BELVG
FOR SUCCESSFUL
E-COMMERCE

AWARD WINNING
ECOMMERCE AGENCY

Magento 2 Certified Professional **Developer Guide**

Section 1: Magento Architecture and Customization Techniques

1.1 Describe Magento's module-based architecture

Magento 2 modules are realized with MVVM architecture:

https://upload.wikimedia.org/wikipedia/commons/8/87/MVVMPattern.png

MVVM has three layers:

1. Model
The Model contains the application's business logic and depends on an associated class—the ResourceModel - for database access. Models depend on service contracts to disclose their functionality to other application layers.
2. View
The View is both structure and layout of what is seen on a screen - the actual HTML. This is achieved in the PHTML files distributed with modules. Such files are associated with each ViewModel in the Layout XML files, sometimes referred to as binders. The layout files can as well assign JavaScript files to be used on the final page.
3. ViewModel
The ViewModel works together with the Model layer and exposes only necessary information to the View layer. In Magento 2, this is handled by the module's Block classes. Note that this was usually part of the Controller role of an MVC system. On MVVM, the controller is only responsible for handling the user flow, meaning that it receives requests and either tells the system to render a view or to redirect the user to another route.

http://devdocs.magento.com/common/images/archi_diagrams_layers_alt4.jpg
Magento 2 architecture consists of 4 layers:

1. Presentation Layer

Presentation Layer is the top layer that contains view elements (layouts, blocks, templates) and controllers.
Presentation Layer usually calls the service layer using service contracts. But, depending on the implementation, it may cross over with the business logic.

2. Service layer
Service layer is the layer between presentation and domain layers. It executes service contracts, which are implemented as PHP interfaces. Service contracts allow to add or change the business logic resource model using the dependency injection file (di.xml). The service layer is also used to provide API access (REST / SOAP or other modules).
The service interface is declared in the / API namespace of the module.
Data (entity) interface is declared inside / Api / Data. Data entities are the structures of data passed to and returned from service interfaces.

3. Domain layer
The domain layer is responsible for business logic that does not contain information about resources and the database. Also, the domain layer may include the implementation of service contracts. Each data model at the domain layer level depends on the resource model, which is responsible for the database access.

4. Persistence layer
The persistence layer describes a resource model that is responsible for retrieving and modifying data in a database using CRUD requests.
It also implements additional features of business logic, such as data validation and the implementation of database functions.

Describe module limitations

1. Certain modules can come in conflict with each other if the dependencies in module.xml (sequence) are specified incorrectly.
2. Not all classes can be overridden with modules.

How do different modules interact with each other?

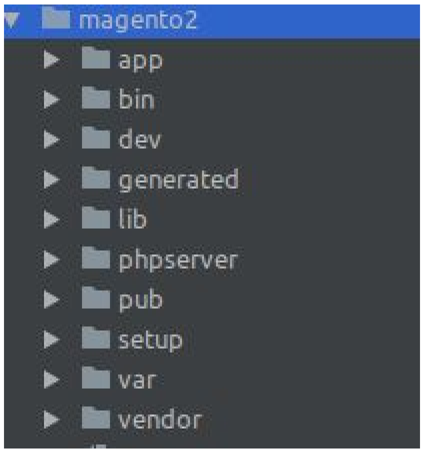
Different modules interact with each other via dependency injection and service contracts. They can also be dependent on other modules when they apply other modules' logic.

What side effects can come from this interaction?

If modules contain di.xml files, such files may sometimes load in the wrong order or try to override the already overridden module functionality. To resolve this situation, use sequence in module.xml.
When modules use other modules' logic and this very module is deleted, the dependencies would be unable to load and an error at the code execution will occur.

1.2 Describe Magento's directory structure

Determine how to locate different types of files in Magento. Where are the files containing JavaScript, HTML, and PHP located?



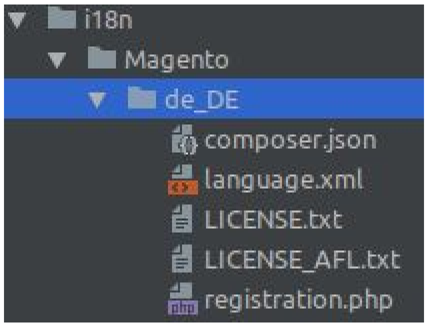
The whole Magento structure can be divided into the following types:

1. Magento root structure
2. Modules structure
3. Themes structure

To begin with, Magento has various areas that allow to determine configuration, view files, etc. for a certain area. Adminhtml (applied to the administration panel) and frontend (applied to frontend parts of the website) are examples of area. From this point, we will use <area> to denote any of the available areas.

Magento root partition structure:

1) app - the directory recommended for Magento components development. It consists of:
    a) design - contains themes
    b) code - contains modules
    c) etc - contains Magento framework settings
    d) i18n - contains the language package. Example:



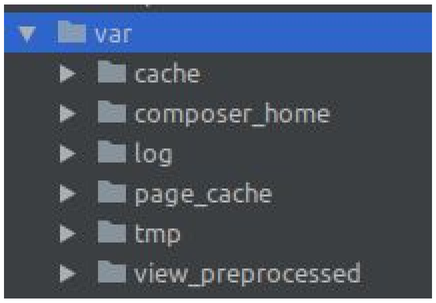
```
▼ i18n
  ▼ Magento
    ▼ de_DE
        composer.json
        language.xml
        LICENSE.txt
        LICENSE_AFL.txt
        registration.php
```

2) bin - there the executed Magento file is located that allows to manage the website via CLI.
3) dev - the directory for Magento test scripts (for more information, follow the link https://devdocs.magento.com/guides/v2.3/mtf/mtf_quickstart.html).
4) generated - contains the generated php files (Factories, Proxies, Interceptors, DI configuration).
5) lib - used for Magento library files.
6) phpserver - contains the router file "router.php" for the php Built-in web server. Allows to use Magento without a third-party web server, like nginx and apache. Here is the example of php Built-in web server launch:
   php -S 127.0.0.1:8082 -t ./pub/ ./phpserver/router.php
7) pub - the directory used to access Magento static files. It contains the following directories:
    a) errors - stores the error pages,
    b) media - stores all media files (product, pages, etc.),
    c) static - stores Magento themes generated files.

This directory can be specified as web root in nginx config or in apache config. Numerous Magento directories contain ".htaccess" files (including root and pub), which allow you to configure apache for a specific directory. Nginx does not support

.htaccess. For nginx, Magento has a nginx.conf.sample file, which is an example of Magento configuration for nginx. This file can be copied, modified, and include the main nginx configuration file.

8) setup - contains Setup Wizard
9) update - contains Magento Updater
10) var - contains all the temporary files. Consists of:



  a) cache - contains cache files if cache utilizes file system as a storage
  b) page_cache - contains FPC (Full Page Cache) files, if FPC utilizes file system as a storage
  c) log - contains Magento logs
  d) report - contains Magento error/exception files that were not intercepted by code
  e) session - contains session files
  f) view_preprocessed - contains style generated files and minified HTML

11) vendor - contains the installed composer packages. When Magento is installed via composer or using official site's archive, then all the standard Magento modules and Magento Framework are located in this folder. In case you install via the official GIT repository, which is recommended only for contributing, then Magento Framework will be located in lib/internal folder, while the default modules - in app/code folder.

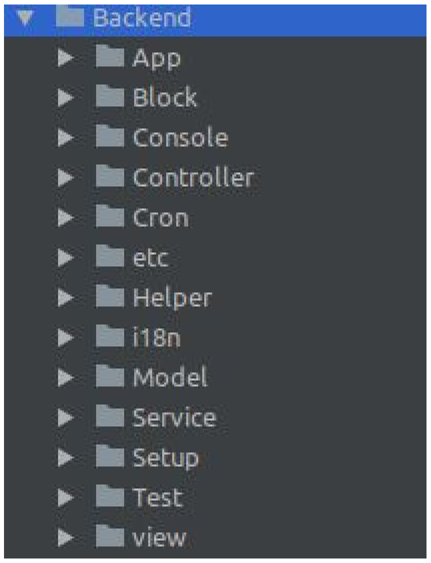
Next, we will proceed with module structure.
Modules can be located in /app/code and /vendor directories.

The modules in /app/code can be found at a similar path - **/app/code/BelVG/MyModule**, where BelVG is vendor name, MyModule - module name (in capitals). PSR-0 standard is used for php classes loading in /app/code directories (https://www.php-fig.org/psr/psr-0/). Example: on request, \BelVG\MyModule\Model\MyModel class will be automatically loaded with app/code/BelVG/MyModule/Model/MyModel.php file.

Modules in /vendor are located at the similar **/vendor/belvg/module-my-module** path, where belvg/module-my-module is the composer package name. Package name contains the names of a vendor and a project. In theory, there are no strict requirements to the package name because it does not impact the module operation. But, for your own comfort, it is better to specify package name as "{vendor-name}/module-{module-name}". Package name should be in lowercase, with words separated by dashes. PSR-4 standard is applied in /vendor directory for loading php classes (https://www.php-fig.org/psr/psr-4/).
Example: when requested, \BelVG\MyModule\Model\MyModel class is automatically loaded from the file inside the /vendor/belvg/module-my-module/Model/MyModel.php composer package.

Hereinafter **<module_dir>** will be used to specify module root directory, for this directory can be located both in app/code and in vendor.

It is recommended to install the third-party modules via composer.



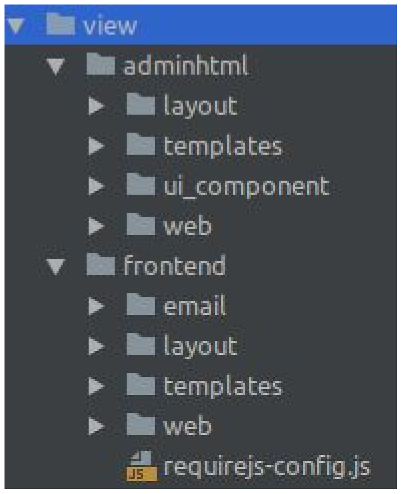
Inside the module directory, the following directories and files are located:

a) etc - required directory where the module configuration is stored.
b) Block - the directory with php class blocks.
c) Model - the directory with models and resource models
d) Controller - contains all actions for the module. Controller\<Controller>\<Action> is the php classes template for all module actions. For instance, for

<front_name>/<controller>/<action>/ query the Controller/<Controller>/<Action>.php file will be requested.

e) Helper - contains helper classes
f) Console - contains php classes for CLI calls via /bin/magento.
g) Api - contains API interfaces.
h) Observer - contains php files of observer classes.
i) i18n - localization files directory
j) Setup - php classes directory that executes certain actions during module installation, upgrade, refresh or deletion. From Magento 2.3 version and further, the directory also contains data and schema patches.
k) Plugins - the directory, containing plugins for classes and interfaces.
l) Ui - contains auxiliary php classes for UI components, like modifies, data providers, etc.
m) view - contains templates, CSS, JS, images, layouts, UI components files. The internal structure of this directory consists of <area> and base, with base storing the files that relate to all areas. <area> and base have the similar file structures.

To make an example, we will give examples of two areas' contents :

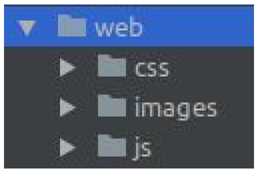


```
view
  adminhtml
    layout
    templates
    ui_component
    web
  frontend
    email
    layout
    templates
    web
    requirejs-config.js
```

This is the internal structure of the directories:

- email - contains email templates. For example, order_new.html
- layout - contains layouts xml files. Commonly, the layout name is formed according to this template: {route_id}_{controller}_{action}.xml
- templates - contains phtml templates' files

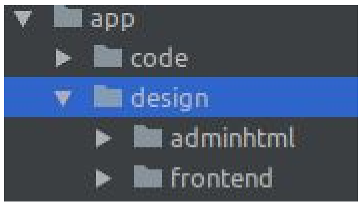
- page_layout - contains files, describing Magento page types (1column.xml, 2columns-left.xml, etc,)
- ui_component - contains .xml files with UI components description
- web - contains static files, template files for KnockoutJS, JS, theme files and images. Below is web directory example:

  css - contains css styles
  images - contains images
  js - contains JavaScript files
  template - contains .html templates for KnockoutJS

n) registration.php - the required file that registers the module in Magento.
o) composer.json - contains composer package configuration.

Next we will describe the design structure.
Magento 2 templates are located in app/design/ and vendor folders. Each Magento 2 theme is connected to a certain area.
Themes for admin panel are located at app/design/adminhtml folder, while app/design/frontend contains themes for frontend.



Themes in app/design are located at <area>/<ThemeVendor>/<theme_name> folders, where <ThemeVendor> is written in capital letter, <theme_name> in lowercase. For example: app/design/frontend/BelVG/my_theme.

Themes in /vendor are located at the following path **/vendor/belvg/theme-frontend-my-theme**, where belvg/theme-frontend-my-theme is composer package name. Similar to modules, there are no requirements to composer package name, but for simplicity, it is better so name them as "<vendor-name>/theme-<area>-<theme-name>".

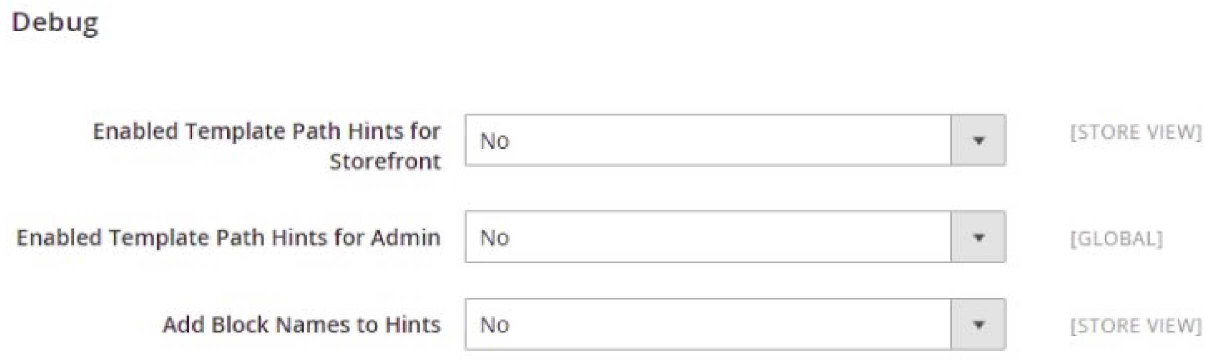
Further in the text, we will use **<theme_dir>** to define the theme root directory, for this directory can be located both in app/design and in vendor.

The following directories and files are located in the directory together with the theme:

a) <ModuleVendor>_<ModuleName> - common template for override view module files. For example, BelVG_CustomModule.
   Inside the directories and files, the structure is similar to view/<area>/ folder inside the module (for example, app/code/BelVG/CustomModule/view/frontend).
b) etc - contains theme configuration files.
c) media - contains theme media files.
d) web - contains theme css/js/fonts files.
e) Theme.xml file - a mandatory file with the theme configurations, like the name of the theme and its parent.
f) requirejs-config.js - contains RequireJS config.
g) registration.php - a mandatory file that registers a theme in Magento.
h) composer.json - contains composer package configuration.

How do you find the files responsible for certain functionality?

To find the files responsible for certain functionality, search among di.xml files (Dependency Injection configuration). Another way is to search in a module or theme layout files, if you need to find certain blocks. The hints, enabled in the store settings (Stores -> Configuration -> Advanced -> Developer), can also prove helpful for this type of search.



Debug

Enabled Template Path Hints for Storefront	No	[STORE VIEW]
Enabled Template Path Hints for Admin	No	[GLOBAL]
Add Block Names to Hints	No	[STORE VIEW]

For actions search, use the directories structure, because actions are located at the module directory the following way: <module_dir>/Controllers/<Controller>/<Action>). Front Name is specified in <module_dir>/etc/<area>/routes.xml, so finding it via the file search is a relatively simple task.

To search for the model functionality, use the full class or interface name (including namespace). Conduct the file search, specifying the full name of the class or the interface (without leading backlash).

In case of SSH access, you can use `grep` to search files with particular content:

```
grep -r 'String that you are looking' [path/to/search]
```

Example (search files with content "Catalog\Model\ProductRepository" in vendor/magento/module-catalog folder):

```
grep -r 'Catalog\\Model\\ProductRepository' vendor/
```

In case you know the file name, you can use `find`:

```
find [path/to/search] -name 'FileNameMask'
```

Example (search all webapi.xml files in `vendor` folder):

```
find vendor/ -name webapi.xml
```

When you know both the filename and its contents:

```
find [path/to/search] -name 'FileNameMask' -exec grep 'String that
you are looking' {} +
```

Example (search *.xml files with content "Catalog\Model\ProductRepository" in vendor folder):

```
find vendor/ -name '*.xml' -exec grep
'Catalog\\Model\\ProductRepository' {} +
```

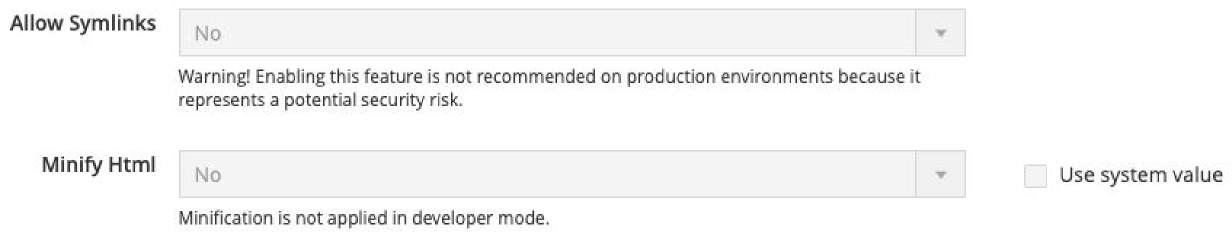
1.3 Utilize configuration XML and variables scope

Determine how to use configuration files in Magento 2. Which configuration files correspond to different features and functionality?

In Magento 2, the configuration is stored at the following locations:

1. In xml files of modules, themes, languages and app/etc folder.
2. In the database in core_config_data table.
3. In app/etc/config.php and app/etc/env.php files.
4. In the framework variables.

One can modify the configuration from the admin panel only when it is stored in core_config_data and is not overridden in pp/etc/config.php, app/etc/env.php or via environment variables. In case it is overridden, it is disabled at the admin panel:


Allow Symlinks: No

Warning! Enabling this feature is not recommended on production environments because it represents a potential security risk.

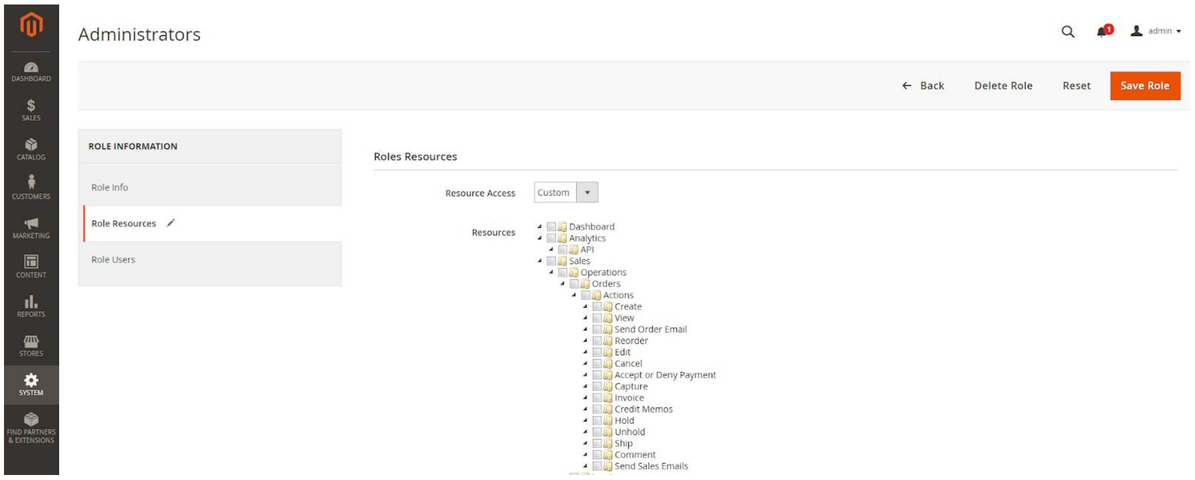
Minify Html: No — Use system value

Minification is not applied in developer mode.

app/etc/config.php and app/etc/env.php files contain Magento basic configuration (for instance, modules list, scopes, themes, database credentials, cache config, override core_config_data config and other). They are generated at the Magento 2 installation. app/etc/config.php file has shared configuration settings, while app/etc/env.php contains settings that are specific to the installation environment. As of the 2.2 release, the app/etc/config.php file is no longer an entry in the .gitignore file. This was done to facilitate pipeline deployment.

Below is the list of xml files inside the module:

1. etc/config.xml - contains the default values of the options from Stores > Configuration in admin panel menu, as well as other options, like class names (for instance, **<model>Amazon\Payment\Model\Method\AmazonLoginMethod</model>**) and attributes (for example, **<account backend_model="Magento\Config\Model\Config\Backend\Encrypted" />**).
2. etc/di.xml and etc/<area>/di.xml - contains the configuration for dependency injection
3. etc/events.xml and etc/<area>/events.xml - the list of the events and observers
4. etc/<area>/routes.xml - routers' list.
5. etc/acl.xml - adds module resources into the resource tree, allowing to set up access for different users.



6. etc/crontab.xml - adds and configures tasks for cron.
7. etc/module.xml - declares module name and version, as well as its dependencies from other modules.
8. etc/widget.xml - stores widget configuration.
9. etc/indexer.xml - declares a new indexation type. There, view_id is specified, which denotes the views described in etc/mview.xml.
10. etc/mview.xml is used to track database changes for a certain entity.
11. etc/webapi.xml - stores configurations for WEB API (REST/SOAP).
12. etc/view.xml - stores product images' values.
13. etc/product_types.xml - describes product types in a store.
14. etc/product_options.xml - describes the types of options that products can have and the classes that render options in the admin.

15. etc/extension_attributes.xml - the ability to add custom attribute, introduced in Magento 2 version. The file describes the attribute and its type, which can be simple, or complex, or have the form of an interface.
16. etc/catalog_attributes.xml - adds attributes to the groups. quote_item, wishlist_item, catalog_product, catalog_category, unassignable, used_in_autogeneration are the standard groups. To learn more, follow the link: https://www.atwix.com/magento-2/how-to-access-custom-catalog-attributes/
17. etc/adminhtml/system.xml - can relate to the admin section solely, adds Stores > Configuration settings and describes form sections and fields.
18. etc/adminhtml/menu.xml - can relate to admin area solely, adding the menu option in the admin panel.

Magento loads different areas and files separately; it also has different file loaders for each file type.

In Magento 2, XML configuration files have areas: global, frontend and adminhtml, crontab, graphql, webapi_rest, webapi_soap. You can find a list of them at Magento\Framework\App\AreaList class, defined via di.xml. Certain xml files can be specified for each area separately and some may not. For instance, event.xml file can be specified for each area (global, frontend, adminhtml, etc.), while module.xml can be specified only for global.

If config file is located at the module etc directory, its values are located in the global area. To specify configuration area, place the config file into etc/<area> folder. This is a new concept, introduced in Magento 2. Previously, in the first version of Magento, the visibility area was defined by a separate branch in XML file. This introduction allows to load configurations for various visibility areas separately. If the same parameters are specified for global and non-global areas (for instance, frontend or adminhtml), they will be merged and loaded together. The parameters, specified in non-global area, or located in etc/<area> folder, have the priority.

Configuration upload is executed in three steps:

1. System level configurations upload. Loading of the files, necessary for Magento 2 launch (like config.php).
2. Global area configurations upload. Loading of the files, located in app/etc/ Magento 2 directory, such as di.xml, as well as files that relate to the global area and are directly located in modules' etc/ folders.

3. Specific areas configurations upload. Loading of the files, located at etc/adminhtml or etc/frontend folders.

Configuration files are merged according to their full xPaths. Specific attributes are defined in the $ idAttributes array as identifiers. When two files are merged, they contain all the nodes and values from the original files. The second XML file either adds or replaces the nodes of the first XML file.

Each XML file type is validated by the corresponding XSD validation scheme. All validation schemes are in etc / directories of the module. The most important schemes are located in the Magento framework-e (vendor / magento / framework); for example, XSD for acl.xml is located in vendor/magento/framework/Acl/etc/acl.xsd directory.

All Magento 2 configuration files are processed by the Magento\Framework\Config\ *classes. These classes load, combine, validate and process configurations, converting them into the needed format array. If one needs to modify the standard loading procedure, they must create a single or several classes that inherit the interfaces:

- \Magento\Framework\Config\DataInterface - allows to get configuration value and merge two configurations together. \Magento\Framework\Config\Data class realizes this interface and saves the configuration in cache in order to speed up the repeated website upload.
- \Magento\Framework\Config\ScopeInterface - allows to specify and get the current scope.
- \Magento\Framework\Config\FileResolverInterface - runs the config files search, returns the array or iterator. Keys are the absolute paths, the value is their content.
- \Magento\Framework\Config\ReaderInterface - reads configuration data. \Magento\Framework\Config\Reader\Filesystem is the standard reader.
- \Magento\Framework\Config\ConverterInterface - converts Merged DOM object into the array.
- \Magento\Framework\Config\SchemaLocatorInterface - specifies the path to validation schemes.
- \Magento\Framework\Config\ValidationStateInterface - defines whether DOM validation with schema is needed.

Magento 2 has two types of validation for XML configuration files: before and after the merge. The schemes can be the same or differ from each other.

One can create the following elements for a custom configuration file:

- XML file
- XSD schema
- Config PHP file
- Config reader
- Schema locator
- Converter

Not all those elements are necessary. Instead of creating them, one can use virtualType in di.xml and create only the following elements:

- XML file
- XSD schema
- Converter

To make an example of configuration file creation, let us examine product_types.xml file from Magento_Catalog module. This file allows each module to add a custom product type; afterwards, the files will get validated and merge.

1. We begin with XSD file creation. Before the merger, Magento_Catalog uses product_types.xsd validation scheme and product_types_merged.xsd scheme for the merged XML file.
2. Create the configuration PHP file for access to the file data; in our case, it will be Config.php. To provide access to the product_types.xml file data, it implements the Magento\Catalog\Model\ProductType\ConfigInterface interface and realizes all its methods.
3. We should get reader class in Config.php in the constructor. In our case, it's Magento\Catalog\Model\ProductType\Config\Reader. This is a small class with a certain $_idAttributes attribute. In $fileName variable at the constructor we define the XML file name.
4. Magento\Catalog\Model\ProductType\Config\SchemaLocator implements two methods: getSchema and getPerFileSchema return the path to merged XSD and common XSD files. In the constructor, we define these paths in $_schema and $_perFileSchema attributes.
5. Convertor class creation. In our case: Magento\Catalog\Model\ProductType\Config\Converter it implements

\Magento\Framework\Config\ConverterInterface and realizes convert method that converts the merged DOM tree of nodes into the array.

1.4 Demonstrate how to use dependency injection

Describe Magento's dependency injection approach and architecture. How are objects realized in Magento?

Dependency Injection is a design pattern based on the inversion of control principle. This pattern centers around relations between the objects and its dependencies. Instead of creating dependencies manually, all the necessary dependencies are passed into the object with the help of external container. This approach allows to avoid strong components coupling, for the object is not required to create custom dependencies. The dependencies container, in its turn, determines which implementations should be passed to the objects at their creation, depending on the necessary behavior or configuration.

Dependency inversion principle claims that high-level classes should use low level objects' abstractions instead of working with them directly.

Object Manager

Magento 2 applies dependency injection for the functionality, which was offered by Mage class in Magento 1.

```
namespace Magento\Backend\Model\Menu;
class Builder
{
    public function __construct(
        Magento\Backend\Model\Menu\ItemFactory $menuItemFactory,
        Magento\Backend\Model\Menu $menu
    ) {
        $this->_itemFactory = $menuItemFactory;
        $this->_menu = $menu;
    }
}
```

ObjectManager is used as dependencies container, configured with di.xml files.
ObjectManager is responsible for:

- Objects' creation in factories and proxys
- Return of one and the same object instance at the continuous requests*
- Selection of the suitable implementation at the interface query
- Automatic class creation depending on constructor arguments.

* If shared=false attribute is specified for a certain type in di.xml, then a new object will be created at the future requests.

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

A centralized process creating object instances decreases code coherency and lowers the incompatibility risk in case the object realization changes.

Identify how to use DI configuration files for customizing Magento.

Each module can have a global and an area-specific di.xml file. Area-specific di.xml files are recommended to be applied for dependencies configuration for presentation layer, while global file - for all the rest.

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

Abstraction-implementation mappings and class rewrites

Abstraction-implementation mappings and class rewrites are applied in case the constructor requests the object by its interface. ObjectManager utilizes this configuration to resolve which implementation should be used for the current area.

```
<config>
    <preference for="Magento\Core\Model\UrlInterface"
type="Magento\Core\Model\Url" />
</config>
```

A similar approach can be used to substitute not only the interfaces, but the classes themselves.

```
<config>
    <preference for="Magento\Core\Model\Url"
type="Vendor\Module\Model\NewUrl" />
</config>
```

Virtual Types

Virtual Type allows to modify any dependencies arguments and therefore modify class behavior without changing the operation of other classes that depend on the original.

```
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:ObjectManager/et
c/config.xsd">
    <virtualType name="moduleConfig"
type="Magento\Core\Model\Config">
        <arguments>
```

```
            <argument name="type" xsi:type="string">system</argument>
        </arguments>
    </virtualType>
</config>
```

ObjectManager should never be called directly, because Magento Framework makes the call automatically. Factory or proxy classes, as well as unit texts or static and magic methods (wakeup, sleep) can be considered exceptions, for they are majorly generated by the framework automatically.
Magento 2 code contains direct ObjectManager calls, which exist only for backward compatibility and should not be used as an example.

Dependencies compilation

```
bin/magento setup:di:compile
```

Magento 2 applies a specific utilita for compiling dependencies of all classes. The utilita creates a file that contains the dependencies of all objects for ObjectManager, based on constructor arguments with the help of php reflection features. Such service classes as factories, proxies and plugins, are generated as well.

1.5 Demonstrate ability to use plugins

Demonstrate how to design complex solutions using the plugin's life cycle. How do multiple plugins interact, and how can their execution order be controlled?

Magento 2 Plugin (Interceptor) is the class that allows to alter the behavior of other classes by calling the custom mode before, after or instead of conflict method call, allowing to minimize the probability of conflicts between various pieces of code that concern the same functionality.

Plugin Configuration

```
<module_dir>/etc/di.xml

<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:ObjectManager/et
c/config.xsd">
   <type name="{ObservedType}">
        <plugin name="{PluginName}" type="{PluginClassName}"
sortOrder="1" disabled="false"/>
    </type>
</config>
```

, where:
`{ObservedType}` - class name, the method that must be altered
`{PluginName}` - plugin name
`{PluginClassName}` - plugin class name
`sortOrder` - plugin call order
`disabled` - ability to disable the plugin

Plugin Methods

Before Methods

Before methods allow to modify target class method arguments before the method is called.

```
Namespace Vendor\Module\Plugin;

class Plugin
{
    public function beforeMethodName(\Vendor\Module\Model\TargetClass
$subject, $arg1, $arg2, $arg3 = null)
    {
        return [$arg1, $arg2, $arg3];
    }
}
```

The name of the plugin method is concatenated 'before' and the method name, which arguments must be altered. The first argument of this method is the class, the method of which is called. The rest of the arguments correspond to the called methods arguments, including default values. It is also possible to use "..." token to get all the arguments (https://www.php.net/manual/en/functions.arguments.php#functions.variable-arg-list.new). Example:

```
public function beforeMyMethod($subject, ...$args)
{
      return $args;
}
```

Before methods should return the arguments array, if they need to be overridden, or null, if there is no need for that.

After Methods

After methods allow to modify the result of the target method.

Namespace Vendor\Module\Plugin;

```
class Plugin
{
    public function afterMethodName(\Vendor\Module\Model\TargetClass
$subject, $result)
    {
        return $result;
    }
}
```

The first argument, the same as before methods, is target class instance, the second is the returned value of the original method, while the third and further are the original method arguments.

Around Methods

Namespace Vendor\Module\Plugin;

```
class Plugin
{
    public function aroundMethodName(\Vendor\Module\Model\TargetClass
$subject, callable $proceed, $arg1, $arg2, $arg3)
    {
        $result = $proceed($arg1, $arg2, $arg3);
        return $result;
    }
}
```

Around methods allow to execute the code before and after the target method in one place.
$proceed argument is PHP closure, that in its turn calls the target method.
Such methods allow to completely substitute the target method.

Plugin Sorting

Plugin sortOrder parameter allows to identify what the order plugin methods will be called in case multiple plugins are observing the same method.

	Plugin1	Plugin2	Plugin3
sort order	10	20	30
before	beforeMethod()	beforeMethod()	beforeMethod()
around		aroundMethod()	aroundMethod()
after	afterMethod()	afterMethod()	afterMethod()

In this case, plugin processing is executed in the following order:

1. Plugin1::beforeMethod()
2. Plugin2::beforeMethod()

3. Plugin2::aroundMethod() (Magento calls the first half until callable)
    a. Plugin3::beforeMethod()
    b. Plugin3::aroundMethod() (Magento calls the first half until callable)
        i. TargetClass::method()
    c. Plugin3::aroundMethod() (Magento calls the second half after callable)
    d. Plugin3::afterMethod()
4. Plugin2::aroundMethod() (Magento calls the second half after callable)
5. Plugin2::afterMethod()
6. Plugin1::afterMethod()

How do you debug a plugin if it doesn’t work? Identify strengths and weaknesses of plugins.

Magento 2 generates an Interceptor class for each class that has plugins. This class inherits the original plugins and contains the code that calls the plugins in the assigned order. Therefore, any plugin debug can be started with the Interceptor class.
Plugins enable to modify the application behavior without the need to modify or substitute the original classes, which allows to impact the application flow flexibly.
On the other hand, application of plugins, around methods in particular, complicates code readability and increases stack trace, as well as the non-functioning application, if the plugins are applied without due accuracy. If you do not call $proceed() in around plugin, the plugins whose sortOrder is larger than the current plugin will not be called. The original method will not be called as well.

What are the limitations of using plugins for customization? In which cases should plugins be avoided?

Compared to Magento 1, class rewrites plugins do not inherit the target class, allowing several plugins that modify one and the same method to have no conflicts with each other. However, due to the same reasons, plugins have certain limitations.

Plugins can not be used with:

1. Final methods and classes
2. Protected/private methods

3. Static methods
4. __construct methods
5. Virtual types
6. Objects that are instantiated before Magento\Framework\Interception is bootstrapped
7. Objects that were initialized not with ObjectManager

The main purpose of plugins is to modify the certain method input, output or execution. In case the data is not modified (for instance, when order details are sent to the 3rd party ERP), then it is recommended to apply observers instead of plugins.

1.6 Configure event observers and scheduled jobs

Events are commonly used in applications to handle external actions or input. Each action is interpreted as an event.

Events are part of the Event-Observer pattern. This design pattern is characterized by objects (subjects) and their list of dependents (observers). It is a very common programming concept that works well to decouple the observed code from the observers. Observer is the class that implements Magento\Framework\Event\ObserverInterface interface.

According to
https://devdocs.magento.com/guides/v2.3/coding-standards/technical-guidelines.html
:
All values (including objects) passed to an event MUST NOT be modified in the event observer. Instead, plugins SHOULD BE used for modifying the input or output of a function.

Therefore, if there is a need to modify the input data, use plugins instead of events.

Demonstrate how to configure observers. How do you make your observer only be active on the frontend or backend?

In Magento 2 there is a special event manager class - Magento\Framework\Event\Manager that fires events. This class can be obtained through dependency injection by defining the dependency in your constructor.
Look for the mentioned notes in the code Magento 2:
$this->eventManager->dispatch(‘event_name’, ['myEventData' => $event_arguments]);

Event Observers in Magento 2 can be configured in a separate file events.xml. It should be created in <module_dir>/etc directory, if observer is associated with globally events, or in <module_dir>/etc/<area> (like <module_dir>/etc/frontend or <module_dir>/etc/adminhtml) if observer to only watch for events in specific area.

```
<?xml version="1.0" encoding="UTF-8"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:Event/etc/events
.xsd">
   <event name="my_module_event">
       <observer name="observer_name"
instance="Namespace\Modulename\Observer\MyObserver" />
   </event>
</config>
```

In instance attribute we declare observer class name. This class has to implement Magento\Framework\Event\ObserverInterface::execute(Observer $observer) method. The $observer object has an $event object (available through $observer->getEvent()), which contains the event’s parameters.

```
namespace Namespace\Modulename\Observer;

use Magento\Framework\Event\ObserverInterface;
use Magento\Framework\Event\Observer;
class MyObserver implements ObserverInterface
{
   public function __construct() {
       //You can use dependency injection
   }

   public function execute(Observer $observer)
   {
      ...
   }
}
```

Demonstrate how to configure a scheduled job

To demonstrate how to configure a scheduled job, we create in the module a crontab.xml file with the similar content:

```
<?xml version="1.0" encoding="UTF-8"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:module:Magento_Cron:etc/cr
ontab.xsd">
    <group id="default">
        <job name="my_module_cron_job"
instance="Vendor\Module\Model\Cron" method="run">
            <!-- Use schedule or config_path, not both -->
            <schedule>0 * * * *</schedule>
            <config_path>my_module/my_group/my_setting</config_path>
        </job>
    </group>
</config>
```

Group element determines to which group cron jobs should be tied. Group is declared in cron_groups.xml file and contains group configurations. The events inside the group have a general queue, while several groups can be launched simultaneously. Example:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:module:Magento_Cron:etc/cr
on_groups.xsd">
    <group id="default">
        <schedule_generate_every>15</schedule_generate_every>
        <schedule_ahead_for>20</schedule_ahead_for>
        <schedule_lifetime>15</schedule_lifetime>
        <history_cleanup_every>10</history_cleanup_every>
        <history_success_lifetime>60</history_success_lifetime>
        <history_failure_lifetime>4320</history_failure_lifetime>
        <use_separate_process>0</use_separate_process>
    </group>
</config>
```

Job element contains name (name of the job), instance (job class name) and method (job method name in the class) attributes. Also, job contains schedule element (http://www.nncron.ru/help/EN/working/cron-format.htm) or config_path (configuration path to the schedule value).

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

Magento 2 stores database configuration in core_config_data table. This configuration can be overridden:

1. In app/etc/config.php file
2. In app/etc/env.php file
3. In the area variables

To learn more, follow the link:
https://devdocs.magento.com/guides/v2.3/config-guide/prod/config-reference-var-name.html

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

In Magento1 we can use of automatically available events. For example:

```
protected function _beforeSave()
{
   ...
   Mage::dispatchEvent($this->_eventPrefix.'_save_before',
$this->_getEventData());
   ...
}
protected function _getEventData()
{
   return array(
       'data_object'       => $this,
       $this->_eventObject => $this,
   );
}
```

This event is triggered before saving object, if it extends Mage_Core_Model_Abstract class. When we create a new class, which extends Mage_Core_Model_Abstract, we can declare $eventPrefix = “namespace_modulename” and use new event namespace_module_save_before.

We have the same ability in Magento2. In class Magento\Framework\Model\AbstractModel:

```
public function beforeSave()
{
 ...
   $this->_eventManager->dispatch($this->_eventPrefix .
'_save_before', $this->_getEventData());
  ....
}
```

In Magento1 and Magento 2 we have the same automatically available events.

Models:

```
[$eventPrefix]_load_before
[$eventPrefix]_load_after
[$eventPrefix]_save_before
[$eventPrefix]_save_after
[$eventPrefix]_save_commit_after
[$eventPrefix]_delete_before
[$eventPrefix]_delete_after
[$eventPrefix]_delete_commit_after
```

Controllers:

```
controller_action_predispatch_[ROUTE_NAME]
controller_action_predispatch_[FULL_ACTION_NAME]
controller_action_postdispatch_[ROUTE_NAME]
controller_action_postdispatch_[FULL_ACTION_NAME]
controller_action_layout_render_before_[FULL_ACTION_NAME]
```

1.7 Utilize the CLI

Describe the usage of bin/magento commands in the development cycle. Which commands are available? How are commands used in the development cycle?

Magento 2 allows you to execute numerous operations using the command line interface. It also contains a large number of commands that allow to flush the cache, change deployment mode or reindex the necessary index quickly. To apply the following commands, you should:

1. Login into your Magento 2 server via SSH as Magento file system owner

```
$ ssh magento_user@server.com
```

2. Change directory to Magento 2 installation directory

```
$ cd /var/www/magento/
```

Afterward, we can use Magento CLI the following way:

```
 $ bin/magento COMMAND
```

For example, list command will put out a list of available actions:

```
 $ bin/magento list
```

Demonstrate an ability to create a deployment process.

A standard Magento 2 deployment process includes the following steps and commands:

1. Enable maintenance mode - `$ bin/magento maintenance:enable`
2. Perform database migrations - `$ bin/magento setup:upgrade`
3. Compile necessary code - `$ bin/magento setup:di:compile`
4. Publish static files - `$ bin/magento setup:static-content:deploy`
5. Disable maintenance mode - `# bin/magento maintenance:disable`

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

Magento 2 can operate in three possible modes:

1. Default - is enabled by default. It is aimed for neither development nor production use. This is an average mode between developer and production modes.
    - At the request, static files are copied into pub/static directory and afterwards are put out from it.
    - Exceptions are not put out at the screen and not recorded in log files
    - Enabled automatic code compilation
    - X-Magento-* HTTP response headers are hidden.

2. Developer - the mode for development. Static files are generated on demand. The mode also increases the amount of debug information. Static files caching is also disabled and static files are put out with symlinks.
    - Exception and backtraces are put out at the screen
    - Automatic code compilation is enabled
    - X-Magento-* HTTP response headers are enabled
    - XML schema validation is enabled
    - Slow application operation due to automatic code compilation on the fly.
3. Production - is applied for working in production environment. Only in this mode, the maximum operation speed is available due to the required static files generation during the deployment, not at the request, like in other modes.

    - Exceptions are recorded only in log files and not put out at the screen
    - Static files are put out from cache (the files are generated in advance with $ php bin/magento setup:static-content:deploy command)
    - Automatic code compilation is disabled (it is executed in advance with $ php bin/magento setup:di:compile command)
    - X-Magento-* HTTP response headers are hidden.

To learn the current deployment mode, enter the command:
$ bin/magento deploy:mode:show

Changing Deployment Mode

To change the current deployment mode, enter the command:

```
$ bin/magento deploy:mode:set {mode} [-s|--skip-compilation]
```

where:

- {mode} is the required mode (default, developer or production)
- –skip-compilation - is an optional parameter that allows to skip code compilation after the deployment mode is changed.

When the deployment mode is changed, the var/cache folders will be cleared, except for generated/metadata, generated/code, var/view_preprocessed, pub/static files .htaccess. To avoid this, apply –skip-compilation flag.

1.8 Demonstrate the ability to manage the cache

Describe cache types and the tools used to manage caches. How do you add dynamic content to pages served from the full page cache?

Magento stores several cache types to prevent repeated data calculation or loading:

- config - Various XML configurations that were collected across modules and merged
- layout - Layout building instructions
- block_html - Page blocks HTML
- collections - Collection data files
- reflection - API interfaces reflection data
- db_ddl - Results of DDL queries, such as index list, foreing keys, columns
- compiled_config - Compilation configuration. It is used for caching by Magento\Framework\Interception\PluginList\PluginList class and works with the compiled di configuration only (generated/metadata/<area>.php).
- eav - Entity types declaration cache. Stores information about entity types and their attributes.
- customer_notification - Customer Notification. At the moment, it is applied to provide the customer with updated session.
- config_integration - stores information about php interfaces
- config_integration_api - stores the information about integrations (deprecated)
- full_page - stores full html code of the pages with http headings
- config_webservice - REST and SOAP configurations, generated WSDL file
- translate - applied for data caching for Magento 2 Translate library

Out-of-the-box Magento supports two FPC types: built-in and Varnish. Varnish is recommended to be installed and applied for production use. Before the page gets into FPC, all its personalized content is deleted; this also applies for both built-in FPC and Varnish.

There are two methods to add dynamic contents for FPC:
First. For this purpose use in layout attribute cacheable=false in any block:

```
<referenceContainer name="content">
    <block class="Magento\Checkout\Block\Onepage\Success"
name="checkout.success" template="success.phtml" cacheable="false"/>
    <block class="Magento\Checkout\Block\Registration"
name="checkout.registration" template="registration.phtml"
cacheable="false"/>
</referenceContainer>
```

Attribute cacheable=false makes page with this block uncacheable by FPC. But be careful with this parameter, because if this block is located in all pages, then all these pages will not be cached by FPC.
Second.
You could use ajax for this purpose. Commonly, Magento has two types of content:

- Public. Public content is stored server side in your reverse proxy cache storage (e.g., file system, database, Redis, or Varnish) and is available to multiple customers. Examples of public content include header, footer, and category listing.
- Private. Private content is stored client side (e.g., browser) and is specific to an individual customer. Examples of private content include shopping cart, message and customer. The data is loaded from the server and saved to localStorage of user's browser. In case the sections are invalidate, they are loaded from the server repeatedly. This is realized by customer data JS module.

The strategy of deferring private content is perhaps best demonstrated by the following example. Starting in Magento_Theme::view/frontend/templates/html/header.phtml, we see the following:

```
<li class="greet welcome" data-bind="scope: 'customer'">
    <!-- ko if: customer().fullname  -->
    <span class="logged-in" data-bind="text: new String('<?=
$block->escapeHtml(__('Welcome, %1!', '%1')) ?>').replace('%1',
customer().fullname)">
    </span>
    <!-- /ko -->
    <!-- ko ifnot: customer().fullname  -->
    <span class="not-logged-in" data-bind='html:"<?=
$block->escapeHtml($welcomeMessage) ?>"'></span>
    <?= $block->getBlockHtml('header.additional') ?>
    <!-- /ko -->
</li>
```

Magento has different FPC for each group of customers, applying X-Magento-Vary Cookie for this. If the page with the specified URL and X-Magento-Vary Cookie is stored in FPC, then it is put out to the user, saving loading time. X-Magento-Vary Cookie value is generated using Magento\Framework\App\Http\Context class. In order to add a custom FPC division (for instance, according to age: < 18 and >= 18), one must call \Magento\Framework\App\Http\Context::setValue($name, $value, $default method.

Describe how to operate with cache clearing. How would you clean the cache?

There are three ways to clean the cache:

- From Admin
- Use bin/magento CLI
  - bin/magento cache:clean
  - bin/magento cache:flush

  Execute the bin/magento setup:config:set –http-cache-hosts=SOME_HOST1,SOME_HOST2:PORT2 command beforehand in order to enable cache clearing in varnish. In this command, SOME_HOST1,SOME_HOST2:PORT2 are the addresses of varnish services, divided by a comma.

- Clean manually
  - File cache: “rm -rf var/cache/*”, “rm -rf var/page_cache/*”
  - Redis: “redis-cli flushall”
  - Restart services (Varnish, Redis)

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

When you clean cache, you do it by tags, and if in cache item is not associated with cache type tag, it will never be removed from storage.
When you flush cache, you remove all cache records in storage. Sometimes it's more preferred than clean cache.

Describe how to clear the cache programmatically.

To clean cache for specific model object, you can use:

```
\Magento\Framework\Event\ManagerInterface->dispatch('clean_cache_by_t
ags', ['object' => $model]);
```

To clean entire cache type, you can use:

```
\Magento\Framework\App\Cache\TypeListInterface->cleanType($typeCode)
```

To invalidate entire cache type, you can use:

```
\Magento\Framework\App\Cache\TypeListInterface->invalidate($typeCode)
```

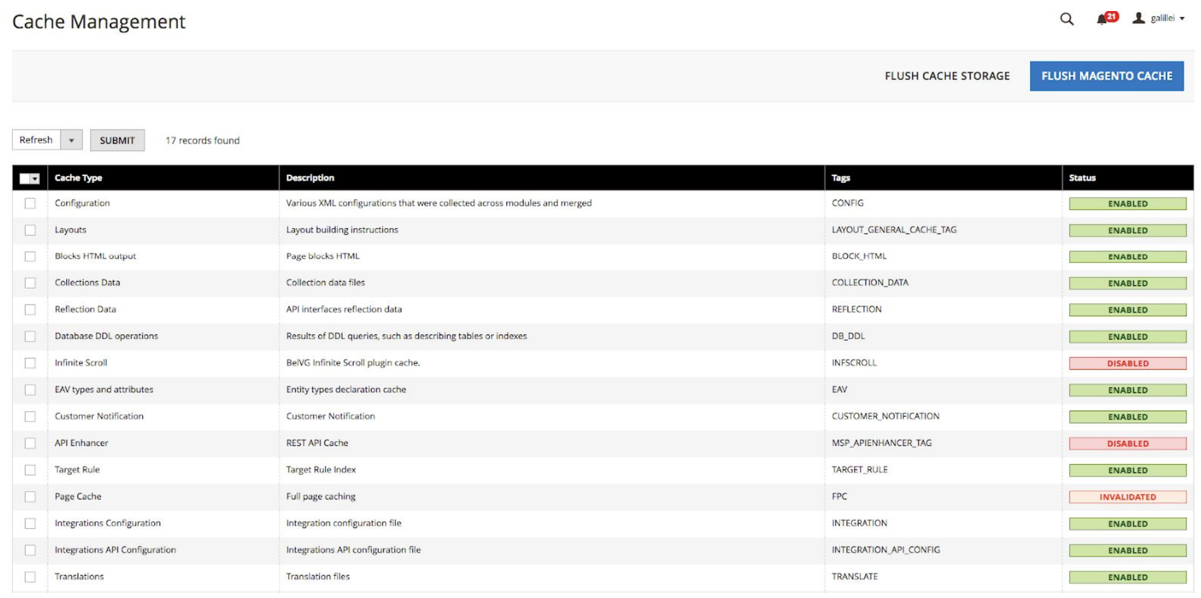
The cache tags are generated at block level, with each block class implementing the IdentityInterface which means they must implement a getIdentities method, which must return a unique identifier. For example:

```
...
namespace Magento\Cms\Block;
...
use Magento\Framework\View\Element\AbstractBlock;
use Magento\Framework\DataObject\IdentityInterface;
...
class Page extends AbstractBlock implements IdentityInterface
{
     ...
     public function getIdentities()
     {
     return [\Magento\Cms\Model\Page::CACHE_TAG . '_' .
$this->getPage()->getId()];
     }
     ...
}
```

When the front controller response is ready, the FPC combines all the block tags from the layout, and then adds them to the response in a X-Magento-Tags custom HTTP header. The different FPC options then handle the header differently. Varnish stores the header along with the rest of the page when it is cached, so no additional work is required. The built-in option however needs some additional code to pull the tags back out of the X-Magento-Tags header so that they can be associated with the response when it is stored in the configured storage (e.g. Redis).

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

You can clean cache from admin panel. System->Cache Management



Cache Type	Description	Tags	Status
Configuration	Various XML configurations that were collected across modules and merged	CONFIG	ENABLED
Layouts	Layout building instructions	LAYOUT_GENERAL_CACHE_TAG	ENABLED
Blocks HTML output	Page blocks HTML	BLOCK_HTML	ENABLED
Collections Data	Collection data files	COLLECTION_DATA	ENABLED
Reflection Data	API interfaces reflection data	REFLECTION	ENABLED
Database DDL operations	Results of DDL queries, such as describing tables or indexes	DB_DDL	ENABLED
Infinite Scroll	BelVG Infinite Scroll plugin cache.	INFSCROLL	DISABLED
EAV types and attributes	Entity types declaration cache	EAV	ENABLED
Customer Notification	Customer Notification	CUSTOMER_NOTIFICATION	ENABLED
API Enhancer	REST API Cache	MSP_APIENHANCER_TAG	DISABLED
Target Rule	Target Rule Index	TARGET_RULE	ENABLED
Page Cache	Full page caching	FPC	INVALIDATED
Integrations Configuration	Integration configuration file	INTEGRATION	ENABLED
Integrations API Configuration	Integrations API configuration file	INTEGRATION_API_CONFIG	ENABLED
Translations	Translation files	TRANSLATE	ENABLED

In this page you can clean/enable/disable cache by cache types or full cache. Also you can clean cache from console:

`bin/magento cache:clean` - clean cache,
`bin/magento cache:clean <cache_type>` - clean cache only for <cache_type>
`bin/magento cache:status` - you could see statuses and types of cache
`bin/magento cache:enable <cache_type>` - enable cache
`bin/magento cache:flush` - flush cache

Section 2: Request Flow Processing

2.1 Utilize modes and application initialization

Identify the steps for application initialization.

Step 1
The request to index.php (entry point) in the site root or in pub/index.php. In developer mode, index.php in the website root is commonly used as an entry point, while for production mode, pub/index.php is recommended for production mode.

Step 2

Connect bootstrap.php file

```
try {
    require __DIR__ . '/app/bootstrap.php';
} catch (\Exception $e) {
where autoload.php is connected (autoloader is loaded)
require_once __DIR__ . '/autoload.php';
```

Step 3

Autoloader allows to call create() method of the Bootstrap class

```
$bootstrap = \Magento\Framework\App\Bootstrap::create(BP, $_SERVER);
That returns Bootstrap object, that, in its turn, contains
ObjectManagerFactory object.
public static function create($rootDir, array $initParams,
ObjectManagerFactory $factory = null)
{
    self::populateAutoloader($rootDir, $initParams);
    if ($factory === null) {
        $factory = self::createObjectManagerFactory($rootDir,
$initParams);
```

```
    }
    return new self($factory, $rootDir, $initParams);
}
```

Step 4
Then, createApplication() of the Bootstrap object is called.
$app = $bootstrap->createApplication(\Magento\Framework\App\Http::class);
In this method, an instance of the Magento\Framework\App\Http class is created with the help of ObjectManager and is returned into index.php.

```
public function createApplication($type, $arguments = [])
{
    try {
        $application = $this->objectManager->create($type,
$arguments);
        if (!($application instanceof AppInterface)) {
            throw new \InvalidArgumentException("The provided class
doesn't implement AppInterface: {$type}");
        }
        return $application;
    } catch (\Exception $e) {
        $this->terminate($e);
    }
}
```

Step 5
Then, call run() method of the Bootstrap object to launch the application that will call launch() method of the application object.

```
$bootstrap->run($app);

public function run(AppInterface $application)
{
    try {
        try {
            \Magento\Framework\Profiler::start('magento');
            $this->initErrorHandler();
            $this->assertMaintenance();
            $this->assertInstalled();
            $response = $application->launch();
            $response->sendResponse();
            \Magento\Framework\Profiler::stop('magento');
        } catch (\Exception $e) {
            \Magento\Framework\Profiler::stop('magento');
            if (!$application->catchException($this, $e)) {
                throw $e;
            }
        }
    } catch (\Exception $e) {
        $this->terminate($e);
    }
}
```

Step 6

An instance of Http object performs the initial routing; as a result, it determines the area from URL and sets in $this->_state->setAreaCode($areaCode). After the area is set, the required configuration for that area is loaded.
Then, an object of \Magento\Framework\App\FrontController class is created and its method - dispatch($this->_request) - is called, to which request is passed.

```
public function launch()
{
    $areaCode =
$this->_areaList->getCodeByFrontName($this->_request->getFrontName())
;
    $this->_state->setAreaCode($areaCode);

$this->_objectManager->configure($this->_configLoader->load($areaCode
));
    /** @var \Magento\Framework\App\FrontControllerInterface
$frontController */
    $frontController =
$this->_objectManager->get(\Magento\Framework\App\FrontControllerInte
rface::class);
    $result = $frontController->dispatch($this->_request);
    // TODO: Temporary solution until all controllers return
ResultInterface (MAGETWO-28359)
    if ($result instanceof ResultInterface) {
        $this->registry->register('use_page_cache_plugin', true,
true);
        $result->renderResult($this->_response);
    } elseif ($result instanceof HttpInterface) {
        $this->_response = $result;
    } else {
        throw new \InvalidArgumentException('Invalid return type');
    }
    // This event gives possibility to launch something before
sending output (allow cookie setting)
    $eventParams = ['request' => $this->_request, 'response' =>
$this->_response];

$this->_eventManager->dispatch('controller_front_send_response_before
', $eventParams);
    return $this->_response;
}
```

Step 7

In dispatch() method of the FrontCotroller class the current router and the current action controller are defined. Then, dispatch() method is called from action controller.

```
public function dispatch(RequestInterface $request)
{
    \Magento\Framework\Profiler::start('routers_match');
    $routingCycleCounter = 0;
    $result = null;
    while (!$request->isDispatched() && $routingCycleCounter++ < 100)
{
        /** @var \Magento\Framework\App\RouterInterface $router */
        foreach ($this->_routerList as $router) {
            try {
                $actionInstance = $router->match($request);
                if ($actionInstance) {
                    $request->setDispatched(true);
                    $this->response->setNoCacheHeaders();
                    if ($actionInstance instanceof
\Magento\Framework\App\Action\AbstractAction) {
                        $result =
$actionInstance->dispatch($request);
                    } else {
                        $result = $actionInstance->execute();
                    }
                    break;
                }
            } catch (\Magento\Framework\Exception\NotFoundException
$e) {
                $request->initForward();
                $request->setActionName('noroute');
                $request->setDispatched(false);
                break;
            }
        }
    }
    \Magento\Framework\Profiler::stop('routers_match');
    if ($routingCycleCounter > 100) {
        throw new \LogicException('Front controller reached 100
router match iterations');
    }
    return $result;
}
```

Step 8

Dispatch() method is implemented in Magento\Framework\App\Action\Action.php.
When we create custom actions, they are inherited from this class.
Action controller returns the object that realizes ResultInterface via execute() method.

```
public function dispatch(RequestInterface $request)
{
    $this->_request = $request;
    $profilerKey = 'CONTROLLER_ACTION:' .
$request->getFullActionName();
    $eventParameters = ['controller_action' => $this, 'request' =>
$request];
    $this->_eventManager->dispatch('controller_action_predispatch',
$eventParameters);
    $this->_eventManager->dispatch('controller_action_predispatch_' .
$request->getRouteName(), $eventParameters);
    $this->_eventManager->dispatch(
        'controller_action_predispatch_' .
$request->getFullActionName(),
        $eventParameters
    );
    \Magento\Framework\Profiler::start($profilerKey);

    $result = null;
    if ($request->isDispatched() && !$this->_actionFlag->get('',
self::FLAG_NO_DISPATCH)) {
        \Magento\Framework\Profiler::start('action_body');
        $result = $this->execute();
        \Magento\Framework\Profiler::start('postdispatch');
        if (!$this->_actionFlag->get('',
self::FLAG_NO_POST_DISPATCH)) {
            $this->_eventManager->dispatch(
                'controller_action_postdispatch_' .
$request->getFullActionName(),
                $eventParameters
            );
            $this->_eventManager->dispatch(
                'controller_action_postdispatch_' .
$request->getRouteName(),
                $eventParameters
            );
$this->_eventManager->dispatch('controller_action_postdispatch',
$eventParameters);
        }
        \Magento\Framework\Profiler::stop('postdispatch');
        \Magento\Framework\Profiler::stop('action_body');
    }
    \Magento\Framework\Profiler::stop($profilerKey);
    return $result ?: $this->_response;
}
```

Step 9
FrontController returns ResultInterface into Application Instance, which puts out a response.

```
public function launch()
{
    $areaCode =
$this->_areaList->getCodeByFrontName($this->_request->getFrontName())
;
    $this->_state->setAreaCode($areaCode);

$this->_objectManager->configure($this->_configLoader->load($areaCode
));
    /** @var \Magento\Framework\App\FrontControllerInterface
$frontController */
    $frontController =
$this->_objectManager->get(\Magento\Framework\App\FrontControllerInte
rface::class);
    $result = $frontController->dispatch($this->_request);
    // TODO: Temporary solution until all controllers return
ResultInterface (MAGETWO-28359)
    if ($result instanceof ResultInterface) {
        $this->registry->register('use_page_cache_plugin', true,
true);
        $result->renderResult($this->_response);
    } elseif ($result instanceof HttpInterface) {
```

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

To receive data from each request, create observer for controller_action_postdispatch event (Magento\Framework\App\Action\Action::dispatch()).
Realization example - class Magento\Customer\Observer\Visitor\SaveByRequestObserver.

Describe how to use Magento modes

Magento 2 can be launched in one of the three modes: developer, production and default. The main difference between the modes is the way Magento will get access to static files (CSS, JavaScript files, images, etc.)
There is also a maintenance mode, but it is aimed at denying access to the system.
To view the current mode, use the CLI command bin/magento deploy:mode:show. To switch modes, use bin/magento deploy:mode:set

What are pros and cons of using developer mode/production mode?

Developer mode

In Developer mode, static view files are generated every time they are requested. The symlinks of them are written to the pub/static directory. If you will change content of JS file, it will be updated in pub/static too because of symlink.
Uncaught exceptions are displayed in the browser instead of being logged. An exception is thrown whenever an event subscriber cannot be invoked.
Magento 2 validates XML files using schemas in this mode.
Use the Developer mode while developing customizations or extensions. The main benefit of this mode is that error messages are visible to you. It should not be used in production because it impacts the performance.

Production mode

You should run Magento in Production mode once it is deployed to a production server. Production mode provides the highest performance in Magento 2.
The most important aspect of this mode is that errors are logged to the file system and are never displayed to the user. In this mode, static view files are not created on the fly when they are requested; instead, they have to be deployed to the pub/static directory using the command-line tool. Any changes to view files require running the deploy tool again.

When do you use default mode?

Default mode

Default mode is how the Magento software operates if no other mode is specified.
In this mode, errors are logged to the files in var/reports and are never shown to the user. Static view files are materialized on the fly and then cached.
In contrast to the developer mode, view file changes are not visible until the generated static view files are cleared.Default mode is not optimized for a production environment, primarily because of the adverse performance impact of static files being materialized on the fly rather than generating and deploying them beforehand. In other words, creating static files on the fly and caching them has a greater performance impact than generating them using the static file creation command line tool.

How do you enable/disable maintenance mode?

bin/magento maintenance:enable
bin/magento maintenance:disable

Describe front controller responsibilities

- Gathering all routers (injected into the constructor using DI)
- Finding a matching controller/router
- Obtaining generated HTML to the response object

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

Front controllers are the first step in handling requests and work flows across all pages. Basically, the front controller controls all other controllers. In Magento 2, it gathers routes, matches controllers, and obtains the HTML generated to the response object. It is not used in the console.

2.2 Demonstrate ability to process URLs in Magento

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

Front Controller

Routing in Magento 2 is based on Front Controller Pattern. Front Controller is a design pattern, in which one component is responsible for processing all the incoming requests, redirecting them to the corresponding components, further results processing and returning the result to the browser.

FrontController iterates via the available routers and, if the action responsible for the current URL is successfully found, calls the \Magento\Framework\App\Action\AbstractAction::dispatch method.

Routers

All routers in Magento 2 should implement \Magento\Framework\App\RouterInterface interface and define \Magento\Framework\App\RouterInterface::match method. This method is responsible for matching and processing URL requests. In case of a successful match, router returns the corresponding action instance. When the needed action is not found, Front Controller is passed to the next router.

Magento 2 has four routers:

1. Base Router (\Magento\Framework\App\Router\Base) - sets module front name, controller and action names, controller module and route name if found. Processes standard Magento URLs.
2. CMS Router (\Magento\Cms\Controller\Router) - applied for processing CMS pages. Sets module name to “cms”, controller name to “page”, action name to “view” and page id depending on the requested page. Then, it forwards request but won't dispatch it. This will result in the next cycle of router checks by Front Controller, where Base Router, based on the set path, will call \Magento\Cms\Controller\Page\View that will indicate the required page.
3. UrlRewrite Router (\Magento\UrlRewrite\Controller\Router) - responsible for URL rewrites. Applies Url Finder to find a corresponding URL in the database and then returns forward the same way as CMS Router.
4. Default Router (\Magento\Framework\App\Router\DefaultRouter) - is applied when other routers are unable to find the suitable action; it is also responsible for 404 page.

In order to find out which module, controller and action are applied now, call the following $request methods (\Magento\Framework\App\Request\Http):

`$request->getControllerModule()` - get controller module (e.g. ‘Magento_Catalog’)
`$request->getControllerName()` - get controller name (e.g. ‘product’)
`$request->getActionName()` - get action name (e.g. ‘view’)

For instance, to define the current module, controller and action, temporarily add at the end of index.php file (or pub/index.php) the following code:

```
$request =
\Magento\Framework\App\ObjectManager::getInstance()->get('\Magento\Fr
amework\App\Request\Http');
var_dump($request->getControllerModule(),
$request->getControllerName(), $request->getActionName());
```

What is necessary to create a custom URL structure?

To create a custom router, first add it to \Magento\Framework\App\RouterList that passes into Front Controller and has all the available routers in the proper order. For this, use di.xml file in our module.
<module_dir>/etc/di.xml:

```
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:ObjectManager/et
c/config.xsd">
   <type name="Magento\Framework\App\RouterList">
       <arguments>
           <argument name="routerList" xsi:type="array">
               <item name="customrouter" xsi:type="array">
                   <item name="class"
xsi:type="string">Vendor\Module\Controller\CustomRouter</item>
                   <item name="disable"
xsi:type="boolean">false</item>
                   <item name="sortOrder" xsi:type="string">22</item>
               </item>
           </argument>
       </arguments>
   </type>
</config>
```

Afterward, create a CustomRouter class. <module_dir>/Controller/CustomRouter.php:

```
<?php
namespace Vendor\Module\Controller;

use Magento\Framework\App\ActionFactory;
use Magento\Framework\App\RouterInterface;
```

```
class CustomRouter implements RouterInterface
{
    protected $actionFactory;

    public function __construct(
        ActionFactory $actionFactory
    ) {
        $this->actionFactory = $actionFactory;
    }
```

```
public function match(\Magento\Framework\App\RequestInterface
$request)
    {
        $identifier = trim($request->getPathInfo(), '/');
        if (strpos($identifier, 'customrouter-test') !== false) {
            $request->setPathInfo('/customrouter/index/index/');
            // or
            $request->setModuleName('customrouter');
            $request->setControllerName('index');
            $request->setActionName('index');
            $request->setControllerModule('Module_Vendor');
            $request->setRouteName('customrouter');
            $request->setParams(['param1' => 'value']);
        } else {
            return false;
        }

        return
$this->actionFactory->create('Magento\Framework\App\Action\Forward');
        // or
        return
$this->actionFactory->create('Vendor\Module\Controller\Index\Index');
    }
}
```

If we applied Forward Action in router, create a routes.xml file to allow Base Router to configure request and find Action Class. If you configured the request and specified Action Class yourself, then there is no need to create such file.
<module_dir>/etc/<area>/routes.xml

```xml
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:App/etc/routes.x
sd">
   <router id="standard">
       <route id="customrouter" frontName="customrouter">
           <module name="Vendor_Module" />
       </route>
   </router>
</config>
```

Specify **standard** in Router id: for frontend, for adminhtml - **admin**. In this file you can also specify the configuration for another router. This configuration is not passed into the router class automatically; instead, it needs to be loaded with \Magento\Framework\App\Route\Config\Reader class. Base Router (\Magento\Framework\App\Router\Base) applies \Magento\Framework\App\Route\Config class for loading. This class loads the configuration only for default router of an area (default router is set via di.xml) and stores it in cache.

Route id is the id of the route. It is applied, for instance, for layout files naming ({route_id}_{controller}_{action}.xml). To get the ID, use $request->getRouteName().

Route frontName is the part of URL:
{magento-base-url}/{frontName}/{controller}/{action}. To get it, use $request->getModuleName().

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

URL Rewrites allow to put our catalog URLs (/catalog/product/id/123) in an easy-to-understand way (/black-tshirt.html). UrlRewrite Router (\Magento\UrlRewrite\Controller\Router) is responsible for their processing; it searches for a necessary URL in the database and, when it is found, it calls $request->setPathInfo('/' . $rewrite->getTargetPath()) first, and then - Forward Action. not-user-friendly path (like catalog/product/id/123) are stored in Target Path.

Finding matching URL rewrite

\Magento\UrlRewrite\Controller\Router::getRewrite method is responsible for database URL search.

```
protected function getRewrite($requestPath, $storeId)
{
   return $this->urlFinder->findOneByData([
       UrlRewrite::REQUEST_PATH => ltrim($requestPath, '/'),
       UrlRewrite::STORE_ID => $storeId,
   ]);
}
```

The method searches for a suitable URL Rewrite in url_rewrites table, based on request path and current store ID. In case the search is successful, \Magento\UrlRewrite\Service\V1\Data\UrlRewrite object is returned.

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

Generating URL rewrite

To create, modify or delete URL rewrites for catalog entities, apply Magento_CatalogUrlRewrite module. This module contains observers, responsible for catalog entities modification events. To clarify, let us make an example of URL Rewrite processing at the event of saving a product.

The process begins in observer method \Magento\CatalogUrlRewrite\Observer\ProductProcessUrlRewriteSavingObserver, called at the catalog_product_save_after event.
Events.xml

```
<event name="catalog_product_save_after">
   <observer name="process_url_rewrite_saving"
instance="Magento\CatalogUrlRewrite\Observer\ProductProcessUrlRewrite
SavingObserver"/>
</event>

public function execute(\Magento\Framework\Event\Observer $observer)
{
   /** @var Product $product */
   $product = $observer->getEvent()->getProduct();

   if ($product->dataHasChangedFor('url_key')
       || $product->getIsChangedCategories()
       || $product->getIsChangedWebsites()
       || $product->dataHasChangedFor('visibility')
   ) {
       $this->urlPersist->deleteByData([
           UrlRewrite::ENTITY_ID => $product->getId(),
           UrlRewrite::ENTITY_TYPE =>
ProductUrlRewriteGenerator::ENTITY_TYPE,
           UrlRewrite::REDIRECT_TYPE => 0,
           UrlRewrite::STORE_ID => $product->getStoreId()
       ]);

       if ($product->isVisibleInSiteVisibility()) {

$this->urlPersist->replace($this->productUrlRewriteGenerator->generat
e($product));
       }
   }
}
```

After product model is acquired from observer, an assessment is run to determine whether the product url_key and visibility have modified, as well as whether the relationship between categories and stores have changed.

If certain elements have changed, then the current URL rewrites for this product are deleted using \Magento\UrlRewrite\Model\Storage\DbStorage::deleteByData method. Afterward, if product settings allow its display, \Magento\CatalogUrlRewrite\Model\ProductUrlRewriteGenerator::generate method is called.

```
public function generate(Product $product, $rootCategoryId = null)
{
   if ($product->getVisibility() ==
Visibility::VISIBILITY_NOT_VISIBLE) {
       return [];
   }

   $storeId = $product->getStoreId();

   $productCategories = $product->getCategoryCollection()
       ->addAttributeToSelect('url_key')
       ->addAttributeToSelect('url_path');

   $urls = $this->isGlobalScope($storeId)
       ? $this->generateForGlobalScope($productCategories, $product,
$rootCategoryId)
       : $this->generateForSpecificStoreView($storeId,
$productCategories, $product, $rootCategoryId);

   return $urls;
}
```

After product visibility is checked, the method gets the collection of all categories that the product is associated with. Then, the dependencies of the current scope call \Magento\CatalogUrlRewrite\Model\ProductUrlRewriteGenerator::generateForGlobalScope or \Magento\CatalogUrlRewrite\Model\ProductUrlRewriteGenerator::generateForSpecificStoreView methods.

Consider the global scope situation.

```
protected function generateForGlobalScope($productCategories,
$product = null, $rootCategoryId = null)
{
   return
$this->getProductScopeRewriteGenerator()->generateForGlobalScope(
       $productCategories,
       $product,
       $rootCategoryId
   );
}

public function generateForGlobalScope($productCategories, Product
$product, $rootCategoryId = null)
{
   $productId = $product->getEntityId();
   $mergeDataProvider = clone $this->mergeDataProviderPrototype;

   foreach ($product->getStoreIds() as $id) {
       if (!$this->isGlobalScope($id) &&

!$this->storeViewService->doesEntityHaveOverriddenUrlKeyForStore(
               $id,
               $productId,
               Product::ENTITY
           )) {
           $mergeDataProvider->merge(
               $this->generateForSpecificStoreView($id,
$productCategories, $product, $rootCategoryId)
```

```
            );
        }
    }

    return $mergeDataProvider->getData();
}
```

For every available store, \Magento\CatalogUrlRewrite\Model\ProductScopeRewriteGenerator::generateForGlobalScope method checks whether a product url_key is duplicated and calls url rewrite generation for this store. As url rewrites for all the available stores are created, they are united in a single array with the \Magento\UrlRewrite\Model\MergeDataProvider class object.

```
public function generateForSpecificStoreView($storeId,
$productCategories, Product $product, $rootCategoryId = null)
{
    $mergeDataProvider = clone $this->mergeDataProviderPrototype;
    $categories = [];
    foreach ($productCategories as $category) {
        if (!$this->isCategoryProperForGenerating($category,
$storeId)) {
            continue;
        }
```

// category should be loaded per appropriate store if category's URL key has been changed

```
  $categories[] = $this->getCategoryWithOverriddenUrlKey($storeId,
$category);
    }

    $productCategories =
$this->objectRegistryFactory->create(['entities' => $categories]);

    $mergeDataProvider->merge(
        $this->canonicalUrlRewriteGenerator->generate($storeId,
$product)
    );
    $mergeDataProvider->merge(
        $this->categoriesUrlRewriteGenerator->generate($storeId,
$product, $productCategories)
    );
    $mergeDataProvider->merge(
        $this->currentUrlRewritesRegenerator->generate(
            $storeId,
            $product,
            $productCategories,
            $rootCategoryId
        )
    );
    $mergeDataProvider->merge(
        $this->anchorUrlRewriteGenerator->generate($storeId, $product,
$productCategories)
    );
    $mergeDataProvider->merge(
        $this->currentUrlRewritesRegenerator->generateAnchor(
            $storeId,
            $product,
            $productCategories,
            $rootCategoryId
        )
    );
    return $mergeDataProvider->getData();
}
```

\Magento\CatalogUrlRewrite\Model\ProductScopeRewriteGenerator::generateForSpecificStoreView method for each of the product categories receives a specific url_key for the current core. Then, with the help of \Magento\UrlRewrite\Model\MergeDataProvider instance, it merges the results of several calls:

a. \Magento\CatalogUrlRewrite\Model\Product\CanonicalUrlRewriteGenerator::generate - creates URL Rewrite that does not contain categories.

b. \Magento\CatalogUrlRewrite\Model\Product\CategoriesUrlRewriteGenerator::generate - creates URL Rewrite that includes all the possible categories.
c. \Magento\CatalogUrlRewrite\Model\Product\CurrentUrlRewritesRegenerator::generate - creates URL Rewrites, including all the possible URL rewrites for the current entity (e.g. custom URL rewrites)
d. \Magento\CatalogUrlRewrite\Model\Product\AnchorUrlRewriteGenerator::generate - generates URL Rewrites for anchor categories.
e. \Magento\CatalogUrlRewrite\Model\Product\CurrentUrlRewritesRegenerator::generateAnchor generates URL Rewrites for anchor categories taking into account the available URL rewrites.

As a result, all the methods are united in a single array and are returned to \Magento\UrlRewrite\Model\Storage\AbstractStorage::replace method, responsible for URL Rewrites persistence.

Describe how action controllers and results function.

Controllers

Controllers in Magento 2 differ from the typical MVC applications' controllers. In MVC applications, controller is a class, while action is the method of this class. In Magento 2, controller is a folder (or php namespace), while action is a class, located in this folder (in this php namespace). **Execute** method of the action returns the result object and occasionally processes input POST data. All actions inherit \Magento\Framework\App\Action\Action class.
Search and initialization of the needed action is performed in router. For instance, in Base Router:

```
public function match(\Magento\Framework\App\RequestInterface
$request)
{
   $params = $this->parseRequest($request);

   return $this->matchAction($request, $params);
}
```

\Magento\Framework\App\Router\Base::parseRequest method is responsible for dividing the requested URL into segments, while \Magento\Framework\App\Router\Base::matchAction method is looking for a suitable action.

```
protected function
matchAction(\Magento\Framework\App\RequestInterface $request, array
$params)
{
   $moduleFrontName = $this->matchModuleFrontName($request,
$params['moduleFrontName']);
   if (empty($moduleFrontName)) {
       return null;
   }

   /**
    * Searching router args by module name from route using it as key
    */
   $modules =
$this->_routeConfig->getModulesByFrontName($moduleFrontName);

   if (empty($modules) === true) {
       return null;
   }

   /**
    * Going through modules to find appropriate controller
    */
   $currentModuleName = null;
   $actionPath = null;
   $action = null;
   $actionInstance = null;

   $actionPath = $this->matchActionPath($request,
$params['actionPath']);
   $action = $request->getActionName() ?: ($params['actionName'] ?:
```

```
$this->_defaultPath->getPart('action'));
   $this->_checkShouldBeSecure($request, '/' . $moduleFrontName . '/'
. $actionPath . '/' . $action);

   foreach ($modules as $moduleName) {
       $currentModuleName = $moduleName;

       $actionClassName = $this->actionList->get($moduleName,
$this->pathPrefix, $actionPath, $action);
       if (!$actionClassName || !is_subclass_of($actionClassName,
$this->actionInterface)) {
           continue;
       }

       $actionInstance =
$this->actionFactory->create($actionClassName);
       break;
   }

   if (null == $actionInstance) {
       $actionInstance =
$this->getNotFoundAction($currentModuleName);
       if ($actionInstance === null) {
           return null;
       }
       $action = 'noroute';
   }

   // set values only after all the checks are done
   $request->setModuleName($moduleFrontName);
   $request->setControllerName($actionPath);
   $request->setActionName($action);
   $request->setControllerModule($currentModuleName);

$request->setRouteName($this->_routeConfig->getRouteByFrontName($modu
leFrontName));
   if (isset($params['variables'])) {
       $request->setParams($params['variables']);
   }
   return $actionInstance;
}
```

Each URL segment contains the information for the required action search. The segments in URL can be presented the following way:

```
{routeFrontName}/{controllerName}/{actionName}
```

Where:
`{routeFrontName}` - route front name as set in routes.xml file
`{controllerName}` - name of the controller
`{actionName}` - name of the action

For URL of custom-module/info/product kind
The route to the action class:

```
<module_dir>/Controller/Info/Product.php
```

How do controllers interact with another?

Forward Result

Forward Result (\Magento\Framework\Controller\Result\Forward) - allows to pass request processing to another controller without a redirect.

```
public function __construct(
    Magento\Framework\Controller\Result\Forward\Factory
$resultForwardFactory
) {
    $this->resultForwardFactory = $resultForwardFactory;
}

public function execute()
{
    $result = $this->resultForwardFactory->create();
    $result->forward('noroute');
    return $result;
}
```

Redirect Result

Redirect Result (\Magento\Framework\Controller\Result\Redirect) - allows to redirect the browser to another URL.

```
public function __construct(
    Magento\Framework\Controller\Result\Redirect\Factory
$resultRedirectFactory
) {
    $this->resultRedirectFactory = $resultRedirectFactory;
}

public function execute()
{
    $result = $this->resultRedirectFactory->create();
    $result->setPath('*/*/index');
    return $result;
}
```

How are different response types generated?

Responses

Action in Magento 2 can return several response types depending on its purpose and desired result.

Page Result

Page Result (\Magento\Framework\View\Result\Page) is the most common response type. Returning the object, Magento calls its renderResult method that performs page rendering based on the corresponding XML layout handle.

```
public function __construct(
    $pageFactory Magento\Framework\View\Result\PageFactory
) {
    $this->pageResultFactory = $pageFactory
}

public function execute()
{
    return $this->pageResultFactory->create();
}
```

JSON Result

JSON Result (\Magento\Framework\Controller\Result\Json) - allows to return the response in JSON format. Can be applied in API or AJAX requests.

```
public function __construct(
    Magento\Framework\Controller\Result\JsonFactory
$jsonResultFactory,
) {
    $this->jsonResultFactory = $jsonResultFactory;
}

public function execute()
{
    $result = $this->jsonResultFactory();

    $class = new Class;
    $class->data = value;
    $result->setData($class);
    return $result;
}
```

Raw Result

Raw Result (\Magento\Framework\Controller\Result\Raw) is utilized if you need to return the results to the browser as

```
is. public function __construct(
    Magento\Framework\Controller\Result\Raw $rawResultFactory ,
) {
    $this->rawResultFactory = $rawResultFactory;
}
public function execute()
{
    $result = $this->rawResultFactory->create();
    $result->setHeader('Content-Type', 'text/xml');
    $result->setContents('<root><block></block></root>);
    return $result;
}
```

2.3 Demonstrate ability to customize request routing

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

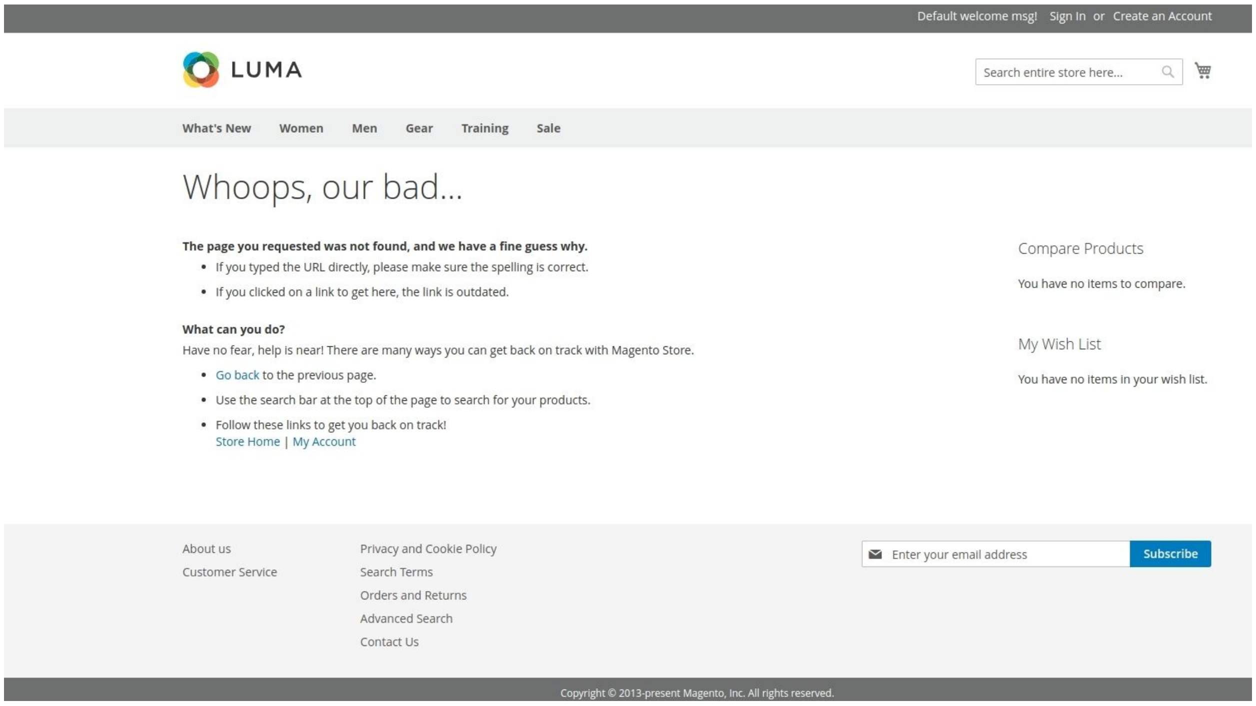
In web applications, such as Magento, routing is the act of providing data from a URL request to the appropriate class for processing. Magento routing uses the following flow:

index.php -> HTTP application -> FrontController -> Routing -> Action processing -> etc

The need to create a new router or modify the current one may appear when one needs to change the default Magento 2 URLs (processed by Base Router) for the custom ones, for instance, /belvg/index/mypage on /mypage.

How do you handle custom 404 pages?

This is how an out-of-the-box Magento 404 page looks. However, some store owners wish to have a custom 404 page, and in Magento 2, this is not a problem, for the platform provides a very flexible tool for modifying this element.


Default welcome msg! Sign In or Create an Account
LUMA
Search entire store here...
What's New
Women
Men
Gear
Training
Sale

Whoops, our bad...

The page you requested was not found, and we have a fine guess why.

- If you typed the URL directly, please make sure the spelling is correct.
- If you clicked on a link to get here, the link is outdated.

What can you do?

Have no fear, help is near! There are many ways you can get back on track with Magento Store.

- Go back to the previous page.
- Use the search bar at the top of the page to search for your products.
- Follow these links to get you back on track!
  Store Home | My Account

Compare Products

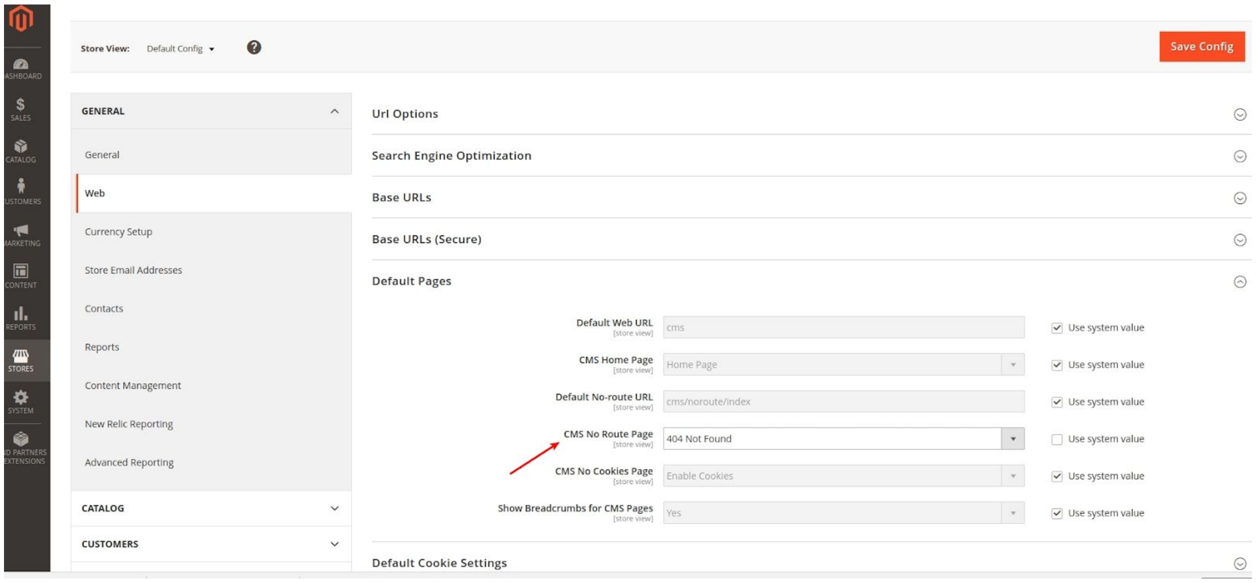
You have no items to compare.

My Wish List

You have no items in your wish list.

Copyright © 2013-present Magento, Inc. All rights reserved.

To modify the “Not Found“ (404 error) page, log in to the admin panel and navigate the following path: Store -> Configuration-> General -> Web -> Default pages -> CMS No Route Page to check what CMS page is set in the configurations.



Store View: Default Config

Url Options

Search Engine Optimization

Base URLs

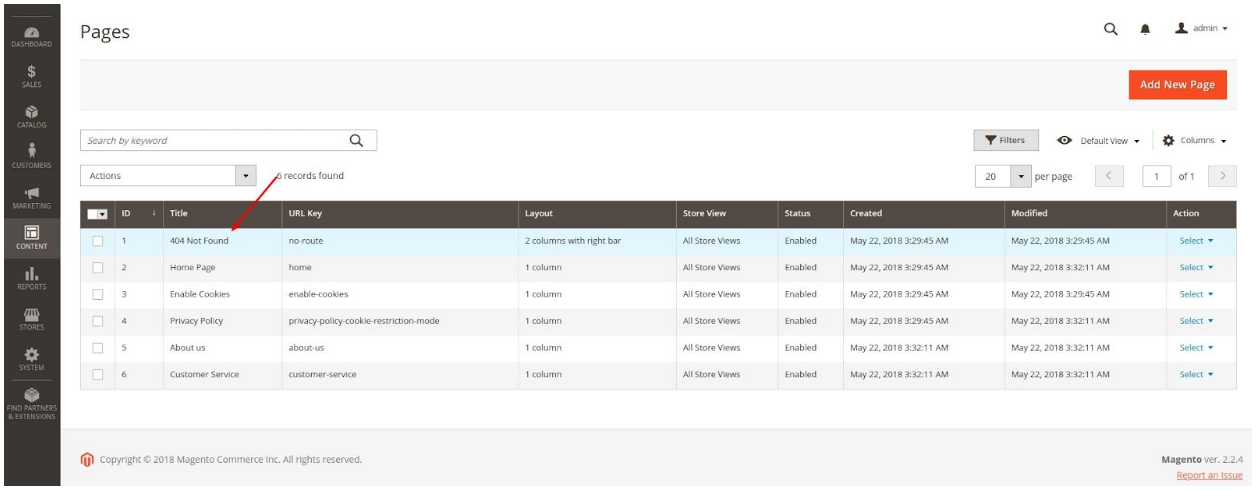
Base URLs (Secure)

Default Pages

Default Web URL [store view]	cms	✓ Use system value
CMS Home Page [store view]	Home Page	✓ Use system value
Default No-route URL [store view]	cms/noroute/index	✓ Use system value
CMS No Route Page [store view]	404 Not Found	Use system value
CMS No Cookies Page [store view]	Enable Cookies	✓ Use system value
Show Breadcrumbs for CMS Pages [store view]	Yes	✓ Use system value

Default Cookie Settings

Navigate to Content -> Pages to find the 404 Not Found page settings.



Pages

6 records found

ID	Title	URL Key	Layout	Store View	Status	Created	Modified	Action
1	404 Not Found	no-route	2 columns with right bar	All Store Views	Enabled	May 22, 2018 3:29:45 AM	May 22, 2018 3:29:45 AM	Select
2	Home Page	home	1 column	All Store Views	Enabled	May 22, 2018 3:29:45 AM	May 22, 2018 3:32:11 AM	Select
3	Enable Cookies	enable-cookies	1 column	All Store Views	Enabled	May 22, 2018 3:29:45 AM	May 22, 2018 3:29:45 AM	Select
4	Privacy Policy	privacy-policy-cookie-restriction-mode	1 column	All Store Views	Enabled	May 22, 2018 3:29:45 AM	May 22, 2018 3:32:11 AM	Select
5	About us	about-us	1 column	All Store Views	Enabled	May 22, 2018 3:32:11 AM	May 22, 2018 3:32:11 AM	Select
6	Customer Service	customer-service	1 column	All Store Views	Enabled	May 22, 2018 3:32:11 AM	May 22, 2018 3:32:11 AM	Select

Copyright © 2018 Magento Commerce Inc. All rights reserved.

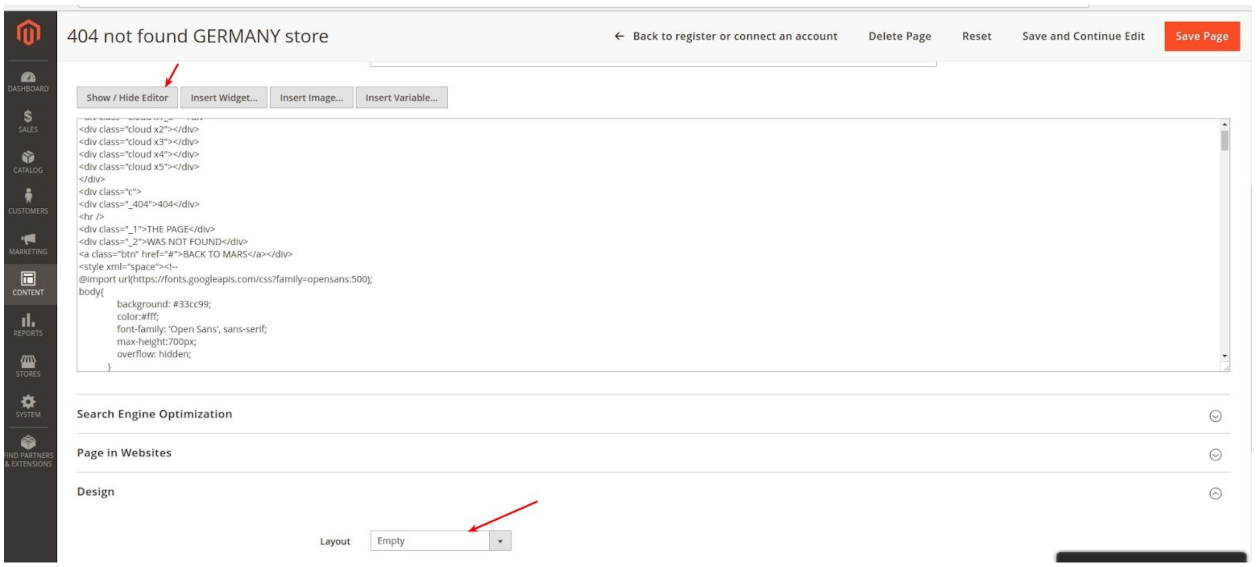
Magento ver. 2.2.4
Report an Issue

Open 404 page settings and modify the page the way you find necessary. You can also create a custom 404 page and set it up the No Route page settings. If you have a Magento multistore, a custom 404 page is a must, because for each store Not Found page contents will differ.

Create a new CMS page. When creating a new page, do not forget to specify Design -> Layout. If there is a need to insert html code into the page content, disable the wysiwyg editor. Empty Page Layout is used by default.

For this example, we used the following resource:

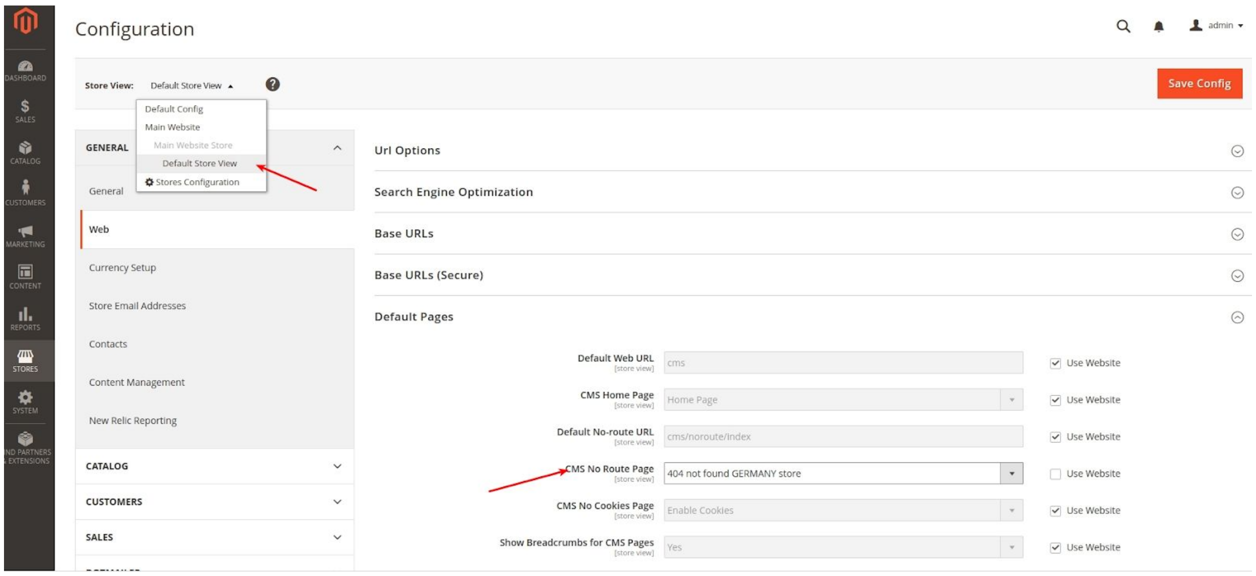
https://codepen.io/sqfreakz/pen/GJRJOY.
Then, copy the code into the CMS page content.


404 not found GERMANY store

```
<div class="cloud x2"></div>
<div class="cloud x3"></div>
<div class="cloud x4"></div>
<div class="cloud x5"></div>
</div>
<div class="c">
<div class="_404">404</div>
<hr />
<div class="_1">THE PAGE</div>
<div class="_2">WAS NOT FOUND</div>
<a class="btn" href="#">BACK TO MARS</a></div>
<style xml="space"><!--
@import url(https://fonts.googleapis.com/css?family=opensans:500);
body{
          background: #33cc99;
          color:#fff;
          font-family: 'Open Sans', sans-serif;
          max-height:700px;
          overflow: hidden;
      }
```

Layout: Empty

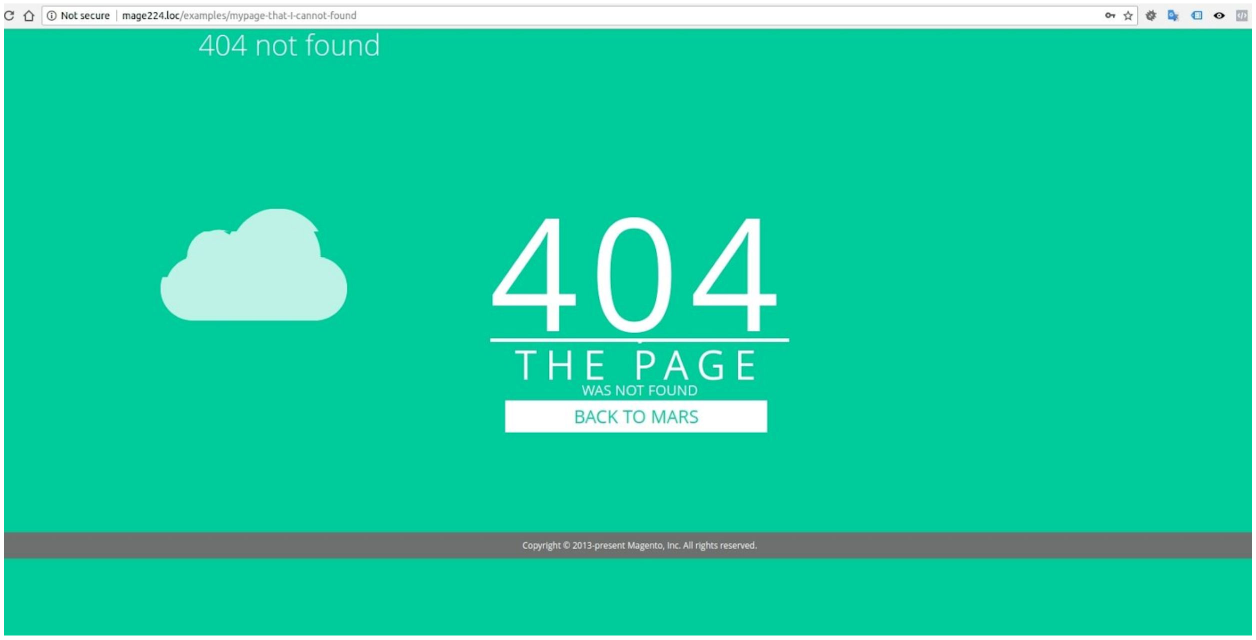
Afterward, switch to the needed store in the No Route Rage settings and select a new CMS page.


Configuration

Store View: Default Store View

Default Web URL [store view]	cms	✓ Use Website
CMS Home Page [store view]	Home Page	✓ Use Website
Default No-route URL [store view]	cms/noroute/index	✓ Use Website
CMS No Route Page [store view]	404 not found GERMANY store	Use Website
CMS No Cookies Page [store view]	Enable Cookies	✓ Use Website
Show Breadcrumbs for CMS Pages [store view]	Yes	✓ Use Website

Press Save Config button. As a result, you will get the following custom 404 page.


Not secure | mage224.loc/examples/mypage-that-I-cannot-found
404 not found
404
THE PAGE
WAS NOT FOUND
BACK TO MARS
Copyright © 2013-present Magento, Inc. All rights reserved.

How to create a custom noRoute handler

Magento also allows to create a custom noRoute handler. Let us make an example of creating an alternative to 404 error page. This will be a noRoute page redirect to the search page, where request path will serve as a search query.

To add a new “noRoute” handler, add to the etc/frontend/di.xml of your module the following:

```
<type name="Magento\Framework\App\Router\NoRouteHandlerList">
   <arguments>
       <argument name="handlerClassesList" xsi:type="array">
           <item name="custombvg" xsi:type="array">
               <item name="class"
xsi:type="string">BelVG\CustomNoRoute\App\Router\NoRouteHandler</item
>
               <item name="sortOrder" xsi:type="string">80</item>
           </item>
       </argument>
```

```
    </arguments>
</type>
```

Afterward, create a custom noRoute handler and add the logic that would execute the redirect.

2.4 Determine the layout initialization process

Determine how layout is compiled. How would you debug your layout.xml files and verify that the right layout instructions are used?

1. \Magento\Framework\View\Layout::build() is called
2. \Magento\Framework\View\Layout\Builder::build() is called
3. \Magento\Framework\View\Model\Layout\Merge::load() is called
4. Handles from the var $handles to the protected field $handles are added.
5. Layout for the current handles is loaded from the cache. In case the layout is not in the cache, then it is generated for each handle the following way, united and stored in cache:
    a. Layout for all handles of the current theme is loaded from cache. In case there is no layout, it is generated the following way and then stored in cache:
        i. A physical theme, based on the current theme, is loaded. Physical is the theme that has a designated folder and which is loaded via registration.php. There are TYPE_PHYSICAL=0, TYPE_VIRTUAL=1, TYPE_STAGING=2 types. Types are stored in **type** column in the **theme** table at the database.
        ii. The search of all *.xml files in the folders layout, page_layout in all enabled modules and current themes (current themes = current theme + all parent themes of it) is executed.

        iii. Each found file is loaded
            1. {{baseUrl}} to current base url, {{baseSecureUrl}} to current base secure url is substituted
            2. Simplexml_load_string is called with the element class \Magento\Framework\View\Layout\Element.
            3. A new xml tag (handle or layout) is added into $layoutStr. The tag's id attribute is also a filename without ".xml" suffix. It contains attributes and the contents of the xml object's core element, loaded in the previous step.
        iv. A new xml object, consisting of $layoutStr, is created.
    b. From this xml only necessary handles are loaded.
    c. Additional handles from the database are loaded.
6. Layout for the current handles is returned.
7. Layout xml object is generated.
8. Layout structure of elements from the loaded XML configuration is generated.

In order to view the merged layout for the current page, temporarily add execute method into controller action before the return element the following code (at the condition that result page is contained in the $resultPage variable):

```
header('Content-Type: text/xml');
$layoutString = $resultPage->getLayout()->getXmlString();
echo '<layouts
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">' .
$layoutString . '</layouts>';
die;
```

This will put out the xml contents of the merged layout directly into the browser.

This can also be achieved using observer.
File <module_dir>/etc/frontend/events.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:Event/etc/events
.xsd">
    <event name="layout_generate_blocks_after">
```

```
        <observer name="vendor_module_layout_generate_blocks"
instance="Vendor\Module\Observer\LayoutGenerateBlockObserver" />
    </event>
</config>
```

File <module_dir>/Observer/LayoutGenerateBlockObserver.php:

```
<?php
namespace Vendor\Module\Observer;

use Magento\Framework\Event\Observer;
use Magento\Framework\Event\ObserverInterface;

class LayoutGenerateBlockObserver implements ObserverInterface
{
    public function execute(Observer $observer)
    {
        header('Content-Type: text/xml');
        $layoutString =
$observer->getEvent()->getLayout()->getXmlString();
        echo '<layouts
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">' .
$layoutString . '</layouts>';
        die;
    }
}
```

Determine how HTML output is rendered.

For layout rendering, \Magento\Framework\View\Layout::getOutput() is called. This method recursively renders child elements.

How does Magento flush output, and what mechanisms exist to access and customize output?

Magento flushes html output the following way:

1. Controller action returns certain $result
2. If result is instance of \Magento\Framework\App\Response\HttpInterface, then the result is returned to the browser
3. In case result is instance of \Magento\Framework\Controller\ResultInterface, then:
    a. $result->renderResult($response) is called, which calls $result->applyHttpHeaders($response) and $result->render($response)
4. Then, $response->sendResponse() method is called. The method sends headers and body to the browser.

There can be different types of $result; for instance:
1. \Magento\Framework\Controller\Result\Raw
2. \Magento\Framework\Controller\Result\Json
3. \Magento\Framework\Controller\Result\Redirect
4. \Magento\Framework\View\Result\Layout
5. \Magento\Framework\View\Result\Page

Each type realizes the logic of headers and body generation for sending to the browser.

Let us consider \Magento\Framework\View\Result\Page.
Render method:

```
protected function render(HttpResponseInterface $response)
{
    $this->pageConfig->publicBuild();
    if ($this->getPageLayout()) {
        $config = $this->getConfig();
        $this->addDefaultBodyClasses();
        $addBlock = $this->getLayout()->getBlock('head.additional');
// todo
        $requireJs = $this->getLayout()->getBlock('require.js');
        $this->assign([
            'requireJs' => $requireJs ? $requireJs->toHtml() : null,
            'headContent' =>
$this->pageConfigRenderer->renderHeadContent(),
            'headAdditional' => $addBlock ? $addBlock->toHtml() :
null,
            'htmlAttributes' =>
$this->pageConfigRenderer->renderElementAttributes($config::ELEMENT_T
YPE_HTML),
            'headAttributes' =>
$this->pageConfigRenderer->renderElementAttributes($config::ELEMENT_T
YPE_HEAD),
            'bodyAttributes' =>
$this->pageConfigRenderer->renderElementAttributes($config::ELEMENT_T
YPE_BODY),
            'loaderIcon' =>
$this->getViewFileUrl('images/loader-2.gif'),
        ]);

        $output = $this->getLayout()->getOutput();
        $this->assign('layoutContent', $output);
        $output = $this->renderPage();
        $this->translateInline->processResponseBody($output);
        $response->appendBody($output);
    } else {
        parent::render($response);
    }
    return $this;
}
```

renderPage method loads the Magento_Theme::root.phtml template that we will explore in the next paragraph.

At the end, the $output we get is added into $response body.

To customize the output, you can:

1. Modify layout xml files.
2. Override phtml templates.
3. Add layout handle to \Magento\Framework\View\Result\Page.
4. Override toHtml or _toHtml method of the block (_toHtml is recommended).
5. Create a custom result class that extends \Magento\Framework\View\Result\Page and apply it in controller actions.

6. Call $result->renderResult($response) in execute method of controller action class, customize $response body and execute return $response.

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

Layout instructions:

- <block> - for adding a new block
- <container> - for adding a new container
- before and after attributes - for specifying the position of a block or container. You may use <block>, <container>, <move> in the blocks.
- <action> - [deprecated] calls block method during the block generation.
- <referenceBlock> and <referenceContainer> - for modifying or deleting the current block and the container correspondingly.
- <move> - for altering the parent or the position of a block or a container.
  <remove> - is used only to remove the static resources linked in a page <head> section.
- <update> - is used to include a certain layout file.
- <argument> - sets arguments for blocks.

block vs. container

- Blocks represents the end of the chain in rendering HTML for Magento.
- Containers contain blocks and can wrap them in an HTML tag.
- Containers do not render any output if there are no children assigned to them.

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

Layout fallback allows to search layout files in all enabled modules and current themes (current themes = current theme + all parent themes of it). In order to perform customization, create a layout file with handle.xml name (handle is layout handle name) in one of the folders where Magento searches for layout files.

For debugging, put out the merged layout for the current page by following the instructions described above.

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

In order to determine which layouts are used for the current page, add the following code into the controller action into the execute method before return (under the condition that result page is contained in the $resultPage variable):

```
var_dump($resultPage->getLayout()->getUpdate()->getHandles());
die;
```

Add ".xml" to the received lines of code and this will be the layout files' names

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

Magento merges the layout xml files into a single one. However, this merge is different from config xml files. In config xml files, xml are merged according to tag names and $idAttributes, while in layout the tags do not merge. Merged layout stores the file contents of all layout modules, the current theme and its parent themes; therefore, merged layout will contain many <body> tags.

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

1. layout area is changed for adminhtml
2. Layout of aclResource tag support is added, allowing to connect the resource to a block and hide the block is user does not have access to the resource.
3. Default block class is changed for Magento\Backend\Block\Template

Section 3: Customizing the Magento UI

3.1 Demonstrate ability to utilize themes and the template structure

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

In case there is a need to modify the web store layout, you can create a new theme to achieve this. To learn how to create a Magento theme, follow the link https://belvg.com/blog/magento-2-creating-a-new-theme.html

How do you define theme hierarchy for your project?

To define theme hierarchy for your project, specify the parent theme in theme.xml file inside the theme:

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

Demonstrate the ability to customize/debug templates using the template fallback process.

In order to customize a certain view file, override it in the current theme. Template fallback process is when Magento 2 searches for a file in the child theme, then in the parent themes and finally in modules. To learn more about this, follow the link: https://devdocs.magento.com/guides/v2.3/frontend-dev-guide/themes/theme-inherit.html

There are the following methods of template debugging:

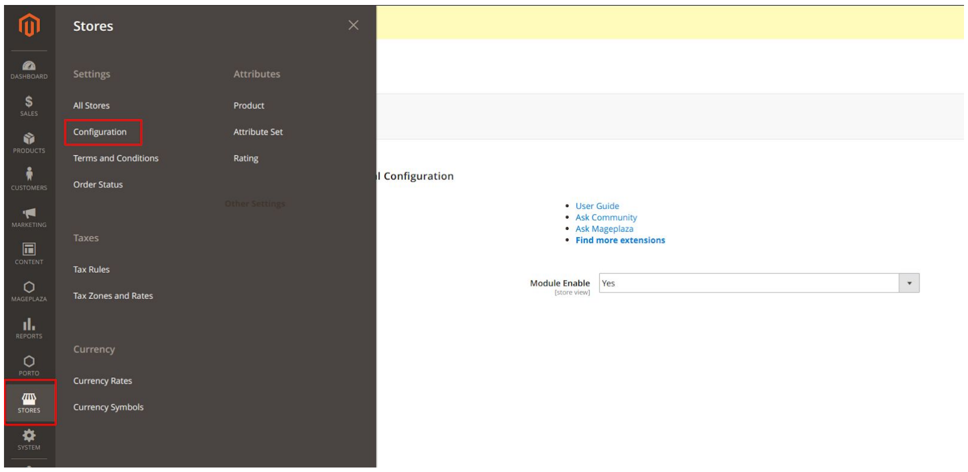
- XDebug (The main peculiarity of this method are breakpoints as well as the ability to view and modify variables at any time)
- Developer mode (the mode for development, displays errors directly on screen)
- Logs in folder var/logs and reports in folder var/reports
- Display Magento reports in browser
- Template path hints (displays templates' paths and names)

To learn more about debugging methods, follow the link (https://belvg.com/blog/developer-mode-and-debugging-in-magento-2.html)

How do you identify which exact theme file is used in different situations?

To easier identify what template file is used in a certain place at the website, enable path hints for it.

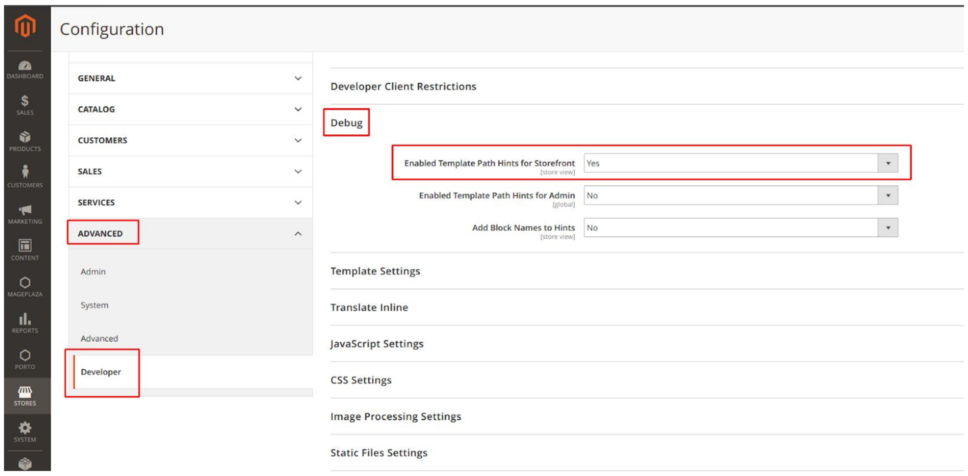
First, log in to the admin panel and navigate to "Stores -> Configuration".



Module Enable [store view]: Yes

(http://prntscr.com/k6d4s6)

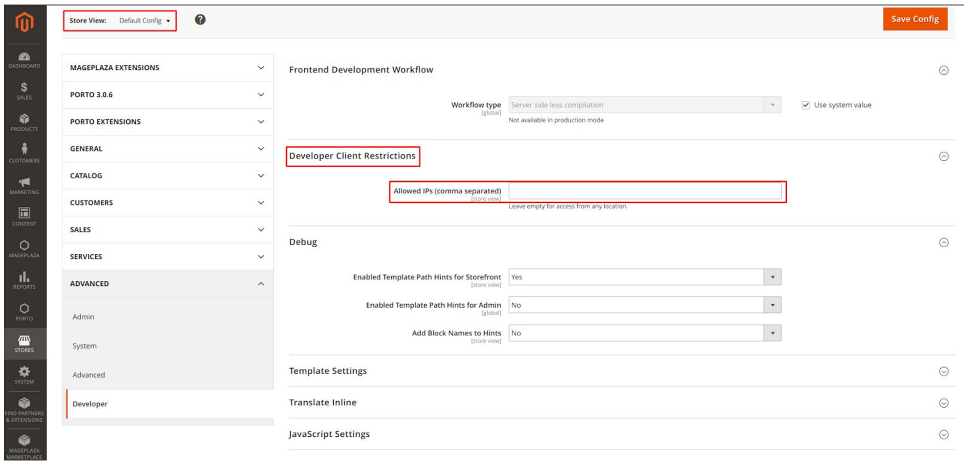
Then, go to "Advanced -> Developer". Expand the "Debug" tab and set "Enabled Template Path Hints for Storefront" attributes at Yes.



Enabled Template Path Hints for Storefront [store view]: Yes
Enabled Template Path Hints for Admin [global]: No
Add Block Names to Hints [store view]: No

(https://prnt.sc/k6d5cg)

In case you have multiple stores, set in the upper right menu the one for which you enable path hints.
When there is a need to enable hints for a certain ip-address, expand the "Developer Client Restrictions" tab and enter the IP-address into the "Allowed IPs (comma separated)" field. If you need to enter several IP-addresses, separate them with commas.



Frontend Development Workflow

Workflow type [global]: Server side less compilation — Use system value
Not available in production mode

Developer Client Restrictions

Allowed IPs (comma separated) [store view]:
Leave empty for access from any location.

Debug

Enabled Template Path Hints for Storefront [store view]: Yes
Enabled Template Path Hints for Admin [global]: No
Add Block Names to Hints [store view]: No

Template Settings

Translate Inline

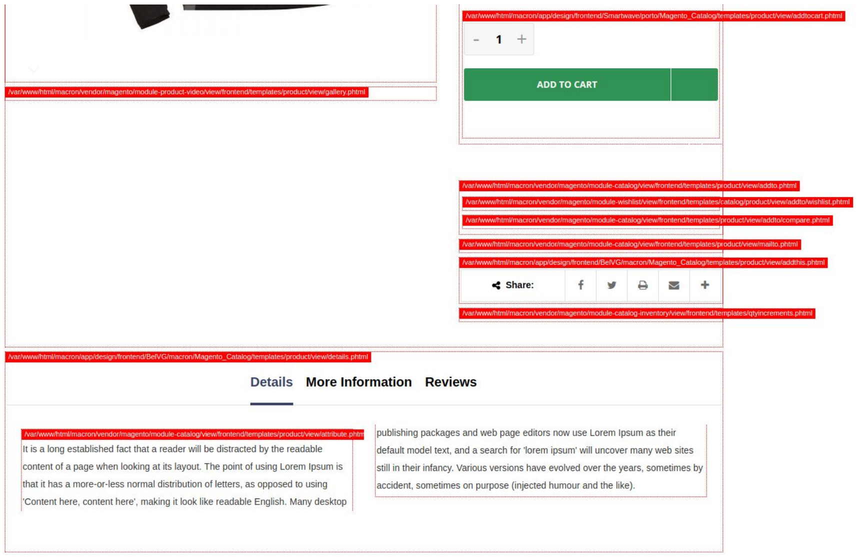
JavaScript Settings

(http://prntscr.com/k6d821)

Another way to enable or disable path hints is via CLI:

- bin/magento dev:template-hints:enable
- bin/magento dev:template-hints:disable

As you have completed the course of actions described above, path hints will appear at the front end of the store.



```
/var/www/html/macron/app/design/frontend/Smartwave/porto/Magento_Catalog/templates/product/view/addtocart.phtml
/var/www/html/macron/vendor/magento/module-product-video/view/frontend/templates/product/view/gallery.phtml
/var/www/html/macron/vendor/magento/module-catalog/view/frontend/templates/product/view/addto.phtml
/var/www/html/macron/vendor/magento/module-wishlist/view/frontend/templates/catalog/product/view/addto/wishlist.phtml
/var/www/html/macron/vendor/magento/module-catalog/view/frontend/templates/product/view/addto/compare.phtml
/var/www/html/macron/vendor/magento/module-catalog/view/frontend/templates/product/view/mailto.phtml
/var/www/html/macron/app/design/frontend/BelVG/macron/Magento_Catalog/templates/product/view/addthis.phtml
/var/www/html/macron/vendor/magento/module-catalog-inventory/view/frontend/templates/qtyincrements.phtml
/var/www/html/macron/app/design/frontend/BelVG/macron/Magento_Catalog/templates/product/view/details.phtml
/var/www/html/macron/vendor/magento/module-catalog/view/frontend/templates/product/view/attribute.phtm
```

It is a long established fact that a reader will be distracted by the readable content of a page when looking at its layout. The point of using Lorem Ipsum is that it has a more-or-less normal distribution of letters, as opposed to using 'Content here, content here', making it look like readable English. Many desktop publishing packages and web page editors now use Lorem Ipsum as their default model text, and a search for 'lorem ipsum' will uncover many web sites still in their infancy. Various versions have evolved over the years, sometimes by accident, sometimes on purpose (injected humour and the like).

(http://prntscr.com/k6d8su)

Following the highlighted paths, we can easily find the needed templates files.

If you know the needed file's unique line (for example, class name, attributes, etc.), then you can search for it among the project files. In PhpStorm, find the project files folder, click the right mouse button, select "Find in Path…" and insert the unique line in the appeared window.
Another way to search for files is using grep command: **grep -R "<div class=\"product-item-inner\">" vendor**

How can you override native files?

In order to override the parent theme files, put the file in the current theme folder at the same path it was located in the parent theme.

Example:
If you need to override a file in the parent theme `<parent_theme_dir>/Magento_Catalog/templates/product/view/gallery.phtml`, then create a file in the child theme `<child_theme_dir>/Magento_Catalog/templates/product/view/gallery.phtml`.

In order to override module's view files, create them at the same path as they were located in the module's view folder, but deleting <area> from the path and adding the module name. Example:

If we want to override the module file vendor/magento/module-catalog/view/**frontend**/templates/product/view/gallery.phtml, create a file in the theme
<child_theme_dir>/**Magento_Catalog**/templates/product/view/gallery.phtml

This method overrides templates files and all the files in web folder. However, if you try to override the layout file the following way, instead of overriding the parent theme file will be added up to. To learn more about the layout files override in Magento 2, follow the link
(https://belvg.com/blog/override-a-layout-in-magento-2.html)

3.2 Determine how to use blocks

Demonstrate an understanding of block architecture and its use in development.

Blocks are PHP classes that provide data for template files (.phtml) and implement rendering. Also, blocks can cache the results of template rendering.

Which objects are accessible from the block?

All blocks extend \Magento\Framework\View\Element\Template, so by default blocks are injected with a \Magento\Framework\View\ Element\Template\Context object and support getters and setters for returning and storage of any objects or data. This object contains a number of other objects that are loaded into the instance:

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

What is the typical block's role?

Separation of blocks and templatesallows you to divide design related logic and design. Blocks are usually, but not always, connected to PHTML template files. Blocks can be thought of as data containers for a template, which represents a piece of the HTML on the page. In layouts XML, you can manage blocks on page and add new ones, set templates to move and delete.

Identify the stages in lifecycle of a block

The block life cycle consists of two phases: generating blocks and rendering blocks. Blocks are instantiated at the moment the layout is created. They are not executed at

that time, just instantiated. Also during this phase, the structure is built, the children of blocks are set, and for each block, the prepareLayout() method is called.
However, rendering occurs in the later rendering phase.
Generating phase:

1. Magento\Framework\View\Page\Config::publicBuild()
2. Magento\Framework\View\Page\Config::build()
3. Magento\Framework\View\Layout\Builder::build()
4. Magento\Framework\View\Layout\Builder::generateLayoutBlocks()
5. Magento\Framework\View\Layout::generateElements()
6. Magento\Framework\View\Layout\GeneratorPool::process()

GeneratePool goes through all generators and generates all scheduled elements. It has generators with the following elements:
Magento\Framework\View\Page\Config\Generator\Head
Magento\Framework\View\Page\Config\Generator\Body
Magento\Framework\View\Layout\Generator\Block
Magento\Framework\View\Layout\Generator\Container
Magento\Framework\View\Layout\Generator\UiComponent
Rendering phase:

1. Magento\Framework\View\Result/Page::render()
2. Magento\Framework\View\Layout::getOutput()

At this moment, we already have all the layout xml for the generated page, and all block classes are created.

3. Magento\Framework\View\Layout::renderElement()
4. Magento\Framework\View\Layout::renderNonCachedElement()

In this method, we check the type of rendered elements. It creates them and returns html using toHtml() method. This method is not recommended to override. If you want to change block rendering, override _toHtml() method.

In what cases would you put your code in the _prepareLayout(), _beforeToHtml(), and _toHtml() methods?

`_prepareLayout()` - most often used for:

- adding child's `Magento\Eav\Block\Adminhtml\Attribute\Edit\Options\AbstractOptions::_prepareLayout()`
- adding tabs on backend `Magento\Backend\Block\System\Design\Edit\Tabs::_prepareLayout()`
- set title `Magento\Wishlist\Block\Customer\Sharing::_prepareLayout()`
- adding pager, breadcrumbs and so on `Magento\Sales\Block\Order\History::_prepareLayout()`
- set renderer `Magento\Catalog\Block\Adminhtml\Form::_prepareLayout()`

`_beforeToHtml()`
- data preparing `Magento\Catalog\Block\Product\ProductList\Related::_beforeToHtml()`
- assign values `Magento\Backend\Block\Widget\Form\Element::_beforeToHtml()`
- adding child's `Magento\Shipping\Block\Adminhtml\Create\Items::_beforeToHtml()`

`_toHtml()`
Using this method, you can manipulate block rendering, complement the condition, make wrappers for html, change the template for the block, etc. `Magento\GroupedProduct\Block\Order\Email\Items\Order\Grouped::_toHtml() , Magento\Sales\Block\Reorder\Sidebar::_toHtml()`

How would you use events fired in the abstract block?

view_block_abstract_to_html_before - use for editing params:
- templates
- caches
- grids

`view_block_abstract_to_html_after` - use for html manipulation - editing, adding n conditions, wrappers

Describe how blocks are rendered and cached

The most important method for rendering a block is Magento\Framework\View\Element\AbstractBlock::toHtml(). First, it runs _loadCache(), and if the cache is missing, then it run _beforeToHtml() after the block is rendered by the method_toHtml(). Afterward, the cache is saved _saveCache($html) and run _afterToHtml($html).

Method _loadCache() uses cache_key return by getCacheKey() and _saveCache($html) - cache_tags obtained by the method getCacheTags(). Cache_key each block is added up like **BLOCK_CLASS::CACHE_KEY_PREFIX. $cache_key** , if this property not defined - **BLOCK_CLASS::CACHE_KEY_PREFIX . sha1(implode('|', $this->getCacheKeyInfo()))**. Cache_tags is an array and consists of a property $cache_tags, if it defined in block and if block instance of Magento\Framework\DataObject\IdentityInterface values returned by the method getIdentities() are added to the array. We can manage $cache_lifetime variable. Value will be in seconds, if you want to disable cache, you can set value to 0 or not set the value at all, because all blocks are non-cacheable by default.

Identify the uses of different types of blocks.

- `Magento\Framework\View\Element\AbstractBlock` - parent block for all custom blocks
- `Magento\Framework\View\Element\Template` - block with template
- `Magento\Framework\View\Element\Text` - just rendering text
- `Magento\Framework\View\Element\FormKey` - return hidden input with form key
- `Magento\Framework\View\Element\Messages` - rendering notification message by type

When would you use non-template block types?

Applying non-template block types is wise when we use simple renderers. Another reason for using such block types is when block content is dynamic generated or stored in database or in containers. For example, if you want form_key in template, you can insert block Magento\Framework\View\Element\FormKey.

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

Using a template block or other block types is recommended when complicated renderers are used, or when you want to customize themes or add new blocks. A prime example is a breadcrumbs block Magento\Catalog\Block\Breadcrumbs. If you want to customize breadcrumbs, you can set template in layout.

3.3 Demonstrate ability to use layout and XML schema

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

Magento layouts contain instructions that allow to:

- Add and delete static resources (JavaScript, CSS, fonts) into the head section of the page;
- Create and modify containers;
- Create and modify blocks;
- Set the templates for blocks;
- Pass arguments into blocks;
- Change the location of the elements (blocks and containers);
- Change elements' order (blocks and containers);
- Delete elements (blocks and containers).

Let us examine the cases of each instruction application:

Adding and deleting static resources

To add static resources to the page, first create default_head_blocks.xml file at the following path: `<theme_dir>/Magento_Theme/layout/default_head_blocks.xml.`

Add a CSS file with the following instruction:

```
<css src="css/my-styles.css"/>
```

To add a locally located JavaScript file, use one of the following instructions:

```
<script src="Magento_Catalog::js/sample1.js"/>
<link src="js/sample.js"/>
```

In order to connect an external resource, add src_type attribute with “url” value. For example:

```
<css
src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.4/css/bootstrap-th
eme.min.css" src_type="url" />
<script
src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.4/js/bootstrap.min
.js" src_type="url" />
```

To add a font, apply link tag and add attributes rel="stylesheet" and type="text/css":

```
<link rel="stylesheet" type="text/css"
src="http://fonts.googleapis.com/css?family=Montserrat"
src_type="url" />
```

Adding or modifying containers

To add a new container, use the following instruction:

```
<container name="you.container" as="youContainer" label="My
```

```
Container" htmlTag="div" htmlClass="my-container" />
```

To modify a container (to add a block, for example), apply referenceContainer instruction:

```
<referenceContainer name="header.panel">
    <block class="YouVendor\YouModule\Block\YouBlock"
name="new.block" />
</referenceContainer>
```

Adding or modifying blocks. Modifying block template and parameters

To create a block, apply block instruction. For example:

```
<block class="Magento\Catalog\Block\Product\View\Description"
name="product.info.sku" template="product/view/attribute.phtml"
after="product.info.type" />
```

To modify a block, apply referenceBlock instruction. Let us use the following block as an example:

```
<block class="Namespace_Module_Block_Type" name="block.example">
    <arguments>
        <argument name="label" xsi:type="string">Block
Label</argument>
    </arguments>
</block>
```

To modify its argument and add a new one, apply the following instruction:

```
<referenceBlock name="block.example">
    <arguments>
        <!-- Modified argument -->
        <argument name="label" xsi:type="string">New Block
Label</argument>
        <!-- New argument -->
        <argument name="custom_label" xsi:type="string">Custom Block
Label</argument>
    </arguments>
</referenceBlock>
```

In case you need to set a template for a block, there are two ways to do this.

Method 1: using **template** attribute:

```
 <referenceBlock name="new.template"
template="Your_Module::new_template.phtml"/>
```

Method 2: using an argument with a name **template**:

```
<referenceBlock name="new.template">
     <arguments>
         <argument name="template"
xsi:type="string">Your_Module::new_template.phtml</argument>
     </arguments>
 </referenceBlock>
```

In our examples, new_template.phtml is the path to the template file, relatively to template folder (<module_dir>/view/<area>/templates or <theme_dir>/<Vendor_Module>/templates).

It should be noted that the templates, set with template attribute, have a higher priority. Therefore, the value in the template attribute will rewrite the one, specified with the help of argument.

Changing of the order or the location of blocks and containers. Deleting blocks and containers

Before and after attributes allow to change the element order inside the parent. Set the name of the element, before or after which we want to place the needed element, as the attributes' values. Using move instruction, we can not only change the element's order, but also modify the parent's order. For example, the following construction will relocate the block named product.info.review into the container with the name product.info.main, and will place it before the container named product.info.price:

<move element="product.info.review" destination="product.info.main" before="product.info.price"/>

To delete a block or container, use the remove attribute of the referenceBlock or referenceContainer instructions correspondingly. For instance, to delete a catalog.compare.sidebar block, apply the following instruction:

<referenceBlock name="catalog.compare.sidebar" remove="true" />

Describe how to create a new layout XML file.

To create a layout file, you first need to create an XML file with the name layout handle at one of the following paths:
<module_dir>/view/<area>/layout/<layout-handle>.xml or
<theme_dir>/<Vendor>_<Module>/layout/<layout-handle>.xml with the contents:

```
<?xml version="1.0"?>
<page xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:View/Layout/etc/
page_configuration.xsd">
    ...
</page>
```

Describe how to pass variables from layout to block.

To pass the variables' values from layout to block, apply <arguments> instruction inside the block. For instance, if we want to pass cache_lifetime:

```
<referenceBlock name="my_block_name">
```

```
    <arguments>
        <argument name="cache_lifetime"
xsi:type="string">3600</argument>
    </arguments>
</referenceBlock>
```

How to modify existing layout files

Magento 2 has a number of instructions, allowing to modify layout in nearly any way. But first, we need to find out how to modify a layout for a particular page. This can be done in configuration XML file of the page, using layout attribute of the page root node. For instance, to modify page Advanced Search layout from the default 1-column to 2-column with left bar, place at the following path <theme_dir>/Magento_CatalogSearch/layout/catalogsearch_advanced_index.xml a file with the such contents:

```
<page layout="2columns-left"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:View/Layout/etc/
page_configuration.xsd">
...
</page>
```

3.4 Utilize JavaScript in Magento

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

Javascript module is a separate *.js file that can import other require.js modules. It has the similar contents:
// File (<module_dir>/view/frontend/web/js/script.js)

```
define(['jquery'], function ($)
    return function(...){
        ...
    };
});
// Other module can look the following way to use the first module:
define([
    'jquery',
    'Vendor_Module/js/script'
], function ($, myModule)
    return function(...){
        ...
        myModule(...);
        ...
    };
});
```

Plain modules

Plain module is a common module that is not inherited from others. You will find an example of a plain module above (Vendor_Module/js/script). In the examples below we will use modules that are inherited with jQuery.widget, uiElement.extend functions. This type of modules should be used when instances of the module are not connected to any element.

jQuery UI widgets

jQuery UI widget allows to create a custom jQuery UI widget with a custom handler. This type of module should be used when instances of the module are connected to certain elements that are not UI components.

Let us examine this case using the following file: vendor/magento/module-multishipping/view/frontend/web/js/payment.js:

```
define([
    'jquery',
    'mage/template',
    'Magento_Ui/js/modal/alert',
    'jquery/ui',
    'mage/translate'
], function ($, mageTemplate, alert) {
    'use strict';

    $.widget('mage.payment', {
        options: {
            ...
        },

        _create: function () {
            ...
        },
    });

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

UiComponents

UiComponents allow to create a custom widget with a custom handler, inherited from uiElement. This type of module is recommended to use when instances of the module are connected to UI components.

Let us examine this case using the following file:
vendor/magento/module-catalog/view/frontend/web/js/storage-manager.js:

```
define([
    'underscore',
    'uiElement',
    'mageUtils',
    'Magento_Catalog/js/product/storage/storage-service',
    'Magento_Customer/js/section-config',
    'jquery'
], function (_, Element, utils, storage, sectionConfig, $) {
    'use strict';

    ...

    return Element.extend({
        defaults: {
            ...
        },

        initialize: function () {
            ...

            return this;
        },
    });
});
```

Describe UI components. In which situation would you use UiComponent versus a regular JavaScript module?

UI components are realized using XML configuration + PHP modifiers and data providers + HTML knockout templates + JS module, based on uiElement + CSS/LESS/SASS styles. They allow to simplify development and significantly decrease code duplication.

Out-of-the-box UI components are heavily used in adminhtml, while at the frontend - only in checkout (the UI components are added into checkout via layout). If JS module is a component inside checkout or in form or grid in adminhtml, then apply UI components. Otherwise, use regular JS module.

Describe the use of requirejs-config.js, x-magento-init, and data-mage-init.

Requirejs-config.js files (located at the <module_dir>/view/<area>/requirejs-config.js and <theme_dir>/requirejs-config.js paths) contain the configuration for RequireJS. To pass the configuration to browser, Magento unites all requirejs-config.js files into one. To learn more about the configuration options for RequireJS, follow the link: https://requirejs.org/docs/api.html#config.

To insert a JS component into a PHTML template, use one of the following variants:

- using the data-mage-init attribute
- using the <script type="text/x-magento-init" /> tag

If you need to initialize JS module without the connection to HTML Element, use `<script type="text/x-magento-init">` and enter `"*"` symbol into the selector field.

```
<script type="text/x-magento-init">
{
    "*": {
        "Vendor_Module/js/myfile": {"parameter":"value"}
    }
}
</script>
```

In case you need to initialize JS module with connection to HTML Element, there are two ways to do this:

1) Use data-mage-init attribute of the element, to which the component should be connected. For example:

```
<div id="element-id"
data-mage-init='{"Vendor_Module/js/myfile":{"parameter":"value"}}'></
div>
```

2) Use <script type="text/x-magento-init">, specifying selector for elements. For each element, a new instance of javascript module will be created. For example:

```
<script type="text/x-magento-init">
{
    "#element-id": {
        "Vendor_Module/js/myfile": {}
    }
}
</script>
```

Initialization in PHTML template is commonly used to pass parameters from php to javascript module. But we can also initialize javascript module in another javascript file.

Example:

```
vendor/magento/module-swatches/view/adminhtml/web/js/visual.js
...
$('[data-role=swatch-visual-options-container]').sortable({
   distance: 8,
   tolerance: 'pointer',
   cancel: 'input, button',
   axis: 'y',
   ....
   }
});
...
```

4.1 Demonstrate ability to use data-related classes

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

Repository can be applied for working with collections in Magento 2. It realizes the Repository pattern that allows to work with collections and entities regardless of their storage place (storing or caching are the implementation details). The pattern itself is located between Domain and Application Service Layer.
In Magento 2, five basic repository functions are realized. They are save, getById, getList, delete, deleteById. Yet, each Repository realization in Magento has a custom interface and the functions are not always implemented in it. For example, in Magento\Quote\Api\GuestCartTotalRepositoryInterface, only get($cartId) method is realized. Therefore, it is recommended to pay attention to a certain Repository class implementation. Let us examine the Repository case using the \Magento\Catalog\Api\ProductRepositoryInterface example and its realization \Magento\Catalog\Model\ProductRepository.

```
<?php
namespace Magento\Catalog\Api;
interface ProductRepositoryInterface
{
    public function save(\Magento\Catalog\Api\Data\ProductInterface $product,
$saveOptions = false);
    public function get($sku, $editMode = false, $storeId = null, $forceReload =
false);
    public function getById($productId, $editMode = false, $storeId = null,
$forceReload = false);
    public function delete(\Magento\Catalog\Api\Data\ProductInterface $product);
    public function deleteById($sku);
    public function getList(\Magento\Framework\Api\SearchCriteriaInterface
$searchCriteria);
}
```

Here, all necessary five functions for working with Repository are realized. In this case, the main class for working with product is \Magento\Catalog\Api\Data\ProductInterface, allowing to save all the products inherited from this class, regardless of their type.

Let us consider the example of getById method realization:

```
<?php
...
protected $productRepository;

      public function __construct(
            ...
            \Magento\Catalog\Api\ProductRepositoryInterface
$productRepository)
      {
            $this->productRepository = $productRepository;
      }
      ...
public function someMethod() {
      $product = $this->productRepository->getById(1);
      return $product->getSku();
}
...
...
public function someDeleteMethod() {
      $this->productRepository->deleteById(1);
}
...
```

In this case, we realize working with ProductRepository by extracting the object from id 1 and returning it to SKU. deleteById method will be realized similarly. In this constructor, we create \Magento\Catalog\Api\ProductRepositoryInterface, not

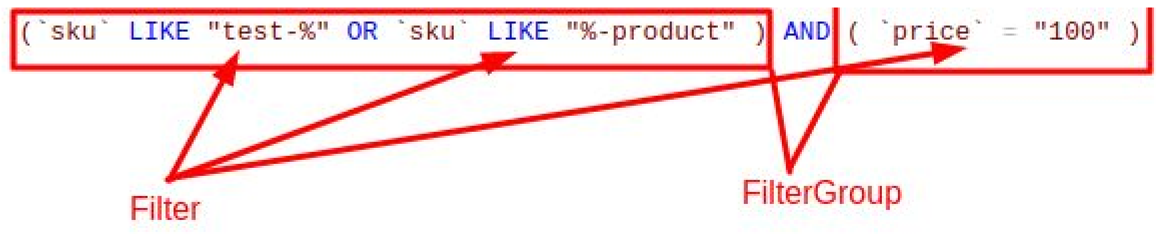
\Magento\Catalog\Model\ProductRepository itself. The dependencies are described in di.xml file. For example, the dependency for the current class is described in vendor/magento/module-catalog/etc/di.xml and looks the following way:
Let us also consider save method realization:

```
...
public function someMethod() {
      $product = $this->productRepository->getById(1);
      $product->setSku('test-sku-1');
      $this->productRepository->save($product);
}
...
...
public function someDeleteMethod() {
      $product = $this->productRepository->getById(1);
      $this->productRepository->delete($product);
}
...
```

We modifies SKU of the project and saved it using save repository method. Delete method deletes the product.

How do you configure and create a SearchCriteria instance using the builder?

In this context, getList method is realized with SearchCriteriaInterface. The interface realizes conditions for requests (for example, it is where in MySQL query). All conditions fall into two categories: Filter and FilterGroup.



```
( `sku` LIKE "test-%" OR `sku` LIKE "%-product" ) AND ( `price` = "100" )
```

We implement the conditions mentioned above using SearchCriteria:

```
…
public function someMethod() {
          $filter =
$this->objectManager->create('Magento\Framework\Api\Filter');
$filter->setField('sku');
$filter->setValue('test-%');
$filter->setConditionType('like');
$filter2 =
$this->objectManager->create('Magento\Framework\Api\Filter');
$filter2->setField('sku');
$filter2->setValue('%-product');
$filter2->setConditionType('like');
$filterGroup =
$this->objectManager->create('Magento\Framework\Api\Search\FilterGrou
p');
$filterGroup>setFilters([$filter, $filter2]);
$filter3 =
$this->objectManager->create('Magento\Framework\Api\Filter');
$filter3->setField('price');
$filter3->setValue('100');
$filter3->setConditionType('eq');
$filterGroup2 =
$this->objectManager->create('Magento\Framework\Api\Search\FilterGrou
p');
$filterGroup2>setFilters([$filter3]);
$searchCriteria- =
$this->objectManager->create('Magento\Framework\Api\SearchCriteriaInt
erface');
$searchCriteria->setFilterGroups([$filterGroup, $filterGroup2]);
$result = $this->productRepository>getList($searchCriteria-);
return $result->getItems();
     }
This code returns the objects array, inherited from
\Magento\Catalog\Api\Data\ProductInterface.
```

How do you use Data/Api classes?

Api in Magento 2 is commonly used to describe interfaces, further used in Model, Helper, Data and other Magento classes. Also, API interfaces can be applied for WebAPI requests (when they are described in webapi.xml). Api directory is located in modules roots, similarly to etc, Model and other directories.

Unlike Api, Api/Data directory contains interfaces for the data, for example, store data or customer data.

An excellent example for explaining the difference between Api and Api/Data is the implementation of \Magento\Catalog\Api\ProductRepositoryInterface and \Magento\Catalog\Api\Data\ProductInterface.
ProductRepository implements a get method that loads a Product object (using ProductFactory) that implements ProductInterface for working with data.

Describe how to create and register new entities.

In Magento 2, entities are unique objects that contain a number of various attributes and/or parameters. Products, orders, users, etc. are all examples of entities.

To create a new entity type, use \Magento\Eav\Setup\EavSetup class. Entity creation and registration is realized via installEntities($entities = null) method, specifying the entity settings as a parameter. For example:

```
...
$eavSetup>installEntities([
\Belvg\Test\Model\Test::ENTITY => [
                'entity_model' =>
Belvg\Test\Model\ResourceModel\Test',
                'table' => \Belvg\Test\Model\Test::ENTITY .
'_entity',
                'attributes' => [
                    'test_id' => [
                        'type' => 'static',
                    ],
                    'first_attribute' => [
                        'type' => 'static',
                    ],
                    'second_attribute' => [
                        'type' => 'static',
                    ],
                ],
            ]
      ]);
...
```

How do you add a new table to the database?

To add a new table, create a Setup script that realizes Magento\Framework\Setup\InstallSchemaInterface or Magento\Framework\Setup\UpgradeSchemaInterface interface, or realize install or upgrade method in it correspondingly. For example:

```
...
public function install(SchemaSetupInterface $setup,
ModuleContextInterface $context)
    {
        $setup->startSetup();
        $table = $setup->getConnection()->newTable(
            $setup->getTable('belvg_test')
        )->addColumn(
            'entity_id',
            \Magento\Framework\DB\Ddl\Table::TYPE_INTEGER,
            null,
            ['identity' => true, 'unsigned' => true, 'nullable' =>
false, 'primary' => true],
            'EntityID'
        )->addColumn(
            'first_attribute',
            \Magento\Framework\DB\Ddl\Table::TYPE_TEXT,
            64,
            [],
            'FirstAttribute'
        );
        $setup->getConnection()->createTable($table);
        $setup->endSetup();
    }
...
```

This part of the code creates a belvg_test table with entity_id and first_attribute fields.

Describe the entity load and save process.

To load and store entity in Magento\Eav\Model\Entity\AbstractEntity model, load and save methods are realized; they carry the logic of saving and loading from the database.

```
...
public function load($object, $entityId, $attributes = [])
   {
       \Magento\Framework\Profiler::start('EAV:load_entity');
       /**
        * Load object base row data
        */
       $object->beforeLoad($entityId);
       $select = $this->_getLoadRowSelect($object, $entityId);
       $row = $this->getConnection()->fetchRow($select);

       if (is_array($row)) {
           $object->addData($row);
           $this->loadAttributesForObject($attributes, $object);

           $this->_loadModelAttributes($object);
           $this->_afterLoad($object);
           $object->afterLoad();
           $object->setOrigData();
           $object->setHasDataChanges(false);
       } else {
           $object->isObjectNew(true);
       }
       \Magento\Framework\Profiler::stop('EAV:load_entity');
       return $this;
   }

...
public function save(\Magento\Framework\Model\AbstractModel $object)
   {
       /**
        * Direct deleted items to delete method
        */
       if ($object->isDeleted()) {
           return $this->delete($object);
       }
       if (!$object->hasDataChanges()) {
           return $this;
       }
       $this->beginTransaction();
       try {
           $object->validateBeforeSave();
           $object->beforeSave();
           if ($object->isSaveAllowed()) {
               if (!$this->isPartialSave()) {
                   $this->loadAllAttributes($object);
               }

               if ($this->getEntityTable() ==
\Magento\Eav\Model\Entity::DEFAULT_ENTITY_TABLE
                   && !$object->getEntityTypeId()
               ) {
                   $object->setEntityTypeId($this->getTypeId());
               }

               $object->setParentId((int)$object->getParentId());


$this->objectRelationProcessor->validateDataIntegrity($this->getEntit
yTable(), $object->getData());

               $this->_beforeSave($object);
               $this->processSave($object);
               $this->_afterSave($object);

               $object->afterSave();
           }
           $this->addCommitCallback([$object,
'afterCommitCallback'])->commit();
           $object->setHasDataChanges(false);
       } catch (DuplicateException $e) {
           $this->rollBack();
           $object->setHasDataChanges(true);
           throw new AlreadyExistsException(__('Unique constraint
violation found'), $e);
       } catch (\Exception $e) {
           $this->rollBack();
           $object->setHasDataChanges(true);
           throw $e;
       }

       return $this;
   }

...
```

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

To extend the existing classes, you can create additional EAV attributes and new fields, as well as create the connected classes that realize their interface.

To create additional fields in the database, apply Setup scripts. Let us examine how to create a field in the database using InstallSchema class as an example:

```
...
public function install(SchemaSetupInterface $setup,
ModuleContextInterface    $context)
  {
    $installer = $setup;
    $installer->startSetup();
    $table = $installer->getTable('custom_table');
    $columns = [
        'custom_field' => [
            'type' => \Magento\Framework\DB\Ddl\Table::TYPE_TEXT,
            'nullable' => false,
            'comment' => 'custom_field',
        ],

    ];
    $connection = $installer->getConnection();
    foreach ($columns as $name => $definition) {
        $connection->addColumn($table, $name, $definition);
    }
    $installer->endSetup();
}
```

We extended the existing class with an additional ‘custom_field’ field in ‘custom_table’ table.

A new attribute can be created using \Magento\Eav\Setup\EavSetup class. Example:

```
...
public function __construct(EavSetupFactory $eavSetupFactory)
     {
        $this->eavSetupFactory = $eavSetupFactory;
    }

    public function install(ModuleDataSetupInterface $setup,
ModuleContextInterface $context)
    {
        $eavSetup = $this->eavSetupFactory->create(['setup' =>
$setup]);
        $eavSetup->addAttribute(
            \Magento\Catalog\Model\Product::ENTITY,
            'custom_attribute',
            [
                'type' => 'text',
                'backend' => '',
                'frontend' => '',
                'label' => 'custom attribute',
                'input' => 'text',
                'class' => '',
                'source' => '',
                'global' =>
\Magento\Eav\Model\Entity\Attribute\ScopedAttributeInterface::SCOPE_G
LOBAL,
                'visible' => true,
                'required' => true,
                'user_defined' => false,
                'default' => '',
                'searchable' => false,
                'filterable' => false,
                'comparable' => false,
                'visible_on_front' => false,
                'used_in_product_listing' => true,
                'unique' => false,
                'apply_to' => ''
            ]
        );
    }
...
```

Extension Attributes were introduced in Magento 2; they are responsible for adding the additional data into the created object. Extension attribute should use ExtensibleDataInterface interface. Also, you are advised to realize getExtensionAttributes and setExtensionAttributes methods in your code.

Code for extension attribute is generated during the compilation process, using \Magento\Framework\Code\Generator that applies etc/extension_attributes.xml file from the module directory. Example of extension_attributes.xml :

```
<extension_attributes for="Magento\Sales\Api\Data\OrderInterface">
<attribute code="custom_extension_attribute"
type="Belvg\Extension\Api\Data\CustomExtensionAttributeInterface" />
</extension_attributes>
```

Unlike the common Magento attributes, extension attribute is not automatically loaded from and stored into the database, which means you need to realize the loading and saving manually. For this purpose, plugins are the best choice; they are declared in di.xml file. Example:

```
<type name="Magento\Sales\Api\OrderRepositoryInterface">
   <plugin name="custom_extension_attribute"
type="Belvg\Extension\Plugin\OrderPlugin"/>
</type>
```

In the plugin, we can realize afterGet and afterSave methods that will contain the loading and saving extension attribute logic.

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

Use addFieldToSelect and addAttributeToSelect methods to specify in the collection the fields for selection.

For example:

```
$productCollection->addFieldToSelect("custom_field");
```

To apply filters to collections, use addAttributeToFilter($field, $condition) and addFieldToFilter($field, $condition) methods.

Conditions can be the following:

```
"eq" => equalValue
"neq" => notEqualValue
"like" => likeValue
"nlike" => notLikeValue
"is" => isValue
"in" => inValues
"nin" => notInValues
"notnull" => valueIsNotNull
"null" => valueIsNull
"moreq" => moreOrEqualValue
"gt" => greaterValue
"lt" => lessValue
"gteq" => greaterOrEqualValue
"lteq" => lessOrEqualValue
"finset" => valueInSet
"from" => fromValue, "to" => toValue

Example:
$productCollection->addFieldToFilter('entity_id', array('in' =>
[1,2,3])
setOrder method is used for sorting and processes both filter and
direction fields. For instance:

$productCollection>setOrder('position','ASC');
```

searchCriteria is used for applying filters in repositories.

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

Database abstraction layer realized in Magento 2 in the capacity of \Magento\Framework\Model\ResourceModel\Db\AbstractDb class, realizing such core methods, as save, delete, load.

Also, the following additional methods are realized atop Zend_Adapter:

`addUniqueField`, `unserializeFields`, `serializeFields`, `hasDataChanged`, `prepareDataForUpdate`, `isObjectNotNew`, `saveNewObject`, `afterSave`, `beforeSave`, `isModified`, `afterLoad`, `beforeDelete`, `afterDelete`, etc.

Database layer can have put our exceptions depending on its realization. For example, PDO/Mysql can put out the following exceptions:

```
Zend_Db_Adapter_Exception
Zend_Db_Statement_Exception
Zend_Db_Exception
Zend_Db_Statement_Pdo
PDOException
LocalizedException
InvalidArgumentException
Exception
DuplicateException
```

4.2 Demonstrate ability to use install and upgrade scripts

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

Magento 2 applies schema/data migrations to provide data persistence and database updatability. The migrations contain instructions for:

1. The necessary tables creation and their completion at the initial installation
2. Database scheme and its information conversion for each available application version.

Magento 2 setup scripts are located in <module_dir>/Setup folder.
InstallSchema and InstallData classes are responsible for installing module the first time, while UpgradeSchema and UpgradeData scripts are used when upgrading module's version.

Running setup scripts

Use CLI command to run migration scripts:

```
$ php bin/magento setup:upgrade
```

If Magento detects a new module, then it will instantiate objects from the Vendor\Module\Setup\InstallSchema and Vendor\Module\Setup\InstallData classes. In case the module version has changed, then Vendor\Module\Setup\UpgradeSchema and Vendor\Module\Setup\UpgradeData will be instantiated. Afterward, the corresponding upgrade methods will be executed.

Versioning

Unlike Magento 1, Magento 2 does not contain the inbuilt migration versioning tools, meaning that a developer must check the current module version manually.

```
class UpgradeSchema implements UpgradeSchemaInterface
{
    public function upgrade(
        \Magento\Framework\Setup\SchemaSetupInterface $setup,
        \Magento\Framework\Setup\ModuleContextInterface $context
) {
        $setup->startSetup();

        if (version_compare($context->getVersion(), '1.5.1') < 0) {
            //code to upgrade to 1.5.1
        }

        if (version_compare($context->getVersion(), '1.5.7') < 0) {
            //code to upgrade to 1.5.7
        }

        $setup->endSetup();
    }
}
```

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

InstallSchema class

This setup script is used for modifying database structure at the module's first installation.

```
<?php

namespace Vendor\Module\Setup;

class InstallSchema implements \Magento\Framework\Setup\InstallSchemaInterface
{
    public function install(
        \Magento\Framework\Setup\SchemaSetupInterface $setup,
        \Magento\Framework\Setup\ModuleContextInterface $context)
    {
        $setup->startSetup();

        $table = $setup->getConnection()->newTable(
            $setup->getTable('custom_table')
        )->addColumn(
            'custom_id',
            \Magento\Framework\DB\Ddl\Table::TYPE_INTEGER,
            null,
            ['identity' => true, 'unsigned' => true, 'nullable' => false, 'primary'
=> true],
            'Custom Id'
        )->addColumn(
            'name',
            \Magento\Framework\DB\Ddl\Table::TYPE_TEXT,
            255,
            [],
            'Custom Name'
        )->setComment(
            'Custom Table'
        );
        $setup->getConnection()->createTable($table);

        $setup->endSetup();
}
```

InstallData class

This setup script is applied for adding and modifying the data at the module’s first installation.

```
<?php

namespace Vendor\Module\Setup;

class InstallData implements
\Magento\Framework\Setup\InstallDataInterface
{
    public function upgrade(
        \Magento\Framework\Setup\ModuleDataSetupInterface $setup,
        \Magento\Framework\Setup\ModuleContextInterface $context)
    {
        $setup->startSetup();

        // data installation code

        $setup->endSetup();
    }
}
```

UpgradeSchema class

The setup script is applied for modifying database structure at the module update.

```
<?php

namespace Vendor\Module\Setup;

class UpgradeSchema implements
\Magento\Framework\Setup\UpgradeSchemaInterface
{
    public function upgrade(
        \Magento\Framework\Setup\SchemaSetupInterface $setup,
        \Magento\Framework\Setup\ModuleContextInterface $context
    ) {
        $setup->startSetup();
        if (version_compare($context->getVersion(), '2.3.1') < 0) {
            // upgrade schema to version 2.3.1
        }
        $setup->endSetup();
    }
}
```

UpgradeData class

This setup script is applied for adding and modifying the data in the event of module upgrade.

```php
<?php

namespace Vendor\Module\Setup;

class UpgradeData implements
\Magento\Framework\Setup\UpgradeDataInterface
{
    public function upgrade(
        \Magento\Framework\Setup\ModuleDataSetupInterface $setup,
        \Magento\Framework\Setup\ModuleContextInterface $context
    ) {
        $setup->startSetup();
        if (version_compare($context->getVersion(), '2.3.1') < 0) {
            // upgrade data to version 2.3.1
        }
        $setup->endSetup();
    }
}
```

Recurring scripts

Recurring scripts are run each time setup:upgrade command is launched and depend on the module's version.

```php
<module_dir>/Setup/Recurring.php

<?php

namespace Vendor\Module\Setup;

class Recurring implements
\Magento\Framework\Setup\InstallSchemaInterface
{
    public function install(
        \Magento\Framework\Setup\SchemaSetupInterface$setup,
        \Magento\Framework\Setup\ModuleContextInterface$context
    ) {
        echo 'Running';
    }
}
```

Section 5: Using the Entity-Attribute-Value (EAV) Model

5.1 Demonstrate ability to use EAV model concepts

Describe the EAV hierarchy structure

EAV (Entity-attribute-value) is a model for storing entity attribute values in a certain storage place. For storage, Magento 2 supports MySQL-compatible databases (like MySQL, MySQL NDB Cluster, MariaDB, Percona, and others).

The table of the classic EAV model has 3 columns:

1. entity (object to which the attribute value must be set)
2. attribute
3. value

In the Flat model, attribute values are stored in the same table as the entities; a separate column is created for each attribute in the table.
In the EAV model, attribute values are stored in a separate table. A separate column is not created for each attribute, and a new row is created for each attribute value of an entity in the EAV table.

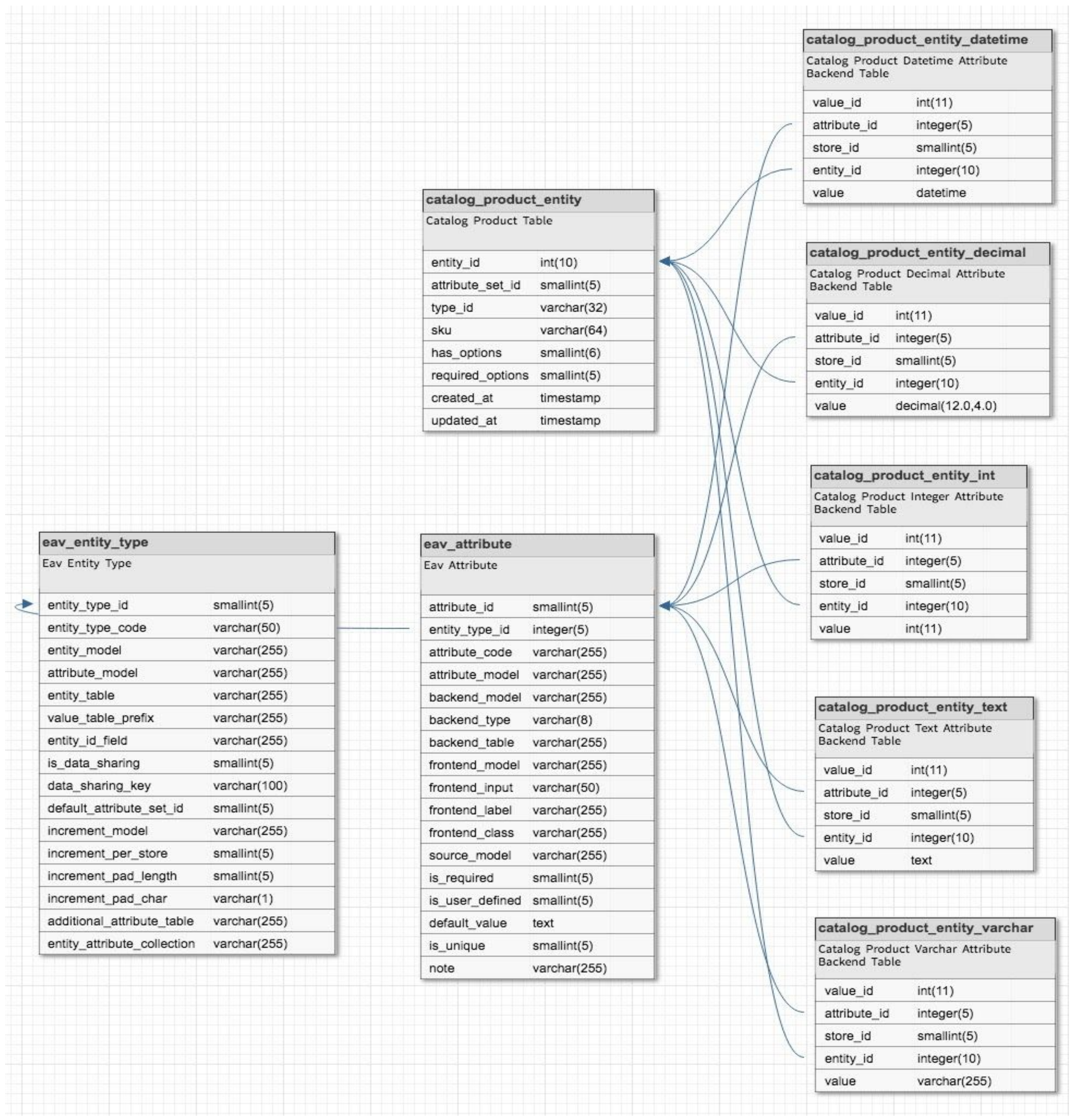
How EAV data storage works in Magento

Differences between a classic EAV model and the one used in Magento

1. A backend type (varchar, int, text …) is assigned for each attribute
2. Each EAV entities type have a separate table. EAV entities types are stored in eav_entity_type table. The names of the tables, where the entities are stored, are located in entity_table column.
3. System EAV entities are stored in eav_entity table
4. Attributes, that have backend type as static, are stored in the same table as entities. These attributes have global scope.

5. Each backend type of each entity type has its own table, in which attribute values are stored. The name of this table is crafted according to the template {entity_table} _ {backend_type}, where entity_table is the name of the entity table and backend_type is the backend type of the attribute. These tables include the following columns: value_id (int), attribute_id (int), store_id ** (int), entity_id * (int), value (depending on the backend type).

To get a better understanding of the connections, study this scheme:



catalog_product_entity_datetime
Catalog Product Datetime Attribute Backend Table

value_id	int(11)
attribute_id	integer(5)
store_id	smallint(5)
entity_id	integer(10)
value	datetime

catalog_product_entity
Catalog Product Table

entity_id	int(10)
attribute_set_id	smallint(5)
type_id	varchar(32)
sku	varchar(64)
has_options	smallint(6)
required_options	smallint(5)
created_at	timestamp
updated_at	timestamp

catalog_product_entity_decimal
Catalog Product Decimal Attribute Backend Table

value_id	int(11)
attribute_id	integer(5)
store_id	smallint(5)
entity_id	integer(10)
value	decimal(12.0,4.0)

catalog_product_entity_int
Catalog Product Integer Attribute Backend Table

value_id	int(11)
attribute_id	integer(5)
store_id	smallint(5)
entity_id	integer(10)
value	int(11)

eav_entity_type
Eav Entity Type

entity_type_id	smallint(5)
entity_type_code	varchar(50)
entity_model	varchar(255)
attribute_model	varchar(255)
entity_table	varchar(255)
value_table_prefix	varchar(255)
entity_id_field	varchar(255)
is_data_sharing	smallint(5)
data_sharing_key	varchar(100)
default_attribute_set_id	smallint(5)
increment_model	varchar(255)
increment_per_store	smallint(5)
increment_pad_length	smallint(5)
increment_pad_char	varchar(1)
additional_attribute_table	varchar(255)
entity_attribute_collection	varchar(255)

eav_attribute
Eav Attribute

attribute_id	smallint(5)
entity_type_id	integer(5)
attribute_code	varchar(255)
attribute_model	varchar(255)
backend_model	varchar(255)
backend_type	varchar(8)
backend_table	varchar(255)
frontend_model	varchar(255)
frontend_input	varchar(50)
frontend_label	varchar(255)
frontend_class	varchar(255)
source_model	varchar(255)
is_required	smallint(5)
is_user_defined	smallint(5)
default_value	text
is_unique	smallint(5)
note	varchar(255)

catalog_product_entity_text
Catalog Product Text Attribute Backend Table

value_id	int(11)
attribute_id	integer(5)
store_id	smallint(5)
entity_id	integer(10)
value	text

catalog_product_entity_varchar
Catalog Product Varchar Attribute Backend Table

value_id	int(11)
attribute_id	integer(5)
store_id	smallint(5)
entity_id	integer(10)
value	varchar(255)

To get / record attribute value, you need:

1. `store_id**`
2. `entity_id*`
3. `Entity_table or entity_type_id`
4. `attribute_id`

If entity_table is unknown, but entity_type_id is known, then entity_table can be obtained from the eav_entity_type table.

Magento receives entity attribute values as one large SQL query, which is generated by the following algorithm:

1. Get all attribute tables for a specific entity_type
2. For each table, do the following:

- a select subquery is created from the current table, which requests value and attribute_id
- the condition is added that entity_id = ID of the requested entity *
- a condition is added for each scope that store_id IN ($ scope-> getValue ()) **
- sorted by store_id in a descending order

3. UNION ALL of all subqueries is performed.

* the field is not necessarily called entity_id. Magento uses $metadata->getLinkField() to get the field name.
** the field is not necessarily called store_id. Magento uses $scope->getIdentifier() to get the field name.

What happens when a new attribute is added to the system

When a new attribute is added, new records are created in the database. Similar records are created for each attribute type.
Let's consider some standard attribute types.

Text Field

Let's examine how to create a new attribute of the Text Field type.

1. A new record in eav_attribute table is made

attribute_id	entity_type_id	attribute_code	...
154	4	test_code	...

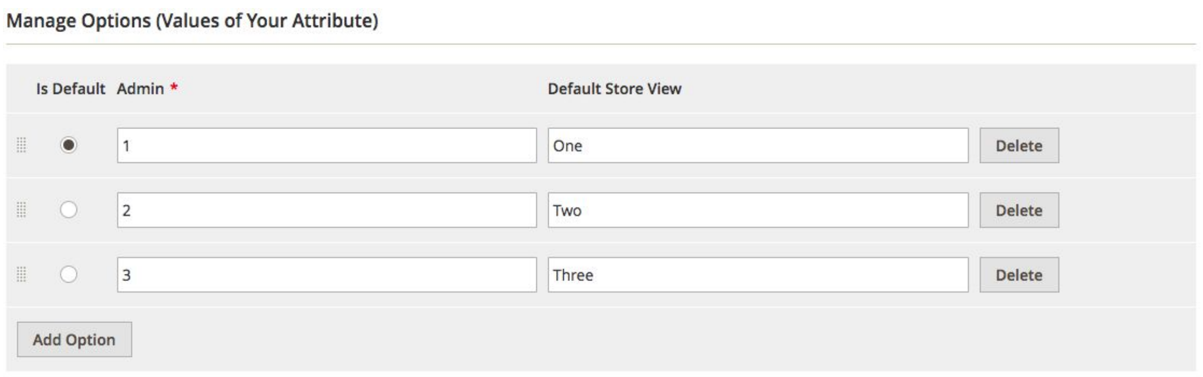
2. One entry per store view is created in the eav_attribute_label table with the Label attribute.

attribute_label_id	attribute_id	store_id	value
17	154	1	Test label

3. A new record in the catalog_eav_attribute table is created

attribute_id	frontend_input_renderer	is_global	...
154	NULL	1	...

Dropdown


Manage Options (Values of Your Attribute)

Is Default	Admin *	Default Store View	
●	1	One	Delete
○	2	Two	Delete
○	3	Three	Delete

Differences compared to Field:

- eav_attribute.frontend_input = “select”
- eav_attribute.source_model = “Magento\Eav\Model\Entity\Attribute\Source\Table”
- eav_attribute.backend_type = “int”
- eav_attribute.default_value = eav_attribute_option.option_id by default
- Lines are added in eav_attribute_option, one for each value

option_id	attribute_id	sort_order
210	155	1
211	155	2
212	155	3

- Lines are added in eav_attribute_option_value

value_id	option_id	store_id	value
208	211	0	2
210	212	0	3
205	210	1	One
207	211	1	Two
209	212	1	Three
206	210	0	1

Price

Differences compared to Text Field:

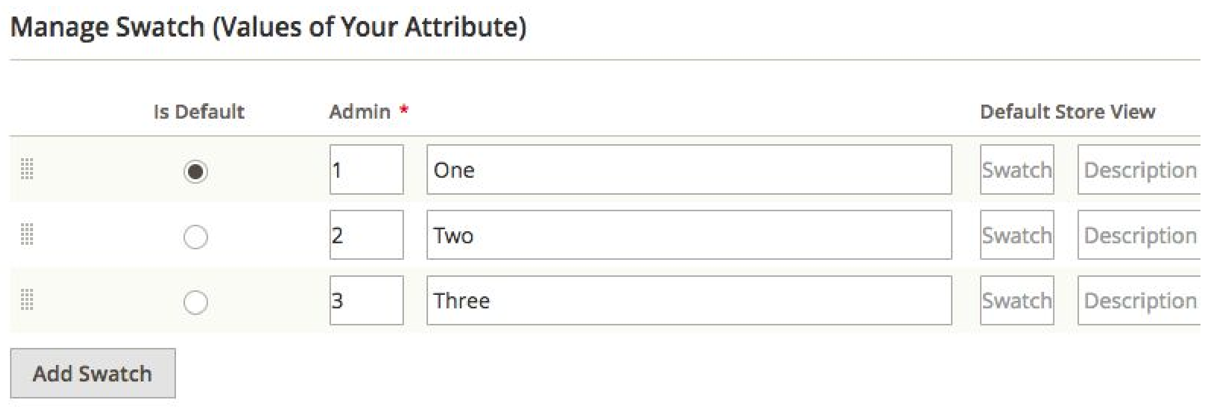
- eav_attribute.frontend_input = “price”
- eav_attribute.backend_model = “Magento\Catalog\Model\Product\Attribute\Backend\Price”
- eav_attribute.backend_type = “decimal”

Media image

Differences compared to Text Field:

- eav_attribute.frontend_input = “media_image”

Text swatch



Is Default	Admin		Default Store View	
●	1	One	Swatch	Description
○	2	Two	Swatch	Description
○	3	Three	Swatch	Description

Differences compared to Dropdown:

- catalog_eav_attribute.additional_data look similarly {"swatch_input_type":"text","update_product_preview_image":"0","use_product_image_for_swatch":0}
- Lines into eav_attribute_option_swatch are added

swatch_id	option_id	store_id	type	value
1	213	0	0	1
6	215	1	0	3
2	213	1	0	1
4	214	1	0	2
3	214	0	0	2
5	215	0	0	3

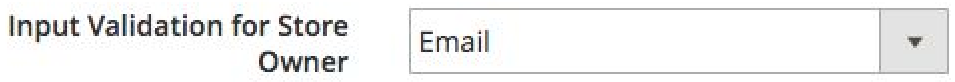
How are attributes presented in the admin?

Attributes consist of a name (Label) + field, the value of which the administrator can change.
The following attribute properties affect the display:

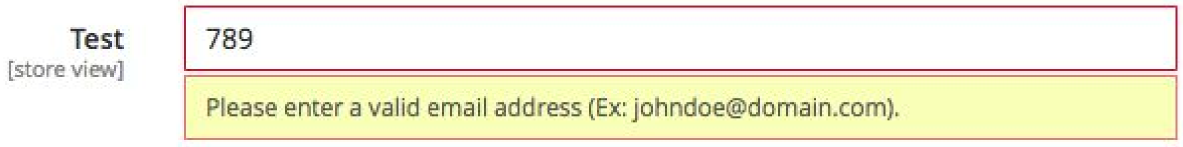
- frontend_model - a class that describes the field display in the frontend section of a site. Inherited from Magento \ Eav \ Model \ Entity \ Attribute \ Frontend \

AbstractFrontend and overrides the getValue method to change the displayed attribute values

- frontend_input - the form element that is displayed in the admin section of the site
- frontend_label - the name of the attribute, displayed in the admin section of the site. It is also displayed in the site frontend, unless otherwise specified in the eav_attribute_label table (in the attribute settings it is changed in the "Manage Labels" tab)
- frontend_class - used to validate the attribute value in the admin section of the site. In the attribute settings, the property is called Input Validation for Store Owner. For e-mail validation, the frontend_class property is set to "validate-email".



Input Validation for Store Owner: Email



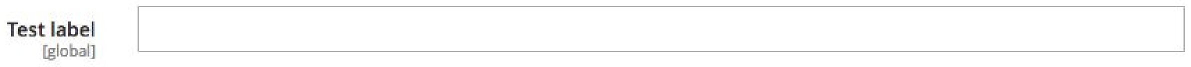
Test [store view]: 789

Please enter a valid email address (Ex: johndoe@domain.com).

Magento has the following default attribute types:

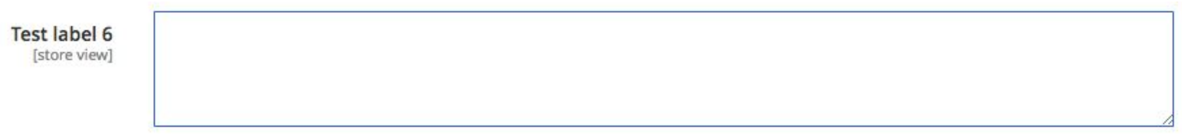
Text Field

Has the form of text input.



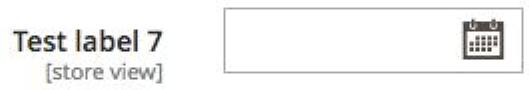
Text Area

Has the form of a many-line input field. A WYSIWYG editor can be enabled for it in the attribute settings.



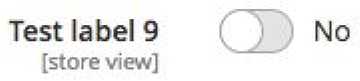
Date

Has the form of a text field for data input and a drop-down calendar.



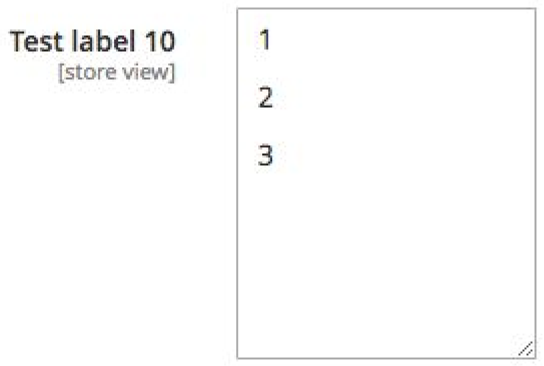
Yes/No

Has the form of a switcher with two entities: Yes or No.



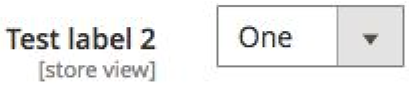
Multiple Select

Has the form of a list that offers to select several values, or no value at all.



Dropdown

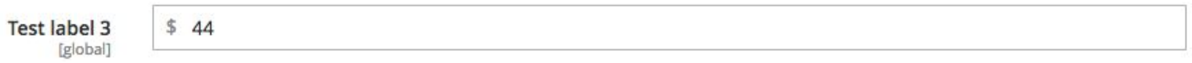
Has the form of a drop-down list that offers to select only one value.


Test label 2 [store view]

One

Price

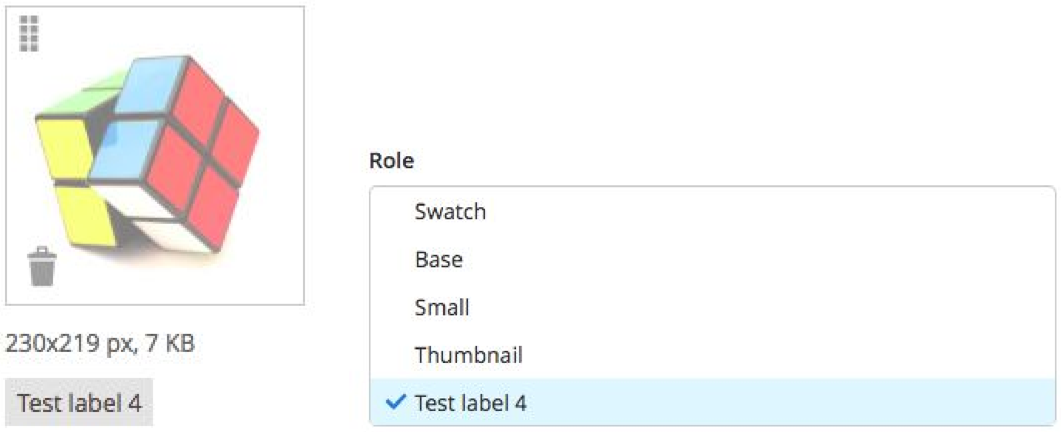
Has the form of a text field with a currency symbol.


Test label 3 [global]

$ 44

Media Image

Media Image is different from any other fields, for it has no field. Instead, an attribute of Media Image type adds role to the product image and video.


230x219 px, 7 KB

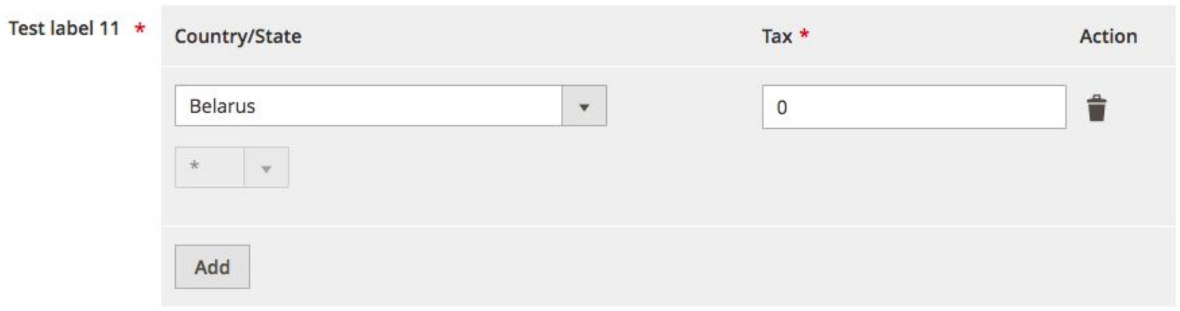
Test label 4

Role

Swatch
Base
Small
Thumbnail
✓ Test label 4

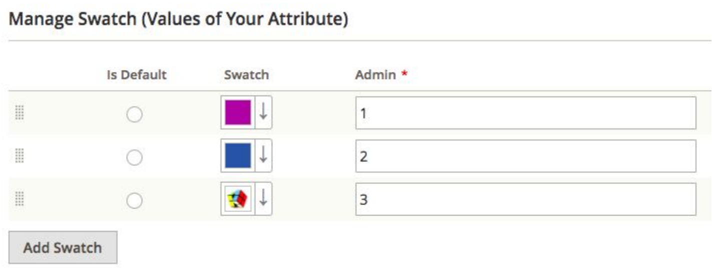
Fixed Product Tax

Allows to modify taxes for countries and its states. Has the form of a table with a country/state and corresponding tax. The type is added by the Magento_Weee module, where WEE stands for Waste Electrical and Electronic Equipment Directive.



Visual Swatch

Visual Swatch is provided for the administrator, the same as Dropdown. The user sees it as a button, filled with color, or containing a picture. This field can be used in Configurable products, for instance, for the color selection.


Manage Swatch (Values of Your Attribute)

	Is Default	Swatch	Admin *
	○	🟪	1
	○	🟦	2
	○		3

Text Swatch

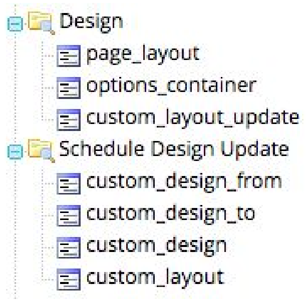
Is provided similarly to Visual Swatch, only the button content is in text format.

What is the role of attribute sets and attribute groups?

Attribute set are applied to display different attributes during various product types editing (eg., shoes and clothes). For example, we have shoe_size and clothing_size attributes, and we need to display the first one for shoes, and the second - for clothes.

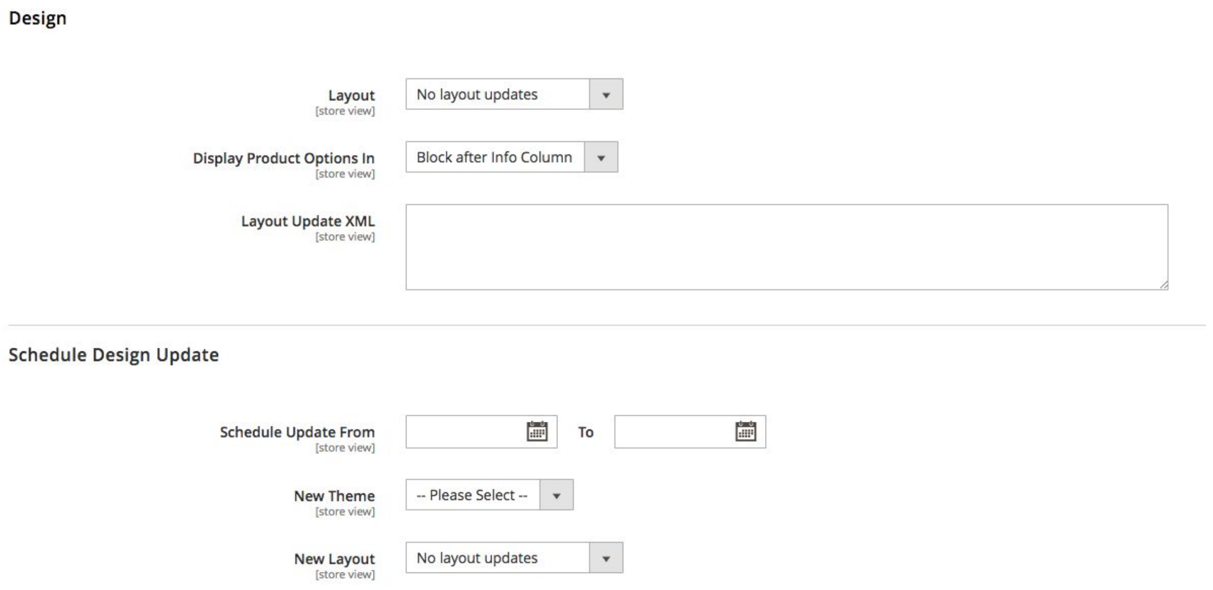
Attribute set allows to hide the unnecessary attributes, display the needed ones, modify the sorting and group. Groups simplify the process of filling product information by the administrator, but have no effect on product loading / storing logic.

Below is a screenshot, where Design and Schedule Design Update are attribute groups, and the nested elements are attributes.
Attributes in attribute set settings:



- Design
  - page_layout
  - options_container
  - custom_layout_update
- Schedule Design Update
  - custom_design_from
  - custom_design_to
  - custom_design
  - custom_layout

Attributes in the product:



Design

Layout [store view]: No layout updates

Display Product Options In [store view]: Block after Info Column

Layout Update XML [store view]:

Schedule Design Update

Schedule Update From [store view]: To

New Theme [store view]: -- Please Select --

New Layout [store view]: No layout updates

Which additional options do you have when saving EAV entities?

The EAV entity modification page offer the following capabilities:

- Default attribute value. Allows to set the value that will be inserted into the attribute field if it is not filled in. This simplifies the process of adding and modifying the entities. These are the potential default attribute values for the product:
  - Text Field - any one-line text
  - Text Area - any multi-line text
  - Date - any date
  - Yes/No - yes or no
  - Multiple Select - any number of options
  - Dropdown - one option
  - Price - no default value
  - Media image - no default value
  - Visual Swatch - one option
  - Text Swatch - one option
  - Fixed Product Tax - no default value
- Scope selection. Scope allows to set different attribute values for different website / store / view. Below is a list of the possible attribute scope for the product:
  - Text Field - store view / website / global
  - Text Area - store view / website / global
  - Date - store view / website / global
  - Yes/No - store view / website / global
  - Multiple Select - store view / website / global
  - Dropdown - store view / website / global
  - Price - website / global (configured in Stores - Configuration - Catalog - Catalog - Price - Catalog Price Scope)
  - Media image - store view / website / global
  - Visual Swatch - store view / website / global
  - Text Swatch - store view / website / global
  - Fixed Product Tax - global
- Attribute set selection (products only). Allows to regroup or hide the attributes that do not suit the current product type.

Before saving an EAV entity at the client side, we have the following features:

- Prohibition to save entities, if there are empty attribute fields with the is_required = 1 feature. It allows to define what attributes of the entity are required to be filled in.
- Validation of the attribute fields values by the algorithm, set in frontend_class. The following frontend classes are supported by default it Magento:
  - validate-number: Decimal Number
  - validate-digits: Integer Number
  - validate-email: Email
  - validate-url: URL
  - validate-alpha: Letters
  - validate-alphanum: Letters (a-z, A-Z) or Numbers (0-9)

Before saving an EAV entity at the server side, we have the following features:

- Check attribute fields with is_required = 1 for fullness. Magento checks the required fields on both the client and server sides.
- Uniqueness check of the attribute fields with is_unique = 1.
- Perform operations in backend_model
  - Validation (validate method). Allows to realize additional server check before saving.
  - Operation before saving (beforeSave method)
  - Operation after saving (afterSave method)

Module catalog has an additional catalog_eav_attribute table for attributes, where the following parameters are stored:

- Frontend Input Renderer
- Is Global
- Is Visible
- Is Searchable
- Is Filterable
- Is Comparable
- Is Visible On Front
- Is HTML Allowed On Front
- Is Used For Price Rules
- Is Filterable In Search
- Is Used In Product Listing
- Is Used For Sorting

- Is Visible In Advanced Search
- Is WYSIWYG Enabled
- Is Used For Promo Rules
- Is Required In Admin Store
- Is Used in Grid
- Is Visible in Grid
- Is Filterable in Grid

These parameters allow to perform a more sophisticated attribute configuration.

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

To customize an attribute, modify the following attribute features:

- backend_model
- source_model
- attribute_model
- frontend_model

Attribute Model

The model allows to perform a more sophisticated attribute setting. By default, Magento\Eav\Model\Entity\Attribute is used, and a model, different from the default one, is rarely applied.

Backend Model

The model is used for processing and validating the attribute values.

By default, Magento\Eav\Model\Entity\Attribute\Backend\DefaultBackend is used.
It allows to define:

- validate
- afterLoad
- beforeSave

- afterSave
- beforeDelete
- afterDelete
- and other

Example:

```
class TestBackend extends
\Magento\Eav\Model\Entity\Attribute\Backend\AbstractBackend
{
    public function validate($object)
    {
      $attribute_code = $this->getAttribute()->getAttributeCode();
        $value = $object->getData($attribute_code);

        if ($value == 'test') {
           throw new
\Magento\Framework\Exception\LocalizedException(__("Value can't be
test"));
        }

        return true;
    }
}
```

Source Model

The model is used for providing the list of the attribute values.

By default, Magento\Eav\Model\Entity\Attribute\Source\Config is used.

Example:

```
class TestSource extends
\Magento\Eav\Model\Entity\Attribute\Source\AbstractSource
{
    public function getAllOptions()
    {
        if (!$this->_options) {
            $this->_options = [
                ['label' => __('Label 1'), 'value' => 'value 1'],
                ['label' => __('Label 2'), 'value' => 'value 2'],
                ['label' => __('Label 3'), 'value' => 'value 3'],
                ['label' => __('Label 4'), 'value' => 'value 4']
            ];
        }
        return $this->_options;
    }
}
```

Frontend Model

Frontend model is used for displaying frontend part of the website.
By default, Magento\Eav\Model\Entity\Attribute\Frontend\DefaultFrontend is applied.

Example:

```
class TestFrontend extends
\Magento\Eav\Model\Entity\Attribute\Frontend\AbstractFrontend
{
    public function getValue(\Magento\Framework\DataObject $object)
    {
        $attribute_code = $this->getAttribute()->getAttributeCode();
        $value = $object->getData($attribute_code);
        return nl2br(htmlspecialchars($value));
    }
}
```

Describe the key differences between EAV and flat table collections

For a developer, there is no big difference between EAV and Flat. Models, resource models and collections are created similarly.

Let us examine how the classes for EAV (City) and Flat (Country) are created.

Collections

EAV

```
<?php
namespace Belvg\Geo\Model\ResourceModel\City;

class Collection extends
\Magento\Eav\Model\Entity\Collection\AbstractCollection
{
    protected function _construct()
    {
        $this->_init(
            'Belvg\Geo\Model\City',
            'Belvg\Geo\Model\ResourceModel\City'
        );
    }
}
```

FLAT

```
<?php
namespace Belvg\Geo\Model\ResourceModel\Country;

class Collection extends
\Magento\Framework\Model\ResourceModel\Db\Collection\AbstractCollecti
on
{
    protected function _construct()
    {
        $this->_init(
            'Belvg\Geo\Model\Country',
            'Belvg\Geo\Model\ResourceModel\Country'
        );
    }
}
```

Resource model

EAV

```
<?php
namespace Belvg\Geo\Model\ResourceModel;

class City extends \Magento\Eav\Model\Entity\AbstractEntity
{
    public function getEntityType()
    {
        if (empty($this->_type)) {
            $this->setType(\Belvg\Geo\Model\City::ENTITY);
        }
        return parent::getEntityType();
    }
}
```

FLAT

```
<?php
namespace Belvg\Geo\Model\ResourceModel;

class Country extends
\Magento\Framework\Model\ResourceModel\Db\AbstractDb
{
    protected function _construct()
    {
        $this->_init(
            'belvg_geo_country',
            'entity_id'
        );
    }
}
```

Models

EAV

```
<?php
namespace Belvg\Geo\Model;

class City extends \Magento\Framework\Model\AbstractModel
{
    const ENTITY = 'belvg_geo_city';

    protected function _construct()
    {
        $this->_init('Belvg\Geo\Model\ResourceModel\City');
    }
}
```

FLAT

```
<?php
namespace Belvg\Geo\Model;

class Country extends \Magento\Framework\Model\AbstractModel
{
    protected function _construct()
    {
        $this->_init('Belvg\Geo\Model\ResourceModel\Country');
    }
}
```

Difference between EAV classes and FLAT

Additionally, the following methods, that contain the attribute as its code or its object, are added into EAV collections:

- addAttributeToSelect converts attribute into its code and calls addFieldToSelect

- addAttributeToFilter converts attribute into its code and calls addFieldToFilter
- addAttributeToSort converts attribute into its code and calls addOrder

Additionally, in EAV resource models, the methods for working with attributes are added and methods load, save, delete are overridden.

EAV utilizes the following classes for working with attributes:
- Magento\Eav\Model\ResourceModel\CreateHandler - to create values in new entities
- Magento\Eav\Model\ResourceModel\UpdateHandler - to modify / delete / add values into the existing entities
- Magento\Eav\Model\ResourceModel\ReadHandler - to get the values

The key difference between EAV and Flat lies in data storage.

The configuration settings in the admin panel have Use Flat Catalog Category and Use Flat Catalog Product options. The settings allow to edit the sources of the uploaded products and categories, changing them from EAV tables for FLAT index tables.

Product attribute gets to the flat table in case it complies to at least one condition:

- backend_type is set as static
- the field filter is enabled
- the attribute is used in the product list
- the attribute is used for sorting

Product collection is a Magento\Catalog\Model\ResourceModel\Product\Collection class. In this class, many methods look the following way:

```
if ($this->isEnabledFlat()) {
...
} else {
...
}
```

Therefore, if Flat is enabled, one action is performed, but in case it is disabled, then another action is triggered.

The situation is different with categories. Magento\Catalog\Model\Category model is initialized the following way:

```
protected function _construct()
    {
        // If Flat Index enabled then use it but only on frontend
        if ($this->flatState->isAvailable()) {
$this->_init(\Magento\Catalog\Model\ResourceModel\Category\Flat::clas
s);
            $this->_useFlatResource = true;
        } else {

$this->_init(\Magento\Catalog\Model\ResourceModel\Category::class);
        }
    }
```

Therefore, different resource models are used.
Collections also divided into two classes, unlike the products:

- Magento\Catalog\Model\ResourceModel\Category\Collection
- Magento\Catalog\Model\ResourceModel\Category\Flat\Collection

In which situations would you use EAV for a new entity

Using EAV for a new entity is advisable in case at least one of the following conditions is true:

1. There is a scope.
2. Admins and modules have the ability to add attributes into an entity or modify attributes backend type.
3. Potentially, the number of columns in Flat model can exceed 1017.
4. Potentially, the required amount of indexes, like INDEX in Flat model, can exceed 64 or reach the amount when the entities' adding / modifying / deletion will be slow.

What are the pros and cons of EAV architecture?

Advantages of EAV over Flat

1. The implementation of SCOPE in the Flat model stores a lot of unnecessary information. For example, if you want to redefine one attribute in another scope, then both EAV and Flat will create one line each. But in EAV, the number of columns is always fixed, while in Flat it can reach 1000. Additionally, there is a problem when the value is inherited from the parent scope. You can mark such values as NULL, but then you must prevent the attributes from being NULL if they are not inherited.

2. Quickly add a new attribute. Adding a new attribute does not change the EAV table in any way. In Flat, you need to add a new column. The ALTER TABLE operation is also slow. This is especially noticeable in large tables. *

3. Change the backend type attribute faster. To change the attribute type in EAV, you need to move the this attribute data from one table to another. For Flat, you need to perform ALTER TABLE. *

4. EAV allows to separate attribute values from entity field values.

5. The following InnoDB restrictions on the table restrict Flat:

- The table may contain no more than 1017 columns
- A table can contain a maximum of 64 indexes of type INDEX

Disadvantages of EAV over Flat

1. Getting entity attribute values in EAV is slower than in Flat **
2. Search by attribute value in EAV is slower than in Flat **
3. In Flat, you can create an index on several attributes to speed up the search **

* As for EAV, when adding / changing an attribute type, ALTER TABLE is performed if the attribute's backend type is static
** To speed up operations, some attributes can be set the backend type as static and, if necessary, create indexes on the column with the attribute in the database, but then the visibility of the attribute will only be global. Another option is to use a flat table as an index if the data is allowed not to be up to date.

5.2 Demonstrate ability to use EAV entity load and save

The Magento\Framework\EntityManager\EntityManager class was introduced in Magento 2.1 for loading, saving, checking for existence, deleting EAV and Flat objects (now it is considered deprecated).
To work via EntityManager, you must provide information about the entity interface in di.xml for MetadataPool and for HydratorPool.

```
<type name="Magento\Framework\EntityManager\MetadataPool">
    <arguments>
        <argument name="metadata" xsi:type="array">
            <item name="MyVendor\MyModule\Api\Data\MyEntityInterface"
xsi:type="array">
                <item name="entityTableName"
xsi:type="string">myvendor_mymodule_myentity_entity</item>
                <item name="eavEntityType"
xsi:type="string">myvendor_mymodule_myentity</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>

<type name="Magento\Framework\EntityManager\HydratorPool">
    <arguments>
        <argument name="hydrators" xsi:type="array">
```

```
            <item name="MyVendor\MyModule\Api\Data\MyEntityInterface"
xsi:type="string">Magento\Framework\EntityManager\AbstractModelHydrator</item>
        </argument>
    </arguments>
</type>
```

Magento\Framework\EntityManager\OperationPool class contains operations array that call EntityManager for loading, saving, existence check and object deletion. The operation is performed by execute method.
Default operations:

```
checkIfExists -
Magento\Framework\EntityManager\Operation\CheckIfExists
read - Magento\Framework\EntityManager\Operation\Read
create - Magento\Framework\EntityManager\Operation\Create
update - Magento\Framework\EntityManager\Operation\Update
delete - Magento\Framework\EntityManager\Operation\Delete
```

By default, CheckIfExists operation checks for the existence of an entry in the main table with a direct SQL query.

By default, Read operation performs three sub-operations:

- ReadMain
- ReadAttributes
- ReadExtensions

By default, Create operation performs three sub-operations:

- CreateMain
- CreateAttributes
- CreateExtensions

By default, Update operation performs three sub-operations:

- UpdateMain
- UpdateAttributes
- UpdateExtensions

By default, Delete operation performs three sub-operations (in reverse order):

- DeleteExtensions

- DeleteAttributes
- DeleteMain

Attribute operations are located in the class Magento \ Framework \ EntityManager \ Operation \ AttributePool.
Extensions operations are located in the class Magento \ Framework \ EntityManager \ Operation \ ExtensionPool.
.
Using overrides of attributes operations, EAV applies the following classes for operations:

- Magento\Eav\Model\ResourceModel\CreateHandler
- Magento\Eav\Model\ResourceModel\UpdateHandler
- Magento\Eav\Model\ResourceModel\ReadHandler

CreateHandler writes attribute values into the new entities.

UpdateHandler takes a snapshot thanks to ReadSnapshot (which is based on ReadHandler). Then, UpdateHandler:

- Changes the value, if the attribute is modified relative to Snapshot and the new value is not empty or the attribute allows empty values
- Deletes, if the attribute is changed relative to Snapshot and the new value is empty and the attribute does not allow empty values
- Creates, if the attribute is absent in Snapshot and the new value is not empty or the attribute allows empty values

ReadHandler performs reading, using the following algorithm:

1. Get all attribute tables for a specific entity_type
2. For each table, the following is performed:

A. a select subquery is created from the current table, which requests value and attribute_id
B. the condition is added that entity_id = ID of the requested entity
C. a condition is added for each scope from the context that store_id IN ($scope-> getValue ())
D. sorted by store_id in descending order

3. Performs UNION ALL of all subqueries.
4. Executes an SQL query
5. Writes the resulting values are written into the $entityData array.

What happens when an EAV entity has too many attributes?

The advantage of EAV, compared to Flat, is that Flat table in InnoDb can not contain more than 1017 columns. In EAV, the number of attributes is limited by the maximum size of innodb tables.

How does the number of websites/stores affect the EAV load/save process?

The number of websites/stores impacts the entity loading / saving.
Let us consider the influence of websites/stores on attributes saving and loading.

In Magento, scope in EAV is realized due to store_id column in the EAV attribute value tables.

- `store_id = 0`, if scope global
- `store_id = ID` of the selected Store View, if not global

Attribute values are loaded the following way:

- `store_id IN (STORE_IDS)`, where `STORE_IDS` are Store View identificators of the current context.

The context is calculated recursively:
In STORE_IDS array, the identificator of the current scope is added.
In case the current scope has Fallback scope, then an operation is repeated for Fallback.

Example: store_id = 5, fallback is specified as 3, and 3 has fallback as 0. Then STORE_IDS is: [5, 3, 0].
In most cases, the standard scope provider Magento \ Store \ Model \ StoreScopeProvider is used.
It works as follows:
Store_id is specified as the identifier of the current Store View
If current store_id! = 0, then it adds a fallback before as 0

It turns out that number of websites / stores does not directly affect * the upload speed, because when StoreScopeProvider is used during loading, the context contains no more than two store_id, i.e. does not depend on the number of websites / stores.

* the number of websites / stores always indirectly affects the loading of attribute values, as the more rows in the attribute value tables, the slower is the load.

Let us consider the value of store_id while maintaining the attribute value.

Attribute scope	All Store Views	Certain Store View
global	store_id = 0	store_id = 0
website	store_id = 0	Entries created / modified for each Store View of the given Website
store view	store_id = 0	store_id = ID Store View

Thus, the attribute saving speed, depending on the number of websites / stores, is affected by the presence of modified attributes from the scope website. The more Store Views are in the Website, the more entries in the database you need to create or modify entries, the slower is the saving.

How would you customize the load and save process for an EAV entity in the situations described here?

To override the operations, add di.xml into the following:

```
<type name="Magento\Framework\EntityManager\OperationPool">
    <arguments>
        <argument name="operations" xsi:type="array">
            <item name="MyVendor\MyModule\Api\Data\MyEntityInterface"
xsi:type="array">
                <item name="checkIsExists"
xsi:type="string">MY_NAMESPACE\CheckIsExists</item>
                <item name="read" xsi:type="string">MY_NAMESPACE\Read</item>
                <item name="create" xsi:type="string">MY_NAMESPACE\Create</item>
                <item name="update" xsi:type="string">MY_NAMESPACE\Update</item>
                <item name="delete" xsi:type="string">MY_NAMESPACE\Delete</item>
            </item>
        </argument>
    </arguments>
</type>
```

You can partially override, for example, a single read operation, then the default operations will be used for the rest of the operations.

We can also override work operations with the attributes for EAV:

```
<type name="Magento\Framework\EntityManager\Operation\AttributePool">
    <arguments>
        <argument name="extensionActions" xsi:type="array">
            <item name="eav" xsi:type="array">
                <item name="MyVendor\MyModule\Api\Data\MyEntityInterface"
xsi:type="array">
                    <item name="read"
xsi:type="string">MY_NAMESPACE\ReadHandler</item>
                    <item name="create"
xsi:type="string">MY_NAMESPACE\CreateHandler</item>
                    <item name="update"
xsi:type="string">MY_NAMESPACE\UpdateHandler</item>
```

```
                </item>
            </item>
        </argument>
    </arguments>
</type>
```

Extensions override:

```
<type name="Magento\Framework\EntityManager\Operation\ExtensionPool">
    <arguments>
        <argument name="extensionActions" xsi:type="array">
            <item name="MyVendor\MyModule\Api\Data\MyEntityInterface"
xsi:type="array">
                <item name="read" xsi:type="array">
                    <item name="myReader"
xsi:type="string">MY_NAMESPACE\ReadHandler</item>
                </item>
                <item name="create" xsi:type="array">
                    <item name="myCreator"
xsi:type="string">MY_NAMESPACE\CreateHandler</item>
                </item>
                <item name="update" xsi:type="array">
                    <item name="myUpdater"
xsi:type="string">MY_NAMESPACE\UpdateHandler</item>
                </item>
            </item>
        </argument>
    </arguments>
</type>
```

5.3 Demonstrate ability to manage attributes

Describe EAV attributes, including the frontend/source/backend structure

Backend Model performs loading, saving, deletion and validation of the attribute.

Source Model provides a list of values for the attributes, which is later used for dropdown/multiselect attributes.

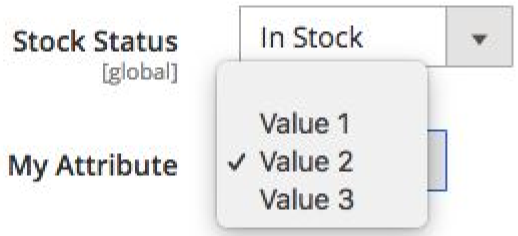
Frontend Model displays the attribute at the frontend side.

How would you add dropdown/multiselect attributes?

You can create attributes through the setup script (Vendor \ Module \ Setup \ InstallData or Vendor \ Module \ Setup \ UpgradeData) or via the admin panel (for products only). In order to create dropdown / multiselect attributes, it is necessary to specify the frontend_input attribute as “select” or “multiselect” for a single or multiple selection respectively. Additionally, you must specify the source model, which will point to a list of possible values. If the source model is Magento \ Eav \ Model \ Entity \ Attribute \ Source \ Table, then you can specify the possible values of this attribute in the option property when creating an attribute through the setup script.

Example:

```php
<?php

namespace Vendor\Module\Setup;

use Magento\Eav\Setup\EavSetupFactory;
use Magento\Framework\Setup\InstallDataInterface;
use Magento\Framework\Setup\ModuleContextInterface;
use Magento\Framework\Setup\ModuleDataSetupInterface;

class InstallData implements InstallDataInterface
{
    /**
     * @var EavSetupFactory
     */
    protected $eavSetupFactory;

    public function __construct(EavSetupFactory $eavSetupFactory)
    {
        $this->eavSetupFactory = $eavSetupFactory;
    }
    public function install(ModuleDataSetupInterface $setup, ModuleContextInterface
$context)
    {
        $eavSetup = $this->eavSetupFactory->create(['setup' => $setup]);

        $eavSetup->addAttribute(
            \Magento\Catalog\Model\Product::ENTITY,
            'my_attribute',
            [
                'type' => 'int',
                'label' => 'My Attribute',
                'input' => 'select',
                'source' =>
\Magento\Eav\Model\Entity\Attribute\Source\Table::class,
                'required' => false,
                'option' => ['values' => ['Value 1', 'Value 2', 'Value 3']]
            ]
        );
    }
}
```



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

EAV attribute has several features that we can specify.

Magento\Eav\Model\Entity\Setup\PropertyMapper class contains the conversion of property names in the setup script into EAV attribute property in the database.

Feature in setup script	Feature in the database	Default value
attribute_model	attribute_model	null
backend	backend_model	null
type	backend_type	varchar
table	backend_table	null
frontend	frontend_model	null
input	frontend_input	text
label	frontend_label	null
frontend_class	frontend_class	null
source	source_model	null
required	is_required	1
user_defined	is_user_defined	0
default	default_value	null
unique	is_unique	0
note	note	null
global	is_global	\Magento\Eav\Model\Entity\Attribute\ScopedAttributeInterface::SCOPE_GLOBAL

You can also specify:

- sort_order - attribute position, recorded in eav_entity_attribute
- group - adds attributes in a certain group
- option - the list of values for dropdown/multiselect attributes

Additionally, Magento/Catalog has the following properties (according to \Magento\Catalog\Model\ResourceModel\Setup\PropertyMapper):

Feature in setup script	Feature in database	Default value

input_renderer	frontend_input_renderer	null
global	is_global	\Magento\Eav\Model\Entity\Attribute\ScopedAttributeInterface::SCOPE_GLOBAL
visible	is_visible	1
searchable	is_searchable	0
filterable	is_filterable	0
comparable	is_comparable	0
visible_on_front	is_visible_on_front	0
wysiwyg_enabled	is_wysiwyg_enabled	0
is_html_allowed_on_front	is_html_allowed_on_front	0
visible_in_advanced_search	is_visible_in_advanced_search	0
filterable_in_search	is_filterable_in_search	0
used_in_product_listing	used_in_product_listing	0
used_for_sort_by	used_for_sort_by	0
apply_to	apply_to	null
position	position	0
used_for_promo_rules	is_used_for_promo_rules	0
is_used_in_grid	is_used_in_grid	0
is_visible_in_grid	is_visible_in_grid	0
is_filterable_in_grid	is_filterable_in_grid	0

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

To create Frontend model, create a class, inherited from Magento\Eav\Model\Entity\Attribute\Frontend\AbstractFrontend, and override getValue method.
Then, set frontend_model as the name of the newly created class.

```
class TestFrontend extends
\Magento\Eav\Model\Entity\Attribute\Frontend\AbstractFrontend
{
    public function getValue(\Magento\Framework\DataObject $object)
    {
     $attribute_code = $this->getAttribute()->getAttributeCode();
        $value = $object->getData($attribute_code);
        return nl2br(htmlspecialchars($value));
    }
}
```

The purpose of the interface is to decrease the code dependency (dependency inversion principle).

Identify the purpose and describe how to implement the interface for attribute source models.

The model is used for providing a list of attribute values for dropdown/multiselect attributes. Source model realization example:

```
class TestSource extends
\Magento\Eav\Model\Entity\Attribute\Source\AbstractSource
```

```
{
    public function getAllOptions()
    {
        if (!$this->_options) {
            $this->_options = [
                ['label' => __('Label 1'), 'value' => 'value 1'],
                ['label' => __('Label 2'), 'value' => 'value 2'],
                ['label' => __('Label 3'), 'value' => 'value 3'],
                ['label' => __('Label 4'), 'value' => 'value 4']
            ];
        }
        return $this->_options;
    }
}
```

For a given dropdown/multiselect attribute, how can you specify and manipulate its list of options?

The values are stored in the eav_attribute_option_value table if the source model is the Magento \ Eav \ Model \ Entity \ Attribute \ Source \ Table class or is inherited from it. The values can be added when creating an attribute in the option property, or you can call the Magento \ Eav \ Setup \ EavSetup-> addAttributeOption ($ option) method.
If the source model is not inherited from Magento \ Eav \ Model \ Entity \ Attribute \ Source \ Table, then, to change the list of values, you can:

- create a plugin on the source model on the getAllOptions method
- create a new source model, inherited from the mutable class, change the behavior of the getAllOptions method and specify the new source model in the attributes

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

Backend modules are created with a purpose to upload / save / delete / validate attribute values.

Example of attribute value validation:

```
class TestBackend extends
\Magento\Eav\Model\Entity\Attribute\Backend\AbstractBackend
{
    public function validate($object)
    {
      $attribute_code = $this->getAttribute()->getAttributeCode();
        $value = $object->getData($attribute_code);

        if ($value == 'test') {
             throw new \Magento\Framework\Exception\LocalizedException(__("Value
can't be test"));
        }

        return true;
    }
}
```

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

Setup scripts are applied to create or modify an attribute Vendor\Module\Setup\InstallData and Vendor\Module\Setup\UpgradeData.

Product attributes can also be created via the admin panel.

To add an attribute, use Magento\Eav\Setup\EavSetup->addAttribute($entityTypeId, $code, $attr) method, for modification - Magento\Eav\Setup\EavSetup->updateAttribute($entityTypeId, $id, $field, $value = null, $sortOrder = null). However, if addAttribute is called and the attribute already exists, then updateAttribute method will be called automatically.
Therefore, if you apply addAttribute method, there is no difference between adding a new attribute and modifying the existing one.

Section 6: Developing with Adminhtml

6.1 Describe common structure/architecture

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

Adminhtml and frontend are areas. Frontend displays the website to a user, while adminhtml is the website admin panel, aimed at those who manage it. In adminhtml, you can create / modify / delete objects and categories, manage orders and customers, configure the website, etc.

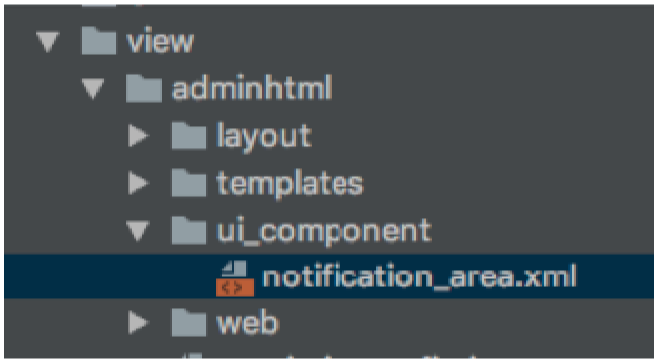
To log in to the admin panel, you need to know its basic URL (for example, https://www.website.com/admin_1pf3534g/) and then pass authentication.

Controller actions and blocks in adminhtml are located in the Adminhtml subfolder of Controller and Block folders respectively.
Controller actions and blocks in adminhtml inherit the classes, different from those inherited by actions and blocks in frontend.
In **adminhtml**:
Block inherits Magento\Backend\Block\Template
Action inherits Magento\Backend\App\AbstractAction
In **frontend**:
Block inherits Magento\Framework\View\Element\Template
Action inherits Magento\Framework\App\Action\Action

UI Components

Adminhtml components in the module view folder can contain an additional ui_component directory. The directory contains UI Components that speed up and simplify development, as well as minimize code duplicates. UI Components are also used in the out-of-the-box frontend in Checkout, but UI Components configuration is written in layout xml files, not in ui components xml files.

ui component file name matches the ui component name (except for the .xml suffix), that can be added into layout using the following instructions:
<uiComponent name="notification_area" aclResource="Magento_AdminNotification::show_list"/>

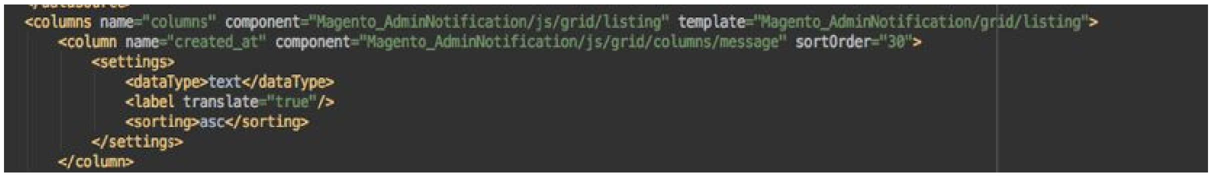


Ui_component folder contains one or several XML declarations for grids or forms.

Standard UI components come in two types:

- Basic
- Secondary

Basic components include form elements and listings.



```xml
<columns name="columns" component="Magento_AdminNotification/js/grid/listing" template="Magento_AdminNotification/grid/listing">
    <column name="created_at" component="Magento_AdminNotification/js/grid/columns/message" sortOrder="30">
        <settings>
            <dataType>text</dataType>
            <label translate="true"/>
            <sorting>asc</sorting>
        </settings>
    </column>
```

On the screenshot, you can see .xml file for template definition. It contains a link to the listing basic component connection, aimed at displaying grids, lists and fragments. Sorting, search and dividing into pages is added to the built-in functions, which are very helpful for new components and modules development.

Below is the list of extensions for the basic components listing:

- Filters
- Pagination

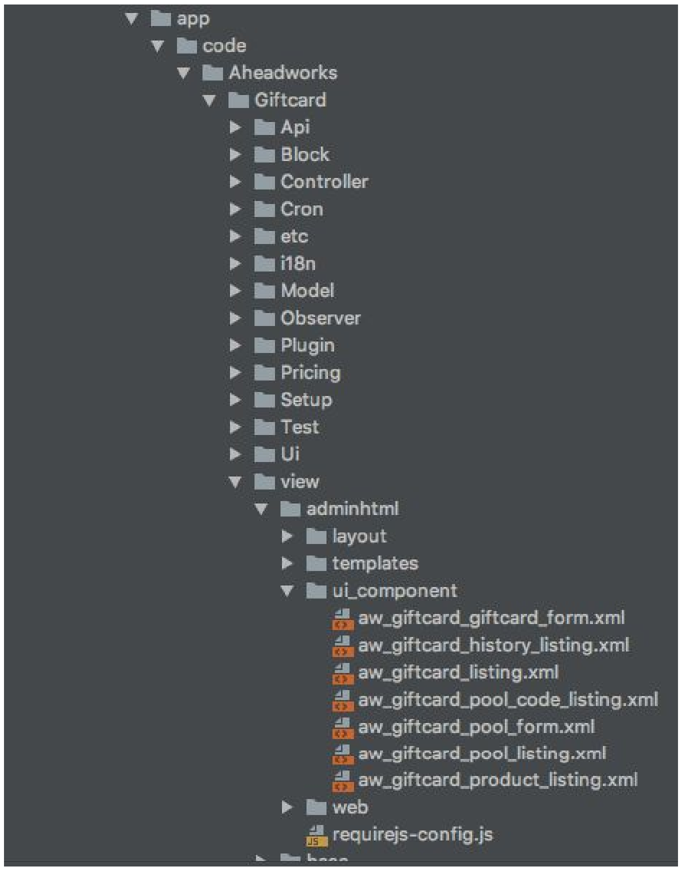
- Tabs
- Buttons and their functional load
- Tables
- Multiselectors
- Loaders
- Editors
- Headings

These are secondary components that serve as additional elements for expanding the basic components. Using them is not obligatory, but it is a good practice for a developer, for they will:

- Create a unified and clear interface
- Simplify and speed up the development process
- Create a simple and intellectually clear environment for support and expansion.

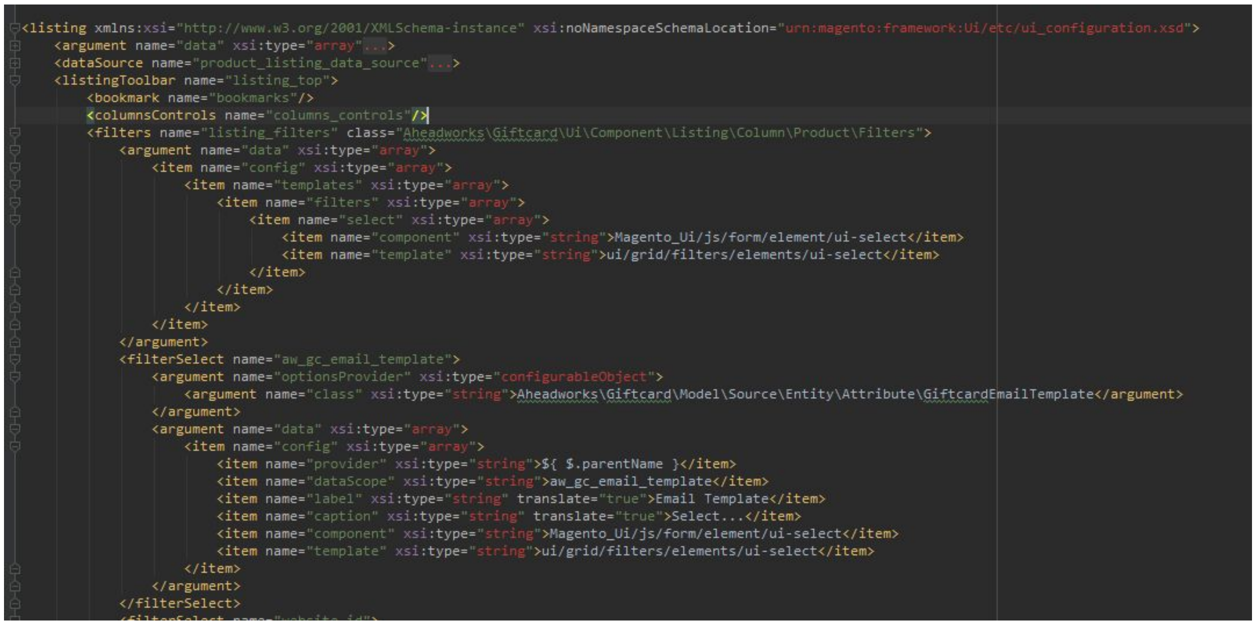
UI_Components enhances development, support and Magento interface expansion, allowing the components to work independently and with no loss to operation speed.

To connect Ui components to your module, located in the admin part of the interface, Magento uses a specialized xml file:
<module_dir>/view/adminhtml/ui_component/<ui_component_name>.xml.



```
▼ app
  ▼ code
    ▼ Aheadworks
      ▼ Giftcard
        ▶ Api
        ▶ Block
        ▶ Controller
        ▶ Cron
        ▶ etc
        ▶ i18n
        ▶ Model
        ▶ Observer
        ▶ Plugin
        ▶ Pricing
        ▶ Setup
        ▶ Test
        ▶ Ui
        ▼ view
          ▼ adminhtml
            ▶ layout
            ▶ templates
            ▼ ui_component
                aw_giftcard_giftcard_form.xml
                aw_giftcard_history_listing.xml
                aw_giftcard_listing.xml
                aw_giftcard_pool_code_listing.xml
                aw_giftcard_pool_form.xml
                aw_giftcard_pool_listing.xml
                aw_giftcard_product_listing.xml
            ▶ web
              requirejs-config.js
```

Then, depending on Magento version (we use Magento 2.2 for demonstration), we set up the necessary components in xml configuration:

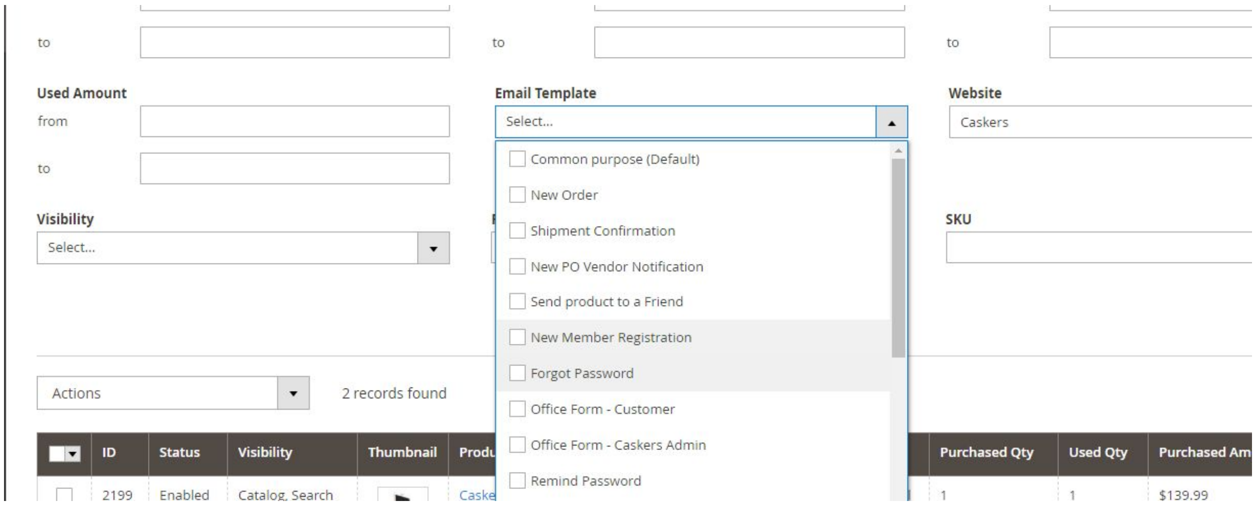


```xml
<listing xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="urn:magento:framework:Ui/etc/ui_configuration.xsd">
    <argument name="data" xsi:type="array"...>
    <dataSource name="product_listing_data_source"...>
    <listingToolbar name="listing_top">
        <bookmark name="bookmarks"/>
        <columnsControls name="columns_controls"/>
        <filters name="listing_filters" class="Aheadworks\Giftcard\Ui\Component\Listing\Column\Product\Filters">
            <argument name="data" xsi:type="array">
                <item name="config" xsi:type="array">
                    <item name="templates" xsi:type="array">
                        <item name="filters" xsi:type="array">
                            <item name="select" xsi:type="array">
                                <item name="component" xsi:type="string">Magento_Ui/js/form/element/ui-select</item>
                                <item name="template" xsi:type="string">ui/grid/filters/elements/ui-select</item>
                            </item>
                        </item>
                    </item>
                </item>
            </argument>
            <filterSelect name="aw_gc_email_template">
                <argument name="optionsProvider" xsi:type="configurableObject">
                    <argument name="class" xsi:type="string">Aheadworks\Giftcard\Model\Source\Entity\Attribute\GiftcardEmailTemplate</argument>
                </argument>
                <argument name="data" xsi:type="array">
                    <item name="config" xsi:type="array">
                        <item name="provider" xsi:type="string">${ $.parentName }</item>
                        <item name="dataScope" xsi:type="string">aw_gc_email_template</item>
                        <item name="label" xsi:type="string" translate="true">Email Template</item>
                        <item name="caption" xsi:type="string" translate="true">Select...</item>
                        <item name="component" xsi:type="string">Magento_Ui/js/form/element/ui-select</item>
                        <item name="template" xsi:type="string">ui/grid/filters/elements/ui-select</item>
                    </item>
                </argument>
            </filterSelect>
```

Configuration file has access to ListingToolbar component, introduced Magento 2.1 version. It is located in the area of Magento_Ui/grid names and is related to listing classes components. Listing Toolbar contains a number of additional components, configured directly in configuration file.
For example, Bookmark is responsible for displaying and saving the positions of all current elements, forms and buttons states in the component. It also accounts for saving and comparing information through the database, directly in ui_bookmark table.

Columns Control is responsible for displaying and hiding active list columns.

Filters component receives the information about the required secondary elements; in this case, it is an element of select form.



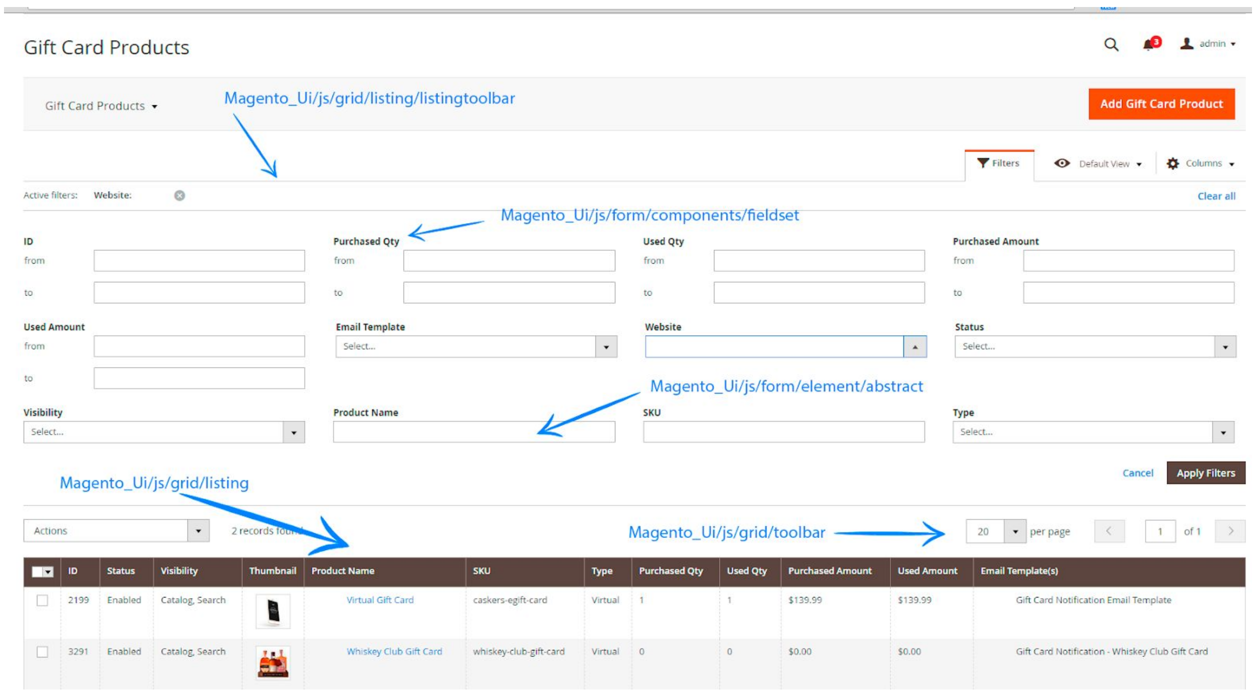
Using <argument name=”class”> attribute, we rewrite the standard element class into the one we need.

Using <argument name=”data”><item name =”name”> attribute, we set the available parameters at the selection list.

All components contain information about the current state and receive users’ values. For example, Columns Control receives the parameters on the minimal or maximal number of the displayed columns.

Afterward, XML configuration files are united into a single one, then into json and are passed into the client like this. Each configuration file of a single ui component, that includes the set parameters in a parent XML, overrides them as new ones.

As you can see, UI components library is the main instrument for the Mangento admin area, and a considerable part of the store's admin panel is built with it:



	ID	Status	Visibility	Thumbnail	Product Name	SKU	Type	Purchased Qty	Used Qty	Purchased Amount	Used Amount	Email Template(s)
	2199	Enabled	Catalog, Search		Virtual Gift Card	caskers-egift-card	Virtual	1	1	$139.99	$139.99	Gift Card Notification Email Template
	3291	Enabled	Catalog, Search		Whiskey Club Gift Card	whiskey-club-gift-card	Virtual	0	0	$0.00	$0.00	Gift Card Notification - Whiskey Club Gift Card

Obtaining UI components skills allows to significantly speed up development and expansion of Magento store interface, as well as to unify its structure for creating a more intuitive area for further support and development.

6.2 Define form and grid widgets

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

Form

Contains modified fields of a certain entity. To create a form, you need to:

1. Create form xml file of ui component configuration: <module_dir>/view/<area>/ui_component/<ui_component_name>.xml
2. Create DataProvider class in the module and define this class in XML configuration file of UI component.
3. Add a set of fields (the Fieldset component with the component of the Field) for entity or to implement the upload of meta info in the DataProvide
4. Add UI component in layout: <uiComponent name="<ui_component_name>"/>

Grid

Contains the list of entities, filters, bookmarks, paging, columns controls. Allows to search entity, modify list sorting, hide and add columns, change their order. To create a grid, you need to:

1. Create listing xml file of UI component configuration: <module_dir>/view/<area>/ui_component/<ui_component_name>.xml
2. Set DataSource
3. Add filter, columns, toolbar, ...
4. Add ui component into layout: <uiComponent name="<ui_component_name>"/>

Describe the grid and form workflow. How is data provided to the grid or form?

At the server:

1. Layout loads UI component
2. XML files of UI component are searched in each enabled module (<module_dir>/view/<area>/ui_component/<ui_component_name>.xml)
3. All found files of UI components are merged in a single configuration object
4. Configuration and definition.xml are merged. Objects from definition.xml have lower priority than single configuration object, received in the previous step
5. The received configuration is transformed into JSON and embedded into the page with Magento\Ui\TemplateEngine\Xhtml\Result class

As a result, we get the following code:

```
<script type="text/x-magento-init">
   {
      "*":{
         "Magento_Ui/js/core/app":{
            "types":{ ... },
            "components":{ ... }
         }
      }
   }
</script>
```

At the client side:

1. RequireJS loads Magento_Ui/js/core/app and passes configuration as a parameter
2. Magento_Ui/js/core/app calls Magento_Ui/js/core/renderer/layout and passes it the configuration
3. layout.js creates instances of UI components and applies the configuration to it
4. HTML template rendering with knockout.js and binding of the component to the template is performed

The data for grid and form is loaded with the help of DataProvider. The difference lies in the moment of information upload.

For the form component, the data of an entity are passed inside the current page's html code in the <script type = "text / x-magento-init"> ... </script> tag.
For the grid component, the data is loaded through an additional ajax request for the controller action **mui / index / render**, processed by the \ Magento \ Ui \ Controller \ Adminhtml \ Index \ Render class for adminhtml area and \ Magento \ Ui \ Controller \ Index \ Render for frontend area .

How can this process be customized or extended?

You can change the form data the following ways:

1. Create a custom DataProvider and specify it in xml file of UI component.
2. Create a plugin for DataProvider::getData() method
3. Create Modifier class and set it in di.xml, if this DataProvider support Modifiers (for example, like \Magento\Catalog\Ui\DataProvider\Product\Form\ProductDataProvider)

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

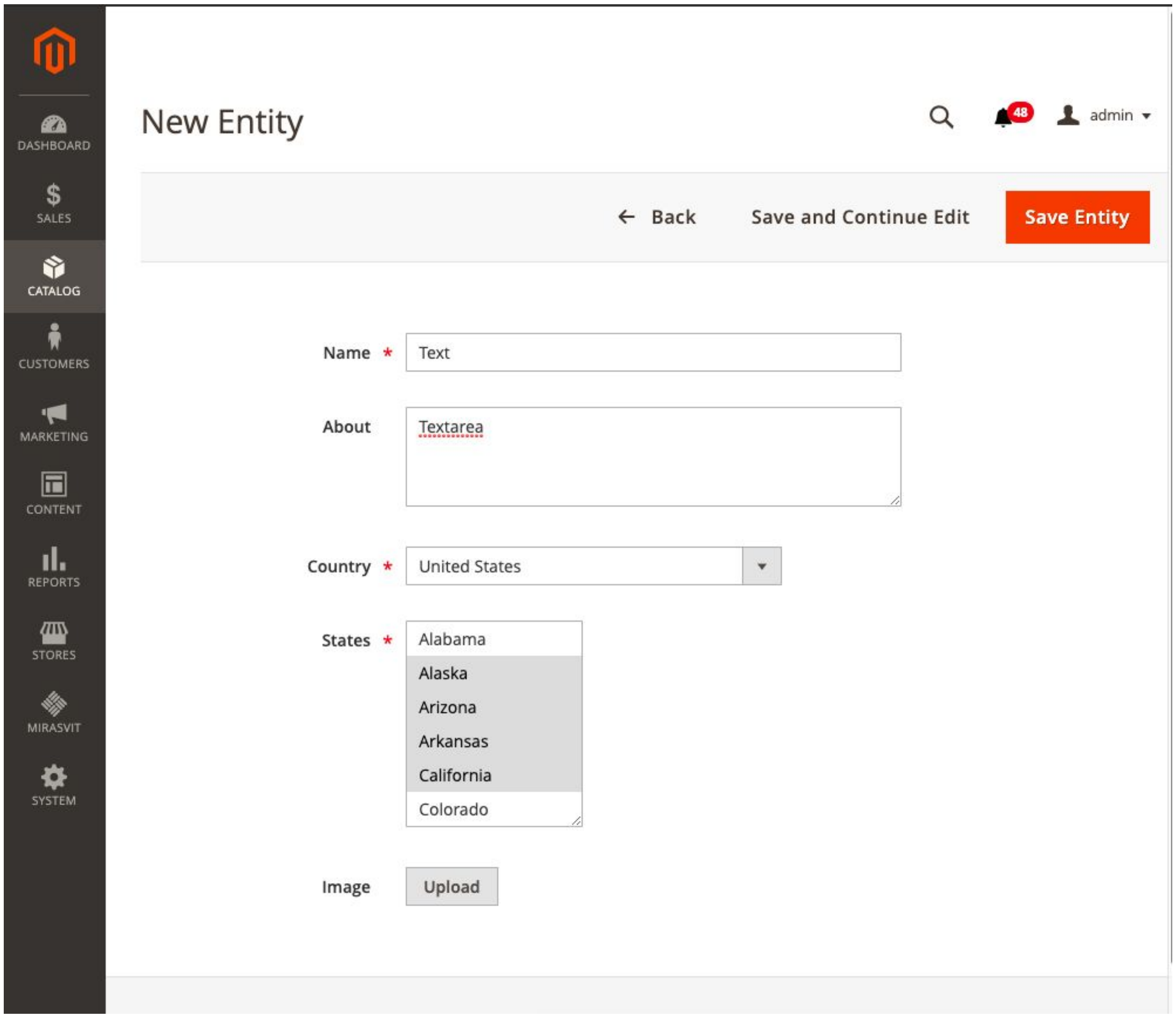
Let us consider the examine of a form. It includes text, textarea, select, multiselect, image. The data is loaded via the Vendor\Module\Model\MyEntity\DataProvider class.

```
<?xml version="1.0" ?>
<form xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:noNamespaceSchemaLocation="urn:magento:module:Magento_Ui:etc/ui_configuration.x
sd">
```

```
    <argument name="data" xsi:type="array">
        <item name="js_config" xsi:type="array">
            <item name="provider"
xsi:type="string">my_form.my_entity_form_data_source</item>
            <item name="deps"
xsi:type="string">my_form.my_entity_form_data_source</item>
        </item>
        <item name="label" translate="true" xsi:type="string">General
Information</item>
        <item name="config" xsi:type="array">
            <item name="dataScope" xsi:type="string">data</item>
            <item name="namespace" xsi:type="string">my_form</item>
        </item>
        <item name="template" xsi:type="string">templates/form/collapsible</item>
        <item name="buttons" xsi:type="array">
            <item name="back"
xsi:type="string">Vendor\Module\Block\Adminhtml\MyEntity\Edit\BackButton</item>
            <item name="delete"
xsi:type="string">Vendor\Module\Block\Adminhtml\MyEntity\Edit\DeleteButton</item>
            <item name="save"
xsi:type="string">Vendor\Module\Block\Adminhtml\MyEntity\Edit\SaveButton</item>
            <item name="save_and_continue"
xsi:type="string">Vendor\Module\Block\Adminhtml\MyEntity\Edit\SaveAndContinueButton
</item>
        </item>
    </argument>
    <dataSource name="my_entity_form_data_source">
        <argument name="dataProvider" xsi:type="configurableObject">
            <argument name="class"
xsi:type="string">Vendor\Module\Model\MyEntity\DataProvider</argument>
            <argument name="name"
xsi:type="string">my_entity_form_data_source</argument>
            <argument name="primaryFieldName" xsi:type="string">id</argument>
            <argument name="requestFieldName" xsi:type="string">id</argument>
            <argument name="data" xsi:type="array">
                <item name="config" xsi:type="array">
                    <item name="submit_url" path="*/*/save" xsi:type="url"/>
                </item>
            </argument>
        </argument>
        <argument name="data" xsi:type="array">
            <item name="js_config" xsi:type="array">
                <item name="component"
xsi:type="string">Magento_Ui/js/form/provider</item>
            </item>
        </argument>
    </dataSource>
    <fieldset name="General">
        <argument name="data" xsi:type="array">
            <item name="config" xsi:type="array">
                <item name="label" xsi:type="string"/>
            </item>
        </argument>

        <field name="name">
            <argument name="data" xsi:type="array">
                <item name="config" xsi:type="array">
                    <item name="dataType" xsi:type="string">text</item>
                    <item name="label" translate="true"
xsi:type="string">Name</item>
                    <item name="formElement" xsi:type="string">input</item>
                    <item name="source" xsi:type="string">MyEntity</item>
                    <item name="sortOrder" xsi:type="number">50</item>
                    <item name="validation" xsi:type="array">
                        <item name="required-entry" xsi:type="boolean">true</item>
                    </item>
                </item>
            </argument>
        </field>

        <field name="about">
            <argument name="data" xsi:type="array">
                <item name="config" xsi:type="array">
                    <item name="dataType" xsi:type="string">text</item>
                    <item name="label" translate="true"
xsi:type="string">About</item>
                    <item name="formElement" xsi:type="string">textarea</item>
                    <item name="source" xsi:type="string">MyEntity</item>
                    <item name="sortOrder" xsi:type="number">65</item>
                </item>
            </argument>
        </field>

        <field name="country_id">
            <argument name="data" xsi:type="array">
                <item name="options"
xsi:type="object">Magento\Directory\Model\Config\Source\Country</item>
                <item name="config" xsi:type="array">
                    <item name="dataType" xsi:type="string">text</item>
                    <item name="label" xsi:type="string"
translate="true">Country</item>
                    <item name="formElement" xsi:type="string">select</item>
                    <item name="source" xsi:type="string">MyEntity</item>
                    <item name="sortOrder" xsi:type="number">70</item>
                    <item name="component"
xsi:type="string">Magento_Ui/js/form/element/country</item>
                    <item name="validation" xsi:type="array">
                        <item name="required-entry" xsi:type="boolean">true</item>
                    </item>
                </item>
            </argument>
        </field>

        <field name="region_ids">
            <argument name="data" xsi:type="array">
                <item name="options"
xsi:type="object">Magento\Directory\Model\ResourceModel\Region\Collection</item>
                <item name="config" xsi:type="array">
                    <item name="dataType" xsi:type="string">int</item>
                    <item name="dataScope" xsi:type="string">region_ids</item>
                    <item name="label" xsi:type="string"
translate="true">States</item>
                    <item name="formElement" xsi:type="string">multiselect</item>
                    <item name="source" xsi:type="string">MyEntity</item>
                    <item name="sortOrder" xsi:type="number">80</item>
                    <item name="validation" xsi:type="array">
                        <item name="required-entry" xsi:type="boolean">true</item>
                    </item>
                    <item name="filterBy" xsi:type="array">
                        <item name="target" xsi:type="string">
                            <![CDATA[${ $.provider }:${ $.parentScope
}.country_id]]></item>
                        <item name="field" xsi:type="string">country_id</item>
                    </item>
                </item>
            </argument>
        </field>

        <field name="image">
            <argument name="data" xsi:type="array">
                <item name="config" xsi:type="array">
                    <item name="label" xsi:type="string">Image</item>
                    <item name="visible" xsi:type="boolean">true</item>
                    <item name="source" xsi:type="string">MyEntity</item>
                    <item name="formElement" xsi:type="string">fileUploader</item>
                    <item name="elementTmpl"
xsi:type="string">ui/form/element/uploader/uploader</item>
                    <item name="previewTmpl"
```

```
xsi:type="string">Caskers_MyEntity/image-preview</item>
                    <item name="sortOrder" xsi:type="number">90</item>
                    <item name="uploaderConfig" xsi:type="array">
                        <item name="url" xsi:type="url"
path="vendor_module/upload/image"/>
                    </item>
                </item>
            </argument>
        </field>
    </fieldset>
</form>
```

Result:



New Entity

Name * Text

About Textarea

Country * United States

States * Alabama, Alaska, Arizona, Arkansas, California, Colorado

Image

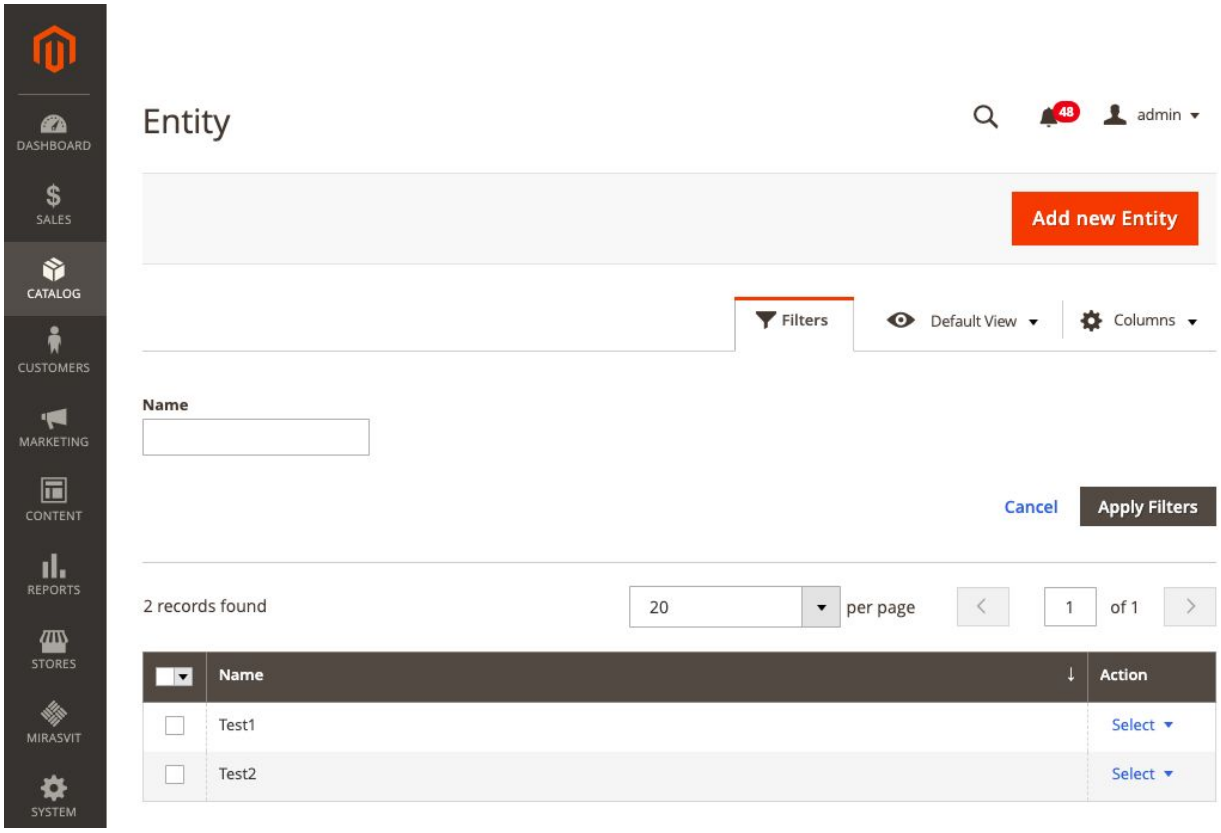
Let us consider the grid. As DataProvider, Magento\Framework\View\Element\UiComponent\DataProvider\DataProvider is used. The collection for grid is set in di.xml:

```
<?xml version="1.0" ?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:noNamespaceSchemaLocation="urn:magento:framework:ObjectManager/etc/config.xsd">
    <type
name="Magento\Framework\View\Element\UiComponent\DataProvider\CollectionFactory">
        <arguments>
            <argument name="collections" xsi:type="array">
                <item name="my_listing_grid_data_source" xsi:type="string">
                    Vendor\Module\Model\ResourceModel\MyEntity\Grid\Collection
                </item>
            </argument>
        </arguments>
    </type>
    <virtualType name="Vendor\Module\Model\ResourceModel\MyEntity\Grid\Collection"
type="Magento\Framework\View\Element\UiComponent\DataProvider\SearchResult">
        <arguments>
            <argument name="mainTable"
xsi:type="string">vendor_my_entity</argument>
            <argument name="resourceModel"
xsi:type="string">Vendor\Module\Model\ResourceModel\MyEntity\Collection</argument>
        </arguments>
    </virtualType>
</config>
```

UI component GRID file:

```
<?xml version="1.0" ?>
<listing xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:noNamespaceSchemaLocation="urn:magento:module:Magento_Ui:etc/ui_configuration.x
sd">
    <argument name="context" xsi:type="configurableObject">
        <argument name="class"
xsi:type="string">Magento\Framework\View\Element\UiComponent\Context</argument>
        <argument name="namespace" xsi:type="string">my_listing_index</argument>
    </argument>
    <argument name="data" xsi:type="array">
        <item name="js_config" xsi:type="array">
            <item name="provider"
xsi:type="string">my_listing_index.my_listing_grid_data_source</item>
            <item name="deps"
xsi:type="string">my_listing_index.my_listing_grid_data_source</item>
        </item>
        <item name="spinner" xsi:type="string">my_listing_columns</item>
        <item name="buttons" xsi:type="array">
            <item name="add" xsi:type="array">
                <item name="name" xsi:type="string">add</item>
                <item name="label" translate="true" xsi:type="string">Add new
Entity</item>
                <item name="class" xsi:type="string">primary</item>
                <item name="url" xsi:type="string">*/*/new</item>
            </item>
        </item>
    </argument>
    <dataSource name="my_listing_grid_data_source">
        <argument name="dataProvider" xsi:type="configurableObject">
            <argument name="class" xsi:type="string">

Magento\Framework\View\Element\UiComponent\DataProvider\DataProvider
            </argument>
            <argument name="name"
xsi:type="string">my_listing_grid_data_source</argument>
            <argument name="primaryFieldName" xsi:type="string">id</argument>
            <argument name="requestFieldName" xsi:type="string">id</argument>
            <argument name="data" xsi:type="array">
                <item name="config" xsi:type="array">
                    <item name="component"
xsi:type="string">Magento_Ui/js/grid/provider</item>
                    <item name="update_url" path="mui/index/render"
xsi:type="url"/>
                    <item name="storageConfig" xsi:type="array">
                        <item name="indexField" xsi:type="string">id</item>
                    </item>
                </item>
            </argument>
        </argument>
    </dataSource>
    <listingToolbar name="listing_top">
        <argument name="data" xsi:type="array">
            <item name="config" xsi:type="array">
                <item name="sticky" xsi:type="boolean">true</item>
            </item>
        </argument>
        <bookmark name="bookmarks"/>
        <columnsControls name="columns_controls"/>
        <filters name="listing_filters"/>
        <paging name="listing_paging"/>
    </listingToolbar>
    <columns name="my_listing_columns">
        <selectionsColumn name="ids">
            <argument name="data" xsi:type="array">
                <item name="config" xsi:type="array">
                    <item name="indexField" xsi:type="string">id</item>
                </item>
            </argument>
        </selectionsColumn>
        <column name="name">
            <argument name="data" xsi:type="array">
                <item name="config" xsi:type="array">
                    <item name="filter" xsi:type="string">text</item>
                    <item name="sorting" xsi:type="string">asc</item>
                    <item name="label" translate="true"
xsi:type="string">Name</item>
                </item>
            </argument>
        </column>
        <actionsColumn
class="Vendor\Module\Ui\Component\Listing\Column\MyEntityActions" name="actions">
            <argument name="data" xsi:type="array">
                <item name="config" xsi:type="array">
                    <item name="indexField" xsi:type="string">id</item>
                </item>
            </argument>
        </actionsColumn>
    </columns>
</listing>
```

Result:



	Name	Action
☐	Test1	Select
☐	Test2	Select

6.3 Define system configuration XML and configuration scope

Define basic terms and elements of system configuration XML, including scopes. How would you add a new system configuration option?

Magento 2 allows to extend the system configuration with modules and integrate the custom module configuration into the Magento 2 system menu.

To add such custom configurations, use <module_dir>/etc/adminhtml/system.xml file.

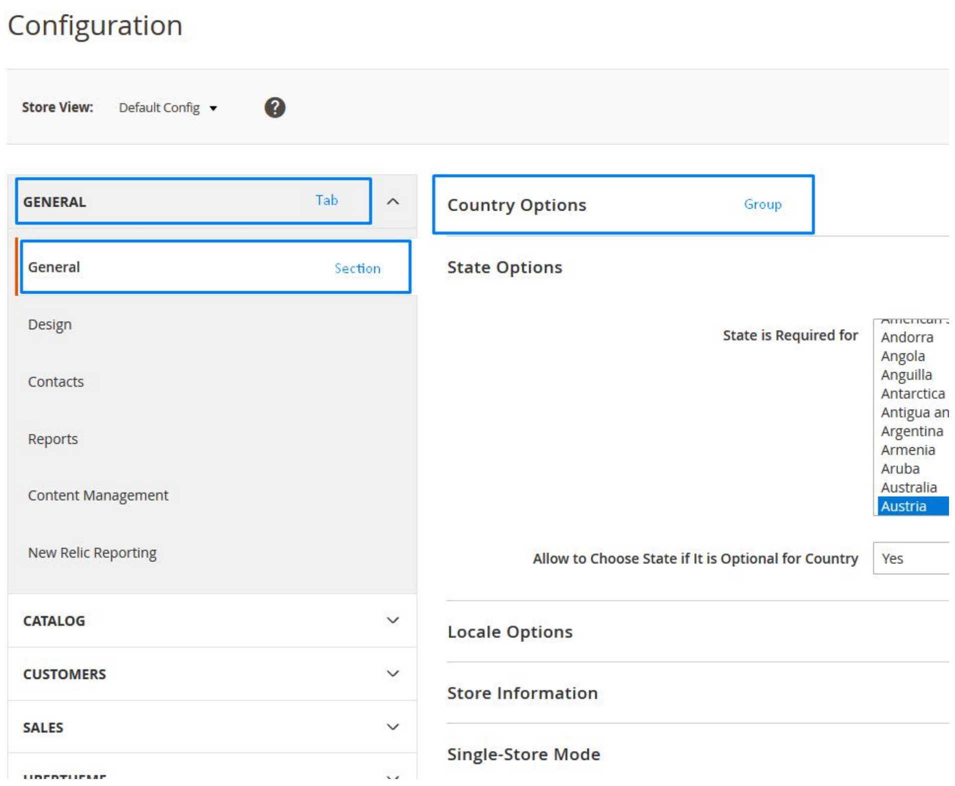
```
<?xml version="1.0"?>
```

```
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:module:Magento_Config:etc/system_file.xs
d">
    <system>
        <tab id="custom_tab" translate="label" sortOrder="100">
            <label>Custom Tab</label>
        </tab>
        <section id="custom_section" translate="label" type="text" sortOrder="100"
showInDefault="1" showInWebsite="1" showInStore="1">
            <label>Custom Config Section</label>
            <tab>custom_tab</tab>
            <group id="general" translate="label" type="text" sortOrder="10"
showInDefault="1" showInWebsite="1" showInStore="1">
                <label>General</label>
<field id="yesno_dropdown" translate="label" type="select" sortOrder="10"
showInDefault="1" showInWebsite="1" showInStore="1">
<label>Custom Yes/No Dropdown</label>
<source_model>Magento\Config\Model\Config\Source\Yesno</source_model>
                </field>
                <field id="custom_dropdown" translate="label" type="select"
sortOrder="10" showInDefault="1" showInWebsite="1" showInStore="1">
                    <label>Dropdown with custom source model example</label>

<source_model>Vendor\Module\Model\Config\Source\Custom</source_model>
                </field>
                <field id="custom_text" translate="label" type="text"
sortOrder="20" showInDefault="1" showInWebsite="1" showInStore="1">
                    <label>Custom Text</label>
                </field>
                <field id="logo" translate="label" type="image" sortOrder="30"
showInDefault="1" showInWebsite="1" showInStore="1">
                    <label>Custom Image</label>

<backend_model>Magento\Config\Model\Config\Backend\Image</backend_model>
                    <upload_dir config="system/filesystem/media"
scope_info="1">logo</upload_dir>
                    <base_url type="media" scope_info="1">logo</base_url>
                    <comment><![CDATA[Allowed file types: jpeg, gif,
png.]]></comment>
                </field>
                <field id="depends_example" translate="label" type="text"
sortOrder="40" showInDefault="1" showInWebsite="1" showInStore="1">
                    <label>Dependant text field example with validation</label>
                    <depends>
                        <field id="*/*/custom_dropdown">1</field>
                    </depends>
                    <validate>validate-no-empty</validate>
                </field>
                <field id="custom_textarea" translate="label" type="textarea"
sortOrder="50" showInDefault="1" showInWebsite="1" showInStore="1">
                    <label>Custom Textarea</label>
                </field>
               <field id="custom_secret" type="obscure" translate="label"
sortOrder="70" showInDefault="1" showInWebsite="1" showInStore="1">
                    <label>Custom Secret Field</label>

<backend_model>Magento\Config\Model\Config\Backend\Encrypted</backend_model>
                </field>
            </group>
        </section>
    </system>
</config>
```

Now let us examine the code details:

```
<tab id="custom_tab" translate="label" sortOrder="100">
    <label>Custom Tab</label>
</tab>
```

This xml code allows to add a new line into Magento 2 configuration menu



Then

```
<section id="custom_section" translate="label" type="text"
sortOrder="100" showInDefault="1" showInWebsite="1" showInStore="1">
    <label>Custom Config Section</label>
    <tab>custom_tab</tab>
    ...
</section>
```

This code adds a new section and assigns it to custom_tab tab. Here, we can use any tab from the existing ones.
showInDefault="1", showInWebsite="1" and showInStore="1" parameters set the scope where our section will be displayed.

Setting Default Value

To set default value for any of the custom settings, create a <module_dir>/etc/config.xml file.

```
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:module:Magento_Store:etc/c
onfig.xsd">
    <default>
        <custom_section>
            <general>
                <yesno_dropdown>1</enable>
                <custom_text>Test Value</display_text>
            </general>
        </helloworld>
    </default>
</config>
```

What is the difference in this process for different option types (secret, file)?

Simple Text Field

```
<field id="custom_text" translate="label" type="text" sortOrder="20"
showInDefault="1" showInWebsite="1" showInStore="1">
    <label>Custom Text</label>
</field>
```

Yes/No Dropdown

```
<field id="yesno_dropdown" translate="label" type="select"
sortOrder="10" showInDefault="1" showInWebsite="1" showInStore="1">
    <label>Custom Yes/No Dropdown</label>
```

```
<source_model>Magento\Config\Model\Config\Source\Yesno</source_model>
</field>
```

Dropdown With The Custom Source model

```
<field id="custom_dropdown" translate="label" type="select"
sortOrder="10" showInDefault="1" showInWebsite="1" showInStore="1">
    <label>Dropdown with custom source model example</label>

<source_model>Vendor\Module\Model\Config\Source\Custom</source_model>
</field>
```

And our custom model can look like this:
<module_dir>/Model/Config/Source/Custom.php

```
<?php
namespace Vendor\Model\Model\Config\Source;
class Custom implements \Magento\Framework\Option\ArrayInterface
{
    /**
     * @return array
     */
    public function toOptionArray()
    {
        return [
            ['value' => 0, 'label' => __('Zero')],
            ['value' => 1, 'label' => __('One')],
            ['value' => 2, 'label' => __('Two')],
        ];
    }
}
```

File Upload

```
<field id="logo" translate="label" type="image" sortOrder="30"
showInDefault="1" showInWebsite="1" showInStore="1">
   <label>Custom Image</label>
<backend_model>Magento\Config\Model\Config\Backend\Image</backend_mod
el>
   <upload_dir config="system/filesystem/media"
scope_info="1">logo</upload_dir>
   <base_url type="media" scope_info="1">logo</base_url>
<comment><![CDATA[Allowed file types: jpeg, gif, png.]]></comment>
</field>
```

Dependent Field

```
<field id="depends_example" translate="label" type="text"
sortOrder="40" showInDefault="1" showInWebsite="1" showInStore="1">
    <label>Dependant text field example with validation</label>
    <depends>
        <field id="*/*/yesno_dropdown">1</field>
    </depends>
    <validate>validate-no-empty</validate>
</field>
```

`<validate>validate-no-empty</validate>` validates that the field is not empty when the configuration is saved.
depends tag allows to display this field only if at yesno_dropdown the Yes value was selected.

Textarea

```
<field id="custom_textarea" translate="label" type="textarea"
sortOrder="50" showInDefault="1" showInWebsite="1" showInStore="1">
    <label>Custom Textarea</label>
```

```
</field>
```

Secret Field

```
<field id="custom_secret" type="obscure" translate="label"
sortOrder="70" showInDefault="1" showInWebsite="1" showInStore="1">
    <label>Custom Secret Field</label>

<backend_model>Magento\Config\Model\Config\Backend\Encrypted</backend
_model>
</field>
```

type="obscure" hides field's value from the frontend, but the information from it will still be saved as plain text. Setting Magento\Config\Model\Config\Backend\Encrypted as a backend model allows to encrypt the data in the database.
Most of the source models are located in app/code/Magento/Config/Model/Config/Source and backend models are located in app/code/Magento/Config/Model/Config/Backend.

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

To access system configuration options programmatically, create a class. Inside it, create a method for obtaining the value of the needed configuration and determine the constant that contains the path to the configuration value. Example:

<module_dir>/Helper/Config.php

```
<?php

namespace BelVG\Test\Helper;

use Magento\Framework\App\Helper\AbstractHelper;
use Magento\Store\Model\ScopeInterface;
```

```
class Config extends AbstractHelper
{
    const XML_PATH_TEST_VALUE = 'belvg/settings/test_value';

    /**
     * @return string
     */
    public function getTestValue()
    {
        return (string)
$this->scopeConfig->getValue(self::XML_PATH_TEST_VALUE,
ScopeInterface::SCOPE_STORE);
    }
}
```

If your configuration values are encrypted and stored in the database, set **backend_model** in etc/config.xml file so that scopeConfig->getValue would return the decrypted configuration values. Example:

```
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:module:Magento_Store:etc/c
onfig.xsd">
    <default>
        <belvg>
            <settings>
                <test_value
backend_model="Magento\Config\Model\Config\Backend\Encrypted"/>
            </settings>
        </belvg>
    </default>
</config>
```

6.4 Utilize ACL to set menu items and permissions

Describe how to set up a menu item and permissions. How would you add a new menu item in a given tab?

Magento 2 allows to customize and add new points to backend menu.
To add new menu, use the following file
<module_dir>/etc/adminhtml/menu.xml

```
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:module:Magento_Backend:etc
/menu.xsd">
    <menu>
        <add id="Vendor_Module::second_level" title="Second Level
Menu" module="Vendor_Module" sortOrder="10"
action="Vendor_Module/action_path"
resource="Magento_Backend::content"
parent="Magento_Backend::system_design
l"/>
    </menu>
</config>
```

where

- id - record identifier; should be unique. {Vendor_Module}::{menu_description}.
- title - displayed title
- module - sets the module that module item belongs to in Vendor_Module format.

- sortOrder - sets the position of a menu item. The options with smaller values will be displayed higher.
- parent - id of other menu item. Defines the menu as embedded.
- action - url pages, which the menu item refers to.
- resource - sets ACL rule, necessary for viewing a certain menu item.

How would you add a new tab in the Admin menu?

The process of adding a new tab is similar to adding a new item, but the item, responsible for a new tab, does not contain a parent item.

```
<module_dir>/etc/adminhtml/menu.xml
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:module:Magento_Backend:etc
/menu.xsd">
    <menu>
        <add id="Vendor_Module::first_level" title="First Level Menu"
module="Vendor_Module" sortOrder="51"
resource="Magento_Backend::content"/>
        <add id="Vendor_Module::second_level" title="Second Level
Menu" module="Vendor_Module" sortOrder="10"
action="Vendor_Module/action_path"
resource="Magento_Backend::content"
parent="Vendor_Module::first_level"/>
    </menu>
</config>
```

How do menu items relate to ACL permissions?

To restrict access to a certain menu item with a certain ACL rule, insert it in the resource parameter during the menu configuration:

<module_dir>/etc/adminhtml/menu.xml

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

```
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:module:Magento_Backend:etc
/menu.xsd">
    <menu>
        <add id="Vendor_Module::first_level" title="First Level Menu"
module="Vendor_Module" sortOrder="51"
resource="Vendor_Module::acl1"/>
        <add id="Vendor_Module::second_level" title="SEcond Level
Menu" module="Vendor_Module" sortOrder="10"
action="Vendor_Module/action_path" resource="Vendor_Module::acl2"
parent="Vendor_Module::first_level"/>
    </menu>
</config>
```

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

Magento 2 allows to separate the Admin Users permissions and create admin users groups with a variety of permissions.

Defining ACL

Use <module_dir>/etc/acl.xml file to create a new ACL rule.

```
<?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">
                <resource id="Vendor_Module::acl1" title="ACL Parent
Rule" sortOrder="51">
```

```
                    <resource id="Vendor_Module::acl2" title="ACL
Child Rule" sortOrder="10"/>
                </resource>
            </resource>
        </resources>
    </acl>
</config>
```

where,

- id - resource identifier. Applied in menu and actions configuration for identifying the resource. It should be unique and set in the following format: Vendor_Module::resource_name.
- title - resource title
- sortOrder - resource order in resource tree

System section > Permissions > User Roles contains all the available roles and their permissions.

System section > Permissions > All Users allows to create new users or modify the current users, as well as assign user roles.

How can you do that programmatically?

To restrict permissions to actions with ACL, override _isAllowed method or ADMIN_RESOURCE constant in the class of this action.

```
protected function _isAllowed()
{
     return $this->_authorization->isAllowed('Vendor_Module::acl2');
}
or
const ADMIN_RESOURCE = 'Vendor_Module::acl2';
```

To create user role, use classes Magento\Authorization\Model\RoleFactory and Magento\Authorization\Model\RulesFactory

```
      $role = $this->roleFactory->create();
      $role->setParentId(0)
            ->setTreeLevel(1)
            ->setSortOrder(1)
            ->setRoleType(Group::ROLE_TYPE)
            ->setUserId(0)
            ->setUserType(UserContextInterface::USER_TYPE_ADMIN)
            ->setRoleName('Example Administrator');
      $role->save();
      /** @var \Magento\Authorization\Model\Rules $rule */
      $rule = $this->rulesFactory->create();
      $rule->setRoleId($role->getId())
            ->setResourceId($this->rootResource->getId())
            ->setPrivilegies(null)
            ->setPermission('allow');
      $rule->save();
```

To create a user, create a class Magento\User\Model\UserFactory:

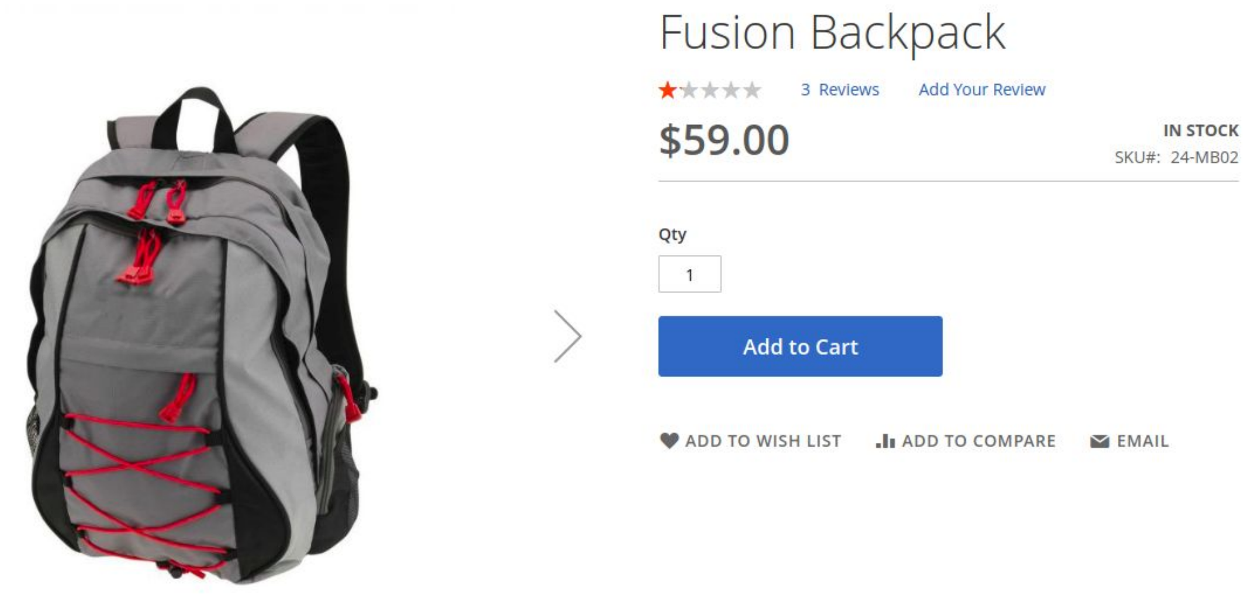
```
            $adminUser = $this->userFactory->create();
            $adminUser->setRoleId(ROLE_ID)
                  ->setEmail('admin' . $i . '@example.com')
                  ->setFirstName('Firstname')
                  ->setLastName('Lastname')
                  ->setUserName('admin' . $i)
                  ->setPassword('123123q')
                  ->setIsActive(1)
                  ->save();
```

7.1 Demonstrate ability to use products and product types

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

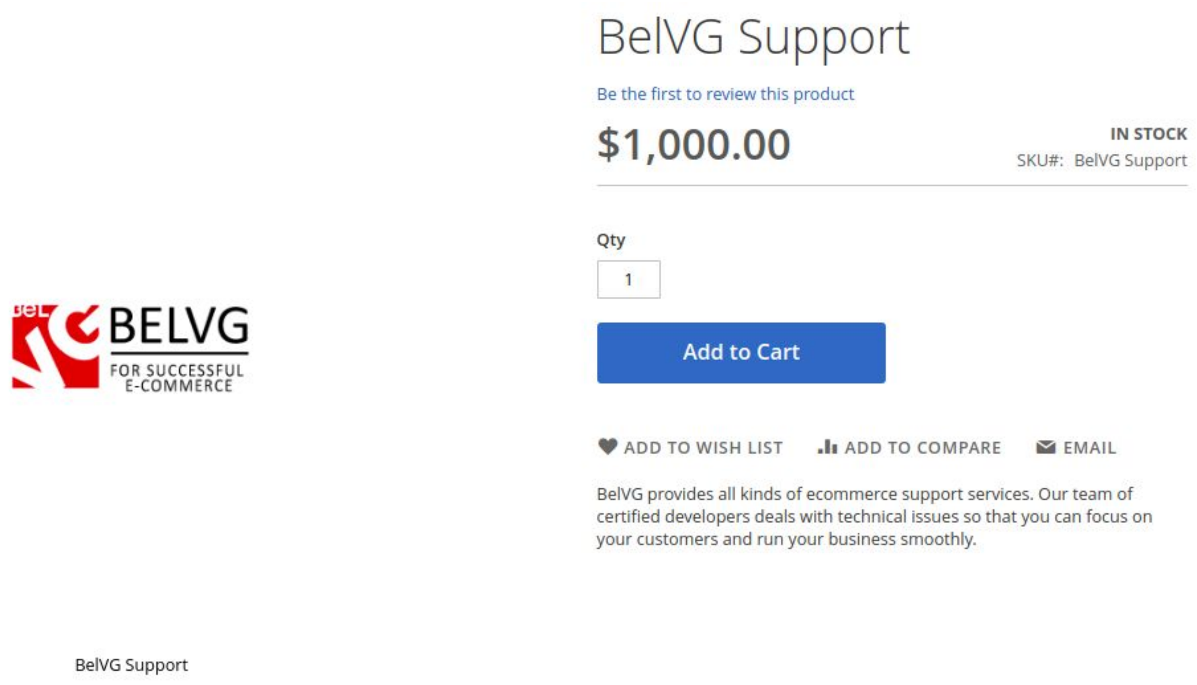
In the out-of-the-box Magento 2 Community Edition, there are six product types:

1. Simple Product. This is a basic and the most popular product type. A single simple product corresponds to a single physically existing product to a unique SKU (Store Keeping Unit).



Simple Product

2. Virtual Product. This product type is for the products that do not exist physically. Paid subscriptions, services, insurances, etc. are examples of Virtual products.


BelVG Support

Be the first to review this product

$1,000.00

IN STOCK

SKU#: BelVG Support

Qty

1

Add to Cart

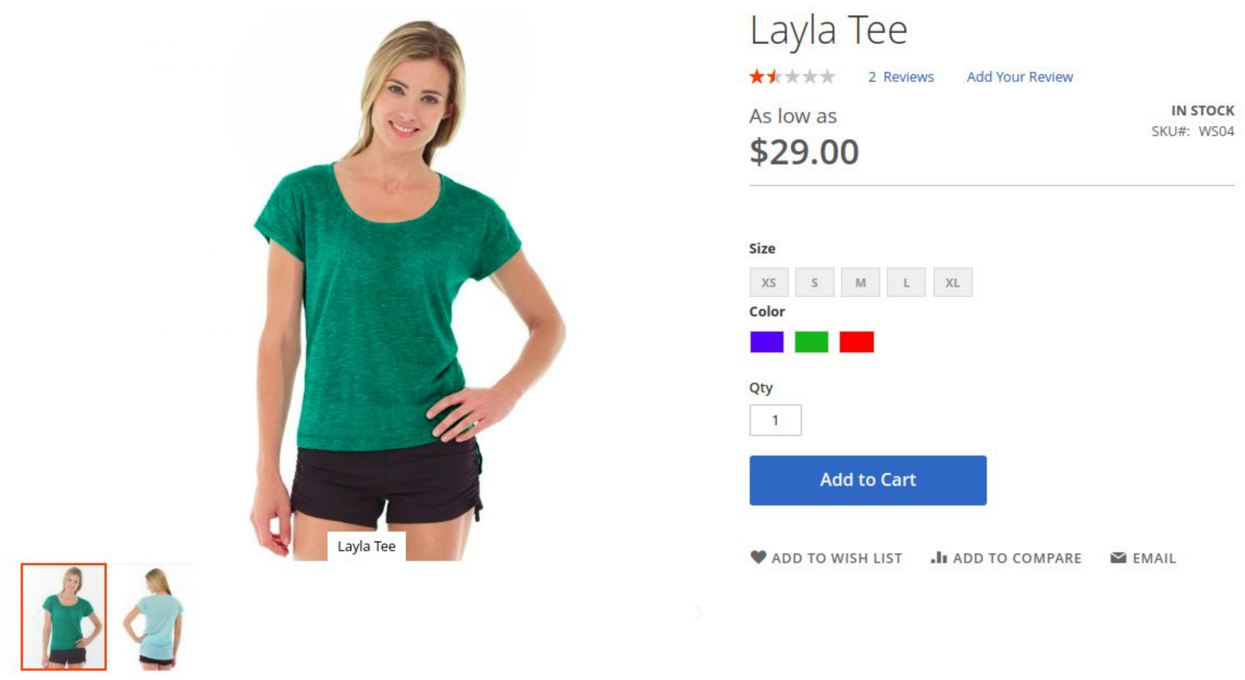
ADD TO WISH LIST ADD TO COMPARE EMAIL

BelVG provides all kinds of ecommerce support services. Our team of certified developers deals with technical issues so that you can focus on your customers and run your business smoothly.

BelVG Support

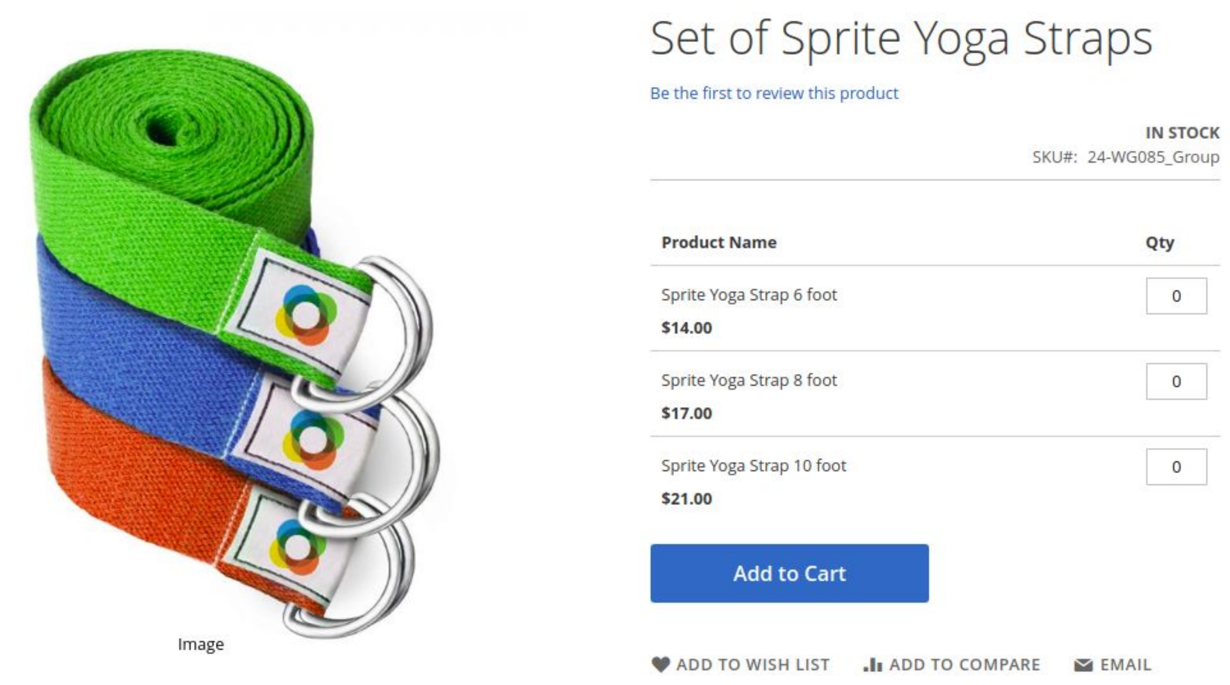
Virtual Product

3. Configurable Product. The type allows to create products with a list of options, for example, for example, a T-shirt in different colors and sizes. Each product option of such configurable product corresponds to a single Simple Product with a unique SKU, allowing the retailer to keep track of each product option stock balance.



Configurable Product

4. Grouped Product. This product type allows to group single Simple or Virtual Products into bundles. Therefore, the customer can buy all the needed items at once, without adding them to the cart separately. Moreover, the customer gets to decide which products from the bundle he wants to purchase and in what amount. The products he or she selected are added to the cart separately.



Set of Sprite Yoga Straps

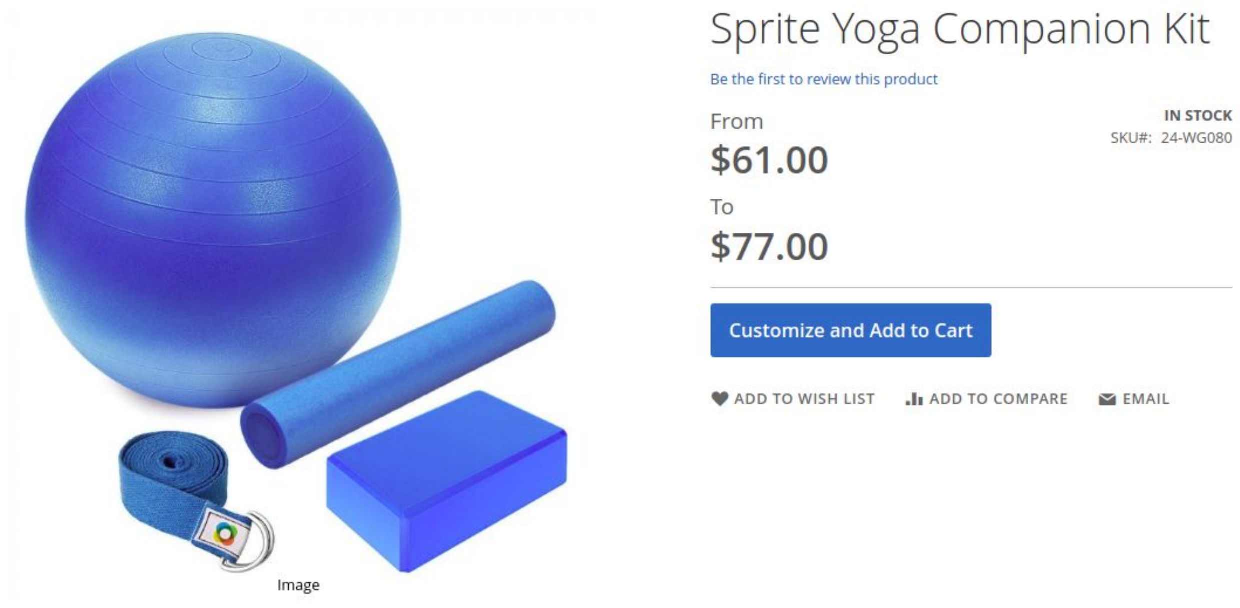
Be the first to review this product

IN STOCK
SKU#: 24-WG085_Group

Product Name	Qty
Sprite Yoga Strap 6 foot **$14.00**	0
Sprite Yoga Strap 8 foot **$17.00**	0
Sprite Yoga Strap 10 foot **$21.00**	0

Grouped Product

5. Bundle Product. This product type allows customers to create a bundle at their wish, using a set of options (for example, a yoga kit).



Sprite Yoga Companion Kit

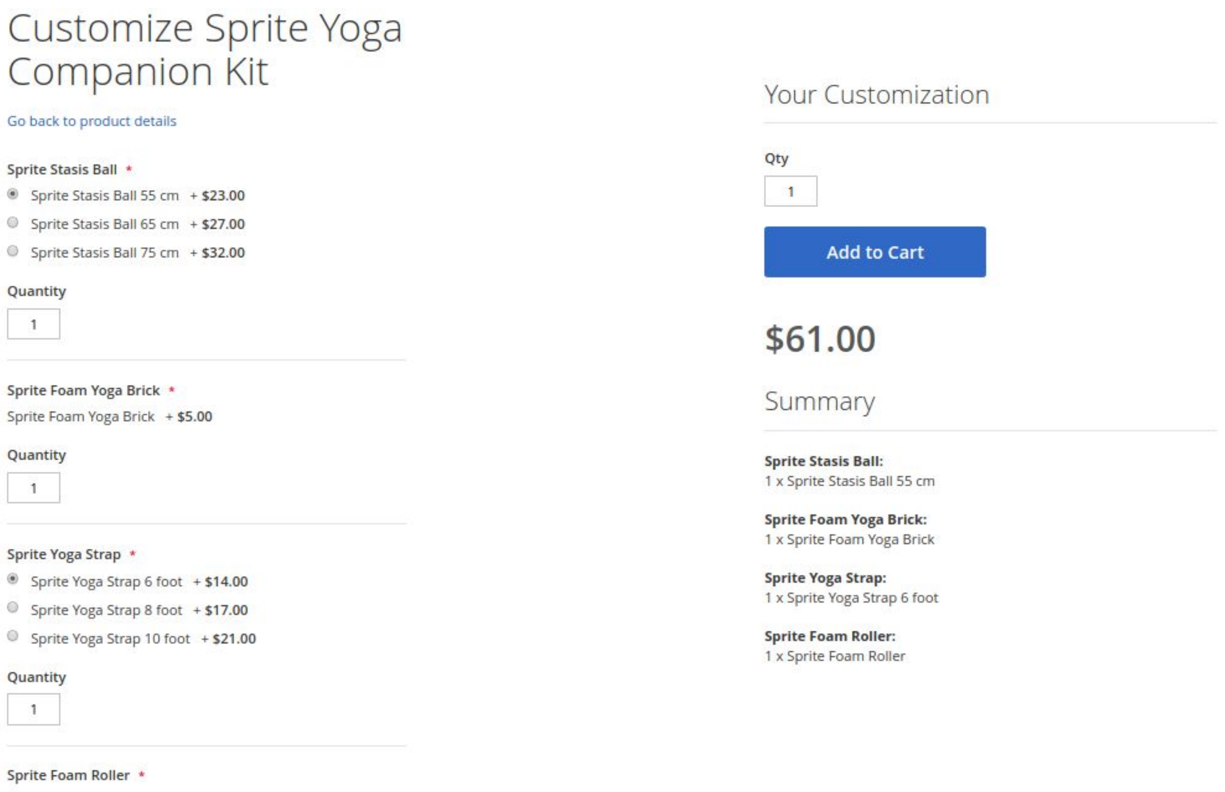
Be the first to review this product

From
$61.00
To
$77.00

IN STOCK
SKU#: 24-WG080

Bundle Product

Each option is a Simple Product or a Virtual Product. Before adding a bundle product to the cart, a customer must customize it with the provided options. The final price will depend on the chosen configuration.



Customize Sprite Yoga Companion Kit

Go back to product details

Sprite Stasis Ball *

- [✓ Sprite Stasis Ball 55 cm + $23.00]
- Sprite Stasis Ball 65 cm + $27.00
- Sprite Stasis Ball 75 cm + $32.00

Quantity: 1

Sprite Foam Yoga Brick *

Sprite Foam Yoga Brick + $5.00

Quantity: 1

Sprite Yoga Strap *

- [✓ Sprite Yoga Strap 6 foot + $14.00]
- Sprite Yoga Strap 8 foot + $17.00
- Sprite Yoga Strap 10 foot + $21.00

Quantity: 1

Sprite Foam Roller *

Your Customization

Qty: 1

$61.00

Summary

Sprite Stasis Ball:
1 x Sprite Stasis Ball 55 cm

Sprite Foam Yoga Brick:
1 x Sprite Foam Yoga Brick

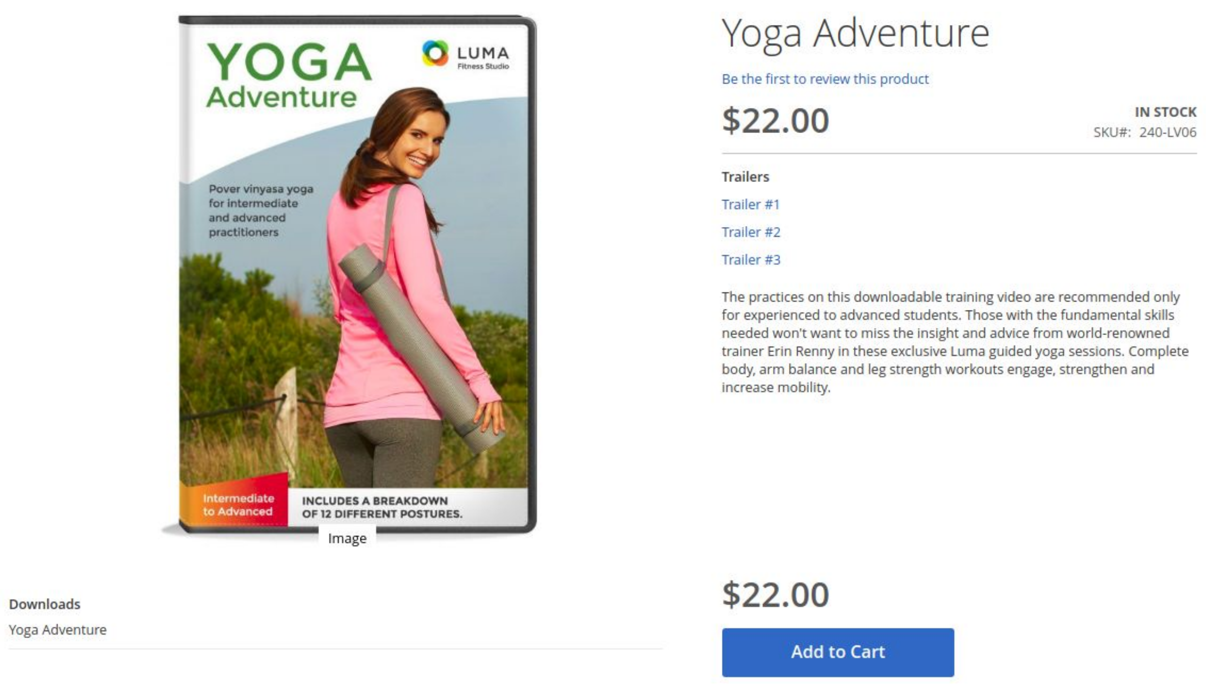
Sprite Yoga Strap:
1 x Sprite Yoga Strap 6 foot

Sprite Foam Roller:
1 x Sprite Foam Roller

Bundle Product parameters

SKU and Weight attributes can be fixed or dynamic. The price for Bundle Product can be set as Price Range (from minimum to maximum) or As Low As (the lowest price possible). The admin can set how the products will be delivered - together or separately.

6. Downloadable Product. This product type is aimed at digital products that can consist of one or several files, downloaded by the customer after the purchase is made.


Yoga Adventure

Be the first to review this product

$22.00

IN STOCK
SKU#: 240-LV06

Trailers

Trailer #1

Trailer #2

Trailer #3

The practices on this downloadable training video are recommended only for experienced to advanced students. Those with the fundamental skills needed won't want to miss the insight and advice from world-renowned trainer Erin Renny in these exclusive Luma guided yoga sessions. Complete body, arm balance and leg strength workouts engage, strengthen and increase mobility.

Downloads

Yoga Adventure

$22.00

YOGA Adventure
LUMA Fitness Studio
Pover vinyasa yoga for intermediate and advanced practitioners
Intermediate to Advanced
INCLUDES A BREAKDOWN OF 12 DIFFERENT POSTURES.
Image
Add to Cart

Downloadable Product

Virtual and Downloadable Products weight nothing, which means there is no need to deliver them and no need to select a delivery option at the checkout. Also, Virtual and Downloadable Products do not have In Stock attribute.

To get all products of a certain type, use the Magento\Catalog\Model\ResourceModel\Product\Collection: $collection->addFieldToFilter('type_id', 'bundle'); class.
As the second parameter, pass the product type name: bundle, configurable, downloadable, grouped, simple or virtual.

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

Basic methods, available for different product types:

- `getSetAttributes` - returns product attributes set;
- `getAttributeById` - returns attribute based on its ID and product ID;

- `isVirtual` - determines if the product is virtual;
- `isSalable` - determines if the product is Salable;
- `isComposite` - determines if the product is composite;
- `canConfigure` - determines if the product is configurable;
- `prepareForCart` - initializes the product for adding to cart;
- `checkProductBuyState` - checks if it is possible to buy the product;
- `getSku` - returns product SKU that has or does not have options;
- `hasOptions` - checks if the product has options;
- `hasWeight` - checks if the product has weight;
- `getRelationInfo` - returns product relations information;
- `getAssociatedProducts` - returns associated products;
- `getChildrenIds` - returns the list of child products' IDs;
- `getParentIdsByChild` - returns the list of parent products' IDs;
- `assignProductToOption` - assigns options to a product.

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

Configurable:
- `getConfigurableAttributes` - gets the attributes, used for subproducts;
- `getUsedProductIds` - gets id subproducts;
- `getProductByAttributes` - gets products by their attribute values;
- `getConfigurableOptions` - gets options list;
- `setImageFromChildProduct` - sets the image of a child product for a parent product, if it was not set previously.

Bundle:
- `getOptions` - gets the list of options;
- `getSelectionsCollection` - gets the selections collection by their id;
- `getSpecifyOptionMessage` - gets the customer message with the request to specify options;
- `checkIsAllRequiredOptions` - checks whether all the required options are selected;
- `checkSelectionsIsSale` - checks if all the options are available for sale.

Downloadable:

- `getLinks` - checks the links for downloading the product;
- `hasLinks` - checks if all the downloadable products have links;
- `getLinkSelectionRequired` - checks if the product can be bought without the selected links;
- `getSamples` - gets the downloadable product samples;
- `hasSamples` - checks if the product has samples for downloading.

It is worth mentioning that getRelationInfo, getAssociatedProducts, getChildrenIds, getParentIdsByChild, etc. methods return empty arrays, even though they are shared by all product types, not only compound ones, like Simple and Virtual. The logic of these methods is realized in the compound types' classes, taking into account this type's specifics.

7.2 Describe price functionality

Identify the basic concepts of price generation in Magento.

As a rule, each product in Magento 2 can have several prices: regular, special, final, etc. Each price type has a class, accountable for calculating the final value of a certain price type. Some price types are available for all the products, while others are specific to a certain product type.
Let us examine the basic price types, available for all product types:

- `base_price` - product price at the default exchange rate;
- `regular_price` - product price at the chosen exchange rate;
- `final_price` - final product price;
- `special_price` - product price with a discount;
- `tier_price` - product price depending on the number of products in the cart and customer group;
- `custom_option_price` - if the product has options, option price can be displayed in percentage (except for configurable products);
- `configured_price` - product price together with options;
- `catalog_rule_price` - product price after catalog rules are applied.

Basic classes, accountable for different price types realization, are located in Magento\Catalog\Pricing\Price names area and extend the Magento\Framework\Pricing\Price\AbstractPrice class. Different product types, as a rule, override the calculation logic for various price types. In this case, the corresponding classes are located in the names area of the corresponding product types. Also, several product types add custom price types:

- downloadable products:
    - `link_price` - price when the product is downloaded from the link provided;
- bundle products:
    - `bundle_option` - price of bundle product option.

Each price type has getValue() and getAmount() methods. getValue() method returns the price value, while getAmount() method returns the final price with all the taxes added.

How would you identify what is composing the final price of the product?

final_price depends on the product type.

For simple and virtual product types, final_price corresponds to the minimal regular_price, catalog_rule_price, special_price, tier_price values.

For configurable products, the price of each option is selected as a minimal value from base_price, tier_price, index_price, catalog_rule_price.
At the same time, when choosing the minimum price, the status of the configurable product option, its availability for a particular site and availability in stock are checked. After determining the final_price of all options, the final price of the configurable product is determined, which will be equal to the lowest cost of its options.

final_price of grouped product equals the minimal final_prices of all the products from the group.

For bundle products, the price is calculated in the same way as for simple products + bundle_option - the cost of all required options multiplied by their number.

How can you customize the price calculation process?

There are several ways to customize the price calculation process:

1. Create a new price type and add it to the basic price set. For this, create a custom module with the class that will realize Magento\Framework\Pricing\Price\BasePriceProviderInterface interface Magento\Framework\Pricing\Price\BasePriceProviderInterface and expand Magento\Framework\Pricing\Price\AbstractPrice class; also, add the necessary instructions in the module's di-file:

```
<virtualType name="MyVendor\MyModule\Pricing\Price\Pool"
type="Magento\Framework\Pricing\Price\Pool">
   <arguments>
       <argument name="prices" xsi:type="array">
           <item name="my_price"
xsi:type="string">MyVendor\MyModule\Pricing\Price\MyPrice</item
>
       </argument>
       <argument name="target"
xsi:type="object">Magento\Catalog\Pricing\Price\Pool</argument>
   </arguments>
</virtualType>
```

2. Create a plugin for the class methods of that price type, the calculation process of which you need to modify; for example, around getValue() method.
3. Override the class, corresponding to the required price type, for a certain product type by using di-file in your module:

```
<virtualType name="MyVendor\MyModule\Pricing\Price\Pool"
type="Magento\Framework\Pricing\Price\Pool">
```

```
    <arguments>
        <argument name="prices" xsi:type="array">
            <item name="regular_price"
xsi:type="string">MyVendor\MyModule\Pricing\Price\MyRegularPric
e</item>
        </argument>
    </arguments>
</virtualType>
```

4. Completely override the price class using the preference instruction in di-file of the module.

Describe how price is rendered in Magento.

Let us examine how the price is rendered, using the process of final_price formatting and display. Magento\Catalog\Pricing\Render\FinalPriceBox class is accountable for the price display. Magento_Catalog::product/price/final_price.phtml is the template used for displaying the block content. In case the product has a special price, by calling the renderAmount() block method, you can display both old and new price. Otherwise, only the final, lowest price is displayed. Then, if the product has options (for example, size), the block with the lowest price option is displayed.

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

To render price in a given place, add you layout block of the Magento\Catalog\Pricing\Render class by passing price_type_code parameter as an argument.

```
<block class="Magento\Catalog\Pricing\Render"
name="product.price.myprice">
    <arguments>
```

```
        <argument name="price_render"
xsi:type="string">product.price.render.default</argument>
        <argument name="price_type_code"
xsi:type="string">final_price</argument>
        <argument name="zone" xsi:type="string">item_view</argument>
    </arguments>
</block>
```

Using data-arguments, you can modify the price box, amount renders and adjustment renders, by modifying their css-classes, id_suffix, id_prefix, etc.

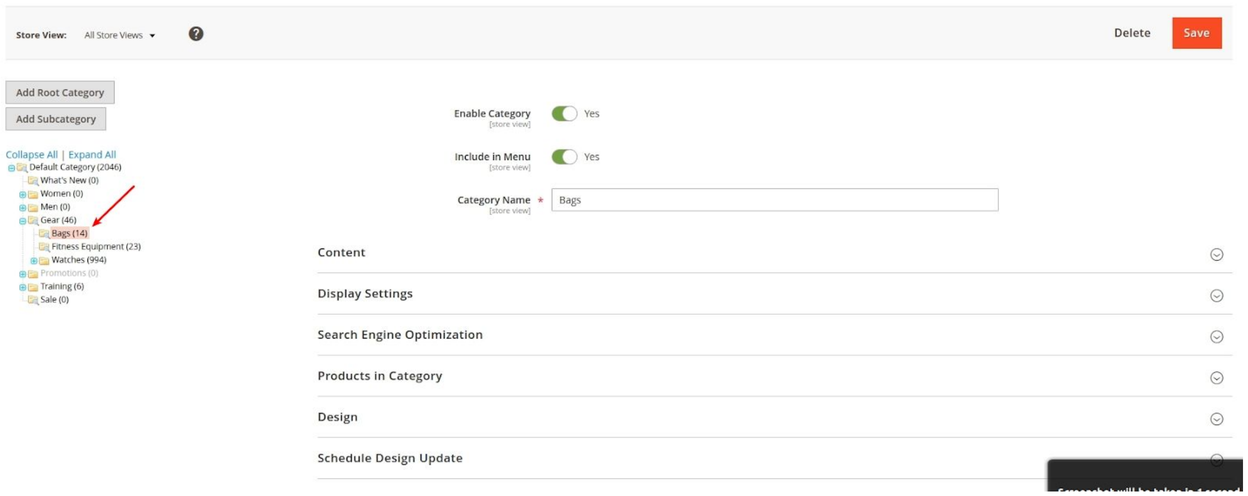
To modify price display template, create a custom catalog_product_prices.xml. There, you can modify render.product.prices block by passing the name of class, block and your template as parameters for the required price type.

```
<layout xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:View/Layout/etc/
layout_generic.xsd">
    <referenceBlock name="render.product.prices">
        <arguments>
            <argument name="myprice" xsi:type="array">
                <item name="prices" xsi:type="array">
                    <item name="final_price" xsi:type="array">
                        <item name="render_class"
xsi:type="string">MyVendor\MyModule\Pricing\Render\FinalPriceBox</ite
m>
                        <item name="render_template"
xsi:type="string">MyVendor_MyModule::product/price/final_price.phtml<
/item>
                    </item>
                </item>
            </argument>
        </arguments>
    </referenceBlock>
</layout>
```

7.3 Demonstrate ability to use and customize categories

Describe category properties and features.

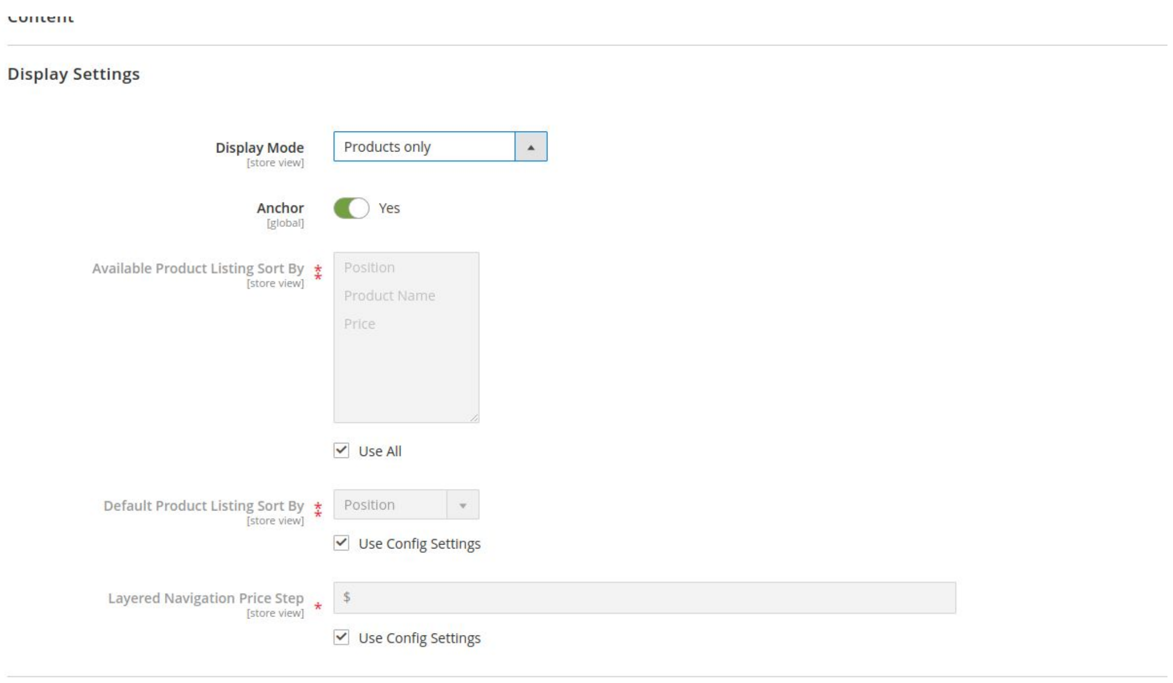
Magento 2 categories were created to conveniently group products according to their types and bundles, as well as to enhance products display. The categories are also used for building top menu. Categories have their own settings, and to modify them, navigate to Catalog -> Categories page and select a category for editing.



Enable Category [store view]: Yes
Include in Menu [store view]: Yes
Category Name * [store view]: Bags

Below are the major settings:
Enable Category - enables and disables category display at the frontend.
Include in Menu - determines whether the category is included in the top menu.

The next tab is Display Settings


Content

Display Settings

Display Mode [store view]	Products only
Anchor [global]	Yes
Available Product Listing Sort By * [store view]	Position Product Name Price ☑ Use All
Default Product Listing Sort By * [store view]	Position ☑ Use Config Settings
Layered Navigation Price Step * [store view]	$ ☑ Use Config Settings

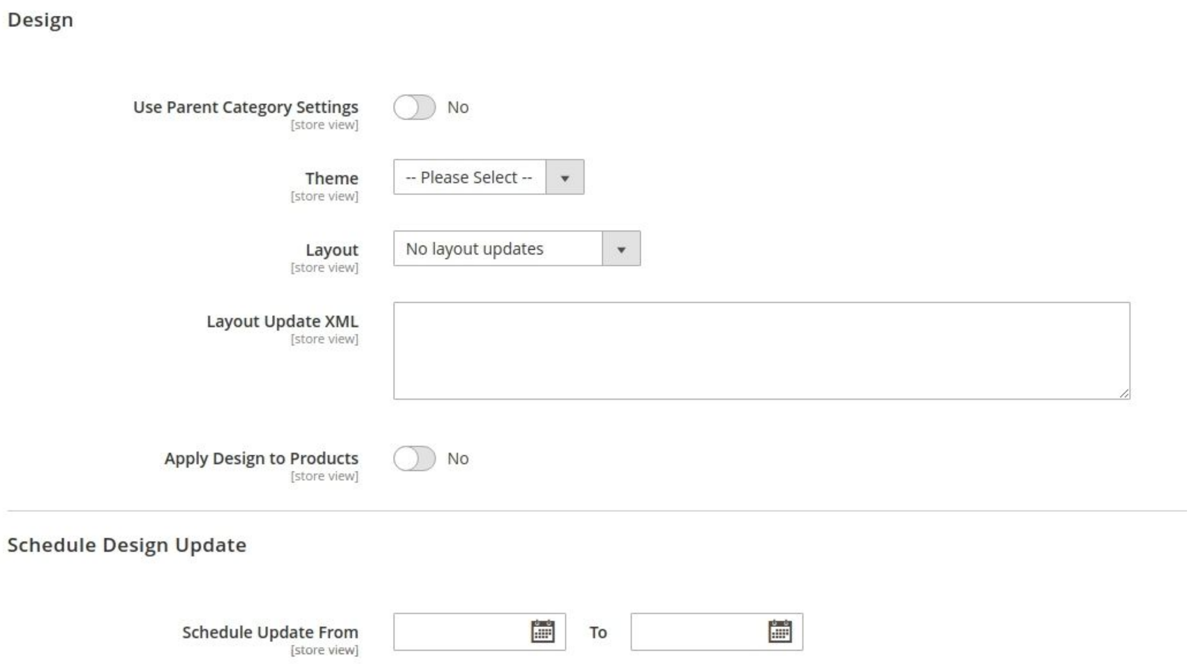
Display mode - allows to switch the category display mode. There are three modes available:

1. Products only - only category / product description will be displayed.
2. Static block only - only description and static block, selected in Tab Content above, will be displayed; the products will not.
3. Static block and Products - both static and products blocks will be displayed.

Anchor - sets the display of the layered navigation in the given category.

The settings from Search Engine Optimization and Products in Category tabs are rather intuitive. The first allows to add meta descriptions to the categories, while the second allows to add the products, displayed in the given category.

Tabs Design and Schedule Design Update


Design

Use Parent Category Settings [store view]	No
Theme [store view]	-- Please Select --
Layout [store view]	No layout updates
Layout Update XML [store view]	
Apply Design to Products [store view]	No

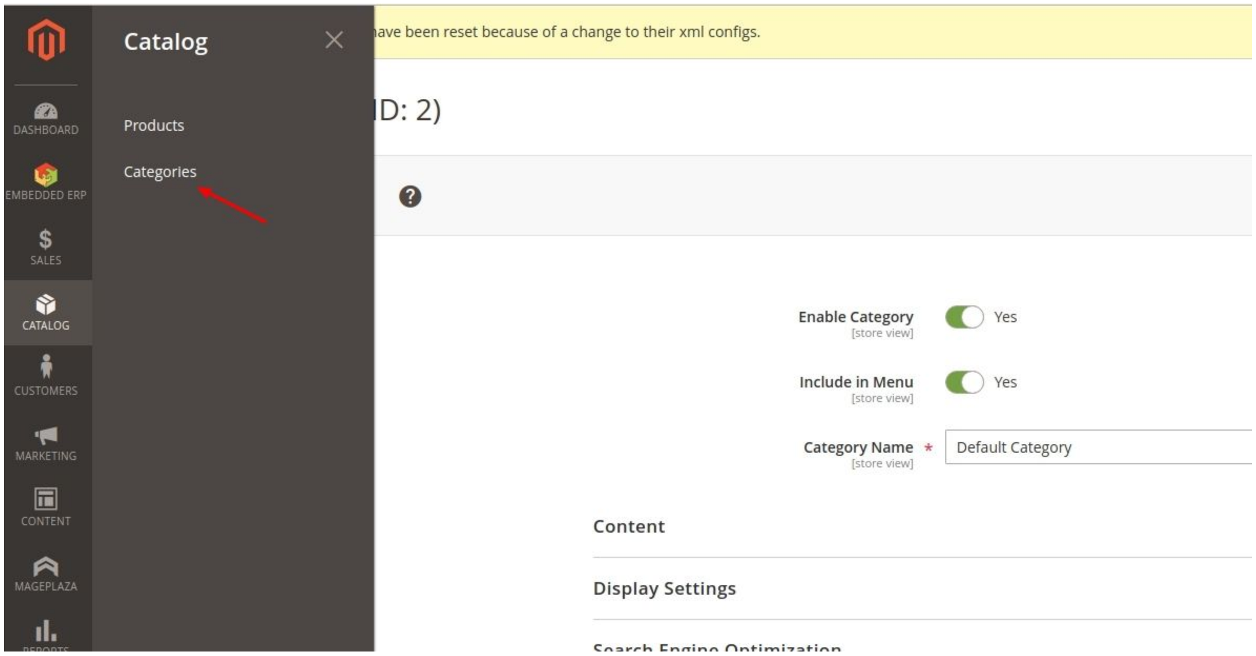
Schedule Design Update

Schedule Update From [store view]	To

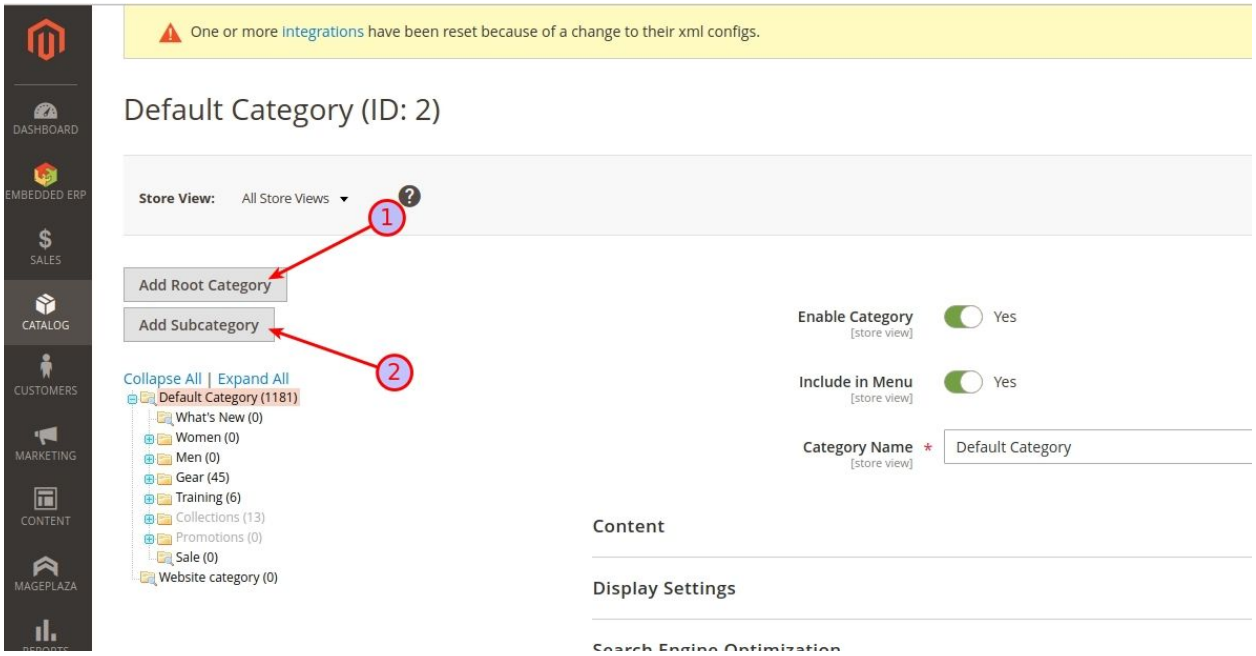
With Use Parent Category Settings, you can set whether the parent category theme will be applied; you can also choose which theme or layout will be applied for the given category (Theme and Layout settings).
Layout Update Xml is aimed at adding xml instructions via the admin interface.
Schedule Update From - sets the period, during which the altered Design settings will apply to the category.

How do you create and manage categories?

To create a category, navigate to Catalog -> Categories


Catalog
Products
Categories
DASHBOARD
EMBEDDED ERP
SALES
CATALOG
CUSTOMERS
MARKETING
CONTENT
MAGEPLAZA
have been reset because of a change to their xml configs.
ID: 2)
Enable Category [store view] Yes
Include in Menu [store view] Yes
Category Name [store view] Default Category
Content
Display Settings

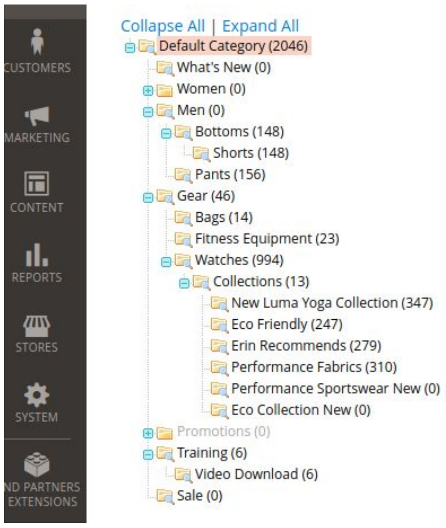
Select the category type - either Root Category or Subcategory.


One or more integrations have been reset because of a change to their xml configs.
Default Category (ID: 2)
Store View: All Store Views
1
2
Add Root Category
Add Subcategory
Collapse All | Expand All
Default Category (1181)
What's New (0)
Women (0)
Men (0)
Gear (45)
Training (6)
Collections (13)
Promotions (0)
Sale (0)
Website category (0)
Enable Category [store view] Yes
Include in Menu [store view] Yes
Category Name [store view] Default Category
Content
Display Settings

Add Root Category button creates a new category in the categories tree root. All the root categories are further used for the multistore settings.
Add Subcategory button creates a subcategory inside the active category.

Describe the category hierarchy tree structure implementation (the internal structure inside the database). What is the meaning of parent_id 0? How are paths constructed?

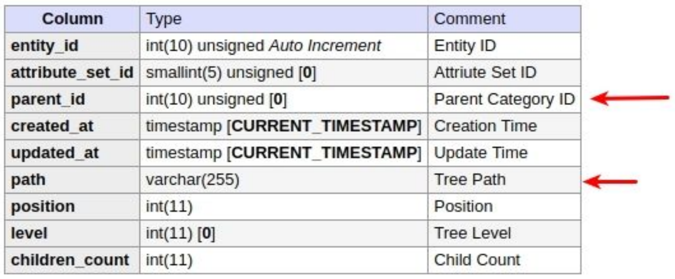
Magento 2 categories are presented in the form of a tree and can have any nesting.



Collapse All | Expand All

- Default Category (2046)
  - What's New (0)
  - Women (0)
  - Men (0)
    - Bottoms (148)
      - Shorts (148)
    - Pants (156)
  - Gear (46)
    - Bags (14)
    - Fitness Equipment (23)
    - Watches (994)
      - Collections (13)
        - New Luma Yoga Collection (347)
        - Eco Friendly (247)
        - Erin Recommends (279)
        - Performance Fabrics (310)
        - Performance Sportswear New (0)
        - Eco Collection New (0)
  - Promotions (0)
  - Training (6)
    - Video Download (6)
  - Sale (0)

To realize category tree in Magento, the following logic is used. The picture below demonstrates the catalog_category_entity table structure, where the information about categories structure is stored.

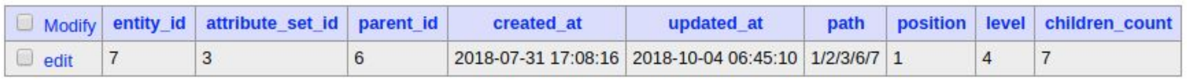


Column	Type	Comment
entity_id	int(10) unsigned *Auto Increment*	Entity ID
attribute_set_id	smallint(5) unsigned **[0]**	Attriute Set ID
parent_id	int(10) unsigned **[0]**	Parent Category ID
created_at	timestamp **[CURRENT_TIMESTAMP]**	Creation Time
updated_at	timestamp **[CURRENT_TIMESTAMP]**	Update Time
path	varchar(255)	Tree Path
position	int(11)	Position
level	int(11) **[0]**	Tree Level
children_count	int(11)	Child Count

The main columns are accountable for category structure:

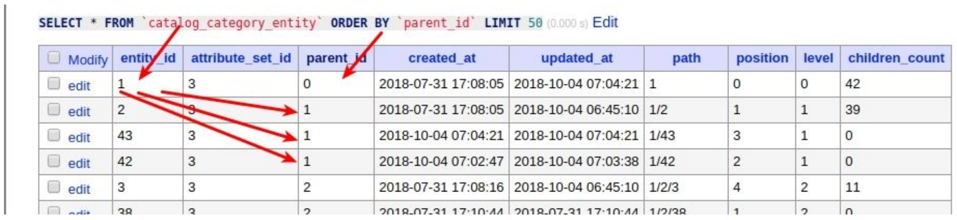
`entity_id` - the category ID.
`parent_id` - parent category ID.

path - path to the category in the categories tree. This is a record with a path to a category in the category tree. Represents a record in the form of a sequential arrangement of entity_id categories, separated by a slash (/).



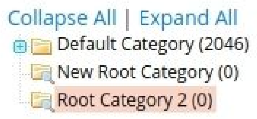
	Modify	entity_id	attribute_set_id	parent_id	created_at	updated_at	path	position	level	children_count
☐	edit	7	3	6	2018-07-31 17:08:16	2018-10-04 06:45:10	1/2/3/6/7	1	4	7

The construction of the category tree begins with an entry with the parent_id = 0 value. This entry has a value of entity_id = 1. Categories whose parent_id = 1 act as Root Categories. And then the construction of the category tree is implemented through the connection parent_id <---> entity_id.



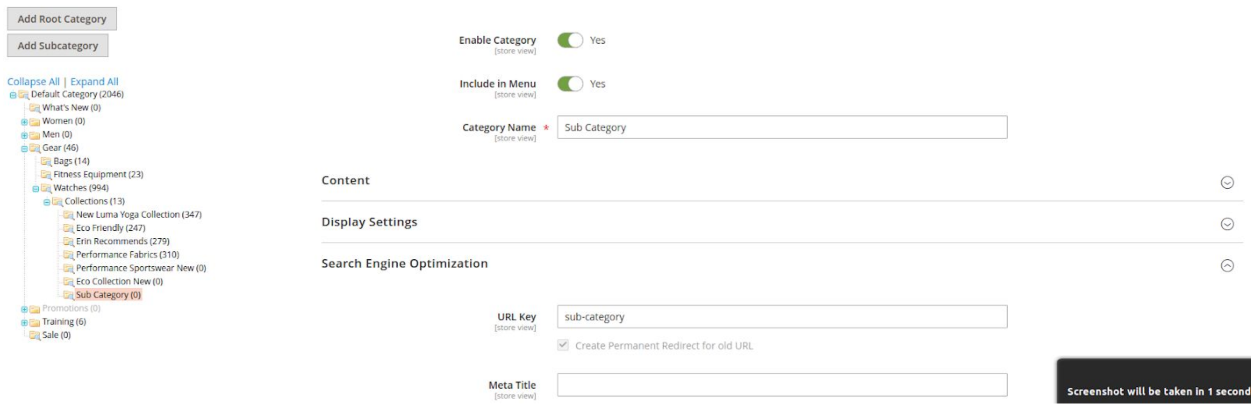
```
SELECT * FROM `catalog_category_entity` ORDER BY `parent_id` LIMIT 50 (0.000 s) Edit
```

	Modify	entity_id	attribute_set_id	parent_id	created_at	updated_at	path	position	level	children_count
☐	edit	1	3	0	2018-07-31 17:08:05	2018-10-04 07:04:21	1	0	0	42
☐	edit	2	3	1	2018-07-31 17:08:05	2018-10-04 06:45:10	1/2	1	1	39
☐	edit	43	3	1	2018-10-04 07:04:21	2018-10-04 07:04:21	1/43	3	1	0
☐	edit	42	3	1	2018-10-04 07:02:47	2018-10-04 07:03:38	1/42	2	1	0
☐	edit	3	3	2	2018-07-31 17:08:16	2018-10-04 06:45:10	1/2/3	4	2	11
☐	edit	38	3	2	2018-07-31 17:10:44	2018-07-31 17:10:44	1/2/38	1	2	0



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

When creating a category, the only required value is Category name. Based on this name, the URL Key is automatically generated, which is the path by which the category will be displayed in the browser.



Enable Category [store view] Yes
Include in Menu [store view] Yes
Category Name * [store view] Sub Category
URL Key [store view] sub-category
☑ Create Permanent Redirect for old URL
Meta Title [store view]

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

In most cases, the strategy for organizing products into categories depends on the particular business that will be presented at the webshop.
One of the organizing products alternatives is to categorize products according to the set of attributes or attribute sets used. Therefore, the products that have the same set of additional attributes will be categorized. This will allow you to get a more intuitive product filter (layered navigation). For example, in a filter for bicycles there will be no size options, etc. Further, the separation of products can be made based on the attributes' values and customers target group (children, adults, men, women).
But in most cases, the marketing department of the company decides on the products structure and organization.

7.4 Determine and manage catalog rules

Identify how to implement catalog price rules. When would you use catalog price rules?

Catalog price rules would be used to set a discount, depending on conditions, for a product or product group. To create them in the admin panel, navigate to Marketing - Catalog Price Rule. Here, the administrator can specify the conditions, under which catalog price rules can be applied to products, actions to be performed, and other parameters.

How do they impact performance?

The impact on page loading is relatively small, since all price calculations occur during reindex. The speed of reindex is affected by: the number of catalog rules, customer groups, products and websites affected. In the worst case, if all catalog price rules affect all customer groups, products and websites, then the number of rows in the catalogrule_product table will be CATALOG_RULES_QTY * CUSTOMER_GROUPS_QTY * PRODUCTS_QTY * WEBSITES_QTY. The catalogrule_product_price table contains 3 times more rows than the catalogrule_product table.

Let us examine one configuration example:
We have one catalog rule that applies to 4 customer groups, one website and 247 products. In this case, the number of lines in the tables will be the following:
catalogrule_product: 1 * 4 * 1 * 247 = 988
Catalogrule_product_price: 988 * 3 = 2964.

Another example:
We have 5 catalog rules, each applies to 4 customer groups, 5 websites and 2000 products. Then, the number of lines will be the following:
catalogrule_product: 5 * 4 * 5 * 2000 = 200000
catalogrule_product_price: 200000 * 3 = 600000.

Now, we will examine the reason why the influence on the page upload is relatively insignificant. Let us turn to Magento code.
In vendor\magento\module-catalog-rule\etc\frontend\events.xml file, set the observer for catalog_product_get_final_price event.
https://i.imgur.com/XuicwTk.png

```
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:Event/etc/events
.xsd">
   <event name="catalog_product_get_final_price">
       <observer name="catalogrule"
instance="Magento\CatalogRule\Observer\ProcessFrontFinalPriceObserver
```

```
" />
   </event>
   <event name="prepare_catalog_product_collection_prices">
       <observer name="catalogrule"
instance="Magento\CatalogRule\Observer\PrepareCatalogProductCollectio
nPricesObserver" />
   </event>
</config>
```

Then, in magento\module-catalog-rule\Observer\ProcessFrontFinalPriceObserver.php file, we get the price and assign it to the product.
https://i.imgur.com/Tx3qb7L.png

```
public function execute(\Magento\Framework\Event\Observer $observer)
{
   $product = $observer->getEvent()->getProduct();
   $pId = $product->getId();
   $storeId = $product->getStoreId();

   if ($observer->hasDate()) {
       $date = new \DateTime($observer->getEvent()->getDate());
   } else {
       $date = $this->localeDate->scopeDate($storeId);
   }

   if ($observer->hasWebsiteId()) {
       $wId = $observer->getEvent()->getWebsiteId();
   } else {
       $wId =
$this->storeManager->getStore($storeId)->getWebsiteId();
   }

   if ($observer->hasCustomerGroupId()) {
       $gId = $observer->getEvent()->getCustomerGroupId();
   } elseif ($product->hasCustomerGroupId()) {
       $gId = $product->getCustomerGroupId();
   } else {
       $gId = $this->customerSession->getCustomerGroupId();
   }

   $key = "{$date->format('Y-m-d H:i:s')}|{$wId}|{$gId}|{$pId}";
   if (!$this->rulePricesStorage->hasRulePrice($key)) {
       $rulePrice =
$this->resourceRuleFactory->create()->getRulePrice($date, $wId, $gId,
$pId);
       $this->rulePricesStorage->setRulePrice($key, $rulePrice);
   }
   if ($this->rulePricesStorage->getRulePrice($key) !== false) {
       $finalPrice = min($product->getData('final_price'),
$this->rulePricesStorage->getRulePrice($key));
       $product->setFinalPrice($finalPrice);
   }
   return $this;
}
```

Then, let's examine what is happening in getRulePrice() method. Navigate to vendor\magento\module-catalog-rule\Model\ResourceModel\Rule.php file and find the method. It will call getRulePrices() method, where the simplest request into the database to catalogrule_product_price table is performed. This is how we get the price.

https://i.imgur.com/3vfkW4u.png

```
public function getRulePrice($date, $wId, $gId, $pId)
{
    $data = $this->getRulePrices($date, $wId, $gId, [$pId]);
    if (isset($data[$pId])) {
        return $data[$pId];
    }

    return false;
}

public function getRulePrices(\DateTimeInterface $date, $websiteId,
$customerGroupId, $productIds)
```

```
{
   $connection = $this->getConnection();
   $select = $connection->select()
       ->from($this->getTable('catalogrule_product_price'),
['product_id', 'rule_price'])
       ->where('rule_date = ?', $date->format('Y-m-d'))
       ->where('website_id = ?', $websiteId)
       ->where('customer_group_id = ?', $customerGroupId)
       ->where('product_id IN(?)', $productIds);

   return $connection->fetchPairs($select);
}
```

How would you debug problems with catalog price rules?

First, make sure that price rule is active. Navigate to Marketing > promotions > Catalog Price Rule and make sure that the status is set at active.
The, apply the rules by pressing the corresponding button Apply Rules
 https://i.imgur.com/99ZRWIp.png
After that, Catalog Rule Product indexes and Product Price, connected to it, are set as invalid. https://i.imgur.com/W1N3aBQ.png

In action
vendor\magento\module-catalog-rule\Controller\Adminhtml\Promo\Catalog\ApplyRules.php, a copy of
\Magento\CatalogRule\Model\Rule\Job class is created, in which applyAll() method is called. https://i.imgur.com/v7FfJK5.png .

```
public function execute()
{
   $errorMessage = __('We can\'t apply the rules.');
   try {
       /** @var Job $ruleJob */
       $ruleJob =
$this->_objectManager->get(\Magento\CatalogRule\Model\Rule\Job::class
);
        $ruleJob->applyAll();

        if ($ruleJob->hasSuccess()) {
            $this->messageManager->addSuccess($ruleJob->getSuccess());

$this->_objectManager->create(\Magento\CatalogRule\Model\Flag::class)
->loadSelf()->setState(0)->save();
        } elseif ($ruleJob->hasError()) {
            $this->messageManager->addError($errorMessage . ' ' .
$ruleJob->getError());
        }
    } catch (\Exception $e) {

$this->_objectManager->create(\Psr\Log\LoggerInterface::class)->criti
cal($e);
        $this->messageManager->addError($errorMessage);
    }

    /** @var \Magento\Backend\Model\View\Result\Redirect
$resultRedirect */
    $resultRedirect =
$this->resultFactory->create(ResultFactory::TYPE_REDIRECT);
    return $resultRedirect->setPath('catalog_rule/*');
}
```

In this method, the index is set as invalid.
https://i.imgur.com/NX4XsEE.png

```
public function applyAll()
{
    try {
        $this->ruleProcessor->markIndexerAsInvalid();
        $this->setSuccess(__('Updated rules applied.'));
    } catch (\Magento\Framework\Exception\LocalizedException $e) {
        $this->setError($e->getMessage());
    }
    return $this;
}
```

Then, launch reindex and flush the cache.
bin/magento indexer:reindex
bin/magento cache:flush

After all these actions and if the price rule is configured correctly (if date range and conditions are both correct), the rules will work.

In case the rules do not work, navigate to catalogrule_product_price table and check the product price there. If the data is incorrect, try checking the logs, disconnecting third-party modules or review the Catalog Price Rule reindex process with xDebug.

Section 8: Customizing the Checkout Process

8.1 Demonstrate ability to use quote, quote item, address, and shopping cart rules in checkout

Quote contains data for creating an order. This data is temporary and can be modified by the user. When the order is created, the user can no longer change this data in it. Magento uses Quote for the following purposes:

- Store products in the cart along with their cost, quantity and options
- Store selected billing address and shipping address
- Store shipping costs
- Store subtotals of prices, additional prices (shipping costs, taxes, etc.) and coupons application to determine the total price
- Store selected payment method

The following tables are aimed at storing Quote in the database:

- quote
- quote_address
- quote_address_item
- quote_id_mask
- quote_item
- quote_item_option
- quote_payment
- quote_shipping_rate

Quote is the responsibility of the Magento\Quote\Model\Quote model. The Magento\Quote\Model\Quote\Address model is responsible for Quote Address. Quote usually has 2 addresses (billing, shipping), but may contain more if there are several delivery addresses or none at all. If quote does not have an address, then Totals will not take into account Price Rules associated with the country. If the quote contains virtual products only, then the delivery address is not taken into account and checked using the isVirtual () method of the Magento/Quote \ Model \ Quote class for quote with virtual products, only the billing address is taken into account. For products in quote, the Magento\Quote\Model\Quote\Item model is responsible. For payment in quote, the Magento\Quote\Model\Quote\Payment model is responsible.

Rules in checkout

With Cart Price Rules, we can modify the final price at the checkout and the shipping costs. It is also possible to configure rules via the admin panel. There is a flexible set of conditions, under which the rules apply. We can add a coupon for applying the discount and indicate its percentage. Also, we can apply a discount or include free shipping to the cart that meets certain conditions. We can see all the settings in the Marketing-> Cart Price Rules-> Add New Rule section. As conditions, we can apply:

- Product attribute combination
- Products subselection
- Conditions combination
- Subtotal
- Total Items Quantity
- Total Weight
- Payment Method
- Shipping Method
- Shipping Postcode
- Shipping Region
- Shipping State/Province
- Shipping Country

The “Magento \ SalesRule \ Model \ Rule” model is responsible for the cart rules, and with it we can programmatically add or get a specific cart rule.

The more there are cart rules without coupons for the current website and for the current customer group, the slower is the processing of cart rules. It is not recommended to have a lot of such cart rules.

Interesting quote fields or methods:

If the “trigger_recollect” flag is set, the “quote” will also be updated when the price changes or the product is set to “disabled” status. The same happens if we change the rules of the CatalogRules catalog, for we can find these methods in the class Magento \ Quote \ Model \ ResourceModel markQuotesRecollectOnCatalogRules () and markQuotesRecollect ().

Quote is also extendable through Magento \ Framework \ Model \ AbstractExtensibleModel, for it supports "extension_attributes". You can register the extension through the extension_attributes.xml file:

```
<extension_attributes for="Magento\Quote\Api\Data\CartInterface">
<attribute code="shipping_assignments"
type="Magento\Quote\Api\Data\ShippingAssignmentInterface[]" />
</extension_attributes>
```

Use “Quote Repository Plugin” to fill in the values with the afterLoad (), beforeSave (), or whenever () functions. Quote does not use “custom_attributes” since they are not EAVs.

Quote address custom attributes:

- “CE” \Magento\Quote\Model\Quote\Address\CustomAttributeList getAttributes() return empty array. To implement it, we need to write a plugin.
- “EE” \Magento\CustomerCustomAttributes\Model\Quote\Address\CustomAttributeList::getAttributes return “customer address attributes” + “customer attributes”

Quote item useful methods:

- Magento\Quote\Model\Quote\Item checkData() is called after adding to cart and updating options
  - Magento\Quote\Model\Quote\Item setQty() - triggers stock validation
  - Magento\Catalog\Model\Product\Type\AbstractType checkProductBuyState() - check if the product can be bought
- Magento\Quote\Model\Quote\Item setCustomPrice()
- Magento\Quote\Model\Quote\Item getCalculationPrice() gets original custom price applied before tax calculation
- Magento\Quote\Model\Quote\Item isChildrenCalculated() checks if there are children calculated or parent item when we have parent quote item and its children

- Magento\Quote\Model\Quote\Item isShipSeparately() - Checking if we can ship products separately (each child separately) or each parent product item can be shipped only like one item
- Magento\Quote\Model\Quote\Item\Compare::compare merges items and adds quantity instead of a new item
- Magento\Quote\Model\Quote\Item representProduct() - compares quote item with some new product, checks product id and custom options
- Magento\Quote\Model\Quote\Item compareOptions() - check if two options array are identical. First options array is prerogative, and second options array checked compared to the first one.

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

Inventory validation

- Use Magento\Quote\Model\Quote\Item setQty() function to declare quote item quantity
- Use event "sales_quote_item_qty_set_after"
- Use function \Magento\CatalogInventory\Model\Quote\Item\QuantityValidator::validate to check product inventory data when quote item quantity declaring.

Add to cart

- Use function Magento\Quote\Model\Quote addProduct() to add a product to the shopping cart Returns error message if product type instance can't prepare product.
- Use function Magento\Catalog\Model\Product\Type\AbstractType prepareForCartAdvanced() to initialize product(s) to add to cart process. The advanced version of function that prepares product for cart (processMode) can be specified there.
- Use function Magento\Quote\Model\Quote\Item\Processor::prepare to set quantity and custom price for quote item
- Use event "sales_quote_product_add_after"
- Use event "sales_quote_save_after", "sales_quote_save_before"
- Use event "checkout_cart_add_product_complete"

Cart update

- Use function \Magento\Quote\Model\Quote updateItemUpdate() quote item information
- Use event "sales_quote_save_after", "sales_quote_save_before"

Describe how to customize the process of adding a product to the cart.

- Applying a plugin to the Magento\Catalog\Model\Product\Type\AbstractType prepareForCartAdvanced() method. This function is used to prepare the product for adding to the cart (that is, the "Quote" object) and is called from the Magento \ Quote \ Model \ Quote addProduct () method.
- Applying a plugin to the Magento\Quote\Model\Quote::addProduct method
- Applying a plugin to the Magento\Quote\Model\Quote::addItem method
- With the help of "catalog_product_type_prepare_full_options" event at the condition of full validation, or "catalog_product_type_prepare_lite_options" at the condition of partial validation, the event is launched right before the product with configurable options is configured into "Quote" element.
- Applying a plugin to the \Magento\Quote\Model\Quote\Item\Processor::prepare method, we can modify the number of products and their custom prices.
- With the help of "sales_quote_product_add_after" event, we can modify attribute values or product price.
- Using the "sales_quote_add_item" event.
- In thecatalog_attributes.xml file in <group name="quote_item"> group, we describe the product attributes that get into "quote"; for example, <attribute name="sku"/>.

Which different scenarios should you take into account?

- Adding into the shopping cart from the catalog
- Adding into the shopping cart from the wishlist
- Adding all items from the wishlist into the shopping cart
- Create an order from the admin / user part of the website
- "Reorder" from the admin / user part of the website

- Configuration of the added product is modification of custom options
- “Quotes” merge at the client authorization in case he or she added products as a guest and already have a “Quote”

8.2 Demonstrate ability to use totals models

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

Custom totals can be used to add an additional tax or discount in Magento Checkout or modify the existing ones.

First, we need to create the <module_dir>/etc/sales.xml file in our module. This file is applied for registration of all the available Magento totals.

```
<module_dir>/etc/sales.xml:
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:module:Magento_Sales:etc/s
ales.xsd">
    <section name="quote">
        <group name="totals">
            <item name="custom_total"
instance="Vendor\Module\Model\Totals\Custom" sort_order="500"/>
        </group>
    </section>
</config>
```

Here, we set \Vendor\Module\Model\Totals\Custom class as our custom total model. This class should inherit \Magento\Quote\Model\Quote\Address\Total\AbstractTotal and realize collect and fetch methods.
Here, we set \Vendor\Module\Model\Totals\Custom class as the model of our custom total. This class should inherit \Magento\Quote\Model\Quote\Address\Total\AbstractTotal and realize methods collect and fetch. Collect method is applied to calculate our total, while fetch method returns the value together with total's code and its name.
Also, \Magento\Quote\Model\Quote\Address\Total $total parameter allows you to affect the result of the other total classes. But, depending on the task, it may be reasonable to use plugins to modify their values.

<module_dir>/Model/Total/Custom.php

```
<?php
Namespace Vendor\Module\Model\Total;

class Custom extends
\Magento\Quote\Model\Quote\Address\Total\AbstractTotal
{
    /**
     * Custom constructor.
     */
    public function __construct()
    {
        $this->setCode('custom_total');
    }

    /**
     * @param \Magento\Quote\Model\Quote $quote
     * @param \Magento\Quote\Api\Data\ShippingAssignmentInterface
$shippingAssignment
     * @param \Magento\Quote\Model\Quote\Address\Total $total
     * @return $this
     */
    public function collect(
        \Magento\Quote\Model\Quote $quote,
        \Magento\Quote\Api\Data\ShippingAssignmentInterface
$shippingAssignment,
        \Magento\Quote\Model\Quote\Address\Total $total
    ) {
        parent::collect($quote, $shippingAssignment, $total);

        $items = $shippingAssignment->getItems();
        if (!count($items)) {
            return $this;
        }

        //we will add an additional amount of 150 to the order as an
example
        $amount = 150;

        $total->setTotalAmount('custom_total', $amount);
        $total->setBaseTotalAmount('custom_total', $amount);
        $total->setCustomAmount($amount);
        $total->setBaseCustomAmount($amount);
        $total->setGrandTotal($total->getGrandTotal() + $amount);
        $total->setBaseGrandTotal($total->getBaseGrandTotal() +
$amount);

        return $this;
    }

    /**
     * @param \Magento\Quote\Model\Quote\Address\Total $total
     */
    protected function
clearValues(\Magento\Quote\Model\Quote\Address\Total  $total)
    {
        $total->setTotalAmount('subtotal', 0);
        $total->setBaseTotalAmount('subtotal', 0);
        $total->setTotalAmount('tax', 0);
        $total->setBaseTotalAmount('tax', 0);
        $total->setTotalAmount('discount_tax_compensation', 0);
        $total->setBaseTotalAmount('discount_tax_compensation', 0);
        $total->setTotalAmount('shipping_discount_tax_compensation',
0);

$total->setBaseTotalAmount('shipping_discount_tax_compensation', 0);
        $total->setSubtotalInclTax(0);
        $total->setBaseSubtotalInclTax(0);
    }

    /**
     * @param \Magento\Quote\Model\Quote $quote
     * @param \Magento\Quote\Model\Quote\Address\Total $total
     * @return array
     */
    public function fetch(
        \Magento\Quote\Model\Quote  $quote,
         \Magento\Quote\Model\Quote\Address\Total $total
    ) {
        return [
            'code' => $this->getCode(),
            'title' => 'Custom Total',
            'value' => 150
        ];
    }

    /**
     * @return \Magento\Framework\Phrase
     */
    public function getLabel()
    {
        return __('Custom Total');
    }
}
```

Displaying Custom Total

Cart and checkout pages

We use knockout.js to display our totals. Therefore, to make our total appear at the cart and checkout pages, we need to add a new JS component into checkout_cart_index.xml and checkout_index_index.xml layouts.

<module_dir>/view/frontend/layout/checkout_cart_index.xml

```
<?xml version="1.0"?>
<page xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:View/Layout/etc/page_configura
tion.xsd">
    <body>
        <referenceBlock name="checkout.cart.totals">
            <arguments>
                <argument name="jsLayout" xsi:type="array">
                    <item name="components" xsi:type="array">
                        <item name="block-totals" xsi:type="array">
                            <item name="children" xsi:type="array">
                                <item name="custom_total" xsi:type="array">
                                    <item name="component"
xsi:type="string">Vendor_Module/js/view/checkout/cart/totals/custom_total</item>
                                    <item name="sortOrder"
xsi:type="string">20</item>
                                    <item name="config" xsi:type="array">
                                        <item name="template"
xsi:type="string">Vendor_Module/checkout/cart/totals/custom_total</item>
                                        <item name="title" xsi:type="string">Custom
Total</item>
                                    </item>
                                </item>
                            </item>
                        </item>
                    </item>
                </argument>
            </arguments>
        </referenceBlock>
    </body>
</page>
```

<module_dir>/view/frontend/layout/checkout_index_index.xml

```
<?xml version="1.0"?>
<page xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" layout="1column"
xsi:noNamespaceSchemaLocation="urn:magento:framework:View/Layout/etc/page_configura
tion.xsd">
    <body>
        <referenceBlock name="checkout.root">
            <arguments>
                <argument name="jsLayout" xsi:type="array">
                    <item name="components" xsi:type="array">
                        <item name="checkout" xsi:type="array">
                            <item name="children" xsi:type="array">
                                <item name="sidebar" xsi:type="array">
                                    <item name="children" xsi:type="array">
                                        <item name="summary" xsi:type="array">
                                            <item name="children" xsi:type="array">
                                                <item name="totals"
xsi:type="array">
                                                    <item name="children"
xsi:type="array">
                                                        <item name="custom_total"
xsi:type="array">
                                                            <item name="component"
xsi:type="string">Vendor_Module/js/view/checkout/cart/totals/custom_total</item>
                                                            <item name="sortOrder"
xsi:type="string">20</item>
                                                            <item name="config"
xsi:type="array">
                                                                <item
name="template"
xsi:type="string">Vendor_Module/checkout/cart/totals/custom_total</item>
                                                                <item name="title"
xsi:type="string">Custom Total</item>
                                                            </item>
                                                        </item>
                                                    </item>
                                                </item>
                                                <item name="cart_items"
xsi:type="array">
                                                    <item name="children"
xsi:type="array">
                                                        <item name="details"
xsi:type="array">
```

```
                                                            <item name="children"
xsi:type="array">
                                                                <item
name="subtotal" xsi:type="array">
                                                                    <item
name="component"
xsi:type="string">Magento_Tax/js/view/checkout/summary/item/details/subtotal</item>
                                                                </item>
                                                            </item>
                                                        </item>
                                                    </item>
                                                </item>
                                            </item>
                                        </item>
                                    </item>
                                </item>
                            </item>
                        </item>
                    </item>
                </argument>
            </arguments>
        </referenceBlock>
    </body>
</page>
```

We can also create the components themselves, together with HTML templates:

<module_dir>/view/frontend/web/js/view/checkout/cart/totals/custom_total.js

```
define(
    [
        'Vendor_Module/js/view/checkout/summary/custom_total'
    ],
    function (Component) {
        'use strict';

        return Component.extend({

            /**
             * @override
             */
            isDisplayed: function () {
                return this.getPureValue() !== 0;
            }
        });
    }
);
```

<module_dir>/view/frontend/web/template/checkout/cart/totals/custom_total.html

```
<!-- ko if: isDisplayed() -->
<tr class="totals custom_total excl">

    <th class="mark" colspan="1" scope="row" data-bind="text:
title"></th>
    <td class="amount">
        <span class="price" data-bind="text: getValue()"></span>
    </td>
</tr>
<!-- /ko →
```

<module_dir>/view/frontend/web/js/view/checkout/summary/custom_total.js

```
define(
    [
        'Magento_Checkout/js/view/summary/abstract-total',
        'Magento_Checkout/js/model/quote',
        'Magento_Catalog/js/price-utils',
        'Magento_Checkout/js/model/totals'
    ],
    function (Component, quote, priceUtils, totals) {
        "use strict";
        return Component.extend({
            defaults: {
                isFullTaxSummaryDisplayed:
window.checkoutConfig.isFullTaxSummaryDisplayed || false,
                template:
'Vendor_Module/checkout/summary/custom_total'
            },
            totals: quote.getTotals(),
            isTaxDisplayedInGrandTotal:
window.checkoutConfig.includeTaxInGrandTotal || false,

            isDisplayed: function() {
                return this.isFullMode() && this.getPureValue() !==
0;
            },

            getValue: function() {
                var price = 0;
                if (this.totals()) {
                    price = totals.getSegment('custom_total').value;
                }
                return this.getFormattedPrice(price);
            },
            getPureValue: function() {
                var price = 0;
                if (this.totals()) {
                    price = totals.getSegment('custom_total').value;
                }
                return price;
            }
        });
    }
);
```

getValue and getPureValue methods return the values of our custom total, but getValue method formats the value, adding two decimal digits and the actual currency symbol.

<module_dir>/view/frontend/web/template/checkout/summary/custom_total.html

```
<!-- ko if: isDisplayed() -->
<tr class="totals custom_total excl">
```

```
    <th class="mark" scope="row">
        <span class="label" data-bind="text: title"></span>
        <span class="value" data-bind="text: getValue()"></span>
    </th>
    <td class="amount">
        <span class="price" data-bind="text: getValue(), attr:
{'data-th': title}"></span>
    </td>
</tr>
<!-- /ko -->
```

Order Emails

To add a new total display into order email, add a new block in sales_email_order_items layout.

<module_dir>/view/frontend/layout/sales_email_order_items.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<page xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:View/Layout/etc/
page_configuration.xsd">
   <body>
       <referenceBlock name="order_totals">
           <block class="Vendor\Module\Block\Order\CustomTotal"
name="order.totals.custom" />
       </referenceBlock>
   </body>
</page>
```

<module_dir>/Block/Order/CustomTotal.php

```
<?php
namespace Vendor\Module\Block\Order;
class CustomTotal extends
\Magento\Framework\View\Element\AbstractBlock
{
   public function initTotals()
   {
       $orderTotalsBlock = $this->getParentBlock();
       $order = $orderTotalsBlock->getOrder();
       if ($order->getCustomAmount() > 0) {
           $orderTotalsBlock->addTotal(new
\Magento\Framework\DataObject([
               'code'       => 'custom_total',
               'label'      => __('Custom Total'),
               'value'      => $order->getCustomAmount(),
               'base_value' => $order->getCustomBaseAmount(),
           ]), 'subtotal');
       }
   }
}
```

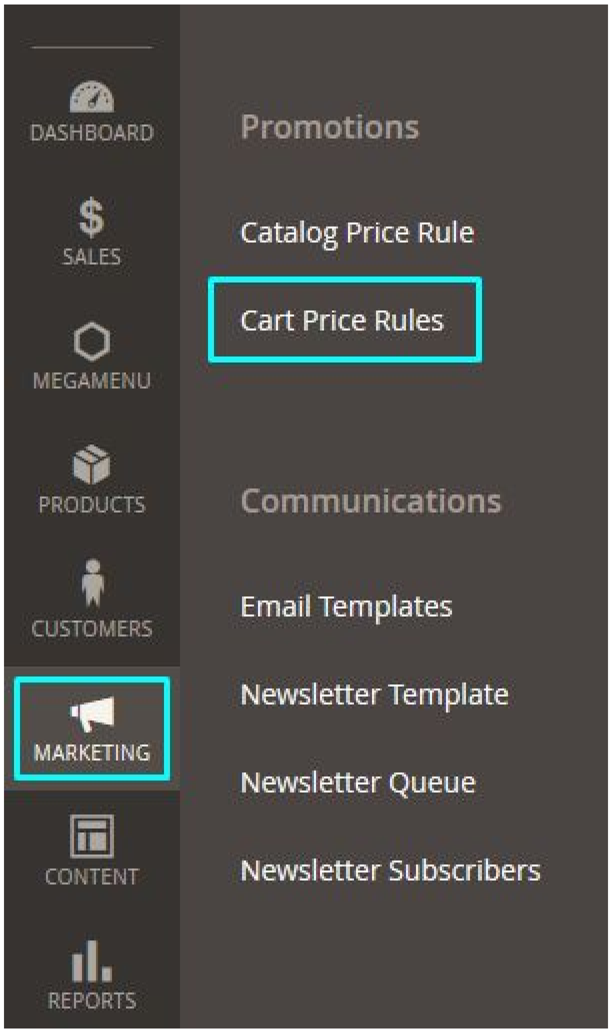
8.3 Demonstrate ability to customize the shopping cart

Describe how to implement shopping cart rules. What is the difference between sales rules and catalog rules?

Cart rules in Magento 2 is a system to set discounts and promotions. Compared to catalog rules, cart rules are applied to the current user's shopping cart, not to each separate product; they may also require that a certain coupon code is entered to get them. Due to this, the discounts are not visible at the catalog pages, but still allow to apply different factors for getting them, like the number of products in the customer's shopping cart, their total price, the categories they belong to, and much more.

Creating cart rules

Commonly, cart rules are created from the admin panel, but they can also be created programmatically.



Log in to the admin panel and navigate to Marketing -> Cart Price Rules. Press "Add New Rule" button and select the rule name, relationship with store and customer groups, conditions for discount and the coupon relevance.

Creating new cart rule programmatically

To create a shipping cart rule programmatically, you have to inject \Magento\SalesRule\Model\RuleFactory class:

```
<?php
....

protected $ruleFactory

public function __construct(
        \Magento\SalesRule\Model\RuleFactory $ruleFactory
    ) {
        $this->rulesFactory = $ruleFactory
    }
```

And then use it like this:

```
<?php
...

$ruleData = [
            "name" => "Custom Cart Rule",
            "description" => "Buy some products and get one more
free",
            "from_date" => null,
            "to_date" => null,
            "uses_per_customer" => "0",
            "is_active" => "1",
            "stop_rules_processing" => "0",
            "is_advanced" => "1",
            "product_ids" => null,
            "sort_order" => "0",
            "simple_action" => "buy_x_get_y",
            "discount_amount" => "1.0000",
            "discount_qty" = >null,
            "discount_step" => "3",
            "apply_to_shipping" => "0",
            "times_used" => "0",
            "is_rss" => "1",
            "coupon_type" => "NO_COUPON",
            "use_auto_generation" => "0",
            "uses_per_coupon" => "0",
            "simple_free_shipping" => "0",
            "customer_group_ids" => [0, 1, 2, 3],
            "website_ids" => [1],
            "coupon_code" => null,
            "store_labels" => [],
            "conditions_serialized" => '',
            "actions_serialized" => ''
        ];

$ruleModel = $this->ruleFactory->create();
$ruleModel->setData($ruleData);
$ruleModel->save();
```

Creating conditions and actions for the rule

For the sake of demonstration, let us create the condition that sets a discount for a certain product, if customer has added more than five items of this product.

```
<?php
....

protected $ruleFactory
protected $productFoundConditionFactory;
protected $productConditionFactory;

public function __construct(
        \Magento\SalesRule\Model\RuleFactory $ruleFactory,
        \Magento\SalesRule\Model\Rule\Condition\Product\FoundFactory
$productFoundConditionFactory,
        \Magento\SalesRule\Model\Rule\Condition\ProductFactory
$productConditionFactory,
    ) {
        $this->rulesFactory = $ruleFactory;
        $this->productFoundConditionFactory =
$productFoundConditionFactory;
        $this->productConditionFactory = $productConditionFactory;
    }

...

$discount = '25';
$sku = 'PRODUCT_SKU';

$shoppingCartPriceRule = $this->rulesFactory->create();
$shoppingCartPriceRule->setName('25% off with multiple products - ' .
$sku)
   ->setDescription('Get 25% off with two or more products)
   ->setFromDate('2000-01-01')
   ->setToDate(NULL)
   ->setUsesPerCustomer('0')
   ->setCustomerGroupIds(array('0','1','2','3',))
   ->setIsActive('1')
   ->setStopRulesProcessing('0')
   ->setIsAdvanced('1')
   ->setProductIds(NULL)
   ->setSortOrder('1')
   ->setSimpleAction('by_percent')
   ->setDiscountAmount($discount)
   ->setDiscountQty(NULL)
   ->setDiscountStep('0')
   ->setSimpleFreeShipping('0')
   ->setApplyToShipping('0')
   ->setTimesUsed('0')
   ->setIsRss('0')
   ->setWebsiteIds(array('1',))
   ->setCouponType('1')
```

```
    ->setCouponCode(NULL)
    ->setUsesPerCoupon(NULL);

$productFoundCondition =
$this->productFoundConditionFactory->create()
    ->setType('Magento\SalesRule\Model\Rule\Condition\Product\Found')
    ->setValue(1) // 1 == FOUND
    ->setAggregator('all'); // match ALL conditions

$productCondition = $this->productConditionFactory->create()
    ->setType('Magento\SalesRule\Model\Rule\Condition\Product')
    ->setAttribute('sku')
    ->setOperator('==')
    ->setValue($sku);
$productFoundCondition->addCondition($productCondition);

$shoppingCartPriceRule->getConditions()->addCondition($productFoundCo
ndition);

$skuCondition = $this->productConditionFactory->create()
    ->setType('Magento\SalesRule\Model\Rule\Condition\Product')
    ->setAttribute('sku')
    ->setOperator('==')
    ->setValue($sku);
$shoppingCartPriceRule->getActions()->addCondition($skuCondition);

$qtyCondition = $this->productConditionFactory->create()
    ->setType('Magento\SalesRule\Model\Rule\Condition\Product')
    ->setAttribute('quote_item_qty')
    ->setOperator('>=')
    ->setValue('2');
$shoppingCartPriceRule->getActions()->addCondition($qtyCondition);

$shoppingCartPriceRule->save();
```

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

For each customer, there can be only one active coupon. Rules cannot add other products to the cart and apply to the item they were assigned to
work with.
Cart rules can slow down the process of adding a product to the cart, as well as checkout and shopping cart pages speed. Performance impact from cart rules can increase in case there is a large number of cart rules with a wide area of application (without connection to website or customer group) and without coupons.

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

- Adding from the product page:

Action: \Magento\Checkout\Controller\Cart\Add
Available events: checkout_cart_add_product_complete, checkout_cart_product_add_after, sales_quote_product_add_After, sales_quote_add_item
Based on the passed product and proper amount of cart model, the corresponding quote item is generated.

- Adding from the wishlist:

Action: \Magento\Wishlist\Controller\Index\Cart
Available events: checkout_cart_product_add_after, sales_quote_product_add_after, sales_quote_add_item

- Reorder:

Action: \Magento\Sales\Controller\AbstractController\Reorder
Available events: checkout_cart_product_add_after, sales_quote_product_add_after, sales_quote_add_item

- Quote merge:

Method: \Magento\Quote\Model\Quote::merge
Available events: sales_quote_add_item

Describe the difference in behavior of different product types in the shopping cart. How are configurable and bundle products rendered?

Magento 2 uses renderers to display products in the shopping cart. Each product type (e.g. simple, configurable, bundle) has a renderer; they are registered in the renderer list with layout instructions as child blocks of the \Magento\Framework\View\Element\RendererList block under the name “checkout.cart.item.renderers”.

To the example’s sake, let us examine a renderer for configurable products:

vendor/magento/module-configurable-product/view/frontend/layout/checkout_cart_item_renderers.xml

```
<?xml version="1.0"?>
<!--
/**
 * Copyright (c) Magento, Inc. All rights reserved.
 * See COPYING.txt for license details.
 */
-->
<page xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:View/Layout/etc/
page_configuration.xsd">
    <body>
        <referenceBlock name="checkout.cart.item.renderers">
            <block
class="Magento\ConfigurableProduct\Block\Cart\Item\Renderer\Configura
ble" as="configurable"
template="Magento_Checkout::cart/item/default.phtml">
                <block
```

```
class="Magento\Checkout\Block\Cart\Item\Renderer\Actions"
name="checkout.cart.item.renderers.configurable.actions"
as="actions">
                    <block
class="Magento\Checkout\Block\Cart\Item\Renderer\Actions\Edit"
name="checkout.cart.item.renderers.configurable.actions.edit"
template="Magento_Checkout::cart/item/renderer/actions/edit.phtml"/>
                    <block
class="Magento\Checkout\Block\Cart\Item\Renderer\Actions\Remove"
name="checkout.cart.item.renderers.configurable.actions.remove"
template="Magento_Checkout::cart/item/renderer/actions/remove.phtml"/
>
                </block>
            </block>
        </referenceBlock>
    </body>
</page>
```

Where \Magento\ConfigurableProduct\Block\Cart\Item\Renderer\Configurable class is applied as renderer and contains methods for displaying the child product picture instead of the parent product picture.

How can you create a custom shopping cart renderer?

First, we need to create a block, \Vendor\Module\Block\Cart\CustomRenderer for example, that will be responsible for html content of a product. The product that we need to display is set in \Magento\Checkout\Block\Cart\AbstractCart::getItemHtml block.

```
public function getItemHtml(\Magento\Quote\Model\Quote\Item $item)
     {
     $renderer =
$this->getItemRenderer($item->getProductType())->setItem($item);
     return $renderer->toHtml();
```

```
      }
```

Therefore, the product will be available in the template of our block by calling $product = $block->getItem();

Also, renderer should be recorded in the list:

<module_dir>/view/frontend/layout/checkout_cart_item_renderers.xml

```
<?xml version="1.0"?>
<page xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:View/Layout/etc/
page_configuration.xsd">
    <body>
        <referenceBlock name="checkout.cart.item.renderers">
            <block class="\Vendor\Module\Block\Cart\CustomRenderer"
as="custom-type"
template="Magento_Checkout::cart/item/default.phtml">
                <block
class="Magento\Checkout\Block\Cart\Item\Renderer\Actions"
name="checkout.cart.item.renderers.custom.actions" as="actions">
                    <block
class="Magento\Checkout\Block\Cart\Item\Renderer\Actions\Edit"
name="checkout.cart.item.renderers.custom.actions.edit"
template="Magento_Checkout::cart/item/renderer/actions/edit.phtml"/>
                    <block
class="Magento\Checkout\Block\Cart\Item\Renderer\Actions\Remove"
name="checkout.cart.item.renderers.custom.actions.remove"
template="Magento_Checkout::cart/item/renderer/actions/remove.phtml"/
>
                </block>
            </block>
        </referenceBlock>
    </body>
</page>
```

Afterwards, all the “custom-type” type products will use our class for display.

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

- Modify product quantity
- Update the cart (required after the quantities are modified)
- Modify configurable/bundle product options
- Delete the product
- Place the product into wishlist (if enabled)
- Add / delete the coupon
- Go to checkout
- Go to multi shipping settings (if enabled)

How can you customize cart edit functionality?

Let checkout/cart/configure call action \Magento\Checkout\Controller\Cart\Configure, which, in its turn, allows the user to edit the product, already added to the shopping cart. Depending on your wish to customize, you can modify either the action or the corresponding layout / templates.

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

To create an extension that deletes one item if another was deleted, use observer for sales_quote_remove_item event.

<module_dir>/etc/events.xml

```xml
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:Event/etc/events
.xsd">
    <event name="sales_quote_remove_item">
        <observer name="removeCartItem"
instance="Vendor\Module\Observer\RemoveCartItem" />
```

```
    </event>
</config>
```

<module_dir>/Observer/RemoveCartItem.php

```
namespace Vendor\Module\Observer;

class RemoveCartItem implements
\Magento\Framework\Event\ObserverInterface
{

  public function execute(\Magento\Framework\Event\Observer
$observer)
  {
    $deletedItem = $observer->getData('quote_item');
    //We can also check id or sku of the deleted cart item, and
depending on the result, delete the other one.
  }
}
```

How do you add a field to the shipping address?

To add a new field into shipping address, you need to:

1. Add a field into layout.

Because shipping and billing addresses forms are generated dynamically, we can do it with a plugin for \Magento\Checkout\Block\Checkout\LayoutProcessor::process method.

2. Create a JS mixin for sending additional information.

We need to modify Magento_Checkout/js/action/set-shipping-information component.

3. Add a new field into address model.

Using extension attributes
/<module_dir>/etc/extension_attributes.xml

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

<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:Api/etc/extensio
n_attributes.xsd">
    <extension_attributes
for="Magento\Quote\Api\Data\AddressInterface">
        <attribute code="custom_field" type="string" />
    </extension_attributes>
</config>
```

4. Get the value at the server side.

Use getExtensionAttributes method of the Magento\Checkout\Api\Data\ShippingInformationInterface interface.

To learn more, follow the link:
https://devdocs.magento.com/guides/v2.2/howdoi/checkout/checkout_new_field.html

8.4 Demonstrate ability to customize shipping and payment methods

Describe shipping methods architecture. How can you create a new shipping method? What is the relationship between carriers and rates?

Creation of custom shipping method starts with creating system.xml and config.xml files in our module. The first one defines the options, available for the configuration of a new shipping method, while the second one sets the default values.

<module_dir>/adminhtml/system.xml

```
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
xsi:noNamespaceSchemaLocation="urn:magento:module:Magento_Config:etc/
system_file.xsd">
    <system>
        <section id="carriers" translate="label" type="text"
sortOrder="320" showInDefault="1" showInWebsite="1" showInStore="1">
            <group id="customshipping" translate="label" type="text"
sortOrder="0" showInDefault="1" showInWebsite="1" showInStore="1">
                <label>Custom Shipping</label>
                <field id="active" translate="label" type="select"
sortOrder="1" showInDefault="1" showInWebsite="1" showInStore="0">
                    <label>Enabled</label>

<source_model>Magento\Config\Model\Config\Source\Yesno</source_model>
                </field>
                <field id="title" translate="label" type="text"
sortOrder="2" showInDefault="1" showInWebsite="1" showInStore="1">
                    <label>Title</label>
                </field>
                <field id="name" translate="label" type="text"
sortOrder="3" showInDefault="1" showInWebsite="1" showInStore="1">
                    <label>Method Name</label>
                </field>
                <field id="price" translate="label" type="text"
sortOrder="5" showInDefault="1" showInWebsite="1" showInStore="0">
                    <label>Price</label>
                    <validate>validate-number
validate-zero-or-greater</validate>
                </field>
                <field id="sort_order" translate="label" type="text"
sortOrder="100" showInDefault="1" showInWebsite="1" showInStore="0">
                    <label>Sort Order</label>
                </field>
                <field id="showmethod" translate="label"
type="select" sortOrder="92" showInDefault="1" showInWebsite="1"
showInStore="0">
                    <label>Show Method if Not Applicable</label>
```

```
<source_model>Magento\Config\Model\Config\Source\Yesno</source_model>
                </field>
            </group>
        </section>
    </system>
</config>
```

Required parameters for each shipping method:

- `active` – shipping method enabled / disabled.
- `model` – path to shipping method model.
- `title` – shipping carrier name; the parameter displays at the front end.
- `sallowspecific` – determines whether the shipping method is available for all countries, or for the certain ones.
- `sort_order` – the order of shipping methods display in the list.

We also use several additional configuration parameters:

- `price`;
- `name`;
- `showmethod` – whether the shipping method is displayed even if it can not be applied to the actual cart / customer.

Config.xml file sets default values for the parameters from system.xml file.

```
<module_dir>/etc/config.xml
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:module:Magento_Store:etc/c
onfig.xsd">
    <default>
        <carriers>
            <customshipping>
                <active>1</active>

<model>Vendor\Module\Model\Carrier\CustomShipping</model>
                <name>Custom Shipping</name>
                <price>5.00</price>
                <title>Custom Shipping</title>
                <sallowspecific>0</sallowspecific>
                <sort_order>100</sort_order>
            </customshipping>
        </carriers>
    </default>
</config>
```

When configuration files are created, we create our shipping method model:

<module_dir>/Model/Carrier/CustomShipping.php

```
<?php

namespace Vendor\Module\Model\Carrier;

class CustomShipping
    extends \Magento\Shipping\Model\Carrier\AbstractCarrier
    implements \Magento\Shipping\Model\Carrier\CarrierInterface
{
    /**
     * Constant defining shipping code for method
     */
    const SHIPPING_CODE = 'customshipping';

    /**
     * @var string
     */
    protected $_code = self::SHIPPING_CODE;

    /**
     * @var \Magento\Shipping\Model\Rate\ResultFactory
     */
    protected $rateResultFactory;

    /**
     * @var
\Magento\Quote\Model\Quote\Address\RateResult\MethodFactory
     */
    protected $rateMethodFactory;

    /**
      * @param \Magento\Framework\App\Config\ScopeConfigInterface
$scopeConfig
      * @param
\Magento\Quote\Model\Quote\Address\RateResult\ErrorFactory
$rateErrorFactory
      * @param \Psr\Log\LoggerInterface $logger
      * @param \Magento\Shipping\Model\Rate\ResultFactory
$rateResultFactory
      * @param
\Magento\Quote\Model\Quote\Address\RateResult\MethodFactory
$rateMethodFactory
      * @param array $data
      */
    public function __construct(
        \Magento\Framework\App\Config\ScopeConfigInterface
$scopeConfig,
        \Magento\Quote\Model\Quote\Address\RateResult\ErrorFactory
$rateErrorFactory,
        \Psr\Log\LoggerInterface $logger,
        \Magento\Shipping\Model\Rate\ResultFactory
$rateResultFactory,
        \Magento\Quote\Model\Quote\Address\RateResult\MethodFactory
$rateMethodFactory,
        array $data = []
    ) {
        $this->rateResultFactory = $rateResultFactory;
        $this->rateMethodFactory = $rateMethodFactory;
        parent::__construct($scopeConfig, $rateErrorFactory, $logger,
$data);
    }

    /**
      * @return array
      */
    public function getAllowedMethods()
    {
        return [self::SHIPPING_CODE => $this->getConfigData('name')];
    }

    /**
     * @param \Magento\Quote\Model\Quote\Address\RateRequest
$request
     * @return bool|\Magento\Shipping\Model\Rate\Result
     */
    public function
collectRates(\Magento\Quote\Model\Quote\Address\RateRequest
$request)
    {
        if (!$this->getActiveFlag()) {
            return false;
        }

        /** @var \Magento\Shipping\Model\Rate\Result $result */
        $result = $this->rateResultFactory->create();

        /** @var \Magento\Quote\Model\Quote\Address\RateResult\Method
$method */
        $method = $this->rateMethodFactory->create();

        $method->setCarrier(self::SHIPPING_CODE);
        // Get the title from the configuration, as defined in
system.xml
        $method->setCarrierTitle($this->getConfigData('title'));

        $method->setMethod(self::SHIPPING_CODE);
        // Get the title from the configuration, as defined in
system.xml
        $method->setMethodTitle($this->getConfigData('name'));

        // Get the price from the configuration, as defined in
system.xml
```

```
        $amount = $this->getConfigData('price');

        $method->setPrice($amount);
        $method->setCost($amount);

        $result->append($method);

        return $result;
    }
}
```

Shipping method model should inherit \Magento\Shipping\Model\Carrier\AbstractCarrier class and realize \Magento\Shipping\Model\Carrier\CarrierInterface interface; it also should have at least two methods:

- `getAllowedMethods`:
  - Should return the options array, available for our method (standard or fast delivery).
- `collectRates`:
  - When returned, false deletes the given shipping method from the list of available ones.
  - An instance of \Magento\Shipping\Model\Rate\Result with the list of available methods.

$_code parameters in this model should contain a unique shipping method code; in our case, it's "customshipping".
$this->getConfigData method allows to get the configuration values for our method (based on the available in system.xml file).

Each shipping carrier can contain one or several shipping rates.

Describe how to troubleshoot shipping methods and rate results. Where do shipping rates come from? How can you debug the wrong shipping rate being returned?

collectRates method can be applied to check the availability of the given method for a certain customer. For instance, if we need to limit the method availability to certain postcodes only, this check should be performed there.

This method can be used to set up shipping methods. The call is performed in collectCarrierRates method of the \Magento\Shipping\Model\Shipping class.

Describe how to troubleshoot payment methods. What are the different payment flows?

In Magento 2, there are three payment method types:

1. Gateway - payment data is passed first into Magento, and then - to the merchant. To enhance safety, use payment data tokenization.
2. Offline - the payment method that does not provide for the connection with any third-party payment provider. Examples: Check/Money Order, Bank Transfer, Purchase Order and Cash on Delivery
3. Hosted - redirects the customer to the payment page that is not a part of Magento 2.

Each gateway payment method is divided into a different number of commands:

- fetch_transaction_information
- order
- authorize
- Capture
- refund
- cancel
- void
- acceptPaymen

- denyPaymen

Each command should be realized as a separate class; therefore, payment methods debugging should begin with the class of the corresponding command.

9.1 Demonstrate ability to customize sales operations

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

To modify the process of order processing, use plugin(https://belvg.com/blog/designing-complex-solutions-using-plugins-in-magento-2.html) or observer.

You can create a plugin for either placeOrder or submitQuote function from Magento\Quote\Model\QuoteManagement class or create an observer for each of the events:

```
sales_model_service_quote_submit_before
sales_model_service_quote_submit_success
sales_model_service_quote_submit_failure
checkout_submit_before
checkout_submit_all_after
```

This allows to integrate a custom logic into the order creation process (for example, sending the data to the third-party ERP system side).

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

Magento 2 has a system of states and statuses that influences the order processing. The difference between a status and a state lies in the following. State is the actual position in order processing flow, so state influences the possible actions during the order processing. For example, you can create an invoice or ship during the state processing, and the order status does not have an effect on any of the actions (with the exception of third-party modules' logic).

State can have several statuses which allows to describe the order process more flexibly. The connection between state and status is stored in sales_order_status_state table.

To modify states or statuses programmatically, use setStatus and setState methods. To add a new record into the order history, apply addStatusToHistory method.

```
$order->setState(\Magento\Sales\Model\Order::STATE_PROCESSING);
$order->setStatus('processing');
$order->addStatusToHistory($order->getStatus(), 'Custom Message');
$order->save();
```

You can create a new status via the admin panel (navigate to Stores -> Order Status) or via setup script (\Magento\Sales\Setup\Patch\Data\InstallOrderStatusesAndInitialSalesConfig class). Adding a new state is possible only via setup script.

Working with an order depending on its state is hardcoded in \Magento\Sales\Model\Order class. To modify their behavior, create plugins for this class methods and, if possible, other methods in Magento_Sales module.

Describe how to customize invoices. How would you customize invoice generation, capturing, and management?

To create an Invoice, use \Magento\Sales\Model\Service\InvoiceService class and its method prepareInvoice that utilizes \Magento\Sales\Model\Order\Invoice. If you need to introduce a custom logic into the invoice creating process, use one of the following events: sales_order_invoice_pay, sales_order_invoice_cancel, sales_order_invoice_register or create a plugin for \Magento\Sales\Model\Service\InvoiceService::prepareInvoice method.

Invoice can have one of the following states:
STATE_OPEN
STATE_PAID
STATE_CANCELED

It can also have two types - online invoice and offline invoice.
Online invoice calls the capture method for payment method, which, in its turn, sends a query to the payment system. Offline invoice modifies the payment information only on Magento side.

Describe refund functionality in Magento. Which refund types are available, and how are they used?

Credit Memo is responsible in Magento 2 for the refund, allowing to return the order partially or completely. Also, refund can be offline and online (depending on the order type). The difference between offline and online lies in the following: offline refund is performed on Magento side and does not send any requests to the payment processing system, while online refund sends a query to the payment system.

Example of _void method in \Magento\Sales\Model\Order\Payment :

```
  protected function _void($isOnline, $amount = null,
$gatewayCallback = 'void')
    {
        $order = $this->getOrder();
        $authTransaction = $this->getAuthorizationTransaction();
        $this->setTransactionId(
            $this->transactionManager->generateTransactionId($this,
Transaction::TYPE_VOID, $authTransaction)
        );
        $this->setShouldCloseParentTransaction(true);

        // attempt to void
        if ($isOnline) {
            $method = $this->getMethodInstance();
            $method->setStore($order->getStoreId());
            $method->{$gatewayCallback}($this);
        }
```
Magento 2 Enterprise Edition also allows to perform the refund into Store Credits, with which a customer can later pay for other items in this store.

10.1 Demonstrate ability to customize My Account

Magento_Customer module is responsible for the user account functioning. If you need to modify in some way the user account (for instance, add new elements, delete the existing ones or change the position of blocks), you need to modify the Magento_Customer module components. We can do this using the modules or our own theme. For the sake of clarity, we will demonstrate everything that we describe on our custom theme; the procedures will be similar for modules as well. We will use the side menu modification in the user account as an example.

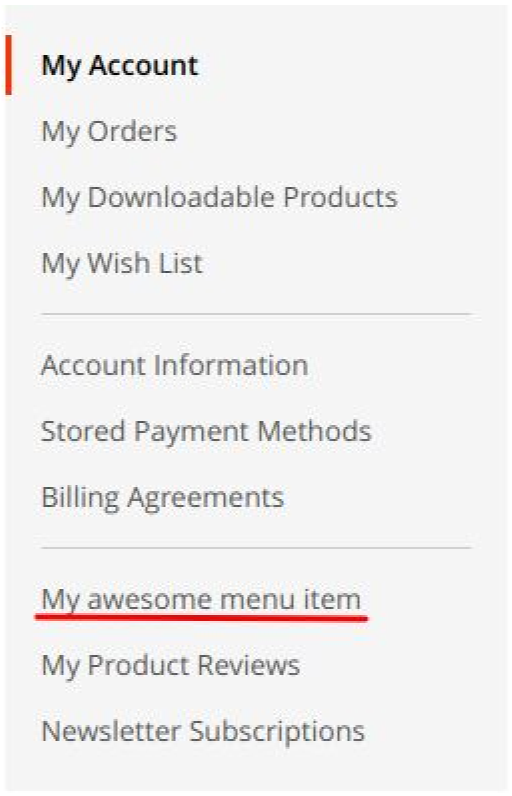
Describe how to customize the "My Account" section. How do you add a menu item?

In order to modify the side menu, we need to modify the customer_account_navigation block from customer_account.xml layout-file. For this, we need to create a customer_account.xml file and place it in our theme at the following path: Magento_Customer/layout/. Adding instructions in this file, we can add or delete new menu item, and change their position. To add a new menu item, add the following instruction into customer_account.xml file:

```
<referenceBlock name="customer_account_navigation">
   <block class="Magento\Customer\Block\Account\SortLinkInterface"
name="customer-account-navigation-address-link">
       <arguments>
           <argument name="label" xsi:type="string"
translate="true">My awesome menu item</argument>
           <argument name="path"
xsi:type="string">path/i/need</argument>
```

```
            <argument name="sortOrder"
xsi:type="number">100</argument>
        </arguments>
    </block>
</referenceBlock>
```

We add a new block of the Magento\Customer\Block\Account\SortLinkInterface type and pass the menu item name, page link and sortOrder parameter as an argument, if we want our menu item to be placed in a certain position. As a result, a new menu item will appear in the user menu:



To delete the existing menu item, apply the following instruction:

```
<referenceBlock name="name-of-the-link-to-remove" remove="true"/>
```

As a parameter, we need to pass the name of the block we want to delete. For example, let us delete the newsletter subscriptions page:

```
<referenceBlock
name="customer-account-navigation-newsletter-subscriptions-link"
remove="true"/>
```

Finally, when we need to change the block order, we can modify the sortOrder argument value. For example, let us make the Wishlist section the first menu item:

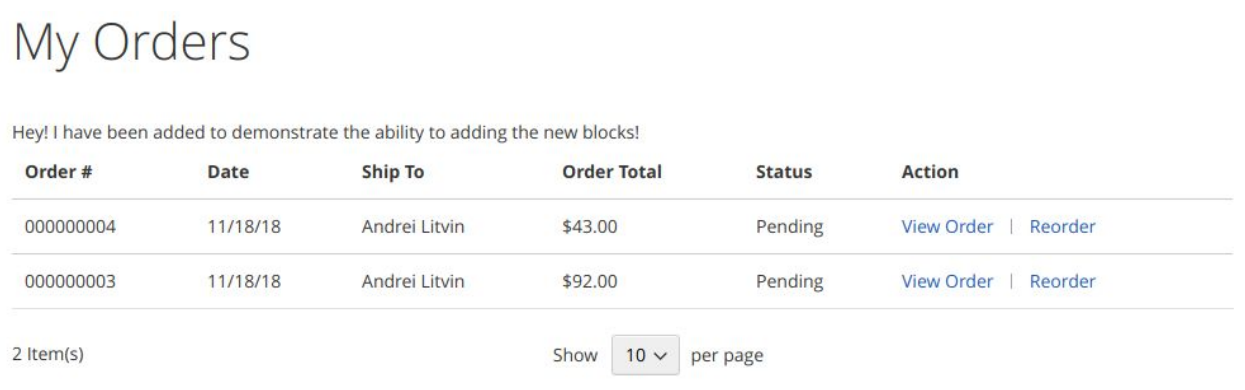
```
<referenceBlock name="customer-account-navigation-wish-list-link">
   <arguments>
       <argument name="sortOrder" xsi:type="number">500</argument>
   </arguments>
</referenceBlock>
```

How would you customize the “Order History” page?

To customize My Orders page, modify the layout-file sales_order_history.xml of the Magento_Sales module. For this, create a sales_order_history.xml file and place it in our theme at the following path: Magento_Sales/layout/. Then, apply any instructions, provided in Magento 2 for layout customization. For example, let us add a text block before the orders list:

```
<referenceContainer name="sales.order.history.info">
   <block class="Magento\Framework\View\Element\Text" name="my_text">
       <arguments>
           <argument name="text" xsi:type="string">Hey! I have been
added to demonstrate the ability to adding the new blocks!</argument>
       </arguments>
   </block>
</referenceContainer>
```

As a result, on the My Orders page in the user account we will see the following text:


My Orders

Hey! I have been added to demonstrate the ability to adding the new blocks!

Order #	Date	Ship To	Order Total	Status	Action
000000004	11/18/18	Andrei Litvin	$43.00	Pending	View Order | Reorder
000000003	11/18/18	Andrei Litvin	$92.00	Pending	View Order | Reorder

2 Item(s) Show 10 per page

To modify the order table contents (for example, add a new column), we need to override the vendor/magento/module-sales/view/frontend/templates/order/history.phtml template.

10.2 Demonstrate ability to customize customer functionality

Describe how to add or modify customer attributes.

To create a new customer attribute, use setup scripts. This is the example of customer attribute creation:

```
...
class InstallData implements
\Magento\Framework\Setup\InstallDataInterface
{
    private $customerSetupFactory;
    private $attributeSetFactory;
    public function __construct(
\Magento\Customer\Setup\CustomerSetupFactory $customerSetupFactory,
\Magento\Eav\Model\Entity\Attribute\SetFactory $attributeSetFactory
)    {
        $this->customerSetupFactory = $customerSetupFactory;
        $this->attributeSetFactory = $attributeSetFactory;
    }

    public function install(
\Magento\Framework\Setup\ModuleDataSetupInterface $setup,
\Magento\Framework\Setup\ModuleContextInterface $context
)    {
        $installer = $setup;
        $installer->startSetup();
        $customerSetup = $this->customerSetupFactory->create(['setup'
=> $setup]);

$customerSetup->addAttribute(\Magento\Customer\Model\Customer::ENTITY
, 'new_attribute', [
            'type' => 'varchar',
            'label' => 'new attribute',
            'input' => 'text',
            'required' => false,
            'visible' => true,
            'user_defined' => true,
            'system' => false,
                'used_in_forms' => [
                    'adminhtml_customer',
                    'adminhtml_checkout',
                    'checkout_register',
                    'customer_account_create',
                    'customer_account_edit',
                ]
            ]);
     }
...
```

Customer attributes can be displayed in different forms, which are set in used_in_forms parameter. The list of forms can be found in customer_form_attribute table.

To modify the attribute, use \Magento\Customer\Setup\CustomerSetupFactory. We can also use updateAttribute method or load the attribute using getAttribute method and modify the parameter via setData.

```
$customerSetup = $this->customerSetupFactory->create(['setup' =>
$setup]);
$customerSetup->updateAttribute(\Magento\Customer\Model\Customer::ENT
ITY, 'new_attribute', 'visible', false);
```

```
$customerSetup = $this->customerSetupFactory->create(['setup' =>
$setup]);
$attribute = $customerSetup->getEavConfig()->getAttribute(
\Magento\Customer\Model\Customer::ENTITY,
'new_attribute')
->setData('used_in_forms', ['adminhtml_customer']);
$attribute->save();
```

Describe how to extend the customer entity. How would you extend the customer entity using the extension attributes mechanism?

Module developers can not modify API Data interfaces, described in core Magento 2, but the majority of the modules have the extension attributes mechanism. Extension attributes are set and stored separately from the data object of the initial class, so everything, connected with data storage and extraction, must be realized by the developer himself.

To create extension attribute, declare it in the etc/extension_attribute.xml file of the module:

```
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:Api/etc/extensio
n_attributes.xsd"> <extension_attributes
for="Magento\Customer\Api\Data\CustomerInterface"> <attribute
```

```
code="new_attribute" type="string" /> </extension_attributes>
</config>
```

To get or modify extension attributes, apply the system of plugins to save, get, getList methods for Product Repository.

Example of extension_attributes for Customer.

File etc/extension_attributes.xml:

```
<?xml version="1.0" encoding="utf-8" ?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:noNamespaceSchemaLocation="urn:magento:framework:Api/etc/extensio
n_attributes.xsd">
    <extension_attributes
for="Magento\Customer\Api\Data\CustomerInterface">
        <attribute code="ext_customer_attribute" type="string" />
    </extension_attributes>
    </config>

etc/di.xml
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:ObjectManager/et
c/config.xsd">
    <type name="Magento\Customer\Api\CustomerRepositoryInterface">
        <plugin name="extensionAttributeExtCustomerAttribute"
type="Belvg\CustomExtAttribute\Model\Plugin\CustomerRepository" />
    </type>
</config>
```

Model/Plugin/CustomerRepository.php

```
<?php
namespace Belvg\CustomExtAttribute\Model\Plugin\CustomerRepository;
...
```

```
class CustomerRepository
{
...
    public function afterGet(CustomerRepositoryInterface $subject,
CustomerInterface $result)
    {
        $extensionAttributes =
$this->getExtensionAttributes($result);
//custom logic
        $extCustomerAttribute =
$this->customGetExtCustomerAttribute();

$extensionAttributes->setExtCustomerAttribute($extCustomerAttribute)
        return $result;
    }

    public function aroundSave(
        CustomerRepositoryInterface $subject,
        callable $proceed,
        CustomerInterface $customer,
        $passwordHash = null
    ) {
            $extCustomerAttribute =
$customer->getExtensionAttributes()->getExtCustomerAttribute();
            $result = $proceed($customer, $passwordHash);
     $this->customSetExtCustomerAttribute($extCustomerAttribute);
     return $result;
}
```

Describe how to customize the customer address. How would you add another field into the customer address?

Magento has a standard set of attributes for the address, stored in customer_address_entity table. Other attributes of the address need to be added as EAV attributes. They will work via Custom Attributes.

To add new fields into customer address, create an EAV attribute using \Magento\Customer\Setup\CustomerSetupFactory.

```
$customerSetup->addAttribute(\Magento\Customer\Api\AddressMetadataInt
erface::ENTITY_TYPE_ADDRESS, 'custom_address_attribute', [
            'type' => 'varchar',
            'label' => 'Custom address attribute',
            'input' => 'text',
            'required' => false,
            'visible' => true,
            'user_defined' => true,
            'sort_order' => 100,
            'position' => 100,
            'system' => 0,
        ]);

        $customerEntity =
$customerSetup->getEavConfig()->getEntityType(\Magento\Customer\Api\A
ddressMetadataInterface::ENTITY_TYPE_ADDRESS);
        $attributeSetId =
$customerEntity->getDefaultAttributeSetId();

        $attributeSet = $this->attributeSetFactory->create();
        $attributeGroupId =
$attributeSet->getDefaultGroupId($attributeSetId);

        $attribute =
$customerSetup->getEavConfig()->getAttribute(\Magento\Customer\Api\Ad
dressMetadataInterface::ENTITY_TYPE_ADDRESS,
            'district');
        $attribute->addData([
                'attribute_set_id' => $attributeSetId,
                'attribute_group_id' => $attributeGroupId,
                'used_in_forms' =>
                    [
                        'adminhtml_customer_address',
                        'customer_address_edit',
                        'customer_register_address',
                    ]
            ]
        );

        $attribute->save();
```

Describe customer groups and their role in different business processes. What is the role of customer groups? What functionality do they affect?

User groups allow to modify taxes and discounts, create separate price rules for different product groups, as well as separate their rights at the store side. Different product groups have different cache for blocks.

There are the following default user groups:

1) NOT LOGGED IN
2) General
3) Wholesale
4) Retailer

NOT LOGGED IN is the only user group that you can not delete (same as you can not delete the default registered users group, but, in contrast to NOT LOGGED IN, it is not static, which means it can be modified). NOT LOGGED IN is assigned to all the visitors without a session and determines the type of shopping cart (guest cart).
General is the default group for the unlogged users.

Describe Magento functionality related to VAT. How do you customize VAT functionality?

Magento is a built-in functionality for working with VAT. VAT depends on seller's country and buyer's address. When a downloadable product is purchased, VAT depends solely on the delivery destination.

To configure VAT, you can navigate to the following path:

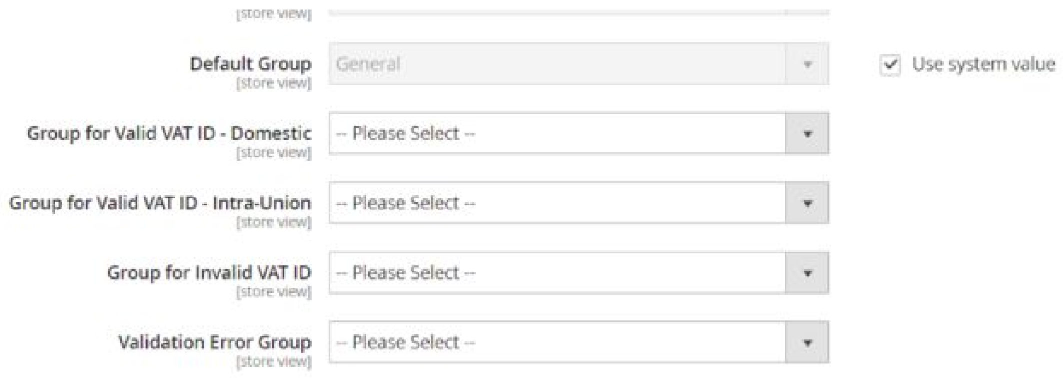
Stores > Configuration > General > General > Store Information
VAT Number - this is where you set the seller's VAT number.

Customers > All Customers > Edit Customer
Account Information > Tax/VAT Number - if set, then VAT will be calculated based on the given field.
Addresses > VAT Number - if selected, VAT will be calculated with VAT taken into account.

Configure > Customers > Customer Configuration
Show VAT Number on Storefront - allows a customer to set VAT Number when the account is created or edited.

Default Value for Disable Automatic Group Changes Based on VAT ID - when selected, it automatically changes the user group after the VAT Number is validated.

The parameters you see at the screenshot below are responsible for group selection after the validation, if the "Default Value for Disable Automatic Group Changes Based on VAT ID" parameter is active.



Default Group [store view]	General	✓ Use system value
Group for Valid VAT ID - Domestic [store view]	-- Please Select --	
Group for Valid VAT ID - Intra-Union [store view]	-- Please Select --	
Group for Invalid VAT ID [store view]	-- Please Select --	
Validation Error Group [store view]	-- Please Select --	

For the correct functioning of VAT in Magento, set the user groups and create the rules and rates for TAX.
Tax can be applied separately to products and users with Product Tax Classes and Customer Tax Classes (Stores > Tax Rules); you can also create taxes for certain areas with Tax Zones and Rates(Stores > Tax Zones and Rates).
To modify Tax, apply sales_quote_collect_totals_before event that calculates the order’s total cost.