1-Magento Architecture and Customization Techniques

1.1 Describe Magento’s module-based architecture

- registration.php
- composer.json autoload/files[] = “registration.php”
- at what stage - when including vendor/autoload.php
- how registered when not in composer - project composer.json autoload/files[] = app/etc/NonComposerComponentRegistration.php

  app/code//cli_commands.php, registration.php app/design//*/registration.php app/i18n//registration.php lib/internal//registration.php lib/internal///registration.php

pub/index.php app/bootstrap.php app/autoload.php vendor/autoload.php vendor/module[]/registration.php – last step in Composer init Magento\Framework\Component\ComponentRegistrar::register(type=‘module’, name=‘Prince_Productattach’, path=’/var/www/…’)

How do different modules interact with each other?

- composer require
- module.xml sequence - hard requirement, affects load order
- DI require instance

What side effects can come from this interaction?

- error when module is missing or disabled
- error when injecting missing class
- (?) null or error when using object manager for missing class ReflectionException - Class MissingClass does not exist objectManager->create() = new $type() or new $type(…args) --> PHP Warning: Uncaught Error: Class ‘MissingClass’ not found

1.2 Describe Magento’s directory structure

- app/code/Vendor/Module
- app/design/frontend/VendorName/theme_name_lowercase
- app/i18n/VendorName/en_US
- vendor/vendor-name/module-theme-package/

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

- view/frontend/web/js
- view/frontend/requirejs-config.js
- view/frontend/layout
- view/frontend/templates

How do you find the files responsible for certain functionality?

1.3 Utilize configuration XML and variables scope

Determine how to use configuration files in Magento. Which configuration files correspond to different features and functionality?

- acl.xml - resource title, sort
- adminhtml/rules/payment_{country}.xml - paypal
- address_formats.xml
- address_types.xml - format code and title only
- cache.xml - name, instance - e.g. full_page=Page Cache
- catalog_attributes.xml - catalog_category, catalog_product, unassignable, used_in_autogeneration, quote_item
- communication.xml
- config.xml - defaults
- crontab.xml - group[], job instance, method, schedule
- cron_groups.xml
- di.xml - preference, plugins, virtual type
- eav_attributes.xml - locked entity attributes (global, unique etc.)
- email_templates.xml - id label file type module – view/frontend/email/name.html
- events.xml - observers, shared, disabled
- export.xml
- extension_attributes.xml - for, attribute code, attribute type
- fieldset.xml
- import.xml
- indexer.xml - class, view_id, title, description
- integration.xml - API resources + pre-configure the integration (After Magento 2.2)
- integration/api.xml - file defines which API resources the integration has access to(Deprecated in Magento 2.2 and all config are moved in integration.xml)
- integration/config.xml - pre-configure the integration, the values cannot be edited from the admin panel ((Deprecated in Magento 2.2 and all config are moved in integration.xml)
- menu.xml
- module.xml - version, sequence
- mview.xml - scheduled updates, subscribe to table changes, indexer model
- page_types.xml
- payment.xml - groups, method allow_multiple_address
- pdf.xml - renders by type (invoice, shipment, creditmemo) and product type
- product_types.xml - label, model instance, index priority, (?) custom attributes, (!) composable types
- product_options.xml
- resources.xml
- routes.xml - Routes, Frontend name, module and extending routes
- sales.xml - collectors (quote, order, invoice, credit memo)
- search_engine.xml - Add custom search engine
- search_request.xml - index, dimensions, queries, filters, aggregations, buckets
- sections.xml - action route placeholder -> invalidate customer sections
- system.xml - adminhtml config
- validation.xml - entity, rules, constraints -> class
- view.xml - vars by module
- webapi.xml - route, method, service class and method, resources
- widget.xml - class, email compatible, image, ttl (?), label, description, parameters
- zip_codes.xml

\Magento\Framework\Config\Reader\Filesystem, \Magento\Framework\Config\ReaderInterface

Gets .xsd names from schema locator, gets full .xml file list from file resolver, merges all files, validates, runs converter to get resulting array.

- read(scope)
  - fileResolver->get(_filename)
  - merge and validate each file (if mode developer)
  - validate merged dom
  - converter->convert(dom) => array
- _idAttributes, _fileName, _schemaFile (from schemaLocator), _perFileSchema (from schemaLocator), filename (menu.xml)
- schemaFile from schemaLocator

\Magento\Catalog\Model\ProductTypes\Config\Converter, \Magento\Framework\Config\ConverterInterface - array in any format

- convert(\DOMDocument $source)

\Magento\Framework\Config\SchemaLocatorInterface - full path to .xsd

- getPerFileSchema - per file before merge
- getSchema - merged file

config merger = \Magento\Framework\Config\Dom

- when merging each file
  - createConfigMerger|merge
  - \Magento\Framework\Config\Dom::_initDom(dom, perFileSchema)
  - \Magento\Framework\Config\Dom::validateDomDocument
  - $dom->schemaValidate
- after all merged
  - \Magento\Framework\Config\Dom::validate(mergedSchema)
  - \Magento\Framework\Config\Dom::validateDomDocument

Sensitive and environment settings

- shared config app/etc/config.php
- sensitive or system-specific app/etc/env.php:

```
    <type name="Magento\Config\Model\Config\TypePool">
       <arguments>
          <!-- sensitive config items -->
          <argument name="sensitive" xsi:type="array">
             <item name="payment/paypal_express/merchant_id"
xsi:type="string">1</item>
             <!-- keys, password, emails, personally identifiable information -->
          </argument>
          <!-- environment specific config items -->
          <argument name="environment" xsi:type="array">
             <item name="payment/paypal_express/debug" xsi:type="string">1</item>
             <!-- URLs, IPs, hosts, modes sandbox/live, email recipients -->
          </argument>
       </arguments>
    </type>
```

sensitive info doesn't get exported with `bin/magento app:config:dump`. use env. params, e.g. `CONFIG__DEFAULT__PAYMENT__TEST__PASWORD` for `payment/test/password`

`bin/magento app:config:dump` :

- system-specific > app/etc/env.php
- shared > app/etc/config.php
- sensitive - skipped

`bin/magento config:sensitive:set`:

- writes to app/etc/env.php

1.4 Demonstrate how to use dependency injection

```
    <argument xsi:type="object">{typeName}</argument>
    <argument xsi:type="object" shared="{shared}">{typeName}
</argument>

    <argument xsi:type="string">{strValue}</argument>
    <argument xsi:type="string" translate="true">{strValue}</argument>
```

- Initial (app/etc/di.xml)
- Global (/etc/di.xml)
- Area-specific (/etc//di.xml)

1.5 Demonstrate ability to use plugins

- can’t use before Magento\Framework\Interception… is initialized
- *after* plugin has access to *arguments*! @since 2.2

```
    class MyBeautifulClass
    {
        use \Magento\Framework\Interception\Interceptor;

        public function __construct($specificArguments1, $someArg2 = null)
        {
            $this->___init();
            parent::__construct($specificArguments1, $someArg2);
        }

        public function sayHello()
        {
            pluginInfo = pluginList->getNext('MyBeautifulClass',
'sayHello')
            ___callPlugins('sayHello', [args], pluginInfo)
        }
    }
```

Magento\Framework\Interception\Interceptor:

- $pluginList = \Magento\Framework\Interception\PluginListInterface
- $subjectType = ‘MyBeautifulClass’
- `___init` - called in in constructor, pluginList = get from object manager, subjectType = class name
- pluginList->getNext
- `___callPlugins`
- `___callParent`

how generated?

```
\Magento\Framework\App\Bootstrap::create
\Magento\Framework\App\Bootstrap::__construct
\Magento\Framework\App\ObjectManagerFactory::create
\Magento\Framework\ObjectManager\DefinitionFactory::createClassDefinition
    \Magento\Framework\ObjectManager\DefinitionFactory::getCodeGenerator
    \Magento\Framework\Code\Generator\Io::__construct
    \Magento\Framework\Code\Generator::__construct
    spl_autoload_register([new \Magento\Framework\Code\Generator\Autoloader,
'load']);

\Magento\Framework\App\ObjectManagerFactory::create
\Magento\Framework\Code\Generator::setGeneratedEntities
\Magento\Framework\App\ObjectManager\Environment\Developer::configureObjectManager

\Magento\Framework\Code\Generator\Autoloader::load
\Magento\Framework\Code\Generator::generateClass
```

Decide how to generate based on file suffix - generator \Magento\Framework\Code\Generator\EntityAbstract

```
array (
  'extensionInterfaceFactory' =>
'\\Magento\\Framework\\Api\\Code\\Generator\\ExtensionAttributesInterfaceFactoryGenerator',
  'factory' => '\\Magento\\Framework\\ObjectManager\\Code\\Generator\\Factory',
  'proxy' => '\\Magento\\Framework\\ObjectManager\\Code\\Generator\\Proxy',
  'interceptor' => '\\Magento\\Framework\\Interception\\Code\\Generator\\Interceptor',
  'logger' => '\\Magento\\Framework\\ObjectManager\\Profiler\\Code\\Generator\\Logger',
    - logs all public methods call
    - Magento\Framework\ObjectManager\Factory\Log -- missing?
  'mapper' => '\\Magento\\Framework\\Api\\Code\\Generator\\Mapper',
    - extractDto() = $this->{$name}Builder->populateWithArray()->create
  'persistor' => '\\Magento\\Framework\\ObjectManager\\Code\\Generator\\Persistor',
    - getConnection, loadEntity, registerDelete, registerNew, registerFromArray, doPersist,
doPersistEntity
  'repository' => '\\Magento\\Framework\\ObjectManager\\Code\\Generator\\Repository', --
deprecated
  'convertor' => '\\Magento\\Framework\\ObjectManager\\Code\\Generator\\Converter',
    - Extract data object from model
    - getModel(AbstractExtensibleObject $dataObject) = getProductFactory()->create()-
>setData($dataObject)->__toArray()
  'searchResults' => '\\Magento\\Framework\\Api\\Code\\Generator\\SearchResults',
    - extends \Magento\Framework\Api\SearchResults
  'extensionInterface' =>
'\\Magento\\Framework\\Api\\Code\\Generator\\ExtensionAttributesInterfaceGenerator',
  'extension' =>
'\\Magento\\Framework\\Api\\Code\\Generator\\ExtensionAttributesGenerator',
    - extension_attributes.xml
    - extends \Magento\Framework\Api\AbstractSimpleObject
    - implements {name}\ExtensionInterface
    - for every custom attribute, getters and setters
  'remote' =>
'\\Magento\\Framework\\MessageQueue\\Code\\Generator\\RemoteServiceGenerator',
)
```

```
Magento\Framework\App\ResourceConnection\Proxy -> type Proxy, name
Magento\Framework\App\ResourceConnection
Magento\Framework\Code\Generator::shouldSkipGeneration - type not detected, or
class exists
\Magento\Framework\Code\Generator::createGeneratorInstance -- new for every file
```

\Magento\Framework\Code\Generator\EntityAbstract - code generation

```
\Magento\Framework\Code\Generator\EntityAbstract::generate
\Magento\Framework\Code\Generator\EntityAbstract::_validateData - class not
existing etc.
\Magento\Framework\Code\Generator\EntityAbstract::_generateCode
  - \Magento\Framework\Code\Generator\ClassGenerator extends
\Zend\Code\Generator\ClassGenerator
-
\Magento\Framework\Code\Generator\EntityAbstract::_getDefaultConstructorDefinition
- \Magento\Framework\Code\Generator\EntityAbstract::_getClassProperties
- \Magento\Framework\Code\Generator\EntityAbstract::_getClassMethods
```

1.6 Configure event observers and scheduled jobs

Demonstrate how to configure observers

- can define observer in global area, then disable in specific area

Observer sortOrder:

- before sortOrder=10, before sortOrder=20, before sortOrder=30 …
- before and around (first half) called together for same plugin!
- around (second half) and after called together for same plugin!

Example:

- pluginA.beforeMethod, pluginA.aroundMethod first half
- pluginB.beforeMethod, pluginB.aroundMethod first half
- pluginC.beforeMethod, `_____
- _____, pluginD.aroundMethod first half
- method()
- _____, pluginD.aroundMethod second half
- pluginC.afterMethod , _____
- pluginB.aroundMethod second half, pluginB.afterMethod
- pluginA.aroundMethod second half, pluginA.afterMethod

Demonstrate how to configure a scheduled job

cron_groups.xml - store view scope:

- default (no separate process)
- index - mview, targetrule
- catalog_event - catalog_event_status_checker - mark event open/closed
- consumers - consumers_runner if configured to run by cron. `bin/magento queue:consumers:start`. PID file var/{$consumer}.pid
- staging - staging_apply_version, staging_remove_updates, staging_synchronize_entities_period
- ddg_automation (dotmailer)

```
    <group id="NAME">
        <job name="NAME" instance="CLASS" method="METHOD">
            <config_path>some/config/path</config_path>
        </job>
        <job name="NAME" instance="CLASS" method="METHOD">
            <schedule>* * * * *</config_path>
        </job>
    </group>
```

run:

- magento cron:run [–group=””]
- pub/cron.php?[group=] in a web browser, protect with basic auth

```
\Magento\Cron\Console\Command\CronCommand::execute
\Magento\Framework\App\Cron::launch
`default` event
\Magento\Cron\Observer\ProcessCronQueueObserver
check for specific group
cleanup
generate
check for standalone process
```

`\Magento\Cron\Model\Config\Data` extends `\Magento\Framework\Config\Data`

- merges \Magento\Cron\Model\Config\Reader\Db::get from Database

  Sample DB structure:

```
default/crontab/GROUP/jobs/JOB/schedule/cron_expr = '* * * * *'
default/crontab/GROUP/jobs/JOB/schedule/config_path = 'some/config/path' -- try to
read schedule from this config, store view scope
default/crontab/GROUP/jobs/JOB/run/model = 'class::method'
```

`bin/magento cron:install` example:

```
#~ MAGENTO START 4d557a63fe1eac8a2827a4eca020c6bb
* * * * * /usr/bin/php7.0 /var/www/m22ee/bin/magento cron:run 2>&1 | grep -v "Ran
jobs by schedule" >> /var/www/m22ee/var/log/magento.cron.log
* * * * * /usr/bin/php7.0 /var/www/m22ee/update/cron.php >>
/var/www/m22ee/var/log/update.cron.log
* * * * * /usr/bin/php7.0 /var/www/m22ee/bin/magento setup:cron:run >>
/var/www/m22ee/var/log/setup.cron.log
#~ MAGENTO END 4d557a63fe1eac8a2827a4eca020c6bb
```

how is separate process ran?
`bin/magento cron:run --group=NAME --bootstrap=standaloneProcessStarted=1`

what is update/cron.php? TODO: find out

what is setup:cron:run? TODO: find out

Identify the function and proper use of automatically available events

Model events \Magento\Framework\Model\AbstractModel:

- `model_load_before`, `{$_eventPrefix}_load_before`
- `model_load_after`, `{$_eventPrefix}_load_after`
- `model_save_commit_after`, `{$_eventPrefix}_save_commit_after`
- `model_save_before`, `{$_eventPrefix}_save_before`
- `model_save_after`, `{$_eventPrefix}_save_after`
- `model_delete_before`, `{$_eventPrefix}_delete_before`
- `model_delete_after`, `{$_eventPrefix}_delete_after`
- `model_delete_commit_after`, `{$_eventPrefix}_delete_commit_after`

Flat collection events \Magento\Framework\Model\ResourceModel\Db\Collection\AbstractCollection:

- `core_collection_abstract_load_before`, `{_eventPrefix}_load_before`
- `core_collection_abstract_load_after`, `{_eventPrefix}_load_after`

only if `_eventPrefix` and `_eventObject` defined: `{prefix}_load_before`, `{prefix}_load_after`

EAV collection events \Magento\Eav\Model\Entity\Collection\AbstractCollection:

- eav_collection_abstract_load_before

\Magento\Framework\Model\AbstractModel:

- _eventObject = 'object'
- _eventPrefix = 'core_abstract', e.g. 'catalog_category'
- _getEventData() - 'data_object' + $_eventObject
- `model_load_before` (object, field=null, value=ID)
- `{_eventPrefix}_load_before`, e.g. `catalog_category_load_before` (object, field, value, data_object, category)

1.7 Utilize the CLI

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

Demonstrate an ability to create a deployment process.

Modes: *default*, *developer*, *production*. MAGE-MODE env variable

Commands:

- `bin/magento deploy:mode:show`
- `magento deploy:mode:set {mode} [-s|--skip-compilation]` – skip compilation when changing to production

cannot switch to default mode, only developer or production

Default:

- errors logged in var/report, not displayed
- static created dynamically - copied! changes not visible. cached

Developer:

- exceptions displayed, not logged
- exception thrown if bad event subscriber.
- var/report detailed
- static created dynamically - symlinked???, changes visible immediately
- error handler - throws exception instead of logging (notice etc.)

Production - max speed, no errors, no file generation:

- admin can't enable/disable cache types
- errors logged, not displayed
- static not created dynamically, must be deployed
- not need for www-data to write, pub/static can be read-only

1.8 Demonstrate the ability to manage the cache

Describe cache types and the tools used to manage caches.

- config
- layout
- block_html
- collections
- db_ddl
- eav
- full_page
- reflection
- translate
- config_integration
- config_integration_api
- config_webservice

Commands:

- `magento setup:db-schema:upgrade`
- `magento cache:status`, `magento cache:enable`, `magento cache:disable`
- `magento cache:clean`, `magento cache:flush`

Init:

frontend:

```
\Magento\Framework\App\ObjectManager\ConfigLoader::load
cacheType = config, frontend = default
\Magento\Framework\App\Cache\Frontend\Pool::_initialize
\Magento\Framework\App\Cache\Frontend\Factory::create
\Zend_Cache::_makeFrontend
\Zend_Cache_Core::__construct
```

backend:

- \Zend_Cache_Backend
- \Zend_Cache_Backend_File
- \Magento\Framework\Cache\Backend\Database

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

1. Mark any block `cacheable="false"` in layout xml - whole page is uncacheable. Example - checkout
2. Disable caching in controller using *headers*: `$page->setHeader('Cache-Control', 'no-store, no-cache, must-revalidate, max-age=0', true);`
3. Marking block property isScopePrivate - will be loaded via AJAX - deprecated
4. ESI when Varnish enabled, set TTL - Example - menu block
5. Configure page variations - extend *http context*, more cached versions of same page - store view, customer group, language, currency, is logged in `\Magento\Framework\App\Http\Context::getVaryString`

Only *GET* and *HEAD* are cached

Clear cache \Magento\Framework\DataObject\IdentityInterface

when giving product page, must somehow send varnish tags

Any block can implement IdentityInterface. After rendering layout and before sending output, all

blocks are examined for implementing this interface. Cache tags are collected as merge of all blocks getIdentities() tags.

```
\Magento\PageCache\Model\Layout\LayoutPlugin::afterGetOutput
  X-Magento-Tags = merge(all blocks.getIdentities)
```

block ListProduct:

- every product[].getIdentities
  - cat_p_{productId}
  - *if changed categories* - cat_p_c_{categoryId}
  - *if changed status* - every category[] cat_p_c_{categoryId}
  - *if frontend* - 'cat_p'
- cat_c_p_{categoryId}

block product/view:

- \Magento\Catalog\Model\Product::getIdentities:
  - cat_p_{productId}
  - *if changed categories* - cat_p_c_{categoryId}
  - *if changed status* - every category[] cat_p_c_{categoryId}
  - *if frontend* - 'cat_p'
- *if current_category* - cat_c_{categoryId}

after reindex, must somehow clean cache

- any indexer.execute – by MView
- any indexer.executeFull
- \Magento\Framework\Indexer\CacheContext::registerTags

plugin \Magento\Indexer\Model\Processor:

\Magento\Indexer\Model\Processor\CleanCache::afterUpdateMview

- event `clean_cache_after_reindex`
- clean cache cacheContext->getIdentities()

\Magento\Indexer\Model\Processor\CleanCache::afterReindexAllInvalid

- event `clean_cache_by_tags`
- clean cache cacheContext->getIdentities()

module-cache-invalidate observer `clean_cache_after_reindex`
\Magento\CacheInvalidate\Observer\InvalidateVarnishObserver::execute
\Magento\CacheInvalidate\Model\PurgeCache::sendPurgeRequest

Describe how to operate with cache clearing.

How would you clean the cache? In which case would you refresh cache/flash cache storage?

Describe how to clear the cache programmatically.

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