FREE eBook

LEARNING magento2

Free unaffiliated eBook created from
Stack Overflow contributors.

#magento2

Table of Contents

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: magento2

It is an unofficial and free magento2 ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official magento2.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with magento2

Remarks

Magento 2 is an open-source e-commerce platform designed to facilitate the common shopping cart structure for webpages. Compared to earlier versions of Magento, the 2.0 version is more streamlined and performant - eliminating problems with table locking and improving on the checkout system for guest users.

Versions

Version	Release Date
2.1.7	2017-05-31
2.1.6	2017-04-11
2.1.5	2017-02-21
2.1.4	2017-02-07
2.1.3	2016-12-14
2.1.2	2016-10-10
2.1.1	2016-08-25
2.1.0	2016-06-23
2.0.14	2017-05-31
2.0.13	2017-02-21
2.0.12	2017-02-07
2.0.11	2016-10-12
2.0.10	2016-10-07
2.0.9	2016-08-04
2.0.8	2016-07-18
2.0.7	2016-05-19
2.0.6	2016-05-13
2.0.5	2016-04-27

Version	Release Date
2.0.4	2016-03-31
2.0.3	2016-03-30
2.0.2	2016-01-28
2.0.1	2016-01-19
2.0.0	2015-11-17

Examples

Installation or Setup

Detailed instructions on getting magento2 set up or installed.

Install Magento 2 on Ubuntu 16.04

NOTES: We are going to install Magento 2 on fresh Ubuntu Server 16.04 LTS with PHP 7.0, MySQL 5.6 and Apache 2.4.

1. Setup Requirements

- Apache 2.2 or 2.4 with mod_rewrite module (or) Nginx >= 1.8.
- PHP 5.5 or later version. PHP 7.0 also supported.
- Required PHP-Modules – PDO/MySQL, mbstring, mcrypt, mhash, SimpleXML, curl, xsl, gd, ImageMagick 6.3.7 (or later) or both, soap, intl, openssl.
- Composer and Git.

You can use the following command to install all of above requirements from default repository (xenial).

```
sudo apt install apache2 git mysql-server
sudo apt install php libapache2-mod-php php-mysql php-dom php-simplexml php-gd
sudo apt install php-curl php-intl php-xsl php-mbstring php-zip php-xml php-mcrypt
```

I recommend to install from homepage instead of Ubuntu repository.

```
curl -sS https://getcomposer.org/installer | php
mv composer.phar /usr/local/bin/composer
chmod +x /usr/local/bin/composer
```

2. Setup Magento 2

a) Download from GitHub

Magento2 code is available under Github repository. Use following command to clone Magento2 repository on your system.

```
cd /var/www/
git clone https://github.com/magento/magento2.git
```

b) Download via Composer

If you don't want to install Magento 2 by cloning from GitHub, it's fine. You can also install it through Composer.

```
cd /var/www
composer create-project --repository-url=https://repo.magento.com/ magento/project-community-
edition magento2
```

Now install all required modules for Magento2 using composer. Wait for the installation process completed. (You won't need this if you are installing Magento 2 via Composer)

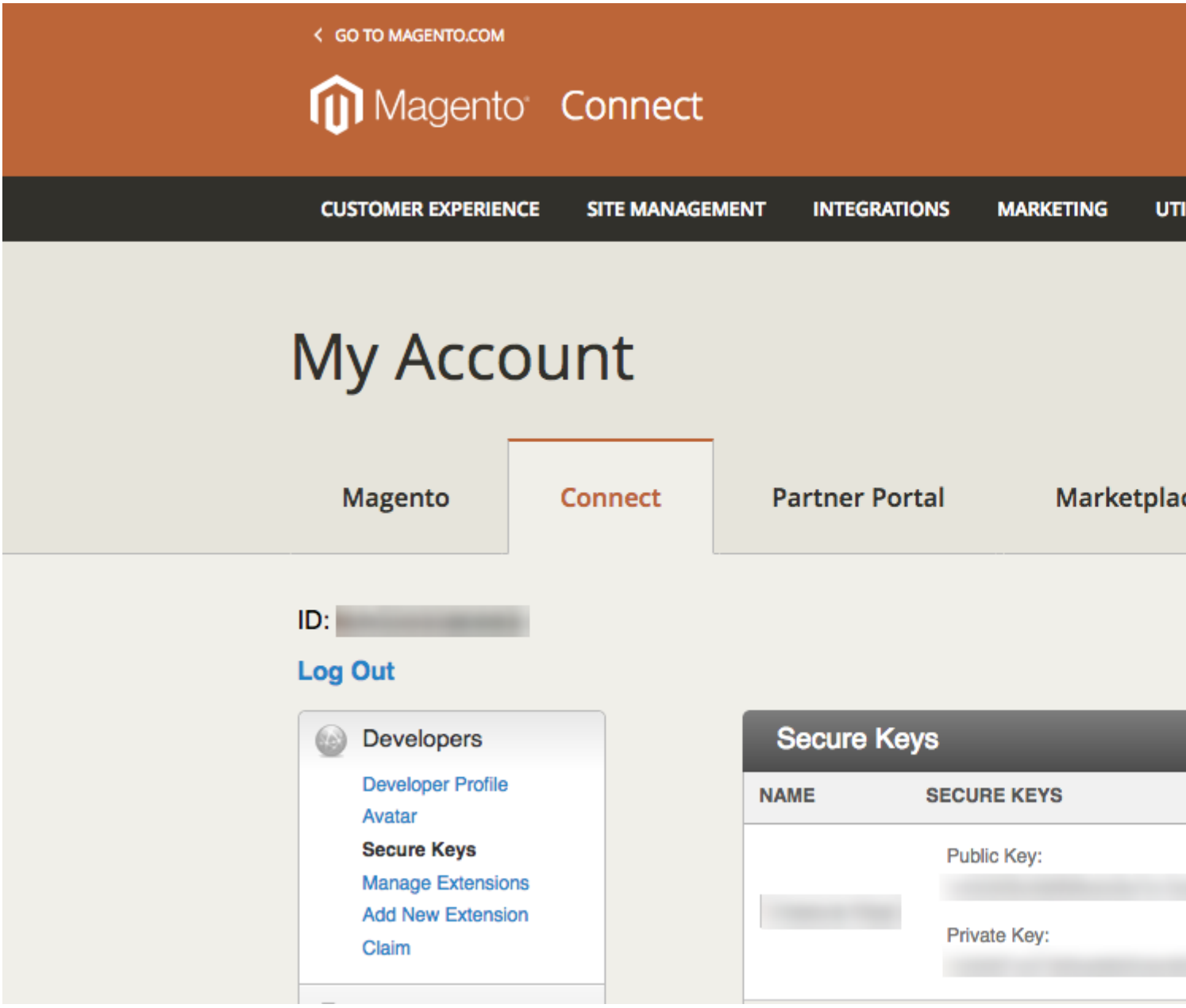
```
cd magento2/
composer install
```

If composer prompts for authentication like below:

```
Loading composer repositories with package information
Installing dependencies (including require-dev) from lock file
- Installing magento/magento-composer-installer (0.1.6)
Downloading: 100%

- Installing braintree/braintree_php (2.39.0)
Downloading: 100%

- Installing justinrainbow/json-schema (1.6.1)
Downloading: 100%

- Installing symfony/console (v2.6.13)
Downloading: 100%

- Installing symfony/process (v2.8.4)
Downloading: 100%

- Installing symfony/finder (v2.8.4)
Downloading: 100%

- Installing seld/jsonlint (1.4.0)
Downloading: 100%

- Installing composer/composer (1.0.0-alpha10)
Downloading: 100%

- Installing magento/composer (1.0.2)
Authentication required (repo.magento.com):
```

```
Username:
Password:
```

Login here https://www.magentocommerce.com/, and use *Public Key* as ***Username*** and *Private Key* as ***Password***.



Now set the permissions on files and directories.

```
sudo chmod -R 755 /var/www/magento2/
sudo chmod -R 777 /var/www/magento2/{pub,var}
```

3. Create Database

Now login to your mysql server with admin privileges and create a database and user for new magento2 installation.

```
mysql -u root -p

mysql> CREATE DATABASE magento;
mysql> GRANT ALL ON magento.* TO magento@'localhost' IDENTIFIED BY 'magento';
mysql> FLUSH PRIVILEGES;
mysql> quit
```

4. Configure Apache VirtualHost and PHP

Create Apache configuration file for your Magento website like `/etc/apache2/sites-available/magento2.example.com.conf` and add following content.

```
<VirtualHost *:80>
   DocumentRoot /var/www/magento2
   ServerName magento2.example.com

   <Directory /var/www/magento2>
       AllowOverride all
   </Directory>
</VirtualHost>
```

Now enable virtualhost using following command.

```
sudo a2ensite magento2.example.com
```

Also make sure to enable Apache rewrite module, which is recommended by Magento.

```
sudo a2enmod rewrite
```

You may want to set PHP `memory_limit` to avoid memory exhausted which is recommened by Magento too.

```
vi /etc/php.ini (find string by press / and type memory_limit)
memory_limit = 768M
```

After doing all above changes, make sure to restart Apache server.

```
sudo systemctl restart apache2.service
```

5. Installing Magento 2 Application

a) Via Web Installer

Let's begin the installation of Magento2 using web installer. Access your magento2 directory on web browser like below. It will redirect you to installation start page.

```
http://magento2.example.com/
```

b) Via Command-Line

Installing Magento 2 by using command line is a miracle, it decreased your installation time from 10min to 1min. By just execute one-line command.

```
cd /var/www/magento2
php bin/magento setup:install --base-url=http://magento2.example.com/ \
--db-host=localhost --db-name=magento \
--db-user=magento --db-password=magento \
--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 --cleanup-database --use-rewrites=1
```

6. Schedule Magento2 Cronjobs

Finally schedule the backgound cronjobs for your magento2 installation. These cronjobs does some activities like, re-indexing, Newsletters, Update of currency rates, sending automatic emails and generating sitemaps etc. To schedule these jobs edit crontab file. **www-data** is Apache 2 user, we should *never* schedule Magento 2 cronjob with root privilege.

```
crontab -u www-data -e
```

A text editor displays. (You might need to choose a text editor first.)

```
* * * * * /usr/bin/php /var/www/magento2/bin/magento cron:run | grep -v "Ran jobs by schedule"
>> /var/www/magento2/var/log/magento.cron.log
* * * * * /usr/bin/php /var/www/magento2/update/cron.php >>
/var/www/magento2/var/log/update.cron.log
* * * * * /usr/bin/php /var/www/magento2/bin/magento setup:cron:run >>
/var/www/magento2/var/log/setup.cron.log
```

Read Getting started with magento2 online: https://riptutorial.com/magento2/topic/2279/getting-started-with-magento2

Chapter 2: Configurable products and their variants.

Examples

Get a parent product and their children.

Here i will show you how to fetch

1. All parent(configuarble products)
2. A parent product and all of its children.

Get a parent product.

We will start by making a simple class that gets all our parent(Configurable products)

```
<?php

namespace Test\Test\Controller\Test;

use Magento\Framework\App\Action\Context;


class Products extends \Magento\Framework\App\Action\Action
{

        public function __construct(
        \Magento\Catalog\Model\ResourceModel\Product\CollectionFactory $_product_res_fac
        )
        {
            $this->_product_res_fac = $_product_res_fac;
        }

        public function getParentProducts()
        {
            return $this->_product_res_fac->create()->addAttributeToSelect('*')-
>addAttributeToFilter('type_id', ['eq' => 'configurable']);
        }

    }
```

As you see above our getParentProducts function will now return all configuarble products we currently have in our system.

Get parent and child products.

Here we first fetch our parent product and the we will get all children products that this parent have.

```
<?php
```

```
namespace Test\Test\Controller\Test;

use Magento\Framework\App\Action\Context;


class Products extends \Magento\Framework\App\Action\Action
{

        public function __construct(
        \Magento\Catalog\Model\Product $productModel
        )
        {
            $this->product= $productModel;
        }

        public function getParentProduct()
        {
            return $this->product->load("a product entity id goes here")
        }

        public function getChildProducts()
        {
            $_children = $this->getParentProduct()->getTypeInstance()->getUsedProducts($this-
>getParentProduct());
        }

    }
```

The function getChildProducts now returns a children collection so you would be able to run it through a foreach loop and get all product attributes that might be on it.

Read Configurable products and their variants. online: https://riptutorial.com/magento2/topic/10790/configurable-products-and-their-variants-

Chapter 3: Custom Theme

Remarks

`luma` theme as parent

```
{
    "name": "magento/luma",
    "description": "N/A",
    "require": {
        "php": "~5.5.0|~5.6.0|~7.0.0",
        "magento/theme-luma": "100.0.*",
        "magento/framework": "100.0.*"
    },
    "type": "magento2-theme",
    "version": "100.0.1",
    "license": [
        "OSL-3.0",
        "AFL-3.0"
    ],
    "autoload": {
        "files": [
            "registration.php"
        ]
    }
}
```

at the end

Run `php bin/magento setup:upgrade` this command after than below commands also needed sometimes

- `php bin/magento setup:static-content:deploy <language_pack_1> <language_pack_2> ... <language_pack_n>`
    - **<language_pack>**: `en_US` `nl_NL` `en_GB` etc
- `php bin/magento cache:flush` or `php bin/magento cache:clean`

Examples

Sample Theme

Theme.xml

`app/design/frontend/Magento/mytheme/theme.xml`

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

app/design/frontend/Magento/mytheme/composer.json

```
{
    "name": "magento/theme-frontend-blank",
    "description": "N/A",
    "require": {
        "php": "~5.5.0|~5.6.0|~7.0.0",
        "magento/theme-frontend-blank": "100.0.*",
        "magento/framework": "100.0.*"
    },
    "type": "magento2-theme",
    "version": "100.0.1",
    "license": [
        "OSL-3.0",
        "AFL-3.0"
    ],
    "autoload": {
        "files": [
            "registration.php"
        ]
    }
}
```

app/design/frontend/Magento/mytheme/registration.php

```
<?php
/**
* Copyright © 2015 Magento. All rights reserved.
* See COPYING.txt for license details.
*/
\Magento\Framework\Component\ComponentRegistrar::register(
    \Magento\Framework\Component\ComponentRegistrar::THEME,
    'frontend/Magento/mytheme',
    __DIR__
);
```

at the end

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

Read Custom Theme online: https://riptutorial.com/magento2/topic/6244/custom-theme

Chapter 4: Dependency Injection

Examples

Argument Replacement

```
<!-- <moduleDir>/etc/<area>/di.xml -->
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:ObjectManager/etc/config.xsd">
<!-- ... -->
    <type name="Vendor\Namespace\Model\SomeClass">
        <arguments>
            <argument name="object"
xsi:type="object">Vendor\Namespace\Model\SomeOtherClass</argument>
        </arguments>
    </type>
</config>
```

Class Preference

```
<!-- <moduleDir>/etc/<area>/di.xml -->
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:ObjectManager/etc/config.xsd">
<!-- ... -->
    <preference
        for="Vendor\Namespace\Model\Example"
        type="Vendor\Namespace\Model\AnotherExample" />
<!-- ... -->
</config>
```

Above Example is a syntax of override core model.

Here is a list of points which will describe you how to make it possible

1. **moduleDir** - Extension directory Like `app/code/custom/extension` here `extension` is your directory in which all the necessary folders of extension will be placed.

2. **area** - area will be `frontend` or `adminhtml`

    - **frontend** - if extension will use functionality of frontend than `di.xml` will goes to in this folder

    - **adminhtml** - if extension will use functionality of adminpanel than `di.xml` will goes to in this folder

    - so it will be `app/code/custom/extension/etc/frontend/di.xml` or `app/code/custom/extension/etc/adminhtml/di.xml`

    - If wants to use both the functionality than `di.xml` file will goes direct in `etc` folder no need to put in `frontend` or `adminhtml` folder. Like - `app/code/custom/extension/etc/di.xml`

3. **for="Vendor\Namespace\Model\Example"** at here, the path of the file which will override functionality of the desired function.

4. **type="Vendor\Namespace\Model\AnotherExample"** at here, the path of the file which will provides functions which will override by `step - 3`

Constructor Injection

```
/**
 * @var \Vendor\Module\Helper\Data
 */
protected $customHelper;

/**
 * Constructor call
 * @param \Vendor\Module\Helper\Data $customHelper
 */
public function __construct(
    \Vendor\Module\Helper\Data $customHelper
)
{
    $this->customHelper = $customHelper;
    parent::__construct();
}
```

Read Dependency Injection online: https://riptutorial.com/magento2/topic/2998/dependency-injection

Chapter 5: Event and observer in magento 2

Examples

How to use custom event and observer ?

Step 1: Create `events.xml` file according to your requirement in `frontend`, `Backend`, or both `YKM/Banner/etc/frontend/events.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../../../../../../lib/internal/Magento/Framework/Event/etc/events.xsd">

    <event name="controller_action_predispatch">
        <observer name="ykm_banner_before" instance="YKM\Banner\Observer\Help" />
    </event>
</config>
```

Step 2:

Create an Observer file `YKM/Banner/Observer/Help.php`

```
<?php
/**
 * Copyright © 2015 Magento. All rights reserved.
 * See COPYING.txt for license details.
 */
namespace Estdevs\Banner\Observer;

use Magento\Framework\Event\ObserverInterface;

class Help implements ObserverInterface
{

    public function execute(\Magento\Framework\Event\Observer $observer) {
       echo "this is good.";
    }
}
```

Read Event and observer in magento 2 online: https://riptutorial.com/magento2/topic/5277/event-and-observer-in-magento-2

Chapter 6: Get products from database

Examples

Get products using the Product Repository

To get products from the database, you need to use Magento 2's repository design pattern. Each module can be bundled with it's own repositories, and the Product Catalog module is not any different.

You can use dependency injection in your class to access the repository. A working example would look like this:

```
class Example
{
    /**
     * @var \Magento\Catalog\Model\ProductRepository
     */
    protected $productRepository;

    /**
     * @param \Magento\Catalog\Model\ProductRepository $productRepository
     */
    public function __construct(
        \Magento\Catalog\Model\ProductRepository $productRepository
    ) {
        $this->productRepository = $productRepository;
    }

    /**
     * Get product by ID
     * @return \Magento\Catalog\Api\Data\ProductInterface
     * @throws \Magento\Framework\Exception\NoSuchEntityException
     */
    public function getProductById(int $productId)
    {
        return $this->productRepository->getById($productId);
    }
}
```

A Repository has more functionality, like saving or deleting a product, as well as getting a list of products and using a filter, but that's beyond the scope of this example.

Read Get products from database online: https://riptutorial.com/magento2/topic/6459/get-products-from-database

Chapter 7: Magento 2 Commands for daily use

Remarks

All the commands can be executed writting only part of them.

For example:

- `php bin/magento cache:flush` can be traslated to:
  - `php bin/magento c:f`
  - `php bin/magento ca:f`
  - `php bin/magento c:fl`
  - `php bin/magento cache:f`
  - `php bin/magento c:flush`
  - etc.

You can write any part, and if it is not ambiguos, it will automatically know which one you want.

Examples

code compilation

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

You might need to delete var/di (including the folder) in order to go through compilation.

```
rm -rf var/di
```

Flush Cache

Flush all Magento Cache

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

Check cache status

```
php bin/magento cache:status
```

Enable Custom or 3rd Party Extensions

Enable and upgrade setup

```
php bin/magento module:enable YKM_Custom
php bin/magento setup:upgrade
```

Disable the Module

```
php bin/magento module:disable YKM_Custom
```

Another One - *module uninstall script is executed and whole module gets deleted afterwards. Only modules installed through Composer can be uninstalled.*

```
php bin/magento module:uninstall YKM_Custom
```

Display list of enabled and disabled modules

```
php bin/magento module:status
```

Update the database schema and data:

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

To see all available commands

```
php bin/magento
```

General List of Commands for Magento 2

```
php bin/magento setup:upgrade                   => Setup Upgrade
php bin/magento setup:di:compile                => Setup: Compile
php bin/magento indexer:reindex                 => Reindex
php bin/magento cache:flush                     => Clear Cache
php bin/magento deploy:mode:set developer       => Enable Developer Mode Magento
(developer/production)
php bin/magento deploy:mode:show                => Show Current Mode Magento
php bin/magento module:status                   => Module: Status
php bin/magento module:disable MODULE_NAME      => Module: Disable
php bin/magento module:enable MODULE_NAME       => Module: Enable
php bin/magento module:uninstall MODULE_NAME    => Module: Uninstall
php bin/magento cron:run                        => Cronjob: Run
```

Read Magento 2 Commands for daily use online:
https://riptutorial.com/magento2/topic/5368/magento-2-commands-for-daily-use

Chapter 8: Module structure

Examples

Catalog Module structure

For now I think the catalog module contains almost everything you can add to a module.

- **Api** - Contains the service contracts. A set of interfaces that should not be changed unless the minor version changes. Not mandatory for a custom module but nice to have for comercial extensions.
    - **Data** - Data interfaces. Each interface must have a model that implements it (example: interface for product model)
    - **ProductRepositoryInterface.php** - interfaces for repositories (must also have an implementation)
    - ... - others as above
- **Block** - blocks used in the layout for frontend and backend
    - **Adminhtml** - blocks used for backend
    - **Category** - frontend related blocks. Can be nested in as many folders as you like, but not mandatory
    - ... - same as above
- **Console** - folder containing cli commands
- **Controller** - contains frontend and backend controllers
    - **Adminhtml** - backend controllers
    - **Category** - frontend related controllers. Can be nested in as many folders as you like, but not mandatory
    - ... - same as above.
- **Cron** - code that should be executed via cron
- **etc** - contains module configuration xml files
    - **frontend** - contains configuration files loaded only on frontend
    - **adminhtml** - contains configuration files loaded only on backend
    - **webapi_rest** - contains configuration files loaded only for the rest api
    - **webapi_soapt** - contains configuration files loaded only for the SOAP api
    - **acl.xml** - ACL definitions
    - **catalog_attributes.xml** - default attributes for catalog entities.
    - **catalog_attributes.xsd** - validation schema for file above.
    - **config.xml** - default values for config settings
    - **crontab.xml** - cron jobs scheduling
    - **di.xml** - dependency injection preferences. (can also reside in adminhtml, frontend, webapi_*)
    - **events.xml** - observers declaration for events (can also reside in adminhtml, frontend)
    - **indexer.xml** - settings for different indexes that need to be executed when data changes
    - **module.xml** - the module declaration file
    - **product_*** - product related settings.

    - **webapi.xml** - webapi declaration paths.
    - **widget.xml** - widgets declarations.
- **Helper** - different module helpers
- **i18n** - language translation files
- **Model** - models, simple as that. they can be nested in as many folders as you like, but it's not mandatory.
- **Observer** - event observer classes
- **Plugin** - `around|before|after` plugins for different public methods.
- **Pricing** - pricing related classes. This is module specific. You can have as many folders as you like like this if you don't want to place them in the models folder.
- **Setup** - install/upgrade related files (installing upgrading schema and data)
- **Test** - unit tests
- **Ui** - ui components related classes.
- **view** - the html related part. The **V** in MVC.
    - **adminhtml** - admin related files
        - **layout** - xml layouts for adminhtml
        - **templates** - phtml templates for adminhtml
        - **ui_compoenent** - ui components related files (declaration)
        - **web** - assets (js, images)
        - **requirejs-config.js** - configuration for require.js
    - **base** - files used for both frontend and backend.
        - can have same subfolder structure as adminhtml
    - **frontend** - frontend related files
        - can have same subfolder structure as adminhtml
- **composer.json** - not mandatory, but nice to have if you distribute your module
- **registration.php** - the module registration file.
- **Licence*.txt, readme.md** - you know what this means. They are not mandatory

Read Module structure online: https://riptutorial.com/magento2/topic/4838/module-structure

Chapter 9: Optimizing Magento 2

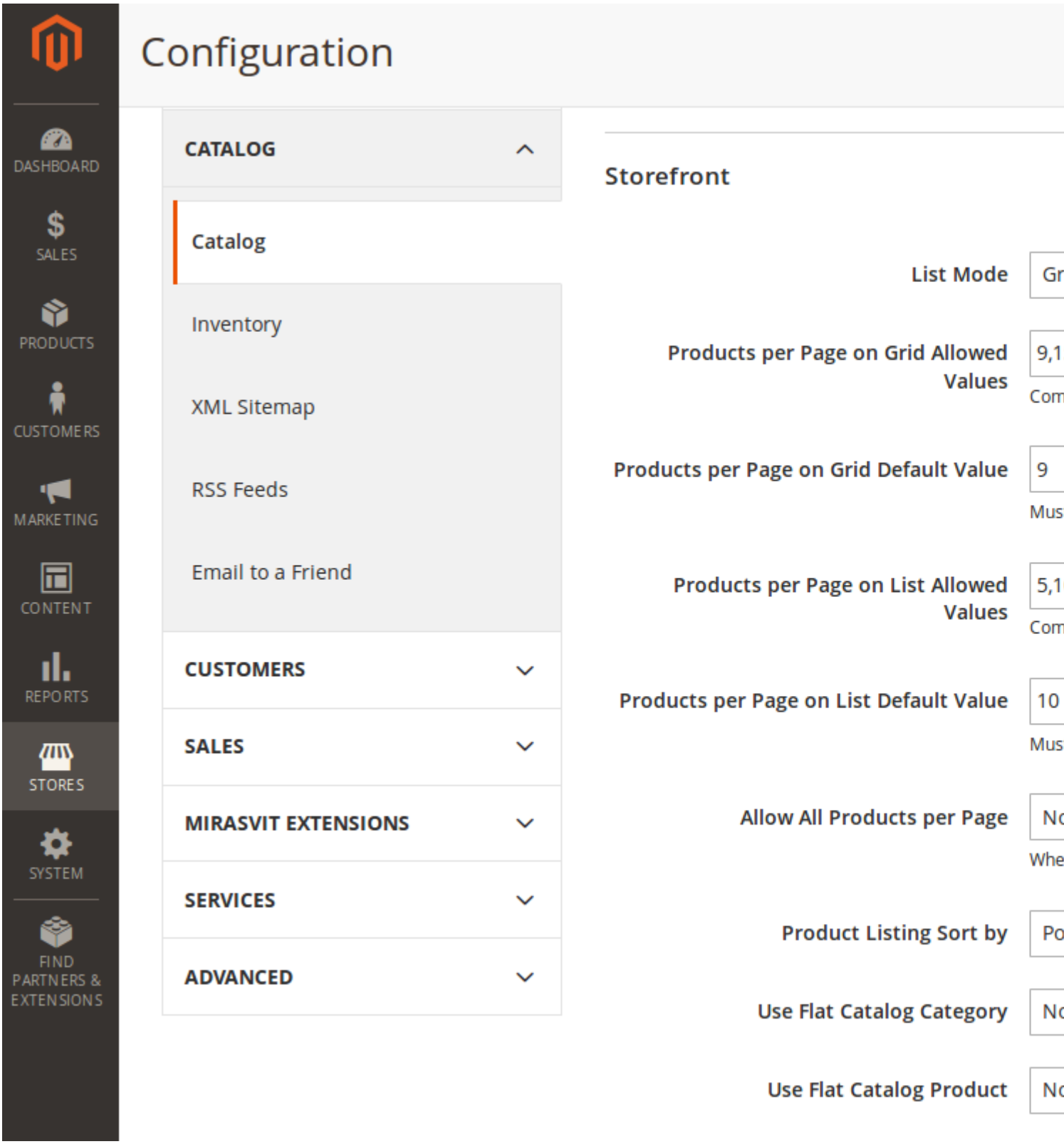
Examples

Configurations to optimize

1. Enable Flat Categories and Products

One of the top reasons of Magento speed issues with database read speed. To fasten the read speed of the database you should enable **Flat Catalog**. This will minify the number of database joins done when showing products and due to that the MySQL query complexity will be reduced.

Go to backend: **STORES > Configuration > CATALOG > Catalog > Use Flat Catalog Category** and put **Yes**.

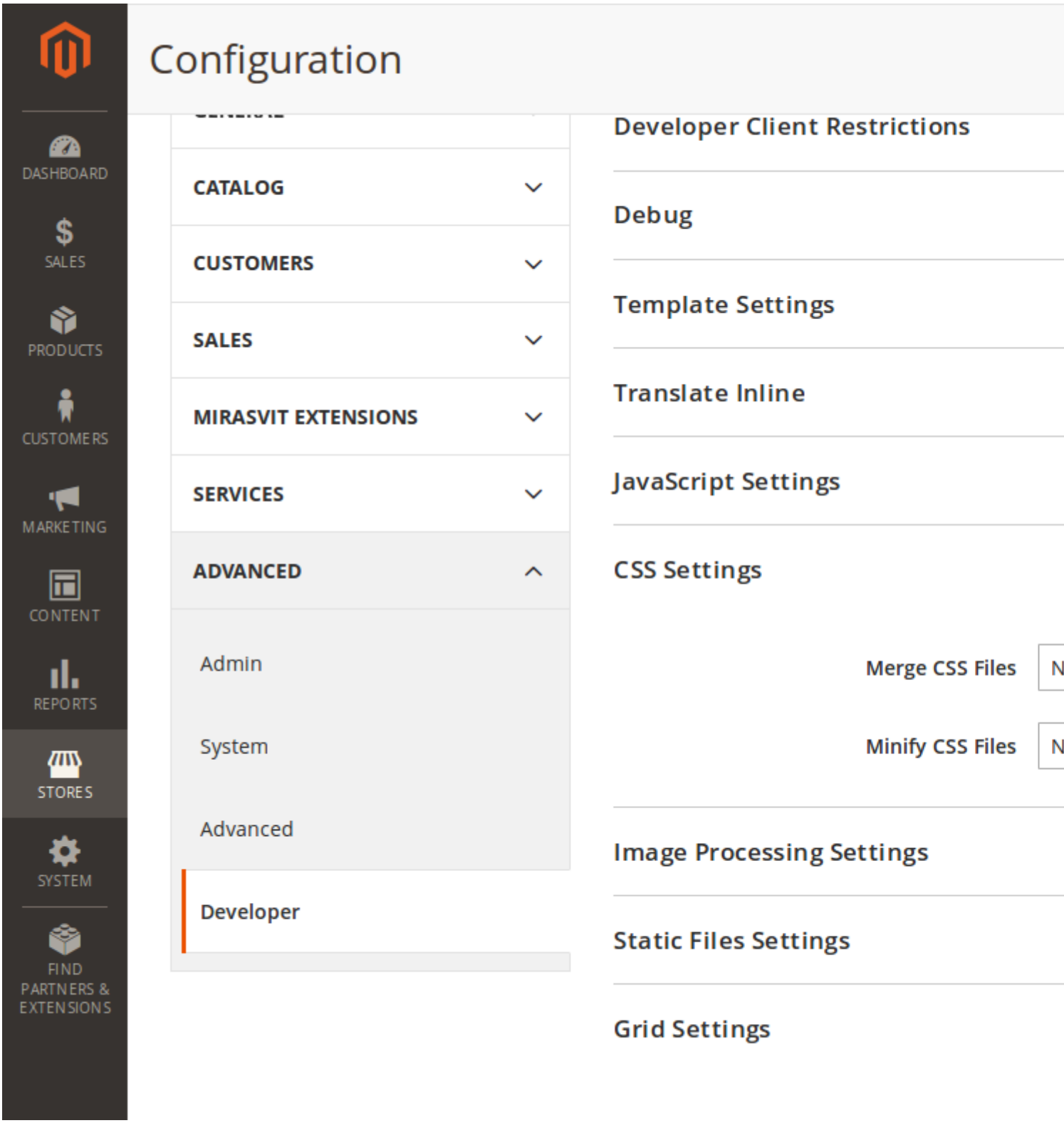


2. Merge CSS and JS Files

The next step you need to follow is merging and minifying CSS and Javascript files, that means making the web page as light as possible for the fast loading. Please put Magento 2 into **production** mode.

Go to backend: **STORES > Configuration > ADVANCED > Developer > CSS Settings** and put

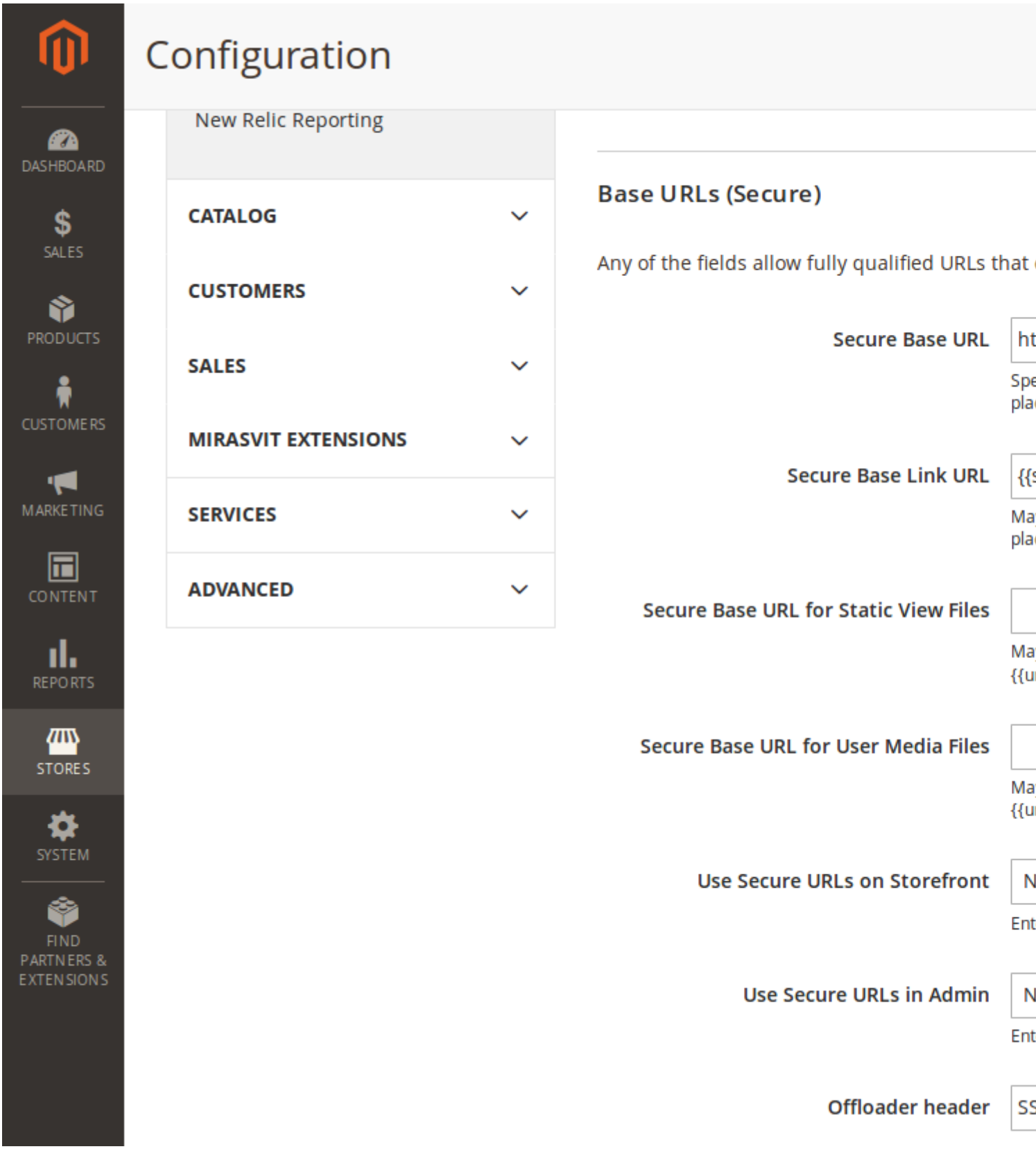
the **Merge CSS Files** and **Minify CSS Files** as **yes**.



3. Content Delivery Network

CDN, or Content Delivery Network, is an interconnected system of cache servers that use geographical proximity as criteria for delivering web content and actually helps your visitors to load pages faster as a result.

One of the Magento 2 features is out-of-the-box support of CDN and here's where you may find set up for it: **STORES > GENERAL > Configuration > Web > Base URLs (Secure)** and input your HTTPS CDN URLs in here and let your customers enjoy fast loading speed.



4. Caching

In the **System > Cache Management** enable your cache.

Cache Management

Flush Cache Storage

Refresh | Submit | 13 records found

	Cache Type	Description	Tags
☐	Configuration	Various XML configurations that were collected across modules and merged	CONF
☐	Layouts	Layout building instructions	LAYOU
☐	Blocks HTML output	Page blocks HTML	BLOC
☐	Collections Data	Collection data files	COLLE
☐	Reflection Data	API interfaces reflection data	REFLE
☐	Database DDL operations	Results of DDL queries, such as describing tables or indexes	DB_D
☐	EAV types and attributes	Entity types declaration cache	EAV
☐	Customer Notification	Customer Notification	CUST
☐	Page Cache	Full page caching	FPC
☐	Integrations Configuration	Integration configuration file	INTEG
☐	Integrations API Configuration	Integrations API configuration file	INTEG
☐	Translations	Translation files	TRAN
☐	Web Services Configuration	REST and SOAP configurations, generated WSDL file	WEBS

Additional Cache Management

Flush Catalog Images Cache — Pregenerated product images files

Flush JavaScript/CSS Cache — Themes JavaScript and CSS files combined to one file

Flush Static Files Cache — Preprocessed view files and static files

https://riptutorial.com/magento2/topic/10177/optimizing-magento-2

Chapter 10: Override i18n language pack

Syntax

- **<Vendor Namespace>** - Here namespace of the vendor custom theme or inbuilt theme namespace **I.E.** `Magento/Luma` Here `luma` is `vendor namespace`
- **<language package directory>** - Here language package directory like `en_us` or `nl_nl` or `en_gb`
- **<language package description>** - Here add description of the package like `English Us Package`
- **<language package code>** - Here code of the language package **I.E** `en_US` or `nl_NL` or `en_GB`

Remarks

After create above files and directories `language_package_code.csv` will goes to `Vendor Namespace` directory

Example

`/app/i18n/luma/en_us/en_US.csv`

or

`/app/i18n/luma/en_gb/en_GB.csv`

or

`/app/i18n/luma/nl_NL/nl_NL.csv`

Examples

Syntax example of override i18n language package

/app/i18n/<Vendor Namespace>/<language package directory>/composer.json

```
{
    "name": "<vendor namespance>/<language package directory>",
    "description": "<language package description>",
    "version": "100.0.1",
    "license": [
        "OSL-3.0",
        "AFL-3.0"
    ],
    "require": {
        "magento/framework": "100.0.*"
    },
    "type": "magento2-language",
```

```
    "autoload": {
        "files": [
            "registration.php"
        ]
    }
}
```

/app/i18n/<Vendor Namespace>/<language pack>/language.xml

```
<?xml version="1.0"?>
<language xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:App/Language/package.xsd">
    <code><language package code></code>
    <vendor><vendor namespace></vendor>
    <package><language package directory></package>
</language>
```

/app/i18n/<Vendor Namespace>/<language pack>/registration.php

```
<?php
\Magento\Framework\Component\ComponentRegistrar::register(
    \Magento\Framework\Component\ComponentRegistrar::LANGUAGE,
    '<vendor namespance>_<language package directory>',
    __DIR__
);
```

Read Override i18n language pack online: https://riptutorial.com/magento2/topic/10789/override-i18n-language-pack

Chapter 11: Upgrading Magento

Examples

Upgrade Magento via Composer

Check your current magento version

```
php bin/magento --version
```

Now Add the latest version to your composer.

```
composer require magento/product-community-edition 2.1.6 --no-update
```

Run Composer Update This will ask for the username and password take from your credentials from your marketplace account.

```
composer update
```

This will start process to start downloading and upgrading your magento

Finally Update you static content and remove var folder

```
rm -rf var/di var/generation
php bin/magento cache:flush
php bin/magento setup:upgrade
php bin/magento setup:di:compile
php bin/magento indexer:reindex
```

Recheck your magento version.

Read Upgrading Magento online: https://riptutorial.com/magento2/topic/9022/upgrading-magento

Chapter 12: Using Dependency Injection To Rewrite Object

Remarks

https://gielberkers.com/magento-2-why-use-rewrites-when-you-can-use-plugins/

http://devdocs.magento.com/guides/v2.0/extension-dev-guide/plugins.html

Examples

Some ways for modify a function in magento 2

Rewrite Class

File: `Namespace/ModuleName/etc/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">
   <preference for="Magento\Catalog\Controller\Product\View"
type="Namespace\ModuleName\Controller\Product\View" />
</config>
```

File: `Namespace\ModuleName\Controller\Product\View.php`

```
class View extends \Magento\Catalog\Block\Product\View
{
    ///Code logic here
}
```

Plugin into object.

File: `Namespace/ModuleName/etc/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\Catalog\Model\Product">
            <plugin name="name_of_plugin"
type="Namespace\ModuleName\Plugin\Catalog\Model\Product" sortOrder="1" disabled="false" />
        </type>
    </config>
```

File: `Namespace\ModuleName\Plugin\Catalog\Model\Product.php`

```
namespace Namespace\ModuleName\Plugin\Catalog\Model;

class Product
{
    public function beforeSetName(
        \Magento\Catalog\Model\Product $product, string $name)
    {
        /// Code logic here
        return $name;
    }

    public function afterGetName(
        \Magento\Catalog\Model\Product $product, string $name)
    {
         /// Code logic here
        return $name;
    }

    public function aroundSave(
        \Magento\Catalog\Model\Product $product, \Closure $proceed)
    {
        $this->doSomethingBeforeSave();
        $result = $proceed();
        if ($result) {
            $this->doSomethingAfterSave();
        }
        return $result;
    }
}
```

Read Using Dependency Injection To Rewrite Object online:
https://riptutorial.com/magento2/topic/6283/using-dependency-injection-to-rewrite-object

Credits

S. No	Chapters	Contributors
1	Getting started with magento2	4444, Community, ehzawad, Marek Skiba, Niroshan Ranapathi, Priyank, Rafael Corrêa Gomes, Toan Nguyen
2	Configurable products and their variants.	Anoxy
3	Custom Theme	Nirav Joshi, Qaisar Satti
4	Dependency Injection	bpoiss, Giel Berkers, Nirav Joshi
5	Event and observer in magento 2	Reena Parekh, Yogendra - eCommerce Developer
6	Get products from database	Giel Berkers, matiaslauriti
7	Magento 2 Commands for daily use	AlexL, Andrew Stepanchuk, Ankit Shah, belfort1, Jignesh Khunt, matiaslauriti, Yogendra - eCommerce Developer
8	Module structure	Akif, belfort1, Giel Berkers, Marius, Tom
9	Optimizing Magento 2	Rafael Corrêa Gomes
10	Override i18n language pack	Nirav Joshi
11	Upgrading Magento	Priyank
12	Using Dependency Injection To Rewrite Object	Dmitri Sologoubenko, HoangHieu, Rafael Corrêa Gomes