Magento Training

Yash Shah
M - 9898402533

Overview

This document contents the key understanding of Magento 2 Architecture, Concepts, Features, Request Flow Processing and examples on how to create a module from scratch.

Contents

g. Understanding of Magento Frontend
h. Understanding of Magento Backend
i. Introduction to
    i. Elasticsearch
    ii. Redis
    iii. Nginx/Apache
    iv. Varnish
j. Introduction to
    i. Docker
    ii. Vagrant box

2. Magento Architecture

a. Magento Directory Structure
b. Magento Module bases Architecture
c. Different Areas of Magento
d. Magento Configuration Files
e. Magento System Configuration with usage of Scope
f. What is Dependency Injection and how to use it
    i. Plugins
    ii. Preference
    iii. Type
    iv. Virtual Type
g. Magento Event Observers
h. Magento Cron Jobs
i. Magento CLI commands

3. Request Flow Processing

a. Available modes in Magento
b. Difference between all modes of Magento
c. Routing in Magento
d. Different types of Controller Response options available in Magento
e. Demonstrate how to use URL rewrites for a catalog product view to a different URL

4. Magento Module Creation

a. Locations in Magento to create module
b. Files needed to create a new module
c. Enable/Disable a module
d. Module Dependencies and its usage
e. What is Factory Class, Proxy Class and what is the difference between them

5. Customizing the Magento UI

a. Demonstrate the ability to customize the Magento UI using themes
b. Demonstrate an ability to create UI customizations using a combination of a block and template
c. Identify the uses of different types of blocks
d. Describe the elements of the Magento layout XML schema, including the major XML directives
e. Create and add code and markup to a given page

6. Working with Databases in Magento

a. Describe the basic concepts of models, resource models, and collections
b. Describe how entity load and save occurs
c. Describe how to filter, sort, and specify the selected values for collections and repositories
d. Write install and upgrade scripts
e. Identify how to use the DDL class in setup scripts
f. Declarative schema usage

7. Developing with Adminhtml

a. Create a controller for an admin router
b. Define basic terms and elements of system configuration, including scopes, website/store/store view
c. Define basic terms and elements of system configuration, including scopes, website/store/store view
d. Set up a menu item
e. Create appropriate permissions for users
f. Admin Grids and Forms

8. Customizing Magento Business Logic

a. Identify/describe standard product types (simple, configurable, bundled, etc.) Describe category properties in Magento

b. Define how products are related to the category Describe the difference in behavior of different product types in the cart

c. Inventory
    i. MSI
    ii. Sources
    iii. Stocks

d. Customers
    i. Customers
    ii. Customer Groups
    iii. Customer Segment
    iv. Store Credit
    v. Rewards

e. Marketing
    i. Catalog Rules
    ii. Cart Rules
    iii. Related Product Rules
    iv. Gift Card Accounts
    v. Newsletters
    vi. Emails
    vii. URL Rewrites

f. CMS
    i. Pages
    ii. Static Blocks
    iii. Widgets
    iv. Page Builder
    v. Themes

g. Reports

h. Sales
    i. Orders

ii. Invoices
iii. Shipments
iv. Returns
v. Credit Memo
vi. Shipping Methods
vii. Order State and Status

i. Describe native shipment functionality in Magento
j. Describe and customize operations available in the customer account area
k. Add or modify customer attributes
l. Customize the customer address

9. API

a. What is API & why is it needed
b. API in Magento
c. Authentication Methods
d. API Authorization
e. Service and Data Interfaces
f. API Response Formats
g. REST
h. SOAP
i. GraphQL

10. Development Practices

Magento Introduction

What is Magento ?

Magento is an open-source e-commerce platform written in PHP. It is one of the most popular open e-commerce systems in the network. This software is created using the Zend Framework.

The software was originally developed by Varien, Inc, a US private company headquartered in Culver City, California, with assistance from volunteers.

More than 100,000 online stores have been created on this platform. The platform code has been downloaded more than 2.5 million times, and $155 billion worth of goods have been sold through Magento-based systems in 2019. Two years ago, Magento accounted for about 30% of the total market share.

Varien published the first general-availability release of the software on March 31, 2008. Roy Rubin, the former CEO of Varien, later sold a share of the company to eBay, which eventually completely acquired and then sold the company to Permira;[5] Permira later sold it to Adobe.

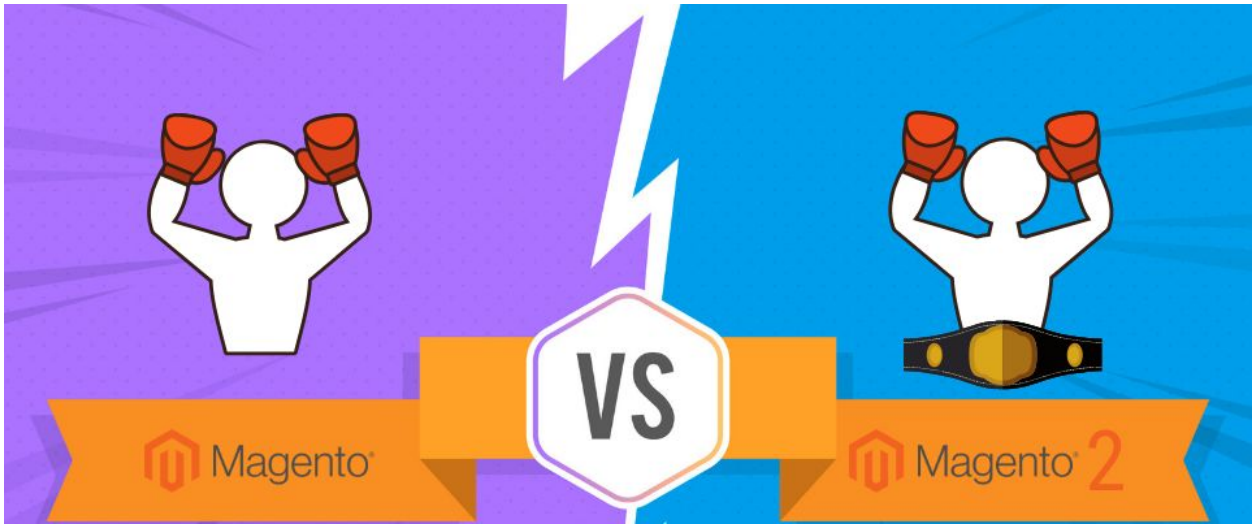
In May 2018 it was announced that Magento would be acquired by Adobe for $1.68bn with a view to integrating it into Adobe Experience Cloud, its Enterprise CMS platform. The acquisition was finalized on June 19, 2018.

On November 17, 2015, Magento 2.0 was released. Among the features changed in V2 are the following: reduced table locking issues, improved page caching, enterprise-grade scalability, inbuilt rich snippets for structured data, new file structure with easier customization, CSS Preprocessing using LESS & CSS URL resolver, improved performance and a more structured code base. Magento employs the MySQL or MariaDB relational database management system, the PHP programming language, and elements of the Zend Framework. It applies the conventions of object-oriented programming and model–view–controller architecture. Magento also uses the entity–attribute–value model to store data. On top of that, Magento 2 introduced the Model-View-ViewModel pattern to its front-end code using the JavaScript library Knockout.js

Reference Link:

https://en.wikipedia.org/wiki/Magento

Magento 1 vs Magento2

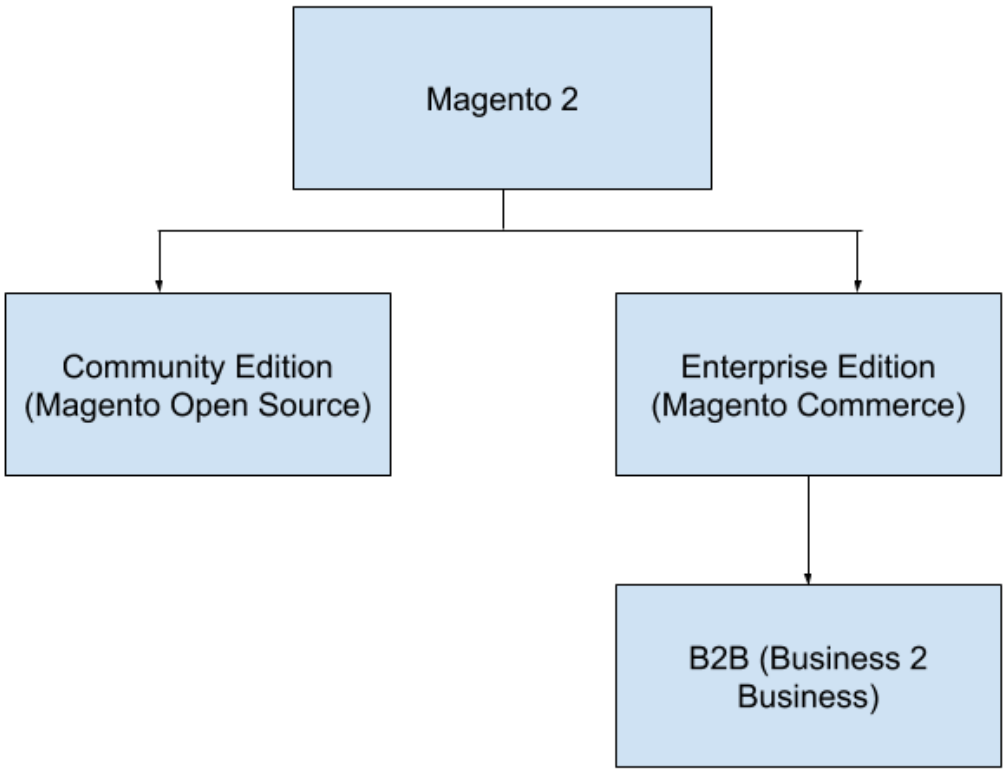


Zend Framework 1	Zend framework 1 and 2
PHP 5.2.x – 5.5.x	PHP 5.6.x / 7.x, this includes various security patches, code and speed improvements
Varnish is not supported	Varnish is compatible with default setup
HTML and CSS	HTML5 and CSS3
Do not support composer	Composer compatible
Old Admin UI	New improved Admin UI
Slower Page load time	50% faster page load time with support of caching
Old module architecture so design files has to be placed in design directory	New architecture so that all the files within a module can be placed in a single directory

No plugins or interceptors that created code trouble between multiple modules	Plugin or interceptor was introduced
APIs were introduced retroactively	API is a core framework feature with Service Contract Architecture
Lower Enterprise Edition Cost, starts from USD $ 18000	Higher Enterprise Edition Cost, starts from USD $ 22000
Supports Apache server	Supports Apache and Nginx

There are many more, but above are the primary differences between them.

Different Editions of Magento



Magento System Requirements

- **Linux:** Linux distributions, such as RedHat Enterprise Linux (RHEL), CentOS, Ubuntu, Debian, and similar. Magento is not supported on: Windows OS, MAC OS
- **2 GB RAM:** Upgrading the Magento applications and extensions you obtain from Magento Marketplaces and other sources can require up to 2GB of RAM. If you are using a system with less than 2GB of RAM
- **Composer:** Composer is required for developers who wish to contribute to the Magento 2 codebase or anyone who wishes to develop Magento extensions.
- **Apache 2.4 or Nginx 1.x :** In addition, you must enable the Apache mod_rewrite and mod_version modules. The mod_rewrite module enables the server to perform URL rewriting. The mod_version module provides flexible version checking for different httpd versions.
- **Mysql 5.6, 5.7**
- **PHP 7.1+**
- **Required PHP extensions**
    - **ext-bcmath**
    - **ext-ctype**
    - **ext-curl**
    - **ext-dom**
    - **ext-gd**
    - **ext-hash**
    - **ext-iconv**
    - **ext-intl**
    - **ext-mbstring**
    - **ext-openssl**
    - **ext-pdo_mysql**
    - **ext-simplexml**
    - **ext-soap**
    - **ext-spl**
    - **ext-xsl**
    - **ext-zip**
    - **lib-libxml**
- **mcrypt (< PHP 7.2)**

Reference Link:

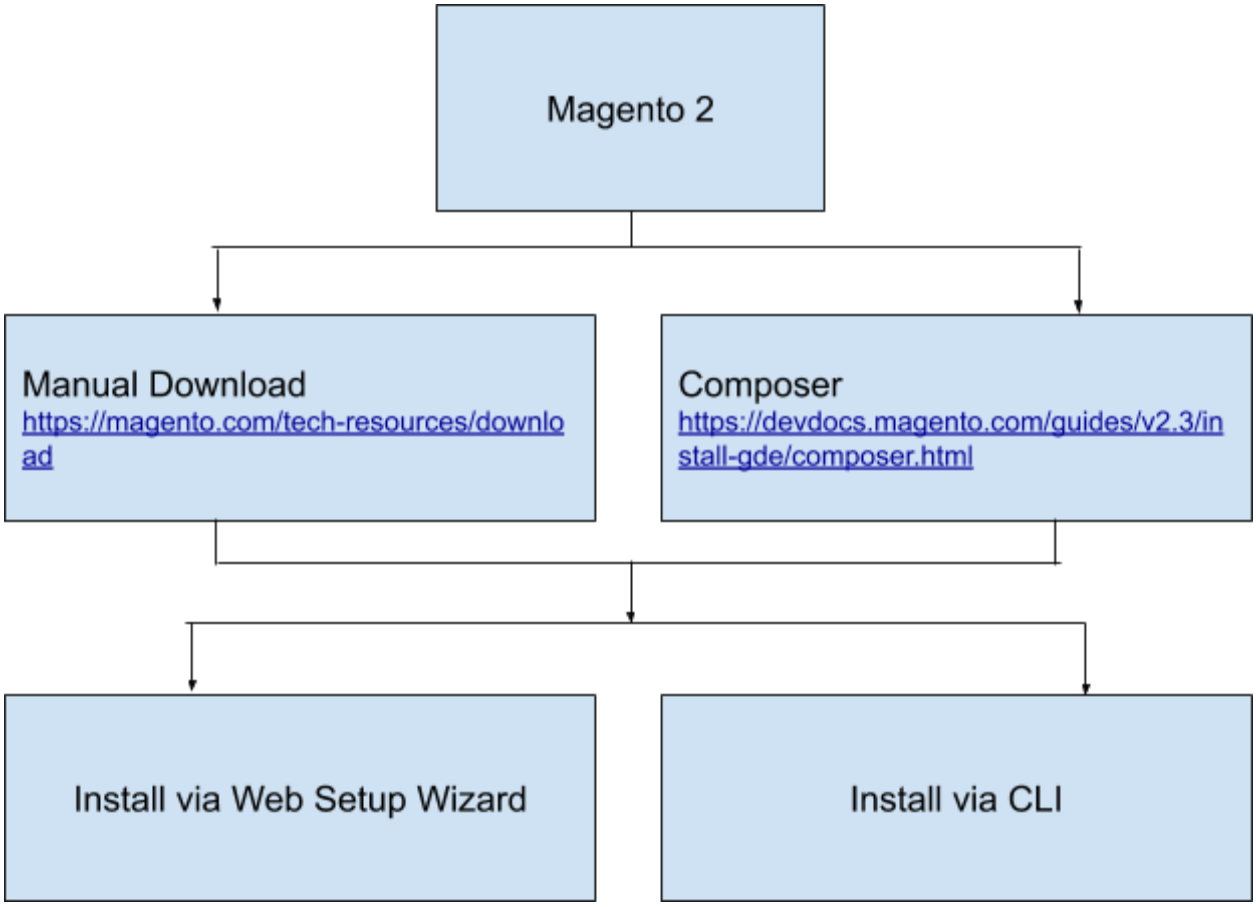
https://devdocs.magento.com/guides/v2.3/install-gde/system-requirements-tech.html

Understand Composer

Reference Link:

https://getcomposer.org/doc/00-intro.md

Download and Install Magento



Well, as shown in the above diagram, there are 2 ways to download magento.

1. **Manual Download**

   In this method, you can goto the below URL and download appropriate Magento version.

   https://magento.com/tech-resources/download

   This will ask you to login before download. Once you login you will be able to download magento source and place it to your installation directory.

2. **Composer**

   This is the most easiest and convenient way to install Magento without any hurdles. You just have to pass the following command in order to download Magento.

To install community edition, enter following command

```
composer create-project --repository=https://repo.magento.com/
magento/project-community-edition[=version] <install-directory-name>
```

To install enterprise edition, enter following command

```
composer create-project --repository=https://repo.magento.com/
magento/project-enterprise-edition[=version]
<install-directory-name>
```

Reference Link:

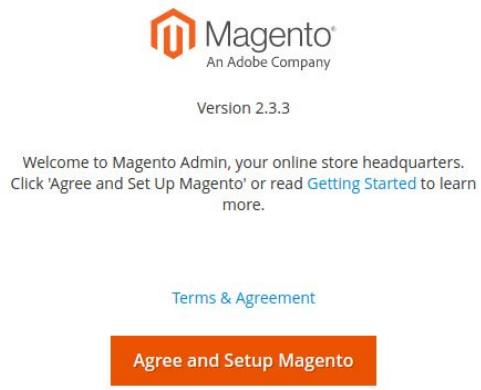
https://devdocs.magento.com/guides/v2.3/install-gde/composer.html

Whatever method you choose from above, It will just download Magento.

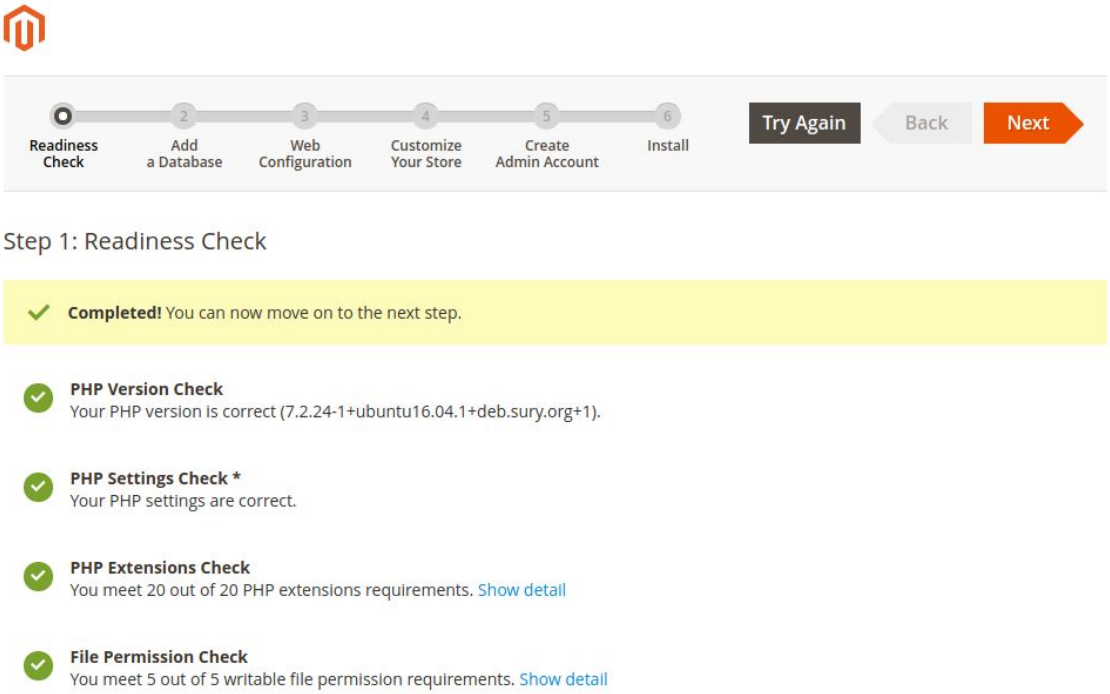
Next you have to install Magento with configuring database, admin credentials, locale timezone etc settings.

There are again 2 ways to install it.

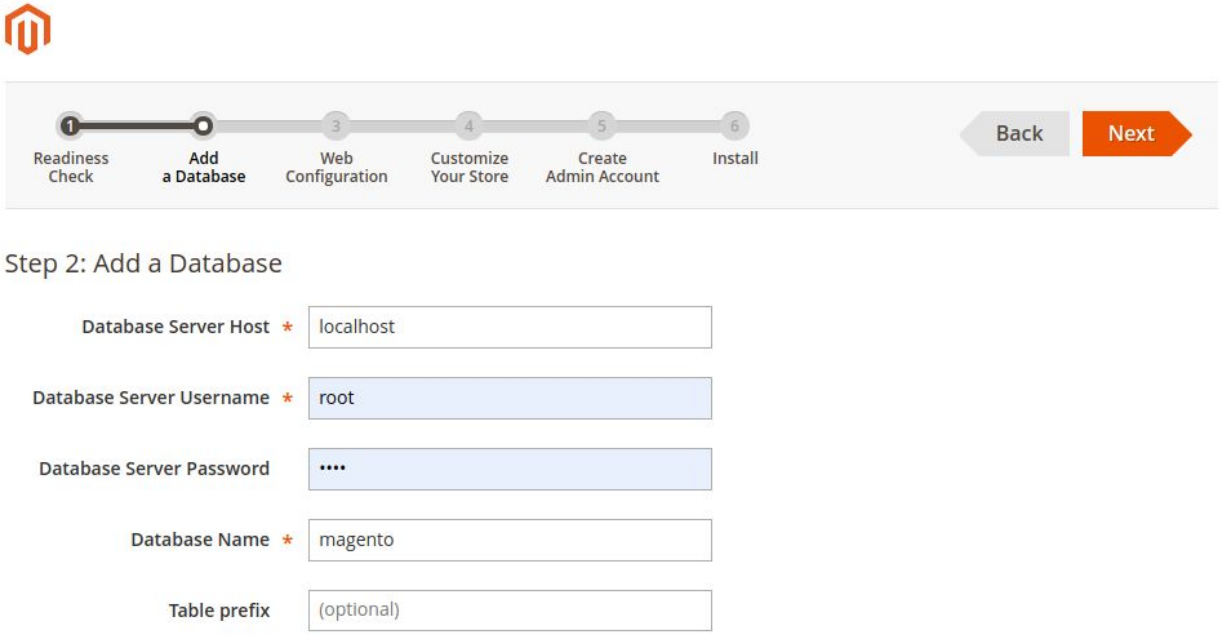
1. **Install via Web Setup Wizard**



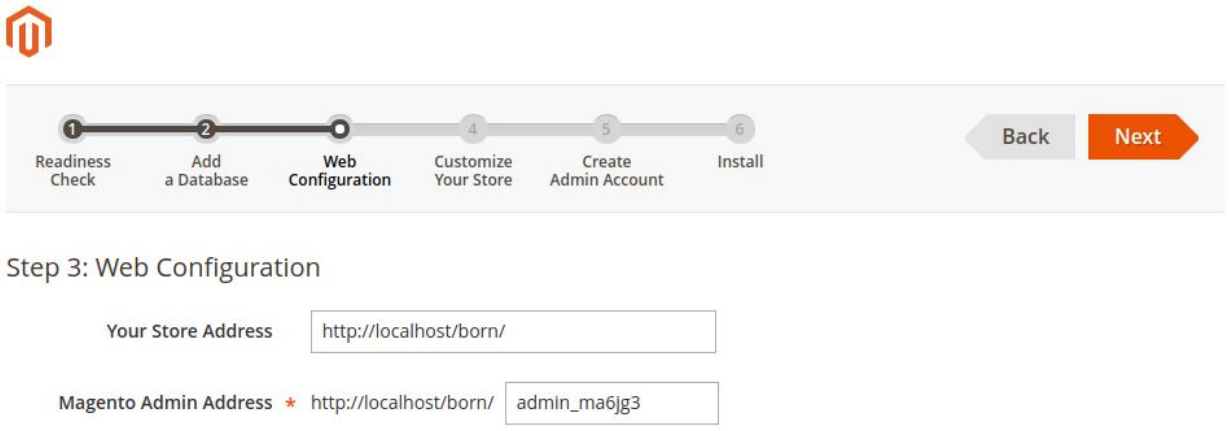
Click on “Agree and Setup Magento”, which will show you next screen.



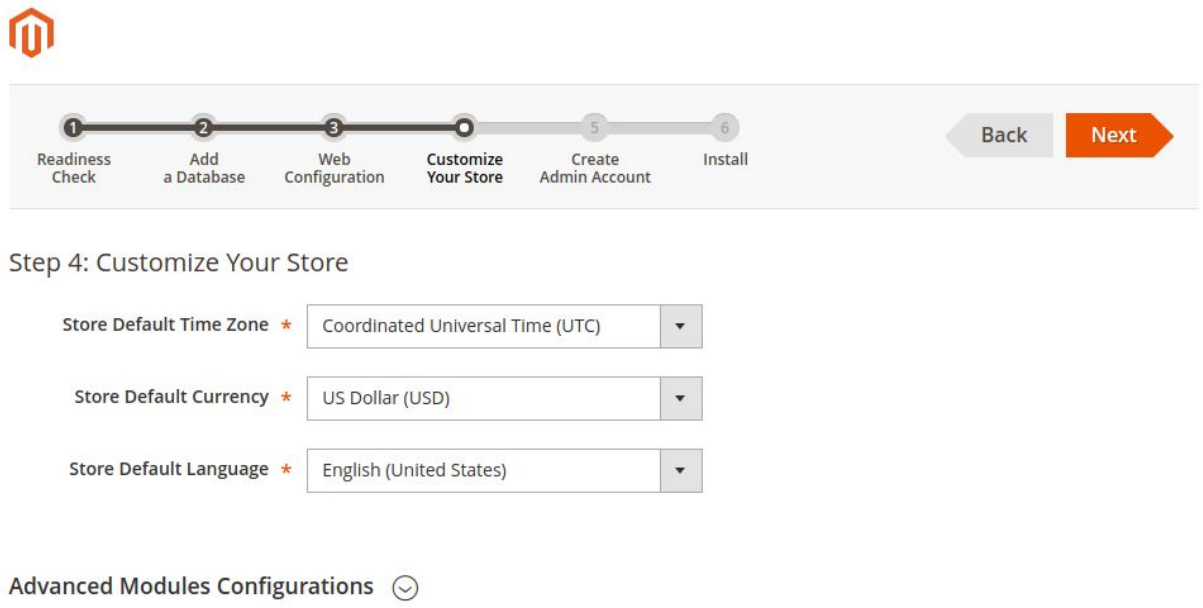
Once you have everything configured properly, you may click on “Next” button.



Configure your database settings here and click on “Next” button.

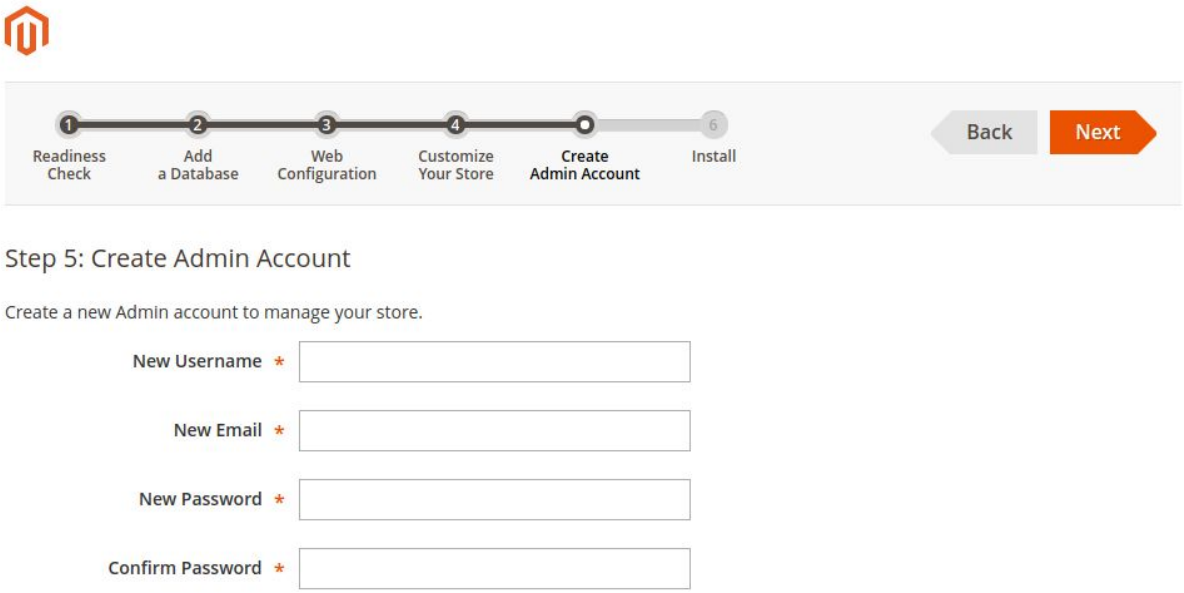


Configure your base URL and admin slug over here and click “Next” button.

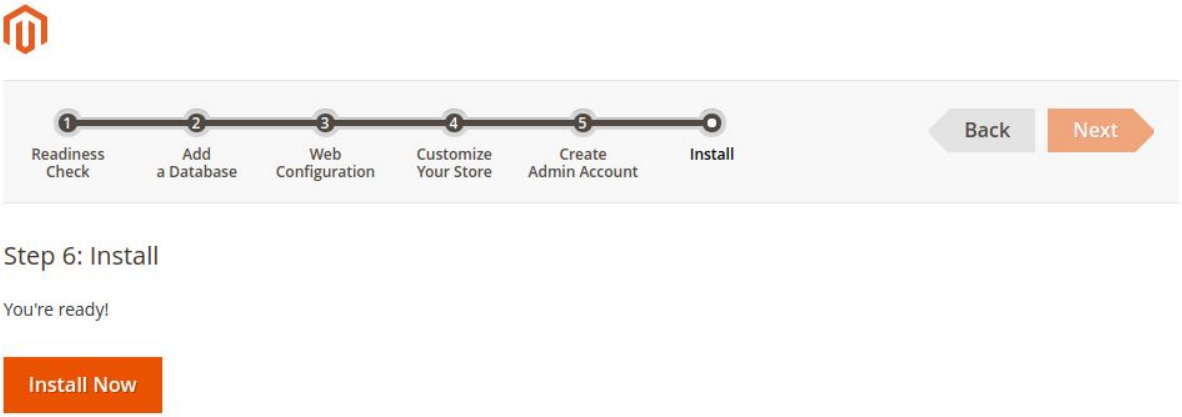


Configure your store timezone, currency and language and click on “Next” button.

There are some “Advanced Module Configuration”, through which you can disable some default modules if not needed.

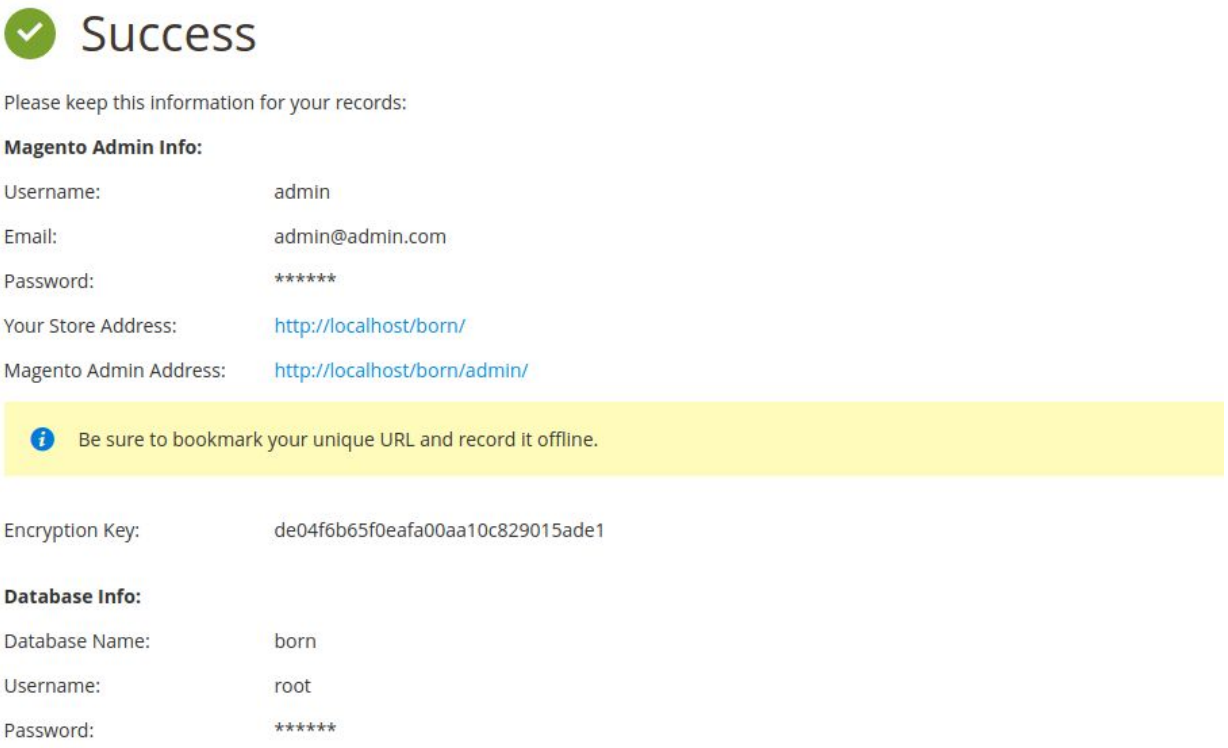


Here you can define admin username, email and password of your store. Fill up the details and click “Next” button.



You are on the last step of installation screen. Just click on Install Now button and system would install Magento with your provided information.

Once the process is done, it will show you below screen.



Now you are ready to play with frontend and backend.

2. **Install via CLI**

This is the most easiest and convenient way to install Magento. In above method, you have to go through so many steps. With this method, you just need to execute below CLI command and it will do everything for you after that.

```
php bin/magento setup:install \
--base-url=http://localhost/magento/ \
--backend-frontname=admin \
--db-host=localhost --db-name=db --db-user=root --db-password=root \
--admin-firstname=Magento --admin-lastname=User --admin-email=user@example.com \
--admin-user=admin --admin-password=admin123 --language=en_US \
--currency=USD --timezone=America/Chicago --use-rewrites=1
```

Just make sure to provide correct parameters for each value and it will do everything for you in one go.

Reference Link:
https://devdocs.magento.com/guides/v2.3/install-gde/composer.html

Assignments:

1. Download Magento 2.3.2 using direct download method
2. Install Magento 2.3.2 using Web Setup Wizard
3. Download Magento 2.3.2 using composer method
4. Install Magento 2.3.2 using CLI method

Understanding of Magento Frontend

Magento 2 Frontend consist of following things.

- Homepage
- Product List Page (PLP)
- Product Details Page (PDP)
- CMS Pages
- Search Results Page
- Shopping Cart Page
- Checkout Page
- Login/Signup Page
- Forget Password Page
- My Account Page
- Customer Address Page
- Customer Order History Page
- Customer Wishlist Page
- Customer Saved Cards Page
- Customer Billing Agreements Page
- Customer Product Reviews Page
- Customer Newsletter Subscription Page

We can apply any theme for Magento Frontend. With the latest Magento setup **“Luma”** theme is applied. You can find default Magento themes in following directories.

- vendor/magento/theme-frontend-blank
- vendor/magento/theme-frontend-luma

If you want to create any custom theme, you can create your own theme inside **"app/design/frontend"** directory. We will go in depth into it later on.

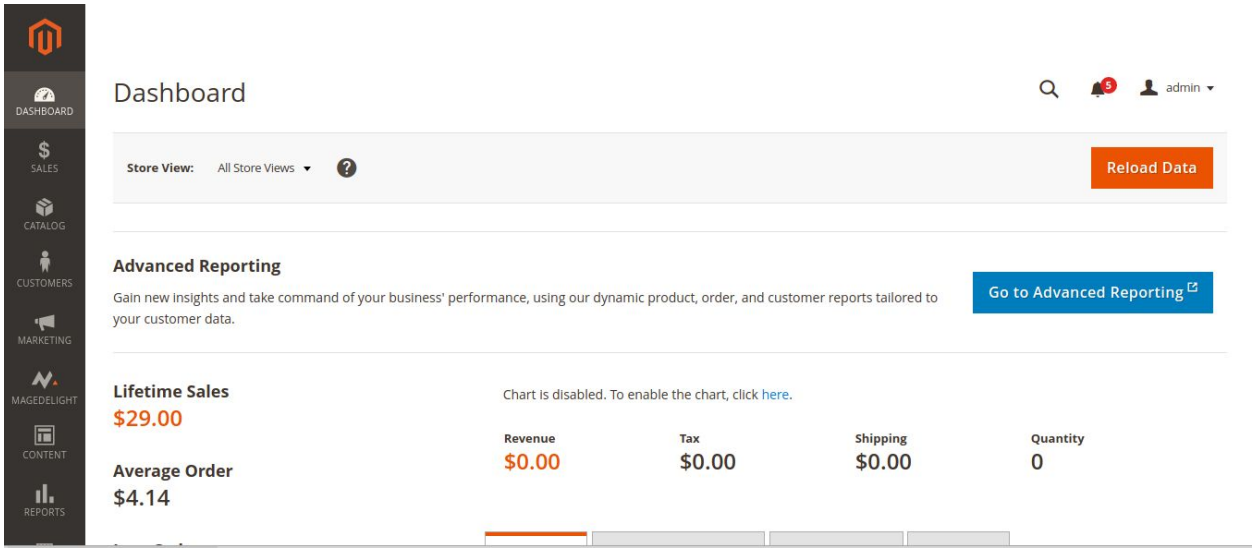
Reference Link:

https://devdocs.magento.com/guides/v2.3/frontend-dev-guide/themes/theme-create.html

Assignments:

1. Explore all the frontend pages mentioned above.
2. Add a product to Shopping Cart and checkout with guest customer
3. Add a product to Shopping Cart and checkout with register customer
4. Reorder and previously ordered item
5. Change Account information from customer panel
6. Change Password information from customer panel
7. Add items to wishlist
8. Edit any Product from Shopping Cart
9. See how out of stock product is can be shown from admin setting to frontend

Understanding of Magento Backend



From Magento Backend, you can manage the following things.

- Products
- Categories
- Customers
- Customer Groups
- Catalog Price Rules
- Shopping Cart Price Rules
- Orders
- Invoices
- Stores
- Websites
- Store Configuration
- Shipping Methods
- Payment Gateways
- CMS Pages
- CMS Blocks
- Widgets
- Reports
- Email Templates
- Cache Management
- Index Management
- Backups
- Custom Variables
- Admin Users
- Import/Export Data
- Web Setup Wizard
- Integrations
- Theme Configurations
- Scheduling Themes
- And many more things

Assignments:

1. Create a new Website, Store and Store View
2. Create a new Root Category, Level 1 Sub Category and Level 2 Sub Category
3. Create a new Product and assign it to multiple Categories
4. Create a CMS Page and add it to Footer
5. Create a CMS Block and assign it to Block inside Category Edit screen
6. Enable/Disable Payment methods from Store Configuration
7. Create a new Customer/Customer Group/Customer Address from Admin
8. Create Catalog and Cart Price Rules from Promotion Menu

9. Add new Currency and rate and reflect them in frontend
10. Create a product attribute with text and dropdown value and assign it to default attribute set.

Introduction to Elasticsearch

Elasticsearch is a search engine based on the Lucene library. It provides a distributed, multitenant-capable full-text search engine with an HTTP web interface and schema-free JSON documents. Elasticsearch is developed in Java.

Elasticsearch is standing as a *NOSQL DB* because:

- it easy-to-use
- Has a great community
- Complatibility with JSON
- Broad use cases

Reference Link:

https://www.elastic.co/what-is/elasticsearch

Introduction to Redis

What is REDIS?

Developed in 2009, REmote Dictionary Server (REDIS) is an open source, NoSql key value database. You can say it's a data structure server for developers to organize and use data efficiently and quickly. Redis allows the user to store vast amounts of data without the limitation of a relational database unlike MongoDB, MySQL, etc. It is written in ANSI C and runs on POSIX like your Macintosh.

Why use REDIS?

It is used for cache management and speeding up the web application by using a structured way to store data in the memory. It is, therefore, faster than conventional database techniques like MySQL, MongoDB, and Oracle.

Redis uses key value storage techniques, that is, every data structure is represented as a key. Redis has a number of keys to represent as many formats, resulting in more operations from the user perspective and reduced load from the client perspective.

Unlike MongoDb, which is a disk-based data storage, Redis holds all its database in memory, using disk only persistence technique, which stores the data in computer RAM, thereby making the processing extremely fast. It uses a memory caching technique which allows users to store data in a more durable and robust manner.

Redis supports the following Data structures. Regardless of their type, they are accessed by a key.

Reference Link:

https://redis.io/topics/introduction

https://codeburst.io/redis-what-and-why-d52b6829813

Introduction to Nginx / Apache

Apache and Nginx are the two most common open source web servers in the world. Together, they are responsible for serving over 50% of traffic on the internet. Both solutions are capable of handling diverse workloads and working with other software to provide a complete web stack.

While Apache and Nginx share many qualities, they should not be thought of as entirely interchangeable. Each excels in its own way and it is important to understand the situations where you may need to reevaluate your web server of choice. This article will be devoted to a discussion of how each server stacks up in various areas.

Apache

The Apache HTTP Server was created by Robert McCool in 1995 and has been developed under the direction of the Apache Software Foundation since 1999. Since the HTTP web server is the foundation's original project and is by far their most popular piece of software, it is often referred to simply as "Apache".

The Apache web server has been the most popular server on the internet since 1996. Because of this popularity, Apache benefits from great documentation and integrated support from other software projects.

Apache is often chosen by administrators for its flexibility, power, and widespread support. It is extensible through a dynamically loadable module system and can process a large number of interpreted languages without connecting out to separate software.

Nginx

In 2002, Igor Sysoev began work on Nginx as an answer to the C10K problem, which was a challenge for web servers to begin handling ten thousand concurrent connections as a requirement for the modern web. The initial public release was made in 2004, meeting this goal by relying on an asynchronous, events-driven architecture.

Nginx has grown in popularity since its release due to its light-weight resource utilization and its ability to scale easily on minimal hardware. Nginx excels at serving static content quickly and is designed to pass dynamic requests off to other software that is better suited for those purposes.

Nginx is often selected by administrators for its resource efficiency and responsiveness under load. Advocates welcome Nginx's focus on core web server and proxy features.

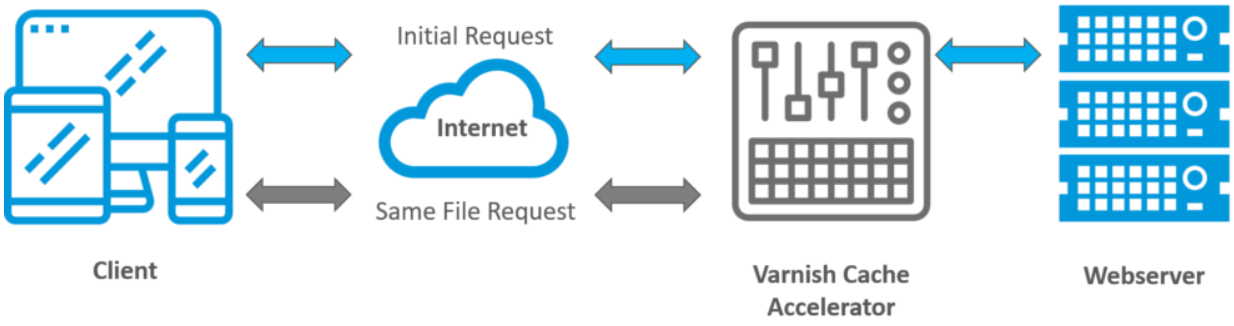
Apache	Nginx
Apache follows multi-threaded approach to process client requests.	Nginx uses an event-driven approach to serve client requests.
It handles dynamic content within the web server itself.	It cannot process dynamic content natively.
It cannot process multiple requests concurrently with heavy web traffic.	It can process multiple client requests concurrently and efficiently with limited hardware resources.
Modules are dynamically loaded or unloaded making it more flexible.	The modules cannot be loaded dynamically. They must be compiled within the core software itself.
Apache is designed to be a web server.	Nginx is both a web server and a proxy server.
A single thread can only process one connection.	A single thread can handle multiple connections.

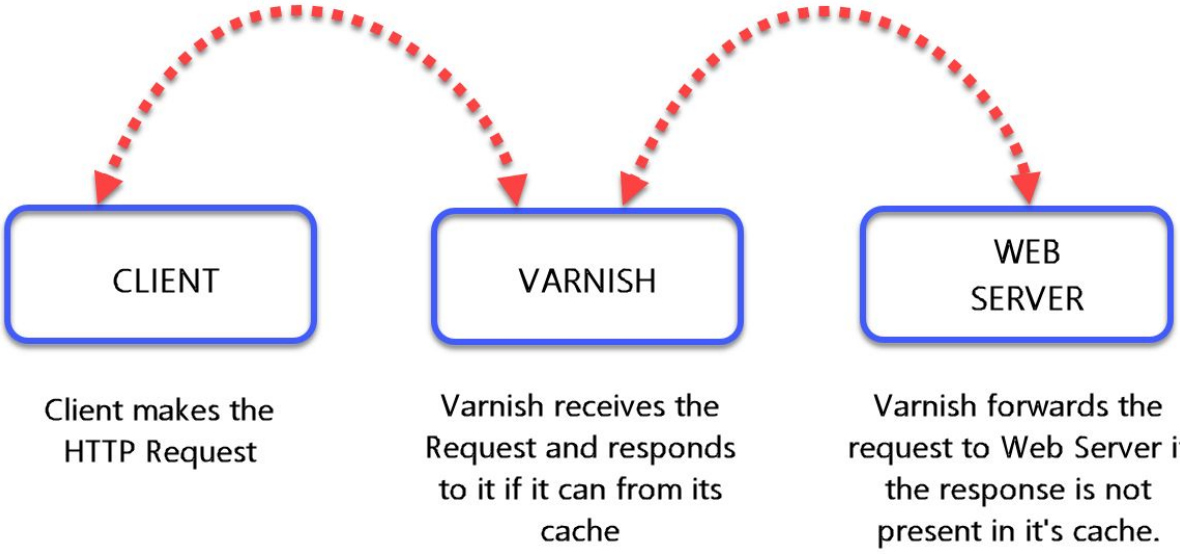
Reference Link:

https://www.digitalocean.com/community/tutorials/apache-vs-nginx-practical-considerations

Introduction to Varnish

Varnish is an HTTP accelerator designed for content-heavy dynamic web sites as well as APIs. In contrast to other web accelerators, such as Squid, which began life as a client-side cache, or Apache and nginx, which are primarily origin servers, Varnish was designed as an HTTP accelerator.





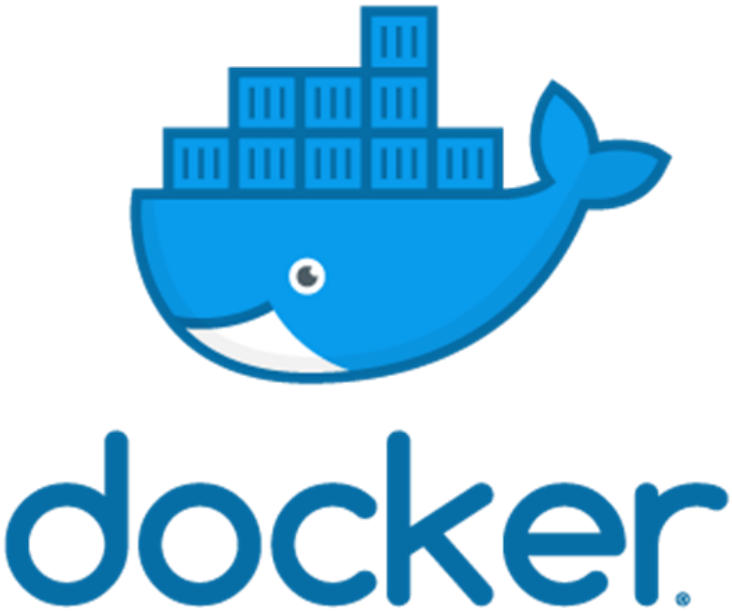
Reference Links:

https://en.wikipedia.org/wiki/Varnish_(software)

https://varnish-cache.org/

Introduction to Docker and Vagrant Box

What is Docker



Docker is a set of platform as a service products that use OS-level virtualization to deliver software in packages called containers. Containers are isolated from one another and bundle their own software, libraries and configuration files; they can communicate with each other through well-defined channels.

Docker is a tool designed to make it easier to create, deploy, and run applications by using containers. Containers allow a developer to package up an application with all of the parts it needs, such as libraries and other dependencies, and ship it all out as one package.

By doing so, thanks to the container, the developer can rest assured that the application will run on any other Linux machine regardless of any customized settings that machine might have that could differ from the machine used for writing and testing the code.

In a way, Docker is a bit like a virtual machine. But unlike a virtual machine, rather than creating a whole virtual operating system, Docker allows applications to use the same Linux kernel as the system that they're running on and only requires applications be shipped with things not already running on the host computer. This gives a significant performance boost and reduces the size of the application.

And importantly, Docker is open source. This means that anyone can contribute to Docker and extend it to meet their own needs if they need additional features that aren't available out of the box.

Reference Link

https://www.docker.com/why-docker

What is Vagrant Box

Vagrant Boxes are prepackaged development environments that are the foundation of Vagrant. In most cases, this is usually just a stripped and naked operating system such as Ubuntu, Debian, or CentOS.

It is same as docker.

A box can be used by anyone on any platform that Vagrant supports to bring up an identical working environment. ... The easiest way to use a box is to add a box from the publicly available catalog of Vagrant boxes.

Reference Link

https://www.vagrantup.com/docs/boxes.html

Magento Architecture

Magento 2 architecture is considered to be aLayered Structure where each and every concept or code has a fixed set of directories to follow so that it is easier to modify behaviour of any feature of magento with easy to implement steps.

At its highest level, Magento's product architecture consists of the core product code plus optional modules. These optional modules enhance or replace the basic product code.

If you are substantially customizing the basic Magento product, module development will be your central focus. Modules organize code that supports a particular task or feature. A module can include code to change the look-and-feel of your storefront as well as its fundamental behavior.

Your modules function with the core Magento product code, which is organized into layers. Understanding layered software pattern is essential for understanding basic Magento product organization.

Layered software is a popular, widely discussed principle in software development. Many resources exist for this topic, but consider consulting Pattern-Oriented Software Architecture for a general discussion.

Advantages of layered application design

Layered application design offers many advantages, but users of Magento will appreciate:

- Stringent separation of business logic from presentation logic simplifies the customization process. For example, you can alter your storefront appearance without affecting any of the backend business logic.
- Clear organization of code predictably points extension developers to code location.

Reference Link

https://devdocs.magento.com/guides/v2.3/architecture/archi_perspectives/ALayers_intro.html

Magento Module Based Architecture

A module is a logical group – that is, a directory containing blocks, controllers, helpers, models – that are related to a specific business feature. In keeping with Magento's commitment to optimal modularity, a module encapsulates one feature and has minimal dependencies on other modules.

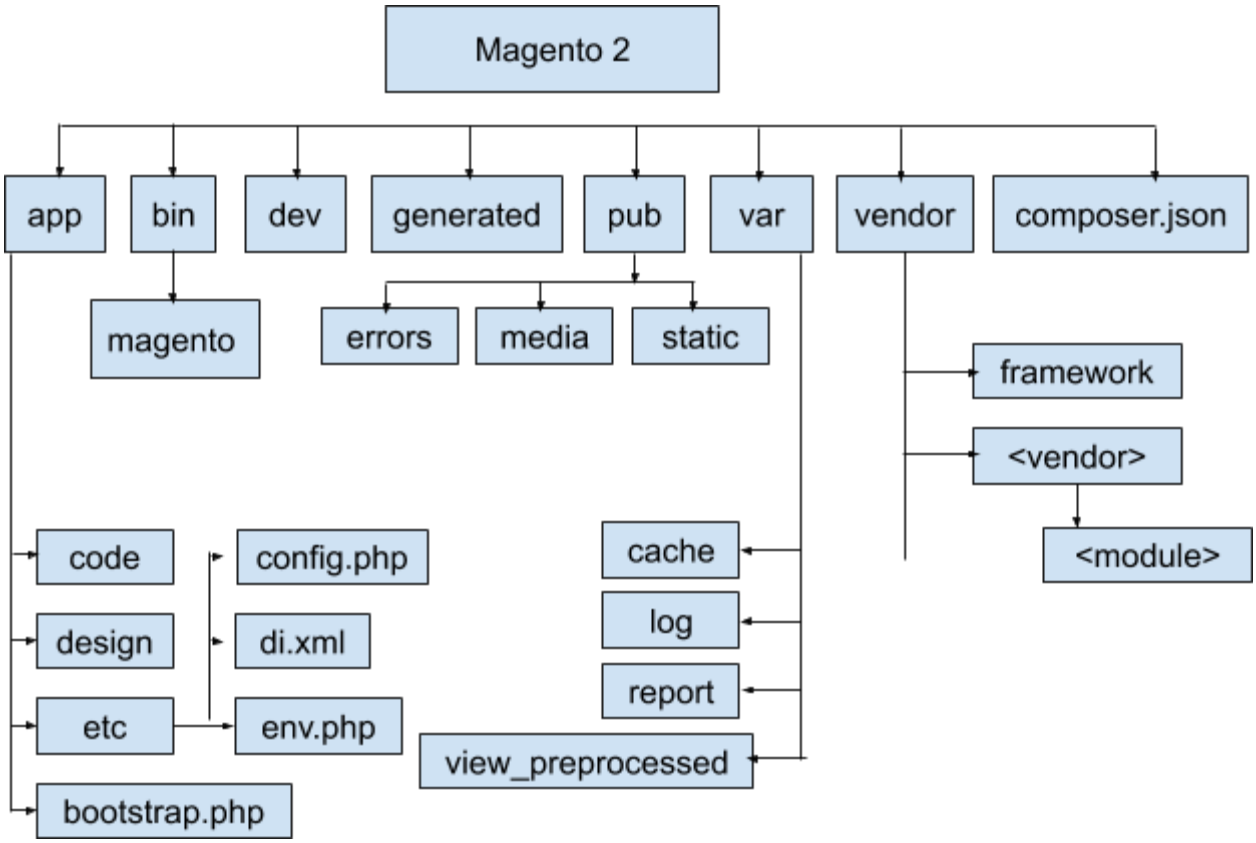
Modules and themes are the units of customization in Magento. Modules provide business features,

with supporting logic, while themes strongly influence user experience and storefront appearance. Both components have a life cycle that allows them to be installed, deleted, and disabled. From the perspective of both merchants and extension developers, modules are the central unit of Magento organization.

The Magento Framework provides a set of core logic: PHP code, libraries, and the basic functions that are inherited by the modules and other components.

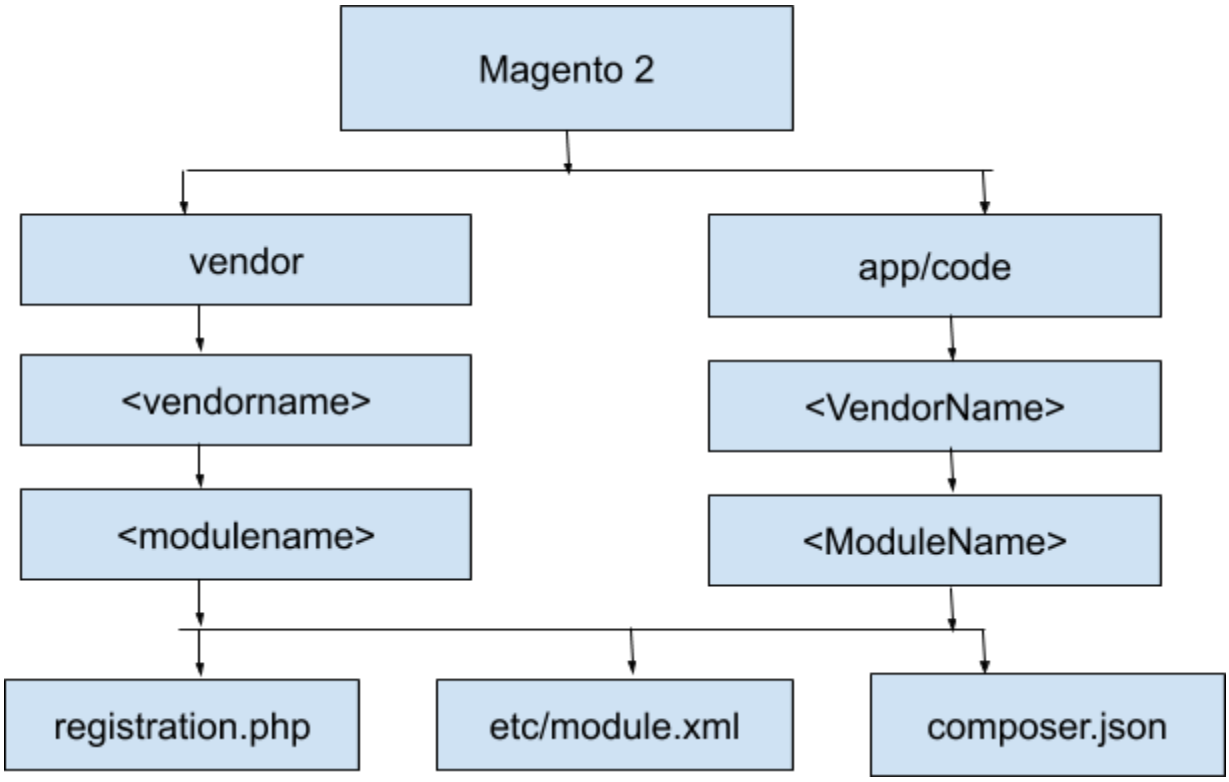
Magento Directory Structure

Some of the important things to know about Magento directory structure is as follows.



Magento Module Directory Structure

A standard magento module structure will have following directories and files.


Magento 2
vendor
app/code
<vendorname>
<VendorName>
<modulename>
<ModuleName>
registration.php
etc/module.xml
composer.json

```
app
    code
        Vendor
            Module
                Api
                    Data
                Block
                    Adminhtml
                Console
                    Command
                Controller
                    Adminhtml
                Cron
                CustomerData
                etc
                    adminhtml
                        di.xml
                        events.xml
                        menu.xml
                        routes.xml
                        system.xml
                    frontend
                    webapi_rest
                    webapi_soap
                    acl.xml
                    config.xml
                    crontab.xml
                    db_schema.xml
                    db_schema_whitelist.json
                    di.xml
                    events.xml
                    indexer.xml
                    module.xml
                Helper
```

```
i18n
    en_US.csv
Model
    ResourceModel
Observer
Plugin
Setup
    InstallSchema.php
    InstallData.php
    UpgradeSchema.php
    UpgradeData.php
    Recurring.php
Test
Ui
view
    adminhtml
    frontend
        layout
            *.xml
        templates
            *.phtml
        web
            css
                *.less
                *.css
            js
                template
                    *.html
            images
                *.jpg
                *.png
        requirejs-config.js
ViewModel
composer.json
registration.php
```

If you want to have a complete look, goto following directory

> vendor/magento/module-catalog/

Here is the explanation of all directories.

Api

This will contain PHP Interfaces related to soap and rest webservices

Api/Data

This will contain PHP Interfaces related to data disclosure of soap and rest webservices

Block

This will contain PHP files those are used to supply data to view files. This is **C**ontroller section of MVC architecture.

Console

All the CLI commands are defined in this directory

CustomerData

Dynamic Data processing is written here which is used to refresh after page load via ajax.

etc

This will contain all the configuration XML files used in a module

etc/adminhtml/menu.xml

This is used to define menu items to show in admin panel

etc/adminhtml/system.xml

This is used to define store configuration settings

etc/frontend/routes.xml

This is used to define routing for store front or adminhtml

etc/webapi_rest.xml

This is used to define settings for restful webapi

etc/webapi_soap.xml

This is used to define settings for soap webapi

etc/acl.xml

This is used to store Access Control List to be used in admin panel

etc/config.xml

This is used to deifine default configuration value for store configuration.

etc/crontab.xml

This is used to define cronjobs to be executed at regular intervals

etc/db_schema.xml
This is used to define db schema of module specific tables

etc/db_schema_whitelist.json
This is used to define db schema master of module specific tables

etc/di.xml
This is used to define dependency injections for the system

etc/events.xml

This is used to define observers for the system

etc/indexer.xml

This is used to define indexer information for the custom module

etc/module.xml

This is used to define module version and dependencies to load module sort order

Helper

This is used to place PHP helpers files

I18n

This is used to store language translation

Model

This is used to define files to access table records

Observer

This is used to code PHP files those are defined in events.xml

Plugin

This is used to define code interceptors for public methods

Setup

This is used to define table creation/updation at the time of installing or updating the module

Test

This is used to define test cases for the module

Ui

This is used to define Ui Components for the module

view

This is used to define layout, templates, and static contents like css, js, images and font files

ViewModel

This is used to define SOLID principals and replacement of a Block file

composer.json

This is used to define module information along with dependent library information that can be retrieved from packagist.com

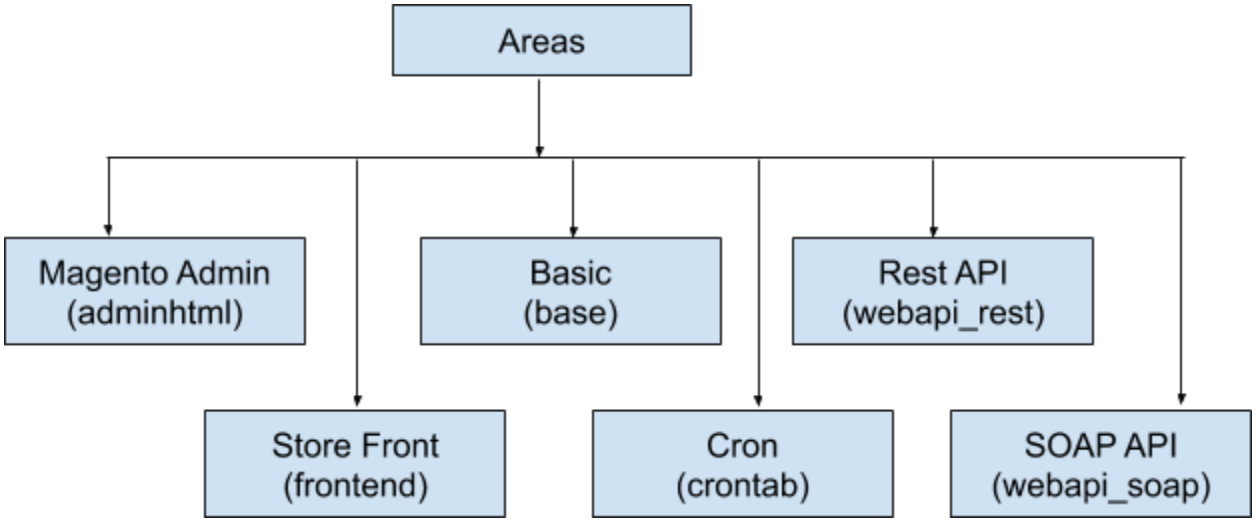
registration.php

This is used to initiate the module.

Different areas of Magento

An area is a logical component that organizes code for optimized request processing. Magento uses areas to streamline web service calls by loading only the dependent code for the specified area. Each of the default areas defined by Magento can contain completely different code on how to process URLs and requests.

For example, if you are invoking a REST web service call, rather than load all the code related to generating user HTML pages, you can specify a separate area that loads code whose scope is limited to answering REST calls.



Magento is organized into these main areas:

Magento Admin (adminhtml): entry point for this area is index.php or pub/index.php. The Admin panel area includes the code needed for store management. The /app/design/adminhtml directory contains all the code for components you'll see while working in the Admin panel.

Storefront (frontend): entry point for this area is index.php or pub/index.php. The storefront (or frontend) contains template and layout files that define the appearance of your storefront.

Basic (base): used as a fallback for files absent in adminhtml and frontend areas.

Cron (crontab): In pub/cron.php, the \Magento\Framework\App\Cron class always loads the 'crontab' area.

You can also send requests to Magento using the SOAP and REST APIs. These two areas

Web API REST (webapi_rest): entry point for this area is index.php or pub/index.php. The REST area has a front controller that understands how to do URL lookups for REST-based URLs.

Web API SOAP (webapi_soap): entry point for this area is index.php or pub/index.php.

Magento Configuration Files

> app/etc/env.php

This file contains settings that are specific to the installation environment.

- Admin Slug
- Encrypt key
- Database settings
- Resource connection settings (Shared database settings)
- Magento mode information
- Session settings (save in db or file system)
- Cache statuses

Have a look at the below image.

```php
<?php
return [
    'backend' => [
        'frontName' => 'admin'
    ],
    'crypt' => [
        'key' => '3167615d5355bad20b440242e90b4ab1'
    ],
    'db' => [
        'table_prefix' => '',
        'connection' => [
            'default' => [
                'host' => 'localhost',
                'dbname' => 'm2_232',
                'username' => 'root',
                'password' => 'root',
                'model' => 'mysql4',
                'engine' => 'innodb',
                'initStatements' => 'SET NAMES utf8;',
                'active' => '1'
            ]
        ]
    ],
    'resource' => [
        'default_setup' => [
            'connection' => 'default'
        ]
    ],
    'x-frame-options' => 'SAMEORIGIN',
    'MAGE_MODE' => 'developer',
    'session' => [
        'save' => 'files'
    ],
    'cache' => [
        'frontend' => [
            'default' => [
                'id_prefix' => 'b62_'
            ],
            'page_cache' => [
                'id_prefix' => 'b62_'
```

> app/etc/config.php

This file contains the list of installed modules, themes, and language packages.

```php
<?php
return [
    'modules' => [
        'Magento_Store' => 1,
        'Magento_AdvancedPricingImportExport' => 1,
        'Magento_Directory' => 1,
        'Magento_Amqp' => 1,
        'Magento_Config' => 1,
        'Magento_Theme' => 1,
        'Magento_Backend' => 1,
        'Magento_Variable' => 1,
        'Magento_Eav' => 1,
        'Magento_Search' => 1,
        'Magento_Backup' => 1,
        'Magento_Customer' => 1,
        'Magento_AdminNotification' => 1,
        'Magento_Authorization' => 1,
        'Magento_BundleImportExport' => 1,
        'Magento_Indexer' => 1,
        'Magento_CacheInvalidate' => 1,
        'Magento_Cms' => 1,
        'Magento_Catalog' => 1,
        'Magento_Security' => 1,
        'Magento_GraphQl' => 1,
        'Magento_CatalogImportExport' => 1,
        'Magento_Quote' => 1,
        'Magento_CatalogInventory' => 1,
        'Magento_Rule' => 1,
        'Magento_Msrp' => 1,
        'Magento_CatalogRule' => 1,
        'Magento_Bundle' => 1,
        'Magento_SalesSequence' => 1,
        'Magento_CatalogUrlRewrite' => 1,
        'Magento_StoreGraphQl' => 1,
        'Magento_Widget' => 1,
```

Reference Link

https://devdocs.magento.com/guides/v2.3/config-guide/config/config-magento.html

Utilize configuration and configuration variables scope

Magento uses a lot of configuration files to divide the functionality into chunks.

Some of them are as follows.

etc/config.xml — contains default option values from Stores > Configuration in the admin panel menu. This menu can be configured at system.xml;

di.xml — contains configurations for the dependency injection;

etc/events.xml — a list of observers and events;

etc/routes.xml — a list of routers;

etc/config.xml — contains the default values for the module settings Stores > Configuration;

etc/acl.xml — adds module resources to a resource tree that allows you to configure access for different users.

etc/crontab.xml — adds and configures the task for the cronjob;

etc/module.xml — announces the name and the version of the module, as well as its dependencies on other modules;

etc/widget.xml — stores the widget settings;

etc/indexer.xml — announces a new kind of indexing. It specifies the view_id parameter, which points at the views described in etc/mview.xml;

etc/mview.xml — describes the representations of all the indices described in etc/indexer.xml;

etc/webapi.xml — defines web API components, which service method to use and which resource to connect for a specific request;

etc/view.xml — contains the properties of product images;

etc/product_types.xml — describes types of products in a store;

etc/product_options.xml — describes the types of options, that can have products and classes to render them;

etc/extension_attributes.xml — a new ability to add a custom attribute appeared in Magento 2. This file describes the attribute, its type, which can be simple or complex and represent an interface;

etc/catalog_attributes.xml — groups attributes;

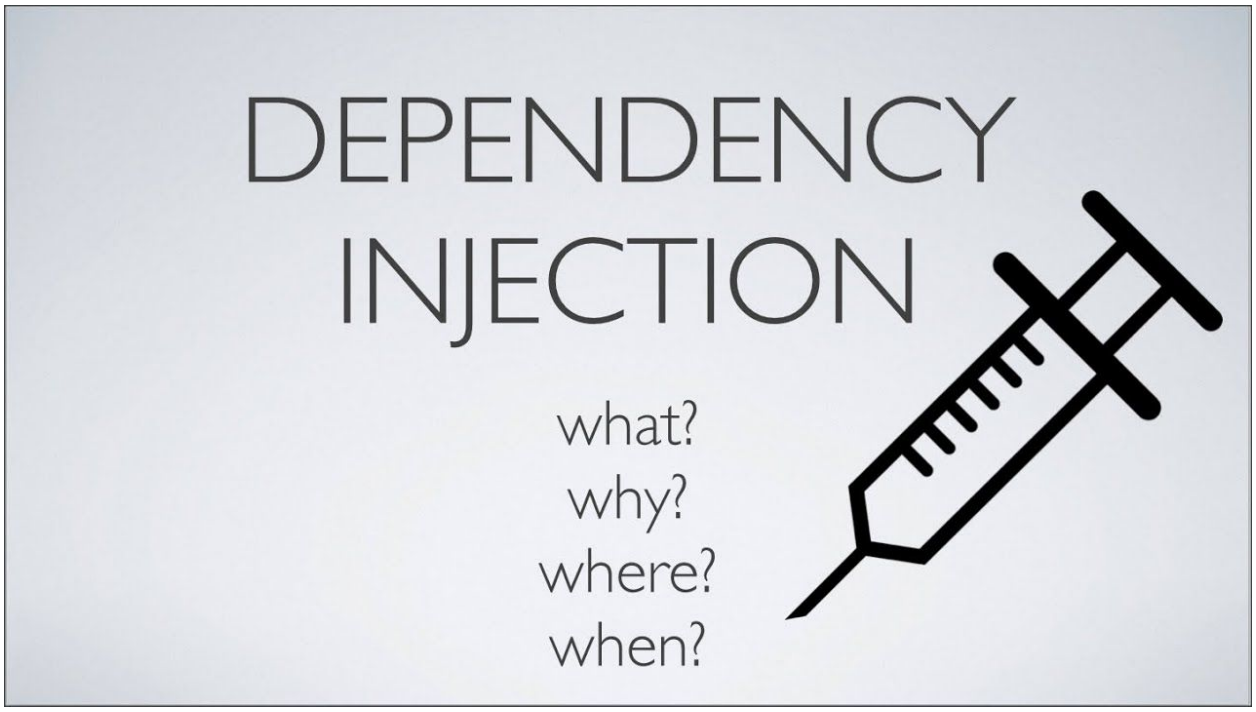
etc/adminhtml/system.xml — can only apply to the admin area, adds tabs to Stores > Configuration, describes sections and fields of a form;

etc/adminhtml/menu.xml — can only apply to the admin area, adds an item to the admin panel menu.

Reference Link

https://belvg.com/blog/configuration-files-and-variables-scope-in-magento-2.html

Demonstrate how to use dependency injection (DI)



What ?

Dependency Injection is a concept to modify the behaviour of a function or a class, without changing them directly.

Why ?

Before going in depth into the code section, we must know what is namespace and use statement in PHP.

A namespace is an identification of any file while use is being used to provide alias to any class.

Sometimes it is needed to modify or extend the functionality of a particular event or a function to fulfil the requirements. Those requirements cannot be static and can change on client to client bases.

For example, consider your client is using an ERP system to manage inventory, order management things. Magento is a selling tool for him. So whatever products he import to ERP system should be added in magento directly and orders placed in Magento should reflect in ERP too. In this type of scenario we can use dependency injection concept. Where our code can be added in certain events like whenever we place an order, our code should be executed. This type addon, we cannot add in default order processing system. It is something out of box functionality for Magento.

When ?

Dependency Injection can be of 5 types.

1. **Type**
   - Type is used to change/add arguments of a constructor of any class.
   Consider and example, you have a class with following definition.

```php
<?php

class A
{
    protected $name;

    public function __construct(
        string $name = 'unknown'
    ) {
        $this->name = $name;
    }

    public function getName()
    {
        return $this->name;
    }
}

$a = new A();
echo $a->getName();

Output:
unknown
```

   Now you want to give default value to the constructor argument, you should define below code in di.xml file

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

<type name="A">
    <arguments>
        <argument name="name" xsi:type="string">MyName</argument>
    </arguments>
</type>
```

Now the code will return you following output.

```php
<?php

class A
{
    protected $name;

    public function __construct(
        string $name = 'unknown'
    ) {
        $this->name = $name;
    }

    public function getName()
    {
        return $this->name;
    }
}

$a = new A();
echo $a->getName();

Output:
MyName
```

That is the power of “type” type of di.xml

2. **Virtual Type**

A virtual type is nothing but a virtual class that does not exists. This is some class defined in XML instead of PHP. Sometimes to shorten the line of codes, we need to create a class with arguments passed in constructor based on from where the class is being called.

Sometimes virtual type is used also to shorten class name.
For example, consider a class is having a following name.

MyVendor\MyModule\Model\ResourceModel\ThisClassIsCreatedWithALongName

Now consider you have to create object of this class in mulitple files. Writing this long class name everywhere is difficult. So in this case, you can create a virtual type to rename it to a short name. Consider following.

```
<virtualType name="ShortName" type="
MyVendor\MyModule\Model\ResourceModel\ThisClassIsCreatedWithALongName">
</virtualType>

<?php

$a = new MyVendor\MyModule\Model\ResourceModel\ThisClassIsCreatedWithALongName()

$a = new ShortName();
```

The both above will be having the same output.

You can also change the constructor argument like the below way.

```
<virtualType name="NameA" type="
MyVendor\MyModule\Model\ResourceModel\ThisClassIsCreatedWithALongName">
    <arguments>
        <argument name="name" xsi:type="string">Name 1</argument>
    </arguments>
</virtualType>

<virtualType name="NameB" type="
MyVendor\MyModule\Model\ResourceModel\ThisClassIsCreatedWithALongName">
    <arguments>
        <argument name="name" xsi:type="string">Name 2</argument>
    </arguments>
</virtualType>
```

```
<?php

namespace MyVendor\MyModule\Model\ResourceModel;

class ThisClassIsCreatedWithALongName
{
    public function __construct(
        $name = 'unknown'
    ) {
        $this->name = $name;
    }

    public function getName()
    {
        return $this->name;
    }
}
$a = new NameA();
$b = new NameB();

echo $a->getName(); // Name 1
echo $b->getName(); // Name 2
```

3. Event Observers

Event Observers are the feature to execute additional functionality on certain events. Some of the events are

- **a.** sales_order_place_before
- **b.** sales_order_place_after
- **c.** catelog_product_save_before
- **d.** Catalog_product_save_after

Consider an example of a requirement. Your client wants to send an email when a new order is placed to the system. In that case, you need to create events.xml to define your observer. Just like below.

```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_order_place_after">
        <observer name="my_custom_event" instance="
        MyVendor\MyModule\Observer\SendCustomEmail" />
    </event>
</config>
```

```php
<?php

namespace MyVendor\MyModule\Observer;

use Magento\Framework\Event\ObserverInterface;

class SendCustomEmail implements ObserverInterface
{
    public function execute(\Magentp\Framework\Event\Observer $observer)
    {
        // do your custom functionality here
    }
}
```

4. **Plugins**

   Plugins are something by the use of which you can modify the behaviour of a function instead of making changes inside it.

   Plugins are of 3 types.

   Before, After and Around

   Before is used to modify the function argument.
   After is used to modify the function return value.
   Around is used to modify the function entire behaviour

   Plugin is defined in the following way.

```
<?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\Catalog\Model\Product">
        <plugin name="my_product_plugin" type="MyVendor\MyModule\Plugin\ProductPlugin"
        sortOrder="10" />
    </type>
</config>
```

Reference Link
https://devdocs.magento.com/guides/v2.3/extension-dev-guide/plugins.html

5. **Code Override**

   Sometimes it is needed to override some code. Using plugins we will only be able to modify public methods, not private methods.

   So to override some private function we may need to override that class. It is defined the following way.

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

<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation
="urn:magento:framework:ObjectManager/etc/config.xsd">
    <preference for="OriginalClass" type="OverriddenClass" />
</config>
```

   Now you can create/override methods of original class into overridden class.

Sample Module

https://github.com/yash7690/magento2-di-samplemodule

Assignments

1. Create a plugin to change product price to add 2$ to each product
2. Create an event observer to log product name whenever a product is saved

Some more thing you should be aware about. That is **Scheduled Jobs** better known as **Cron Jobs**

Cron Jobs is executed every minute depending upon how frequently you want it to run. It is defined in the following file.

> etc/crontab.xml

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

<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation
="urn:magento:module:Magento_Cron:etc/crontab.xsd">
    <group id="default">
        <job name="my_custom_cronjob" instance="MyVendor\MyModule\Cron\MyCronJob" method="
        execute">
            <schedule>* * * * *</schedule>
        </job>
    </group>
</config>
```

Cronjob runs based on group. A group is a separate process for a cronjob. Default Magento is having following cron groups.

- Default
- Index

Reference Link

https://devdocs.magento.com/guides/v2.3/config-guide/cron/custom-cron-ref.html

As shown in the above image, a cron job has the following parameters.

Name: Identity of your cron job

Instance: Which file to call when cronjob executes

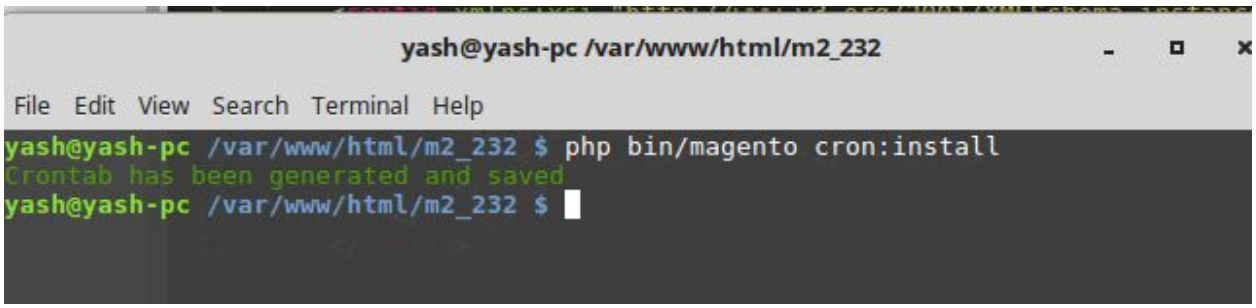
Method: Name of the function of the file to be executed

Schedule: Cron Schedule to be executed

For more information on magento cronjobs, visit below link.

https://devdocs.magento.com/guides/v2.3/cloud/configure/setup-cron-jobs.html

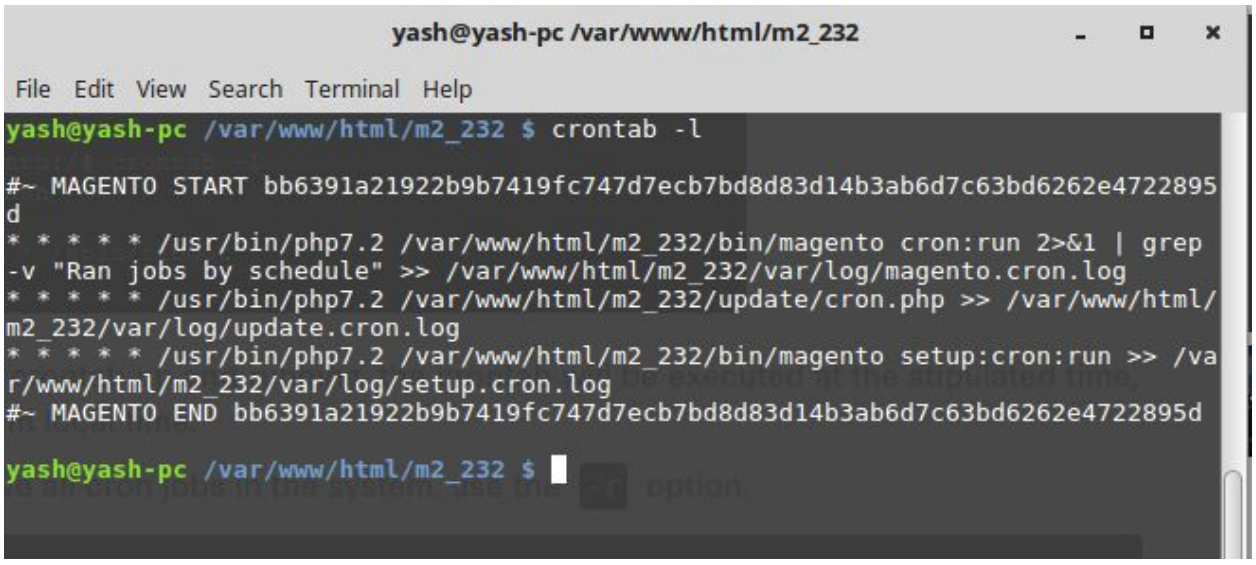
To configure a cron job for your magento instance into the system. You must define it via CLI using following command.



After setting up the cron into the system, you can verify it by following way.

> crontab -l

This will give you the following output.

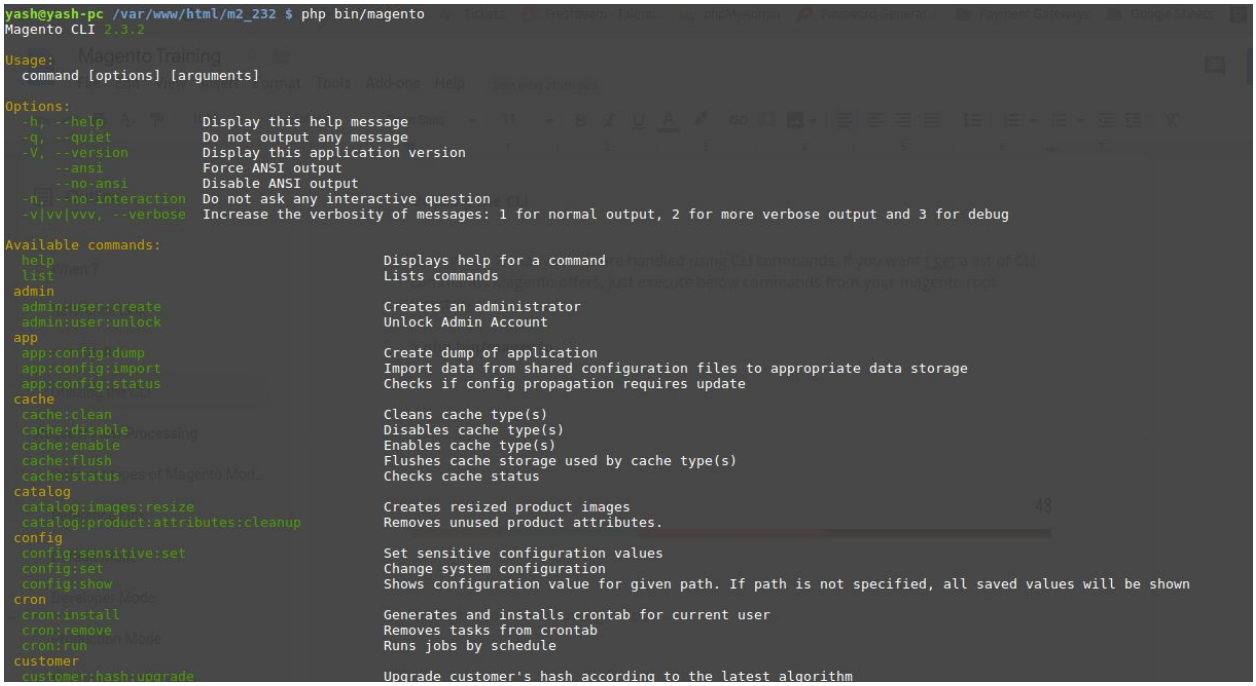


Utilizing the CLI

In Magento 2, many things are handled using CLI commands. If you want t get a list of CLI commands Magento offers, Just execute below commands from your magento root directory.

> php bin/magento

It will give you the following output.



Some of the important CLI commands you should be aware about.

php bin/magento setup:upgrade	This command is used whenver you want to install a new module or upgrade version of an existing module.
php bin/magento setup:di:compile	Used to compile PHP files for plugins, preferences and other types of elements configured in di.xml file

php bin/magento setup:static-content:deploy -f	Used to generate static content deployment. -f option is used for force deployment. This is needed in developer mode.
php bin/magento cache:flush	Used to purge all magento + non magento caches. NOn Magento referes to external caches like varnish.
php bin/magento cache:clean	Used to purge all magento specific caches only.
php bin/magento cache:enable	To enable all types of caches
php bin/magento cache:disable	To disable all types of caches
php bin/magento indexer:reindex	Used to reindex the data
php bin/magento sampledata:deploy	Used to deploy sampedata into Magento system
php bin/magento module:status <module_name>	To find out if a module is enabled or disabled
php bin/magento module:enable <module_name>	To enable a module
php bin/magento module:disable <module_name>	To disable a module
php bin/magento maintenance:enable	To enable maintenance mode
php bin/magento maintenance:disable	To disable maintenance mode
php bin/magento mainteance:allow-ips	Add ip address for maintenance mode
php bin/magento deploy:mode:show	To show current deployment mode. default, developer or production

php bin/magento deploy:mode:set <mode>	To set deployment mode to be one of the default, developer or production
php bin/magento cron:install	To install a cronjob
php bin/magento cron:run	To run a cronjob
php bin/magento setup:remove	To remove a cronjob

Assignments

1. Try with all of above CLI commands and check output yourself.

Describe how extensions are installed and configured

Whenever a new extension is installed, to make it working with the Magento Instance, you must execute following commands in the same order.

> php bin/magento setup:upgrade

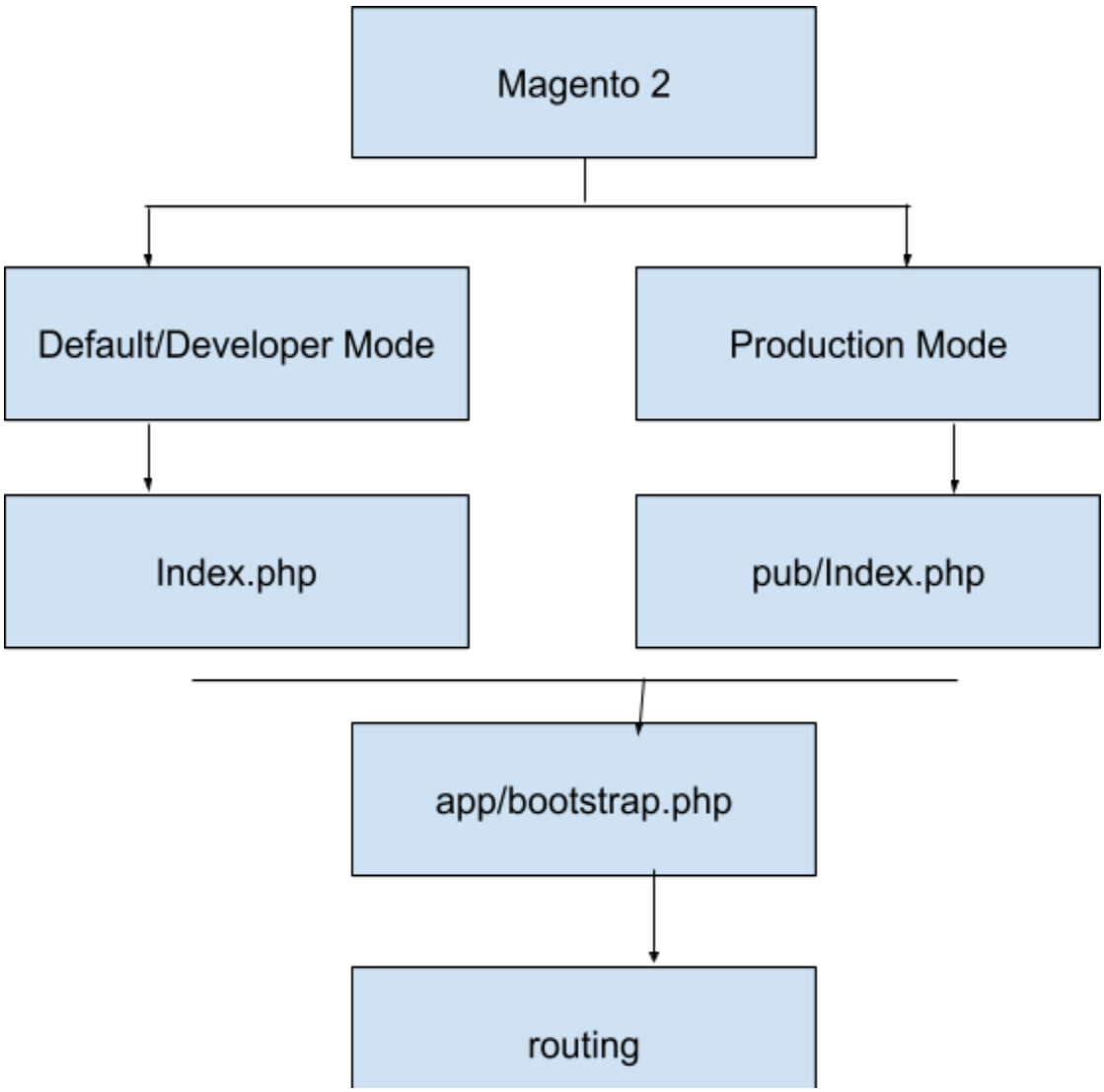
> php bin/magento setup:di:compile

> php bin/magento setup:static-content:deploy -f

After these commands, your new extension will be effective in magento. Once the extension is installed, it must provide some section of system config which you can explore.

Request Flow Processing

Magento Request Flow



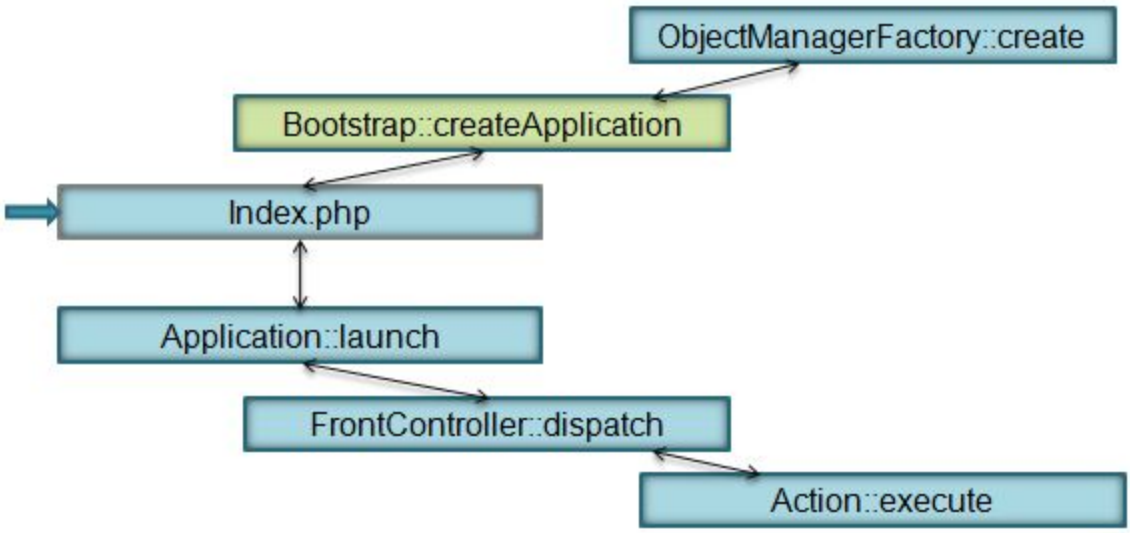
As shown in the above diagram, Magento first calls one fo the following files based on current deployment mode.

- Index.php (Default and Developer Mode)

- pub/Index.php (Production Mode)

This file creates an object of **app/bootstrap.php,** this file further launches the application and look for appropriate routing and dispatches the controller action to provide the output.

Kindly note that all the controller will have an execute method to further process the output.



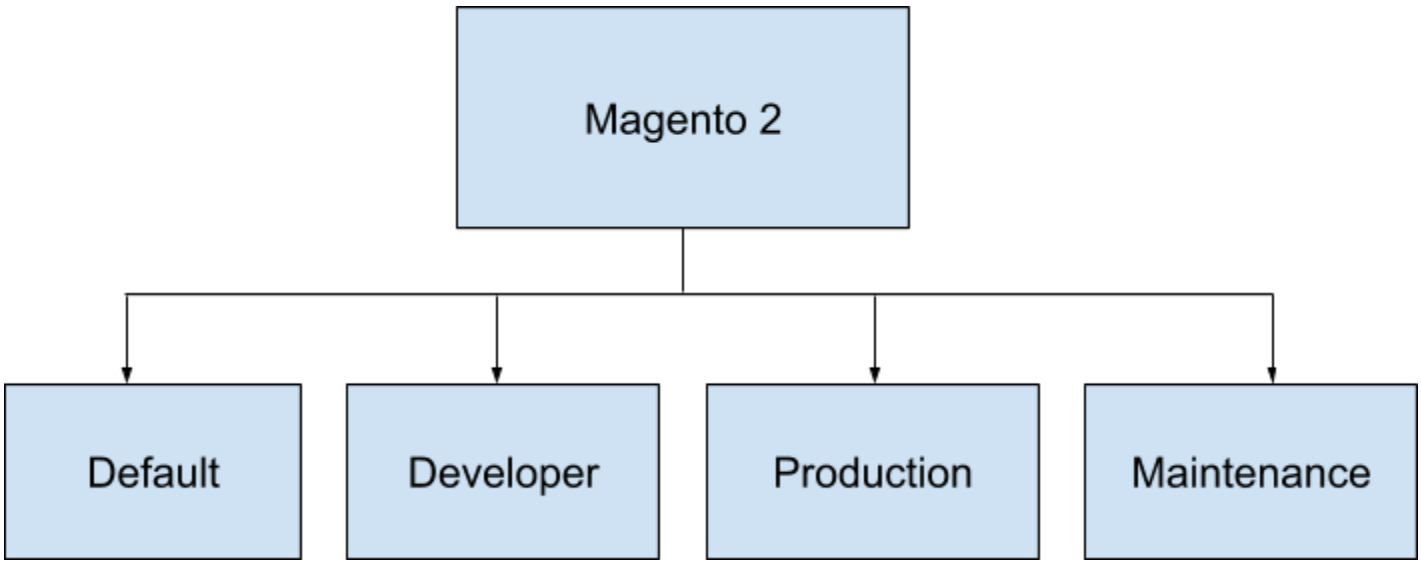

```
$bootstrap = \Magento\Framework\App\Bootstrap::create(BP, $_SERVER);
/** @var \Magento\Framework\App\Http $app */
$app = $bootstrap->createApplication('Magento\Framework\App\Http');
$bootstrap->run($app);
```

Reference Link

https://www.dckap.com/blog/request-flow-in-magento-2/

Different types of Magento Modes

Magento supports 3 types of application modes as shown below.



Reference Link:

https://devdocs.magento.com/guides/v2.3/config-guide/bootstrap/magento-modes.html

As shown above, it provides 3 different modes.

1. Default Mode
2. Developer Mode
3. Production Mode
4. Maintenance Mode

All modes have their own purpose, usage and limitations.

Default Mode

When you install a fresh Magento, it will be having default mode. Enables you to deploy the Magento application on a single server without changing any settings. However, default mode is not optimized for production.

In this mode,

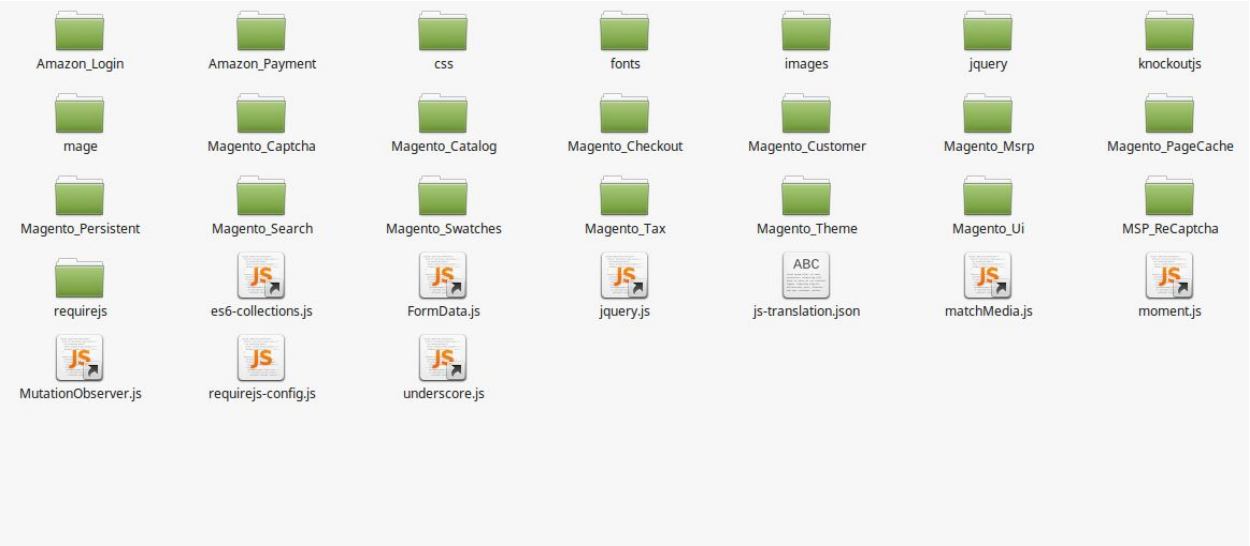
- Static view file caching is enabled

- Exceptions are not displayed to the user; instead, exceptions are written to log files.
- Hides custom X-Magento-* HTTP request and response headers

Developer Mode

Intended for development only, this mode:

- Disables static view file caching



- Provides verbose logging
- Enables automatic code compilation
- Enables enhanced debugging
- Shows custom X-Magento-* HTTP request and response headers
- Results in the slowest performance
- Shows errors on the frontend

Production Mode

Intended for deployment on a production system, this mode:

- Does not show exceptions to the user (exceptions are written to logs only).
- Serves static view files from cache only.

- Prevents automatic code file compilation. New or updated files are not written to the file system.
- Does not allow you to enable or disable cache types in Magento Admin.

You can check and change mode anytime using following CLI commands.

> php bin/magento deploy:mode:show (To check current mode)

> php bin/magento deploy:mode:set developer (To set developer mode)

> php bin/magento deploy:mode:set production (To set production mode)

Maintenance Mode

Run Magento in maintenance mode to take your site offline while you complete maintenance, upgrade, or configuration tasks. In maintenance mode, the site redirects visitors to a default Service Temporarily Unavailable page.

You can create a custom maintenance page, manually enable and disable maintenance mode, and configure maintenance mode to allow visitors from authorized IP addresses to view the store normally.

Reference Link:

https://devdocs.magento.com/guides/v2.3/comp-mgr/trouble/cman/maint-mode.html#compman-trouble-maint-create

Maintenance mode can be enable/disable via following CLI command.

```
php bin/magento maintenance:status
php bin/magento maintenance:enable
php bin/magento maintenance:disable
php bin/magento maintenance:enable --ip=192.168.0.1 --ip=192.168.0.2
```

Whenever you enable maintenance mode with or without specific ip addresses, it will create the following file.

> var/.maintenance.ip

This file will have all the ip addresses that has been restricted to view the site as a part of maintenance mode.

Different Between Developer and Production Mode

Developer Mode	Production Mode
Disables static view file caching (symbolic link to static view files)	Cache static view files (physical files instead of symbolic link)
Shows errors on the frontend	Does not show exceptions to the user (exceptions are written to logs only).
Shows custom X-Magento-* HTTP request and response headers (Varnish)	Hides custom X-Magento-* HTTP request and response headers (Varnish)
Allow you to enable or disable cache types in Magento Admin	Does not allow you to enable or disable cache types in Magento Admin.
Enables Developer menu in store configuration	Disables Developer menu in store configuration
Automatic code compilation	Prevents automatic code compilation

Routers in Magento

Routers can be of 3 types in Magento.

1. Static Routing
2. Dynamic Routing

3. Database Driven Routing

In general, there are 2 areas where we can define routing. Either it can be for admin area or it can be for frontend.

Please refer the following sample code to understand all types of routing.

https://github.com/yash7690/magento2-routing-samplemodule

If we are creating routing for admin, we have to define it in the following way.

> etc/adminhtml/routes.xml

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

<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="
urn:magento:framework:Module/etc/module.xsd">
    <router id="admin">
        <route id="adminsamplemodule" frontName="adminsamplemodule">
            <module name="MyVendor_AdminSampleModule" />
        </route>
    </router>
</config>
```

I will explain the terms in **Static Routing** section.

If we are creating routing for frontend, we have to define it in the following way.

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

<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="
urn:magento:framework:Module/etc/module.xsd">
    <router id="standard">
        <route id="adminsamplemodule" frontName="adminsamplemodule">
            <module name="MyVendor_AdminSampleModule" />
        </route>
    </router>
</config>
```

Here, the route id would be standard

Static Routing

Route that can be fixed slug can be considered to be static routing.

For example, if you want to show http://your-domain.com/generic page,

Where generic is fixed and cannot be changed, you can define it in the following way.

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

<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="
urn:magento:framework:Module/etc/module.xsd">
    <router id="standard">
        <route id="generic_route" frontName="generic">
            <module name="MyVendor_AdminSampleModule" />
        </route>
    </router>
</config>
```

Here router id=standard stands for frontend route.

route id=”generic_route” stands for layout xml file name should start with generic_route_<controller>_<action>.xml

route frontName=”generic” stands for URI parameter that the browser is supposed to have.

So let’s say your URL is http://your-domain.com/generic/index/index

It would be **generic_route_index_index.xml,** the layout file name should have id parameter, and the URL should have frontName parameter.

The purpose here is, layout file name should only contain alphanumeric characters, - (dash) or special characters are not allowed. So you can have different id and frontName attributes.

Dynamic Routing

If you are developing some module, where let's assume you created a page which shows some special product collection. You wan admin to define its URI slug dynamically in admin store configuration. Thats where this kind of routing comes into picture.

To achieve this, you have to add the following bunch of code in the following file.

> etc/frontend/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\App\RouterList">
        <arguments>
            <argument name="routerList" xsi:type="array">
                <item name="myvendor_mymodule_custom_routing" xsi:type="array">
                    <item name="class" xsi:type="string">
                    MyVendor\MyModule\Controller\Router\CustomRoute</item>
                    <item name="disable" xsi:type="boolean">false</item>
                    <item name="sortOrder" xsi:type="string">20</item>
                </item>
            </argument>
        </arguments>
    </type>
</config>
```

Here, you need to add your custom router to the router list.

> Controller/Router/CustomRoute.php

```php
<?php
namespace MyVendor\MyModule\Controller\Router;
class CustomRouter implements \Magento\Framework\App\RouterInterface
{
    protected $actionFactory;
    protected $_response;

    public function __construct(
        \Magento\Framework\App\ActionFactory $actionFactory,
        \Magento\Framework\App\ResponseInterface $response
    ) {
        $this->actionFactory = $actionFactory;
        $this->_response = $response;
    }

    public function match(\Magento\Framework\App\RequestInterface $request)
    {
        $identifier = $request->getOriginalPathInfo();
        $condition = new \Magento\Framework\DataObject(['identifier' => $identifier, 'continue' =>
            true]);
        $identifier = $condition->getIdentifier();

        if ($condition->getRedirectUrl()) {
            $this->_response->setRedirect($condition->getRedirectUrl());
            $request->setDispatched(true);
            return $this->actionFactory->create('Magento\Framework\App\Action\Redirect');
        }

        if (!$condition->getContinue()) {
            return null;
        }

        $sitemap_url = 'custom-router';
        if($identifier == $sitemap_url) {
            $request->setModuleName('<>route_id')->setControllerName('<controller>')->
                setActionName('index');
            $request->setAlias(\Magento\Framework\Url::REWRITE_REQUEST_PATH_ALIAS, $identifier);
            return $this->actionFactory->create('Magento\Framework\App\Action\Forward');
        }

        return null;
```

So let's say someone hits http://your-domain.com/custom-router

It will hit MyVendor/MyModule/Controller/<controller>/<action>.php file.

Will look more into it in practicals.

Database Driven Routing

Product URLs, Category URLs and CMS Pages URls are the best example of it.

A product with the URL Key: joust-duffle-bag.html can be open using following ways.

http://your-domain.com/catalog/product/view/id/1

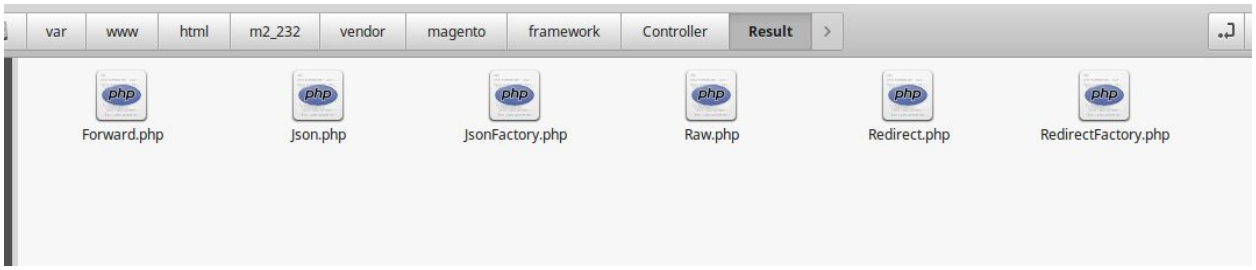
http://your-domain.com/joust-duffle-bag.html

This is the best example of Database Driven Routing.

Assignments

1. Create a module with following details.
   Vendor Name: MyVendor
   Module Name: RoutingExample
   Create a frontend route with id="routing_example" and frontName="routing-example" and add a block in layout.xml to render a simple "HI" into it.
2. Create a store configuration to take routing from user on store view bases and apply the same above thing that should work with admin entered routing URL

Demonstrate the ability to create a frontend controller with different response types (HTML/JSON/redirect)



Have a look at following directory.

> vendor/magento/framework/Controller/Result

A Controller can respond from following list of types.

1. Forward
2. Json
3. Raw
4. Redirect

Forward

If you do not want to change the URL but call a different controller, you should use this type of response

JSON

If you want the response to be json formatted, you should use this type of response

Raw

If you want to return simple html string, you can use this type of response

Sample Module:

https://github.com/yash7690/magento2-frontendresponse-samplemodule

Above example contains all types of response types.

Assignments

1. Create a module with all 4 types of responses.

How to use URL rewrites for a catalog product view to a different URL

URL rewrite is an important feature in Magento 2. It is a very easy interface to achieve the functionality.

First the question is why it is needed.

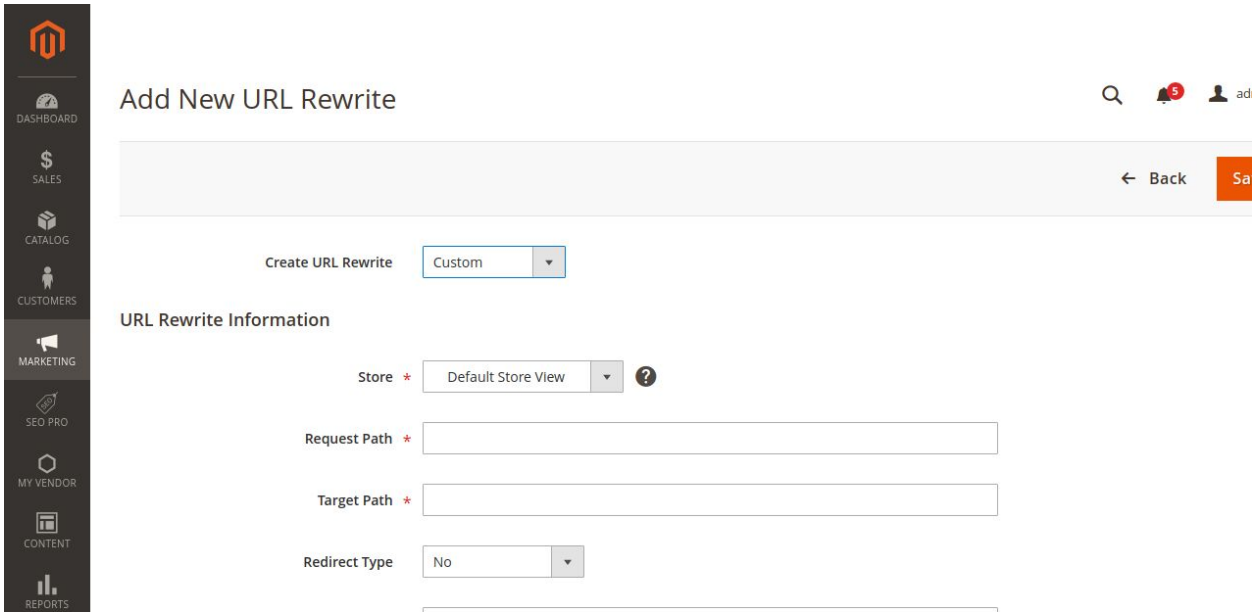
- SEO purposes user friendly URL
- Allow multiple URLs for the same product
- Consider the case, you deleted a product to system, but its URL is still present on some blogs. So instead of showing 404 page, you can rewrite the URL to some CMS page showing this product is deleted.

We can explore this feature by login to admin panel. Goto

> Marketing -> SEO & Search -> URL Rewrites

Here you can see default URL rewrites as well as create your custom one for

- Category
- Product
- CMS Page
- Custom



Store

You can choose your store for which this redirection is going to be set for

Request Path

Which path, url is having to match the redirection

Target Path

Which path, should be called on entered Request Path

Redirect Type

- **No**
  You want to use Forward action and do not want to change URl
- **Temporary (302)**
  It will be redirected to target path with status code of 302

- **Permanent (301)**
  It will be redirected to target path with status code of 301

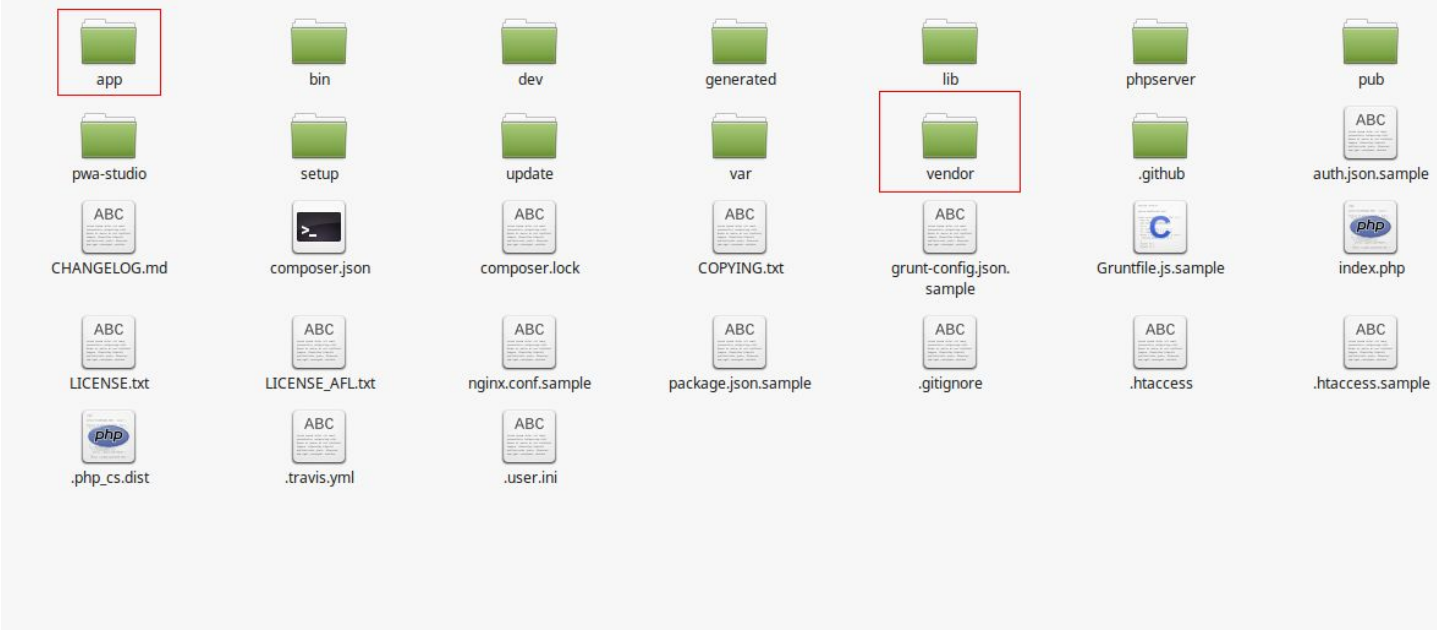
Assignments

1. Create a URL rewrite entry for a custom URL keeping request path to be a product URL. Create a CMS page that is having content of Product is deleted. Set the Target Path to be that CMS page URL. Set Redirect Type to be 301 and check the behaviour.

Magento Module Creation

Locations in Magento to create module

In Magento, we can find modules at following 2 places.



- **vendor**

"Vendor" directory is considered to be untouchable directory. In this directory, we will find majorly 2 types of modules.

- Magento default modules
- Modules those will be installed via composer or Magento Marketplace

You will see a list of modules inside **"vendor/magento"** directory.

For eg

- module-catalog
- module-cms
- module-sales
- Module-checkout

Here in **"vendor/magento"**, magento is the name of the module provider. However with default magento installation, you will see some more vendors like below.

- vendor/amzn/amazon-pay-and-login-with-amazon-core-module

- vendor/amzn/amazon-pay-module
- vendor/dotmailer/dotmailer-magento2-extension

- **app/code**

Modules, those are a part of custom requirement is developed inside this directory. As this is placed inside “app/code” directory, it is considered to be customized at any moment, though it is not advisable.

However in Magento cloud environment, they allow to change code of modules in this directory, but not inside vendor directory.

Why Vendor and Module name is needed.

In Magento, if you are developing a module, You need to consider 2 factors or names.

- Vendor Name (Company Name)
- Module Name (Extension Name)

Why Magento is following this structure is because a vendor or a company can offer more than one extension. By using this, they can group all their modules in one single vendor directory.

For example, Lets say your company name is “MyVendor”, and you are offering multiple extensions, you can have the following directory structure.

As you can see above image, if you are offering 4 different extensions, you can club them in a single vendor directory, **“MyVendor”** in this case.

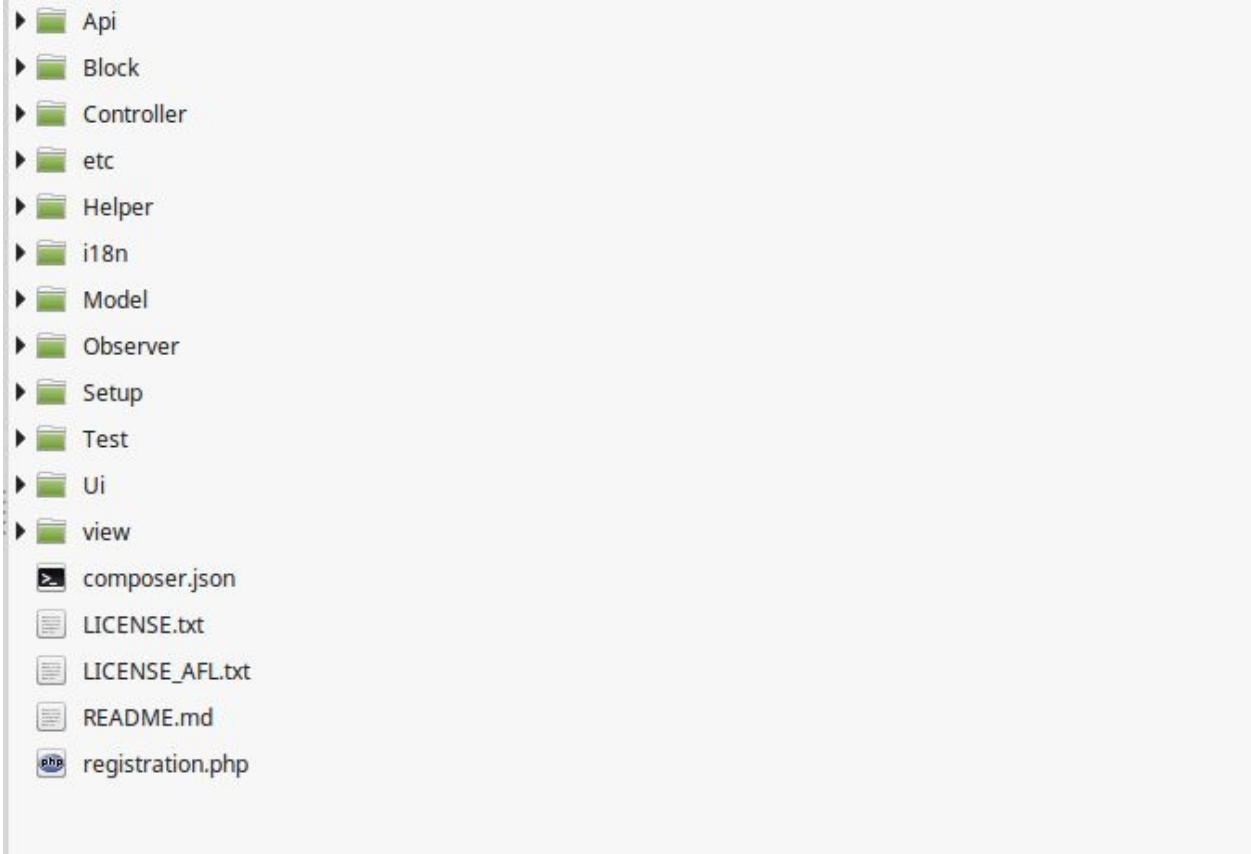
So that's how naming convention works. In the coming explanation, we will consider following vendor and module name.

Vendor Name: MyVendor

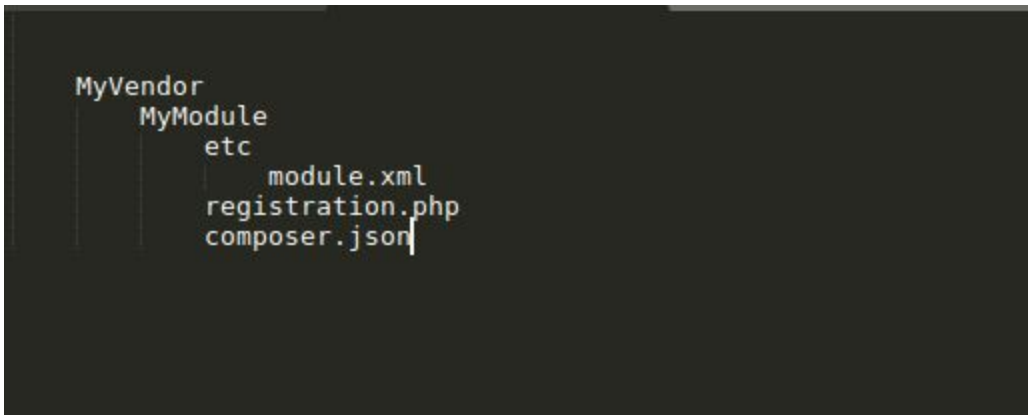
Module Name: MyModule

Files needed to create a new module

Please goto "vendor/magento/module-cms" directory, and you will see a list of directories and files.



To create a basic magento module, all you need to do is create following basic files.



Reference Link:
https://devdocs.magento.com/guides/v2.3/extension-dev-guide/build/module-file-structure.html

https://inchoo.net/magento-2/how-to-create-a-basic-module-in-magento-2/

These are the essential files to create a basic module.

registration.php

There can be 4 types of modules in Magento.

- Module
- Theme
- Language
- Library

Reference Link:

https://devdocs.magento.com/guides/v2.3/extension-dev-guide/build/component-registration.html

For the time being, we will first understand the module part from above 4 types.

The code of the **registration.php** will look something like below.

```php
<?php

\Magento\Framework\Component\ComponentRegistrar::register(
    \Magento\Framework\Component\ComponentRegistrar::MODULE,
    'MyVendor_MyModule',
    __DIR__
);
```

Here the first argument we are passing is the type of module. As explained above, it can have the following possible values.

```php
\Magento\Framework\Component\ComponentRegistrar::MODULE

\Magento\Framework\Component\ComponentRegistrar::THEME

\Magento\Framework\Component\ComponentRegistrar::LANGUAGE

\Magento\Framework\Component\ComponentRegistrar::LIBRARY
```

The second argument is the name of the module.

And the third argument will be the module directory path.

__DIR__ is a PHP magic method and will return current directory path of the **registration.php**

etc/module.xml

This is another module configuration file and it is needed to specify module version and module sequence. The code will look something like below.

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

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

Module name contains the name of the module.

setup_version is the current module version.

Let's say you have created a module with version **1.0.0**, and published the module. After a week you are adding some more columns to existing table, you can create some script to add columns by increasing module version. Or if you find some bugs in existing module, you can solve them and release the new updated version of module by increasing the setup_version parameter.

Here you will see sequence tag. It is used to provide the sort order of module from being loaded. Sometimes you need to load your module after some module.

For example, if you are creating some checkout level customizations, You want **Magento_Checkout** module to be loaded first and then your module. There are some use cases like if multiple modules override same file so which file will get called. This can be sorted with use of sequence tag. We will see this in some examples later on about the sequence in order to have brief understanding.

composer.json

Composer.json contains a JSON object filled with useful information as below.

- Name and Description of your module
- Required php versions in order to install the module
- Required third party libraries in order to install the module

For example, you are creating a module for elasticsearch, you will need PHP elasticsearch library to use certain functions to interact with elasticsearch server.

- License information
- Author Information

```
{
    "name": "myvendor/mymodule",
    "description": "This is a sample module",
    "require": {
        "php": "~5.6.5|7.0.2|7.0.4|~7.0.6|~7.1.0|~7.1.3|~7.2.0",
        "php/elasticsearch": "^1.5.7"
    },
    "type": "magento2-module",
    "license": "GPL-3.0",
    "version": "1.0.0",
    "authors": [
        {
            "name": "MyVendor",
            "email": "info@myvendor.com"
        }
    ],
    "autoload": {
        "files": [
            "registration.php"
        ],
        "psr-4": {
            "MyVendor\\MyModule\\": ""
        }
    }
}
```

Reference Link:

https://getcomposer.org/doc/01-basic-usage.md

https://getcomposer.org/doc/articles/versions.md

Once you create a module with the above files,you need to execute following CLI commands in order to tell magento to process it.

php bin/magento setup:upgrade

php bin/magento setup:di:compile

php bin/magento setup:static-content:deploy -f

In order to enable module, you can use following CLI command.

php bin/magento module:enable MyVendor_MyModule

In order to disable a module, you can use following CLI command.

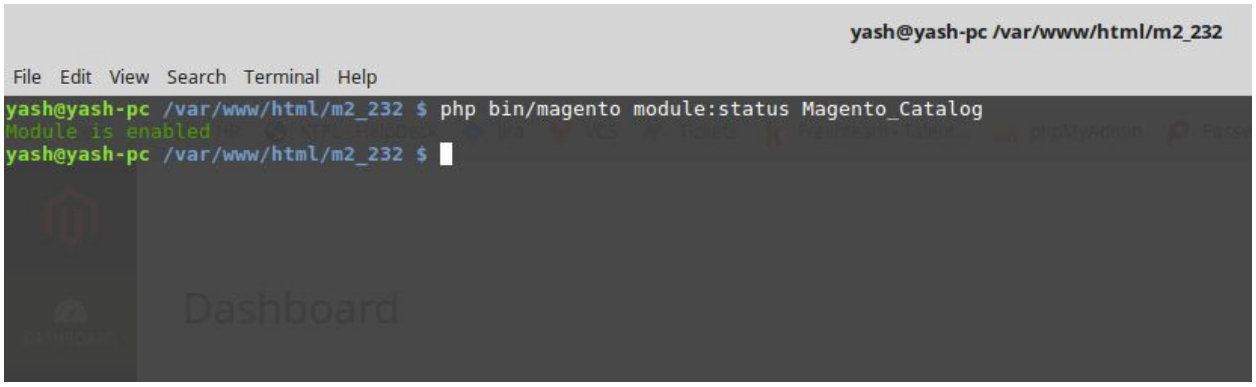
php bin/magento module:disable MyVendor_MyModule

You can check weather a module is enabled or not using following commad.

php bin/magento module:status

php bin/magento module:status MyVendor_MyModule

The first command will give you status of all installed modules, while the second command will give you status of particular module.



Reference Link:

https://devdocs.magento.com/guides/v2.3/install-gde/install/cli/install-cli-subcommands-enable.html

However, there is another way as weill to enable/disable module. You can login to admin panel and goto **System -> Web Setup Wizard -> Module Manager**

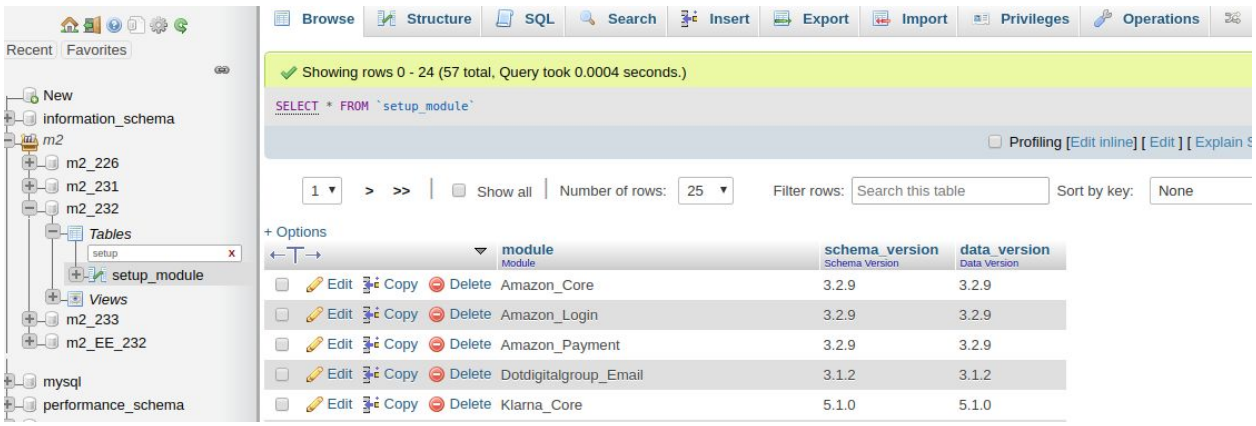
And choose action to enable/disable a module from there.

Effects

Whenever you install a module in magento, after performing the setup:upgrade command, you will see your module name at following 2 places.

- app/etc/config.php
- Setup_module database table

```php
<?php
return [
    'modules' => [
        'Magento_Store' => 1,
        'Magento_AdvancedPricingImportExport' => 1,
        'Magento_Directory' => 1,
        'Magento_Amqp' => 1,
        'Magento_Config' => 1,
        'Magento_Theme' => 1,
        'Magento_Backend' => 1,
        'Magento_Variable' => 1,
        'Magento_Eav' => 1,
        'Magento_Search' => 1,
        'Magento_Backup' => 1,
        'Magento_Customer' => 1,
        'Magento_AdminNotification' => 1,
        'Magento_Authorization' => 1,
        'Magento_BundleImportExport' => 1,
        'Magento_Indexer' => 1,
        'Magento_CacheInvalidate' => 1,
        'Magento_Cms' => 1,
        'Magento_Catalog' => 1,
        'Magento_Security' => 1,
        'Magento_GraphQl' => 1,
        'Magento_CatalogImportExport' => 1,
        'Magento_Quote' => 1,
        'Magento_CatalogInventory' => 1,
```



In the **app/etc/config.php** file, you will see module entry with value 0 or 1.

0 indicates that the module is disabled

1 indicates that the module is enabled.

So if you want to enable or disable a module, you can do that by changing value in this file as well but it is not a standard way to do that.

CLI command and Admin Web Setup Wizard explained above, are the correct ways to enable or disable a module according to best practices.

Download:

http://yashshah.net/magento2/basic_module.zip

Assignment:

- Create a basic module with following Information
  - Vendor Name: Born
  - Module Name: SampleModule
- See its entry in relevant files and database table
- Enable/Disable module via all below
  - Command Line
  - Web Setup Wizard
  - app/etc/config.php

Factory vs Proxy

To understand **Factory** and **Proxy**, We first need to understand how to create class instances in Magento 2.

Normally in PHP, we are creating any class instance with **new** keyword. Just like below.

```
class A
{
    public function __construct()
    {

    }
}

$a = new A();
```

In Magento, there are 2 ways to create an instance of a class.

1. Using constructor arguments
2. Using ObjectManager

You will see the first method to be used in entire Magento. The second method is available but it is not advisable to use it. There are reasons behind it which we will discuss later on.

See the following File.

vendor/magento/module-cms/Helper/Page.php

```
class Page extends \Magento\Framework\App\Helper\AbstractHelper
{
    public function __construct(
        \Magento\Framework\App\Helper\Context $context,
        \Magento\Framework\Message\ManagerInterface $messageManager,
        \Magento\Cms\Model\Page $page,
        \Magento\Framework\View\DesignInterface $design,
        \Magento\Cms\Model\PageFactory $pageFactory,
        \Magento\Store\Model\StoreManagerInterface $storeManager,
        \Magento\Framework\Stdlib\DateTime\TimezoneInterface $localeDate,
        \Magento\Framework\Escaper $escaper,
        \Magento\Framework\View\Result\PageFactory $resultPageFactory
    ) {
        $this->messageManager = $messageManager;
        $this->_page = $page;
        $this->_design = $design;
        $this->_pageFactory = $pageFactory;
        $this->_storeManager = $storeManager;
        $this->_localeDate = $localeDate;
        $this->_escaper = $escaper;
        $this->resultPageFactory = $resultPageFactory;
        parent::__construct($context);
    }

    public function getPageUrl($pageId = null)
    {
        /** @var \Magento\Cms\Model\Page $page */
        $page = $this->_pageFactory->create();
        if ($pageId !== null) {
            $page->setStoreId($this->_storeManager->getStore()->getId());
            $page->load($pageId);
        }

        if (!$page->getId()) {
            return null;
        }

        return $this->_urlBuilder->getUrl(null, ['_direct' => $page->getIdentifier()]);
    }
}
```

Here, you will see that whatever classes they want to use, they have injected in constructor and later on use that in the same class. Factory and Proxy comes in picture depends on the scenario.

Factory

Factory is nothing but an architecture to use an existing instance of an object or create a new fresh instance of an object in Magento way.

Consider the following core PHP concept.

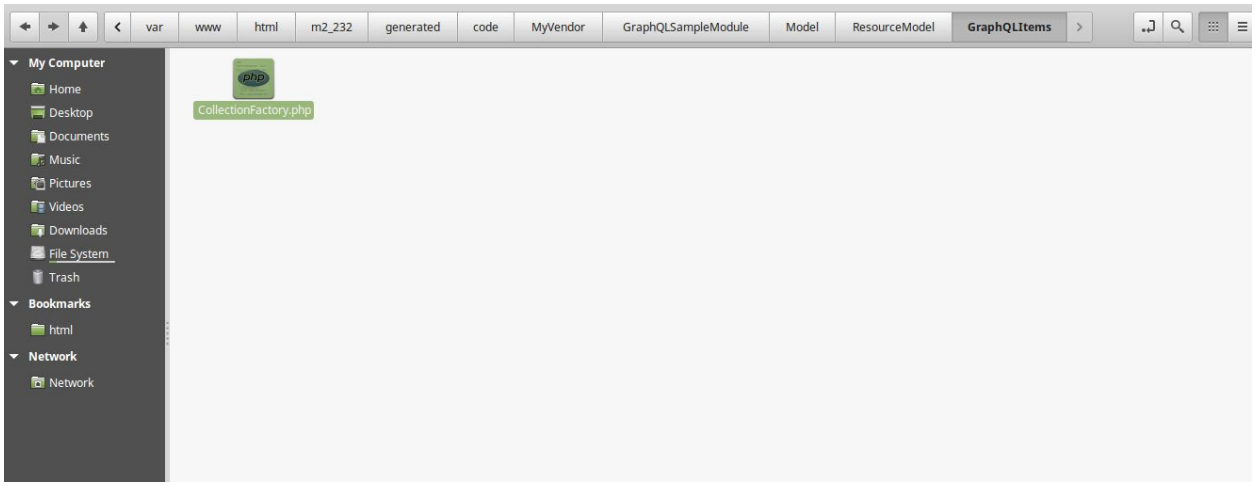
```
class A
{
    private $name = 'unknown';

    public function getName()
    {
        return $name;
    }

    public function setName($name)
    {
        $this->name = $name;
    }
}

$a = new A();
$a->setName("xyz");

echo $a->getName();

$a = new A();
echo $a->getName();

Output:
// xyz
// unknown
```

Here, we are creating an instance of class A 2 times. So both times it will create a new variable in memory with all properties of A. Which will consume more memory into the system.

There are times when we do not need any new object. We can use the same object which has already been created. In some different class or file. This concept is called Factory.

Any Class Appended the "**Factory**" keyword at the end, will be used as Factory class and will be generated in the "**generated**" directory.

Whenever you create a class in any module. It will create a Factory Class file in generated directory.



This factory will have 1 method to access.

- create()
  This will create a new instance of an object

Consider the following example that demonstrate the factory usage.

https://github.com/yash7690/magento2-factory-samplemodule

Why this is needed ?

Let's say you are developing something on product listing page, a custom product listing in sidebar. You want to have Product Collection class. If you use the get method, you will receive existing filters applied to product. But you do not want those filters. So in this case, all you need to use is a create method, which will create a fresh instance without any filters.

This is generally passed in constructor so that we can use it in our system.

If you want to have already instantiated class, You can inject class without Factory keyword at the end.

Proxy

A Proxy class is something that is initializing when its first method is getting called. Lets say you have 10 classes to be injected in your constructor out of which there are 3 classes to be called based on conditions.

Those classes have again a list of another classes as constructor arguments. Loading those classes without fulfilling conditions will make your system heavy and slow.

Consider following example.

```
class A
{
    public function __construct(
        AnotherClass1,
        AnotherClass2
    ) {

    }
}

class AnotherClass2
{
    public function __construct(
        AnotherClass10,
        AnotherClass11,
        AnotherClass12,
        AnotherClass13
    ) {

    }
}


$a = new A(AnotherClass1, AnotherClass2);

$a = new A(AnotherClass1, AnotherClass2Proxy);
```

Customizing the Magento UI

Demonstrate the ability to customize the Magento UI using themes

You are already familiar with how to create a module in Magento. At that time, it was mentioned that a component can be of 4 types.

- Module
- Theme
- Language
- Library

We will talk about them theme now.

You can always customize magento theme in a very easy way.

Reference Link
https://devdocs.magento.com/guides/v2.3/frontend-dev-guide/themes/theme-create.html

The high-level steps required to add a new theme in the Magento system are the following:

1. Create a directory for the theme under app/design/frontend/<Vendor>/<theme>.
2. Add a declaration file theme.xml and optionally create etc directory and create a file named view.xml to the theme directory.
3. Add a composer.json file.
4. Add registration.php.
5. Create directories for CSS, JavaScript, images, and fonts.
6. Configure your theme in the Admin panel.

Sample Module

https://github.com/yash7690/magento2-sampletheme

Your theme.xml will look like this.

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

<theme xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation
="urn:magento:framework:Config/etc/theme.xsd">
    <title>MyVendor Custom Theme</title> <!-- your theme's name -->
    <parent>Magento/luma</parent> <!-- the parent theme, in case your theme inherits from
    an existing theme -->
    <media>
        <preview_image>media/preview.jpg</preview_image> <!-- the path to your theme's
        preview image -->
    </media>
</theme>
```

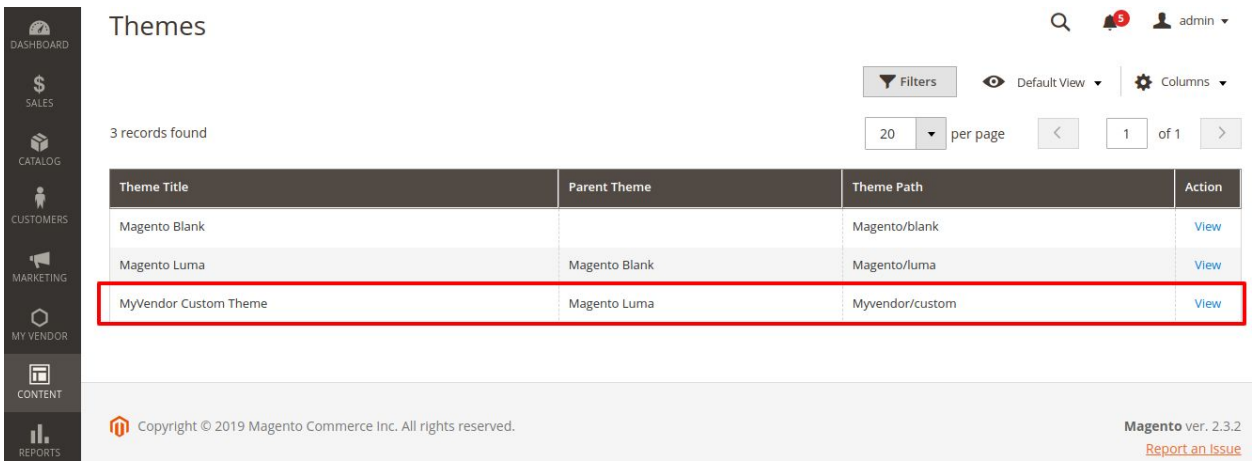
title: This will be theme title which will be shown in admin panel

parent: The name of parent theme which this theme can inherit

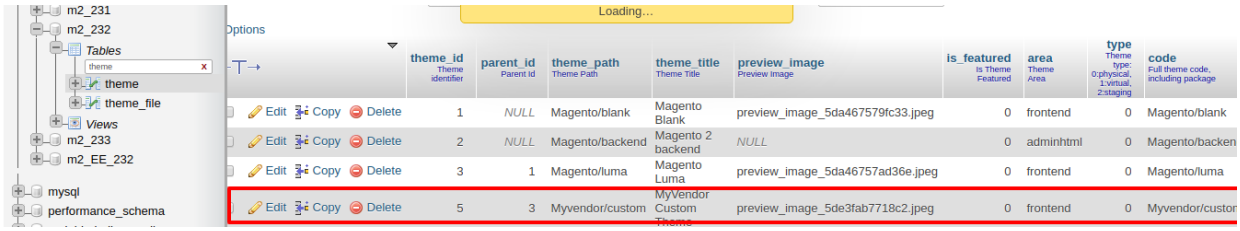
preview_image: when you view the theme in admin panel, it will show you this image.

Once you install the theme, login to your admin panel and visit below path.

> Menu -> Content -> Design -> Themes



Also check table "**theme**" in your database, Your theme entry will fall into the same table.



Now it is time to apply your theme.

To apply a theme, goto

> **Menu -> Content -> Configuration -> Edit Appropriate Store View -> Select Your theme -> Save**

Once you saved the theme, create following directory structure.

> web/css/source

Copy **_sources.less** file and paste into above directory.

Import **mycustom.less** file at the end of above file.

Create mycustom.less file and add following code.

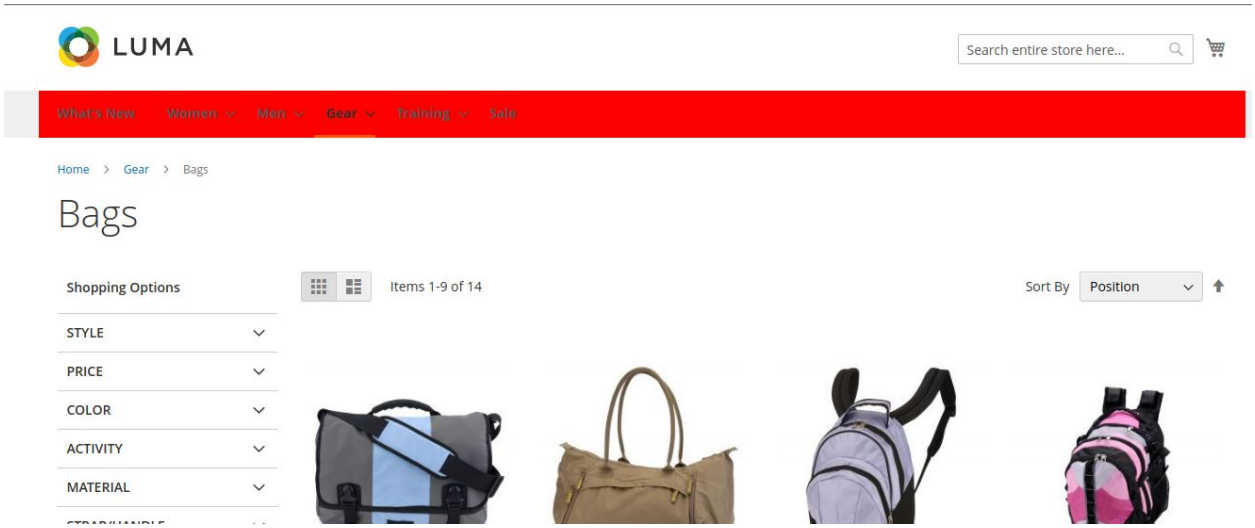
```
.navigation ul {
    background-color: red;
}
```

Now clear static content by applying following commands.

> rm -rf var/view_preprocessed/* pub/static/frontend/*

> php bin/magento setup:static-content:deploy -f

Now visit a product listing page and see the menu background color. It will be red just as below image.



So this way you can apply new theme and play with it according to your needs.

There are much more than this in theming but you can explore it yourself.

Assignments

1. Create a custom theme and check its reflections
2. Also upload theme logo
3. Check how to add custom favicon icon for your theme
4. Schedule your theme for specific time frame settings

Describe the elements of the Magento layout XML schema, including the major XML directives

Any layout xml in magento can have basically following types of elements.

- container

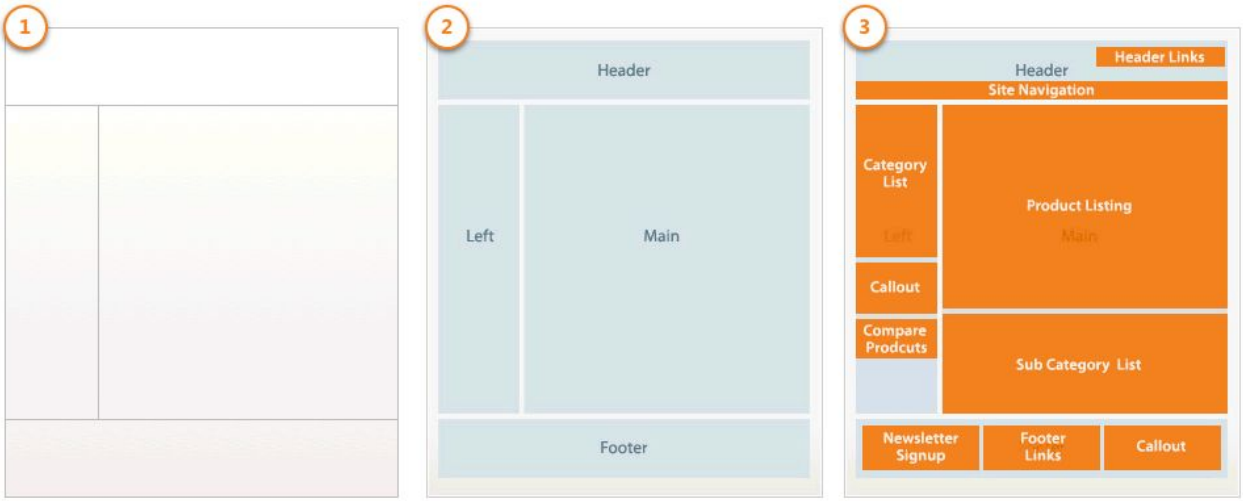
- block
- referenceContainer
- referenceBlock

In Magento, the basic components of page design are layouts, containers, and blocks. A layout represents the structure of a web page.

Containers represent the placeholders within that web page structure

And blocks represent the UI controls or components within the container placeholders

These terms are illustrated and defined below.



(1) Layouts provide the structures for web pages using an XML file that identifies all the containers and blocks composing the page. The details of layout XML files are described later in this section.

(2) Containers assign content structure to a page using container tags within a layout XML file. A container has no additional content except the content of included elements. Examples of containers include the header, left column, main column, and footer.

(3) Blocks render the UI elements on a page using block tags within a layout XML file. Blocks use templates to generate the HTML to insert into its parent structural block. Examples of blocks include a category list, a mini cart, product tags, and product listing.

Container

A container is considered to be a group of blocks, that is a wrapper html element of blocks to be combined into a unit.

Consider following example.

```
<container name="category.view.container" htmlTag="div"
htmlClass="category-view" after="-">
```

Look at the following URL.

https://devdocs.magento.com/guides/v2.3/frontend-dev-guide/layouts/xml-instructions.html

Block

A block is considered to be small unit that is resposible to output html content with different business logic.

For example, a product detail page will show product images, add to cart button, produc options etc. So we can use different blocks to render them to chunk codebase into smaller units which makes system easier to modify the code anywhere wih the use of additional 2 elements.

referenceContainer

If you want to add any other block to any existing container, you can use this element with the name of the original container. Then you can add your own elements.

The best example of it can be adding any javascript like google analytics at the end of the page using following referenceBlock.

<referenceContainer name="before.body.end">

You can add your custom block here to add any additional code. Try with this.

referenceBlock

Same as refereceContainer, if you want to add some additional block to to any existing block, you can use this tag.

For example, consider below code.

```
<referenceBlock name="header.links">
        <block name="additional_header_link">
          <arguments>
            <argument name="label" xsi:type="string" translate="true">Additional Header Link</argument>
            <argument name="path" xsi:type="string">a/b/c</argument>
          </arguments>
        </block>
      </referenceBlock>

      <referenceBlock name="footer_links">
        <block name="additional_footer_link">
          <arguments>
            <argument name="label" xsi:type="string" translate="true">Additional Link</argument>
            <argument name="path" xsi:type="string">x/y/z</argument>
```

```
        </arguments>
    </block>
</referenceBlock>
```

Apart from the above 4 elements, layout xml can have ui components to render grid and form which we will be seeing in developing with adminhtml section.

Demonstrate an ability to create UI customizations using a combination of a block and template

Block and template are related to each other. Every Block in Magento is associated with a template to render.

This definition can be tightly coupled within the layout xml or in the block php file $_template variable.

Consider the following example.

```
<?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_configuration.xsd">
    <body>
        <referenceContainer name="content">
            <block class="MyVendor\RoutingSampleModule\Block\DynamicBlock" name="
            my_custom_routing_block" template="MyVendor_RoutingSampleModule::dynamic.phtml
            "></block>
        </referenceContainer>
    </body>
</page>
```

In this, a block node has a class, name and template nodes.

Class: class is used to define a PHP class name, which is used to supply there data to template file. In addition to the above example, consider below template file.

```
<?= $block->callMyFunction() ?>
```

So the function **callMyFunction,** will have a definition in the class name mentioned in class node of Block file.

This way you can have any number of blocks associated with a template. Data processor to be the Block class file where you can have a dependency on other class.

One more interesting topic Magento has released in ViewModels.

I will explain the concept of view models using practical.

Layout xml file can have multiple as well as multi level blocks.

Consider below example.

```
<?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_configuration.xsd">
    <body>
        <referenceContainer name="content">
            <block class="MyVendor\RoutingSampleModule\Block\DynamicBlock" name="
            block1" template="MyVendor_RoutingSampleModule::dynamic.phtml">
                <block class="MyVendor\RoutingSampleModule\Block\DynamicBlock" name="
                nested1block" template="MyVendor_RoutingSampleModule::dynamic.phtml">
                    <block class="MyVendor\RoutingSampleModule\Block\DynamicBlock"
                    name="nested2block" template="
                    MyVendor_RoutingSampleModule::dynamic.phtml"></block>
                </block>
            </block>

            <block class="MyVendor\RoutingSampleModule\Block\DynamicBlock" name="
            block2" template="MyVendor_RoutingSampleModule::dynamic.phtml"></block>
        </referenceContainer>
    </body>
</page>
```

So a block can have inner block. In the template file, an inner block can be rendered with **$this->getChild('name_of_the_block')** method.

Now we will add title of about us page using block injection.

Sample Module

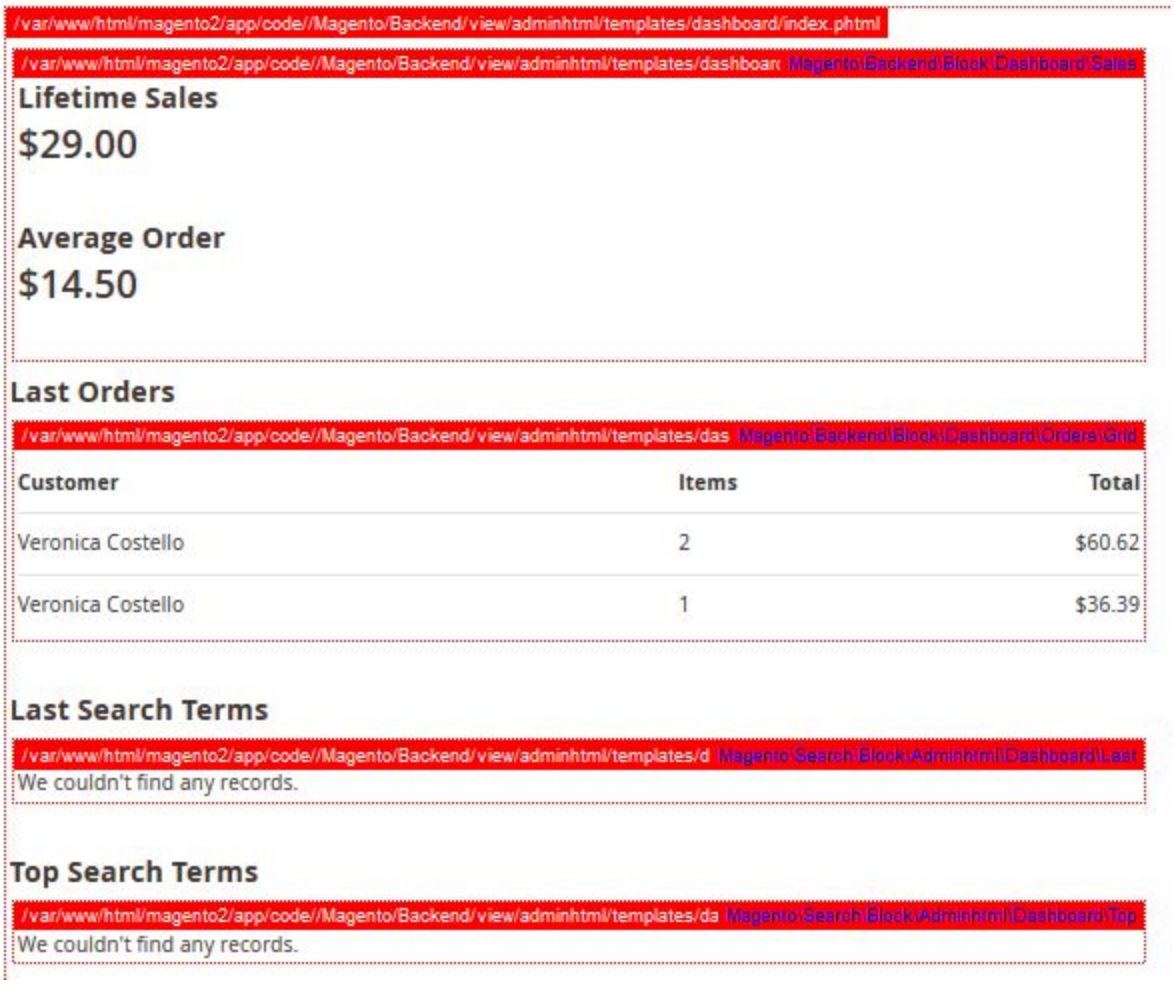
https://github.com/yash7690/magento2-blockinjectionsamplemodule

Things interesting to know

When you are working on frontend, on any live website, it can take some time to identify which area comes from which block and template.

Magento gives and easy interface to identify this, which is called "Enable Template Path Hints".

You can enable this via admin panel from store configuration tab. As a result, it will show you something like below image.



So it makes pretty much easy to identify the things.

If you want to know the basic page layout, look at the body class, and it will pretty much tell you which layout it gets called. You can also add other layouts in controller file. See and debug layout for product details page.

Assignments

- Create your own controller, router to print a simple hello using block. Call any block function to render output.
- Add some custom information on product details page after add to cart button using layout xml updates

- Add some custom information on product details page with specific product id “1” after add to cart button
- Add a new layout xml x_y_z.xml and render some block for any specific product via admin product edit page, layout update.
- Check in controller, $page->getLayout()->getUpdate()->getHandles() output in either product page or your own custom page to know what layout updates are being considered.

Working with Databases in Magento

Working with database is a well managed codebase in Magento 2. Magento 2 has implemented in such a way that it can be managed creating some simple files.

To access any database, we just need to create 3 files.

- **Model/MyModel.php**
- **Model/ResourceModel/MyModel.php**
- **Model/ResourceModel/MyModel/Collection.php**

The first file **Model/MyModel.php** is used to get and set data to your table.

The second file **Model/ResourceModel/MyModel.php** is having the same purpose. The difference here is if you want to make some join, group by or any other mysql related thing, you can write the code in this file.

The **Collection.php** is used to fetch table rows based on some condition.

Reference Link

https://www.mageplaza.com/magento-2-module-development/how-to-create-crud-model-magento-2.html

To make it very simple, remember following steps.

Model will extend **\Magento\Framework\Model\AbstractModel**

Model will have a **_contruct** method, that will **_init ResourceModel**

ResourceModel will extend **\Magento\Framework\Model\ResourceModel\Db\AbstractDb**

ResourceModel will have **_construct** method, that will **_init** with 2 parameters.

- Table name
- Primary key

Collection will extend
\Magento\Framework\Model\ResourceModel\Db\Collection\AbstractCollection

Collection will have **_contruct** method, that will **_init Model** and **ResourceModel** as arguments.

Consider the following code.

Model

```php
<?php

namespace MyVendor\MyModule\Model;

class MyModel extends \Magento\Framework\Model\AbstractModel implements \Magento\Framework\
    DataObject\IdentityInterface
{
    const CACHE_TAG = 'myvendor_mymodule_mymodel';
    protected $_cacheTag = 'myvendor_mymodule_mymodel';
    protected $_eventPrefix = 'myvendor_mymodule_mymodel';

    protected function _construct()
    {
        $this->_init(
            MyVendor\MyModule\Model\ResourceModel\MyModel::class
        );
    }
}
```

ResourceModel

```php
<?php

namespace MyVendor\MyModule\Model\ResourceModel;

class MyModel extends \Magento\Framework\Model\ResourceModel\Db\AbstractDb
{
    public function __construct(
        \Magento\Framework\Model\ResourceModel\Db\Context $context
    )
    {
        parent::__construct($context);
    }

    protected function _construct()
    {
        $this->_init('table_name', 'primary_key_colmn');
    }

}
```

Collection

```php
<?php

namespace MyVendor\MyModule\Model\ResourceModel\MyModel;

class Collection extends \Magento\Framework\Model\ResourceModel\Db\Collection\
    AbstractCollection
{
    protected $_idFieldName = 'primary_key_column';
    protected $_eventPrefix = 'myvendor_mymodule_mymodel_collection';
    protected $_eventObject = 'mymodel_collection';

    protected function _construct()
    {
        $this->_init(
            \MyVendor\MyModule\Model\MyModel::class,
            \MyVendor\MyModule\Model\ResourceModel\MyModel::class
        );
    }

}
```

Also you need to know about **DDL** here.

DDL is **D**ata **D**efinition **L**anguage or **D**ata **D**eclaration **L**anguage. Rather to write straight forward sql queries, In setup scripts, you need to write code in DDL format. It is easier to understand and extend.

Let's create one simple module to save, load and print collection of a table.

Reference Link

https://inchoo.net/magento-2/setup-scripts-magento-2/

Sample Module

https://github.com/yash7690/magento2-db-samplemodule

Declarative Schema Usage

Declarative Schema aims to simplify the Magento installation and upgrade processes. Previously, developers had to write database scripts in PHP for each new version of Magento. Various scripts were required for

- Installing and upgrading the database schema
- Installing and upgrading data
- Invoking other operations that are required each time Magento was installed or upgraded

When a customer upgrades Magento to a version several releases ahead of the installed version, the upgrade script for each intermediate release still executes. Developers were required to fully understand what each install and upgrade script contained. They needed to account for this complexity when creating extensions.

The new declarative schema approach allows developers to declare the final desired state of the database and has the system adjust to it automatically, without performing redundant operations. Developers are no longer forced to write scripts for each new version. In addition, this approach allows data be deleted when a module is uninstalled.

Implementing declarative schema is not a requirement for Magento 2.3. However, upgrade scripts will be phased out in favor of declarative schema.

Declarative schema has 2 parts.

1. Schema Patch

2. Data Patch

The schema patch is written inside following file.

> **etc/db_schema.xml**

> **etc/db_schema_whitelist.json**

To generate a whitelist json file, you need following command to be executed.

php bin/magento setup:db-declaration:generate-whitelist --module-name=<module_name>

Reference Link

https://devdocs.magento.com/guides/v2.3/extension-dev-guide/declarative-schema/migration-commands.html

Data Patch

Instead of creating long long file for version management, Magento has introduced a new way to apply patches.

The Data Patches can be applied in the following directory.

> Setup/Patch/Data/<patch_name>.php

Reference Link

https://devdocs.magento.com/guides/v2.3/extension-dev-guide/declarative-schema/data-patches.html

```php
<?php
    /**
     * Copyright © Magento, Inc. All rights reserved.
     * See COPYING.txt for license details.
     */

    namespace Magento\DummyModule\Setup\Patch\Data;

    use Magento\Framework\Setup\Patch\DataPatchInterface;
    use Magento\Framework\Setup\Patch\PatchRevertableInterface;

    /**
     */
    class DummyPatch
        implements DataPatchInterface,
        PatchRevertableInterface
    {
        /**
         * @var \Magento\Framework\Setup\ModuleDataSetupInterface
         */
        private $moduleDataSetup;

        /**
         * @param \Magento\Framework\Setup\ModuleDataSetupInterface $moduleDataSetup
         */
        public function __construct(
            \Magento\Framework\Setup\ModuleDataSetupInterface $moduleDataSetup
        ) {
            /**
             * If before, we pass $setup as argument in install/upgrade function, from now we
             * inject it with DI. If you want to use setup, you can inject it, with the same
             */
            $this->moduleDataSetup = $moduleDataSetup;
        }

        /**
         * {@inheritdoc}
         */
        public function apply()
        {
            $this->moduleDataSetup->getConnection()->startSetup();
            //The code that you want apply in the patch
            //Please note, that one patch is responsible only for one setup version
            //So one UpgradeData can consist of few data patches
            $this->moduleDataSetup->getConnection()->endSetup();
        }

        /**
         * {@inheritdoc}
         */
        public static function getDependencies()
        {
            /**
             * This is dependency to another patch. Dependency should be applied first
             * One patch can have few dependencies
             * Patches do not have versions, so if in old approach with Install/Ugrade data s
             * versions, right now you need to point from patch with higher version to patch
             * But please, note, that some of your patches can be independent and can be inst
             * So use dependencies only if this important for you
             */
            return [
```

A Data Patch can have 4 functions.

- apply()

  In this function, we have to write code to apply the patch for

- getDependencies()

  In this function, we have to write down name of the classes or other patches, this patch is depending upon. Like the sort order of patch.

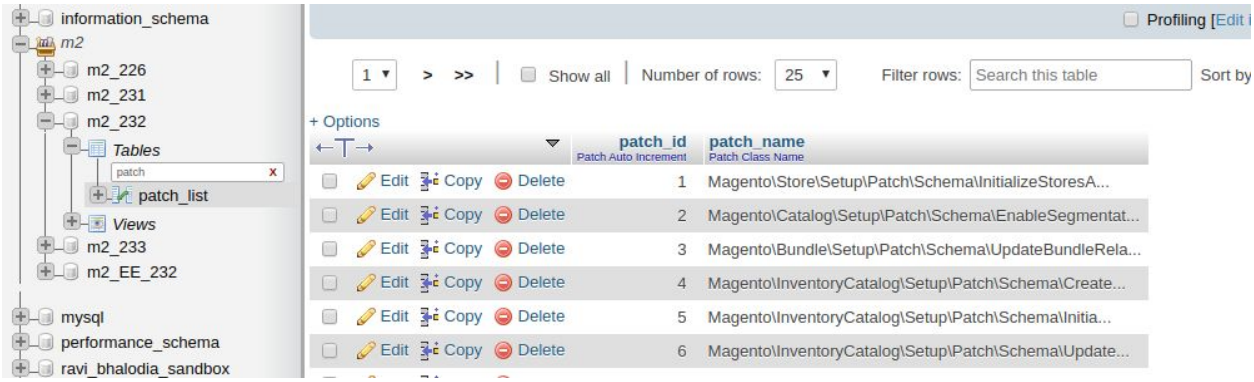
- revert()

  If apply function does not execute correctly, this will get called to revert the action.

- getAlias()

  The alias name for this patch.

Whenever a patch is applied, it is saved in a database table named **patch_list,** so that teh patch is not applied another time.



Assignments

1. Create db schema with a table ‘post’ with 3 fields.
   a. Post_id
   b. Post_title
2. Create white list json of this schema.
3. Add another column post_content using db schema
4. Remove column post_content using db_schema and see how it works
5. Create 2 data patches and apply all 4 methods to insert some data into your table.

Developing with Adminhtml

Adminhtml development consists of 3 major things.

1. **Admin Menu with ACL**

   This will create menu items in admin panel with appropriate user rights.

   ACL stands for Access Control List. With the use of this, super admin can defined if specific sub admin can have specific rights to view/change things in admin area.

2. **Admin Grid & Form**

   This is standard CRUD operation for any entity. Orders, Invoices, Products, Categories etc are the best examples of it.

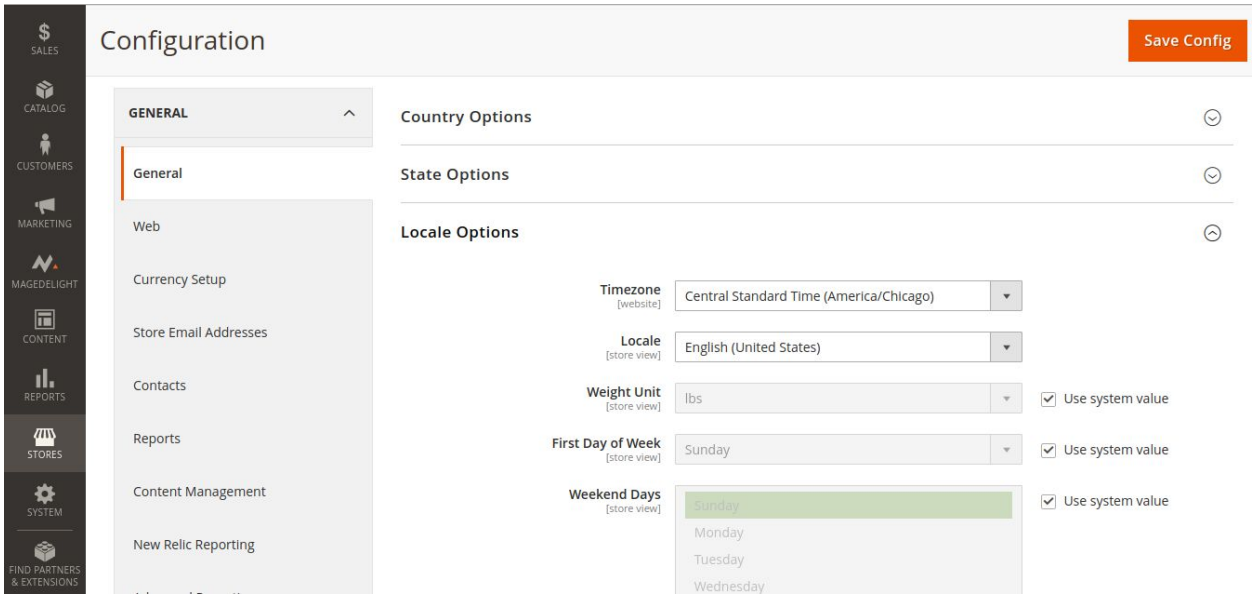
3. **Admin Store Configuration**

   This is related to configurations based on following layers.

   a. **Global**
   b. **Website**
   c. **Store**

   Global is considered to be Magento level setting.
   Website is considered to be Website level setting. For example, product prices. The scope of the product price is on website level.
   Store is considered to be Store level setting. For example, you can have some translation for on bases of Store View,

Here is an example of Store Configuration.



Some of them, You can see will be having global scope, some may have Website scope, while some may have Store View scope.

Payment Gateways enable/disable are considered to be website level scope, while their titles can be considered to be having Store View scope.

We will see an in depth example on how to create them with a module.

Before we start, Have a look at the following sample module.

https://github.com/yash7690/magento2-admin-samplemodule

Setup a Menu Item

To create an admin menu item, you need to create the following file.

> **etc/adminhtml/menu.xml**

This will look something like this.

```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="MyVendor_AdminSampleModule::root_menu"
            title="My Vendor"
            module="MyVendor_AdminSampleModule"
            sortOrder="45"
            resource="MyVendor_AdminSampleModule::root_menu" />

        <add id="MyVendor_AdminSampleModule::manage_items"
            title="Manage Items"
            module="MyVendor_AdminSampleModule"
            sortOrder="10"
            action="adminsamplemodule/items/index"
            parent="MyVendor_AdminSampleModule::root_menu"
            resource="MyVendor_AdminSampleModule::manage_items" />

        <add id="MyVendor_AdminSampleModule::configuration"
            title="Configuration"
            module="MyVendor_AdminSampleModule"
            sortOrder="99"
            parent="MyVendor_AdminSampleModule::root_menu"
            action="adminhtml/configuration/myvendor"
            resource="MyVendor_AdminSampleModule::configuration" />
    </menu>
</config>
```

Following is the description for each attribute.

id: This is the identification of a menu item, which can be used to give this menu item as parent to some other menu item or to make it selected from the controller.

title: TItle to be displayed

module: Which module, this menu item belongs to.

sortOrder: sortOrder of menu item to be displayed. This is useful when a menu item has sub menus to be created from different modules, we can define sortOrder as per our choice.

parent: parent menu item of current menu item. Here the id attribute of parent menu item will be passed.

resource: ACL resource so that this menu item should be visible to current admin or not.

action: Clicking on menu, should be redirected to this URL. We just need to provide URL slug. Base URL will be added by Magento itself.

dependsOnConfig: This can be used to show/hide menu item based on some system configuration value.

You must be wondering, what is ACL over here.

ACL file is used to manage which admin users can access which areas. The file is being created at following directory.

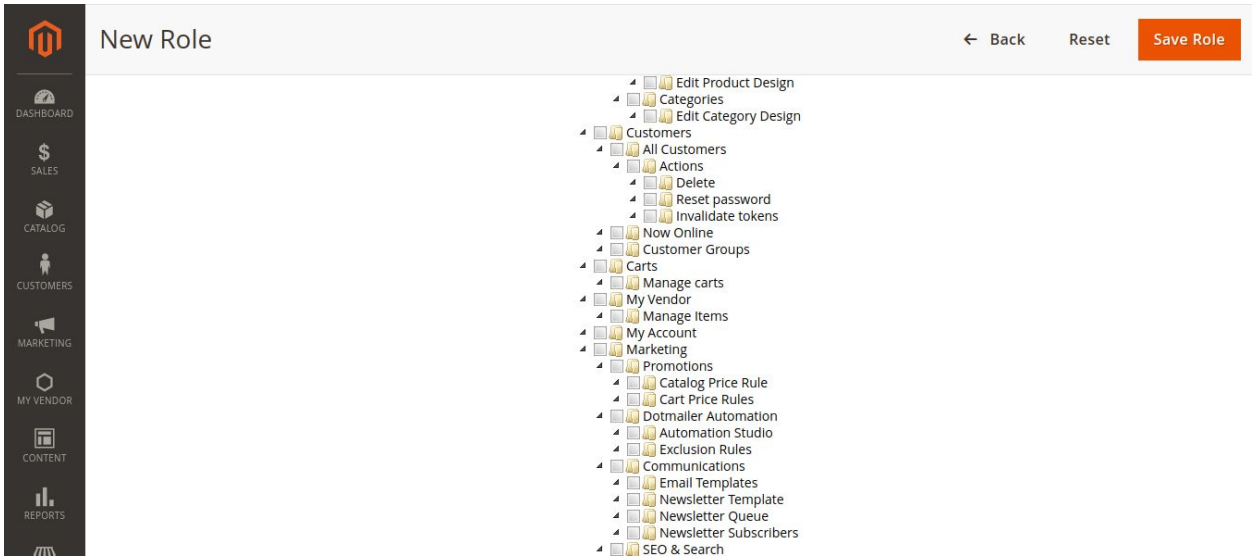
> **etc/acl.xml**

It will look something like this.

```
<?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="MyVendor_AdminSampleModule::root_menu" title="My Vendor" sortOrder="45">
                    <resource id="MyVendor_AdminSampleModule::manage_items" title="Manage Items"
                    sortOrder="10" />
                </resource>

                <resource id="Magento_Backend::stores">
                    <resource id="Magento_Backend::stores_settings">
                        <resource id="Magento_Config::config">
                            <resource id="MyVendor_AdminSampleModule::configuration" title="MyVendor
                            Configuration" sortOrder="45" />
                        </resource>
                    </resource>
                </resource>
            </resource>
        </resources>
    </acl>
</config>
```

You can check its effect via **System -> User Roles -> Add New Role -> Role Resources**

Here your created custom roles would be appear.



Go back to the menu.xml and see resource attribute. This is link to that.

System Configuration

Now we will come to what is store or system configuration. This is something we can manage store wise or website wise or globally.

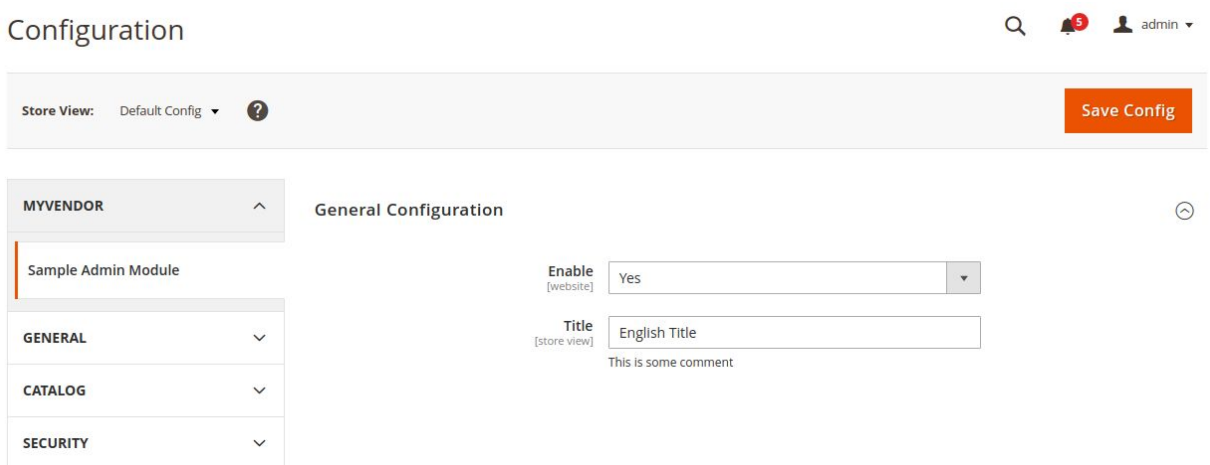
To create store configuration, all you need to create following file.

> **etc/adminhtml/system.xml**

The file will look something like this.

```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>
        <tab id="myvendor_adminsamplemodule" translate="label" sortOrder="45">
            <label>MyVendor</label>
        </tab>
        <section id="sampleadminmodule" translate="label" type="text" showInDefault="1" showInWebsite="1
        " showInStore="1" sortOrder="10">
            <class>separator-top</class>
            <label>Sample Admin Module</label>
            <tab>myvendor_adminsamplemodule</tab>
            <resource>MyVendor_AdminSampleModule::configuration</resource>

            <group id="general" translate="label" type="text" showInDefault="1" showInStore="1"
            showInWebsite="1" sortOrder="10">
                <label>General Configuration</label>

                <field id="enable" translate="label" type="select" showInDefault="1" showInWebsite="1"
                showInStore="0" sortOrder="10">
                    <label>Enable</label>
                    <source_model>Magento\Config\Model\Config\Source\Yesno</source_model>
                </field>

                <field id="title" translate="label" type="text" showInDefault="1" showInWebsite="1"
                showInStore="1" sortOrder="20">
                    <label>Title</label>
                    <depends>
                        <field id="enable">1</field>
                    </depends>
                    <comment><![CDATA[This is some comment]]></comment>
                </field>
            </group>
        </section>
    </system>
</config>
```

You can see that enable field has scope of website while title field has scope of store view. This will look something like below in admin panel.



Now we will learn how to create a controller in admin area.

Whenever we need to create a new URL in admin, we need to define the route in routes.xml

> etc/adminhtml/routes.xml

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

<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="
urn:magento:framework:Module/etc/module.xsd">
    <router id="admin">
        <route id="adminsamplemodule" frontName="adminsamplemodule">
            <module name="MyVendor_AdminSampleModule" />
        </route>
    </router>
</config>
```

This file tells Magento that whenever “adminsamplemodule” is found in URL, it will call controller inside this module.

We will now learn to display Grid and Form in admin area using routing. We will do this with the help of layout xml and ui component.

The layout xml is formed with URL parameters. Let’s say if the url is below,

http://localhost/m2_232/admin/adminsamplemodule/items/index/key/6efdba8ad2a70ef96573b39d7349329cfeee6a789f7677501673049204fd0282/

The layout xml will be named as “**adminsamplemodule_items_index.xml**” and we will be assigning Grid UI component to it.

Admin Grid and Forms

Before we get into this section, we need to get in touch with the following terms.

- layout

  This is responsible for rendering different blocks on a selected screen. The layout

file formed with joining the URL segment with an underscore, appended with .xml file extension.

This is located at

> view/<area>/layout/*.xml

- ui_component

  This is responsible to render a Grid or a Form in the admin panel. The primary purpose of it is being used is so that any custom module can add their customized fields in the same form.

  For example, consider Product Edit page. If you want to create EAV attributes programmatically and want to provide then in admin panel, you can do this by creating the same name ui_component.

  This is located at

  > view/<area>/ui_component/*.xml

  To render a grid component with the use of layout.xml, it must be defined the following way.

```
<?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">
    <update handle="styles"/>
    <body>
        <referenceContainer name="content">
            <uiComponent name="myvendor_adminsamplemodule_items_listing"/>
        </referenceContainer>
    </body>
</page>
```

  The Ui component defined here must be created at following location.

  > view/<area>/ui_component/ui_component_name.xml

  This UI Compoenent can be used to either create a Grid or a Form element.

  In case of Grid, it must be defined in following way.

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

<listing xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:
noNamespaceSchemaLocation="urn:magento:module:Magento_Ui:etc/ui_configuration.xsd">
    <argument name="data" xsi:type="array">
        ...
    </argument>
    <dataSource name="myvendor_adminsamplemodule_items_listing_data_source">
        <argument name="dataProvider" xsi:type="configurableObject">
            ...
        </argument>
    </dataSource>

    <listingToolbar name="listing_top">
        ...
    </listingToolbar>

    <columns name="spinner_columns">
        <selectionsColumn name="ids">
            <argument name="data" xsi:type="array">
                <item name="config" xsi:type="array">
                    <item name="resizeEnabled" xsi:type="boolean">false</item>
                    <item name="resizeDefaultWidth" xsi:type="string">55</item>
                    <item name="indexField" xsi:type="string">item_id</item>
                </item>
            </argument>
        </selectionsColumn>
        <column name="item_id">
            <argument name="data" xsi:type="array">
                <item name="config" xsi:type="array">
                    <item name="filter" xsi:type="string">textRange</item>
                    <item name="sorting" xsi:type="string">asc</item>
                    <item name="label" xsi:type="string" translate="true">ID</item>
                </item>
            </argument>
        </column>
        <column name="title">
            <argument name="data" xsi:type="array">
                <item name="config" xsi:type="array">
                    <item name="filter" xsi:type="string">text</item>
```

Its starts with **<listing>** node. The **DataProvider** node holds the value of grid collection name object which is defined in **di.xml** file.

The **listingToolbar** node contains massActions, pagination, filters etc settings.

If you are using the Form element, it should be somewhat like below.

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

<form xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation=
"urn:magento:module:Magento_Ui:etc/ui_configuration.xsd">
    <argument name="data" xsi:type="array">
        <item name="js_config" xsi:type="array">
            <item name="provider" xsi:type="string">
            myvendor_adminsamplemodule_items_form.items_form_data_source</item>
        </item>
        <item name="label" xsi:type="string" translate="true">Item Information</item>
        <item name="template" xsi:type="string">templates/form/collapsible</item>
    </argument>
    <settings>
        <buttons>
            <button name="save_and_continue" class="MyVendor\AdminSampleModule\Block\Admin
            html\Items\Edit\Buttons\SaveAndContinueButton"/>
            ...
        </buttons>
        <namespace>myvendor_adminsamplemodule_items_form</namespace>
        <dataScope>data</dataScope>
        <deps>
            <dep>myvendor_adminsamplemodule_items_form.items_form_data_source</dep>
        </deps>
    </settings>
    <dataSource name="items_form_data_source">
        <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>
        <settings>
            <submitUrl path="adminsamplemodule/items/save"/>
        </settings>
        <dataProvider class="MyVendor\AdminSampleModule\Model\Items\DataProvider" name="
        items_form_data_source">
            <settings>
                <requestFieldName>id</requestFieldName>
                <primaryFieldName>item_id</primaryFieldName>
```

This file starts with **<form>** tag and its dataSource node contains the value dataProvider file.

- Ui Component

  This is responsible to create actions on grid like

  View
  Edit
  Delete

- DataProvider

  This is responsible for providing data to a form ui component.

Will have a deep understanding of these terms in the above sample module

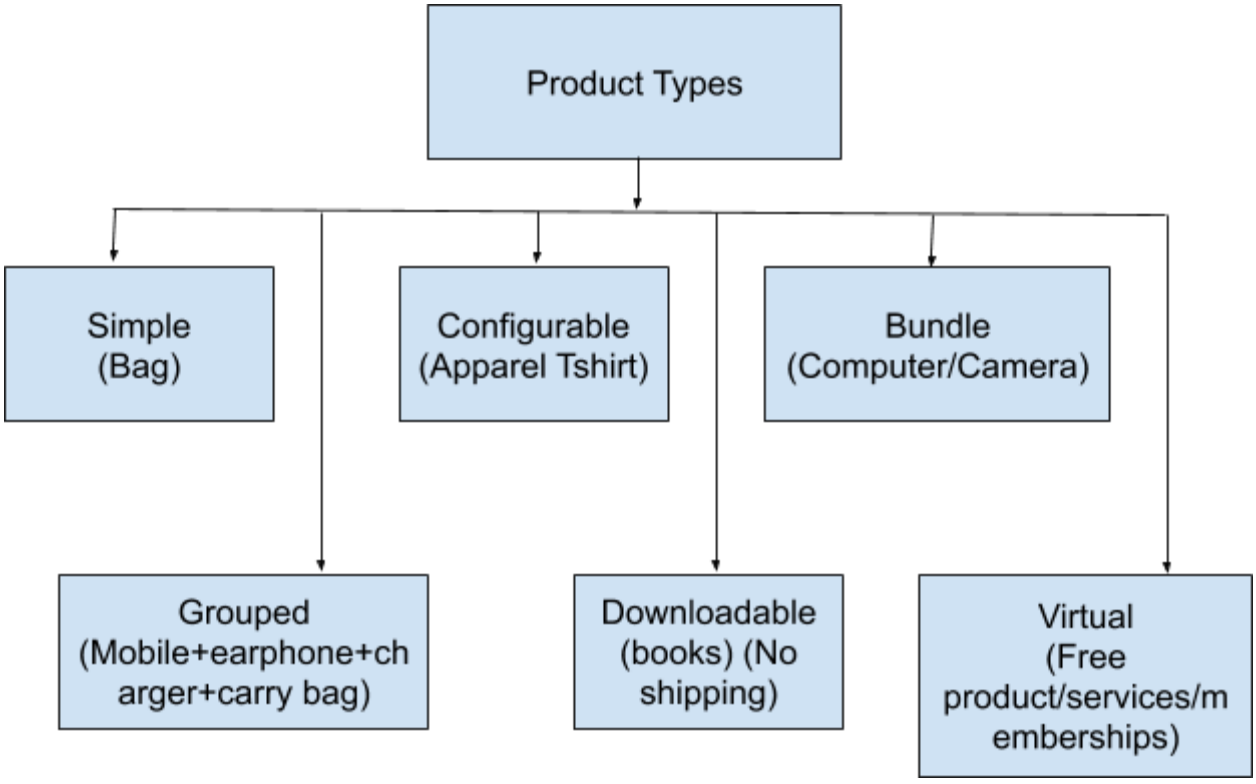
Assignments

1. Create a new module with following information.
    a. VendorName: Blog
    b. Module Name: Post
2. Create a setup script to add following columns.
    a. Post_id
    b. Title
    c. Content
    d. Slug (required, valid URL)
    e. Status
    f. Created_at
    g. Updated_at
3. Create a setup script to install 100 dummy data
4. Create a main menu item “Blog‘ with following Sub menu items.
    a. Manage Posts
    b. Configuration
5. Manage post will have CRUD operations
6. Configuration will have following options.
    a. Enable module - yes/no (website)
    b. Page Title (store view)
    c. Page SEO Url (store view) (required with valid URL js validation)
7. Create a new admin user who will have rights only to view your module and nothing else. And check if acl works or not.

Customizing Magento Business Logic

Identify/describe standard product types

Magento offers 6 default product types.



Simple Product

A simple product is a physical item with a single SKU. Simple products have a variety of pricing and of input controls which makes it possible to sell variations of the product. Simple products can be used in association with grouped, bundle, and configurable products.

Configurable Product

A configurable product appears to be a single product with lists of options for each variation. However, each option represents a separate, simple product with a distinct SKU, which makes it possible to track inventory for each variation.

Grouped Product

A grouped product presents multiple, standalone products as a group. You can offer variations of a single product, or group them for a promotion. The products can be purchased separately, or as a group.

Virtual Product

Virtual products are not tangible products, and are typically used for products such as services, memberships, warranties, and subscriptions. Virtual products can be used in association with grouped and bundle products.

Bundle Product

A bundle product let customers “build their own” from an assortment of options. The bundle could be a gift basket, computer, or anything else that can be customized. Each item in the bundle is a separate, standalone product.

Downloadable Product

A digitally downloadable product that consists of one or more files that are downloaded. The files can reside on your server or be provided as URLs to any other server.

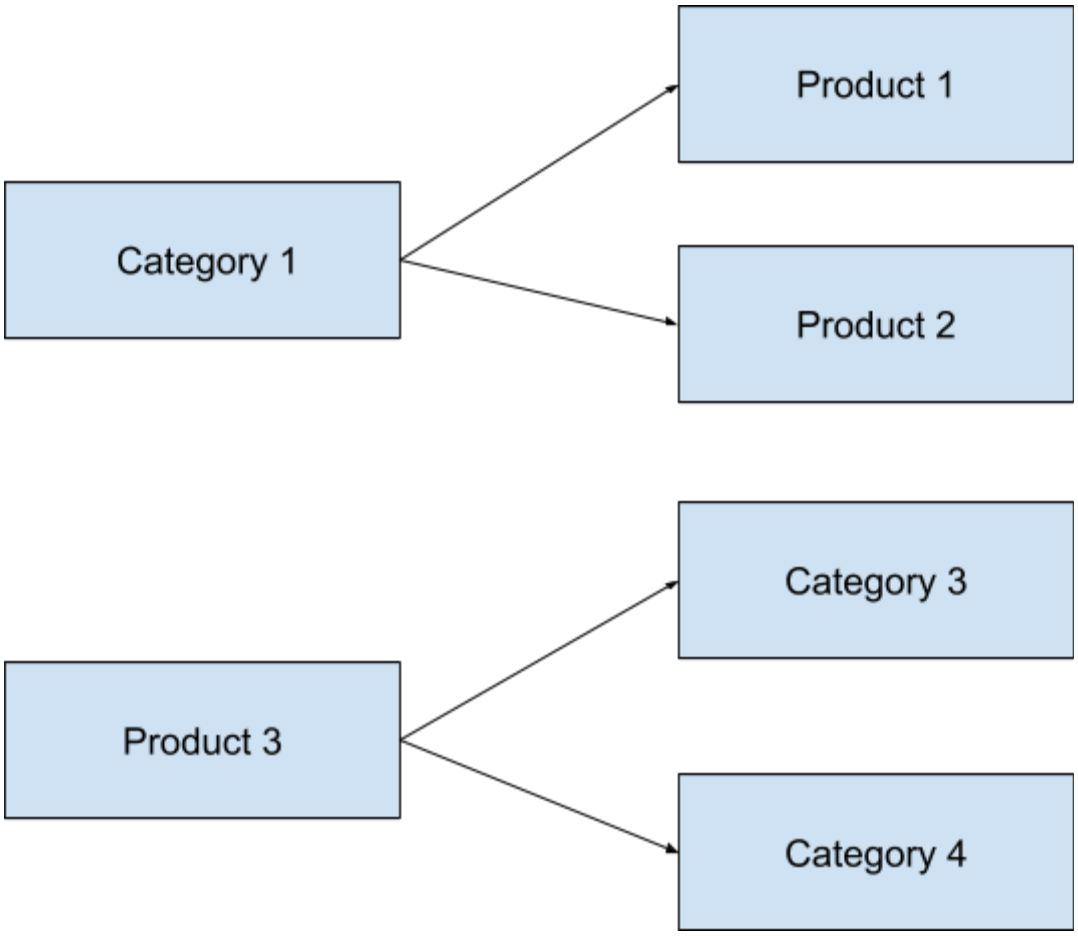
Describe category properties in Magento

When you create a category in the admin panel, you can dow following things

- Assign Products
- Assign Products sort order to display
- Standard attributes like
    - Status: enable/disable
    - Name
    - Image
    - Show in Menu
    - Description
- CMS block associated with that category to be shown
- Display Mode
    - Products only
    - Static block only
    - Products and static block both
- Anchor: display setting for categories allows a category to "inherit" and display all products from its child categories. When Is Anchor is set to "Yes", products assigned to child categories will be combined and displayed in the parent category along with any products directly assigned to it
- Available sort by options in product listing screen
- Default sorting option
- SEO information like
    - Url
    - Meta title
    - Meta keywords
    - Meta description
- Design Updates
    - Theme
    - Layout
    - Layout Update XML
    - Schedule update

Above does not need any explanation and we can check them practically to have a brief idea.

Define how products are related to the category



A product is having many-to-many relationships with the category.

Like a product can be assigned multiple categories same as a single category can have multiple products. There are no bindings.

You can simply check this by editing any category and assign products to it or edit any product and assign a category to it.

Describe the difference in behavior of different product types in the cart

The product being added to cart is created with the selected option you are choosing from product details page.

For example, if you add a simple product without any option to cart, it will always increase the qty of the same product.

But if you have defined custom options for that product, it will show you 2 different products if those are being added with different option value being selected from product details page.

Consider the case of configurable product, where you have a T-shirt with different sizes and colors. So 2 tshirts being added with different color will create 2 different cart items whereas the same color t shirt, qty will be increased.

We will see this practically.

Inventory

Inventory is a primary thing needed by any merchant.

Magento offers you to manage the stock as well as if for some items, you do not want stock management, there are options in admin panel.

Whenever you are configuring a product, you have an option to manage stock, current qty, qty of product to be notified to admin when reached that level etc.

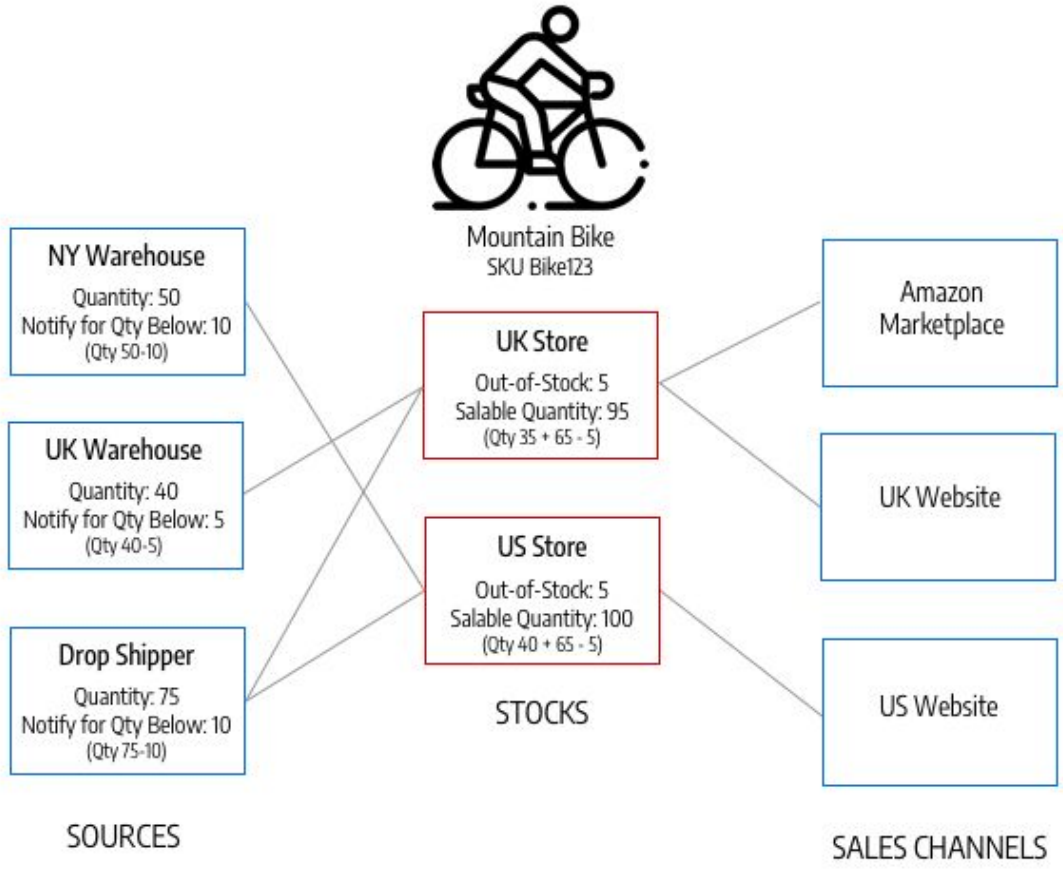
Whenever you place any order, the stock is being decreased when the product is shipped.

Check this using practical.

MSI

MSI stands for Multi Source Inventory. It is a new concept magento introduced with its latest 2.3 version.

To understand the MSI, simply consider following scenario.



In the following URL, the best example is given.

https://www.mageplaza.com/kb/multi-source-inventory-in-magento-2.html

However, this concept is recently launched by Magento and need some time to get into it.

Customers

A magento customer is someone who are registering themselves from frontend. You can see all the customers in admin panel through manage customers panel.

You can search/filter customers based on various fields.

Go through admin customer grid to know more.

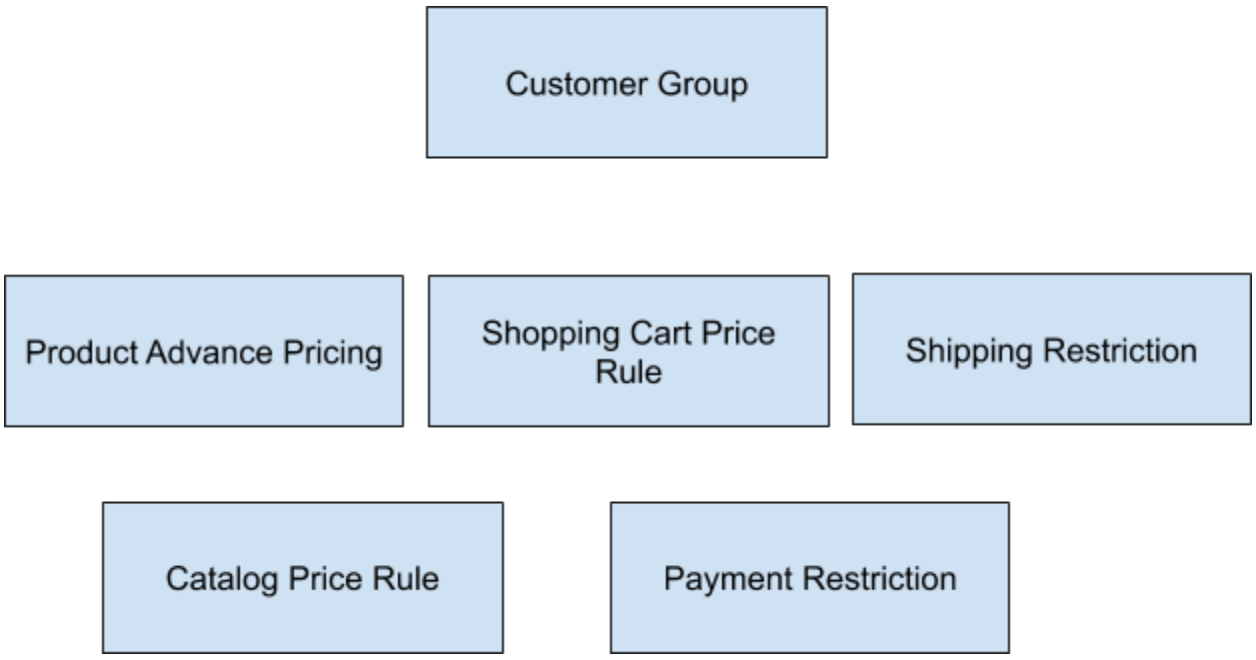
Customer Groups

A customer group is a group of customers so that i can be used to divide customers into different segments like

- General
- Wholesale
- Retailer

The primary purpose of creating a customer group and device customers into different groups is that admin can offer promotions, discounts, special pricing based on these groups.

A customer group relates to following things.



Customer Segments

This feature is offered by Magento Enterprise Edition, and not available with community edition.

Reference Link

https://docs.magento.com/m2/ee/user_guide/marketing/customer-segment-create.html

https://docs.magento.com/m2/ee/user_guide/marketing/customer-segment-price-rule.html

Customer segments allow you to dynamically display content and promotions to specific customers, based on properties such as customer address, order history, shopping cart contents, and so on. You can optimize marketing initiatives based on targeted segments with shopping cart price rules and banners. You can also generate reports and export the list of targeted customers. Because customer segment information is constantly refreshed, customers can become associated and disassociated from a segment as they shop in your store.

Consider an example of creating a shopping cart rule based on any address attribute like shipping address, shipping city etc.

Store Credit

This is Magento Enterprise Edition feature and not available in community edition, however there are several custom extensions in the market like ewallet, store credit etc.

Store credit is an amount that is restored to a customer account. Customers can use their store credit to pay for purchases, and administrators can use store credit for cash refunds. Gift card balances can be credited to the customer's account, instead of using the gift card code for future purchases.

After an order is paid and invoiced, all of the order, or a portion of it, can be refunded by issuing a credit memo. A credit memo differs from a refund because the amount of the credit is restored to the customer's account where it can be used for future purchases. In some cases, a refund can be given at the same time that a credit memo is issued, and applied to the customer's balance of store credit. The amount of store credit that is available in the customer's account is specified in the configuration.

Reference Link

https://docs.magento.com/m2/ee/user_guide/sales/store-credit-using.html

Rewards

Rewards refers to give some points to customer on certain events like

- Registration
- Newsletter signup
- First order
- Converting invitation to
    - Customer
    - Order
- Review Submission and so on

This is again an enterprise edition feature which is not available in community edition.

Reference Link

https://docs.magento.com/m2/ee/user_guide/configuration/customers/reward-points.html

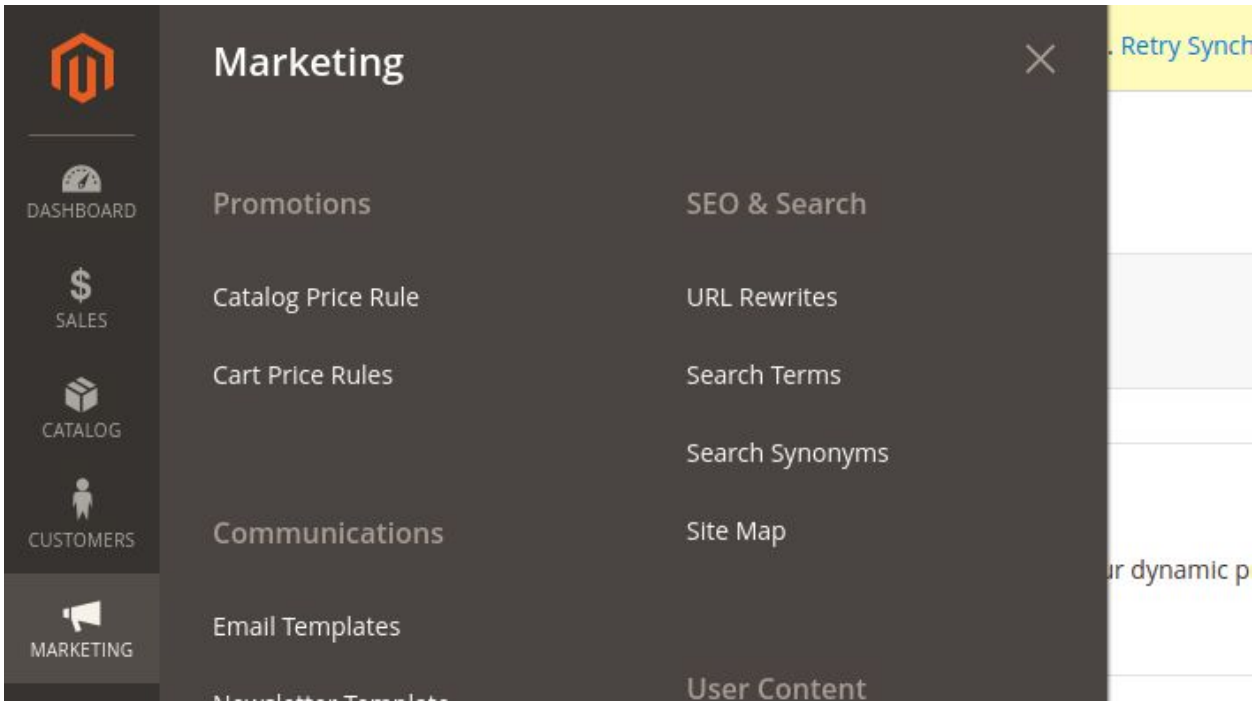
Marketing

Marketing is a group of functionalities associated with either price rules marketing or seo marketing.

It is basically having following subtypes to know about.

- Catalog Rules
- Cart Rules
- Related Product Rules
- Gift Card Account (Enterprise edition)
- Newsletters
- Emails
- URL rewrites

Catalog Rules

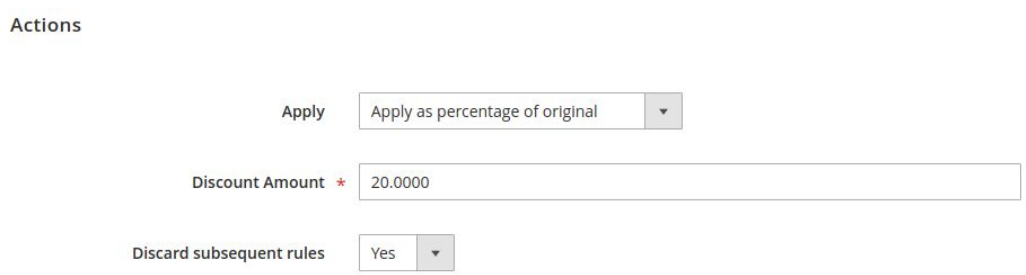


A Catalog Price Rule is created to change a product price before adding it to the shopping cart that is visible on product listing and details page based on several conditions. You can put website conditions, customer group conditions and product attribute conditions as well.

Just go through the Marketing menu and create one Catalog Price Rule for all customer groups and website. Reindexing is assential after making changes over here.

Then see the reflection on frontend how the price change is being displayed.

The important options here to choose is shown in below image.



The apply condition has 4 options to choose from.

1. **Apply as percentage of original**

   If you choose this, this will apply percentage discount of product price.

   For example, if a product price is 200$ and you keep amount to be 20 and select this option, product price will be shown as 160$ on product detail page

2. **Apply as fixed amount**

   If you choose this, this will apply fixed discount of product price.

   For example, if a product price is 200$ and you keep amount to be 20 and select this option, product price will be shown as 180$ on product detail page

3. **Adjust final price to this percentage**

   For example, if a product price is 200$ and you keep amount to be 20 and select this option, product price will be shown as 40$ on product detail page

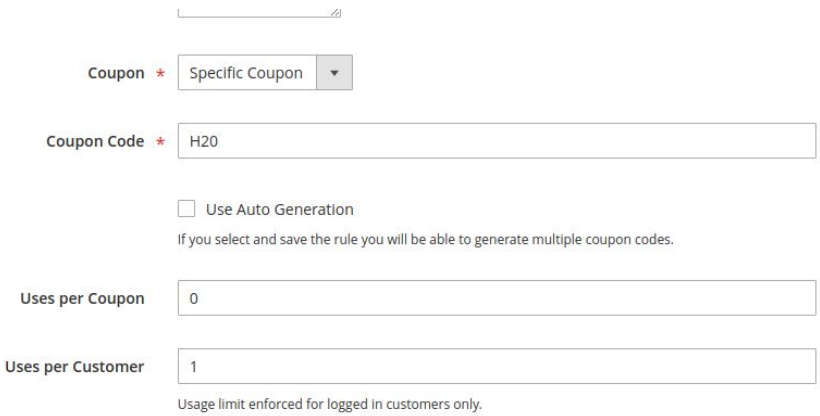
4. **Adjust final price to discount value**

   For example, if a product price is 200$ and you keep amount to be 20 and select this option, product price will be shown as 20$ on product detail page

Cart Rules

This is similar to catalog price rules. The primary difference is that the cart rules is applicable after catalog price rules. This is not shown on product details page. But the pricing will come to picture after adding a product to cart.

A shopping cart rules divided into 2 major parts.



- Cart rule with coupon code

  In this, you will have a coupon code to apply to textbox shown in shopping cart and checkout page. Applying that coupon, you will be eligible to grab the promotion.

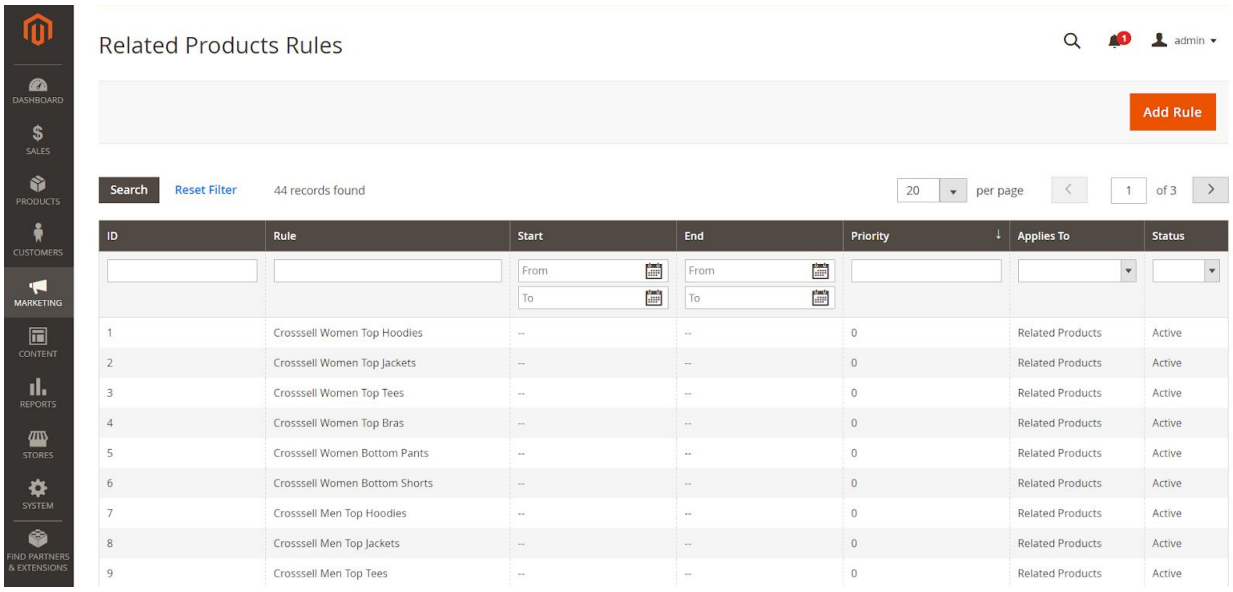
- Cart rule without coupon code

  In this, no coupon code is there. System will apply the discount automatically if conditions you defined are matched.

Related Products Rule

This is a feature offered by Magento Enterprise edition and not available in Magento community edition.

Related product rules give you the ability to target the selection of products that are presented to customers as related products, up-sells, and cross-sells. Each product rule can be associated with a customer segment to produce a dynamic display of targeted merchandising.

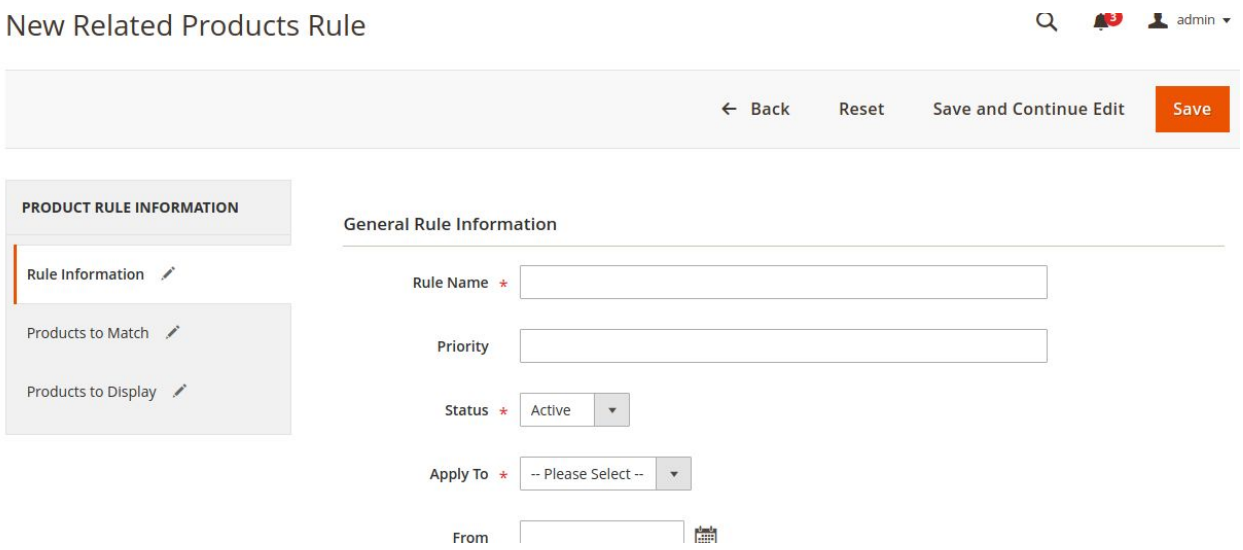


Reference Link

https://docs.magento.com/m2/ee/user_guide/marketing/product-related-rules.html

https://docs.magento.com/m2/ee/user_guide/marketing/product-related-rule-create.html

To create a new related product rule, you just need to enter appropriate metadata as shown in below screen.



You can create it and see the reflection on frontend after clearing the cache.

Gift Card Accounts

This is a feature offered by Magento Enterprise edition and not available in Magento community edition.

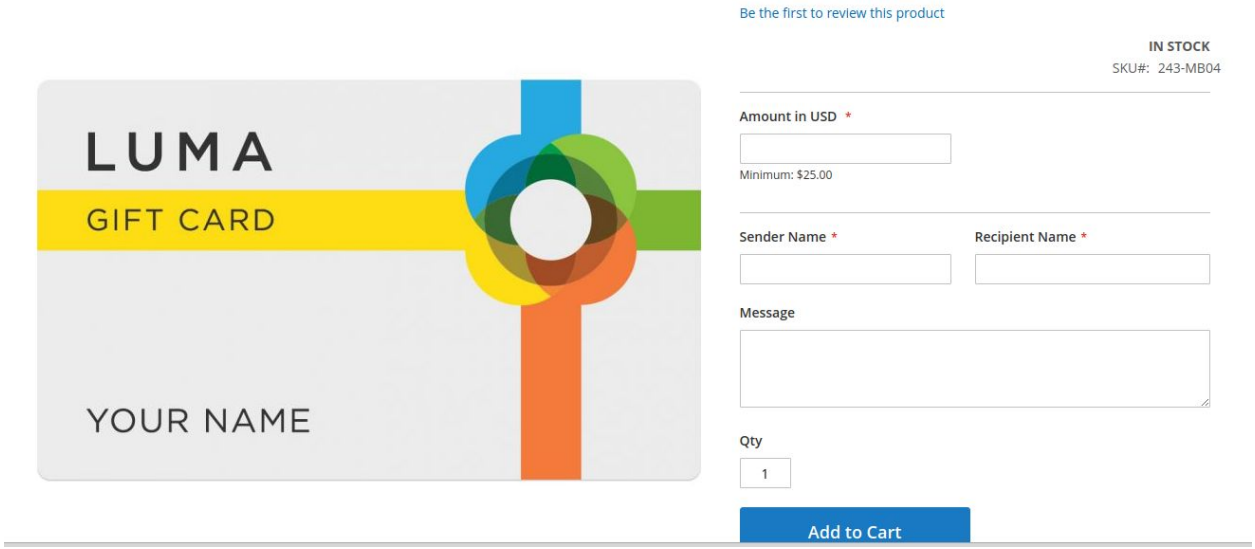
A gift card account is automatically created for each Gift Card that is purchased. The value of the gift card can then be applied toward the purchase of a product in your store. You can also create gift card accounts from the Admin as a promotion or service for customers. The gift card account number corresponds to the gift card code.

To configure a Gift Card account, visit below link.

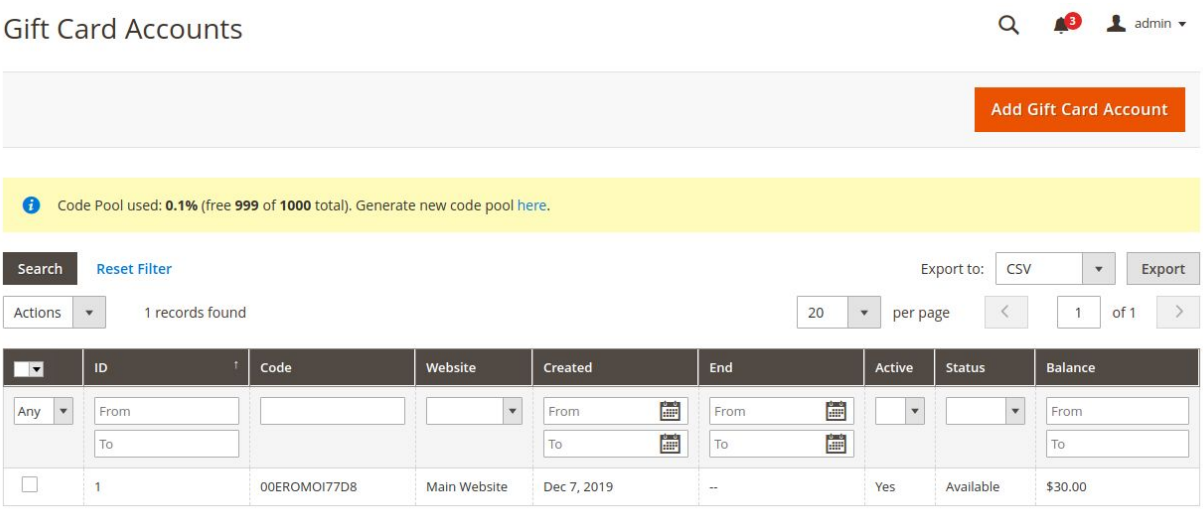
https://docs.magento.com/m2/ee/user_guide/catalog/product-gift-card-account-configuration.html

A Gift card is also a type of product offered by EE of Magento. You can see it with the default data. In gift card type of product, you can enter minimum and maximum value of gift card to offer.

When someone is offering a giftcard to someone, he will be having below screen.



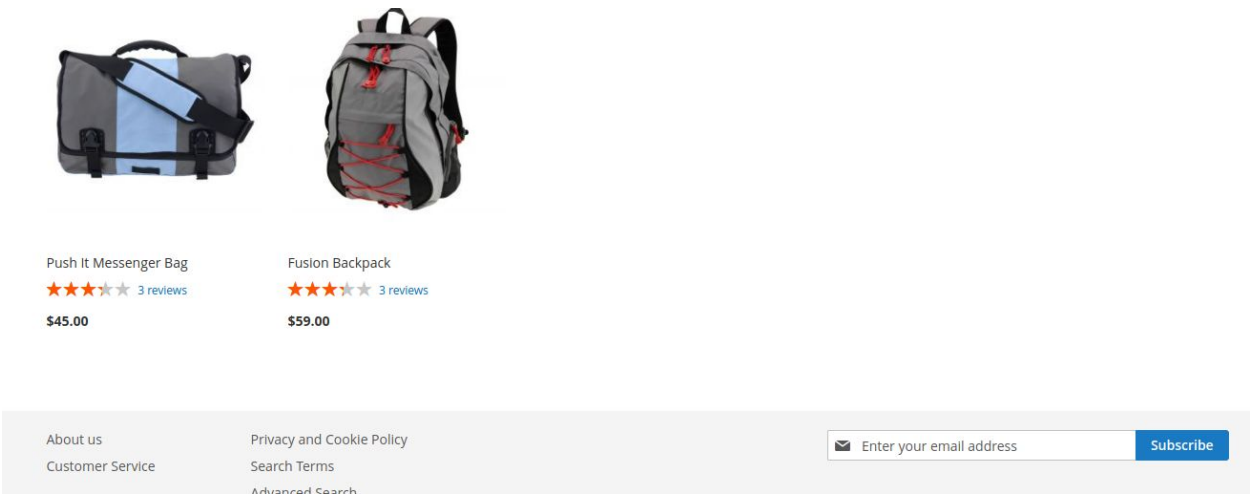
Whenever an order is placed and invoice is paid, you can see the gift card entry in marketing -> manage gift card section.

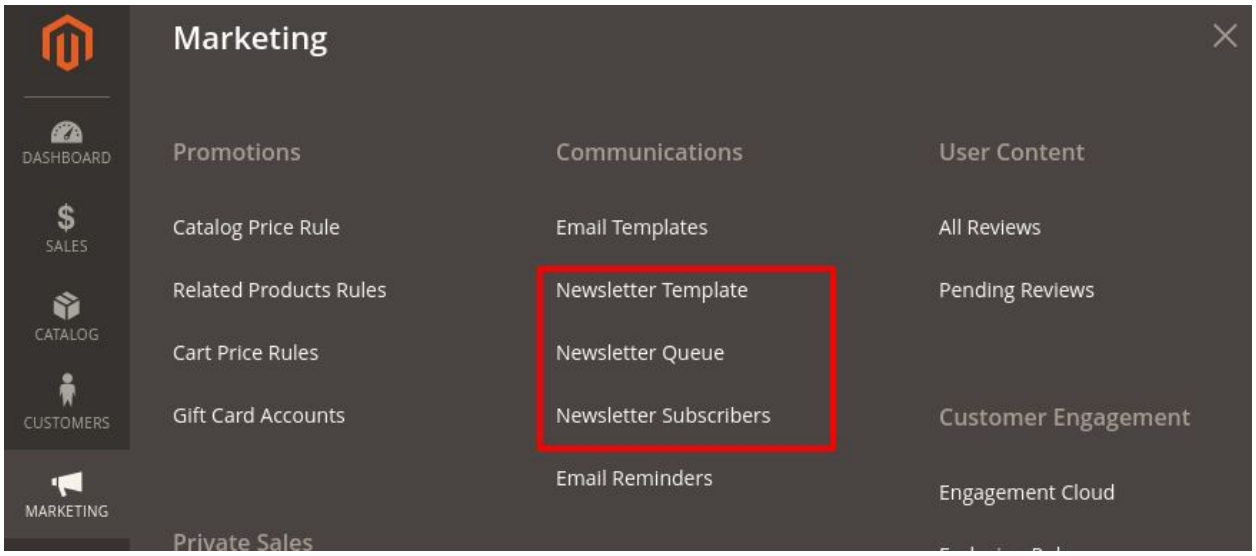


Newsletters

Newsletter functionality are divided basically into 4 parts.

1. Subscribe Email for newsletter (Frontend)
2. Create Newsletter Template (Admin -> Marketing)
3. Send Template to Queue (Admin -> Marketing)
4. Newsletter Subscribers (Admin - > Marketing)





Will explain the in depth functionality via practicals.

Email Reminders

This is also an Enterprise edition feature.

The purpose of an email reminder is encourage people who have visited your store to take advantage of a promotion and make a purchase. Email reminders can be automatically sent to customers when a specific set of conditions is met. For example, you might send a reminder to customers who have added something to their cart or wishlist, but have not yet made a purchase. You can use email reminders to encourage customers to return to your store, and include a coupon code as an incentive. Coupon codes can be automatically generated for each batch of email reminders, to give you control over the offers that are associated with each batch.

Email reminders can be triggered after a certain number of days have passed since a cart was abandoned, or for any other condition you want to define, such as total cart value, quantity, items in the cart, and so on.

Reference Link

https://docs.magento.com/m2/ee/user_guide/marketing/email-reminder-rules.html

https://docs.magento.com/m2/ee/user_guide/marketing/email-reminder-rules-create.html

URL Rewrites

This is offered by Magento community edition.

We have already covered this in routing section.

The URL Rewrite tool lets you change any URL that is associated with a product, category, or CMS page. When the rewrite goes into effect, any links that point to the previous URL are redirected to the new address.

The terms rewrite and redirect are often used interchangeably, but refer to slightly different processes. A URL rewrite changes the way a URL appears in the browser. A URL redirect updates the URL that is stored on the server. A URL redirect can be either temporary or permanent. Your store uses URL rewrites and redirects to make it easy for you to change the URL key of a product, category, or page and preserve existing links.

By default, automatic URL redirects are enabled for your store and the Create Permanent Redirect for old URL checkbox is selected under the URL key field of each product.

Reference Link

https://docs.magento.com/m2/ce/user_guide/marketing/url-rewrite.html

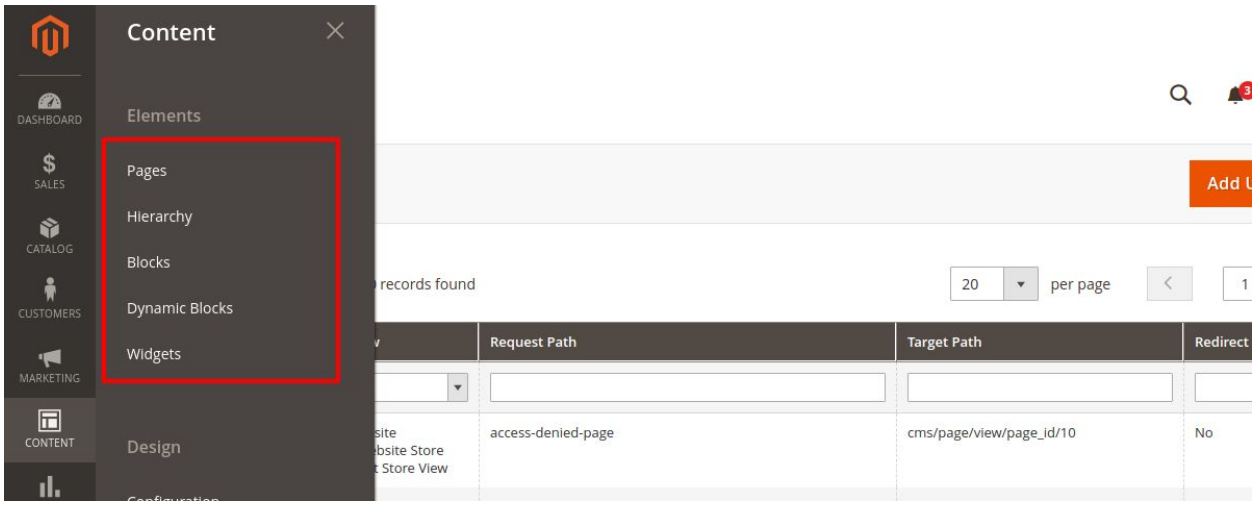
CMS

As we all are familiar with CMS, which stands for **C**ontent **M**anagement **S**ystem. It is further divided into following parts.

1. Pages
2. Static Blocks
3. Widgets

Pages

Magento offers functionality to create as many pages as you want through the admin panel.



Some sample CMS pages are

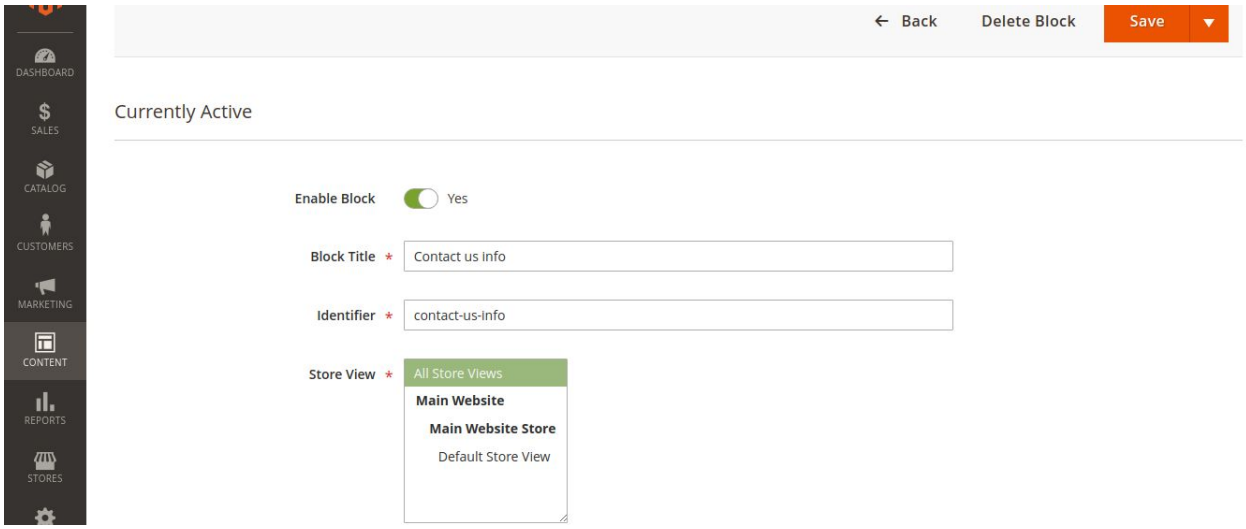
- About Us
- Contact Us
- Privacy Policy
- Terms and Conditions

You can create a CMS page with any SEO friendly URL. Create one yourself to explore more.

Static Blocks

The purpose of static blocks is that it is a bunch of information to be displayed anywhere. For example, you can create a static block and show it to different CMS pages, category pages, product detail pages based on your requirement.

Any static block will have an identifier so that it is easy to render that in any of the above mentioned critaria.



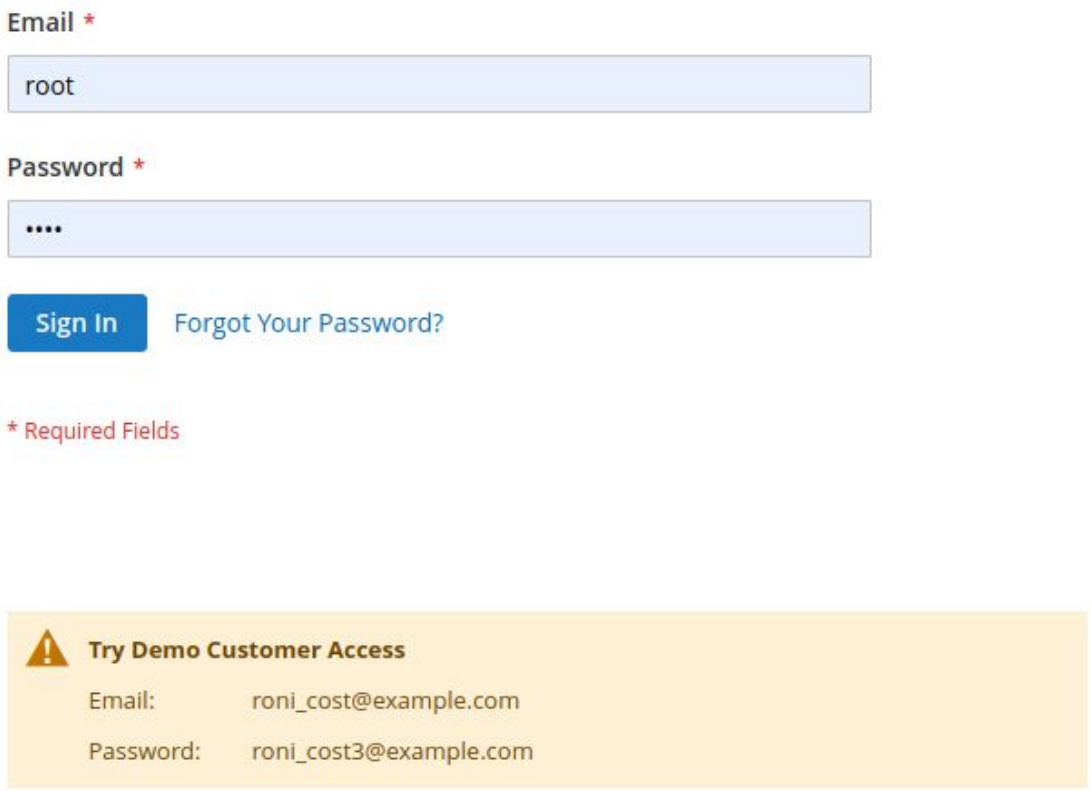
Some of the examples of static blocks are creating it to render contact information on different pages, some size chart to be included on similar kind of products etc.

Widgets

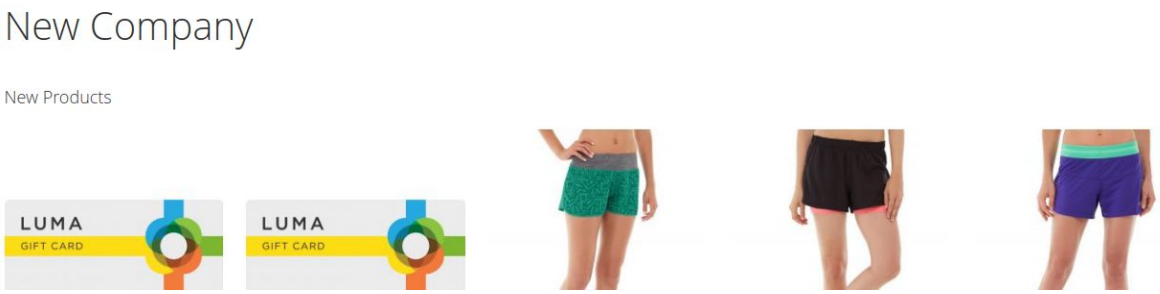
Widgets provide powerful features in Magento 2, that are used to add dynamic or static content to store's pages. Here are the widgets that are available by default:

- CMS Page Link
- CMS Static Block
- Catalog Category Link
- Catalog New Products List
- Catalog Product Link
- Catalog Products List
- Orders and Returns
- Recently Compared Products
- Recently Viewed Products

The best example of a widget you can see is demo login information on login page.



A sample product list widget is as follows.



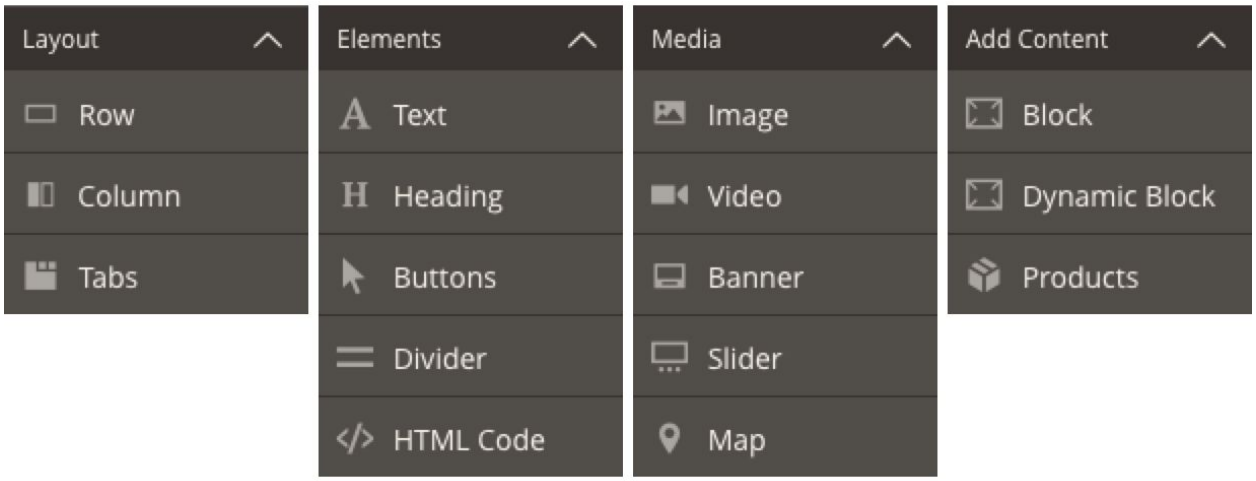
You can create a new product list widget on all or specific CMS pages to be shown to discover more.

We can also create custom widget using custom module that has steps to follow. But this depends on client requirement and customization.

Page Builder

A Page Builder is a new way of creating a page offered by Magento Enterprise Edition.

Page Builder is a Magento extension for creating content by dragging-and-dropping pre-built controls. We call these controls "content types." The available content types are shown in Page Builder's menu:



These content types provide several key features, including:

- **Drag-and-drop** functionality for content creation.
- **Live previews** of how the content will look on the storefront.
- **Form editors** for entering and customizing the content.

For end-users, this means no coding required. For you as a developer, it means you will use content types to customize Page Builder in order to meet the end-user's needs. There are two ways to customize Page Builder using content types:

- Extend existing content types
- Create new content types

Reference Link

https://devdocs.magento.com/page-builder/docs/

This is used when creating any description field for CMS Page, Block, Category, Product etc.

Themes

A theme is a component of Magento application which provides a consistent look and feel (visual design) for entire application area (for example, storefront or Magento admin) using a combination of custom templates, layouts, styles or images.

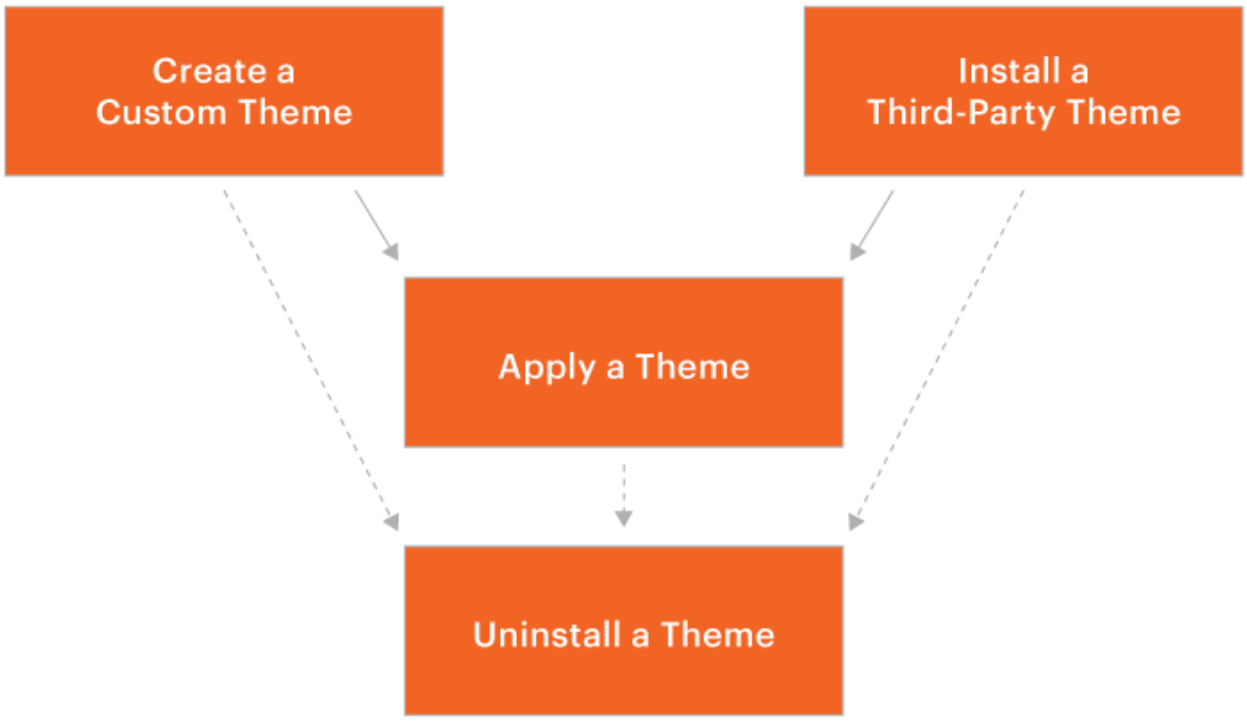
Themes are designed to override or customize view layer resources, provided initially by modules or libraries.

Themes are implemented by different vendors (frontend developers) and intended to be distributed as additional packages for Magento system similar to other components.

Out-of-the-box Magento application provides two design themes: Luma, as a demonstration theme, and Blank as a basis for custom theme creation.

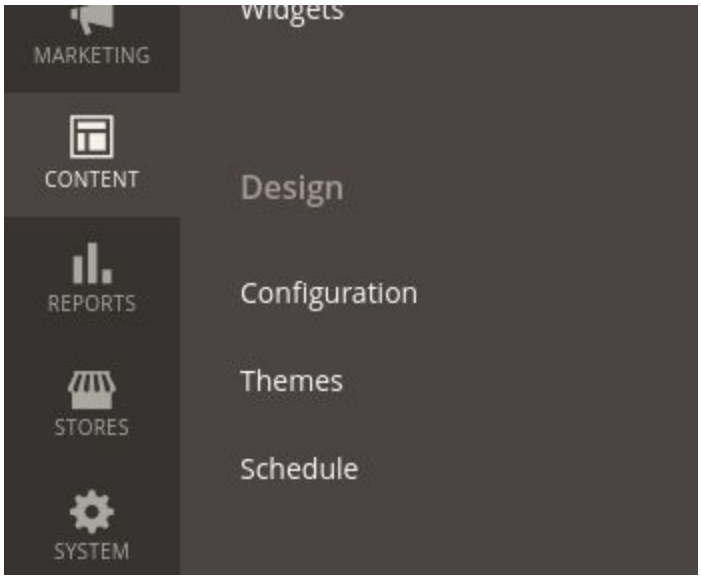
There are no restrictions on using the demonstration Luma theme for a live store, but if you want to customize the default design, you need to create a new theme. We strongly recommend not to change the default Luma and Blank theme files, because if you do edit the default files, your changes can be overwritten by the new version of the default files during upgrades.

Your new theme can be a standalone new theme, or it can inherit from the default or any other existing one. The theme inheritance concept implemented in the Magento system allows you to change only certain theme files, and inherit other required files from a parent theme.



To revise, after installing a theme, You can instantly apply it or you can apply some festival theme during some festival duration and can configured earlier as well.

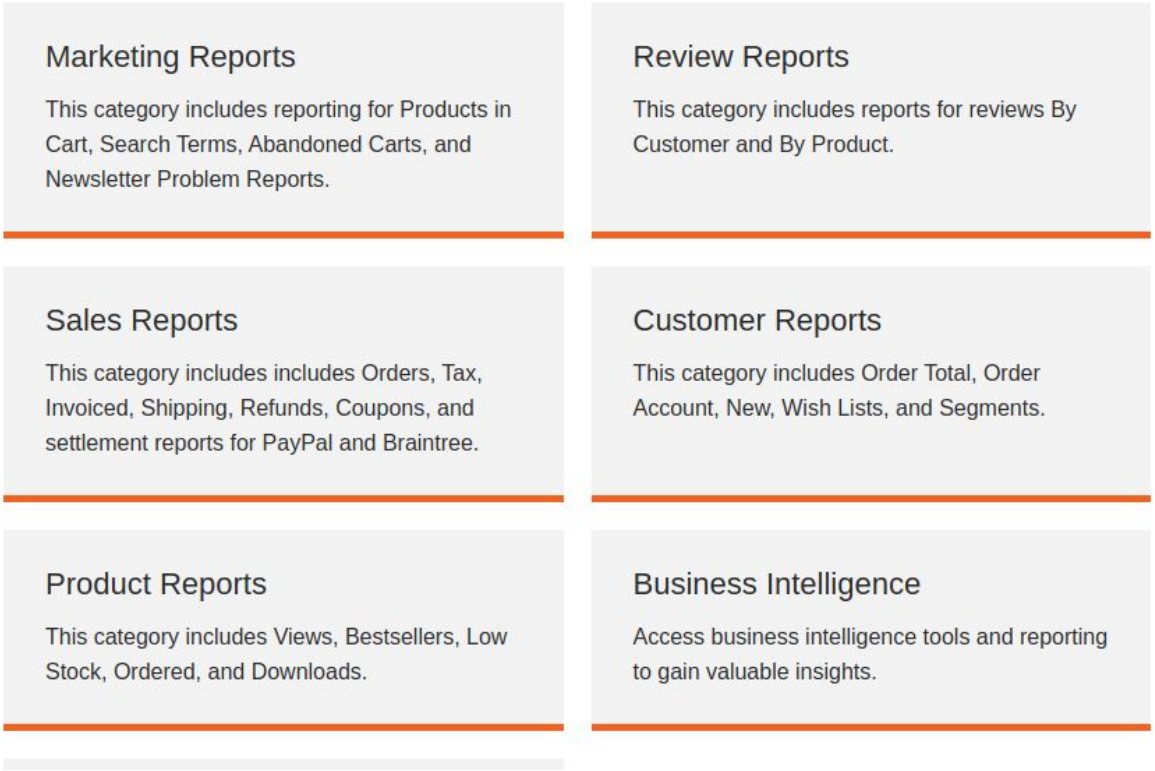
This can be seen using



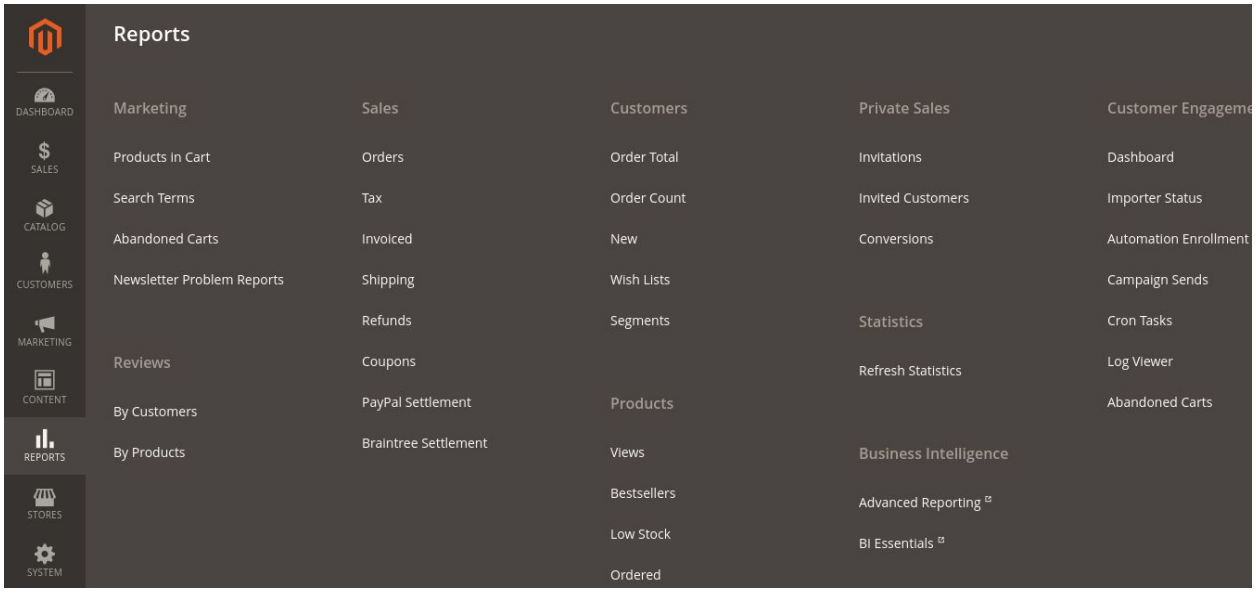
You can explore all three menus.

Reports

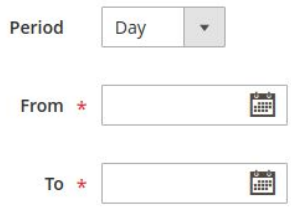
Learn how to filter data and online generate reports. The report data can be opened in a spreadsheet or imported into other applications.



Magento Offers various types of reports. There is altogether a different menu item to look for reports.



There are primary three fields important every report will must have.



Period

Perios offers to select from Day, Month or Year. So that your data will be grouped by selected parameter.

So you can have a number of orders created for each day/month or year.

The addon here we should know is they offer empty rows as well in report so that you can know which day/month or year you do not have business in case of seeing sales order report.

Some of the important initial reports you should explore are

Sales Order

You can check for your selected duration, how many orders are being created/updated with any specified order statuses.

Sales Shipping

You can check for your selected duration, how many shippings are being created/updated with any specified order statuses.

Customer Order Total

This shows number of order user has placed for your duration.

Customer Order Count

This shows number of order along with average and total amount for your selected duration.

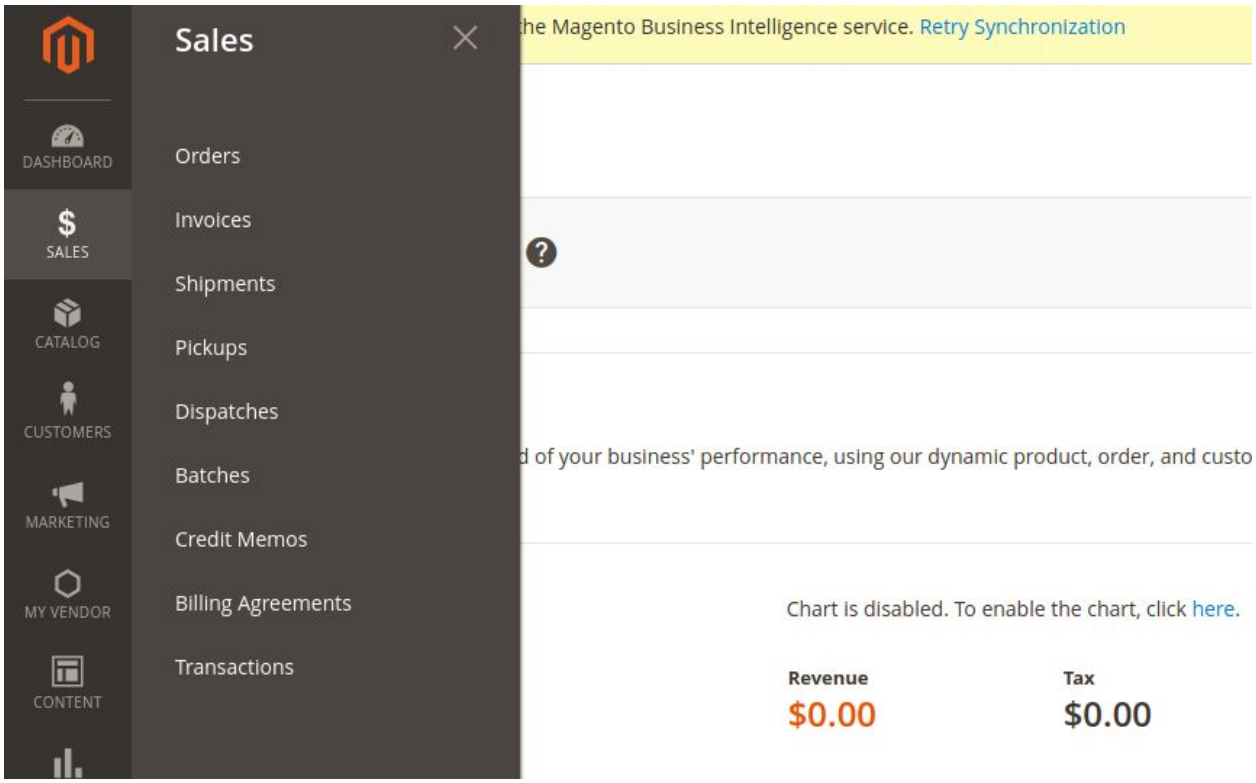
Product Views

Number of time a product is being visited by customers for selected duration

Product Best Sellers

In a tenure of last 6 months what products are considered best seller.

Sales



Sales is a group of sales related items like

- Orders
- Invoices
- Shipments
- Returns
- Credit Memo

Orders

An order can be placed from frontend as well as from the admin panel can be seen here.

Invoices

When someone make a payment, magento created invoice, All the invoices can be found here

Shipments

Same as Invoices, all shipments can be found here

Returns

For any reason, customer claims to return a product, those entries will be found here

Credit Memo

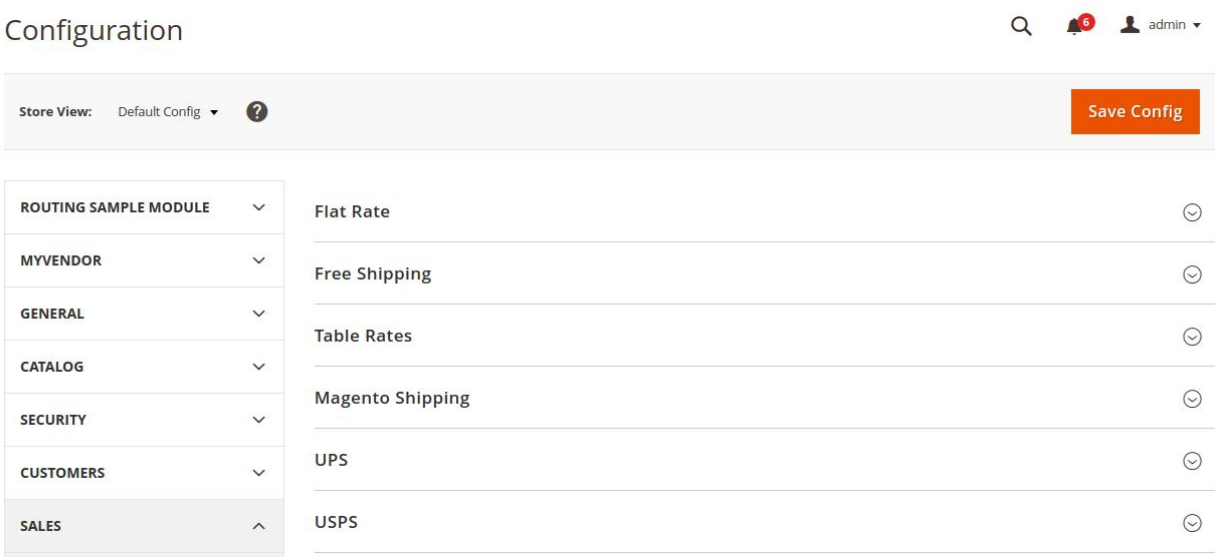
For any reason, customer wants a refund, can be found here.

Shipping Methods

To place an order with any physical product to be delivered, a shipping method is required to choose by a customer. Different shipping method can have different pricing. An ideal shipping method works on weight of a product to be delivered.

You can configure available shipping method from

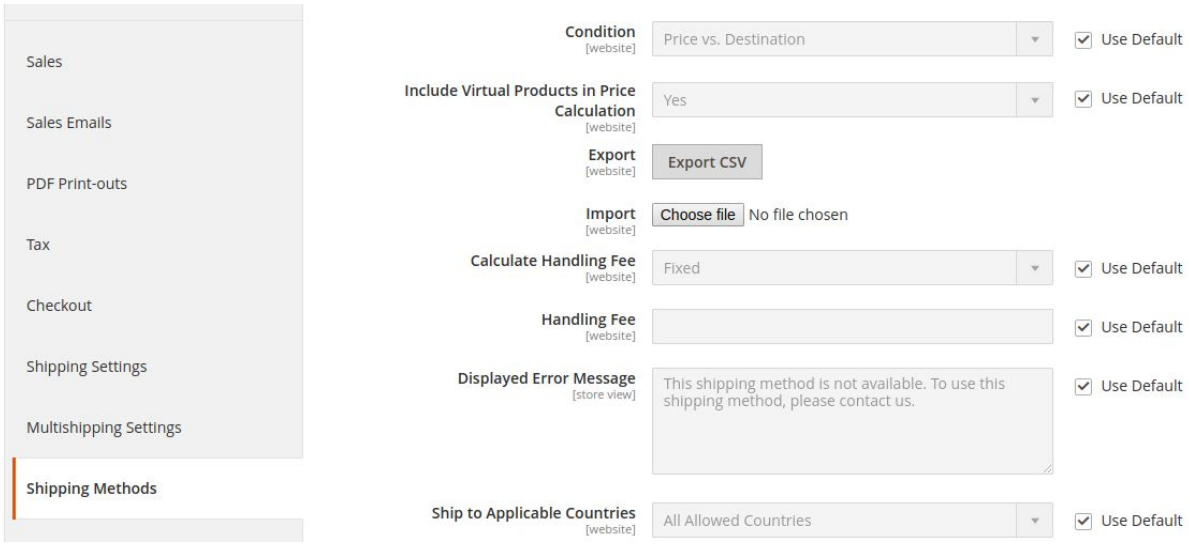
Store -> Configuration -> Sales -> Shipping Methods



You can offer Free Shipping, Flat Rate Shipping where shipping price is defined for entire cart, as well as per item or table rates.

A table rate is something that will be applied by origin and shipping address calculation.

For example, consider you have an origin at Delhi and customer address is Mumbai, you can define rate by choosing a storee view, download sample file, edit that file and apply that.



The exported file will look like below

```
Country,Region/State,"Zip/Postal Code","Order Subtotal (and above)","Shipping Price"
USA,*,*,0.0000,15.0000
USA,*,*,50.0000,10.0000
USA,*,*,100.0000,5.0000
USA,AK,*,0.0000,20.0000
USA,AK,*,50.0000,15.0000
USA,AK,*,100.0000,10.0000
USA,HI,*,0.0000,20.0000
USA,HI,*,50.0000,15.0000
USA,HI,*,100.0000,10.0000
```

So you can change this file to configure your rate.

Payment Methods

Payment methods are listed under following screen.

Store -> Configuration -> Sales -> Payment Methods

Magento offers many payment methods by default which are ready to use. You jus need to configure your merchant account and you are all set.

Some of them are as follows.

- Cash On Delivery
- Check / Money Order
- Bank Transfer
- Authorize..net
- Paypal with different varients
- Braintree
- Amazon Pay

You can install any other payment gateway module as well like Payytm, CCavenue iDeal etc as well as you can develop your own payment gateway using module creation process.

Order State and Status

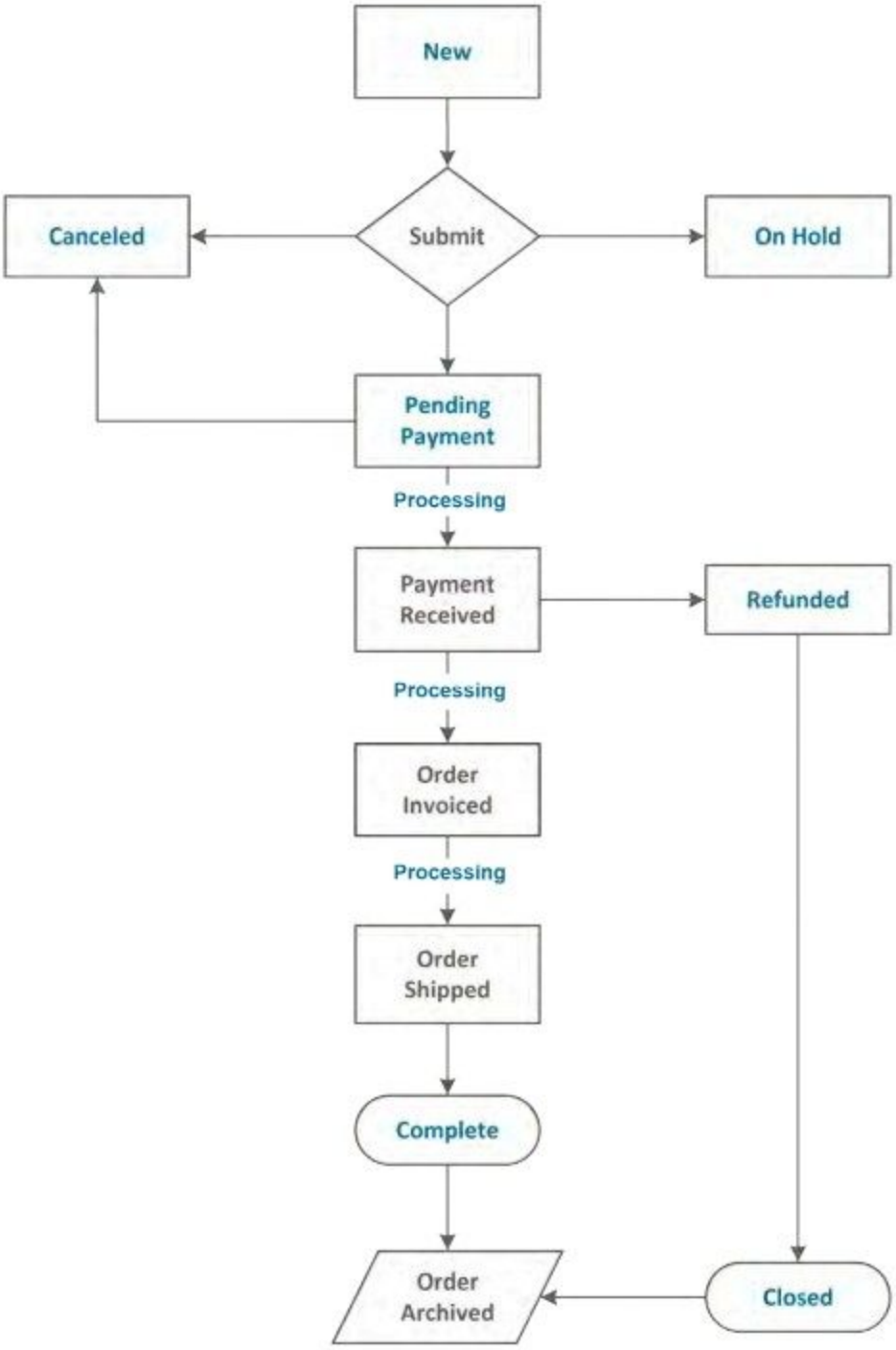
State and Status are 2 different types to understand.

A state is a fixed set of information for an order while a status is something so that one state can have multiple state.

Consider an example, that you have office in Delhi and Order has o be delivered to Mumbai. Now You have shipped your order from delhi. It takes several days to come to mumbai. Also it will reach to mumbai godown and then from godown it will be delivered to your customer. So from Delhi to Mumbai Godown it is still under shipping state but with different status. When it left Delhi, it is having Shipped status with Shipped State. When i reaches to mumbai, it is again Shipped state, with Shipped to local godown as status.

So this way a state and a status is connected to each other.

A standard order processing flow can be understood by below Flow chart.



Magento offers below order states.

- Cancelled

- Closed
- Complete
- Payment Review
- Processing
- On Hold
- Pending
- Pending Payment

To explain status, let's consider an example with Processing State.

Consider an example of you have purchased a tshirt and paid via online payment method. The state of your order is now Processing.

Consider following 2 stages

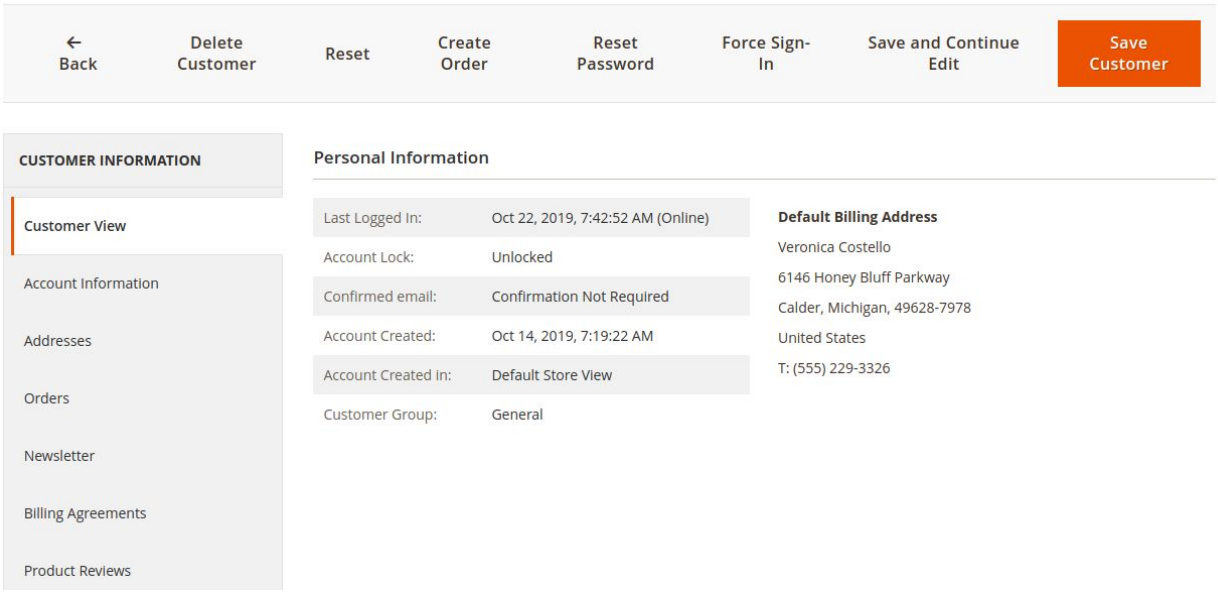
- Gift wrap is done
- Submitted to admin staff to courier

You can create custom statuses in admin panel, as per the business needs so that you can track information correctly.

Describe native shipment functionality in Magento

As mentioned, Magento 2 provides by default many shipping methods. We will configure table rate shipping method for different region. Place multiple orders with multiple addresses and see how the order processing is done.

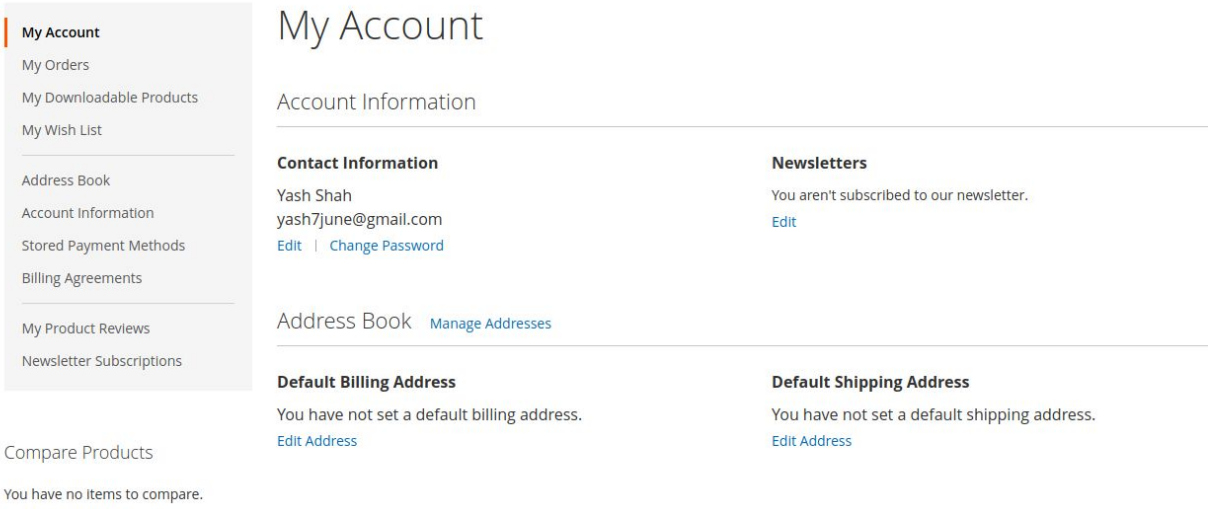
Describe and customize operations available in the customer account area



From the admin panel, You can take following actions.

- Create New Customer
- Edit Customer Information
- See their personal details
- Add/edit addresses
- Revoke Tokens
- Reset Password
- See Orders
- Newlettes
- Billing Agreements
- Product Reviews
- Wishlist

From the store front, a customer can be manage things mentioned in below image.



Just explore the panel to get in touch.

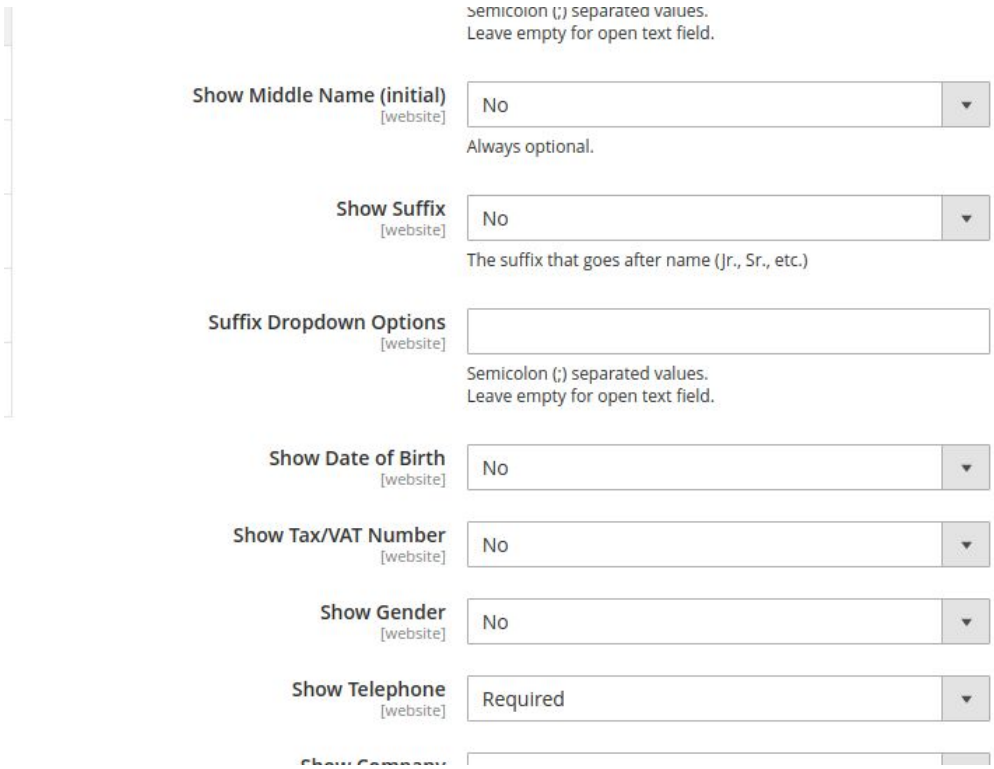
Add or modify customer attributes

You can add certain options like you want to have following information from a customer or not.

- Prefix
- Date of Birth
- Tax/Vat Number
- Suffix
- Gender
- Telephone is required or not
- Company
- Fax

All these you can find at

Store -> Configuration -> Customer -> Customer Configuration -> Name & Address Options



You can play with all and see the reflection on frontend.

You can also create custom customer attributes using extension attribute feature of magento but that is a long way.

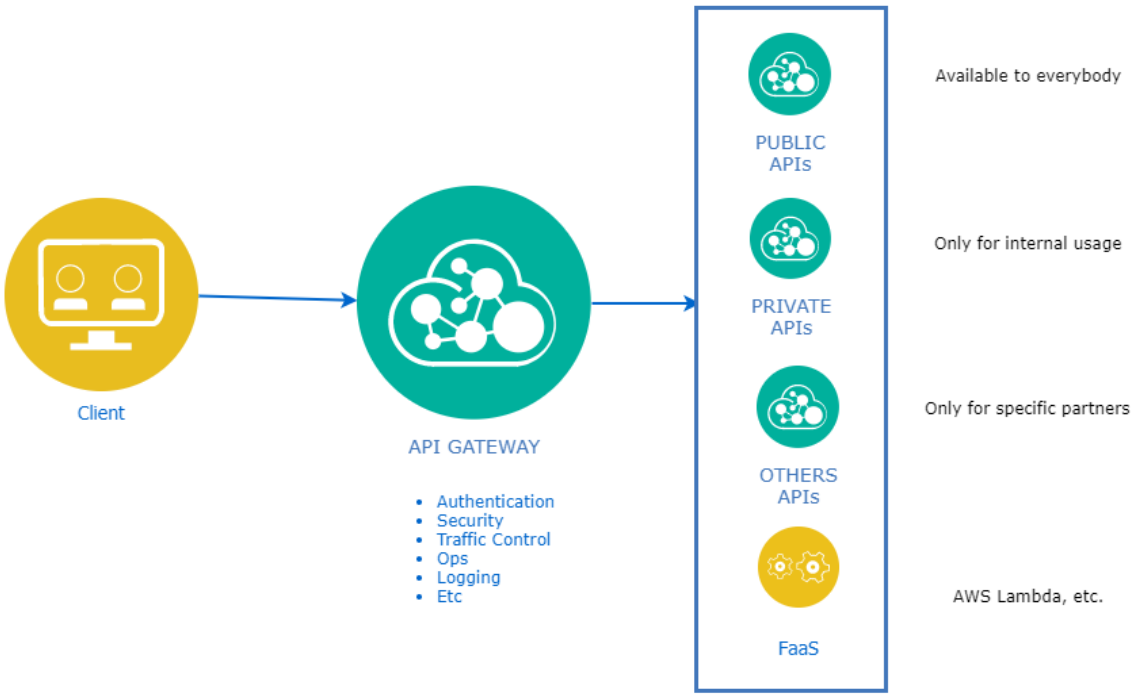
Customize the customer address

You can create your custom template to render address.

API

What is API and why it is used ?

API or an **A**pplication **P**rogramming **I**nterface is an interface or communication protocol between a client and a server intended to simplify the building of client-side software.



Well, in a layman's language, an API is an interface to exchange data between 2 parties. Consider a real life example **BookMyShow** ,**MakeMyTrip or Redbus**.

If we talk about **BookMyShow,** they books ticket for a third party cinema hall. And there are a variety of applications doing the same thing. So **BookMyShow** cannot have control over seats available or not. They call some API service of a particular cinema hall to check for the availability. Same thing applies to **MakeMyTrip** for **Hotel** and **Flight Booking**. And same for **RedBus**.

The end solution provider must provide some sort of interface to these applications so that they can access their data and process them.

There are different types of API protocol and authentication methods. Some are publically available while to access others, we may need to authenticate the request via some Access Token.

API in Magento

Today's market is all about mobiles. People surf mobile much more than desktops/laptops for online shopping.

In terms of Magento, API is needed to give access of Category, Product, Orders, Wishlist, Checkout information so that it can be used while creating a mobile app. Magento uses a standard way called **Service Contract** to fulfil this requirement.

Authentication Methods

Client	Authentication Method Process
Mobile application	Registered users use token-based authentication to make web API calls using a mobile application. The token acts like an electronic key that provides access to the API(s). 1. As a registered Magento user, you request a token from the Magento token service at the endpoint that is defined for your user type. 2. The token service returns a unique authentication token in exchange for a username and password for a Magento account. 3. To prove your identity, specify this token in the `Authorization` request header on web API calls.

Third-party application	Third-party applications use OAuth-based authentication to access the web APIs. 1. The third-party Integration registers with Magento. 2. Merchants authorize extensions and applications to access or update store data.
JavaScript widget on the Magento storefront or Magento Admin	Registered users use session-based authentication to log in to the Magento storefront or Magento Admin. A session is identified by a cookie and time out after a period of inactivity. Additionally, you can have a session as a guest user without logging in. 1. As a customer, you log in to the Magento storefront with your customer credentials. As an administrator, you log in to the Magento Admin with your administrator credentials. 2. The Magento web API framework identifies you and controls access to the requested resource.

Reference Link:

https://devdocs.magento.com/guides/v2.3/get-started/authentication/gs-authentication.html

API Authorization

Reference Link:

https://devdocs.magento.com/guides/v2.3/get-started/authentication/gs-authentication-token.html

Service Contracts

Reference Link:

https://devdocs.magento.com/guides/v2.3/extension-dev-guide/service-contracts/service-contracts.html

https://devdocs.magento.com/guides/v2.3/rest/list.html

API Response Formats

An API can have following response formats.

1. XML response
2. JSON response

3. Plain response

Reference Link:

https://devdocs.magento.com/guides/m1x/api/rest/response_formats.html

REST & SOAP

Rest and Soap both are types of webservices with some differences.

Rest is Representational State Transfer while Soap is Simple Object Access Protocol.

Reference Link:

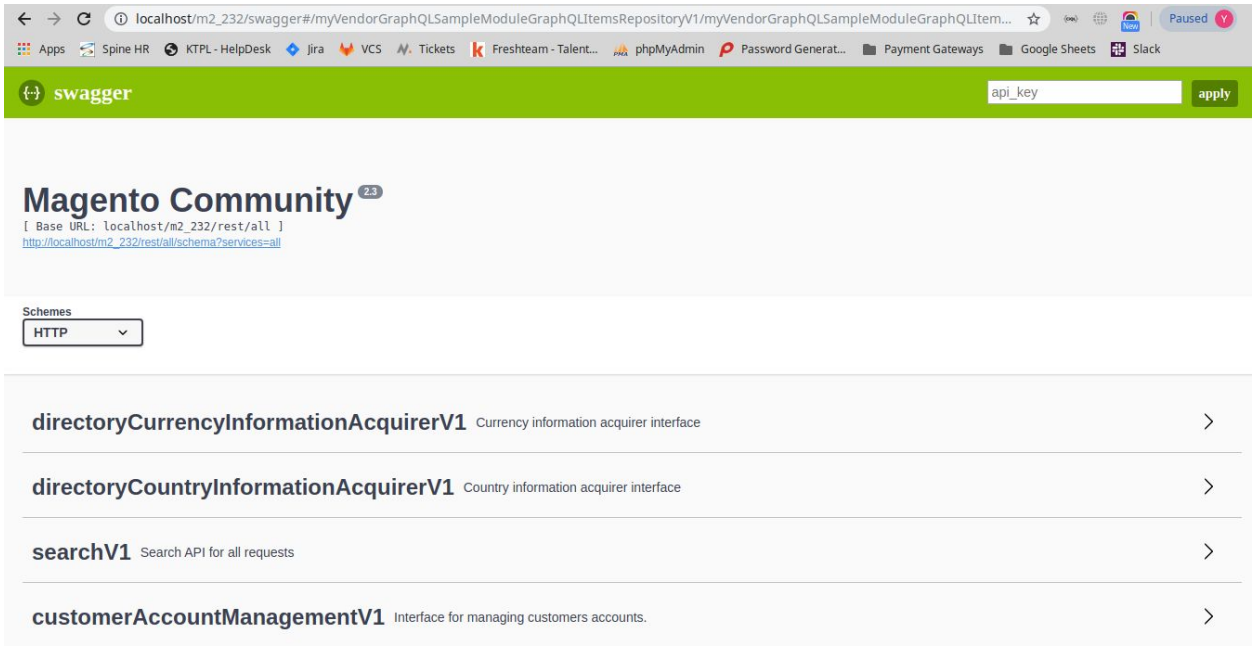
https://dzone.com/articles/difference-between-rest-and-soap-api

In Magento 2, You can see and call all the apis with the use of swagger.

Just hint the following URL in your browser and you will be able to see all available APIs.

http://your-website.com/swagger

It will give you below response.



You can click on any API and check the results.

A standard file structure to create an API is as follows.

```
app
    code
        MyVendor
            MyModule
                Api
                    Data
                        MyModelInterface.php
                            - This is getter and setter class. It contains all publically exposed methods
                        MyModelSearchResultInterface.php
                            - This contains getItems and setItems. Used to retrieve collections based on exposed
                            method from above file
                    MyModelRepositoryInterface.php
                        - This containsmethod listing those are available for API like
                            getById
                            getList
                            deleteById
                        - This is just declaration, definition will be given in target file.
                etc
                    di.xml
                        - This sets preference from Interfaces to real files

                        <preference name="MyVendor\MyModule\Api\MyModelRepositoryInterface"
                        type="MyVendor\MyModule\Model\MyModelRepository" />

                        <preference name="MyVendor\MyModule\Api\Data\MyModelInterface"
                        type="MyVendor\MyModule\Model\MyModel" />

                        <preference name="MyVendor\MyModule\Api\Data\MyModelSearchResultInterface"
                        type="Magento\Framework\Api\SearchResults" />
                    webapi.xml
                        -
                        <route method="GET" url="/V1/yourslug">
                            <service class="MyVendor\MyModule\Api\MyClassRepositoryInterface" method="getList"/>
                            <resources>
                                <resource ref="anonymous"/>
                            </resources>
                        </route>
                Model
                    ResourceModel
                        MyModel
                            Collection.php
                        MyModel.php
                    MyModel.php
                    MyModelRepository.php
```

You can find a sample API module at the below link.

https://github.com/yash7690/magento2-graphql

Assignments:

1. Create a new module that creates table “posts” with following columns.
    a. post_id
    b. title
    c. content

    d. created_at
    e. updated_at
2. Insert 100 records via InstallData.php
3. Create Api Structure as above to get list of posts with search and pagination
4. Set resource from anonymous to oauth based token

GraphQL

GraphQL is an open-source data query and manipulation language for APIs, and a runtime for fulfilling queries with existing data. GraphQL was developed internally by Facebook in 2012 before being publicly released in 2015.

```
{
  user(id: 4802170) {
    id
    name
    isViewerFriend
    profilePicture(size: 50)  {
      uri
      width
      height
    }
    friendConnection(first: 5) {
      totalCount
      friends {
        id
        name
      }
    }
  }
}
```

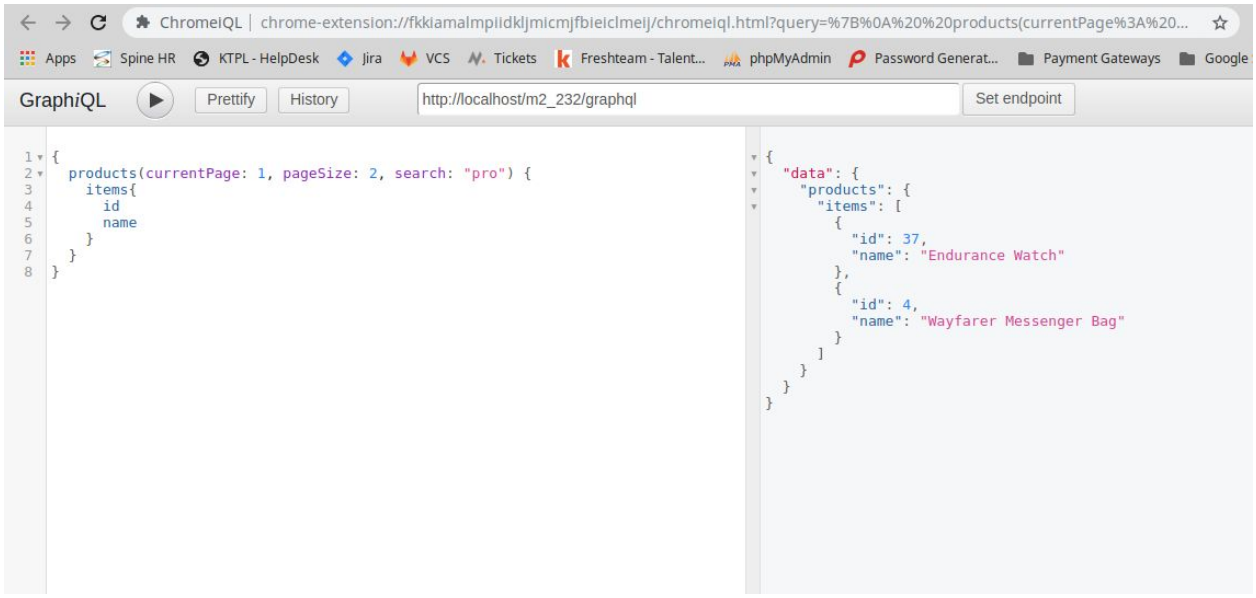
```
{
  "data": {
    "user": {
      "id": "4802170",
      "name": "Lee Byron",
      "isViewerFriend": true,
      "profilePicture": {
        "uri": "cdn://pic/4802170/50",
        "width": 50,
        "height": 50
      },
      "friendConnection": {
        "totalCount": 13,
        "friends": [
          {
            "id": "305249",
            "name": "Stephen Schwink"
          },
          {
            "id": "3108935",
            "name": "Nathaniel Roman"
```

GraphQL support was added in Magento 2 with 2.3 version.

GraphiQL is an in-browser tool for writing, validating, and testing GraphQL queries. You can download the extension from your browser's app store. For the Google Chrome browser, the ChromeiQL extension will do the job.

https://chrome.google.com/webstore/detail/chromeiql/fkkiamalmpiidkljmicmjfbieiclmeij?hl=en

Here is the basic example of Magento 2 GraphQL. This example get product listing based on provided criteria. The most different thing here is , it will only output those information which is asked in the request. No more no less.



All you need to do here is, setup endpoints in the textbox shown in image.

The endpoint would be your http://your-domain.com/graphql

Now you can enter your query in left side panel which will be an auto complete tool and the output will fall in to right side panel.

File structure to create graphql would be as follows.

```
app
    code
        MyVendor
            MyModule
                etc
                    schema.graphqls
                        type Query {
                            namespace (
                                inputParam1: Int = 1
                                inputParam2: String
                            ): OutputType
                            @resolver: (class: "MyVendor\MyModule\Model\Resolver\MyClass")
                        }
                        type OutputType
                        @doc (description: "Description") {
                            outputParam1: String @doc (description: "description")
                            outputParam2: String @doc (description: "description")
                        }
                Model
                    Resolver
                        MyClass.php
                            - public function resolve()
```

You can find a sample graphql module at the below link.

https://github.com/yash7690/magento2-graphql

Reference Link:

https://devdocs.magento.com/guides/v2.3/graphql/

Development Best Practices

Highlights

- Use Of Indentation
- Code Comments
- Proper function names
- Use of camelCase and Underscore
- No of lines should not exceed more than 20

PHP

- Indent with should consist of 4 spaces.
- Maximum length of any line in PHP is 120 characters.
- When defining filename it is not recommended to use numbers. For eg. (File1.php, File2.php). You can use underscore (_) or dash character when needed (-). Spaces are strictly prohibited.
- Function names should be defined by letters only. Numbers are permitted in function names but are discouraged in most cases.
- Function names must always start with a lowercase letter. When a function name consists of more than one word, the first letter of each new word must be capitalized. This is commonly called "camelCase" formatting.
- Names must fully describe their purpose and behavior. I.E. getById() or getBySku()
- Method length should not be more than 20 lines. You should try to break methods in smaller codes which will be more readable.
- Avoid the below practise in your code
    - Raw SQL queries
    - SQL queries inside a loop
    - Direct class instantiation
    - Unnecessary collection loading
    - Excessive code complexity
    - Use of PHP superglobals

Magento Standards

- You must strictly follow the module folder structure to place your files.
- You must create your modular structure following service contract structure.

- Magento admin forms and grid must be created using ui components.
- 

  Never use **ObjectManager**. Using it decreases ability for third parties to customize your code.

- 

  Single Responsibility.

All Classes should have only Single Responsibility which is entirely encapsulated

Mixing different behaviors in one class (e. g. classic Helper) greatly decreases extensibility and increases coupling in most of the cases

Allows for easy replacement of any specific behavior by providing a single point for change.

- Apply code comments using following way.
    - Your methods should always have the appropriate doc blocks. **@param**, **@returns** , and **@throws** should be automatically used.

      This makes it easier to know what your method needs and what it returns. Don't make other developers guess.

    - Inline comments should only be used if they are necessary. Things such as deep method recursion or highly abstracted calls can use a quick comment to show what the intention is.

- Class members visibility
    - All non-public properties and methods should be private.

```
    public $bar;
    protected $foo;
    private $baz;

    public function foo() {}
    protected function bar() {}
    private function baz() {}
```

    s
- Follow the below principles
    - Never Edit the Core
    - Rewrite Classes the Proper Way
    - Rewrite Only What Needs Changed
    - Create files in the Right Places
- Magento EQP should be performed after the development has been completed. It is compulsory to clear the level 1 of EQP. The module will/should not be live in magedelight till Level 1 clearance.
  Magento2 EQP

Before Going Live

- Magento EQP should be clear
- Inline or unwanted comment should be removed
- Debug code should not exist in the final package. For example echo, console.log()

Useful Links

- **Composer**

  https://devdocs.magento.com/guides/v2.3/extension-dev-guide/build/composer-integration.html

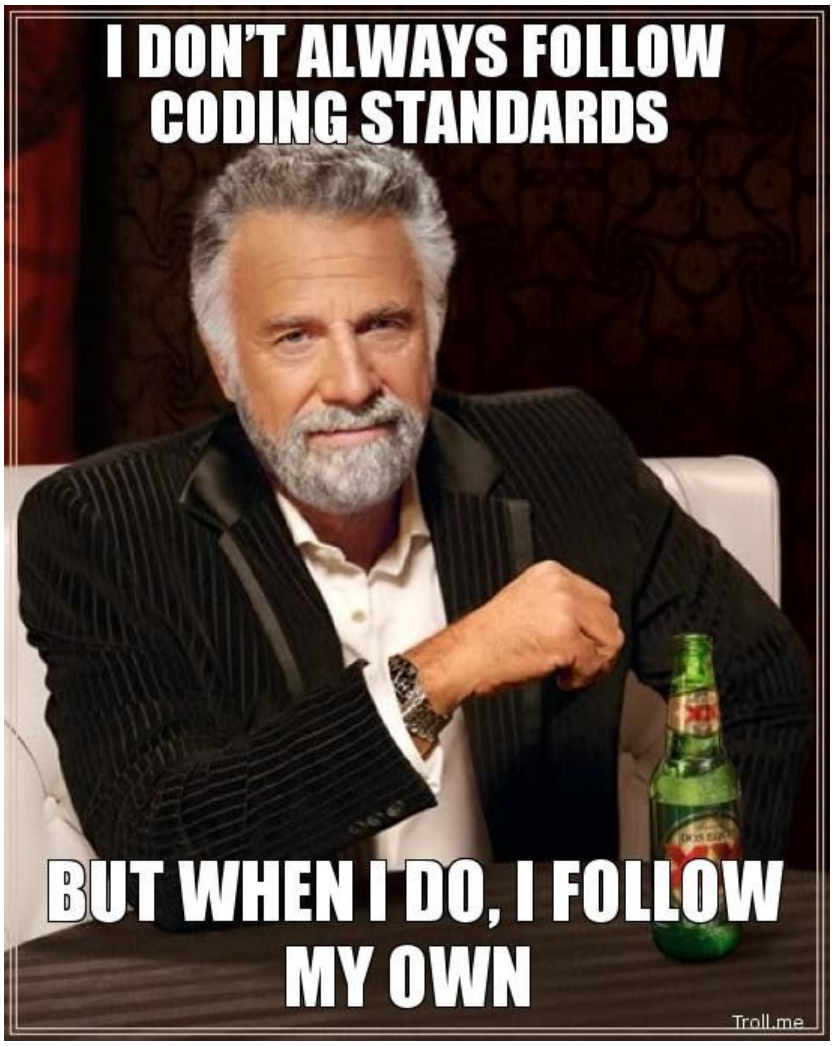
- **Service Contract**

  https://devdocs.magento.com/guides/v2.3/extension-dev-guide/service-contracts/service-contracts.html

- **Magento 2 Technical Guidelines**

  https://devdocs.magento.com/guides/v2.2/coding-standards/technical-guidelines.html

Last but not the least,


I DON'T ALWAYS FOLLOW CODING STANDARDS
BUT WHEN I DO, I FOLLOW MY OWN
Troll.me