each

product can specify its own stock indexer in the `stockIndexerModel` class. This is utilized in bundle, configurable, and grouped products because their stock status is dependent on the status of child products.

The EAV indexer

This is used to aggregate attributes for search. With the introduction of ElasticSearch for Open Source in Magento 2.3 and the deprecation of MySQL search, this indexer will become irrelevant.

Estimate indexing customization efforts when making architectural decisions. Estimate indexing process complexity and time for different given conditions. The impact of indexing for frequent catalog updates.

Experience is necessary to achieve this objective as there is more information than can be written here.

Indexing customizations are among the more difficult to make in Magento 2 and to do it right. The majority of indexing operations aggregate data with complex SQL queries. See `\Magento\Catalog\Model\ResourceModel\Product\Indexer\Price\DefaultPrice::getSelect` for an example of such a query.

If indexing is occurring frequently, browsing the category pages can become unusable. That is because some indexing tables are truncated and data is copied back in. If a customer browses to the category page while the tables are empty, the `INNER JOIN` queries will eliminate results, and the empty category listing will be cached by the full-page caching.

What steps does Magento perform when indexing? What is the role of the `indexer_state` table

The steps that Magento takes are outlined in detail above.

The `indexer_state` table stores the state of the indexers (aptly named). There are three states: `working`, `valid` and `invalid`.

How to register a new indexer

Declare the indexer in the `etc/indexer.xml` file. Your indexer must implement `\Magento\Framework\Indexer\ActionInterface`.

How does the price indexer work? Which events trigger it? How do different product types declare their indexers?

Most of this is covered above, except for: "which events trigger a price indexer?"

- Command line `indexer:reindex`
- Cron, for rows that have been updated.
- Saving a product.
- Mass updating products.

Practical experience:

- Set a breakpoint in `\Magento\Catalog\Model\Indexer\Product\Price::executeRow`.
- Save a product.
- Evaluate the call stack—how does the indexer get triggered?

How important is an order of indexers in price indexing?

The order is very important. Price indexers are dependent on the `catalogrule` indexer successfully completing. Otherwise, the pricing will not take into account these rules.

Inventory indexer overview, inventory indexing for different product types

This is covered above.

What is the scope of a price in the price index? How difficult is it to extend the scope of an index?

Magento's default is that prices are in the global (default) or website scope. However, all of the pricing in the `catalog_product_index_price` table is for the website scope.

Changing the scope to, say, the store view would involve changing quite a number of touchpoints. In regards to indexing, you would need to update all pricing index tables (and the SQL queries to populate the data). This would likely involve adding an extra join or query parameters.

Keep in mind that price modifiers are available to adjust prices once they are in the temporary price index table (see `\Magento\Catalog\Model\ResourceModel\Product\Indexer\Price\SimpleProductPrice::executeByDimensions`).

This is an example of a SQL query: `\Magento\Catalog\Model\ResourceModel\Product\Indexer\Price\Query\BaseFinalPrice::getQuery`

Impact of many stores/websites on price indexing

Every website (not store) that is present will increase the number of rows in the `catalog_product_price_index` table.

There are four conditional columns: `entity_id` (product ID), `customer_group_id`, `website_id` and `tax_class_id`. To figure the number of rows in the table, multiply the number of valid entries for each of these columns (except tax class). If you have 2,000 products, 4 customer groups, and 2 websites: 16,000 rows. If you have 3 websites: 24,000 rows.

As you can see, the number of rows can grow exponentially.

Custom price modifiers: Pros and cons of customizing the index versus adjusting the native features like price rules or custom options

Anytime you can avoid modifying price indexes, you are saving yourself maintenance. While it is possible to do so, utilizing or repurposing native features is often the best way.

The upside to customizing the index is that price lookups will be faster. The downside is it takes more work to accomplish this.

Remember that indexing is useful only for pricing and information shown on category and search pages (also in related/cross-sell/up-sell).

6.6 DEMONSTRATE UNDERSTANDING OF CATALOG STAGING AND ITS IMPACT ON THE SYSTEM

Flow modifications introduced by staging (triggers, `row_id`, data versions). Staging-related modifications of the indexing process

I am unable to find any triggers related to staging (they are present when setting the indexer to reindex on save).

The `row_id` primarily affects the flat tables (see `\Magento\CatalogStaging\Plugin\Helper\Product\Flat\FlatColumnsDefinition` and `\Magento\CatalogStaging\Plugin\Helper\Product\Flat\FlatIndexes`).

However, the question you might ask is: "How does the indexer only index the active product?" The answer was discussed a few chapters back. There is a Select renderer (`\Magento\Staging\Model\Select\FromRenderer::render`) that adds the filters for the currently-active product version.

Further Reading:

- Magento 2 Partial Index

This was also discussed previously. When joining in the `catalog_product_entity` table, Magento will automatically add the appropriate filter. However, when joining a product to attribute values, you must join on the `row_id` and not `entity_id`.

Catalog triggers

See above.

6.7 DEMONSTRATE AN UNDERSTANDING OF THE PRODUCT SEARCH FRAMEWORK

How to customize Elastic search

ElasticSearch is found in:

- `vendor/magento/module-elasticsearch`
- This module depends on the official ElasticSearch PHP Composer package.

While you can customize the Magento portion of the ElasticSearch implementation, the only official ElasticSearch customization is configuring stop words. Stop words are a language's most-used words but are the least relevant for searches (example: a, and, the, their).

Read the DevDocs link below to understand how to update stopwords for a given locale.

Further Reading:

- Install and configure Elasticsearch
- Configure Elasticsearch stopwords

6.8 DEMONSTRATE UNDERSTANDING OF IMPORTING PRODUCTS AND CATEGORIES IN MAGENTO

Frequent imports, massive imports, import with many attributes

Background:

Before we can understand the effects of imports at scale, we must look at what happens in an import.

- Product imports happen in: `\Magento\CatalogImportExport\Model\Import\Product::_saveProducts`
- When importing products, this is what happens for all products:
- Categories are created (example: `Default Category/Women/Bottoms/Shorts,Default Category/Collections/Erin Recommends,Default Category` would create each category path that is separated by a comma). These IDs are assigned to a cache for later use.
- Images are copied from `pub/media/import` to `pub/media/catalog`.
- Tax class is created if it doesn't already exist.
- The row data is converted to attribute values (via the source model, see

`\Magento\CatalogImportExport\Model\Import\Product\Type\AbstractType::prepareAttributesWithDefaultValueForSave`). Note that commas in unintended places can break the import.

- The row data is iterated:
  - Attribute is loaded.
  - Attribute value is converted to a database-friendly value.
- `\Magento\CatalogImportExport\Model\Import\Product::saveProductEntity`
  - In `before` plugin, Staging sets the `created_in` and `updated_in` fields to defaults.
- Data is inserted into the `catalog_product_entity` table.
  - `\Magento\CatalogImportExport\Model\Import\Product::_saveProductWebsites`
  - Product IDs are associated with websites in the `catalog_product_website` table.
- `\Magento\CatalogImportExport\Model\Import\Product::_saveProductCategories`
  - Products IDs are associated with categories in the `catalog_category_product` table.
- `\Magento\CatalogImportExport\Model\Import\Product::_saveProductTierPrices`
  - Tier pricing is inserted into the database.
- `\Magento\CatalogImportExport\Model\Import\Product\MediaGalleryProcessor::saveMediaGallery`
  - Media gallery is inserted or updated.
- `\Magento\CatalogImportExport\Model\Import\Product::_saveProductAttributes`
  - Attributes are inserted or updated en masse (not line-by-line).

- `catalog_product_import_bunch_save_after` event is triggered
    - Generates URL rewrites.

After the above process happens, each product type can run custom actions (for example, to automatically associate parent configurable products with children). Running these actions after the initial import ensures that all SKUs are present for their association.

Links are then saved.

Inventory is saved. This is when the catalog_product reindex process happens.

Product option options are saved.

There are several after plugins for the primary import method (`\Magento\ImportExport\Model\Import::importSource`). This is an example where the sort order for plugins is critical.

- All product-related indexes are invalidated (twice).
- Catalog permissions are imported.

The reindex that runs with the every-one-minute cron will pick up these changes and reindex them.

Frequent imports

The risk with frequent imports is largely dependent on their size and how much data is being inserted. Frequent imports with few products would have little impact on website performance. But, if there are many attributes, site performance could suffer due to all of the attributes needing to be loaded.

Massive imports

Magento has streamlined this as best as possible to use mass-inserts and updates. However, these new imports will need to be re-indexed and that is likely the biggest sticking point.

Import with many attributes

This could be problematic no matter where the attributes are used in Magento. For the import, Magento loads these attributes in from the database and stores them in memory. There is the potential that you could run into memory limit errors if there are too many attributes.

Issues related to a frequent import, for example every minute

See discussion above.

Compare importing products using the native import, save by model, or custom SQL

Native import should be the most reliable as the attribute's data is validated, non-existent categories are created, and media files are properly moved around. Additionally, Magento provides access points to ensure that all systems are properly adjusted (options, configurable associations, related products).

Save by model is the slowest approach. But, it is quite easy to do. However, the indexing is called for each product that is created.

Custom SQL would be slightly faster than native import. The challenge is ensuring that all data is present and properly inserted into the database. Also, one must be careful that exceptions (for example, missing categories) are handled appropriately.

Specifics of importing a catalog with many attributes and attribute sets

If there are a large number of attributes, spread out over many attribute sets, you might consider breaking up the imports by attribute set. This will reduce the number of attributes loaded in memory and might resolve out of memory issues.

Importing categories and product relations

Categories are imported by populating the `categories` column. Place the path of category names, each separated by a "/", with category paths separated by a ",". Magento will create categories with these names.

Keep a couple of things in mind:

- Trailing spaces are not trimmed.
- Magento will try to create a category even for a blank name (ex. accidentally adding in two slashes "//").
- Putting commas into category names will break the category paths.

Product relations are imported in the respectively-named columns. These are comma-separated.

7 CUSTOMIZING THE CHECKOUT PROCESS

10 questions

Professional Developer Plus

SWIFTOTTER SOLUTIONS

7.1 UNDERSTAND THE MAGENTO QUOTE ARCHITECTURE AND CUSTOMIZING QUOTE-RELATED FUNCTIONALITY

Quote-related objects, total models and the price calculation process, the add to cart process, custom add to cart operations, customizations of the price calculation, taxes and discount, various display settings

Quote-related objects

Quote functionality is in the `Magento\Quote` namespace. Here are the most notable classes:

- Quote entity: `\Magento\Quote\Model\Quote`
  - Product (item): `\Magento\Quote\Model\Quote\Item`
  - Address: `\Magento\Quote\Model\Quote\Address`
    - Shipping Rate: `\Magento\Quote\Model\Quote\Address\Rate`
    - Item (items that are assigned to an address): `\Magento\Quote\Model\Quote\Address\Item`
    - Totals: `\Magento\Quote\Model\Quote\Address\Total\Grand` (for example)
  - Payment: `\Magento\Quote\Model\Quote\Payment`

The above outlines how quotes are structured.

Total models

Totals come together to ultimately generate the grand total for an order.

Note: totals in Magento are comprised of two numbers, the total and the base total (for example, grand total and base grand total). The base total is the total in the base currency of the website (Store > Configuration > General > Currency). The total is in the currency of the store view.

Totals are calculated per address. The amounts are tallied and assigned to the quote object. Unless a store is using multi-address checkout or the customer is checking out with a virtual cart, the total calculators are run twice: once for billing and once for shipping.

Total calculators usually inherit `\Magento\Quote\Model\Quote\Address\Total\AbstractTotal`. They must implement `\Magento\Quote\Model\Quote\Address\Total\CollectorInterface`.

Practical experience:

- Set a breakpoint in: `\Magento\Quote\Model\Quote\TotalsCollector::collect` and add an item to the cart. Step through each segment of this process.
    - `\Magento\Quote\Model\Quote\Address\Total\Grand::collect` is always called for each calculation.

Price calculation process

The Subtotal (`\Magento\Quote\Model\Quote\Address\Total\Subtotal`) calculator determines a product's price in the cart:

- First, the final price is retrieved. Remember, the final price can take into account a quantity (it does, in this instance). However, the price that the final price returns is per-item.
- The quantity is multiplied by the final price. Quote Items can have a `custom_price` set. If this is set, the final price is discarded, and the custom price is used instead (`\Magento\Quote\Model\Quote\Item\AbstractItem::getCalculationPriceOriginal`).
- The item totals are added together to form the subtotal.

Practical experience:

Set a breakpoint in `\Magento\Quote\Model\Quote\Address\Total\Subtotal::_initItem` and add an item through the cart.

Add to cart process

The most common point for adding products to the cart is: `\Magento\Quote\Model\Quote::addProduct`. This method takes the following actions:

- Configures options, configurable options (`\Magento\Catalog\Model\Product\Type\AbstractType::_prepareProduct`)
- Determines the number of items to add to the cart.
- These products are represented as candidates to add to the cart.
- If this product is already in the cart:
    - `\Magento\Quote\Model\Quote\Item::representProduct` checks if the product's ID matches and if the options match.
    - If there is a match, this item is updated, instead of adding a new item to the cart.
- If the product is not in the cart:
    - The item is initialized (`\Magento\Quote\Model\Quote\Item\Processor::init`)

- Details are set on the item.
- The item is added to the quote.

Entry points for adding to the cart:

`\Magento\Wishlist\Controller\Index\Cart::execute`

`\Magento\Checkout\Controller\Cart\Add::execute`

`\Magento\Quote\Model\Quote\Item\Repository::save` (this one calls `setItems` instead of `addProduct` or `addItem`).

`\Magento\Sales\Model\AdminOrder\Create::addProduct`

Practical experience:

Set a breakpoint in `\Magento\Quote\Model\Quote::addProduct` and add an item to the cart.

Custom add to cart operations

The Magento add controller provides a good example (although the wrapper for the functionality is deprecated).

```
$quote->addProduct(/** … **/);
$quote->getShippingAddress()->setCollectShippingRates(true);
$quote->collectTotals();
$this->quoteRepository->save($this->getQuote());
```

It is important to call `collectTotals` to ensure that the quote has the latest information regarding the items.

Customizations of the price calculation

To override the price in the cart, use the `custom_price` data key. This overrides the standard price calculation (using `getFinalPrice`).

Practical experience:

- `\Chapter7\FreeGift\Plugin\AddFreeGiftToCart`
- Utilizes `custom_price` to adjust the price of an item in the cart.
- Note that this example is very basic and does not cover all scenarios.

Taxes and discount

- Magento provides broad support for many types of taxes:
- VAT
- Sales tax
- Fixed product tax (`Magento\Weee`). This allows you to configure flat product taxes. In Europe, WEEE is a flat tax for each electronic sold, to cover recycling it. In Store configuration, this is called Fixed Product Taxes. To enable for a product, you must create a product attribute with the Fixed Product Tax attribute type.

Further Reading:

- Magento_Weee module

Practical experience:

- Review Store Configuration > Sales > Tax, Checkout
- Configure a fixed sales tax rate for a store.
- Configure VAT rates for a store.

The quote merge functionality

The method is found in `\Magento\Quote\Model\Quote::merge`.

Each item is compared. The method that compares the quote items is: `\Magento\Quote\Model\Quote\Item\Compare::compare`. If the comparison returns `false`, a new item will be added. If the comparison returns `true`, the quote item's quantity is increased.

So, what's in the compare method?

- The item's options are retrieved.
- The option codes (keys) are compared using the `array_diff_key` method. If there is any variation, a merge is not possible.
- The values are also compared. Note that these comparisons are using non-strict equality.

Practical experience:

- Set a breakpoint in `\Magento\Quote\Model\Quote::merge`.
- Add an item to your cart.
- Log into your customer account.

Programmatically add a free gift when a certain condition is met

The most simple use-case is demonstrated with a plugin. There are a couple of other plugins you will need to add in order to create robust functionality: handling for when other cart items are updated or deleted.

Practical experience:

- Set a breakpoint in `\Chapter7\FreeGift\Plugin\AddFreeGiftToCart::afterAddProduct`
- Add an item to your cart.
- Extend the example to cover product updating and deleting.

Programmatically set a price for a product

This is done with the `setCustomPrice()` and `setOriginalCustomPrice()` methods. Once these are set on an item, they override the product's final price and serve as the price for the subtotal calculation.

Taxes with discounts calculation

This is an area of tax law compliance. Some jurisdictions require taxing after the discount is applied and others (it seems quite rare) require taxing before the discount is applied. The default is the former: tax is calculated on the discounted price. This is configured in Store > Configuration > Sales > Tax > Calculation Settings > Apply Discount on Prices.

The impact of many shopping cart price rules on performance

The performance impact comes for when there are many rules with no coupon code and they are automatically applied. The conditions are first filtered by Website, Customer Group, and Coupon code (see `\Magento\SalesRule\Model\Validator::_getRules`).

Every time the cart's totals are collected, these rules are re-applied for each item (see `\Magento\SalesRule\Model\RulesApplier::applyRules`). The conditions are filtered, and the actions are run. This can become very expensive.

Practical experience:

- Set a breakpoint in `\Magento\SalesRule\Model\Quote\Discount::collect` and navigate to the cart page.

Programmatically create shopping cart price rules

The `$_POST` data is filtered and applied to the `\Magento\SalesRule\Model\Rule` model using the `loadPost` method. It is then saved.

Practical experience:

Set a breakpoint in `\Magento\SalesRule\Controller\Adminhtml\Promo\Quote\Save::execute` and save a new Cart Price Rule.

Importing coupon codes

There is no way in the admin to import coupon codes.

Coupon codes are stored in the `salesrule_coupon` table. Create a CSV and import this file directly into the table, and the coupon codes will be available.

Further Reading:

- How to Import Coupons Codes in Magento 2

Extend the shopping cart price rules with custom entities

Good news! There are multiple ways to add new conditions to Sales Rules (unlike Catalog Price Rules). You can use plugins (`\Magento\SalesRule\Model\Rule\Condition\Combine::getNewChildSelectOptions`) or the `salesrule_rule_condition_combine` event.

Rules are added with the following notation:

```
$rule[] = [
    'value' => \MyCompany\MyModule\RuleCondition\CustomRule::class,
    'label' => __('My Custom Rule')
];
```

Programmatically separate line items in the shopping cart so there are two line items instead of one with qty=2

There are two methods that handle this function.

First, is when you add another product to the cart (that has the same product ID as already in the cart). `\Magento\Quote\Model\Quote\Item::representProduct` handles this comparison. The product IDs are checked, then whether or not the product is a child versus a parent, and finally the item's options.

Second, is when carts are merged. There is very similar functionality (with the exception of the parent/child item check) in `\Magento\Quote\Model\Quote\Item\Compare::compare`.

Practical experience:

- Set a breakpoint in `\Magento\Quote\Model\Quote\Item::representProduct` and add a similar item to the cart.

Adding a new total model and evaluating its impact on taxes and discounts

Totals are added in `etc/sales.xml`:

```
<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="subtotal" instance="MyCompany\MyModule\
               Total\Fee" sort_order="150"/>
        </group>
    </section>
</config>
```

The impact on taxes and discounts is determined by the `sortOrder` attribute. The higher the sort order the more totals will already have been calculated. Grand total's sort order is `550`.

Discounts and taxes are calculated on each item and not on the order's subtotal or other totals. As such, new totals do not affect these items.

Further Reading:

- How to Use Total Models in Magento 2

Shipping discounts behavior and customizations

This is configured through Cart Price Rules. The actions tab contain the updates that are made to the cart.

One option is to apply the discount to the shipping amount. Or, if you are like most merchants, you will just give the customer free shipping.

WebAPI for quote operations. Create a quote, add an item, create a coupon, add a discount

The WebAPI is a powerful option for managing the frontend customer experience.

Note that there are two types of endpoints for accessing the cart: `/rest/V1/carts/mine/items` and `/rest/V1/guest-carts/{cartId}`. The difference is that with a guest cart, you have to store and manage the cart ID. Initializing a guest cart requires calling `POST /rest/V1/guest-carts/` and storing the `quote_id`. The challenge, then, is integrating the guest cart from the API with the real checkout. The API does not set cookies, and you will likely need to build a custom controller (which can set cookies) to assign the current session's quote ID to be the API's quote ID.

Note that the checkout loads the quote ID from `window.checkoutConfig.quoteData.entity_id` (`vendor/magento/module-checkout/view/frontend/web/js/model/quote.js`).

We opted for a Javascript demonstration for this as many backend developers touch Javascript at least once in their lifetime (possibly more?).

Practical experience:

- Log into a Magento customer account.
- https://lc.commerce.site/chapter7quoteapi
- `app/code/Chapter7/QuoteApi/view/frontend/web/js/items.js`

Further Reading:

- Rest API's

7.2 DEMONSTRATE AN ABILITY TO CUSTOMIZE AND EXTEND THE CHECKOUT PROCESS

Checkout steps, the REST API, customizations of the checkout API, the order placement process. The impact of many concurrent order placements

Checkout steps

#1 Shipping: enter the shipping address and choose shipping method.

#2 Payment: choose payment method, set billing address, apply discount code, gift card, and store credit.

Practical experience:

- Run through the checkout process yourself. It's pretty simple, add items to the cart and click the Proceed to Checkout button.

The REST API

The default Magento checkout utilizes the REST API for persisting checkout details. This enables easy extension points for customizations to add details.

However, many times it is safer to create a new REST API route than to modify an existing route. With the ease of utilizing Javascript mixins, this is quite simple to do.

Practical experience:

- Open Chrome Developer Tools and set a breakpoint in `vendor/magento/module-checkout/view/frontend/web/js/model/shipping-save-processor/default.js` (or `/static/versionMYVERSION/frontend/Magento/luma/en_US/Magento_Checkout/js/model/shipping-save-processor.js`), the `saveShippingInformation`. Select a shipping method, and click Next.

Further Reading:

- Familiarize yourself with the REST API calls in the `quote` namespace: Rest API's

Customizations of the checkout API

This is discussed in more detail forthcoming.

Further Reading:

- Customize Checkout

The order placement process

Javascript implementation details:

- The submit button is located inside of the payment method form (example: `vendor/magento/module-offline-payments/view/frontend/web/template/payment/checkmo.html`).
- Payment method renderers inherit (`vendor/magento/module-checkout/view/frontend/web/js/view/payment/default.js`).
- Clicking the Place Order button normally calls `placeOrder` in the above Javascript class.
- This calls the Place Order action (`vendor/magento/module-checkout/view/frontend/web/js/action/place-order.js`).
- Endpoints:
    - Customer logged in: `/rest/V1/carts/mine/payment-information`: `Magento\Checkout\Api\PaymentInformationManagementInterface::savePaymentInformationAndPlaceOrder`
    - Not logged in: `/rest/V1/guest-carts/:quoteId/payment-information`:
    - `Magento\Checkout\Api\GuestPaymentInformationManagementInterface::savePaymentInformationAndPlaceOrder`

PHP implementation details:

- Guest: `\Magento\Checkout\Model\GuestPaymentInformationManagement::savePaymentInformationAndPlaceOrder`
- Logged in: `\Magento\Checkout\Model\PaymentInformationManagement::savePaymentInformationAndPlaceOrder`
- Both end up calling `\Magento\Quote\Model\QuoteManagement::placeOrder`.

Actions:

- Magento first checks to see if this is a B2B customer and if they have permission to place orders.
- `checkout_submit_before` is called.

- `\Magento\Quote\Model\QuoteManagement::submitQuote`
- The quote is validated (see implementations of `\Magento\Quote\Model\ValidationRules\QuoteValidationRuleInterface`).
- If a customer checksout, the quote is updated with the customer address information (actually, a customer can be created if the `$quote->getPasswordHash()` has a value).
- *The order ID is reserved.*
- The order and order address objects are created from the quote.
- The payment is converted from the quote to the order.
- The items are converted.
- Order data is set.
- `sales_model_service_quote_submit_before` is triggered.
- Place order is called:
  - B2B sets order extension attributes (if applicable).
  - `\Magento\Sales\Model\Order::place`
  - `sales_order_place_before`
  - Payment is authorized or captured. (`\Magento\Sales\Model\Order\Payment::place`)
  - `sales_order_place_after`
  - Coupon usages are updated. (`\Magento\SalesRule\Plugin\CouponUsagesIncrement::afterPlace`)
  - Order is saved.
- `sales_model_service_quote_submit_success` is called
  - This reindexes the catalog inventory index. (`\Magento\CatalogInventory\Observer\ReindexQuoteInventoryObserver::execute`)
  - Order email is sent, if there is no redirect flag. (`\Magento\Sales\Model\Order\Email\Sender\OrderSender::send`)

- `checkout_submit_all_after`
- The checkout session contains the latest order and quote information.

Practical experience:

- Set a breakpoint in `\Magento\Checkout\Model\GuestPaymentInformationManagement::savePaymentInformationAndPlaceOrder` and `\Magento\Checkout\Model\PaymentInformationManagement::savePaymentInformationAndPlaceOrder`.
- Step through the checkout process for both guest and customer.

The impact of many concurrent order placements

As you can see in the checkout action outline above, checkouts touch a majority of the Magento system's major features. This means that checkout is likely the most difficult place to scale. One impact is the single point of failure: the inventory catalog. This is improved in Magento 2.3.

Adding a step to the checkout after the payment step, or between the payment and shipping steps

Adding a step after the payment step involves a number of touchpoints. First, it is probable you will need to modify each payment type as the Place Order button is found in the payment template. You can likely use a mixin to modify existing functionality, but there will be a number of permutations to check.

Adding a step between the shipping and payment steps involves registering the step with the correct sort order (see example in further reading).

Further Reading:

- Customize Checkout
- `vendor/magento/module-checkout/view/frontend/web/js/model/step-navigator.js`
- `vendor/magento/module-checkout/view/frontend/web/js/view/shipping.js`, `initialize` method

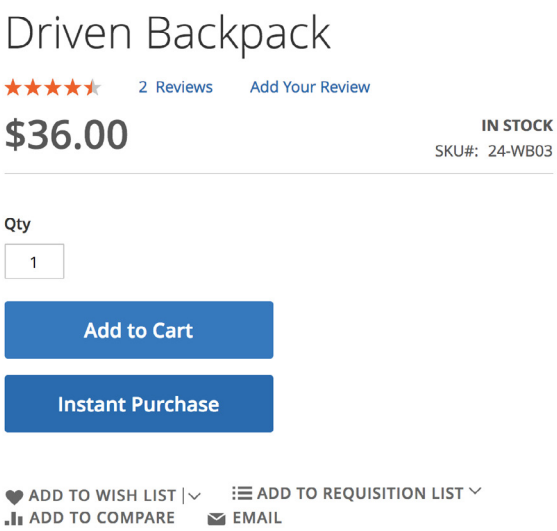
Implementing a "one-click" checkout, evaluate possible pitfalls such as discounts being applied or canceled when the order is placed

I am interpreting this as instant checkout.

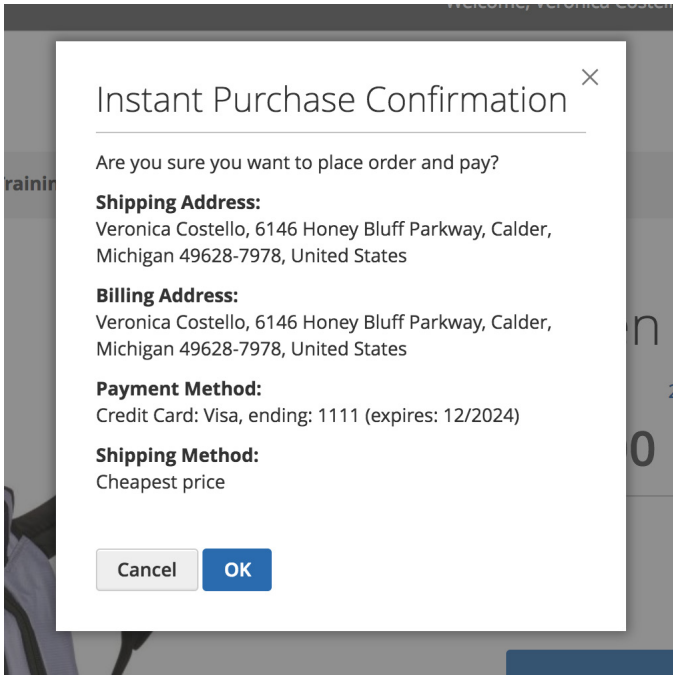
This is possible with:

- A logged-in customer
- A payment method that is enabled for vault and instant purchase (Braintree Credit Card, Braintree PayPal, and PayPal Payflow Pro)

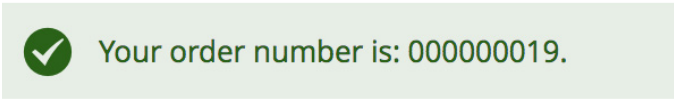
Product page:



Confirmation window:



Success:



The biggest danger is that the customer is charged more than they are expecting. Ensuring good customer communication is the most important part of these considerations.

Practical experience:

- Sign up for a Braintree sandbox.
- Configure Braintree in Magento.
- Set a breakpoint in `\Magento\InstantPurchase\Controller\Button\PlaceOrder::execute` and click Instant Purchase.

The checkout REST API: Modifying the native flow (separate calls to save payment and to place the order)

One thing to keep in mind here is utilizing payment types that send the credit card through the server. Unless the merchant is using an offline payment method, it is likely a poor idea to save payment, if even possible.

That is why the save payment information and place order are the same method.

Extending the checkout REST API. Adding new parameters to different API endpoints. Using extension attributes

This is covered above under their respective sections.

Issues related to simultaneous order placement, inventory locking

The problem is usually related to table locking on the inventory table. As described below, every time orders are placed, the order item quantity is decremented.

In these methods (`\Magento\CatalogInventory\Model\StockManagement::registerProductsSale` and `\Magento\CatalogInventory\Model\ResourceModel\Stock::correctItemsQty`), the stock is loaded and then saved. In some cases, the entire stock item row is saved. Other times, just the `qty` is updated. Either way, this is a single point that multiple processes can access at the same moment.

Determining the exact moment when the stock is decremented during an order placement

It's done during the `sales_model_service_quote_submit_before` and `checkout_submit_all_after` events. The latter is never (if ever) triggered as the former always runs first.

The `sales_model_service_quote_submit_before` event is triggered before the `try/catch` block surrounding the place order process

(see `\Magento\Quote\Model\QuoteManagement::submitQuote`). This means that an error could cause inventory to be out of sync. The good news is Magento thought of that and attached an observer to the `sales_model_service_quote_submit_failure` event (see `\Magento\CatalogInventory\Observer\RevertQuoteInventoryObserver`).

See:

- `\Magento\CatalogInventory\Observer\SubtractQuoteInventoryObserver::execute`
- `\Magento\CatalogInventory\Observer\CheckoutAllSubmitAfterObserver::execute`

Practical experience:

- Set a breakpoint in the above observers.
- Evaluate the call stack and step through the code.

Customizing the order placement such that it uses message queues

Instant checkout provides the inspiration on how message queues can be used to place orders. In instant checkout, all payment details have been stored in a vault and other order details can be saved to the quote. As such, this can be securely passed through the message queueing system.

Note that you will likely want to reserve the order increment ID before sending the quote to the queue. This is so that you can tell the user their order number.

You will also need to build notification mechanisms for if the payment method is declined and a way for the customer to change the payment details.

Horizontal sharding of orders tables to improve order placement capacity. What challenges need to be resolved?

Horizontal sharding of order tables is when you split the table out by rows. For example, orders placed with a last name that starts with A-M goes into one order table. Orders placed with a last name that starts with N-Z goes into another order table. The result is that there is less activity on each table.

The challenge is then knowing where to locate the orders. Looking up an order is no longer selecting by ID or increment ID. Instead, you need to know more information about the order.

Further Reading:

- Super-Scaling Magento
- Split database performance solution

7.3 CREATE AND DEBUG SHIPPING AND PAYMENT METHODS IN MAGENTO

Different types of payment methods: gateway, offline, hosted. The gateway payment methods framework. The payment method availability logic. The shipping rates calculation process, debugging shipping rates and table rates customizations

Gateway, offline, hosted payment methods

Payment methods pretty much fit into one of these categories. Gateway is when the payment details (for example, credit card) is sent to Magento and then on to the merchant. This can also apply to when the credit card is tokenized (Braintree or Stripe) on the frontend and the authorization/capture happens from Magento.

Offline is for payment types that are not connected to an external service. Examples of this are Check/Money Order, Bank Transfer, Purchase Order and Cash on Delivery (found in `\Magento\OfflinePayments`). These payment types offer a Credit Memo but only for the entire order and not per invoice.

Hosted payment methods involve redirecting the customer to a third-party page. This used to be the old PayPal Express in Magento 1. Now, the only two native types that are hosted in Magento are Payflow Express and Payflow Bill Me Later.

When a customer selects a hosted payment method in the checkout, they click a button (often blue, but the color may vary depending on the merchant's preferences) which takes them to the payment provider's website. Here, the payment details are entered and the provider communicates the relevant details back to Magento. PayPal communicates back with IPN.

Gateway payment methods framework

The majority of information online about creating online payment methods relies on the wonderfully simple extension of `\Magento\Payment\Model\Method\Cc`. This class (and the abstract payment method) is deprecated as of Magento 2.1. As such, I hope this brief overview will provide good insight.

The payment method class for the gateway method framework is `\Magento\Payment\Model\Method\Adapter`. This is the class on to which the configuration is bolted.

Instead of a central class that you create containing all information necessary to complete a payment transaction, the central class is fixed and relies on configuration to customize necessary functions (adapter pattern). To see this in action, enable either our example payment method or a sandbox for true online payment method and set a breakpoint in `\Magento\Payment\Model\Method\Adapter::getConfigData`.

Each payment method in the gateway payment method framework must have:

- A value handler pool (see `SuperPaymentsValueHandlerPool`). For the most part, this is a configuration loader. However, you can set custom handlers for any value. See the `can_void` handler. Observe the two virtual types configured to provide system configuration values

(`SuperPaymentsConfigValueHandler` and `SuperPaymentsConfigValues` in `app/code/Chapter7/PaymentMethod/etc/di.xml`). This allows the proper system configuration node to be referenced.

- A validator pool (see `SuperPaymentsValidatorPool`). No validators are necessary, but the class needs to be specified. This, combined with the virtual country validator (`SuperPaymentsCountryValidator`), demonstrates how to create custom validators. You can utilize the `global` validator to be always run.
- Command pool. This is a list of configured commands. See the next section.

Payment commands

Payment commands are specific actions to be performed on a payment. Here is a list of commands found in `\Magento\Payment\Model\Method\Adapter` (they match the method name):

- `fetch_transaction_information`
- `order`
- `authorize`
- `capture`
- `refund`
- `cancel`
- `void`
- `acceptPayment`
- `denyPayment`

To add a new command:

- Locate the command pool type (or virtual type) for the payment method you are modifying.
- Add the command to the `commands` argument.
- Create configuration for the command to extend `Magento\Payment\Gateway\Command\GatewayCommand`.
- The command must specify a `requestBuilder` (generating the request for the gateway), `transferFactory` (generating a transfer), `client` (sending the request), and the `handler` (receiving the data).

Further Reading:

- Adapter pattern

Practical experience:

- Run `bin/magento Chapter7_PaymentMethods`
- Run `bin/magento setup:upgrade`
- The amount of customization is far more than we have space to discuss here. Understand that the possibilities are limitless through configuration (and certainly with plugins, if necessary).
- Read through each class specified in `app/code/Chapter7/PaymentMethod/etc/di.xml`

Payment method availability logic

- Magento calls the `isActive` method on the payment method adapter.
- This calls `getConfiguredValue` on the adapter.
- This checks the value handler pool to see if handler is registered for this value. See `SuperPaymentsValueHandlerPool` for an example of how to register a new value handler. You would create one for the `active` type.
- If no handler is specified, and if the payment method is properly configured, `\Magento\Payment\Gateway\Config\Config::getValue` loads the value from store configuration.

Practical experience:

- Set a breakpoint in `\Magento\Payment\Model\PaymentMethodList::getList` and go to the checkout page.

Shipping rates calculation process

Above, we discussed address totals. The shipping rates calculation process follow this model. The calculator (`\Magento\Quote\Model\Quote\Address\Total\Shipping::collect`) extends `\Magento\Quote\Model\Quote\Address\Total\AbstractTotal`.

`\Magento\Quote\Model\Quote\Address::requestShippingRates` assembles a shipping rate request. This object should contain all information necessary to obtain shipping rates. Note the `FreeMethodWeight` is how much weight should NOT be calculated in the rate request. For example, if a package weight 2.5kgs, and the free method weight is 0.5kgs, rates will be retrieved for shipping 2.0kgs.

A request can also limit the carrier. This is based on the `limit_carrier` data key in a quote address object. This gives you an easy way to restrict options to one choice.

Other study points:

- `\Magento\Shipping\Model\Shipping::collectRates` iterates through all shipping providers. This is one place to understand why your carrier is skipped over.
- `\Magento\Shipping\Model\Shipping::collectCarrierRates` loads shipping rates from a specific carrier (if the carrier is active).

Shipping method calculators are similar to the old way of payment methods. There is one file that (usually) extends an abstract class and adds the lookup functionality.

Calculators return a `\Magento\Shipping\Model\Rate\Result` class. This is a container for storing `rates`. Rates are of type `\Magento\Quote\Model\Quote\Address\RateResult\Method`.

In our example, we have configured the FedEx carrier. An online carrier is one that is connected to a third-party service or live rating. One potential impact to understand is that when these services go down, it can bring a merchant's site down, too (we have had this happen).

By default, online shipping carriers offer a free shipping method and a free shipping up to option. This is seen in `\Magento\Shipping\Model\Carrier\AbstractCarrierOnline::getMethodPrice`. If free shipping does not apply to this method, the next step is to calculate the handling fee (`\Magento\Shipping\Model\Carrier\AbstractCarrier::getFinalPriceWithHandlingFee`). Handling fees can be

per package or per order. The number of packages is determined by maximum weight, and nothing smarter.

Practical experience:

- Set a breakpoint in `Magento\Quote\Model\Quote\Address\Total\Shipping::collect` and navigate to the checkout, enter your address, and step through the collection process.
- You can also configure an online shipping method (in this example, FedEx), and set a breakpoint in `\Magento\Fedex\Model\Carrier::collectRates`. Evaluate the call stack and step through the rate collection process. Note that the products ordered must have a weight.

Debugging shipping rates and table rates customizations

Above gives a good overview for debugging shipping rates. Most often, the shipping rate has been disabled (by a country exclusion, invalid credentials for online carriers, or errors returned from online carriers).

Table rates provide one carrier name for a list of rates (you cannot specify both, say, "Expedited Shipping" and "Standard Shipping"). Rates are uploaded **per website** and cannot apply to the global scope. The available columns for table rates are:

- Country
- Region/State
- Zip/Postal Code
- Order Subtotal (and above)
- Shipping Price

`\Magento\OfflineShipping\Model\ResourceModel\Carrier\Tablerate::getRate` locates the rates for the shipping request.

The gateway framework vs. AbstractMethod

The abstract method for payments has been deprecated. However, all offline payment methods still use it, so I am unsure the true status (it makes no sense to use the gateway framework for offline payment methods).

The structure of offline payment methods. What makes them "offline"?

Offline payments have no connection to a third-party gateway. Examples are payment by check, purchase order and manual bank transfer.

Partial invoices/refunds for payment methods

Payment methods must be enabled for partial invoices. The base payment method provider exposes two protected variables: `$_canCapturePartial` and `$_canRefundInvoicePartial`.

Add a hosted-type payment method (redirects, custom order review page)

`\Magento\Quote\Model\Quote\Payment::getOrderPlaceRedirectUrl` calls `getConfigData` on the payment adapter class. This means that we can use configuration to specify a redirect url, if pertinent.

The Payflow Express payment method provides an example of a custom order review page.

Add a new shipping method. Customize the logic for getting a list of available shipping methods

Perhaps the most simple example of a shipping method is: `\Magento\OfflineShipping\Model\Carrier\Freeshipping`.

Steps to create a new shipping method:

- Create default configuration in `etc/config.xml` for `default/carriers/[carrier name]`.
- Create store configuration options in `etc/adminhtml/system.xml`.
- Create a shipping carrier model that extends `\Magento\Shipping\Model\Carrier\AbstractCarrier` (for your sanity) and `\Magento\Shipping\Model\Carrier\CarrierInterface` (for Magento's sanity).
- If desired, add validation rules for your shipping type (see `vendor/magento/module-offline-shipping/view/frontend/layout/checkout_cart_index.xml`).

The shipping rate calculation flow. Debug shipping rates, identify plugins that that influence the shipping rate calculation process

This is discussed above.

Customizing table rates. Add a new "column" to the "table", change the logic of selecting the rate.

Table rates are stored in `shipping_tablerate`. Create a plugin for `\Magento\OfflineShipping\Model\ResourceModel\Carrier\Tablerate\RateQuery::prepareSelect` and add additional query parameters. The request is available in `\Magento\OfflineShipping\Model\ResourceModel\Carrier\Tablerate\RateQuery::getRequest`.

Gateway shipping methods. Remote call flow. How to avoid the remote call if it is unnecessary

`AbstractCarrierOnline` offers an easy way to cache quotes: `_getCachedQuotes($requestString);` It is your responsibility to generate the request cache key, but after that, the abstract carrier will handle the cache storage mechanism.

Practical experience:

- Enable and configure FedEx shipping in sandbox mode.
- Set a breakpoint in `\Magento\Fedex\Model\Carrier::collectRates`.
- Navigate to the checkout and enter your address information.

8 MAGENTO COMMERCE FEATURES

8 questions

Professional Developer Plus

SWIFT OTTER SOLUTIONS

8.1 DEMONSTRATE AN ABILITY TO USE MESSAGE QUEUES

Create a listener/publisher. Use cases for using message queues

Creating a message queue with a publisher and a consumer is quite simple. This will cover how to create one of these yourself.

There are several pieces necessary to realize a functioning message queue.

First, there are a couple of concepts to understand.

- A publisher (`etc/queue_publisher.xml`) is configured to send messages to a topic.
- A topic (configured in `etc/communication.xml`) is a way to categorize messages to consumers.
- A consumer is configured to listen for messages with specific topics.
- Queues route topics to consumers (this happens in `etc/queue_topology.xml`).
- Consumers accept messages and perform actions on them (`etc/queue_consumer.xml`).

Files to create:

- A schedule (publisher): creates an operation (`\Magento\AsynchronousOperations\Api\Data\OperationInterface`) and sends it to the publisher for the specific topic.
- A consumer: accepts the operation (`\Magento\AsynchronousOperations\Api\Data\OperationInterface`) from the queue and performs an action on it.

The DevDocs links below have more information. The goal of this section is to provide a basic working knowledge so you can get into the code and execute it yourself, giving a deeper understanding.

Practical example:

- See `app/code/Queueing/BasicMessageQueue`
- Run `bin/magento module:enable Queueing_BasicMessageQueue`
- Run `bin/magento setup:upgrade` (this installs the consumer)
- Run `bin/magento queue:consumers:start example_consumer`
- Then visit https://lc.commerce.site/index.php/chapter8queue/

Further Reading:

- AMQP 0-9-1 Model Explained
- RabbitMQ
- Message Queues
- Configure message queues
- How to Configure and use RabbitMQ in Magento 2.3

Use cases

Message queues are a powerful way to offload immediate processing needs to another process. One thing to understand is that queues are one-way: you send information to them, but it defeats the purpose (or is impossible) for the publisher to also receive the result.

Another way to look at message queues is similar to the Magento event system. You dispatch (or, publish) a message to the event system. This event contains

data (or, operation). Magento routes the message to the appropriate observer (or, consumer).

The Magento technical guidelines dictate that no changes are made to event object data (there are a couple of exceptions). Because of this, most event observers could become message queue consumers.

Examples:

- GDPR compliance in downloading customer data, then sending an email.
- Sending order email.
- Decrementing stock item quantities when products are ordered.
- Saving product data.
- All reindex operations.

I believe that Magento's admin interface will become more performant as the message queues are adopted.

How to register a listener/publisher for a queue

See above.

How to configure a queue

See above.

What is a use case to use message queues?

See above.

8.2 DEMONSTRATE UNDERSTANDING OF CUSTOMER SEGMENTATION

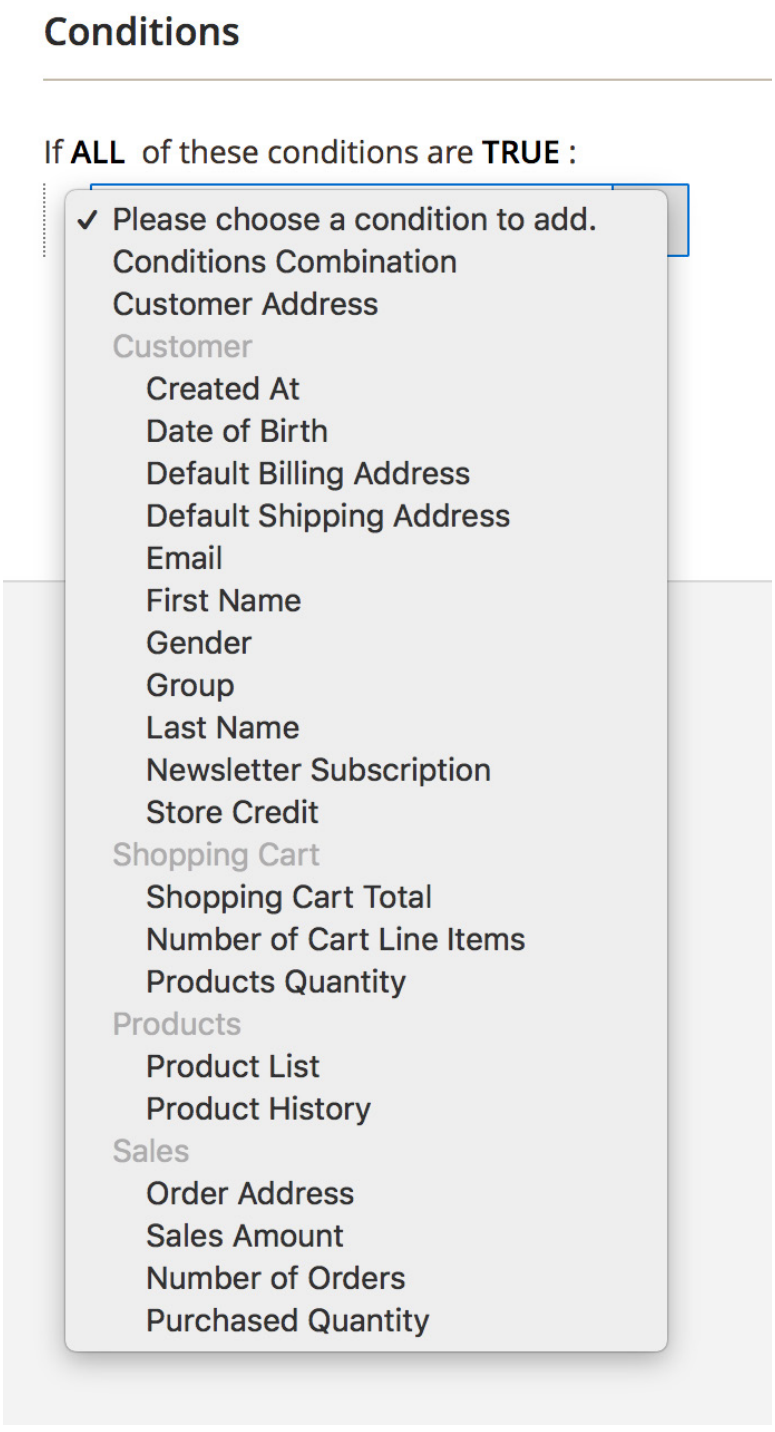
Describe the customer segments functionality, its use cases, impact on performance, and programmatical operations with customer segments

Describe the customer segments functionality

Customer segments allow a merchant to locate all customers that match a criteria. This is different than customer groups in that customers are assigned to a customer group, but segments pick which customers match the criteria.

Customer segments are found in Customer > Customer Segments.

Here is the default list of customer segment filters:



Practical experience:

- Create a new customer segment.
- Specify a criteria.
- Locate the customers.
- Try to change customer segment Apply To: does it work?

Customer segments are stored in:

- `magento_customersegment_segment`: stores base customer segment details. Note that both the serialized conditions and the condition SQL statement to fetch the customers are stored in this table.
- `magento_customersegment_website`: assignment of the customer segment to the website.
- `magento_customersegment_event`: events on which to refresh each customer segment.
- `magento_customersegment_customer`: customers that fit within the segment.

How it works:

Customer / segment associations are "indexed" and stored in `magento_customersegment_customer`. Events trigger updates to the associations stored in this table (see `vendor/magento/module-customer-segment/etc/events.xml` and `vendor/magento/module-customer-segment/etc/frontend/events.xml`). Most of the events are triggered when a customer takes a specific action.

Magento loads determine if the current event matches an event to be processed for a customer segment (see the `magento_customersegment_event` table).

For each segment that is valid for this customer, the customer is added to the matched segments (`\Magento\CustomerSegment\Model\Customer::addCustomerToWebsiteSegments`). This includes updating the database table (`magento_customersegment_customer`) and adding the segments to the session and the HTTP context.

Effect of Apply To:

If a customer segment's Apply To is set to Visitors, no customers will be matched. Instead, this rule will apply to the frontend.

Whenever a segment's rules are changed, all associated customers are deleted from `magento_customersegment_customer` and then re-added through `\Magento\CustomerSegment\Model\ResourceModel\Segment::aggregateMatchedCustomers`.

Use cases for customer segments

- Apply catalog and cart price rules (you need to use a condition for this).
- Display banners (dynamic blocks in 2.3).
- Reporting for marketing campaigns.

Impact on performance

Because customer segments are re-calculated each time a customer (or visitor) takes almost any cart or catalog-impacting action on the frontend, having many segments can take a toll on performance.

Practical experience:

- Set a breakpoint in: `\Magento\CustomerSegment\Model\Customer::processEvent`.
- Create a new customer segment and a customer that matches.
- Log in with that customer.
- Observe the actions taking place.

Programmatic operations with customer segments

- Fetch all applied segments for a customer: `\Magento\CustomerSegment\Model\Customer::getCustomerSegmentIdsForWebsite`
- Trigger customer segment refresh for a customer: `\Magento\CustomerSegment\Model\Customer::processEvent`
- Rebuild an entire segment's customer associations: `\Magento\CustomerSegment\Model\Segment::matchCustomers`

Programmatically create a customer segment

Similar to creating catalog and cart price rules, the easiest way to observe the process is to set a breakpoint in `\Magento\CustomerSegment\Controller\Adminhtml\Index\Save::execute`. Then, save a customer segment. Observe the `$model->loadPost()` and the data structure being pulled into the model.

Extend customer segments with new entities or attributes

New entity:

Create a class that extends `\Magento\Rule\Model\Condition\Combine`. Rules implement `_prepareConditionsSql` to add conditions to the SQL expression to locate customers.

Add a new condition with an `after` plugin on `\Magento\CustomerSegment\Model\Segment\Condition\Combine::getNewChildSelectOptions`:

```
public function afterGetNewChildSelectOptions($subject, $result)
{
    $result[] = [
        'value' => \MyCompany\MyModule\Model\Segment\Condition\
          MyCondition::class,
        'label' => 'My Condition',
        'available_in_guest_mode' => true
    ];

    return $result;
}
```

New attribute:

Customer segment customer attributes must: 1) have a label, 2) not be a file or image, 3) and be marked to use in a customer segment.

Understand the database structure of customer segments

Discussed above.

8.3 DEMONSTRATE UNDERSTANDING OF ADVANCED CAPABILITIES IN MAGENTO COMMERCE

Store credits, reward points, RMA, gift cards

Store credits

Store credits reside in the `Magento\CustomerBalance` module. They are found in Customers > [choose customers] > Store credit tab.

You can make all credit memos be automatically returned through Store Credit with Store Configuration > Customers > Store Credit > Refund Store Credit Automatically.

Practical experience:

- Create a customer.
- Add store credit to their account.
- Purchase something using the Store Credit payment method.
- Purchase something using a normal payment method.
- Refund latter order to Store Credit.

Reward points

Reward points create a loyalty program for customers. When customers take specific actions, they get X number of points. They can then use these points to purchase merchandise.

Reward points are configured in Store Configuration > Customers > Reward Points. The exchange rate (points conversion to dollar value) is in Stores > Reward Exchange Rates. Here, you can specify a website, Customer Group, direction (Points to Currency and Currency to Points), and the exchange rate.

Additionally, you can use a cart price rule to add a fixed number of reward points for a qualifying order.

The other important point to note is the configuration for Reward Points Expiry Calculation. This default to static, which means that when the balance increases, the expiration date is reset to the number of days specified. However, setting to dynamic means that you can change the days to expiration, and it will be figured into all future calculations.

Practical experience:

- Create a customer.
- Ensure reward points are enabled and configured to add points for specific actions.
- Place an order.
- Add a reward exchange rate.
- Make a purchase with reward points.

Further Reading:

- Reward Points System on Magento 2 Enterprise Edition

RMA

The RMA module provides a streamlined process for approving and accepting returns. This is found in `Magento\Rma`. Configuration is found in Store Configuration > Sales > RMA Settings.

Orders must be invoiced before they are eligible for a return.

Returns have their own attribute type (Stores > Attributes > Returns).
If you need to add a new type of refund, you can easily add it to the Resolution attribute.

Practical experience:

- Enable RMA in store configuration.
- Place an order and invoice it.

- In the customer account, locate the order and create a return.
    - Set a breakpoint in `\Magento\Rma\Model\ResourceModel\Item::getOrderItems` to observe the selection criteria for determining eligible order items.
- Process the return in the admin panel.
- Create a credit memo. This is not done automatically and requires the admin to complete and submit a credit memo.

In the admin panel, RMA items follow a workflow of authorizing items, receiving them (qty in the Returned field), and approving them. Note that the person receiving the return must also change the status each time a QTY field is updated.

Items

Product	SKU	Remaining	Requested	Authorized	Returned	Approved	Return Reason	Item Condition	Resolution	Status	Action
Overnight Duffle	24-WB07		1	1			Wrong Size	Unopened	Store C	Pendin	Details Split

If you need to make changes to the workflow, you can enjoy updating the old-style Magento forms (not uiComponents). Depending on your experience, this may or may not be a plus.

Further Reading:

- Configuring Returns
- Returns Attribute
- Magento 2 Product Return RMA

Gift Cards

Gift cards are a product type. They are found in the `\Magento\GiftCard` namespace. Gift cards are physical (shipped, thus shipping calculations are applied) or virtual (via email). Configuration is found in Store Configuration > Sales > Gift Cards.

Examples:

- https://lc.commerce.site/luma-mailed-gift-card
- https://lc.commerce.site/luma-virtual-gift-card

Purchased gift cards are visible under Marketing > Gift Card Accounts. By default, the gift card account is created when the order is invoiced. You can change this to create the gift card account when the order is placed (although, the problem is that the purchasing credit card might authorize but fail to capture and the customer now has a free gift card).

Each gift card has its own code that comes from a pool of generated codes (`magento_giftcardaccount_pool`). These codes are configured in store configuration. Once a card is associated with a gift card code, the `status` column for the code (in the pool table) is switched to `1`.

Further Reading:

- Creating a Gift Card
- Gift Card Workflow

Compare RMA with refunds

The RMA process provides a structured way to receive products. However, there is no connection with credit memos. An admin must track what products are returned and then create a credit memo for those items.

Customizing the RMA process

Because Commerce provides the ability to edit RMA attributes, this solves many customization needs for the RMA process.

However, here are a couple of other locations that are helpful to tap into RMA functionality:

- RMA item tab: `\Magento\Rma\Block\Adminhtml\Rma\Edit\Tab\Items\Grid`
- List items available to be returned: `\Magento\Rma\Model\ResourceModel\Item::getOrderItems`
- Don't forget that RMAs have extension attributes. This is a great way to export additional RMA information to an ERP.

Customizing store credits/reward points accrual/spending

Store configuration (Stores > Configuration > Customers > Reward Points) contains the majority of customizations necessary:

- Rewards Points Balance Redemption Threshold: prevents customers from using points until they have achieved a certain number of points.
- Cap Reward Points Balance At: prevents customer from obtaining too many points.
- Reward Points Expire in (days): have reward points fall off after this number of

days. See `\Magento\Reward\Model\ResourceModel\Reward\History\Collection::addExpirationDate` and `magento_reward_history`.

- Reward Points Expiry Calculation:
  - Dynamic: if this is set, when you change the Expire In (above) value, the expiration for rewards is also updated.
  - Static: if this is set, changing Expire In (above) value only changes new reward values.

Points for customizing reward points / store credits:

- `\Magento\Reward\Model\ResourceModel\Reward\History`
- `\Magento\Reward\Cron\ScheduledPointsExpiration::execute` (entry point for expiration customization)
- `\Magento\Reward\Cron\ScheduledBalanceExpireNotification::execute` (entry point for expiration customization)
- `\Magento\CustomerBalance\Model\Balance`

8.4 DEMONSTRATE UNDERSTANDING OF TARGET RULES

Customer rules database structure, use cases, comparison to other product relations, operate with target rules programmatically

Customer rules (Related Products Rules or Target Rules) are found in Marketing > Related Products Rules. The goal is to reduce administration time for

maintaining upsells, cross-sells, and related products while also providing intuitive selection for products.

Target Rules have these conditions:

- Apply to (upsell, cross-sell or related product)
- Customer segments

Target rules have two tabs: Products to Match and Products to Display. The former selects products that these rules apply to. The latter selects products to show on these products (in the list that is selected).

Further Reading:

- Creating a Related Product Rule

Customer rules database structure

- `magento_targetrule`: master table for storing target records. Note the `action_select` column which provides additional filtering based on the current product. For example, if you look at this column using sample data, many rules are filtering to ensure that a product is in a category (see the `Crosssell Women Top Jackets` rule).
- `magento_targetrule_customersegment`: table that specifies which customer segments the rule applies to.
- `magento_targetrule_product`: stores associations of rules that are associated with a product.

Note that there are other Target Rule tables (`magento_targetrule_index`, `magento_targetrule_index_crosssell`, `magento_targetrule_crosssell_product`, `magento_targetrule_index_related`,

`magento_targetrule_index_related_product`, `magento_targetrule_index_upsell`, `magento_targetrule_index_upsell_product`). I am unable to find where or how these tables are utilized, which leads me to believe they may be abandoned.

Here is an example of the filter stored in the `action_select` column (the embedded `SELECT`):

SELECT `e`.`entity_id` FROM `catalog_product_flat_1` AS `e`

WHERE (((SELECT COUNT(*) FROM `catalog_category_product` WHERE (product_id=e.entity_id) AND (`category_id` IN('5'))) > 0)) AND (e.entity_id NOT IN('1404')) LIMIT 4

Use cases

The primary use case to ease product associations. This is valuable because one way to improve a website's conversion rate and average order value is by showing additional products that are pertinent or interesting.

Comparison to other product relations

Target rule products can operate in addition to or replace products specified on a product level. Because target rules operate on a more global level, some product suggestions may not be overly relevant. In these cases, an administrator can add the required products to the applicable product list.

Operate target rules programmatically

To load a list of related products for a particular product (source is `\Magento\TargetRule\Block\Catalog\Product\ProductList\AbstractProductList::_getTargetRuleProductIds`):

```
/** @var $indexModel \Magento\TargetRule\Model\Index **/
$indexModel = $this->_getTargetRuleIndex()->setType(
    $this->getProductListType()
    // 1 = related, 2 = upsell, 3 = cross-sell
    // see \Magento\TargetRule\Model\Rule
)->setLimit(
    $limit
)->setProduct(
    $this->getProduct()
)->setExcludeProductIds(
    $excludeProductIds
);
if ($limit !== null) {
    $indexModel->setLimit($limit);
}

return $indexModel->getProductIds();
```

The above first configures a filter then it:

- Iterates through all applicable customer segments and loads products for each (see `\Magento\TargetRule\Model\ResourceModel\Index::getProductIds`). Before going to the database, the cache is consulted.

- Products are loaded by rule index and customer segment ID (default is `0` for no segment). See `\Magento\TargetRule\Model\ResourceModel\Index::_matchProductIdsBySegmentId`.
    - This method locates rules that apply to the conditions specified.
    - Product IDs are loaded with the `\Magento\TargetRule\Model\ResourceModel\Index::_getProductIdsByRule` method.

Practical learning:

- Set breakpoints in the classes mentioned above.
- Navigate to https://lc.commerce.site/nadia-elements-shell (target rule #2 applies to this product by default).
- Step through the product selection criteria.

Target rules vs. related products

See above.

Extending target rules with a custom attribute or a custom entity

To allow a custom attribute to be available for selection, ensure the attribute's type is a common type (i.e. not file, image, or fixed product tax), and it is visible (see `\Magento\Rule\Model\Condition\Product\AbstractProduct::loadAttributeOptions`).

To create your own target rule condition, create an `after` plugin for `\Magento\TargetRule\Model\Rule\Condition\Product\Attributes::getNewChildSelectOptions` to append another rule.

Create a rule programmatically, understand the database structure of a rule

The best way to accomplish this is by creating a rule in the admin panel and setting a breakpoint in `\Magento\TargetRule\Controller\Adminhtml\Targetrule\Save::execute`. Step through the code and replicate the data structure in your custom code.

What is the performance impact of having many target rules in the system?

For the initial page load (before the full page cache takes over responding to a page), having too many target rules will cause a bottleneck around the processing and application of the rules.

How does editing a product or adding new products affect existing target rules?

The rules are reindexed.

Practical experience:

- Set a breakpoint in `\Magento\TargetRule\Model\Indexer\TargetRule\AbstractAction::_reindexByProductIds`.
- Save a product that is associated with rules.

9 UNDERSTANDING MAGENTO SECURITY

3 questions

Professional Developer Plus

SWIFT OTTER SOLUTIONS

9.1 DEMONSTRATE UNDERSTANDING OF FRONTEND SECURITY WITH MAGENTO

Filter input and escape output; password and sensitive data hashing; validate user file uploads; how to prevent cross site scripting vulnerabilities

Filter input and escape output

Filtering input is important to prevent saving malicious code. Escaping is a secondary layer of protection to ensure that anything that made it through filters is rendered benign.

Magento provides a couple of form filters. These implement the `\Magento\Framework\Data\Form\Filter\FilterInterface` and provide the `inputFilter` and `outputFilter` methods.

Both customers and EAV abstract provide filtering when utilizing the `extractValue` method.

Practical experience:

- Set a breakpoint in `\Magento\Customer\Model\Metadata\Form::extractData`
- Create a customer.

Escaping output is available in blocks with the `escape*` methods in `\Magento\Framework\View\Element\AbstractBlock`:

- `escapeHtml`: essentially runs the`htmlspecialchars` method (unless you provide a list of allowed tags).

- `escapeJs`: escapes a string so that it can appear in quotes in Javascript.
- `escapeHtmlAttr`: escapes code echoed into an HTML attribute.

Further Reading:

- Escape Methods

Password and sensitive data hashing

Password hashing is a one-way "encryption" (hashing). Subsequent validations are hashed, and the hash is compared. This removes the need to save plain text or encrypted passwords.

The `\Magento\Framework\Encryption\Encryptor` class provides a standard set of tools to hash and encrypt/decrypt.

Practical experience:

- `\Magento\Customer\Controller\Account\CreatePost::execute`

How to prevent cross site scripting vulnerabilities

Cross-site scripting happens when a malicious user enters code into a website and the website renders that code in plain text (in the source).

Ensure that all output (whether loaded from the database or direct visitor input) is escaped. Even if an admin is in control of the output (for example, products), don't trust it (admin accounts are cracked).

Further Reading:

- Cross site scripting

How to validate field values before inserting them into the database

If possible, utilize and enforce types if the input value is an integer or float (and casting to that type). This eliminates many attack vectors. However, accepting integer and float values is not entirely possible.

The `Magento\Framework\Data\Form\Filter\FilterInterface` provides a standardized way to run specific filters against input:

- `\Magento\Framework\Data\Form\Filter\Striptags` ensures there are no HTML tags present.
- `\Magento\Framework\Data\Form\Filter\Escapehtml` renders HTML tags harmless.
- `\Magento\Framework\Data\Form\Filter\Trim` eliminates extra string characters.

How to escape output to remove HTML tags

In a block, call `$block->escapeHtml()`.

Using secure encryption and hashing functions to store sensitive values in the database

See above.

Validate file uploads, the file type, the file contents, and the file metadata

While the Zend library includes a good number of validators, its use is deprecated and will be eliminated. As such, we will focus on validating images.

The `validateUploadFile` opens the file to ensure that it is a valid image. If more careful validation is needed, use the new Zend validator (for example `\Zend\Validator\File\FilesSize`).

Practical experience #1 (example of Zend validation):

- Add a product option to a product of the file type.
- Navigate to the product on the frontend.
- Set a breakpoint in `\Magento\Catalog\Model\Product\Option\Type\File::validateUserValue`
- Set an image, and add the product to the cart.
- Pay careful attention to: `\Magento\Catalog\Model\Product\Option\Type\File\ValidatorFile::validate`

Practical experience #2:

- Set a breakpoint in `\Magento\Catalog\Controller\Adminhtml\Product\Gallery\Upload::execute`
- Set a breakpoint in `\Magento\Framework\Image\Adapter\AbstractAdapter::validateUploadFile`
- Open a product, and upload an image.
- Step through the above method.

9.2 DEMONSTRATE UNDERSTANDING ADMINHTML SECURITY WITH MAGENTO

CSRF tokens; stored XSS; adminhtml secret key; security configuration; securing email from injection; preventing admin privilege escalation

CSRF tokens

CSRF tokens ensure that the page was linked to from another validated page. The way it does this is use a session token that creates a hash that is utilized on the next page load. When you navigate to the linked page, the hash is verified against the session.

There are two Magento features built for this: backend URL keys (secret key) and form keys. These will be discussed in more detail later.

This effectively eliminates someone emailing you a link to your website that you click, and it does something malicious (if you are logged in).

Stored XSS

This is when a user saves malicious details into the database (using even legitimate means like creating a customer account or placing an order). While it is not good to store malicious code in the database, the double-trouble comes when rendering that code. In the case of the problem that SUPEE-7405 fixed (Magento 1), people could create an email address with Javascript code.

Further Reading:

- Magento CSFR Protection

Adminhtml secret key

This is that really long string of characters that is appended to each admin URL:

```
https://lc.commerce.site/index.php/admin_dev/admin/index/index/
key/5010facb5b81eba2e80c2daf41c5b7c3e7b94b5eba9314fd5265e176d7cd51cc/
```

The good news is we all know to **never** disable it.

Security configuration

Merchant-configurable options are found in Store > Configuration > System > Admin. These include setting a custom Admin Base URL, enforcing account security (like preventing account sharing, ensuring secret key is enabled, admin session length, forced password change, etc.).

In addition, it is important to leverage Role Resources (and Role Scopes in Commerce) to ensure that admins have limited access to data.

Practical experience:

- Familiarize yourself with Magento security configurations.

Securing email from injection

Ensure that reCaptcha (or, Magento's unfortunate implementation of a captcha idea) is enabled. This greatly reduces bots from creating Magento accounts for most everyone in the world.

Preventing admin privilege escalation

Privilege escalation is when a user is able to access something that they are not allowed to do but those with more permissions are.

Preventing this involves carefully writing code and keeping the Magento application up to date.

Example:

- https://magento.com/security/patches/magento-2.2.7-and-2.1.16-security-update, see PRODSECBUG-2153, PRODSECBUG-2063, PRODSECBUG-2126, PRODSECBUG-2152, and MAGETWO-94370

How to prevent CSRF attacks; the importance of the form_key field in forms

Magento's backend controllers automatically append the secret key (thanks to preferences specifying `\Magento\Backend\Model\Url`), so that part is easy unless someone disabled it.

The `form_key` is easy to add to forms, so it is a sign of a very poor quality developer (in my opinion) not to use this everywhere.

```
<form method="POST">
<?= $block->getBlockHtml('form_key') ?>
</form>
```

It's that easy.

All Magento adminhtml controllers should extend `\Magento\Backend\App\AbstractAction`. Extending this class automatically checks form keys for POST and secret key for all other requests.

How is the admin secret key generated in URLs?

Anytime the adminhtml (backend) `url` model is used to create a URL the route name, controller, action, and form key are appended and then hashed (see above).

Practical learning:

- Set a breakpoint in `\Magento\Backend\Model\Url::getSecretKey`.
- Navigate to an admin page.

Security related admin configurations

See above.

Security by obscurity, the custom admin path

While security by obscurity is not ideal as the ONLY means of security, it does provide an additional layer of security for a web application.

There are two places to specify the admin path:

- in `app/etc/env.php` in the `backend/frontName` array.
- in Store Configuration > Advanced > Admin.

Further Reading:

- Using a Custom Admin URL

9.3 DEMONSTRATE UNDERSTANDING OF DIFFERENT TYPES OF ATTACKS AND PREVENTING THEM WITH MAGENTO

Remote code execution; local and remote file inclusions; session hijacking; SQL injection; insecure functions; directory traversal attacks

Note:

I strongly recommend reading each link in this section. This information should be reviewed periodically to ensure it is at the forefront of your mind while customizing Magento applications.

Remote code execution

This happens when an attacker is able to get PHP executed. This could give the attacker access to system secrets, allow them to install skimmer technology or the ability to steal lists of customers.

Further Reading:

- Code Injection

Local and remote file inclusions

This happens when an attacker is able to get a local (on the file system) or a remote file executed in the production environment. Again, this allows them to access almost anything they want.

One way this can happen is if a code or class path is stored in the database. For example, most rules (catalog price rule, cart price rule, customer segments, target rules) store class paths in the database.

What helps to mitigate this is that Magento validates classes once they have been instantiated to ensure they implement the correct values. See `\Magento\CustomerSegment\Model\ConditionFactory::create`.

Further Reading:

- Testing for Local File Inclusion

Session hijacking

This happens when an attacker gets the session ID from the `frontend`, `adminhtml`, or `PHPSESSID` cookie.

This is mitigated by enabling session validation options in Store Configuration > General > Web. However, this can lead to dropped sessions (i.e. customers losing cart contents).

Further Reading:

- Session Hijacking

SQL injection

This happens when an attacker finds an input that is not sanitized before an operation happens on the database. Good developers escape input and use parameters wherever possible.

```
// BAD:
$this->getConnection()->query("SELECT * FROM catalog_product_
entity WHERE entity_id = " . $productId);
// GOOD
$this->select()
    ->from('catalog_product_entity')
    ->where('entity_id = ?', (int)$value);
```

Not only is the latter easier to read, it is protected against SQL injections. Unfortunately, I have seen modules with the former code.

Further Reading:

- SQL Injection

Insecure functions

I strongly recommend to NEVER write insecure functions. All kidding aside, being a conscientious developer is the best antidote.

The context here would be not using dangerous PHP functions. There is little to no reason to ever use `eval`. Be careful when using system commands like `exec` as this can give unfettered access to the production environment. If possible, use the Magento wrappers for any file system commands as they not only have error handling built-in but there is also some security enhancements.

Further Reading:

- Dangerous PHP Functions

Directory traversal attacks

If there is a way to specify a file path, like `../var/import/file.csv`, a malicious attacker could specify `../../../etc/passwd`.

Further Reading:

- Path Traversal

PHP functions that should never be used, per Magento recommendations (eval, serialize, etc.)

Magento recommends against using:

- `eval`: I can't think of a reason why you would need to do this.
- `serialize` and `unserialize`: utilize `json_encode` and `json_decode` and inject data into the appropriate model instead of letting PHP automatically unserialize into a class.
- `md5`: use `hash`
- `srand`: use `\Magento\Framework\Math\Random::getRandomString`
- `mt_srand`: use `\Magento\Framework\Math\Random::getRandomString`
- The `ArrayObject` class.

Further Reading:

- Writing Secure Code

How to prevent directory traversals attacks when uploading a file

Utilize the `\Magento\Framework\App\Filesystem\DirectoryResolver::validatePath` method. This ensures that the specified path matches the root directory to Magento.

Importance of the HttpOnly property when setting a cookie

`HttpOnly` cookies are cookies that Javascript cannot read in `document.cookie`. If malicious code is installed onto your site, this ensures they cannot read the session cookie, thus allowing unwanted access to the website.

Further Reading:

- HttpOnly
- Secure and HttpOnly Cookies

Serving Magento from the base vs. the pub/ directory

It's amazing that we keep coming across sites that are served from the base directory.

Why serve from `pub/`? This limits exposure to possibly sensitive information should the server be misconfigured. I have come across a number of websites where their `var/` directory is publicly accessible. One of those websites had multiple human-readable SQL data dumps present—not good.

Serving from the `pub/` directory prevents this from even happening.

Retrieving the user IP from the REMOTE_ADDR header vs. the X-Forwarded-For header

You can normally fetch the user's IP address using the `Magento\Framework\HTTP\PhpEnvironment\RemoteAddress::getRemoteAddress()`.

However, in cases of load balancing or a web application firewall (like CloudFlare), the IP address can be CloudFlare's IP address, which is quite useless. Proxies utilize the `X-Forwarded-For` header to provide the original IP address.

Further Reading:

- How to add alternative Http headers to Magento 2
- How does Cloudflare handle HTTP Request headers?

File operations influenced by user-submitted request parameters, how to stop directory traversal attacks and restrict access to dirs/files

This is discussed above. Ideally, you want to limit any input that a user (frontend or backend) has directly on the file system. But, if it is a must that they have access, carefully ensure that the path is properly validated.

Further Reading:

- Secure cron.php to run in a browser
- X-Frame-Options header

How to write secure SQL queries

Ensure that you are using parameters. Use type casting wherever the value should be an integer or a float. Don't write direct SQL queries.

APPENDIX

Professional Developer Plus

SWIFT OTTER SOLUTIONS

DISCUSSION OF THE ENTITY MANAGER

This isn't particularly relevant to the overall discussion of staging (where this is mentioned). Entity Manager is the framework that staging is bolted on to. As such, it warrants a mention, but not in the flow of the book.

Magento recommends against using an entity manager for any new work: `vendor/magento/framework/EntityManager/EntityManager.php`. As such, we will not discuss this too deeply other than to provide pointers as to where you can learn more.

The idea behind EntityManager is that you use configuration (in `di.xml`) to assemble most of the functionality that is needed—and that this would simplify the process of storing and managing data. Ultimately, EntityManager proved too complex as the simple abstract resource model does the job quite nicely.

The `Magento\Framework\EntityManager\MetadataPool` stores the schema information associated with updating the database. Here is a snippet from `vendor/magento/module-catalog/etc/di.xml`:

```
<type name="Magento\Framework\EntityManager\MetadataPool">
  <arguments>
    <argument name="metadata" xsi:type="array">
      <item name="Magento\Catalog\Api\Data\ProductInterface" xsi:type="array">
        <item name="entityTableName" xsi:type="string">catalog_product_entity</item>
        <item name="eavEntityType" xsi:type="string">catalog_product</item>
        <item name="identifierField" xsi:type="string">entity_id</item>
        <item name="entityContext" xsi:type="array">
          <item name="store" xsi:type="string">Magento\Store\Model\StoreScopeProvider</item>
        </item>
      </item>
    </argument>
  </arguments>
</type>
```

Note that there are a number of events that are available for the EntityManager.

Load/Create/Delete/Update:

- `entity_manager_[action]_before`
- `$entityType_[action]_before` (for the entity type of `Magento\Cms\Api\Data\PageInterface` the result will be `magento_cms_api_data_pageinterface_load_before`)
- `$entityType_[action]_after`
- `entity_manager_[action]_after`

Attributes and Extensions (EntityManager):

The only examples I find are extensions: `vendor/magento/module-sales-rule/etc/di.xml`

Loading:

First, the metadata is loaded. This object contains information about the entity to be saved, including references to additional tables (for example in the CMS page entity, `sequence_cms_page`). Second, the hydrator is loaded. This takes raw data that has been loaded from the database and returns a hydrated entity model. The data is loaded from the database, then hydrated (`\Magento\Framework\EntityManager\Operation\Read\ReadMain::execute`). All linked fields are then loaded, and the data is merged. This happens for both extensions (`\Magento\Framework\EntityManager\Operation\Read\ReadExtensions`) and attributes (`\Magento\Framework\EntityManager\Operation\Read\ReadAttributes`).

Saving:

Similar to loading, the same classes are loaded. The before events are run, then the data is saved: first, the row data, next the attribute data, finally, the extension data.

Practical experience:

- Navigate to a CMS page in adminhtml.
- Set a breakpoint in `\Magento\Framework\EntityManager\Operation\Read::execute`.
- Refresh.

Further Reading:

- Entity Manager
- Recipe for a good Data Model on Magento 2