Magento 2.1

Developer Basics

November 2, 2016

Plan

Smile
OPEN SOURCE SOLUTIONS

Plan

1. Magento 2 Fundamentals
   - About Magento 2
   - Requirements
   - Preparing the Training Project with Ansible
   - Preparing the Training Project manually

Smile
OPEN SOURCE SOLUTIONS

Plan

Smile
OPEN SOURCE SOLUTIONS

About Magento 2

- Most advanced OpenSource eCommerce solution
- Based on Zend Framework (1.12.* and 2.4.*)
- Using Magento Framework
- Development started at the beginning of 2010 by Varien
  - First dev beta release 2014-12-18
  - First public release 2015-11-17
- Current versions
  - Community 2.1.2: Open Software License 3.0
  - Enterprise 2.1.2: Magento Enterprise Edition License
- Main Magento's drawbacks
  - Software complexity and slowness
  - Lack of documentation (it's better now)

Smile
OPEN SOURCE SOLUTIONS

Plan

Smile
OPEN SOURCE SOLUTIONS

Requirements

- Linux (Windows not supported in production)
- Apache >=2.2 or Nginx >=1.8
- MySQL >=5.6 (Oracle or Percona)
- PHP 5.6 (>= 5.6.5)
- PHP 7.0 (>= 7.0.6)
- PHP 7.1
- warning, Magento 2.1 does not support PHP 5.5 anymore
  Sources:
  `http://devdocs.magento.com/guides/v2.1/install-gde/system-requirements-2.1-tech.html`

Smile
OPEN SOURCE SOLUTIONS

Requirements

- Needed PHP extensions:
  - bc-math(for EE only)
  - curl
  - GD, ImageMagick
  - intl
  - mbstring
  - mcrypt
  - mhash
  - openssl
  - PDO/MySQL
  - SimpleXML
  - soap
  - xml
  - xsl
  - zip
- Minimal PHP configuration
  - memory_limit = 768M
  - max_execution_time = 180000 (see provided .htaccess)

Smile
OPEN SOURCE SOLUTIONS

Manual Installation

1. Using composer:
   `http://devdocs.magento.com/guides/v2.1/install-gde/prereq/integrator_install.html`
2. Or Download from
   `https://www.magentocommerce.com/download`

Smile
OPEN SOURCE SOLUTIONS

System permissions

The write permissions for apache are needed on the following directories:

- app/etc
- pub/media
- pub/static
- var

Be very careful when touching a folder called `var`, do not change permissions on the system's `/var`

Plan

Smile
OPEN SOURCE SOLUTIONS

Preparing the Training Project with Ansible

- Using Smile Magento2 Architecture Skeleton

  `https://git.smile.fr/magento2/architecture-skeleton`

Smile
OPEN SOURCE SOLUTIONS

Preparing the Training Project with Ansible

Prerequisites on the host machine

- Install some packages:

```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get install curl
sudo apt-get install php-cli
php -v
```

Preparing the Training Project with Ansible

Prerequisites on the host machine

- Install Python-LDAP

  `sudo apt-get install python-ldap`

- upload your SSH key to the LDAP : `https://wiki.smile.fr/view/Systeme/UsingSmileLDAP#Upload_your_SSH_key_to_the_LDAP`

Preparing the Training Project with Ansible

Prerequisites on the host machine

- Install GIT

  ```
  sudo apt-get install git
  ```

- Add your public SSH key: `https://git.smile.fr/profile/keys`

Preparing the Training Project with Ansible

Prerequisites on the host machine

- Install Ansible 2.1:

```
sudo apt-get purge ansible
sudo apt-get install python-crypto python-httplib2 python-jinja2
sudo apt-get install python-paramiko python-pkg-resources python-yaml
sudo apt-get install python-pip
sudo pip install ansible==2.1.1.0
ansible --version
```

- More information:

https://wiki.smile.fr/view/Systeme/AnsibleIntro

Smile
OPEN SOURCE SOLUTIONS

Preparing the Training Project with Ansible

Prerequisites on the host machine

- Install the LXC package:

  `sudo apt-get install smile-lxc`

- Usages: `https://wiki.smile.fr/view/Dirtech/LxcForDevs`

Smile
OPEN SOURCE SOLUTIONS

Preparing the Training Project with Ansible

Prerequisites on the host machine

- Install Composer:

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

- Add the Smile repositories to Composer:

```
https://wiki.smile.fr/view/PHP/HowToConfigComposer
```

Preparing the Training Project with Ansible

Prerequisites on the host machine

- Configure Composer: ${HOME}/.composer/auth.json

```
{
    "github-oauth": {
        "github.com": "[Your Github key]"
    },
    "http-basic": {
        "repo.magento.com": {
            "username": "[Public Key]",
            "password": "[Private Key]"
        }
    }
}
```

- Get your Github authentication keys:

  `https://github.com/settings/tokens`

- Get your Magento authentication keys: `http://devdocs.magento.com/guides/v2.1/install-gde/prereq/connect-auth.html`

Smile
OPEN SOURCE SOLUTIONS

Preparing the Training Project with Ansible

Initialise your project (1/2)

- Follow the steps of the **Initialise your project** part
- Step 1: init the project

```
cd ~/
mkdir projects
cd projects
bash <(curl -sL https://git.smile.fr/magento2/architecture-skeleton/raw/master/init.sh)
> name:        magento2
> version:     CE
> sample data: Y
> smile user:  [enter]
> separate:    N
> confirm:     Y
```

- Step 2: create the LXC

```
cd magento2
sudo cdeploy
./architecture/scripts/provision.sh lxc
```

- Step 3: Install Magento 2 database

```
./architecture/scripts/install.sh lxc
```

Smile
OPEN SOURCE SOLUTIONS

Preparing the Training Project with Ansible

Initialise your project (2/2)

- Step 4: Basic Configuration + First Commit on Git

  https://git.smile.fr/magento2/architecture-skeleton/blob/develop/architecture/docs/init.md#step-5

- Step 5: Try Magento 2
  - Front: `http://magento2.lxc`
  - Back: `http://magento2.lxc/admin/`
    (user: admin, password: magent0)

Smile
OPEN SOURCE SOLUTIONS

Preparing the Training Project with Ansible

Some Important Scripts

- Read the **Script list** part
- **./architecture/scripts/cache-clean.sh** to clean the caches
- **./architecture/scripts/generate-urn-catalog.sh** to generate the URN catalog for PHPStorm
- **./architecture/scripts/setup-upgrade.sh** to execute the new setup files

Some Important Tools

- Read the **Analyze your code** part
- **bin/spbuilder** for PHPUnit, CodeSniffer, ...
- **bin/SmileAnaliser** for specific Magento 2 profiles

Smile
OPEN SOURCE SOLUTIONS

Plan

Smile
OPEN SOURCE SOLUTIONS

Preparing the Training Project manually

- Using LXC virtualisation system
- LXC name : magento2
- System : Debian 8 Jessie
- Composer will be used only from the host
- Never launch it with the root user in the LXC

Preparing the Training Project manually

Prerequisites on the host machine

- Install some packages:

```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get install curl
sudo apt-get install php-cli
sudo apt-get install git
sudo apt-get install smile-lxc
```

Smile
OPEN SOURCE SOLUTIONS

Preparing the Training Project manually

Prerequisites on the host machine

- Install Composer:

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

- Add the Smile repositories to Composer:

```
https://wiki.smile.fr/view/PHP/HowToConfigComposer
```

Preparing the Training Project manually

Prerequisites on the host machine

- Configure Composer: ${HOME}/.composer/auth.json

```
{
    "github-oauth": {
        "github.com": "[Your Github key]"
    },
    "http-basic": {
        "repo.magento.com": {
            "username": "[Public Key]",
            "password": "[Private Key]"
        }
    }
}
```

- Get your Github authentication keys:

  `https://github.com/settings/tokens`

- Get your Magento authentication keys: `http://devdocs.magento.com/guides/v2.1/install-gde/prereq/connect-auth.html`

Smile
OPEN SOURCE SOLUTIONS

Preparing the Training Project manually

- Get the Magento 2 sources

```
cd ~/
mkdir projects
cd projects
composer create-project --repository-url=https://repo.magento.com/
    magento/project-community-edition magento2 --ignore-platform-reqs --no-install
cd magento2
composer config bin-dir ./bin
composer install --ignore-platform-reqs
```

Smile
OPEN SOURCE SOLUTIONS

Preparing the Training Project manually

Prepare the lxc (see 01-init-without-ansible)

- Put the training folder ./architecture in your project
- Put the training file ./lxcfile in your project
- Deploy the LXC

```
sudo cdeploy
```

- Verify the LXC

```
ssh smile@magento2.lxc
sudo -u www-data php -v
mysql -h localhost -u magento2 -p magento2
password: [!magento2]
exit
exit
```

- Commit on git and open the project under your favorite IDE !

Smile
OPEN SOURCE SOLUTIONS

Preparing the Training Project manually

Install Magento 2

- Launch the Setup Wizard: http://magento2.lxc/setup/
- Click on "Agree and Setup Magento"
- Click on "Start Readiness Check",
- Click on "Next"

Preparing the Training Project manually

Install Magento 2

- Enter the database information
  - Host: localhost
  - User: magento2
  - Password: !magento2
  - database: magento2
- Enter the Web configuration information
  - Store Address: http://magento2.lxc/
  - Magento Admin Address: admin
- Enter the Store information
  - Time Zone: Central European Standard Time (Europe/Paris)
  - Currency: Euro
  - Default Language: English (United States)
  - Advanced Modules Configurations: Select All

Smile
OPEN SOURCE SOLUTIONS

Preparing the Training Project manually

Install Magento 2

- Enter the Admin Account information
- Click on "Install Now"
- Open the "Console Log"
- Finished!
- Go on http://magento2.lxc/

Preparing the Training Project manually

Install Magento 2

- First time in the Back Office: http://magento2.lxc/admin/
- The indexers are invalid... How to reindex in CLI ?

```
ssh smile@magento2.lxc
cd /var/www/magento2/
sudo -u www-data bin/magento indexer:reindex
sudo -u www-data bin/magento cache:clean
exit
```

- Configure the cron

```
ssh root@magento2.lxc
crontab -e -u www-data
*/1 * * * * /var/www/magento2/bin/magento cron:run
exit
```

Smile
OPEN SOURCE SOLUTIONS

Preparing the Training Project manually

Magento 2 and PHP-Storm

- In order to use auto-validation of the XML files, PHP-Storm must kown where the XSD files are.
- Magento can generate automatically the URN catalog for PHP-Storm:

```
ssh smile@magento2.lxc
cd /var/www/magento2

chmod 666 .idea/misc.xml
sudo -u www-data ./bin/magento dev:urn-catalog:generate .idea/misc.xml
chmod 664 .idea/misc.xml
```

Close PHP-Storm before generating the misc.xml file

Smile
OPEN SOURCE SOLUTIONS

Preparing the Training Project manually

Magento 2 and Sample Data

```
ssh smile@magento2.lxc
cd /var/www/magento2
./bin/magento sampledata:deploy
=> Username [see in your .composer/auth.json file]
=> Password [see in your .composer/auth.json file]
=> Store credentials [No]
sudo -u www-data ./bin/magento setup:upgrade
sudo -u www-data ./bin/magento cache:clean
```

Preparing the Training Project manually

- Commit all the files

```
cd [PROJECT]/
git add --all .
git status
git commit . -m "installing magento2"
```

Plan

2 Key notions

- Scope notion
- Product types

Plan

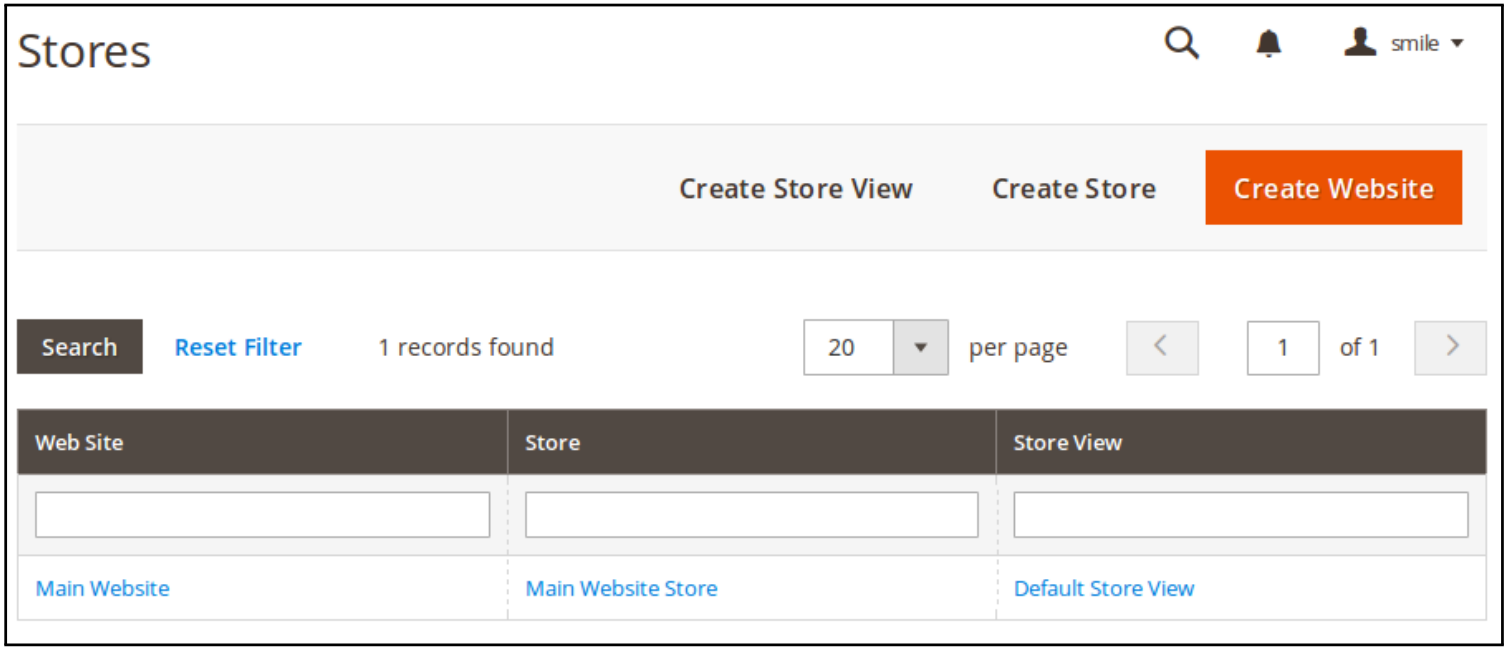
Smile
OPEN SOURCE SOLUTIONS

Scope notion

Magento is organized in 3 types of scopes.

- Website
- Store
- Store view

Go on the Back Office >Stores >Settings >All Stores



Smile
OPEN SOURCE SOLUTIONS

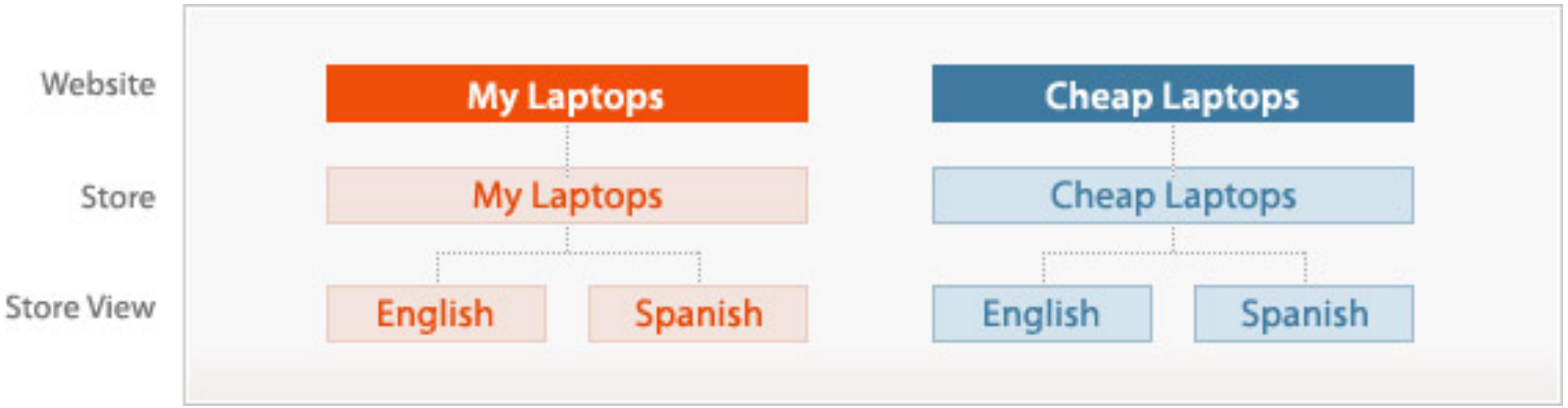
Scope notion>Website

- Definition
  - Collection of stores that share the same customer information (login, orders and cart), currency, payments, taxes, shipping etc

Smile
OPEN SOURCE SOLUTIONS

Scope notion>Store

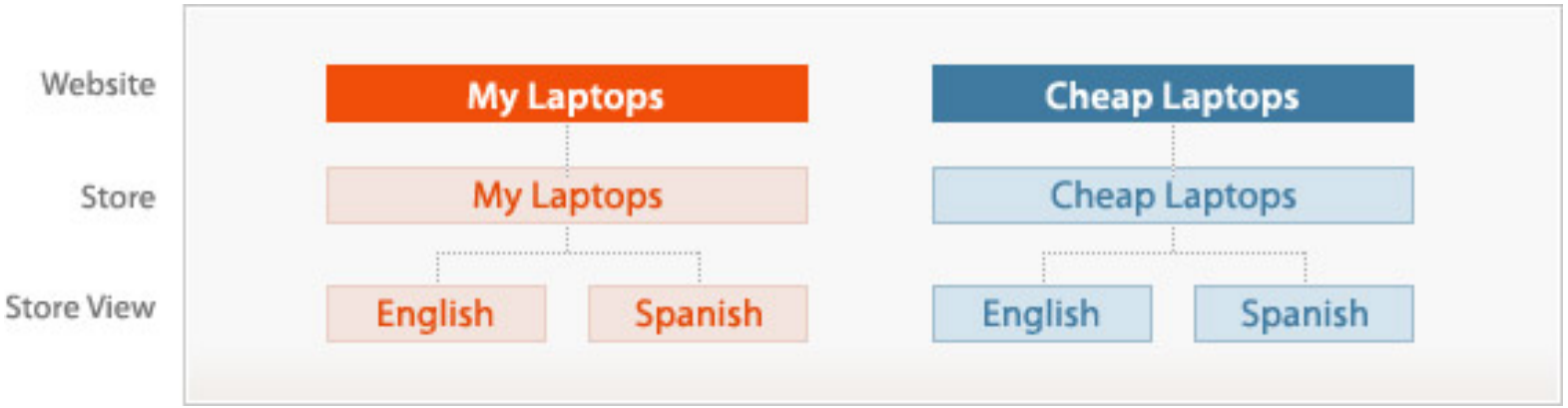
- Definition
  - A collection of store view
  - The root category is defined at the store level



Smile
OPEN SOURCE SOLUTIONS

Scope notion>Store View

- Definition
  - The view of a website in a specific language



Smile
OPEN SOURCE SOLUTIONS

Plan

2 Key notions

- Scope notion
- Product types

Smile
OPEN SOURCE SOLUTIONS

Product types 1/2

- Simple product
  - No specificities, base product type, mostly used
  - Eg.: book
- Grouped product
  - A grouped product is a package made with two or more simple products
  - Price, description, images, etc. can be specified on their own
  - Eg.: camera + SD card sold together
- Configurable product
  - Some attributes can be chosen by customer between a set of variants
  - Each variant is a simple product associated to the configurable in backend
  - Eg.: t-shirt with choice of color and size (each combinaison is an existing simple product)

Smile
OPEN SOURCE SOLUTIONS

Product types 2/2

- Bundle product
  - Looks like grouped product, but user can create it's own set based on predifined choices
  - Eg.: computer
    - motherboard and CPU are mandatory, user can choose each in a given list, qty is limited to one
    - mouse, keyboard are optionnals
    - extra RAM can be added, optionnal, qty is 0 to N
- Virtual product
  - Not a physical product, esp. does not manage stock qty
  - Eg.: all kind of services
- Downloadable product
  - Does not exists physicaly, but can be downloadable
  - Eg.: a training video
- **Try using simple products as much as you can**

Smile
OPEN SOURCE SOLUTIONS

Plan

3 Architecture

- Magento directory structure
- Magento modes
- Magento areas
- Magento module structure
- Configuration files

Plan

Smile
OPEN SOURCE SOLUTIONS

Magento directory structure

- /path/to/magento/root/
  - **app/** : application code (see after)
  - **bin/** : magento shell file
  - **dev/** : unit test files
  - **lib/internal** : php libraries that can't be installed via composer
  - **lib/web** : web libraries (jquery, wysiwyg, ...)
  - **phpserver/** : if you want to use the Built-in webserver in PHP
  - **pub/** : the public folder
    - errors
    - media
    - static
  - **setup/** : the setup app folder (see System >Tools >Web Setup Wizard)
  - **update/** : the update app folder (same tool)
  - **var/**
    - cache

Smile
OPEN SOURCE SOLUTIONS

Magento directory structure

The **vendor/** folder

- Contain all the libraries used by Magento
- Contain all the Magento core files in **magento** namespace folder
  - **framework** : The Magento framework library files
  - **language-xx_xx** : The Magento language packages filess
  - **magento2-base** : The Magento basic structure
  - **module-xxxxx** : The Magento modules files
  - **theme-xxx-xxx** : The Magento themes files
  - **zendframework1** : The Zend Framework v1 files
- Never modify any of thoses files
- to update the librairies:

```
cd ~/projects/magento2/
composer update --ignore-platform-reqs
```

Smile
OPEN SOURCE SOLUTIONS

Magento directory structure

The **app/** folder

- **etc/**
  - App configuration (env.php, list of activated modules, ...)
- **code/**
  - Specific modules code (no more pools like core, local, ...)
- **design/**
  - Specific Theme and template

Smile
OPEN SOURCE SOLUTIONS

Plan

Smile
OPEN SOURCE SOLUTIONS

Magento modes

- They are three primary **modes** available
  - Developer
  - Production
  - Default
- There is also a maintenance mode

Magento modes

Developer Mode

- Static file materialization is not enabled
- Uncaught exceptions displayed in the browser
- Exceptions thrown in error handler, not logged
- System loggin in var/report, highly detailed

Magento modes

Production Mode

- Deployment phase on the production system; hightest performance
- Exceptions are not displayed to the user – written to logs only
- This mode disables static file materialization
- The Magento docroot can have read-only permissions.

Magento modes

Default Mode

- Used when no other mode is specified
- Hides exceptions from the user and writes them to log files
- Static file materialization is enabled
- Not recommended / Not optimized for production

Magento modes

Maintenance Mode

- Used to mage a site unavailable to the public during updates or other changes
- Detect the **var/.maintenance.flag** file
- Can use a authorized list of ip in the **var/.maintenance.ip** file

Magento modes

Specify a Mode

- In the apache virtualhost: **SetEnv MAGE_MODE developer**

Plan

Smile
OPEN SOURCE SOLUTIONS

Magento areas

- They are six main **areas** for configuration files
  - global
  - frontend
  - adminhtml
  - webapi_rest
  - webapi_soap
  - crontab
- see the **\Magento\Framework\App\Area** class
- see the **etc** folder of the **Magento_Catalog** module for a example.

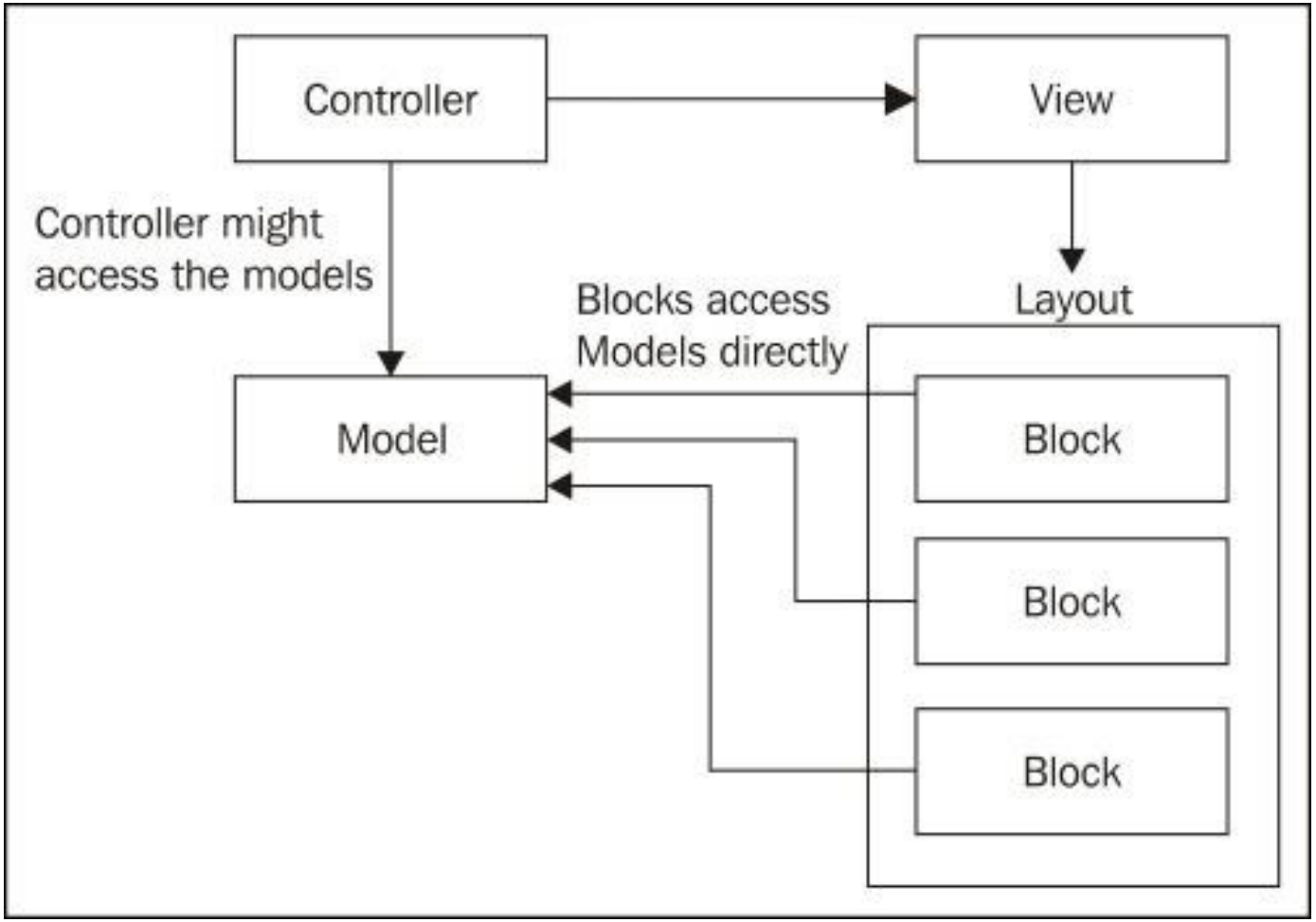
Smile
OPEN SOURCE SOLUTIONS

Plan

Smile
OPEN SOURCE SOLUTIONS

Magento module structure

Design pattern MVC (Model, View, Controller)



Smile
OPEN SOURCE SOLUTIONS

Magento module structure

Magento Module Architecture

- A Magento module encapsulates for a functional perimeter (catalog, customer, cataloginventory)
  - controllers
  - models
  - display (blocks)
  - ...
- One module cannot be responsible of multiple features
- Multiple modules cannot be responsible for one feature

Magento module structure

Module directories **app/code/MyNamespace/MyModule/**

- **Api/** Api Interface files for all the module classes
- **Block/** Fontend and backend block code and methods
- **Config/** Specific config readers of the module
- **Console/** Console command files for the Magento CLI
- **Controller/** Controller logic for navigation (both FO and BO)
- **Cron/** Cron classes
- **etc/** XML and XSD Module configuration files
- **Helper/** Utilitary functions
- **i18n/** CSV language files
- **Model/** Base logic and methods
- **Observer/** Observer classes (see after)
- **Plugin/** Plugin classes (see after)
- **Rewrite/** Rewrite classes (see after)
- **Setup/** Update files (adding or modifying a database

Smile
OPEN SOURCE SOLUTIONS

Magento module structure

Helloworld module (see 02-helloworld) 1/4

- Module folder: **./src/app/code/Training/Helloworld**
- create **./etc/module.xml** file:

```
<?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="Training_Helloworld" setup_version="1.0.0">
    </module>
</config>
```

- create **./registration.php** file:

```
<?php
\Magento\Framework\Component\ComponentRegistrar::register(
    \Magento\Framework\Component\ComponentRegistrar::MODULE,
    'Training_Helloworld',
    __DIR__
);
```

Smile
OPEN SOURCE SOLUTIONS

Magento module structure

Helloworld module (see 02-helloworld) 2/4

- create **./etc/frontend/routes.xml** file:

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

Smile
OPEN SOURCE SOLUTIONS

Magento module structure

Helloworld module (see 02-helloworld) 3/4

- create **./Controller/Index/Index.php** file:

```php
<?php
/**
 * Magento 2 Training Project
 * Module Training/Helloworld
 */
namespace Training\Helloworld\Controller\Index;

/**
 * Action: Index/Index
 *
 * @author    Laurent MINGUET <lamin@smile.fr>
 * @copyright 2016 Smile
 */
class Index extends \Magento\Framework\App\Action\Action
{
    /**
     * Execute the action
     *
     * @return void
     */
    public function execute()
    {
        $this->getResponse()->appendBody('Hello World !');
    }
}
```

Smile
OPEN SOURCE SOLUTIONS

Magento module structure

Helloworld module (see 02-helloworld) 4/4

- Test: `http://magento2.lxc/helloworld/`
- Error 404 not found ?! Why ?
- Ask Magento to clean the cache and to register the module ! Needed each time
  - you create a new module
  - you add a new php class
  - you modify the parameters of a PHP class constructor
  - you modify a XML config file
- How to clean the caches:

```
sudo -u www-data bin/magento cache:clean
```

- How to detect new modules and to launch new setups:

```
sudo -u www-data bin/magento cache:clean
sudo -u www-data rm -rf var/di/* var/generation/*
sudo -u www-data bin/magento setup:upgrade
```

Smile
OPEN SOURCE SOLUTIONS

Plan

Smile
OPEN SOURCE SOLUTIONS

Configuration files

- All the XML configuration files of a module are in the **etc/** folder
- All the XML configuration files use a XSD schema file
  - URN Link: urn:magento:framework:Module/etc/module.xsd
  - Real file: ./vendor/magento/framework/Module/etc/module.xsd
- The files directly in the **etc/** folder are for all the areas
- The files in a specific area folder are only for this area
- Each XML configuration files of each modules are merged
- A XSD schema file can exist to validate the merged file
- Look at magento:module_catalog module:
  - ./etc/product_options.xsd
  - ./etc/product_options_**merged**.xsd

Smile
OPEN SOURCE SOLUTIONS

Plan

4 Concepts

- Object Manager Factory
- Dependency Injection
- Events and Observers
- Plugins
- Rewrites

Plan

4 Concepts

- Object Manager Factory
- Dependency Injection
- Events and Observers
- Plugins
- Rewrites

Object Manager Factory

- Replace Mage::getModel, Mage::getSingleton, ... of M1
- Method **create**: return a new instance
- Method **get**: return a singleton
- Never use the global Object Manager Factory
- Use only the Object Manager Factory of the object you want
- Bad usage:

  `\Magento\Catalog\Controller\Adminhtml\Category\RefreshPath::37`
- Good usage:

  `\Magento\Catalog\Block\Adminhtml\Category\Widget\Chooser::96`
- Object Manager Factories are automatically generated by Magento 2

Smile
OPEN SOURCE SOLUTIONS

Plan

Smile
OPEN SOURCE SOLUTIONS

Dependency Injection

- Methods must not instentiate objects when they need them
- Methods that have to create object must use the Object Manager Factory of this object
- Methods must just use the objects that have been injected in the class
- Two injection methods
  - The constructor asks for the objects the class needs to work
  - The method asks for the specific objects it needs to work
- Always prefer asking for interfaces instead of final classes

Dependency Injection

- Constructor Injection Example:

  `\Magento\Framework\url`

- Method Injection Example:

  `\Magento\Backend\Model\Menu\Builder::getResult`

- Object Manager Factory Example:

  `\Magento\Framework\CurrencyFactory`

Dependency Injection

di.xml files

- You can tell the name of the real class to use automatically when a interface or a class is asked by a constructor

```
<preference for="Magento\Cms\Api\Data\PageInterface" type="Magento\Cms\Model\Page" />
```

- For a specific object, you can specify the parameters to use.

  It allows you to pass some specific objects and values

```
<type name="Magento\Catalog\Helper\Product">
    <arguments>
        <argument name="catalogSession" xsi:type="object">
            Magento\Catalog\Model\Session\Proxy
        </argument>
        <argument name="reindexPriceIndexerData" xsi:type="array">
            <item name="byDataResult" xsi:type="array">
                <item name="tier_price_changed" xsi:type="string">tier_price_changed</item>
            </item>
            ...
        </argument>
        <argument name="reindexProductCategoryIndexerData" xsi:type="array">
            <item name="byDataChange" xsi:type="array">
                <item name="category_ids" xsi:type="string">category_ids</item>
                ...
            </item>
        </argument>
        <argument name="productRepository" xsi:type="object" shared="false">
            Magento\Catalog\Api\ProductRepositoryInterface\Proxy
```

Smile
OPEN SOURCE SOLUTIONS

Dependency Injection

When asking for a object, The Magento 2 Object Manager:

- Analyse the parameters asked by the constructor of the asked class
- Use **./etc/di.xml** and **./etc/[area]/di.xml** files to prepare the parameters
- Create the asked object by giving all the parameters to the constructor
- Return the asked object

Dependency Injection

Object: **Injectable** and **Non-injectable**

- **Injectable**: An object (typically a singleton) that can be instantiated by the object manager
- **Non-injectable**: An object that cannot be instantiated by the object manager.
  Typically, this object
  - has a transient lifestyle
  - requires external input to be properly created
  - example: **Magento\Catalog\Model\Product**
- Most models are not injectable

Smile
OPEN SOURCE SOLUTIONS

Dependency Injection

Object: **Injectable** and **Non-injectable**

- **Injectable** can request for other **Injectable** objects in the constructor
- **Injectable** can not request for **Non-injectable** objects in the constructor
- If **Injectable** object produces **Non-injectable** object, it has to require the factory of this **Non-injectable** object in its constructor
- If **Injectable** object performs actions on a **Non-injectable** object, it has to recieve it as a method argument

Dependency Injection

Compiler tool

- Reads all the class definition using reflection
- Generates all the required factories
- Generates interceptors for all classes that have plugins (see after)
- Compile definitions for all modules and libraries
- And others...
- Run the compiler tool:
  `sudo -u www-data ./bin/magento setup:di:compile`
- See the result in **./var/di** and in **./var/generation**

Smile
OPEN SOURCE SOLUTIONS

Dependency Injection

Dependency Injection practice (see 03-di) 1/5

- In the previous module **Training/Helloworld**
- Create a new action for the url `http://magento2.lxc/helloworld/product/index`
- Get the **id** parameter in the url and load the corresponding product
- Display the name of the product

Dependency Injection practice (see 03-di) 2/5

- New File: **./Training/Helloworld/Controller/Product/Index.php**

```php
<?php
/**
 * Magento 2 Training Project
 * Module Training/Helloworld
 */
namespace Training\Helloworld\Controller\Product;

/**
 * Action: Product/Index
 *
 * @author    Laurent MINGUET <lamin@smile.fr>
 * @copyright 2016 Smile
 */
class Index extends \Magento\Framework\App\Action\Action
{

}
```

Dependency Injection practice (see 03-di) 3/5

- Ask for the Product Factory in the constructor

```php
<?php
    /**
     * @var \Magento\Catalog\Model\ProductFactory
     */
    protected $productFactory;

    /**
     * PHP Constructor
     *
     * @param \Magento\Framework\App\Action\Context $context
     * @param \Magento\Catalog\Model\ProductFactory $productFactory
     */
    public function __construct(
        \Magento\Framework\App\Action\Context $context,
        \Magento\Catalog\Model\ProductFactory $productFactory
    ) {
        parent::__construct($context);

        $this->productFactory = $productFactory;
    }
```

Dependency Injection practice (see 03-di) 4/5

- Load the asked product

```php
<?php
    /**
     * Get the asked product
     *
     * @return \Magento\Catalog\Model\Product|null
     */
    protected function getAskedProduct()
    {
        // get the asked id
        $id = (int) $this->getRequest()->getParam('id');
        if (!$id) {
            return null;
        }

        // get the product
        $product = $this->productFactory->create()->load($id);
        if (!$product->getId()) {
            return null;
        }

        return $product;
    }
```

Dependency Injection practice (see 03-di) 5/5

- Display the product name

```php
<?php
    /**
     * Execute the action
     *
     * @return void
     */
    public function execute()
    {
        $product = $this->getAskedProduct();
        if (is_null($product)) {
            $this->_forward('noroute');
            return;
        }

        $this->getResponse()->appendBody('Product: '.$product->getName());
    }
```

Plan

Smile
OPEN SOURCE SOLUTIONS

Events and Observers

Event Design Pattern

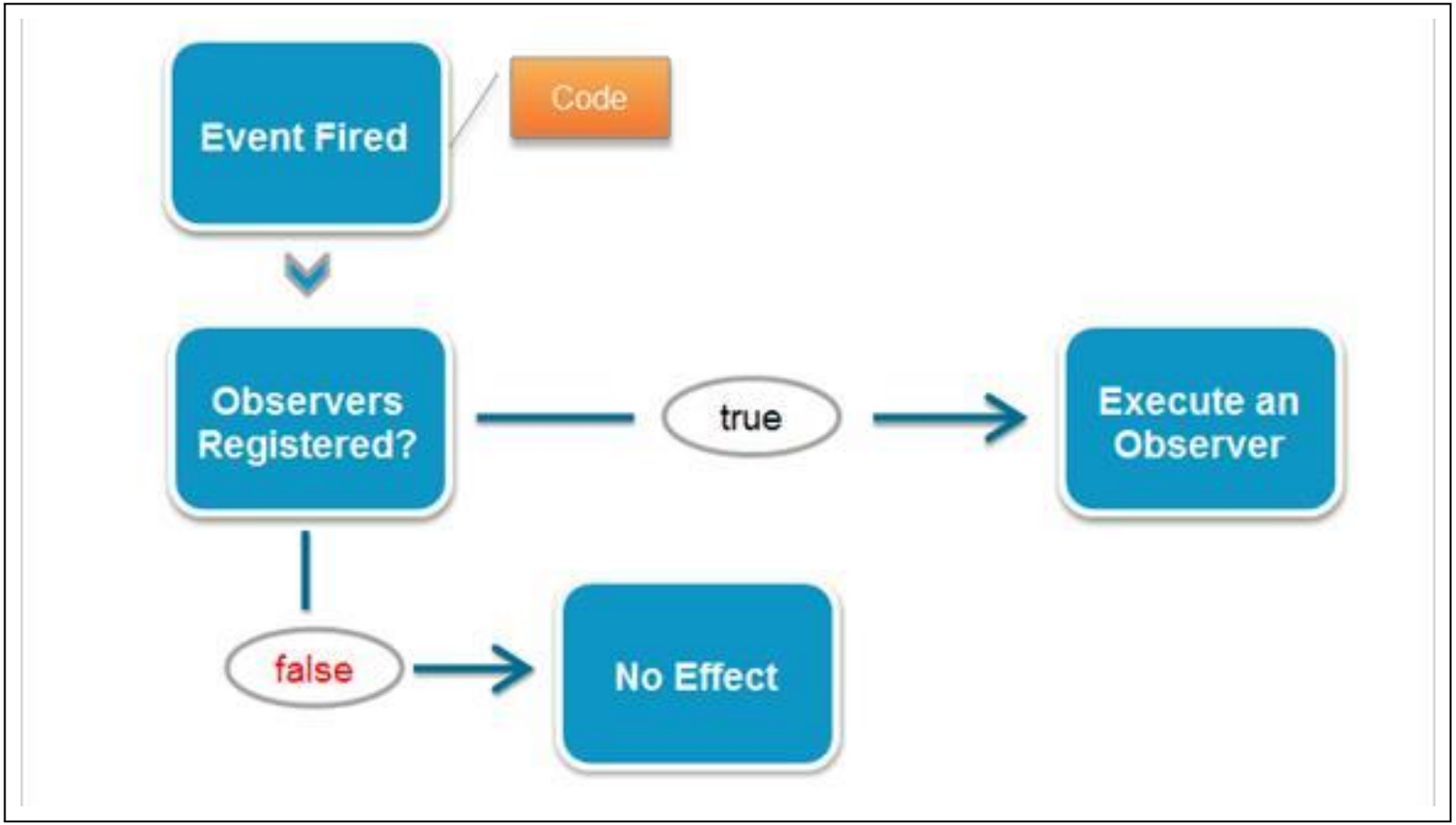
- Commonly used in applications to handle external actions or input
- Each action is interpreted as an event.
- Part of the event-observer pattern
- Events trigger objects to notify their observers of any state changes, usually by calling one of their methods

How to fire a **Event**

```php
<?php
class Foo
{
    protected $eventManager;

    public function __construct(
        \Magento\Framework\Event\ManagerInterface $eventManager
    ) {
        $this->eventManager = $eventManager;
    }

    public function Bar()
    {
        // something before
        $number = new stdClass();
        $number->value = rand(1000, 9999);

        // call the event
        $this->eventManager->dispatch('foo_bar_prepare_number', ['number' => $number]);

        // something after
        $number->value = $number->value*10;

        return $number;
    }
}
```

Events and Observers

Schema



Smile
OPEN SOURCE SOLUTIONS

Events and Observers

Observer

- Class that implements **\Magento\Framework\Event\ObserverInterface**
- One required method **execute**
- with one required parameter: \Magento\Framework\Event\Observer **$observer**
- Registered in **./etc/events.xml** to execute it in all area
- Registered in **./etc/[area]/events.xml** to execute it only in a specific area
- Good Practices : Put the class in the **Observer** folder of your module

Smile
OPEN SOURCE SOLUTIONS

Events and Observers

How to register a **Observer** on an Event

```
<?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="controller_front_send_response_before">
        <observer
            name="foo-bar-send-response-log"
            instance="Foo\Bar\Observer\SendResponseLogOutput"
            shared="true"
        />
    </event>
</config>
```

Smile
OPEN SOURCE SOLUTIONS

Events and Observers

Event-Observer practice (see 04-event) 1/6

- In the previous module **Training/Helloworld**
- Prepare a new observer **PredispatchLogUrl**
- Register this observer
  - On **frontend**
  - On the event **controller_action_predispatch**
- Log the current **path info** in the **./var/log/debug.log** file

Event-Observer practice (see 04-event) 2/6

- New File: **./Training/Helloworld/Observer/PredispatchLogUrl.php**

```php
<?php
/**
 * Magento 2 Training Project
 * Module Training/Helloworld
 */
namespace Training\Helloworld\Observer;

use Magento\Framework\Event\ObserverInterface;

/**
 * Observer PredispatchLogUrl
 *
 * @author    Laurent MINGUET <lamin@smile.fr>
 * @copyright 2016 Smile
 */
class PredispatchLogUrl implements ObserverInterface
{

}
```

Event-Observer practice (see 04-event) 3/6

- Method: **execute**

```php
<?php
    /**
     * Log the url
     *
     * @param \Magento\Framework\Event\Observer $observer Magento Observer Object
     *
     * @return void
     */
    public function execute(\Magento\Framework\Event\Observer $observer)
    {
        \\ @Todo
    }
```

Event-Observer practice (see 04-event) 4/6

- New File:
  ./Training/Helloworld/etc/frontend/events.xml

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

    <event name="controller_action_predispatch">
        <observer
            name="training-helloworld-predispatch-log"
            instance="Training\Helloworld\Observer\PredispatchLogUrl"
            shared="true"
        />
    </event>
</config>
```

Event-Observer practice (see 04-event) 5/6

- Inject the logger in the observer

```php
<?php
    /**
     * Object logger
     * @var \Psr\Log\LoggerInterface
     */
    protected $logger;

    /**
     * PredispatchLogUrl constructor.
     *
     * @param \Psr\Log\LoggerInterface $log Magento Logger Interface
     *
     * @return PredispatchLogUrl
     */
    public function __construct(\Psr\Log\LoggerInterface $log)
    {
        $this->logger = $log;
    }
```

Event-Observer practice (see 04-event) 6/6

- Log the pathinfo

```php
<?php
    /**
     * Log the url
     *
     * @param \Magento\Framework\Event\Observer $observer Magento Observer Object
     *
     * @return void
     */
    public function execute(\Magento\Framework\Event\Observer $observer)
    {
        $url = $observer->getEvent()->getRequest()->getPathInfo();
        $this->logger->debug('Current Url : '.$url);
    }
```

Plan

Smile
OPEN SOURCE SOLUTIONS

Plugins

Definition

- Used to extend/change the behavior of any public method within a Magento class
- Change the behavior of the original class, but not the class itself
- Can not be used on final classes, final methods, and classes created without dependency injection.
- Allows you to execute specific code before, after, or around a public method

Plugins

Definition

- One original method can have lots of plugins, executed in the choosen order
- A plugin class does not implement any interface or extend any class
- Declared in the **di.xml** files
- Good Practices : Put the class in the **Plugin** folder of your module

Plugins

How to register a **Plugin** on a **Class** in the **di.xml** file

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

    <!-- PLUGIN -->
    <type name="Magento\Catalog\Model\Product">
        <plugin
            name="Foo-Bar-Product-Plugin"
            type="Foo\Bar\Plugin\Product"
            sortOrder="10"
            disabled="false"
        />
    </type>

</config>
```

Smile
OPEN SOURCE SOLUTIONS

Before-Listener method

```php
<?php
    /**
     * Set the name - Before
     *
     * @param \Magento\Catalog\Model\Product $subject Product Model
     * @param string                         $value   value to set
     *
     * @return [string]
     */
    public function beforeSetName(\Magento\Catalog\Model\Product $subject, $value)
    {
        $value = mb_strtolower($value);

        return [$value];
    }
```

After-Listener method

```php
<?php
    /**
     * Get the name - After
     *
     * @param \Magento\Catalog\Model\Product $subject Product Model
     * @param string                         $result  Value to return
     *
     * @return string
     */
    public function afterGetName(\Magento\Catalog\Model\Product $subject, $result)
    {
        return 'My Name Is '.$result;
    }
```

Around-Listener method

```php
<?php
    /**
     * around setLastname method
     *
     * @param \Magento\Catalog\Model\Product $subject Product Model
     * @param \Closure                       $proceed The next plugin or method
     * @param string                         $name    The name to use
     *
     * @return string
     */
    public function aroundSetName(\Magento\Catalog\Model\Product $subject, \Closure $proceed, $name)
    {
        // something before
        $name = mb_strtoupper($name);

        $result = $proceed($name);

        if ($result) {
            // something after
        }

        return $result;
    }
```

Plugins

Plugin practice (see 05-plugin) 1/4

- In the previous module **Training/Helloworld**
- Prepare a new plugin on the Customer Data model: **\Magento\Customer\Model\Data\Customer**
- Add a plugin before the **setFirstname** method to transform the value in Title Case

Plugin practice (see 05-plugin) 2/4

- New File: **./Training/Helloworld/Plugin/Model/-Data/Customer.php**

```php
<?php
/**
 * Magento 2 Training Project
 * Module Training/Helloworld
 */
namespace Training\Helloworld\Plugin\Model\Data;

/**
 * Plugin Customer
 *
 * @author    Laurent MINGUET <lamin@smile.fr>
 * @copyright 2016 Smile
 */
class Customer
{

}
```

Plugin practice (see 05-plugin) 3/4

- Add a plugin before the **setFirstname** method

```php
<?php
    /**
     * Modify the first name
     *
     * @param \Magento\Customer\Model\Data\Customer $subject Customer Model
     * @param string                                $value   value parameter
     *
     * @return array
     * @SuppressWarnings(PHPMD.UnusedFormalParameter)
     */
    public function beforeSetFirstname(\Magento\Customer\Model\Data\Customer $subject, $value)
    {
        $value = mb_convert_case($value,  MB_CASE_TITLE);

        return [$value];
    }
```

Plugin practice (see 05-plugin) 4/4

- Declare the plugin in the **./etc/di.xml** file

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

    <!-- PLUGIN -->
    <type name="Magento\Customer\Model\Data\Customer">
        <plugin
            name="training-helloworld-customer-plugin"
            type="Training\Helloworld\Plugin\Model\Data\Customer"
            sortOrder="10"
        />
    </type>
</config>
```

Plan

Smile
OPEN SOURCE SOLUTIONS

Rewrites

Definition

- Replace a original Magento class by a specific one, to modify its behavior
- Use the dependency injection declaration in **di.xml**

Rewrites

How to **Rewrite** a class

- In the **./etc/module.xml**, the new module must depends on the original module to rewrite
- In the **./etc/di.xml** or **./etc/[AREA]/di.xml**, a new preference must be added to use the new class
- The new class must extends the original rewrited class or implement the original interface
- Good Practices : Put the class in the **Rewrite** folder of your module
- Good Practices : Always use Plugin instead of Rewrite, if possible

Rewrites

Rewrite practice (see 06-rewrite) 1/5

- In the previous module **Training/Helloworld**
- Prepare a new rewrite on the Catalog Product Model: **\Magento\Catalog\Model\Product**
- Rewrite the **getName** method to add the text " (Hello World)" at the end.

Smile
OPEN SOURCE SOLUTIONS

Rewrite practice (see 06-rewrite) 2/5

- New File:
  ./Training/Helloworld/Rewrite/Model/Product.php

```php
<?php
/**
 * Magento 2 Training Project
 * Module Training/Helloworld
 */
namespace Training\Helloworld\Rewrite\Model;

/**
 * Rewrite \Magento\Catalog\Model\Product
 *
 * @author    Laurent MINGUET <lamin@smile.fr>
 * @copyright 2016 Smile
 */
class Product extends \Magento\Catalog\Model\Product
{

}
```

Rewrites

Rewrite practice (see 06-rewrite) 3/5

- Update the **./etc/module.xml** file to add the dependency

```
<?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="Training_Helloworld" setup_version="1.0.0">
        <sequence>
            <module name="Magento_Catalog" />
        </sequence>
    </module>
</config>
```

Rewrites

Rewrite practice (see 06-rewrite) 4/5

- Update the **./etc/di.xml** file to declare the rewrite

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

    <!-- REWRITE -->
    <preference for="Magento\Catalog\Model\Product" type="Training\Helloworld\Rewrite\Model\Product" />

</config>
```

Smile
OPEN SOURCE SOLUTIONS

Rewrites

Rewrite practice (see 06-rewrite) 5/5

- Rewrite the **getName** method

```php
<?php
    /**
     * Get the name of the product
     *
     * @return string
     */
    public function getName()
    {
        return parent::getName() . ' (Hello World)';
    }
```

Smile
OPEN SOURCE SOLUTIONS

Plan

5 Models

- Model, Resource, Collection, and Entity Manager
- Model - EAV
- Model - Practice
- Api, Data, and Repository
- Web Api
- Setup: install and upgrade
- Practice - Seller - Part 1 - Model / API / Setup

Plan

5 Models

- Model, Resource, Collection, and Entity Manager
- Model - EAV
- Model - Practice
- Api, Data, and Repository
- Web Api
- Setup: install and upgrade
- Practice - Seller - Part 1 - Model / API / Setup

Smile
OPEN SOURCE SOLUTIONS

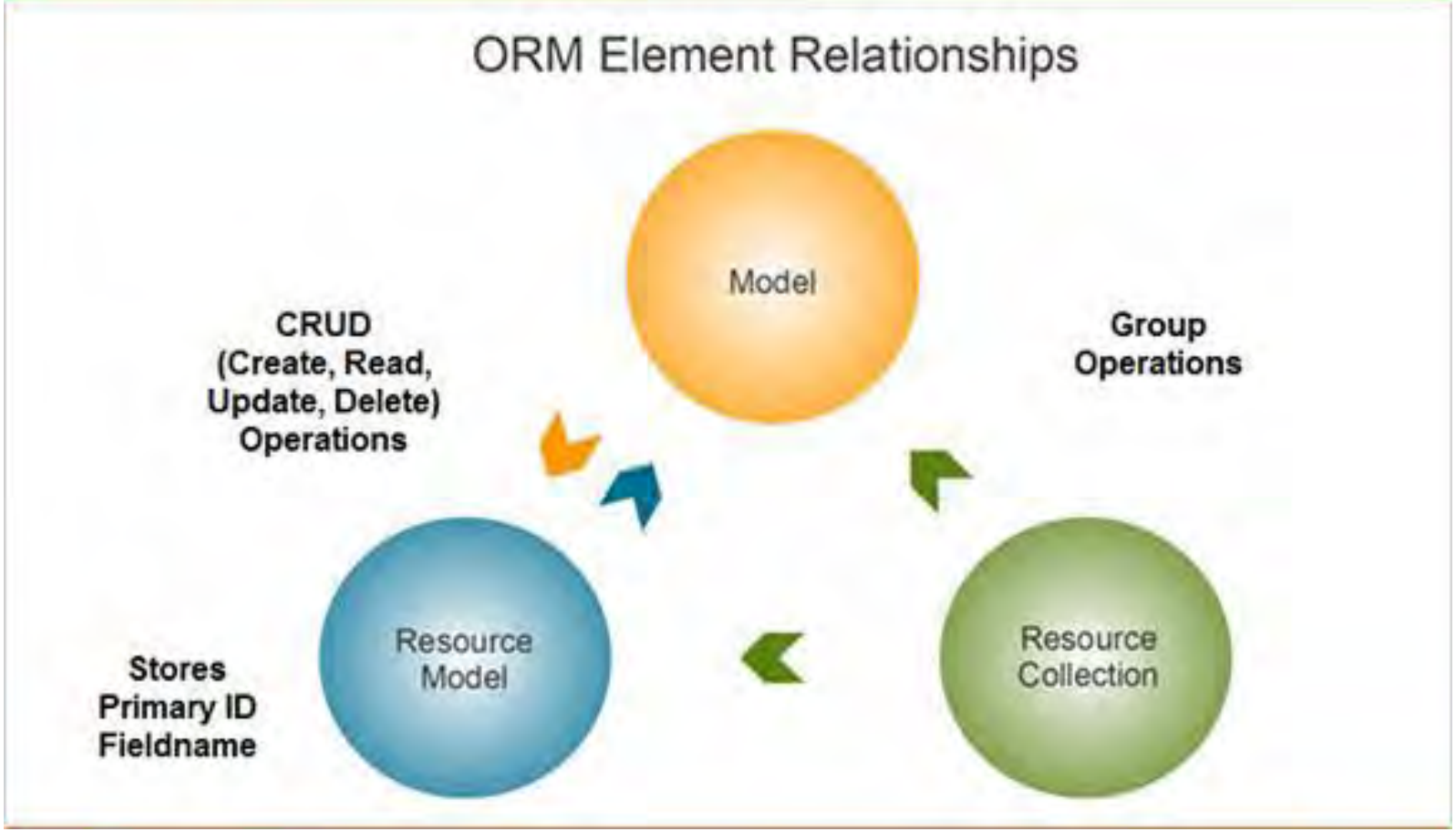
Model, Resource, Collection, and Entity Manager

Magento **Object Relational Mapping** elements

- **Models**: Data + behavior; entities
- **Resource Models**: Data mappers for storage structure (legacy, it uses the entity manager)
- **Collections**: model sets & related functionality, such as filtering, sorting, and paging
- **Entity Manager**: load, save, delete entities
- **Resources**: such as a database connection via adapters

Model, Resource, Collection, and Entity Manager

Magento **Object Relational Mapping** elements



Smile
OPEN SOURCE SOLUTIONS

Model, Resource, Collection, and Entity Manager

- Not all **Models** are ORM entities.
- ORM entity extends **AbstractModel**
- A Model must implement a interface that declares setters and getters for API. Example: \Magento\Cms\Api\Data\BlockInterface

Smile
OPEN SOURCE SOLUTIONS

Model, Resource, Collection, and Entity Manager

Model

- **AbstractModel** provides old legacy CRUD operations (via the Resource Model)
- load(): Read
- save(): Create & Update
- delete(): Delete
- you must not use them, but use directly the entity manager
  (or at least the methods on the resource model)

Smile
OPEN SOURCE SOLUTIONS

Model, Resource, Collection, and Entity Manager

Resource Model

- extends **\Magento\Framework\Model\ResourceModel\Db\AbstractDb**
- has legacy save, delete, load methods, but you must not used them because it does not use the new entity manager
- you must redefine thoses methods to use the entity manager
- can access to the database with the unified **getConnection** method for specific queries that can not be done by the entity manager

Smile
OPEN SOURCE SOLUTIONS

Model, Resource, Collection, and Entity Manager

Entity Manager

- Class **Magento\Framework\EntityManager\EntityManager**
- provide the crud methods (save, delete, load)
- use the metadata tool defined via dependency injection
- can access to the database

Model, Resource, Collection, and Entity Manager

Entity Manager

- It provides automatic events
- the **data interface name** is used automatically for the prefix of the following events:
  - xxxx_save_before
  - xxxx_save_after
  - xxxx_delete_before
  - xxxx_delete_after
  - xxxx_load_before
  - xxxx_load_after
- The method getEntity() can be used to get the current entity in the observer
- look at ->**dispatchEntityEvent** to see them

Smile
OPEN SOURCE SOLUTIONS

Collection filtering with methods **addFieldToFilter** and **addAttributeToFilter**

array('eq' => 'uk')	⇒	WHERE (m.code = 'uk')
array('neq' => 'uk')	⇒	WHERE (m.code != 'uk')
array('like' => 'uk')	⇒	WHERE (m.code like 'uk')
array('nlike' => 'uk')	⇒	WHERE (m.code not like 'uk')
array('is' => 'uk')	⇒	WHERE (m.code is 'uk')
array('in' => array('uk'))	⇒	WHERE (m.code in ('uk'))
array('nin' => array('uk'))	⇒	WHERE (m.code not in ('uk'))
array('notnull' => true)	⇒	WHERE (m.code is not null)
array('null' => true)	⇒	WHERE (m.code is null)
array('gt' => 'uk')	⇒	WHERE (m.code > 'uk')
array('lt' => 'uk')	⇒	WHERE (m.code < 'uk')
array('gteq' => 'uk')	⇒	WHERE (m.code >= 'uk')
array('lteq' => 'uk')	⇒	WHERE (m.code <= 'uk')
array('finset' => array('uk'))	⇒	WHERE (find_in_set('uk', m.code))
array('from' => 'uk', 'to' => 'uk)	⇒	WHERE (m.code >= 'uk' and m.code <= 'uk')

Model, Resource, Collection, and Entity Manager

Models and Cache

- When saving / deleting a model, it has to purge the corresponding blocs and pages in the cache
- The model must:
  - implement \Magento\Framework\DataObject\IdentityInterface
  - have a constant **CACHE_TAG** (that can be used in the blocks for example)
  - have a property **_cacheTag**
  - have the method **getIdentities** that return the list of the concerned tags
  - update the property **_cacheTag** after save and delete action to add the cache tag with the id suffix

Smile
OPEN SOURCE SOLUTIONS

Plan

Smile
OPEN SOURCE SOLUTIONS

Entity Attribute Value Schema

- EAV = Entity Attribute Value
  - "Flat" mode (common):
    - 1 object type <=>1 table
    - object attributes <=>columns of the tables
  - EAV: split objects and theirs attributes into distinct tables
- Benefit of EAV
  - **Flexibility**: an object structure can be changed without modifying tables structure
- **Drawbacks**
  - **Slowness**
  - Concentration of data in a small number of tables
  - Difficulty to develop
    - Magento API makes it easier to deal with EAV

Smile
OPEN SOURCE SOLUTIONS

EAV Schema

Product Flat Table

id	sku	name	description	price	manufacturer
1	pro-1	Debian	Debian CD of the last version	2	Debian
2	rasp-pi	Rasperry Pi	Ultra low cost computer	25	R.P. Inc

Category Flat Table

id	name	url_key	level
1	Software	software	2
2	Hardware	hardware	2

EAV Entity Type Table

id	type
1	product
2	category
3	order
4	invoice

EAV Attribute Value Table

id	entity_id	type_id	attribute	value
1	1	1	sku	pro-1
2	1	1	name	Debian
3	1	1	price	2
4	1	2	name	Software
5	1	2	url_key	software
6	1	2	level	2
7	2	2	name	Hardware
8	2	2	url_key	hardware
9	2	2	level	2
4	2	1	sku	rasp-pi
5	2	1	name	Rasperry Pi
6	2	1	price	25

Magento EAV Implementations

- Magento's EAV optimizations
  - Objects splited by class
    - catalog_product_entity
    - customer_entity
    - customer_address_entity
    - ...
  - Attributes splited by types
    - customer_address_entity_int
    - customer_address_entity_varchar
    - customer_address_entity_text
    - ...
- **Shorter tables (faster)**
- **Needs a lot of joins**

Smile
OPEN SOURCE SOLUTIONS

EAV in Magento

- EAV used for the most important objects in Magento
  - Product
  - Category
  - Customer
  - Customer address
  - ...
- The Resource Model of an EAV Model extends \Magento\Eav\Model\Entity\AbstractEntity
- Specific entity manager EAV operators are used

  \Magento\Framework\EntityManager\Operation\Read\ReadAttributes

Smile
OPEN SOURCE SOLUTIONS

Attributes in Magento

- Definition
  - Caracteristics of a model
  - Each model share the same attributes, each instance can have different values for a given attribute
- Type and values
  - An attribute has a given type, close to mysql one
  - Types are:
    - static (directly in the main entity table. Ex: product's sku)
    - int
    - decimal
    - varchar
    - textarea (can contain HTML blocks)
    - datetime
    - select (use of an association table of ID ->value[s])
    - multiselect (use the same association table)

Smile
OPEN SOURCE SOLUTIONS

Product attributes

- Product attributes are stored using EAV
- Some "standards attributes":
  - catalog_product_entity_datetime
  - catalog_product_entity_decimal
  - catalog_product_entity_int
  - catalog_product_entity_text
  - catalog_product_entity_varchar
- Some product specifics ones:
  - catalog_product_entity_gallery
  - catalog_product_entity_media_gallery
  - catalog_product_entity_tier_price
- Use the following Mysql query to see all the tables

```
show tables like 'catalog_product_entity%';
```

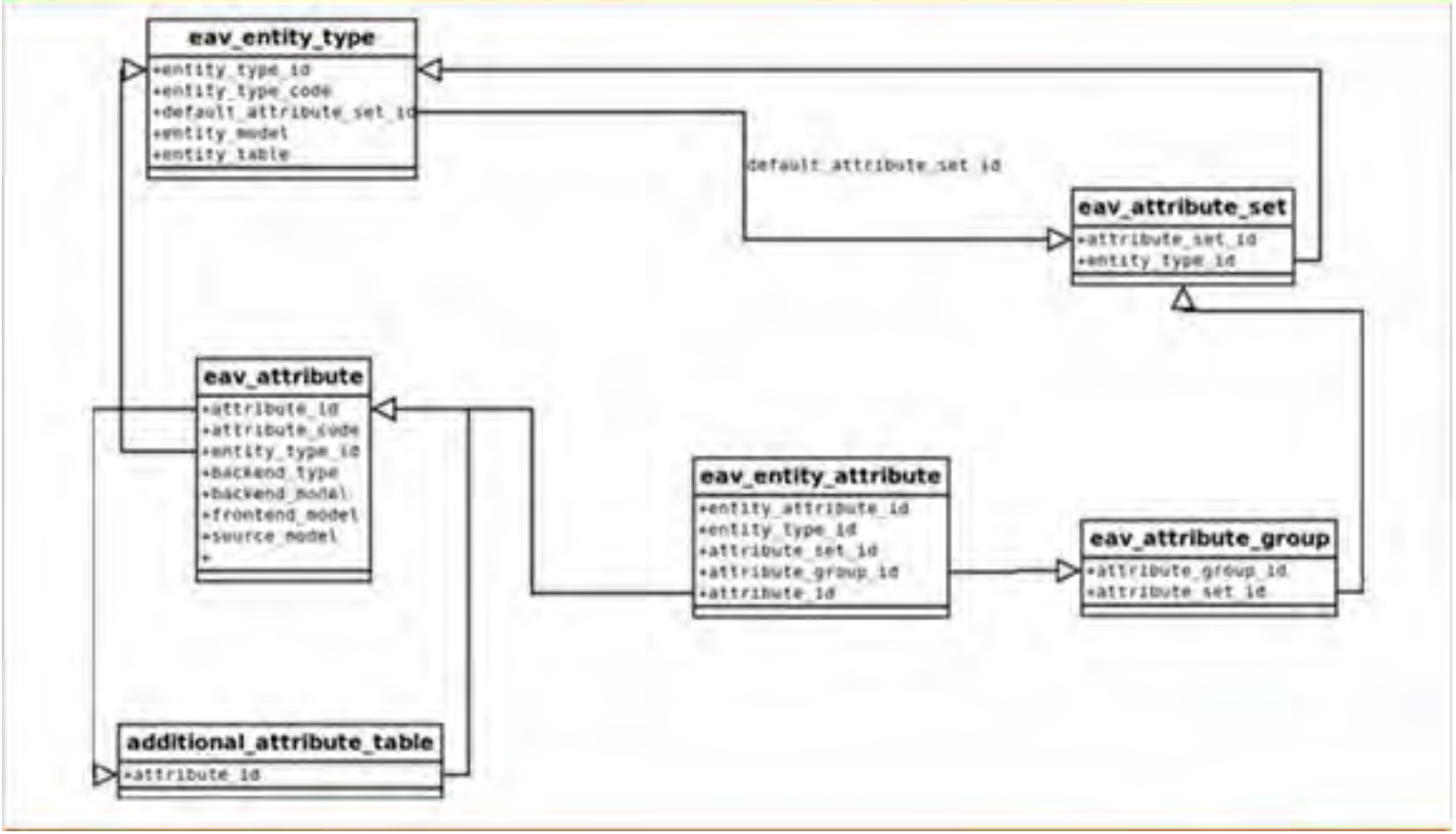
Smile
OPEN SOURCE SOLUTIONS

Product attribute sets

- An **attribute set** represents a kind of product
- Defines all available attributes for an EAV entity
  - So all the models do not share the same attributes, true for products, customer, etc.
- Exemple:
  - T-shirt attribute set contains (among others) a color and a size attribute
  - Book attribute set have no color nor size, but have page_nb and author

Product attribute sets

Tables



Smile
OPEN SOURCE SOLUTIONS

Plan

Smile
OPEN SOURCE SOLUTIONS

Model - Practice

Model practice (see 07-model) 1/6

- In the previous module **Training/Helloworld**
- Create a new frontend controler product/categories
- Ask for the Product and Category Collection factories in the constructor
- Get all the products where the name contains "bag"
- Get the name of all the categories where thoses products are (use only one collection)
- For each product, display its name and the list the name of its associated categories
  (Go fast : do not use the layout / block / template system)

Smile
OPEN SOURCE SOLUTIONS

Model - Practice

Model practice (see 07-model) 2/6

- New File:
  ./Training/Helloworld/Controller/Product/Categories.ph

```php
<?php
/**
 * Magento 2 Training Project
 * Module Training/Helloworld
 */
namespace Training\Helloworld\Controller\Product;

/**
 * Action: Product/Categories
 *
 * @author    Laurent MINGUET <lamin@smile.fr>
 * @copyright 2016 Smile
 */
class Categories extends \Magento\Framework\App\Action\Action
{
    /**
     * Execute the action
     *
     * @return void
     */
    public function execute()
    {
        $this->getResponse()->appendBody('@todo');
    }
}
```

Smile
OPEN SOURCE SOLUTIONS

Model - Practice

Model practice (see 07-model) 3/6

- Ask for the Product and Category Collection factories in the constructor

```php
<?php
    /**
     * @var \Magento\Catalog\Model\ResourceModel\Product\CollectionFactory
     */
    protected $productCollectionFactory;

    /**
     * @var \Magento\Catalog\Model\ResourceModel\Category\CollectionFactory
     */
    protected $categoryCollectionFactory;

    /**
     * @param \Magento\Framework\App\Action\Context                          $context
     * @param \Magento\Catalog\Model\ResourceModel\Product\CollectionFactory  $productCollectionFactory
     * @param \Magento\Catalog\Model\ResourceModel\Category\CollectionFactory $categoryCollectionFactory
     *
     * @return Index
     */
    public function __construct(
        \Magento\Framework\App\Action\Context $context,
        \Magento\Catalog\Model\ResourceModel\Product\CollectionFactory $productCollectionFactory,
        \Magento\Catalog\Model\ResourceModel\Category\CollectionFactory $categoryCollectionFactory
    ) {
        parent::__construct($context);

        $this->productCollectionFactory = $productCollectionFactory;
        $this->categoryCollectionFactory = $categoryCollectionFactory;
    }
```

Smile
OPEN SOURCE SOLUTIONS

Model - Practice

Model practice (see 07-model) 4/6

- Get all the products where the name contains "bag"

```php
<?php
    /**
     * Execute the action
     *
     * @return void
     */
    public function execute()
    {
        /** @var \Magento\Catalog\Model\ResourceModel\Product\Collection $productCollection */
        $productCollection = $this->productCollectionFactory->create();
        $productCollection
            ->addAttributeToFilter('name', array('like' => '%bag%'))
            ->addCategoryIds()
            ->load();


        ...
    }
```

Smile
OPEN SOURCE SOLUTIONS

Model - Practice

Model practice (see 07-model) 5/6

- Get the name of all the categories where thoses products are

```php
<?php
    /**
     * Execute the action
     *
     * @return void
     */
    public function execute()
    {
        ...

        $categoryIds = [];
        foreach ($productCollection as $product) {
            /** @var \Magento\Catalog\Model\Product $product */
            $categoryIds = array_merge($categoryIds, $product->getCategoryIds());
        }
        $categoryIds = array_unique($categoryIds);

        /** @var \Magento\Catalog\Model\ResourceModel\Category\Collection $catCollection */
        $catCollection = $this->categoryCollectionFactory->create();
        $catCollection
            ->addAttributeToFilter('entity_id', array('in' => $categoryIds))
            ->addAttributeToSelect('name')
            ->load();
        $categories = [];
        foreach ($catCollection as $category) {
            /** @var $category \Magento\Catalog\Model\Category */
            $categories[$category->getId()] = $category->getName();
        }
```

Smile
OPEN SOURCE SOLUTIONS

Model - Practice

Model practice (see 07-model) 6/6

- For each product, display its name and the list the name of its associated categories

```php
<?php
    /**
     * Execute the action
     *
     * @return void
     */
    public function execute()
    {
        ...

        $html = '<ul>';
        foreach ($productCollection as $product) {
            $html.= '<li>';
            $html.= $product->getId().' => '.$product->getSku().' => '.$product->getName();
            $html.= '<ul>';
            foreach ($product->getCategoryIds() as $categoryId) {
                $html.= '<li>'.$categoryId.' => '.$categories[$categoryId].'</li>';
            }
            $html.= '</ul>';
            $html.= '</li>';
        }
        $html.= '</ul>';

        $this->getResponse()->appendBody($html);
    }
```

Smile
OPEN SOURCE SOLUTIONS

Plan

Smile
OPEN SOURCE SOLUTIONS

Api, Data, and Repository

- API = all the interfaces in the api folder of a module
- Modules only communicate through the API
- Never use a class of an external module.
- Use only the API interfaces and its declared methods
- Never use a method of an implemented interface that is not in this interface

Api, Data, and Repository

3 main **API groups**

- Data Api
- Repository API
- Operationnal API

Api, Data, and Repository

Data Api

- In the **./Api/Data/** folder
- Interfaces named **XxxxInterface**
- Only access to the datas of a object via setter and getter
- No CRUD operations
- Good practice: add constants for the Table name and for the columns name
  Example: \Magento\Customer\Api\Data\CustomerInterface

Smile
OPEN SOURCE SOLUTIONS

Api, Data, and Repository

Repository Api

- In the **./Api/** folder
- Interfaces named **XxxxRepositoryInterface**
- Contain CRUD operations, with methods like **getById**, **getList**, **save**, and **deleteById**
  Example: \Magento\Customer\Api\CustomerRepositoryInterface
- Good practice: Associate a **Repository Interface** to a **Search Result Interface**
  Example:
  \Magento\Customer\Api\Data\CustomerSearchResultsInterface

Smile
OPEN SOURCE SOLUTIONS

Api, Data, and Repository

Operationnal Api

- In the **./Api/** folder
- Interfaces named **XxxxManagementInterface**
- Drives business operations supplied by this module

  Example: \Magento\Customer\Api\AccountManagementInterface

Smile
OPEN SOURCE SOLUTIONS

Api, Data, and Repository

- Ability to customize based on the documentation
- Better decoupling
- Minimizing conflicts
- Ability to rely on the interface, not on implementation
- Magento upgrade are much safer to execute without anything braking

Api, Data, and Repository

Repository Implementation Example:
\Magento\Store\Model\StoreRepository

```php
<?php
    /**
     * Retrieve store by id
     *
     * @param int $id
     * @return \Magento\Store\Api\Data\StoreInterface
     * @throws NoSuchEntityException
     */
    public function getById($id)
    {
        if (isset($this->entitiesById[$id])) {
            return $this->entitiesById[$id];
        }
        $store = $this->storeFactory->create();
        $store->load($id);
        if ($store->getId() === null) {
            throw new NoSuchEntityException(__('Requested store is not found'));
        }
        $this->entitiesById[$id] = $store;
        $this->entities[$store->getCode()] = $store;
        return $store;
    }
```

Smile
OPEN SOURCE SOLUTIONS

Api, Data, and Repository

Repository Implementation Example:
\Magento\Store\Model\StoreRepository

```php
<?php
    /**
     * Retrieve store by code
     *
     * @param string $code
     * @return \Magento\Store\Api\Data\StoreInterface
     * @throws NoSuchEntityException
     */
    public function get($code)
    {
        if (isset($this->entities[$code])) {
            return $this->entities[$code];
        }
        $store = $this->storeFactory->create();
        $store->load($code, 'code');
        if ($store->getId() === null) {
            throw new NoSuchEntityException(__('Requested store is not found'));
        }
        $this->entities[$code] = $store;
        $this->entitiesById[$store->getId()] = $store;
        return $store;
    }
```

Smile
OPEN SOURCE SOLUTIONS

Api, Data, and Repository

Repository Implementation Example:
\Magento\Cms\Model\BlockRepository

```php
<?php
    /**
     * Save Block data
     *
     * @param \Magento\Cms\Api\Data\BlockInterface $block
     * @return Block
     * @throws CouldNotSaveException
     */
    public function save(Data\BlockInterface $block)
    {
        $storeId = $this->storeManager->getStore()->getId();
        $block->setStoreId($storeId);
        try {
            $this->resource->save($block);
        } catch (\Exception $exception) {
            throw new CouldNotSaveException(__($exception->getMessage()));
        }
        return $block;
    }
```

Smile
OPEN SOURCE SOLUTIONS

Api, Data, and Repository

Repository Implementation Example:
\Magento\Cms\Model\BlockRepository

```php
<?php
    /**
     * Delete Block
     *
     * @param \Magento\Cms\Api\Data\BlockInterface $block
     * @return bool
     * @throws CouldNotDeleteException
     */
    public function delete(Data\BlockInterface $block)
    {
        try {
            $this->resource->delete($block);
        } catch (\Exception $exception) {
            throw new CouldNotDeleteException(__($exception->getMessage()));
        }
        return true;
    }
```

Smile
OPEN SOURCE SOLUTIONS

Repository Implementation Example: **\Magento\Cms\Model\BlockRepository** 1/4

```php
<?php
    /**
     * Load Block data collection by given search criteria
     *
     * @SuppressWarnings(PHPMD.CyclomaticComplexity)
     * @SuppressWarnings(PHPMD.NPathComplexity)
     * @param \Magento\Framework\Api\SearchCriteriaInterface $criteria
     * @return \Magento\Cms\Model\ResourceModel\Block\Collection
     */
    public function getList(\Magento\Framework\Api\SearchCriteriaInterface $criteria)
    {
        $searchResults = $this->searchResultsFactory->create();
        $searchResults->setSearchCriteria($criteria);

        ...

        return $searchResults;
    }
```

Repository Implementation Example: **\Magento\Cms\Model\BlockRepository** 2/4

```php
<?php
    ...

    $collection = $this->blockCollectionFactory->create();
    foreach ($criteria->getFilterGroups() as $filterGroup) {
        foreach ($filterGroup->getFilters() as $filter) {
            if ($filter->getField() === 'store_id') {
                $collection->addStoreFilter($filter->getValue(), false);
                continue;
            }
            $condition = $filter->getConditionType() ?: 'eq';
            $collection->addFieldToFilter($filter->getField(), [$condition => $filter->getValue()]);
        }
    }
    $searchResults->setTotalCount($collection->getSize());

    ...
```

Repository Implementation Example: \Magento\Cms\Model\BlockRepository 3/4

```php
<?php
        ...

        $sortOrders = $criteria->getSortOrders();
        if ($sortOrders) {
            foreach ($sortOrders as $sortOrder) {
                $collection->addOrder(
                    $sortOrder->getField(),
                    ($sortOrder->getDirection() == SortOrder::SORT_ASC) ? 'ASC' : 'DESC'
                );
            }
        }

        ...
```

Repository Implementation Example: \Magento\Cms\Model\BlockRepository 4/4

```php
<?php
    ...

    $collection->setCurPage($criteria->getCurrentPage());
    $collection->setPageSize($criteria->getPageSize());
    $blocks = [];
    /** @var Block $blockModel */
    foreach ($collection as $blockModel) {
        $blockData = $this->dataBlockFactory->create();
        $this->dataObjectHelper->populateWithArray(
            $blockData,
            $blockModel->getData(),
            'Magento\Cms\Api\Data\BlockInterface'
        );
        $blocks[] = $this->dataObjectProcessor->buildOutputDataArray(
            $blockData,
            'Magento\Cms\Api\Data\BlockInterface'
        );
    }
    $searchResults->setItems($blocks);

    ...
```

How to build a Search Criteria ? Example **\Magento\Customer\Model\GroupManagement**

```php
<?php
    public function getLoggedInGroups()
    {
        $notLoggedInFilter[] = $this->filterBuilder
            ->setField(GroupInterface::ID)
            ->setConditionType('neq')
            ->setValue(self::NOT_LOGGED_IN_ID)
            ->create();
        $groupAll[] = $this->filterBuilder
            ->setField(GroupInterface::ID)
            ->setConditionType('neq')
            ->setValue(self::CUST_GROUP_ALL)
            ->create();
        $searchCriteria = $this->searchCriteriaBuilder
            ->addFilters($notLoggedInFilter)
            ->addFilters($groupAll)
            ->create();
        return $this->groupRepository->getList($searchCriteria)->getItems();
    }
```

Api, Data, and Repository

Filter Builder methods

- setField
- setConditionType
- setValue
- create

Api, Data, and Repository

Search Criteria Builder methods

- addFilter
- addFilters
- setFilterGroups
- addSortOrder
- setSortOrders
- setPageSize
- setCurrentPage
- create

Api, Data, and Repository

Search Criteria methods

- getFilterGroups
- setFilterGroups
- getSortOrders
- setSortOrders
- getPageSize
- setPageSize
- getCurrentPage
- setCurrentPage

Api, Data, and Repository

API practice (see 08-api) 1/5

- In the previous module **Training/Helloworld**
- Create a new frontend controler product/search
- Ask for the following objects factories in the constructor
  - ProductRepositoryInterface
  - SearchCriteriaBuilder
  - FilterBuilder
  - SortOrderBuilder
- Get the first 6 products, ordering by name desc, with:
  - description like %comfortable%
  - name like %bruno%

Smile
OPEN SOURCE SOLUTIONS

API practice (see 17-api) 2/5

- New file **./Controller/Product/Search.php**

```php
<?php
namespace Training\Helloworld\Controller\Product;

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

    /**
     * Execute the action
     *
     * @return void
     */
    public function execute()
    {
        // @todo
    }

}
```

API practice (see 17-api) 3/5

- Ask for objects in the constructor

```php
<?php

    protected $productRepository;
    protected $searchCriteriaBuilder;
    protected $filterBuilder;
    protected $sortOrderBuilder;

    public function __construct(
        \Magento\Framework\App\Action\Context $context,
        \Magento\Catalog\Api\ProductRepositoryInterface $productRepository,
        \Magento\Framework\Api\SearchCriteriaBuilder $searchCriteriaBuilder,
        \Magento\Framework\Api\FilterBuilder $filterBuilder,
        \Magento\Framework\Api\SortOrderBuilder $sortOrderBuilder
    ) {
        parent::__construct($context);

        $this->productRepository = $productRepository;
        $this->searchCriteriaBuilder = $searchCriteriaBuilder;
        $this->filterBuilder = $filterBuilder;
        $this->sortOrderBuilder = $sortOrderBuilder;
    }
```

API practice (see 17-api) 4/5

- Get the products list

```php
<?php
    protected function getProductList()
    {
        $filterDesc[] = $this->filterBuilder
            ->setField('description')
            ->setConditionType('like')
            ->setValue('%comfortable%')
            ->create();

        $filterName[] = $this->filterBuilder
            ->setField('name')
            ->setConditionType('like')
            ->setValue('%Bruno%')
            ->create();

        $sortOrder = $this->sortOrderBuilder
            ->setField('name')
            ->setDirection(\Magento\Framework\Api\SortOrder::SORT_DESC)
            ->create();

        $searchCriteria = $this->searchCriteriaBuilder
            ->addFilters($filterDesc)
            ->addFilters($filterName)
            ->addSortOrder($sortOrder)
            ->setPageSize(6)
            ->setCurrentPage(1)
            ->create();

        return $this->productRepository->getList($searchCriteria)->getItems();
    }
```

Smile
OPEN SOURCE SOLUTIONS

API practice (see 17-api) 5/5

- Display the result

```php
<?php
    /**
     * Execute the action
     *
     * @return void
     */
    public function execute()
    {
        $products = $this->getProductList();

        foreach ($products as $product) {
            $this->outputProduct($product);
        }
    }

    /**
     * output a product
     *
     * @param \Magento\Catalog\Api\Data\ProductInterface $product product to display
     *
     * @return void
     */
    protected function outputProduct(\Magento\Catalog\Api\Data\ProductInterface $product)
    {
        $this->getResponse()->appendBody(
            $product->getSku().' => '.$product->getName().'<br />'
        );
    }
```

Plan

Smile
OPEN SOURCE SOLUTIONS

Web Api

- Allows exposure of the **Module API** through the Web API
- The **webapi.xml** file of each module defines how the Module API will be exposed
- Specific areas **webapi_rest** and **webapi_soap** can be used for specific DI
- Step of the process:
  - Call to a **URL**
  - Use the **webapi.xml** file to know the corresponding API and Resources
  - Check the **ACL** for the asked Resources
  - Define interface implementations with specific **di.xml** file
  - Call the API method and return the result

Smile
OPEN SOURCE SOLUTIONS

Web Api

webapi.xml example:

```xml
<?xml version="1.0"?>
<routes
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="urn:magento:module:Magento_Webapi:etc/webapi.xsd">

    <route url="/V1/products" method="POST">
        <service class="Magento\Catalog\Api\ProductRepositoryInterface" method="save"/>
        <resources>
            <resource ref="Magento_Catalog::products" />
        </resources>
    </route>
</routes>
```

- **route[url]**: the corresponding Web API URL to use
- **route[method]**: the http method to use
- **service[class]**: the API interface that corresponds to this url and http method
- **service[method]**: the API method that corresponds to this url and http method
- **resources**: the list of the needed resources for ACL

Smile
OPEN SOURCE SOLUTIONS

Web Api

REST Webservice API
See the example file **./09-apiweb/../rest.php**

- Create a admin user with the acls you want to use
- Make a first POST request to
  ./rest/V1/integration/admin/token
  with the **username** and **password** information in json format
  ⇒ it will return a token that must be used in all the other requests with the header "Authorization: Bearer TOKEN"
- Then make your other request

Smile
OPEN SOURCE SOLUTIONS

Web Api

SOAP Webservice API
See the example file **./09-apiweb/../soap.php**

- Init a SOAP token in System >Integration
- Init a Zend Soap Client with the good WSDL:
  ⇒ http://magento2.lxc/soap?wsdl&services=[module][interface][version]
  ⇒ http://magento2.lxc/soap?wsdl&services=catalogProductRepositoryV1
  And the good options (see example) !
- Call the function you need:
  ⇒ [module][interface][Version][Method]
  ⇒ catalogProductRepositoryV1Get
- The function parameters are exactly the same as defined in the PHPDoc of the interface

Smile
OPEN SOURCE SOLUTIONS

Plan

5 Models

- Model, Resource, Collection, and Entity Manager
- Model - EAV
- Model - Practice
- Api, Data, and Repository
- Web Api
- Setup: install and upgrade
- Practice - Seller - Part 1 - Model / API / Setup

Smile
OPEN SOURCE SOLUTIONS

Setup: install and upgrade

- In the **setup** folder of each module
- 4 setup files:
  - **InstallSchema**
  - **UpgradeSchema**
  - **InstallData**
  - **UpgradeData**
- Version of the module and sequence order in the **./etc/module.xml** file

```
<?xml version="1.0"?>
<config xmlns:xsi="..." xsi:noNamespaceSchemaLocation="...">
    <module name="Magento_Sales" setup_version="2.0.1">
        <sequence>
            <module name="Magento_Rule"/>
            <module name="Magento_Catalog"/>
            <module name="Magento_Customer"/>
            <module name="Magento_Payment"/>
            <module name="Magento_SalesSequence"/>
        </sequence>
    </module>
</config>
```

- Processed modules version are registered in the **setup_module** table

Smile
OPEN SOURCE SOLUTIONS

Setup: install and upgrade

InstallSchema

- Implements **InstallSchemaInterface**
- Must contains only modifications on the database schema
- Executed only once during the first install of the module

Smile
OPEN SOURCE SOLUTIONS

Setup: install and upgrade

InstallSchema.php

```php
<?php
namespace Magento\Catalog\Setup;

use Magento\Framework\Setup\InstallSchemaInterface;
use Magento\Framework\Setup\ModuleContextInterface;
use Magento\Framework\Setup\SchemaSetupInterface;

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

        ...

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

Smile
OPEN SOURCE SOLUTIONS

Setup: install and upgrade

UpgradeSchema

- Implements **UpgradeSchemaInterface**
- Must contains only modifications on the database schema
- Run after an install and upon subsequent upgrades
- One class for all the version updates, with test on the current version of the module

Setup: install and upgrade

UpgradeSchema.php

```php
<?php
namespace Magento\Catalog\Setup;

use Magento\Framework\Setup\UpgradeSchemaInterface;
use Magento\Framework\Setup\ModuleContextInterface;
use Magento\Framework\Setup\SchemaSetupInterface;

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

        if (version_compare($context->getVersion(), '2.0.1', '<')) {
            ...
        }

        if (version_compare($context->getVersion(), '2.0.2', '<')) {
            ...
        }

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

Smile
OPEN SOURCE SOLUTIONS

Setup: install and upgrade

InstallData

- Implements **InstallDataInterface**
- Must contains insertion / modifications of datas
- Run after Schema setups, executed only once during the first install of the module

Setup: install and upgrade

InstallData.php

```php
<?php
namespace Magento\Catalog\Setup;

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

class InstallData implements InstallDataInterface
{
    public function install(ModuleDataSetupInterface $setup, ModuleContextInterface $context)
    {
        ...
    }
}
```

Smile
OPEN SOURCE SOLUTIONS

Setup: install and upgrade

UpgradeData

- Implements **UpgradeDataInterface**
- Must contains insertion / modifications of datas
- Run after an install and Schema setups and upon subsequent upgrades
- One class for all the version updates, with test on the current version of the module

Setup: install and upgrade

UpgradeData.php

```php
<?php
namespace Magento\Catalog\Setup;

use Magento\Framework\Setup\UpgradeDataInterface;
use Magento\Framework\Setup\ModuleContextInterface;
use Magento\Framework\Setup\ModuleDataSetupInterface;

class UpgradeData implements UpgradeDataInterface
{
    public function upgrade(ModuleDataSetupInterface $setup, ModuleContextInterface $context)
    {
        if (version_compare($context->getVersion(), '2.0.1') < 0) {
            ...
        }

        if (version_compare($context->getVersion(), '2.0.2') < 0) {
            ...
        }
    }
}
```

Smile
OPEN SOURCE SOLUTIONS

Plan

5 Models

- Model, Resource, Collection, and Entity Manager
- Model - EAV
- Model - Practice
- Api, Data, and Repository
- Web Api
- Setup: install and upgrade
- Practice - Seller - Part 1 - Model / API / Setup

Smile
OPEN SOURCE SOLUTIONS

Practice - Seller - Part 1 - Model / API / Setup

Seller Module (see 10-seller-part1)

- In a new module **Training/Seller**
- Create API, Model, Resource Model, Collection and Setup to manage new Seller entity
- The mysql table will be named **training_seller**
- It will have 5 fields:
  - seller_id (varchar 64, primary key)
  - identifier (varchar 64, required)
  - name (varchar 255, required)
  - created_at (datetime, automatic on save)
  - updated_at (datetime, automatic on save)

Practice - Seller - Part 1 - Model / API / Setup

New **Module** structure

- etc/module.xml
- registration.php
- composer.json (for module publishing)
- README.md (for module publishing)

Practice - Seller - Part 1 - Model / API / Setup

File **Api/Data/SellerInterface.php**

- Describe how the model will work
- Must have getter and setter methods
- getSellerId, getCreatedAt and getUpdatedAt can return a null value because they are not yet defined on a new object
- the PhpDoc is **very** important, to generated automatically the Soap WSDL
- Best Practice: add constants for table name and fields name

Practice - Seller - Part 1 - Model / API / Setup

File **Api/SellerRepositoryInterface.php**

- Describe what the repository will expose
- getById will take an integer and will return a SellerInterface
- getByIdentifier will take an string and will return a SellerInterface
- getList will take a search criteria, and will return a SearchResult (see after)
- save will take a SellerInterface and will return the saved SellerInterface
- deleteById will take an integer and will return true if the seller has been deleted
- deleteByIdentifier will take an string and will return true if the seller has been deleted
- do not forget the exceptions NoSuchEntityException and CouldNotSaveException

Smile
OPEN SOURCE SOLUTIONS

Practice - Seller - Part 1 - Model / API / Setup

File **Api/Data/SellerSearchResultsInterface.php**

- Describe the type of entity that will be returned by the search result of the repository method getList
- Must extends Magento\Framework\Api\SearchResultsInterface
- Must define the param type of the method setItems to SellerInterface[]
- Must define the return type of the method getItems to SellerInterface[]

Smile
OPEN SOURCE SOLUTIONS

Practice - Seller - Part 1 - Model / API / Setup

File **Model/Seller.php**

- Must extends Magento\Framework\Model\AbstractModel
- Must implements Magento\Framework\DataObject\IdentityInterface
- Must implements Training\Seller\Api\Data\SellerInterface
- The protected method _construct must call the method _init to link the model to the resource model
- The public method getIdentities must be implemented for cache usage, with the constant CACHE_TAG and the protected property _cacheTag
- The public method getSellerId must use the native method getId
- The public method setSellerId must use the native method setId

Smile
OPEN SOURCE SOLUTIONS

Practice - Seller - Part 1 - Model / API / Setup

File **Model/ResourceModel/TraitResource.php**

- Implements generic behaviours to use the new entity manager in a resource model
- It can be used in any resource model
- The public method getConnection allows to get the good mysql connection linked to the object
- The public method loadWithEntityManager allows to load an object
- The public method saveWithEntityManager allows to save an object
- The public method deleteWithEntityManager allows to delete an object

Smile
OPEN SOURCE SOLUTIONS

Practice - Seller - Part 1 - Model / API / Setup

File **Model/ResourceModel/Seller.php** 1/2

- Must extends Magento\Framework\Model\ResourceModel\Db\AbstractDb
- Must use Training\Seller\Model\ResourceModel\TraitResource
- The public method __construct must call the trait method constructTrait
- The public method __construct must ask for Magento\Framework\Stdlib\DateTime\DateTime
- The protected method _construct must call the method _init to link the resource model to the database

Smile
OPEN SOURCE SOLUTIONS

Practice - Seller - Part 1 - Model / API / Setup

File **Model/ResourceModel/Seller.php** 2/2

- The public method load must use the trait method loadWithEntityManager
- The public method save must use the trait method saveWithEntityManager
- The public method delete must use the trait method deleteWithEntityManager
- The protected method _beforeSave can be used to update the fields created_at and updated_at
- The public method deleteIds can be implemented to manage mass delete (used after for the Back-Office actions)

Smile
OPEN SOURCE SOLUTIONS

Practice - Seller - Part 1 - Model / API / Setup

File **Model/ResourceModel/Seller/Collection.php**

- The protected method _construct must call the method _init to link the collection to the model and to the resource model
- Can implement the method toOptionArray to automatically generate an array that can be used for select fields

Practice - Seller - Part 1 - Model / API / Setup

File **Model/Repository/AbstractRepository.php**

- Implements generic behaviours for repositories
- It can be used in any repository that has to manage a flat model
- The protected method setIdentifierFieldName allows you to define if the model has a identifier field (like code, sku, ...)
- The protected method getEntityById allows to load an entity by its id
- The protected method getEntityByIdentifier allows to load an entity by its identifier
- The protected method getEntities allows to load a list of entities, regarding to a search criteria
- The protected method saveEntity allows to save an entity
- The protected method deleteEntity allows to delete an entity

Smile
OPEN SOURCE SOLUTIONS

Practice - Seller - Part 1 - Model / API / Setup

File **Model/Repository/Seller.php** 1/2

- Must implements Training\Seller\Api\SellerRepositoryInterface
- Must extends Training\Seller\Model\Repository\AbstractRepository
- Must extends the public method __construct to :
  - Give the seller model factory (auto generated object manager)
  - Give the seller resource model
  - Give the seller search result interface
  - Set the object identifier with the protected method setIdentifierFieldName

Smile
OPEN SOURCE SOLUTIONS

Practice - Seller - Part 1 - Model / API / Setup

File **Model/Repository/Seller.php** 2/2

- The public method getById will use getEntityById
- The public method getByIdentifier will use getEntityByIdentifier
- The public method getList will use getEntities
- The public method save will use saveEntity
- The public method deleteById will use getEntityById and deleteEntity
- The public method deleteByIdentifier will use getEntityByIdentifier and deleteEntity

Smile
OPEN SOURCE SOLUTIONS

Practice - Seller - Part 1 - Model / API / Setup

File **etc/di.xml**

- Defines the classes to use for the 3 new API interfaces
- Defines the repository to use for the new seller model
- Defines the metadata of the new seller model for the entity manager
- Defines the hydrator tool to use for the new seller model

Practice - Seller - Part 1 - Model / API / Setup

File **etc/events.xml**

- Defines the legacy observers (allows to use the old methods loadBefore and others)

Practice - Seller - Part 1 - Model / API / Setup

File **etc/acl.xml**

- Defines the new resource "Training_Seller::manage" that an user must have to be able to use the new webservices

Practice - Seller - Part 1 - Model / API / Setup

File **etc/webapi.xml**

- Defines how the web api will be exposed
  - GET /V1/seller/id/:objectId => getById
  - GET /V1/seller/identifier/:objectIdentifier => getByIdentifier
  - GET /V1/seller/ => getList
  - POST /V1/seller/ => save
  - DELETE /V1/seller/id/:objectId => deleteById
  - DELETE /V1/seller/identifier/:objectIdentifier => deleteByIdentifier

Practice - Seller - Part 1 - Model / API / Setup

File **Setup/InstallSchema.php**

- Defines the new table trainint_seller, with :
  - an auto-increment primary key on the seller_id field
  - an unique index on the identifier field

Practice - Seller - Part 1 - Model / API / Setup

File **Setup/InstallData.php**

- Ask for the Seller Model Factory in the constructor
- Create a "main" seller

Practice - Seller - Part 1 - Model / API / Setup

File **_extra/scripts/../rest.php**

- Test the web api in rest mode

File **_extra/scripts/../soap.php**

- Test the web api in soap mode

File **_extra/scripts/../full.php**

- Create 100 sellers (for testing)

Plan

Plan

Smile
OPEN SOURCE SOLUTIONS

Routing

- **Routing** converts a request URL into a style Magento can handle, and then finds the class that will be able to process it.
- A Magento-style URL consists of 3 parts + parameters: catalog/product/view/id/5
  - catalog: The **frontname** of the module (declared in ./etc/[AREA]/routes.xml)
  - product: The **name of the group** of actions (the folders in ./Controller)
  - view: The **name of the action** (the file in ./Controller/GROUP/)
  - /id/5: The **parameters** id=5

Smile
OPEN SOURCE SOLUTIONS

Routing

The **routing** process:

- Defining all available routers \Magento\Framework\App\RouterList
- Searching the routeur that will
  - Understand the asked URL
  - Convert the URL to a Magento-style URL
  - Parse the request parameters
  - Identifying the controller class that will process the url
- Executing the identified controller class
- See \Magento\Framework\App\FrontController::dispatch)

Smile
OPEN SOURCE SOLUTIONS

Routing

Routers:

- Implements **\Magento\Framework\App\RouterInterface** with method **match**
- Magento\Backend\App\Router
  Match a asked **magento-style url** with a real backend controller class (it is the main backend router)
- Magento\Framework\App\Router\Base
  Match a asked **magento-style url** with a real frontend controller class (it is the main frontend router)
- Magento\UrlRewrite\Controller\Router
  Use the **url_rewrite** mysql table to match a asked url with a magento-style url.

```
select * from url_rewrite where entity_type='product'
```

- Magento\Cms\Controller\Router
  Match a asked url with an existing CMS page
- Magento\Framework\App\Router\DefaultRouter

Smile
OPEN SOURCE SOLUTIONS

Plan

Smile
OPEN SOURCE SOLUTIONS

Controller

- A controller class can only process a **single action**
- Frontend Controller:
  - Extends **\Magento\Framework\App\Action\Action**
  - Contains the method **execute()** to execute the action
- Backend Controller:
  - Extends **\Magento\Backend\App\Action**
  - Contains the method **execute()** to execute the action
  - Contains the method **_isAllowed()** to protect the action

Smile
OPEN SOURCE SOLUTIONS

Controller

Backend controller

- Extends **\Magento\Backend\App\Action**
  - Empty class, not usefull...
- Extends **\Magento\Backend\App\AbstractAction**
  - Ask for the needed object to manage a back Controller, like
    Autorization, Auth, Current Helper, Backend Url Manager, Form Key validator, Admin Session, ...
- Extends **\Magento\Framework\App\Action\Action**
  - Ask for the needed object to manage a front Controller, like
    Object Manager, Event Manager, Frontend Url Manager, View Manager, Redirect Manager, Message Manager, ...
- Extends **\Magento\Framework\App\Action\AbstractAction**
  - Ask for the needed object to manage a classic Controller, like

Smile
OPEN SOURCE SOLUTIONS

Plan

Smile
OPEN SOURCE SOLUTIONS

Practice - Seller - Part 2 - Routeur / Controller

Seller Module (see 11-seller-part2)

- In the module **Training/Seller**
- Create new frontend action seller/seller/index to display the list of the sellers
- Create new frontend action seller/seller/view to display a seller from an identifier
- Create new backend action training_seller/seller/index to display the info about a seller
- Create a new router to analyse the urls /sellers.html and /seller/[identifier].html

Smile
OPEN SOURCE SOLUTIONS

Practice - Seller - Part 2 - Routeur / Controller

File **etc/frontend/routes.xml**

- Define the frontname to use for the frontend controllers of the module : seller

File **etc/adminhtml/routes.xml**

- Define the frontname to use for the backend controllers of the module : training_seller

File **etc/adminhtml/menu.xml**

- Define the new entry Training Seller in the admin menu

Practice - Seller - Part 2 - Routeur / Controller

File **Controller/Seller/AbstractAction.php**

- Generic behaviors for the seller actions (like asking for the seller repository)

File **Controller/Seller/Index.php**

- Display the list of the sellers, using the repository

File **Controller/Seller/View.php**

- Display a specific seller, using the repository

Practice - Seller - Part 2 - Routeur / Controller

File **Controller/Adminhtml/Seller/AbstractAction.php**

- Generic behaviors for the seller actions (like asking for the seller model factory)
- Implement the protected method _isAllowed to use the resource acl Training_Seller::manage

File **Controller/Adminhtml/Seller/Index.php**

- Display the infos about the main seller

Practice - Seller - Part 2 - Routeur / Controller

File **Controller/Routeur.php**

- New router to manage the urls /sellers.html and /seller/[identifier].html
- Need to use the action factory
- Must implement the public method match, that must return an action, if matched.

Practice - Seller - Part 2 - Routeur / Controller

File **etc/frontend/di.xml**

- Add the new router to the router list

Plan

Smile
OPEN SOURCE SOLUTIONS

View and Layout

Templates (phtml)

- Handle HTML, Javascript, and some PHP
- Always executed by a Block
- In the folder **./view/[AREA]/templates/** of each module

View and Layout

Blocks (php)

- Allow you to move reusable functionnality from template files into classes
- The same block can be assign to different templates
- Extends the class **\Magento\Framework\View\Element\AbstractBlock**
- Implements the interface **\Magento\Framework\View\Element\BlockInterface**
- In the folder **./Block/** of each module
- Be carefull: specific blocks for frontend or backend
  - \Magento\Framework\View\Element\Template
  - \Magento\Backend\Block\Template

Smile
OPEN SOURCE SOLUTIONS

View and Layout

Focus on **AbstractBlock** class, method **toHtml**

```php
<?php
    public function toHtml()
    {
        $this->_eventManager->dispatch('view_block_abstract_to_html_before', ['block' => $this]);
        if ($this->_scopeConfig->getValue(
            'advanced/modules_disable_output/' . $this->getModuleName(),
            \Magento\Store\Model\ScopeInterface::SCOPE_STORE
        )) {
            return '';
        }

        $html = $this->_loadCache();
        if ($html === false) {
            if ($this->hasData('translate_inline')) {
                $this->inlineTranslation->suspend($this->getData('translate_inline'));
            }

            $this->_beforeToHtml();
            $html = $this->_toHtml();
            $this->_saveCache($html);

            if ($this->hasData('translate_inline')) {
                $this->inlineTranslation->resume();
            }
        }
        $html = $this->_afterToHtml($html);

        return $html;
    }
```

Smile
OPEN SOURCE SOLUTIONS

Blocks and Cache

- Block Cache: 3 properties to init in the php constructor
  - **cache_lifetime**: the lifetime of the cache in second
  - **cache_key**: the key of the block cache
  - **cache_tags**: the list of the concerned object tags of this block
- FPC Cache
  - The block must implement \Magento\Framework\DataObject\IdentityInterface
  - The block must implement the method **getIdentities** that return the list of the concerned object tags of this block
- Good Practice: use the method **getIdentities** for the value of the **cache_tags** property

View and Layout

Layouts (xml)

- Allow to define how a page will be rendered, by specify the blocks and the templates to use
- One XML layout file per action
- 2 possible root nodes:
  - **page**: renders a complete html page
  - **layout**: renders only a section of a html page for the response.
- 2 principale part's types:
  - **block**
  - **container** :
    - contains others blocks and containers
    - renders all its children
    - does not display anything directly if no children
- In the folder **./view/[AREA]/layout/** of each module

Smile
OPEN SOURCE SOLUTIONS

View and Layout

Layouts / Root node **page**
Can have 4 different sub-root nodes:

- **html**
  - sub node **attribute**,
    with "name" and "value" attributes

Smile
OPEN SOURCE SOLUTIONS

View and Layout

Layouts / Root node **page**

Can have 4 different sub-root nodes:

- **head**
  - sub node **attribute**,
    with "name" and "value" attributes
  - sub node **css**,
    with "src" attribute
  - sub node **script**,
    with "src" attribute
  - sub node **link**,
    with "src", "defer", and "ie_condition" attributes
  - sub node **remove**,
    with "src" attribute
  - sub node **meta**,
    with "name" and "content" attributes
  - sub node **title**

Smile
OPEN SOURCE SOLUTIONS

View and Layout

Layouts / Root node **page**

Can have 4 different sub-root nodes:

- **body**
  - sub node **attribute**,
    with "name" and "value" attributes
  - sub node **container**,
    with "name", "htmlTag", "htmlClass", "htmlId", "label" attributes
  - sub node **block**,
    with "name" attribute and others that depend on the block type
  - sub node **referenceContainer**,
    with "name", "display", "remove", ... attributes
  - sub node **referenceBlock**,
    with "name", "display", "remove" attributes
  - sub node **move**,
    with "element", "destination", "before", "after" attributes

Smile
OPEN SOURCE SOLUTIONS

View and Layout

Layouts / Root node **page**
Can have 4 different sub-root nodes:

- **update**, with "handle" attribute
  - Allow to define on which main layout (handle) the current layout is based on.
  - if no **update** node, the main layout used is the default one.
  - Example: Magento_Checkout::checkout_cart_configure.xml is based on Magento_Catalog::catalog_product_view.xml

Smile
OPEN SOURCE SOLUTIONS

View and Layout

Page **Layouts**

- Define how the page will be defined globally
- Use only containers
- defined in the module **Magento_theme**

View and Layout

Frontend Page **Layouts**

- **empty**:
  ./view/base/page_layout/empty.xml
- **1 column**:
  ./view/frontend/page_layout/1column.xml
- **2 columns-left**:
  ./view/frontend/page_layout/2columns-left.xml
- **2 columns-right**:
  ./view/frontend/page_layout/2column-right.xml
- **3 columns**:
  ./view/frontend/page_layout/3columns.xml

View and Layout

Backend Page **Layouts**

- **empty**:
  ./view/adminhtml/page_layout/admin-empty.xml
- **1 column**:
  ./view/adminhtml/page_layout/admin-1column.xml
- **2 columns left**:
  ./view/adminhtml/page_layout/admin-2columns-left.xml
- **login**:
  ./view/adminhtml/page_layout/admin-login.xml

View and Layout

Layout example:
Module **Magento_Customer**, frontend action **forgotpassword**
./view/frontend/layout/customer_account_forgotpassword.xml

```xml
<?xml version="1.0"?>
<page
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    layout="1column"
    xsi:noNamespaceSchemaLocation="urn:magento:framework:View/Layout/etc/page_configuration.xsd">
    <head>
        <title>Forgot Your Password</title>
    </head>
    <body>
        <referenceBlock name="root">
            <action method="setHeaderTitle">
                <argument translate="true" name="title" xsi:type="string">Password forgotten</argument>
            </action>
        </referenceBlock>
        <referenceContainer name="content">
            <block
                class="Magento\Customer\Block\Account\Forgotpassword"
                name="forgotPassword"
                template="form/forgotpassword.phtml">
                <container name="form.additional.info" as="form_additional_info"/>
            </block>
        </referenceContainer>
    </body>
</page>
```

Smile
OPEN SOURCE SOLUTIONS

Plan

Smile
OPEN SOURCE SOLUTIONS

Practice - Seller - Part 3 - Layout / Block / Template

Seller Module (see 12-seller-part3)

- In the module **Training/Seller**
- Use layouts, templates, and blocks for the frontend actions index and view
- The index action must contain the possibility to filter on the name, and to chose the sort order, without paging

Practice - Seller - Part 3 - Layout / Block / Template

File **Helper/Url.php**

- Generic helper to manage the frontend urls
- getSellersUrl: get the url to display the list of the sellers
- getSellerUrl: get the url to display one seller, from its identifier

Practice - Seller - Part 3 - Layout / Block / Template

File **Controller/Seller/AbstractAction.php**

- Updated to ask for other tools via DI
  - FilterBuilder
  - SortOrderBuilder
  - ResultPageFactory
  - Registry

Practice - Seller - Part 3 - Layout / Block / Template

File **Controller/Seller/Index.php**

- The execute method must save the Search Result object to the registry (to use it in the view)
- The execute method must return a Result Page object (to build automatically the view)
- The getSearchCriteria method must be updated to:
  - add the name filter order to the Search Criteria
  - add the sort order to the Search Criteria

Practice - Seller - Part 3 - Layout / Block / Template

File **Controller/Seller/View.php**

- The execute method must save the Seller object to the registry (to use it in the view)
- The execute method must return a Result Page object (to build automatically the view)

Practice - Seller - Part 3 - Layout / Block / Template

File **Block/Seller/AbstractBlock.php**

- Will ask for generic usefull tools : Url Helper and Magento Registry
- Will provide 2 shortcuts for the url methods

Practice - Seller - Part 3 - Layout / Block / Template

File **Block/Seller/Index.php**

- The block depends on all the Sellers (see getIdentities method)
- The getSearchResult method returns the Search Result object
- The getCount method returns the number of founded sellers.
- The getSearchName method returns the value of the name filter
- The getSortOrder method returns the value of the sort order

Practice - Seller - Part 3 - Layout / Block / Template

File **Block/Seller/View.php**

- The block depends on the current Seller object (see getIdentities method)
- The getCurrentSeller method returns the Seller object

Practice - Seller - Part 3 - Layout / Block / Template

File
view/frontend/layout/training_seller_seller_view.xml

- Use the 1column layout to diplay the filter in the left sidebar
- New block "seller.view" in the "content" container
- Always add the module prefix for the phtml template files

Practice - Seller - Part 3 - Layout / Block / Template

File
view/frontend/layout/training_seller_seller_index.xml

- Use the 2columns-left layout to diplay the filter in the left sidebar
- New block "seller.list" in the "content" container
- New block "seller.list.filter" in the "sidebar.main" container
- They can use the same PHP block class
- Always add the module prefix for the phtml template files

Practice - Seller - Part 3 - Layout / Block / Template

File **view/frontend/templates/seller/view.phtml**
File **view/frontend/templates/seller/list.phtml**
File **view/frontend/templates/seller/list/filter.phtml**

- Use the $block object to access to its public methods
- Always use the escapeHtml method to add protection
- Always use the __ function to be i18n ready
- Add /* @escapeNotVerified */ for already secured output

Plan

Smile
OPEN SOURCE SOLUTIONS

Practice - Seller - Part 4 - Layout Update

Seller Module (see 13-seller-part4)

- In the previous module **Training/Seller**
- Add a link "Sellers" in the "header.links" block
- Add on the top of the content of all the frontend pages a block:
  - type: template
  - phtml to use : new header.phtml file
  - parameters: border_color and background_color
- The header.phtml file:
  - display a div that uses border_color and background_color
  - contain only a link "[Sellers list]"
- On the Category page, move this new block on the top of the sidebar
- On the Product page, use different border and background colors
- On the Sellers pages, remove this new block

Smile
OPEN SOURCE SOLUTIONS

Practice - Seller - Part 4 - Layout Update

File **view/frontend/layout/default.xml**

- This file allows to change all the pages of the frontend
- Add a block "training.seller.header.link" on the existing block "header.links"
  - Use the generic Html\Link block type
  - Use the "label" argument to set the label of the link
  - Use the "path" argument to set the path of the link
  - Do not work to use the "translate" property on the label
  - Use the Seller Url helper to get the url automatically
- Add a block "training.seller.content.top" on the existing block "content.top"
  - Use the generic Template block type
  - Set new argument "background_color"
  - Set new argument "border_color"

Smile
OPEN SOURCE SOLUTIONS

Practice - Seller - Part 4 - Layout Update

File **view/frontend/templates/header.phtml**

- Use the $this variable to access to the Magento Template Engine
- Use its "helper" method to get the Seller Url helper
- Use the "getData" method of the block to acces to the colors

Practice - Seller - Part 4 - Layout Update

File **view/frontend/layout/catalog_category_view.xml**

- Move the block "training.seller.content.top" on the top of the container "sidebar.main"

File **view/frontend/layout/catalog_product_view.xml**

- Change the colors of the block "training.seller.content.top"

File **view/frontend/layout/training_seller_seller_index.xml**

- Update the layouts to remove the "training.seller.content.top"

File **view/frontend/layout/training_seller_seller_view.xml**

- Update the layouts to remove the "training.seller.content.top"

Smile
OPEN SOURCE SOLUTIONS

Plan

Smile
OPEN SOURCE SOLUTIONS

Practice - Seller - Part 5 - Admin

Seller Module (see 14-seller-part5)

- In the previous module **Training/Seller**
- Add the following admin actions
  - Show the list of sellers (with advanced magento listing UI component)
  - Row-edit a seller (directly in the list)
  - Mass delete of sellers (directly in the list)
  - Create / Edit a seller (with advanced magento form UI component)
  - Delete a seller (from the form or from the list)

Practice - Seller - Part 5 - Admin

File **Model/ResourceModel/Seller/Grid/Collection.php**

- Needed by the advanced listing UI component
- Must implement the SearchResultInterface interface
- The protected constructor must be overridden to use the DataProvider Document model instead of the Seller model
- The methods getAggregations and setAggregations manage the facet aggregation of the search result
- The methods getSearchCriteria and setSearchCriteria are fake
- The method getTotalCount returns the size of the collection
- The methods setTotalCount and setItems are fake

File **etc/di.xml**

- Needed by the advanced listing UI component
- Add the new Seller Grid Collection to the DataProvider

Practice - Seller - Part 5 - Admin

File **Ui/Component/Listing/SellerActions.php**

- Needed by the advanced listing UI component
- Define the actions to display for each row
- Must extends Magento\Ui\Component\Listing\Columns\Column
- Override the constructor to ask for the URL builder
- The public method prepareDataSource prepares, for each row, the list of the available actions

Practice - Seller - Part 5 - Admin

File **Ui/Component/Form/SellerDataProvider.php**

- Needed by the advanced form UI component
- Define how to get the data to display in the form
- Must extends Magento\Ui\DataProvider\AbstractDataProvider
- Must ask for Magento\Framework\App\Request\DataPersistorInterface to get the data from session if a validation error occurs.
- The public method getData:
  - Get the data from the database
  - Override the data with the data in session (if they exist)

Smile
OPEN SOURCE SOLUTIONS

Practice - Seller - Part 5 - Admin

File
view/adminhtml/ui_component/training_seller_seller_listing.xml
1/5

- Define how the advanced listing UI component will be used
- The main node is **listing**.
- The used sub nodes are:
  - **argument**
  - **dataSource**
  - **listingToolbar**
  - **columns**

Smile
OPEN SOURCE SOLUTIONS

Practice - Seller - Part 5 - Admin

File

view/adminhtml/ui_component/training_seller_seller_listing.xml

2/5

Sub Node **argument**

- The item **js_config** defines the data source to use by the js
- The item **spinner** defines the columns to use
- The item **buttons** defines the buttons to display on the top of the listing

Practice - Seller - Part 5 - Admin

File

view/adminhtml/ui_component/training_seller_seller_listing.xml

3/5

Sub Node **dataSource**

- The item **class** defines the data provider to use
- The item **name** defines the name of the data source. It is linked with the di.xml file to now the name of the collection to use.
- The item **primaryFieldName** defines the name of the db primary key
- The item **requestFieldName** defines the name of the request field for the primary key
- The item **data** defines the js component to use, and how to use it.

Smile
OPEN SOURCE SOLUTIONS

Practice - Seller - Part 5 - Admin

File

view/adminhtml/ui_component/training_seller_seller_listing.xml

4/5

Sub Node **listingToolbar**

- The item **data** allows to define the sticky config (display the toolbar on the top)
- The item **bookmark** allows to enable the bookmark functionnality
- The item **columnsControls** allows to enable the columns controls functionnality
- The item **paging** allows to enable the paging functionnality
- The item **filterSearch** allows to enable the full search functionnality
- The item **filters** allows to enable the filters functionnality
- The item **massaction** allows to define the list of the mass actions

Practice - Seller - Part 5 - Admin

File

view/adminhtml/ui_component/training_seller_seller_listing.xml

5/5

Sub Node **columns**

- The item **data** allows to define the edit inline config
- The item **selectionsColumn** allows to define the field to use for mass action
- The item **column** allows to define the column of each field
  - The item **label** defines the label of the column
  - The item **filter** defines if the column is filterable
  - The item **dataType** defines the type of the column
  - The item **editor** defines the validator for edit inline
  - The item **sorting** defines the default sort
  - The item **sortOrder** defines the order of the columns
- The item **actionsColumn** allows to define the action column

Practice - Seller - Part 5 - Admin

File **view/adminhtml/layout/training_seller_seller_index.xml**

- The page contains only the new seller listing UI component

File **Controller/Adminhtml/Seller/AbstractAction.php**

- Update the constructor to ask for usefull tools like the result page factory

File **Controller/Adminhtml/Seller/Index.php**

- Use the result page factory to generate the page

Practice - Seller - Part 5 - Admin

File **Controller/Adminhtml/Seller/MassDelete.php**

- Method geSellerIds: get the list of the seller ids to delete
- Method execute: delete the ids and redirect to the list
- Use the messageManager property to display a success message

File **Controller/Adminhtml/Seller/InlineEdit.php**

- The output format must be json, the json factory must be asked in the constructor
- Method getResult: prepare the output in json format
- Method execute: save the seller data, only if it is an ajax call

Smile
OPEN SOURCE SOLUTIONS

Practice - Seller - Part 5 - Admin

File **Block/Adminhtml/Seller/Edit/AbstractButton.php**

- Generic behavior to manage buttons on a edit form UI component
- The abstract method getButtonData will return all the button's info
- The public method getObjectId return the current seller_id, with a validation

File **Block/Adminhtml/Seller/Edit/BackButton.php**

- Display the "Back" button, to return to the index action

File **Block/Adminhtml/Seller/Edit/ResetButton.php**

- Display the "Reset" button, to reset the edit form

Smile
OPEN SOURCE SOLUTIONS

Practice - Seller - Part 5 - Admin

File **Block/Adminhtml/Seller/Edit/SaveButton.php**

- Display the "Save" button, to submit the edit form

File **Block/Adminhtml/Seller/Edit/SaveAndContinue-Button.php**

- Display the "Save and Continue" button, to submit the edit form and continue on the edit form

File **Block/Adminhtml/Seller/Edit/DeleteButton.php**

- Display the "Delete" button, to delete the current seller

Practice - Seller - Part 5 - Admin

File

view/adminhtml/ui_component/training_seller_seller_form.xml

1/4

- Define how the advanced from UI component will be used
- The main node is **form**.
- The used sub nodes are:
  - **argument**
  - **dataSource**
  - **fieldset**

Smile
OPEN SOURCE SOLUTIONS

Practice - Seller - Part 5 - Admin

File

view/adminhtml/ui_component/training_seller_seller_form.xml

2/4

Sub Node **argument**

- The item **js_config** defines the data source to use by the js
- The item **config** defines the generic config of the form
- The item **template** defines the form template to use
- The item **buttons** defines the buttons to display on the top of the from

Practice - Seller - Part 5 - Admin

File

view/adminhtml/ui_component/training_seller_seller_form.xml

3/4

Sub Node **dataSource**

- The item **class** defines the data provider to use
- The item **name** defines the name of the data source
- The item **primaryFieldName** defines the name of the db primary key
- The item **requestFieldName** defines the name of the request field for the primary key
- The item **config** defines the submit url to use
- The item **js_config** defines the js component to use

Practice - Seller - Part 5 - Admin

File

view/adminhtml/ui_component/training_seller_seller_form.xml

4/4

Sub Node **fieldset**

- The item **data** allows to define the label of the fieldset
- The item **field** corresponds to a html field
  - The item **sortOrder** defines the display order (int)
  - The item **visible** defines the visibility
  - The item **dataType** defines the type of data
  - The item **label** defines the label to display
  - The item **formElement** defines the type of form element
  - The item **source** defines the source of the data object
  - The item **dataScope** defines the field of the data object
  - The item **validation** defines the field validator

Smile
OPEN SOURCE SOLUTIONS

Practice - Seller - Part 5 - Admin

File **view/adminhtml/layout/training_seller_seller_edit.xml**

- The page contains only the new seller form UI component

File **Controller/Adminhtml/Seller/Edit.php**

- Use the result page factory to generate the page

File **Controller/Adminhtml/Seller/Save.php**

- Use the seller model factory to load the current seller and save the values
- Use the dataPersistor to save the values in the session
- Use the messageManager to save a message in the session
- Use the redirect factory to redirect to the index page

File **Controller/Adminhtml/Seller/Delete.php**

- Use the messageManager to save a message in the session
- Use the redirect factory to redirect to the index page

Plan

7 Others

- Practice - Seller - Part 6 - Upgrade
- Practice - Seller - Part 7 - Customer Attribute
- Practice - Seller - Part 8 - Product Attribute
- Practice - Seller - Part 9 - Extension Attribute
- Create a new type of xml config file
- Create a new type of xml config file - Practice

Plan

7 Others

- Practice - Seller - Part 6 - Upgrade
- Practice - Seller - Part 7 - Customer Attribute
- Practice - Seller - Part 8 - Product Attribute
- Practice - Seller - Part 9 - Extension Attribute
- Create a new type of xml config file
- Create a new type of xml config file - Practice

Practice - Seller - Part 6 - Upgrade

Seller Module (see 15-seller-part6)

- In the previous module **Training/Seller**
- Add a new field description on the seller entity
- Add it to the edit form, using a WYSIWYG field
- Display it on frontend

Practice - Seller - Part 6 - Upgrade

File **Api/Data/SellerInterface.php**

- Add the constant FIELD_DESCRIPTION
- Add the public method getDescription
- Add the public method setDescription

File **Model/Seller.php**

- Implement public method getDescription
- Implement public method setDescription

Practice - Seller - Part 6 - Upgrade

File **Setup/UpgradeSchema.php**

- Add the column "description" to the seller table, if version $< 1.0.1$

File **etc/module.xml**

- Upgrade the setup version to 1.0.1

Practice - Seller - Part 6 - Upgrade

File **view/adminhtml/ui_component/training_seller_seller_form.xml**

- Add the wysiwyg field "description" to the form

File **view/frontend/templates/seller/view.phtml**

- Display the description

Plan

Smile
OPEN SOURCE SOLUTIONS

Practice - Seller - Part 7 - Customer Attribute

Seller Module (see 16-seller-part7)

- In the previous module **Training/Seller**
- Create a new customer attribute that allows to select a seller.

Practice - Seller - Part 7 - Customer Attribute

File **Option/Seller.php**

- New class that will prepare the list of the sellers, for the sources of the new attribute
- Ask for the Seller Collection Factory in the constructor
- The protected method getOptions will prepare the list of the sellers, with a local cache
- You must implement the public method getAllOptions

Practice - Seller - Part 7 - Customer Attribute

File **Setup/UpgradeData.php**

- Add the new customer attribute "training_seller_id", if version $<$ 1.0.2
- The id of the seller will be saved in database in the integer table
- The form field will be a select field
- The list of the items will come from the new Option Seller class
- The new attribute must be added to the adminhtml_customer form
- The EAV config cache must be cleared after each modification

File **etc/module.xml**

- Upgrade the setup version to 1.0.2

Smile
OPEN SOURCE SOLUTIONS

Plan

Practice - Seller - Part 8 - Product Attribute

Seller Module (see 17-seller-part8)

- In the previous module **Training/Seller**
- Create a new global product attribute "Training Sellers" that allows to select sellers
- This new attribute will be available in a new attribute group "Training"
- This new attribute must only be available for simple and configurable bag products
- On the product view, a new tab "Sellers" will be added, to display the list of the sellers, with a link to the page of each seller

Smile
OPEN SOURCE SOLUTIONS

Practice - Seller - Part 8 - Product Attribute

File **Setup/UpgradeData.php**

- Add the new product attribute "training_seller_ids", if version $<$ 1.0.3
- The ids of the selected sellers will be saved in database using the ArrayBackend class in the varchar table
- The form field will be a multiselect field
- The list of the items will come from the Option Seller class
- The new attribute must be added to the "Training" group of the "bag" attribute set
- The EAV config cache must be cleared at the end

File **etc/module.xml**

- Upgrade the setup version to 1.0.3

Smile
OPEN SOURCE SOLUTIONS

Practice - Seller - Part 8 - Product Attribute

File **view/frontend/layout/catalog_product_view.xml**

- Add a new block in the block product.info.details, to add a new tab
- The title of this block will be "Sellers"
- A new type of block will be used: Training\Seller\Block\Product\Sellers
- A specific template file will be used: product/sellers.phtml

Practice - Seller - Part 8 - Product Attribute

File **Helper/Data.php**

- We will use all the search criteria builder classes, and the seller repository
- The public method getProductSellerIds will return the seller ids linked to a product
- The public method getSearchCriteriaOnSellerIds will build a search criteria, filtered on a list of seller ids
- The public method getProductSellers will used thoses 2 methods to get the list of the sellers linked to a product

Practice - Seller - Part 8 - Product Attribute

File **Block/Product/Sellers.php**

- We will use the registry, all the search criteria builder classes, and the seller repository
- The public method getCurrentProduct will return the current product (saved in the registry)
- The public method getProductSellers will use the data helper to get the list of the sellers linked to the current product
- Do not forget to use a local cache in the getProductSellers method
- Do not forget to implement the getIdentities method
- Do not forget the cache configuration of the block

Smile
OPEN SOURCE SOLUTIONS

Practice - Seller - Part 8 - Product Attribute

File **view/frontend/templates/product/sellers.phtml**

- Use the getProductSellers method of the block to get the list of the sellers to display
- For the "no sellers" case, you must have a empty html output to hide the tab
- Do not forget to protect the output with the public method escapeHtml

Plan

Smile
OPEN SOURCE SOLUTIONS

Practice - Seller - Part 9 - Extension Attribute

Seller Module (see 18-seller-part9)

- In the previous module **Training/Seller**
- Create an Extension Attribute for the API, to add the list of the sellers linked to a product, when using the REST or SOAP api.
- Help: look at the catalog inventory module, for the stock_item extension attribute

Practice - Seller - Part 9 - Extension Attribute

File **etc/extension_attributes.xml**

- Add the attribute "sellers" to the list of the extension attributes of the product data interface

File **etc/di.xml**

- Add a plugin on the product model, to load the sellers linked to a product automatically after product loading

Practice - Seller - Part 9 - Extension Attribute

File **Plugin/Model/Product.php**

- Will use the Data Helper to load the sellers linked to a product
- Will use the product extension factory to initialise it if needed
- The public method afterLoad will:
  - Get the extension attributes from the product
  - Prepare them if needed
  - Get the list of the sellers linked to the current product
  - Add the list to the extension attributes
  - Save them to the product

Smile
OPEN SOURCE SOLUTIONS

Plan

Smile
OPEN SOURCE SOLUTIONS

Create a new type of xml config file

How to create a new config file?

- Reader: PHP class that is used to read the xml file
- SchemaLocator: PHP class that encapsulates path to the XSD schema files
- Converter: PHP class that convert XML to PHP array
- Schema: XSD schema file
- Interface: PHP Interface that specifies how the data can be accessed from another module
- Config: PHP Class that implements the PHP Interface, to get access to the config values

Plan

Create a new type of xml config file - Practice

Practice (see 19-config)

- In a new module **Training_Shop**
- Create new XSD schema file **etc/shops.xsd**
  - List of **shop** elements in a main **config** element
  - At least one **shop** element
  - Shop Attribute **code** (required, unique)
  - Shop Attribute **state** (required, restricted to open/close)
  - Shop Attribute **name** (required)
  - Shop Attribute **address** (required)
  - Shop Attribute **city** (required)
- Create a new XML configuration file **etc/shops.xml** that uses it
- Create All the needed php files to use this new config xml file
- Create frontend actions to use thems (without layout/block)

Smile
OPEN SOURCE SOLUTIONS

Create a new type of xml config file - Practice

File **Config/Shop/SchemaLocator.php**

- To specify the path of the **etc/shops.xsd** schema file

File **Config/Shop/Converter.php**

- To convert the XML into a PHP Array

File **Config/Shop/Reader.php**

- To specify the name of the **shops.xml** schema file
- Use the SchemaLocator and the Converter

Create a new type of xml config file - Practice

File **Api/Config/ShopInterface.php**

- To define how the config values will be readable from others modules

File **Config/Shop.php**

- To access to the config values
- Implements **Api/Config/ShopInterface.php**
- Specify the cache key of the config

File **etc/di.xml**

- Specify the Config class to use when asking to the Config interface

Smile
OPEN SOURCE SOLUTIONS

Create a new type of xml config file - Practice

Create action **Index/Index**

- Ask for the Config Interface in the constructor
- Get the list of the shops
- Display the name and code of each shops

Try it: `http://magento2.lxc/shop/index/index`

Create action **Index/View**

- Ask for the Config Interface in the constructor
- Get the asked code in the URL using getRequest method
- Get the asked shop
- Display all the informations about the shop

try it:
`http://magento2.lxc/shop/index/view/code/xxxx`

Smile
OPEN SOURCE SOLUTIONS

Plan