Magento 2.2

Developer Basics

March 16, 2018

Plan

Plan

1. Reminder
   - SOLID Principles
   - PSR
   - Composer
   - GIT
   - Root User, Delivery User, Execution User

Plan

1. Reminder
   - SOLID Principles
   - PSR
   - Composer
   - GIT
   - Root User, Delivery User, Execution User

SOLID Principles

- **S**ingle responsibility principle
  A class should have only a single responsibility

SOLID Principles

- **S**ingle responsibility principle
  A class should have only a single responsibility
- **O**pen / closed principle
  Software entities should be open for extension, but closed for modification.

SOLID Principles

- **S**ingle responsibility principle
  A class should have only a single responsibility
- **O**pen / closed principle
  Software entities should be open for extension, but closed for modification.
- **L**iskov substitution principle
  Ojects in a program should be replaceable with instances of their subtypes without altering the correctness of that program

SOLID Principles

- **S**ingle responsibility principle
  A class should have only a single responsibility
- **O**pen / closed principle
  Software entities should be open for extension, but closed for modification.
- **L**iskov substitution principle
  Ojects in a program should be replaceable with instances of their subtypes without altering the correctness of that program
- **I**nterface segregation principle
  Many client-specific interfaces are better than one general-purpose interface

SOLID Principles

- **S**ingle responsibility principle
  A class should have only a single responsibility
- **O**pen / closed principle
  Software entities should be open for extension, but closed for modification.
- **L**iskov substitution principle
  Ojects in a program should be replaceable with instances of their subtypes without altering the correctness of that program
- **I**nterface segregation principle
  Many client-specific interfaces are better than one general-purpose interface
- **D**ependency inversion principle
  High-level modules should not depend on low-level modules. Both should depend on abstractions. Abstractions should not depend on details. Details should depend on abstractions.

Plan

1 Reminder
- SOLID Principles
- PSR
- Composer
- GIT
- Root User, Delivery User, Execution User

PSR

- **P**HP **S**tandards **R**ecommendations

PSR

- **P**HP **S**tandards **R**ecommendations
- Website: `http://www.php-fig.org/psr/`

PSR

- **P**HP **S**tandards **R**ecommendations
- Website: `http://www.php-fig.org/psr/`
- PSR-1 - Basic Coding Standard

PSR

- **P**HP **S**tandards **R**ecommendations
- Website: `http://www.php-fig.org/psr/`
- PSR-1 - Basic Coding Standard
- PSR-2 - Coding Style Guide

PSR

- **P**HP **S**tandards **R**ecommendations
- Website: `http://www.php-fig.org/psr/`
- PSR-1 - Basic Coding Standard
- PSR-2 - Coding Style Guide
- PSR-3 - Logger Interface

PSR

- **P**HP **S**tandards **R**ecommendations
- Website: `http://www.php-fig.org/psr/`
- PSR-1 - Basic Coding Standard
- PSR-2 - Coding Style Guide
- PSR-3 - Logger Interface
- PSR-4 - Autoloader

Plan

1 Reminder
- SOLID Principles
- PSR
- Composer
- GIT
- Root User, Delivery User, Execution User

Composer

- **Composer** is a tool for dependency management in PHP

Composer

- **Composer** is a tool for dependency management in PHP
- It allows you to declare the libraries your project depends on

Composer

- **Composer** is a tool for dependency management in PHP
- It allows you to declare the libraries your project depends on
- It will manage (install/update) them for you

Composer

- **Composer** is a tool for dependency management in PHP
- It allows you to declare the libraries your project depends on
- It will manage (install/update) them for you
- It handles class autoloading

Composer

- **Composer** is a tool for dependency management in PHP
- It allows you to declare the libraries your project depends on
- It will manage (install/update) them for you
- It handles class autoloading
- Website: `https://getcomposer.org/`

Composer

- **Composer** is a tool for dependency management in PHP
- It allows you to declare the libraries your project depends on
- It will manage (install/update) them for you
- It handles class autoloading
- Website: `https://getcomposer.org/`
- Main Repository: `https://packagist.org/`

Composer

- **composer init** - Init a new project

  Creates the **composer.json** file

Composer

- **composer init** - Init a new project

  Creates the **composer.json** file

- **composer require namespace/library** - Add a library

  Adds the library to the **composer.json** file

  Saves the version of each library in the **composer.lock** file

Composer

- **composer init** - Init a new project

  Creates the **composer.json** file

- **composer require namespace/library** - Add a library

  Adds the library to the **composer.json** file

  Saves the version of each library in the **composer.lock** file

- **composer install** - Install all the libraries

  Uses the **composer.lock** file to determine the version to use for each library

Composer

- **composer init** - Init a new project

  Creates the **composer.json** file

- **composer require namespace/library** - Add a library

  Adds the library to the **composer.json** file

  Saves the version of each library in the **composer.lock** file

- **composer install** - Install all the libraries

  Uses the **composer.lock** file to determine the version to use for each library

- **composer update** - Update all the libraries

  Saves the updated version of each library in the **composer.lock** file

Composer

- The **vendor** directory contains all the PHP libraries downloaded by composer

Composer

- The **vendor** directory contains all the PHP libraries downloaded by composer
- You must commit the **composer.json** file to your VCS

Composer

- The **vendor** directory contains all the PHP libraries downloaded by composer
- You must commit the **composer.json** file to your VCS
- You must commit the **composer.lock** file to your VCS

Composer

- The **vendor** directory contains all the PHP libraries downloaded by composer
- You must commit the **composer.json** file to your VCS
- You must commit the **composer.lock** file to your VCS
- You must ignore the **vendor** directory in your VCS

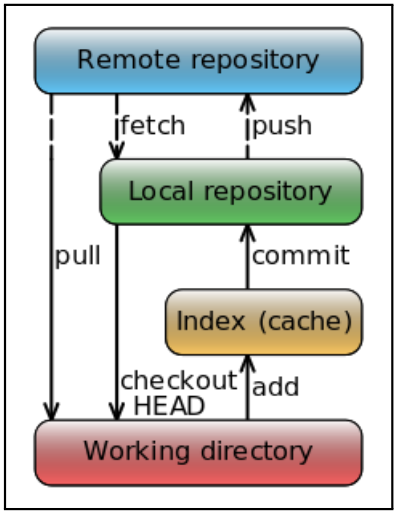
Plan

GIT

- **Git** is a distributed revision control and source code management system

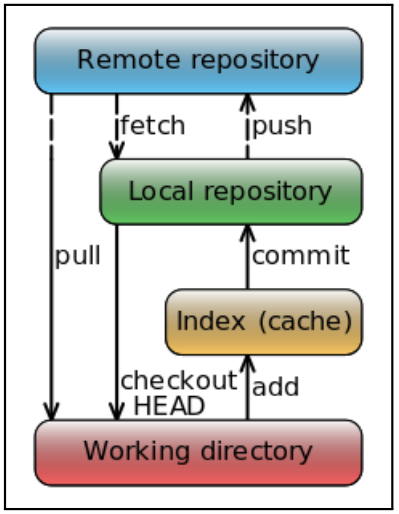
GIT

- **Git** is a distributed revision control and source code management system
- Main commands

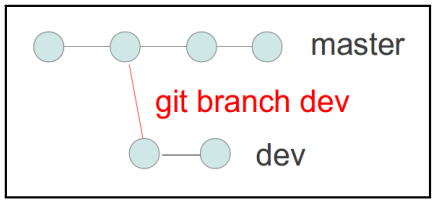


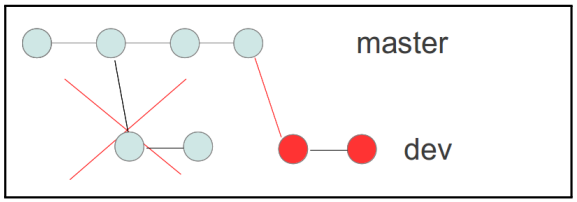
GIT

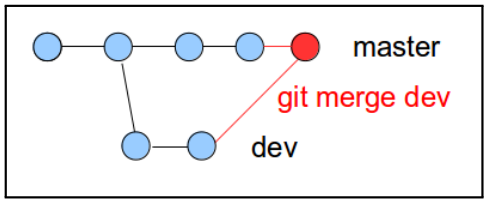
- **Git** is a distributed revision control and source code management system
- Main commands



- Branch / Rebase / Merge







Plan

1 Reminder
- SOLID Principles
- PSR
- Composer
- GIT
- Root User, Delivery User, Execution User

Root User, Delivery User, Execution User

- **Root** user
  Must be used only to install / update the server, or for specific maintenance tasks

Root User, Delivery User, Execution User

- **Root** user
  Must be used only to install / update the server, or for specific maintenance tasks
- **Execution** user - www-data
  Must be used only to execute the web application

Root User, Delivery User, Execution User

- **Root** user
  Must be used only to install / update the server, or for specific maintenance tasks
- **Execution** user - www-data
  Must be used only to execute the web application
- **Delivery** user - smile
  Must be used only to deploy the web application
  Must have sudo right on the execution user
  All the application files must be owned by this user, except some specific files like logs, media, ...

Root User, Delivery User, Execution User

- **Root** user
  Must be used only to install / update the server, or for specific maintenance tasks
- **Execution** user - www-data
  Must be used only to execute the web application
- **Delivery** user - smile
  Must be used only to deploy the web application
  Must have sudo right on the execution user
  All the application files must be owned by this user, except some specific files like logs, media, ...
- **Execute a php file**

```
ssh smile@myserver
sudo -u www-data php myScript.php
```

Plan

2 Magento 2 Fundamentals

- About Magento 2
- Requirements
- Preparing the Training Project (smile - with Ansible)
- Preparing the Training Project (external - Manually)

Plan

About Magento 2

- Most advanced OpenSource eCommerce solution

About Magento 2

- Most advanced OpenSource eCommerce solution
- Based on Zend Framework (1.13 and 2.5)

About Magento 2

- Most advanced OpenSource eCommerce solution
- Based on Zend Framework (1.13 and 2.5)
- Using Magento Framework

About Magento 2

- Most advanced OpenSource eCommerce solution
- Based on Zend Framework (1.13 and 2.5)
- 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

About Magento 2

- Most advanced OpenSource eCommerce solution
- Based on Zend Framework (1.13 and 2.5)
- 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
  - Open Source (community) 2.2.5
    Open Software License 3.0
  - Commerce (enterprise) 2.2.5
    Magento Enterprise Edition License

About Magento 2

- Most advanced OpenSource eCommerce solution
- Based on Zend Framework (1.13 and 2.5)
- 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
  - Open Source (community) 2.2.5
    Open Software License 3.0
  - Commerce (enterprise) 2.2.5
    Magento Enterprise Edition License
- Main Magento's drawback
  - Software complexity and slowness

Plan

Requirements

- Linux (Windows not supported)
- Apache >=2.2 or Nginx >=1.8
- MySQL >=5.6 (Oracle or Percona)
- PHP >7.0.6 | 7.1.0
- Warning, Magento 2.2 does not support PHP 5 anymore
  Source: `http://devdocs.magento.com/guides/v2.2/install-gde/system-requirements-tech.html`

Requirements

- Required 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)

Manual Installation

1. Using composer: `http://devdocs.magento.com/guides/v2.2/install-gde/prereq/integrator_install.html`

Manual Installation

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

System permissions

The web user needs write permissions for the following directories:

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

Be very careful with the `var` directory, do not change permissions on the system's `/var`

Plan

Preparing the Training Project (smile - with Ansible)

- Using Smile Magento 2 Architecture Skeleton

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

Preparing the Training Project (smile - 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 (or php7-cli)
php -v
```

Preparing the Training Project (smile - 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 (smile - with Ansible)

Prerequisites on the host machine

- Install GIT

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

- Configure GIT

  ```
  git config --global user.name "Firstname Lastname"
  git config --global user.email your.email@domain.com
  ```

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

Preparing the Training Project (smile - 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==2.1.6.0
ansible --version
```

- More information:

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

Preparing the Training Project (smile - with Ansible)

Prerequisites on the host machine

- Install the LXC package:

  ```
  sudo apt-get install smile-lxc
  ```

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

Preparing the Training Project (smile - 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 (smile - 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/v2.2/install-gde/prereq/connect-auth.html`

Preparing the Training Project (smile - with Ansible)

Initialize your project (1/2)

- Follow the steps of the **Initialize your project** part

Preparing the Training Project (smile - with Ansible)

Initialize your project (1/2)

- Follow the steps of the **Initialize 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
> magento cloud:          N
> magento edition:        CE
> magento version:        2.2.5
> magento sample data:    Y
> separate architecture:  N
> smile module:           N
> smile user:             [enter]
> confirm:                Y
```

Preparing the Training Project (smile - with Ansible)

Initialize your project (1/2)

- Follow the steps of the **Initialize 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
> magento cloud:          N
> magento edition:        CE
> magento version:        2.2.5
> magento sample data:    Y
> separate architecture:  N
> smile module:           N
> smile user:             [enter]
> confirm:                Y
```

- Step 2: create the LXC

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

Preparing the Training Project (smile - with Ansible)

Initialize your project (1/2)

- Follow the steps of the **Initialize 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
> magento cloud:          N
> magento edition:        CE
> magento version:        2.2.5
> magento sample data:    Y
> separate architecture:  N
> smile module:           N
> smile user:             [enter]
> 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
```

Preparing the Training Project (smile - with Ansible)

Initialize 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`

Preparing the Training Project (smile - with Ansible)

Initialize 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)

Preparing the Training Project (smile - with Ansible)

Some Important Scripts

- Read the **Script list** part

Preparing the Training Project (smile - with Ansible)

Some Important Scripts

- Read the **Script list** part
- **./architecture/scripts/cache-clean.sh** to clean the caches

Preparing the Training Project (smile - 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

Preparing the Training Project (smile - 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 new setup files

Preparing the Training Project (smile - 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 new setup files

Some Important Tools

- Read the **Analyze your code** part

Preparing the Training Project (smile - 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 new setup files

Some Important Tools

- Read the **Analyze your code** part
- **bin/spbuilder** for PHPUnit, CodeSniffer, ...

Magento 2 and Varnish

- You must use Varnish as page cache

```
Stores > Configuration > Advanced > System > Full Page Cache

Access list:  localhost,myfront1,10.0.3.1
Backend host: localhost
Backend port: 82
Grace period: 300
```

Preparing the Training Project (smile - with Ansible)

Magento 2 and Varnish

- You must use Varnish as page cache

```
Stores > Configuration > Advanced > System > Full Page Cache

Access list:  localhost,myfront1,10.0.3.1
Backend host: localhost
Backend port: 82
Grace period: 300
```

- You can compare the VCL provided by magento, with the one in the skeleton

Magento 2 and HTTPS

- You must enable HTTPS

```
Stores > Configuration > General > Web > Base URLs (Secure)

Secure Base URL:                https://magento2.lxc/
Use Secure URLs on Storefront: Yes
Use Secure URLs in Admin:       Yes
```

Magento 2 and HTTPS

- You must enable HTTPS

```
Stores > Configuration > General > Web > Base URLs (Secure)

Secure Base URL:                https://magento2.lxc/
Use Secure URLs on Storefront: Yes
Use Secure URLs in Admin:       Yes
```

- Clean the cache (using the architecture script)

Plan

Preparing the Training Project (external - Manually)

- Using LXC virtualization system
- LXC name : magento2
- System : Debian 9 Stretch
- Composer will be used only from the host
- Never launch it with the root user in the LXC

Preparing the Training Project (external - 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 php-mcrypt php-curl php-intl
sudo apt-get install php-soap php-xsl php-xdebug
sudo apt-get install smile-lxc
```

Preparing the Training Project (external - Manually)

Prerequisites on the host machine

- Install GIT

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

- Configure GIT

```
git config --global user.name "Firstname Lastname"
git config --global user.email your.email@domain.com
```

- Add your public SSH key on your github account

Preparing the Training Project (external - Manually)

Prerequisites on the host machine

- Install Composer:

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

Preparing the Training Project (external - 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/v2.2/install-gde/prereq/connect-auth.html`

Preparing the Training Project (external - Manually)

- Get the Magento 2 sources

```
cd ~/
mkdir projects
cd projects
composer create-project --repository-url=https://repo.magento.com/
    magento/project-community-edition=2.2.5 magento2 --ignore-platform-reqs --no-install
cd magento2
composer config bin-dir ./bin
composer config platform.php "7.0.19"
composer config platform.ext-bcmath "1"
composer config platform.ext-ctype "1"
composer config platform.ext-gd "1"
composer config platform.ext-spl "1"
composer config platform.ext-dom "1"
composer config platform.ext-simplexml "1"
composer config platform.ext-mcrypt "1"
composer config platform.ext-hash "1"
composer config platform.ext-curl "1"
composer config platform.ext-iconv "1"
composer config platform.ext-intl "1"
composer config platform.ext-xsl "1"
composer config platform.ext-mbstring "1"
composer config platform.ext-openssl "1"
composer config platform.ext-zip "1"
composer config platform.ext-pdo_mysql "1"
composer config platform.ext-soap "1"
composer config platform.ext-xml "1"
composer config platform.ext-xmlwriter "1"
composer install
```

Preparing the Training Project (external - Manually)

- Get the Magento 2 sources

```
cd ~/
mkdir projects
cd projects
composer create-project --repository-url=https://repo.magento.com/
    magento/project-community-edition=2.2.5 magento2 --ignore-platform-reqs --no-install
cd magento2
composer config bin-dir ./bin
composer config platform.php "7.0.19"
composer config platform.ext-bcmath "1"
composer config platform.ext-ctype "1"
composer config platform.ext-gd "1"
composer config platform.ext-spl "1"
composer config platform.ext-dom "1"
composer config platform.ext-simplexml "1"
composer config platform.ext-mcrypt "1"
composer config platform.ext-hash "1"
composer config platform.ext-curl "1"
composer config platform.ext-iconv "1"
composer config platform.ext-intl "1"
composer config platform.ext-xsl "1"
composer config platform.ext-mbstring "1"
composer config platform.ext-openssl "1"
composer config platform.ext-zip "1"
composer config platform.ext-pdo_mysql "1"
composer config platform.ext-soap "1"
composer config platform.ext-xml "1"
composer config platform.ext-xmlwriter "1"
composer install
```

- Add the Sample Data

```
composer suggest --no-dev | grep "magento/" | grep "sample-data" | sed 's/$/=^100.2/'
  | xargs composer require
```

Preparing the Training Project (external - 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
```

- Install requirements on the LXC

```
ssh root@magento2.lxc
cd /var/www/magento2
./architecture/script/provision.sh
exit
```

- Commit on git and open the project under PhpStorm!

Preparing the Training Project (external - Manually)

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

- Verify the LXC

```
ssh smile@magento2.lxc

sudo -u www-data php -v
mysql -h mydb -u magento2 -p magento2
password: [!magento2]
exit

telnet myredis 6379
flushall
quit

telnet myredis 6380
flushall
quit

exit
```

Preparing the Training Project (external - Manually)

Install Magento 2

- Launch the Setup Wizard: http://magento2.lxc/setup/

Preparing the Training Project (external - Manually)

Install Magento 2

- Launch the Setup Wizard: http://magento2.lxc/setup/
- Click on "Agree and Setup Magento"

Preparing the Training Project (external - Manually)

Install Magento 2

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

Preparing the Training Project (external - 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 (external - Manually)

Install Magento 2

- Enter the database information
  - Host: mydb
  - User: magento2
  - Password: !magento2
  - database: magento2

Preparing the Training Project (external - Manually)

Install Magento 2

- Enter the database information
  - Host: mydb
  - User: magento2
  - Password: !magento2
  - database: magento2
- Enter the Web configuration information
  - Store Address: http://magento2.lxc/
  - Magento Admin Address: admin

Preparing the Training Project (external - Manually)

Install Magento 2

- Enter the database information
  - Host: mydb
  - 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

Preparing the Training Project (external - Manually)

Install Magento 2

- Enter the Admin Account information

Preparing the Training Project (external - Manually)

Install Magento 2

- Enter the Admin Account information
- Click on "Install Now"

Preparing the Training Project (external - Manually)

Install Magento 2

- Enter the Admin Account information
- Click on "Install Now"
- Open the "Console Log"

Preparing the Training Project (external - 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 (external - Manually)

Install Magento 2

- First time in the Back Office: http://magento2.lxc/admin/

Preparing the Training Project (external - 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
```

Preparing the Training Project (external - 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 smile@magento2.lxc
sudo -u www-data crontab -e
*/1 * * * * /var/www/magento2/bin/magento cron:run
exit
```

Preparing the Training Project (external - 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 smile@magento2.lxc
sudo -u www-data crontab -e
*/1 * * * * /var/www/magento2/bin/magento cron:run
exit
```

- Better rights

```
ssh root@magento2.lxc
cd /var/www/magento2/app/etc/
chown smile.www-data config.php env.php
exit
```

Preparing the Training Project (external - Manually)

Magento 2 and Redis - Session

- You must modify the ./app/etc/env.php file

```
'session' => array (
    'save' => 'redis',
    'redis' => array (
        'host' => 'myredis',
        'port' => '6380',
        'database' => '1',
    ),
),
```

Preparing the Training Project (external - Manually)

Magento 2 and Redis - Cache

- You must modify the ./app/etc/env.php file

```
'cache' => array (
    'frontend' => array (
        'default' => array (
            'backend' => 'Cm_Cache_Backend_Redis',
            'id_prefix' => 'm2modules_',
            'backend_options' => array (
                'server' => 'myredis',
                'port' => '6379',
                'persistent' => '',
                'database' => '1',
                'force_standalone' => '0',
                'connect_retries' => '1',
                'read_timeout' => '10',
                'automatic_cleaning_factor' => '0',
                'compress_data' => '1',
                'compress_tags' => '1',
                'compress_threshold' => '20480',
                'compression_lib' => 'gzip',
            ),
        ),
    ),
),
```

Preparing the Training Project (external - Manually)

Magento 2 and Varnish

- You must modify the ./app/etc/env.php file

```
'http_cache_hosts' => array (
    0 => array (
      'host' => 'myfront1',
      'port' => '81',
    ),
),
```

Preparing the Training Project (external - Manually)

Magento 2 and Varnish

- You must modify the ./app/etc/env.php file

```
'http_cache_hosts' => array (
    0 => array (
      'host' => 'myfront1',
      'port' => '81',
    ),
),
```

- Clean the cache

```
ssh smile@magento2.lxc
cd /var/www/magento2
./architecture/script/cleancache.sh
exit
```

Preparing the Training Project (external - Manually)

Magento 2 and Varnish

- You must modify the ./app/etc/env.php file

```
'http_cache_hosts' => array (
    0 => array (
      'host' => 'myfront1',
      'port' => '81',
    ),
),
```

- Clean the cache

```
ssh smile@magento2.lxc
cd /var/www/magento2
./architecture/script/cleancache.sh
exit
```

- You must use Varnish as page cache

```
Stores > Configuration > Advanced > System > Full Page Cache

Access list:  localhost,myfront1,10.0.3.1
Backend host: localhost
Backend port: 82
Grace period: 300
```

Preparing the Training Project (external - Manually)

Magento 2 and Varnish

- You must modify the ./app/etc/env.php file

```
'http_cache_hosts' => array (
    0 => array (
      'host' => 'myfront1',
      'port' => '81',
    ),
),
```

- Clean the cache

```
ssh smile@magento2.lxc
cd /var/www/magento2
./architecture/script/cleancache.sh
exit
```

- You must use Varnish as page cache

```
Stores > Configuration > Advanced > System > Full Page Cache

Access list:  localhost,myfront1,10.0.3.1
Backend host: localhost
Backend port: 82
Grace period: 300
```

- You must enable the probe in /etc/varnish/magento2.vcl

Preparing the Training Project (external - Manually)

Magento 2 and HTTPS

- You must enable HTTPS

```
Stores > Configuration > General > Web > Base URLs (Secure)

Secure Base URL:                https://magento2.lxc/
Use Secure URLs on Storefront: Yes
Use Secure URLs in Admin:       Yes
```

Preparing the Training Project (external - Manually)

Magento 2 and HTTPS

- You must enable HTTPS

```
Stores > Configuration > General > Web > Base URLs (Secure)

Secure Base URL:                https://magento2.lxc/
Use Secure URLs on Storefront: Yes
Use Secure URLs in Admin:       Yes
```

- Clean the cache

```
ssh smile@magento2.lxc
cd /var/www/magento2
./architecture/script/cleancache.sh
exit
```

Preparing the Training Project (external - Manually)

Magento 2 and PhpStorm

- In order to use validate the XML files, PhpStorm must know where the XSD files are located.

Preparing the Training Project (external - Manually)

Magento 2 and PhpStorm

- In order to use validate the XML files, PhpStorm must know where the XSD files are located.
- Magento can generate automatically the URN catalog for PhpStorm:
  Close PhpStorm before generating the misc.xml file

Preparing the Training Project (external - Manually)

Magento 2 and PhpStorm

- In order to use validate the XML files, PhpStorm must know where the XSD files are located.
- Magento can generate automatically the URN catalog for PhpStorm:
  Close PhpStorm before generating the misc.xml file

```
cd ~/projects/magento2
echo "<?xml version=\"1.0\" encoding=\"UTF-8\"?><project/>" > ./.idea/misc.xml
chmod 666 .idea/misc.xml

ssh smile@magento2.lxc
cd /var/www/magento2
sudo -u www-data bin/magento dev:urn-catalog:generate .idea/misc.xml
exit

chmod 664 .idea/misc.xml
sed -i "s/\/var\/www/\/home\/training\/projects/g" .idea/misc.xml
```

Preparing the Training Project (external - Manually)

- Commit all the files

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

Plan

3 Key notions

- Scope notion
- Product types

Plan

Scope notion

Magento is organized in 3 types of scopes.

Scope notion

Magento is organized in 3 types of scopes.

- Website

Scope notion

Magento is organized in 3 types of scopes.

- Website
- Store

Scope notion

Magento is organized in 3 types of scopes.

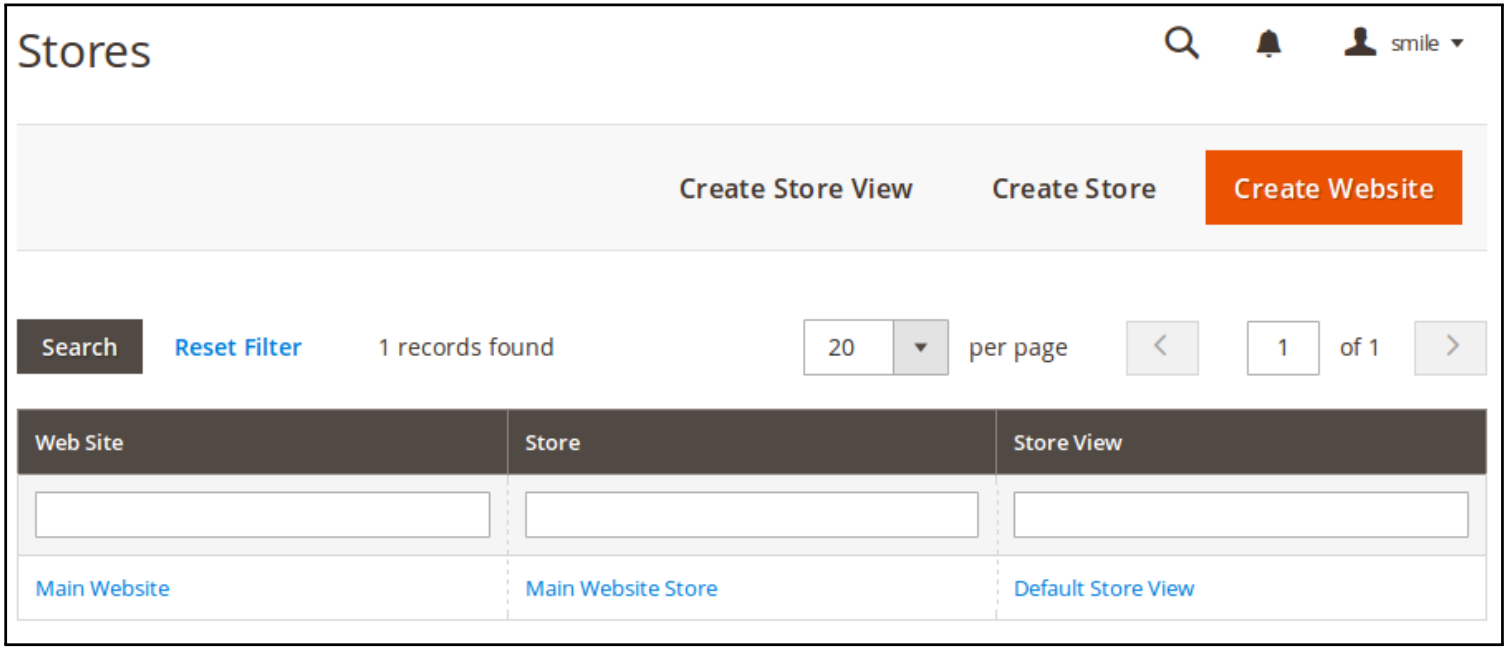
- Website
- Store
- Store view

Scope notion

Magento is organized in 3 types of scopes.

- Website
- Store
- Store view

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

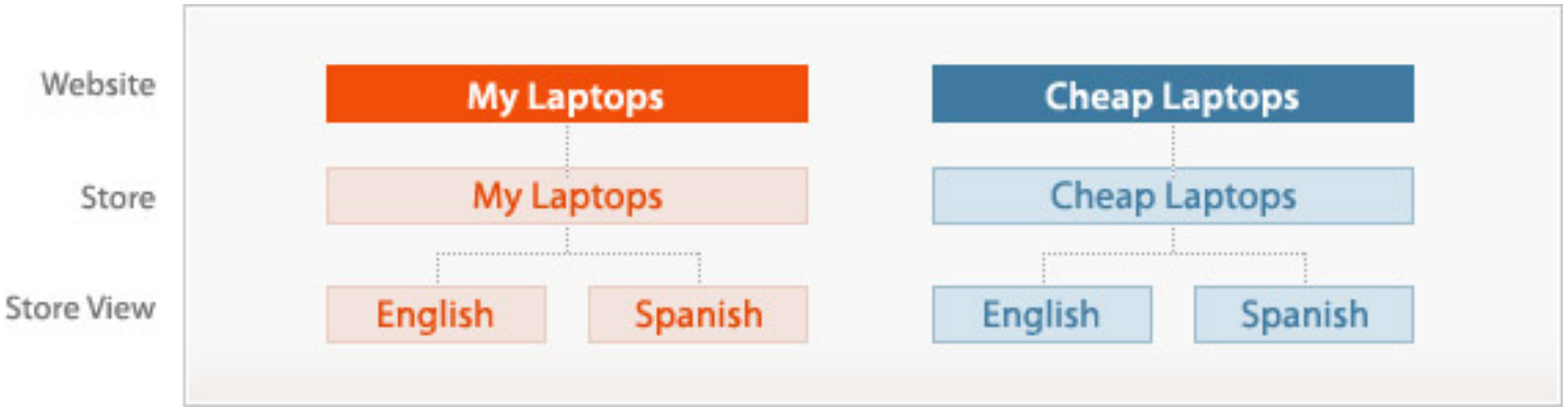


Scope notion>Website

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

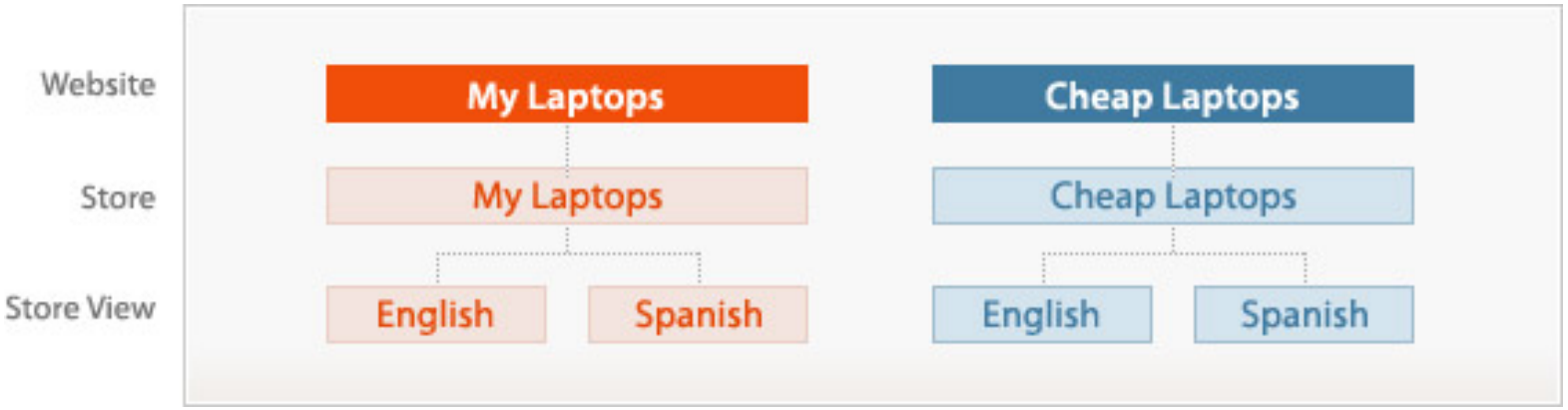
Scope notion>Store

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



Scope notion>Store View

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



Plan

Product types 1/2

- Simple product
  - No specificities, base product type, mostly used
  - Eg.: book

Product types 1/2

- Simple product
  - No specificities, base product type, mostly used
  - Eg.: book
- Configurable product
  - Product with a drop-down list of options for each variant
  - Each variant is a simple product associated to the configurable product
  - Eg.: t-shirt with choice of color and size (each combination of size and color refers to a simple product)

Product types 1/2

- Simple product
  - No specificities, base product type, mostly used
  - Eg.: book
- Configurable product
  - Product with a drop-down list of options for each variant
  - Each variant is a simple product associated to the configurable product
  - Eg.: t-shirt with choice of color and size (each combination of size and color refers to a simple product)
- Grouped product
  - A grouped product is a package of two or more simple products
  - Price, description, images, etc. can be specified on their own
  - Eg.: camera + SD card sold together

Product types 2/2

- Bundle product
  - A bundle is a "build your own", customizable product
  - Each item in a bundle is a simple product
  - Eg.: computer
    - motherboard and CPU are mandatory, user can choose each in a given list, qty is limited to one
    - mouse, keyboard are optional
    - extra RAM can be added, optional, qty is 0 to N

Product types 2/2

- Bundle product
  - A bundle is a "build your own", customizable product
  - Each item in a bundle is a simple product
  - Eg.: computer
    - motherboard and CPU are mandatory, user can choose each in a given list, qty is limited to one
    - mouse, keyboard are optional
    - extra RAM can be added, optional, qty is 0 to N
- Virtual product
  - Not a physical product, esp. does not manage stock qty
  - Eg.: all kind of services

Product types 2/2

- Bundle product
  - A bundle is a "build your own", customizable product
  - Each item in a bundle is a simple product
  - Eg.: computer
    - motherboard and CPU are mandatory, user can choose each in a given list, qty is limited to one
    - mouse, keyboard are optional
    - extra RAM can be added, optional, 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 physically, but can be associated with a downloadable file
  - Eg.: a training video

Product types 2/2

- Bundle product
  - A bundle is a "build your own", customizable product
  - Each item in a bundle is a simple product
  - Eg.: computer
    - motherboard and CPU are mandatory, user can choose each in a given list, qty is limited to one
    - mouse, keyboard are optional
    - extra RAM can be added, optional, 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 physically, but can be associated with a downloadable file
  - Eg.: a training video
- **Try using simple products as much as you can**

Plan

4 Architecture

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

Plan

Magento directory structure

- /path/to/magento/root/
  - **app/** : application code (see after)

Magento directory structure

- /path/to/magento/root/
  - **app/** : application code (see after)
  - **bin/** : magento shell file

Magento directory structure

- /path/to/magento/root/
  - **app/** : application code (see after)
  - **bin/** : magento shell file
  - **dev/** : unit test files

Magento directory structure

- /path/to/magento/root/
  - **app/** : application code (see after)
  - **bin/** : magento shell file
  - **dev/** : unit test files
  - **generated/** : factory classes, plugin interceptors, ...

Magento directory structure

- /path/to/magento/root/
  - **app/** : application code (see after)
  - **bin/** : magento shell file
  - **dev/** : unit test files
  - **generated/** : factory classes, plugin interceptors, ...
  - **lib/internal** : php libraries that can't be installed with composer
  - **lib/web** : web libraries (jquery, wysiwyg, ...)

Magento directory structure

- /path/to/magento/root/
  - **app/** : application code (see after)
  - **bin/** : magento shell file
  - **dev/** : unit test files
  - **generated/** : factory classes, plugin interceptors, ...
  - **lib/internal** : php libraries that can't be installed with composer
  - **lib/web** : web libraries (jquery, wysiwyg, ...)
  - **phpserver/** : if you want to use the Built-in PHP webserver

Magento directory structure

- /path/to/magento/root/
  - **app/** : application code (see after)
  - **bin/** : magento shell file
  - **dev/** : unit test files
  - **generated/** : factory classes, plugin interceptors, ...
  - **lib/internal** : php libraries that can't be installed with composer
  - **lib/web** : web libraries (jquery, wysiwyg, ...)
  - **phpserver/** : if you want to use the Built-in PHP webserver
  - **pub/** : public folder
    - errors
    - media
    - static

Magento directory structure

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

Magento directory structure

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

Magento directory structure

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

Magento directory structure

vendor/ folder

Magento directory structure

vendor/ folder

- Contains all the dependencies used by Magento

Magento directory structure

vendor/ folder

- Contains all the dependencies used by Magento
- Contains all the Magento core files, in the **magento** directory
  - **framework** : Magento framework library files
  - **language-xx_xx** : Magento language package files
  - **magento2-base** : Magento basic structure
  - **module-xxxxx** : Magento modules files
  - **theme-xxx-xxx** : Magento themes files
  - **zendframework1** : Fork of Zend Framework v1

Magento directory structure

vendor/ folder

- Contains all the dependencies used by Magento
- Contains all the Magento core files, in the **magento** directory
  - **framework** : Magento framework library files
  - **language-xx_xx** : Magento language package files
  - **magento2-base** : Magento basic structure
  - **module-xxxxx** : Magento modules files
  - **theme-xxx-xxx** : Magento themes files
  - **zendframework1** : Fork of Zend Framework v1
- Never modify any of thoses files
  They should by ignored by the VCS
- To update the libraries:

```
cd ~/projects/magento2/
composer update
```

Magento directory structure

app/ folder

Magento directory structure

app/ folder

- **etc/**
  - App configuration (env.php, list of activated modules, ...)

Magento directory structure

app/ folder

- **etc/**
  - App configuration (env.php, list of activated modules, ...)
- **code/**
  - Contains the custom modules

Magento directory structure

app/ folder

- **etc/**
  - App configuration (env.php, list of activated modules, ...)
- **code/**
  - Contains the custom modules
- **design/**
  - Contains the custom themes

Plan

Magento modes

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

Magento modes

Developer Mode

- Static file materialization is enabled
- Uncaught exceptions are displayed in the browser
- Exceptions are managed by an error handler, and logged
- System logging in var/report, highly detailed

Magento modes

Production Mode

- Deployment phase on the production system, highest performance
- Exceptions are not displayed to the user – written to logs only
- This mode disables static file materialization
- The Magento docroot must have read-only permissions
- Specific docroot: ./pub (the virtualhost must be modified)

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

Specify a Mode

- In the Apache virtualhost: **SetEnv MAGE_MODE developer**
- Using the console:

```
sudo -u www-data bin/magento deploy:mode:set developer
```

Magento modes

Maintenance Mode

- Used to make a site unavailable to the public during deployments or other changes
- Detects the **var/.maintenance.flag** file
- Can use a authorized list of ips in the **var/.maintenance.ip** file

Magento modes

Maintenance Mode

- Used to make a site unavailable to the public during deployments or other changes
- Detects the **var/.maintenance.flag** file
- Can use a authorized list of ips in the **var/.maintenance.ip** file
- Can be enabled or disabled using the console:

```
sudo -u www-data bin/magento maintenance:enable
sudo -u www-data bin/magento maintenance:disable
```

Plan

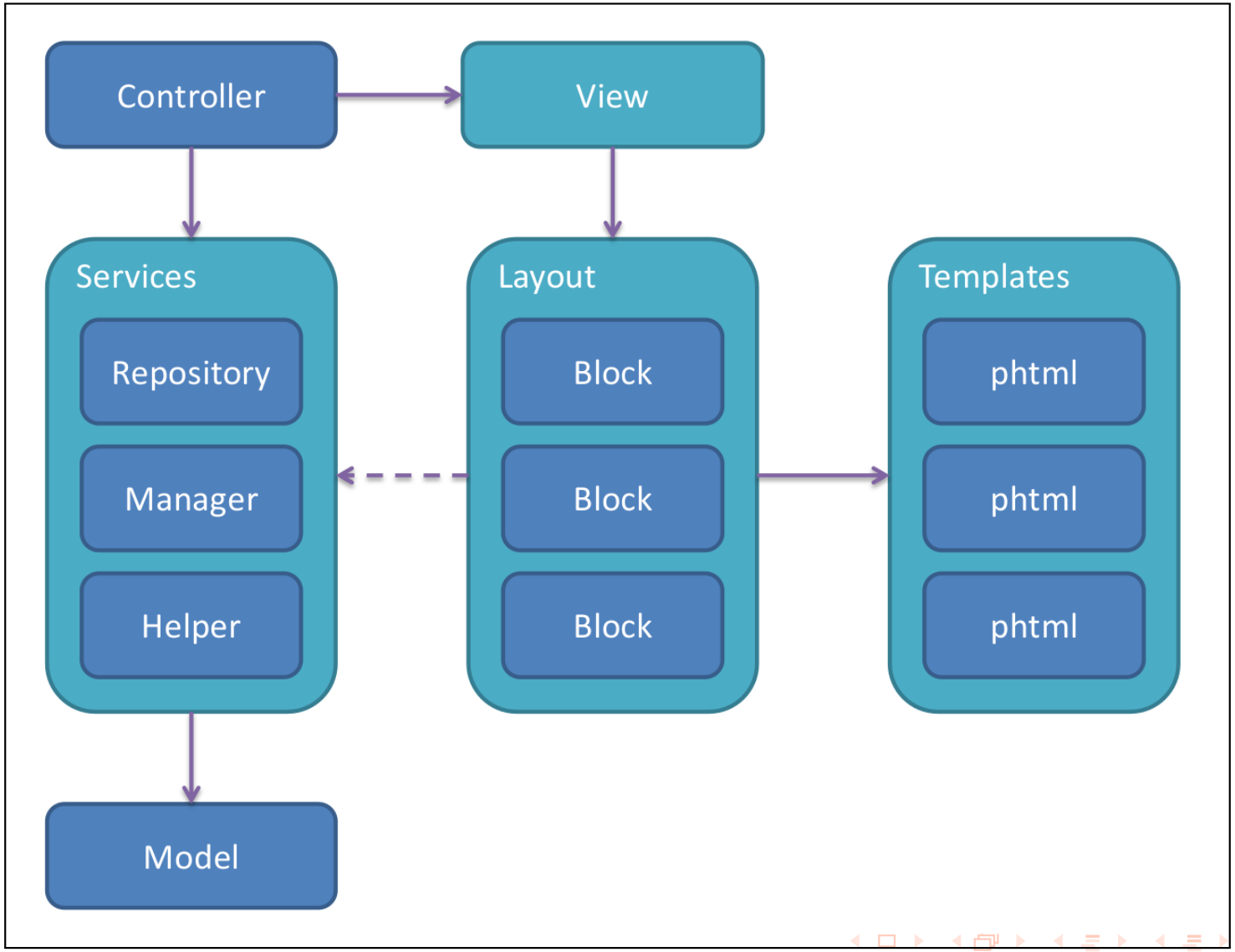
Magento areas

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

Plan

Magento module structure

Design pattern MVC (Model, View, Controller)



Magento module structure

Magento Module Architecture

Magento module structure

Magento Module Architecture

- A Magento module encapsulates some elements of functional scope (i.e : catalog, customer, cataloginventory):

Magento module structure

Magento Module Architecture

- A Magento module encapsulates some elements of functional scope (i.e : catalog, customer, cataloginventory):
  - controllers
  - models
  - display (blocks)
  - ...

Magento module structure

Magento Module Architecture

- A Magento module encapsulates some elements of functional scope (i.e : catalog, customer, cataloginventory):
  - controllers
  - models
  - display (blocks)
  - ...
- One module must not manage several features
- Several modules must not manage the same feature

Magento module structure

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

Magento module structure

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

- **Api/** Api interface files for all the module classes

Magento module structure

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

- **Api/** Api interface files for all the module classes
- **Block/** Fontend and backend blocks

Magento module structure

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

- **Api/** Api interface files for all the module classes
- **Block/** Fontend and backend blocks
- **Config/** Specific XML config readers

Magento module structure

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

- **Api/** Api interface files for all the module classes
- **Block/** Fontend and backend blocks
- **Config/** Specific XML config readers
- **Console/** Console command classes for the Magento CLI

Magento module structure

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

- **Api/** Api interface files for all the module classes
- **Block/** Fontend and backend blocks
- **Config/** Specific XML config readers
- **Console/** Console command classes for the Magento CLI
- **Controller/** Controller logic for navigation (both FO and BO)

Magento module structure

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

- **Api/** Api interface files for all the module classes
- **Block/** Fontend and backend blocks
- **Config/** Specific XML config readers
- **Console/** Console command classes for the Magento CLI
- **Controller/** Controller logic for navigation (both FO and BO)
- **Cron/** Cron classes

Magento module structure

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

- **Api/** Api interface files for all the module classes
- **Block/** Fontend and backend blocks
- **Config/** Specific XML config readers
- **Console/** Console command classes for the Magento CLI
- **Controller/** Controller logic for navigation (both FO and BO)
- **Cron/** Cron classes
- **etc/** XML and XSD configuration files

Magento module structure

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

- **Api/** Api interface files for all the module classes
- **Block/** Fontend and backend blocks
- **Config/** Specific XML config readers
- **Console/** Console command classes for the Magento CLI
- **Controller/** Controller logic for navigation (both FO and BO)
- **Cron/** Cron classes
- **etc/** XML and XSD configuration files
- **Helper/** Classes with utilitary functions (should be avoided)

Magento module structure

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

- **Api/** Api interface files for all the module classes
- **Block/** Fontend and backend blocks
- **Config/** Specific XML config readers
- **Console/** Console command classes for the Magento CLI
- **Controller/** Controller logic for navigation (both FO and BO)
- **Cron/** Cron classes
- **etc/** XML and XSD configuration files
- **Helper/** Classes with utilitary functions (should be avoided)
- **i18n/** CSV language files

Magento module structure

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

- **Api/** Api interface files for all the module classes
- **Block/** Fontend and backend blocks
- **Config/** Specific XML config readers
- **Console/** Console command classes for the Magento CLI
- **Controller/** Controller logic for navigation (both FO and BO)
- **Cron/** Cron classes
- **etc/** XML and XSD configuration files
- **Helper/** Classes with utilitary functions (should be avoided)
- **i18n/** CSV language files
- **Model/** ORM classes + classes with the base logic

Magento module structure

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

- **Api/** Api interface files for all the module classes
- **Block/** Fontend and backend blocks
- **Config/** Specific XML config readers
- **Console/** Console command classes for the Magento CLI
- **Controller/** Controller logic for navigation (both FO and BO)
- **Cron/** Cron classes
- **etc/** XML and XSD configuration files
- **Helper/** Classes with utilitary functions (should be avoided)
- **i18n/** CSV language files
- **Model/** ORM classes + classes with the base logic
- **Observer/** Observer classes (see after)

Magento module structure

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

- **Api/** Api interface files for all the module classes
- **Block/** Fontend and backend blocks
- **Config/** Specific XML config readers
- **Console/** Console command classes for the Magento CLI
- **Controller/** Controller logic for navigation (both FO and BO)
- **Cron/** Cron classes
- **etc/** XML and XSD configuration files
- **Helper/** Classes with utilitary functions (should be avoided)
- **i18n/** CSV language files
- **Model/** ORM classes + classes with the base logic
- **Observer/** Observer classes (see after)
- **Plugin/** Plugin classes (see after)

Magento module structure

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

- **Api/** Api interface files for all the module classes
- **Block/** Fontend and backend blocks
- **Config/** Specific XML config readers
- **Console/** Console command classes for the Magento CLI
- **Controller/** Controller logic for navigation (both FO and BO)
- **Cron/** Cron classes
- **etc/** XML and XSD configuration files
- **Helper/** Classes with utilitary functions (should be avoided)
- **i18n/** CSV language files
- **Model/** ORM classes + classes with the base logic
- **Observer/** Observer classes (see after)
- **Plugin/** Plugin classes (see after)
- **Rewrite/** Rewrite classes (see after)

Magento module structure

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

- **Api/** Api interface files for all the module classes
- **Block/** Fontend and backend blocks
- **Config/** Specific XML config readers
- **Console/** Console command classes for the Magento CLI
- **Controller/** Controller logic for navigation (both FO and BO)
- **Cron/** Cron classes
- **etc/** XML and XSD configuration files
- **Helper/** Classes with utilitary functions (should be avoided)
- **i18n/** CSV language files
- **Model/** ORM classes + classes with the base logic
- **Observer/** Observer classes (see after)
- **Plugin/** Plugin classes (see after)
- **Rewrite/** Rewrite classes (see after)
- **Setup/** Update files (updating the database structure, inserting new data or updating configuration)

Magento module structure

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

- **Api/** Api interface files for all the module classes
- **Block/** Fontend and backend blocks
- **Config/** Specific XML config readers
- **Console/** Console command classes for the Magento CLI
- **Controller/** Controller logic for navigation (both FO and BO)
- **Cron/** Cron classes
- **etc/** XML and XSD configuration files
- **Helper/** Classes with utilitary functions (should be avoided)
- **i18n/** CSV language files
- **Model/** ORM classes + classes with the base logic
- **Observer/** Observer classes (see after)
- **Plugin/** Plugin classes (see after)
- **Rewrite/** Rewrite classes (see after)
- **Setup/** Update files (updating the database structure, inserting new data or updating configuration)
- **view/** phtml template files, XML layout files, static files (CSS, JS)

Helloworld module (see 02-helloworld) 1/4

Magento module structure

Helloworld module (see 02-helloworld) 1/4

- Module folder: **./src/app/code/Training/Helloworld**

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>
```

Helloworld module (see 02-helloworld) 1/4

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

```xml
<?xml version="1.0"?>
<config
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="urn:magento:framework:Module/etc/module.xsd">
    <module name="Training_Helloworld" setup_version="1.0.0">
    </module>
</config>
```

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

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

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>
```

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;

use Magento\Framework\App\Action\Action;
use Magento\Framework\Controller\Result\Raw as ResultRaw;
use Magento\Framework\Controller\ResultFactory;

/**
 * Action: Index/Index
 *
 * @author    Laurent MINGUET <lamin@smile.fr>
 * @copyright 2018 Smile
 */
class Index extends Action
{
    /**
     * {@inheritdoc}
     */
    public function execute()
    {
        /** @var ResultRaw $result */
        $result = $this->resultFactory->create(ResultFactory::TYPE_RAW);
        $result->setContents('Hello World!');
        return $result;
    }
}
```

Magento module structure

Helloworld module (see 02-helloworld) 4/4

- Test: `https://magento2.lxc/helloworld/`

Magento module structure

Helloworld module (see 02-helloworld) 4/4

- Test: `https://magento2.lxc/helloworld/`
- Error 404 not found?! Why?

Magento module structure

Helloworld module (see 02-helloworld) 4/4

- Test: `https://magento2.lxc/helloworld/`
- Error 404 not found?! Why?
- Ask Magento to clean the cache and to register the module! Required when:
  - You create a new module
  - You modify the parameters of a PHP class constructor
  - You modify a XML config file

Helloworld module (see 02-helloworld) 4/4

- Test: `https://magento2.lxc/helloworld/`
- Error 404 not found?! Why?
- Ask Magento to clean the cache and to register the module! Required when:
  - You create a new module
  - You modify the parameters of a PHP class constructor
  - You modify a XML config file
- How to clean the cache:

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

Helloworld module (see 02-helloworld) 4/4

- Test: `https://magento2.lxc/helloworld/`
- Error 404 not found?! Why?
- Ask Magento to clean the cache and to register the module! Required when:
  - You create a new module
  - You modify the parameters of a PHP class constructor
  - You modify a XML config file
- How to clean the cache:

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

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

```
sudo -u www-data bin/magento setup:upgrade
```

Plan

Configuration files

- All the XML configuration files of a module are in the **etc/** folder

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

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 at the root of the **etc/** folder are shared by all areas
- The files in a specific area folder are only for this area

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 at the root of the **etc/** folder are shared by all areas
- The files in a specific area folder are only for this area
- XML configuration files of each module are merged together
- An XSD schema file may exist to validate the merged file
- Look at magento:module_catalog module:
  - ./etc/product_options.xsd
  - ./etc/product_options_**merged**.xsd

Plan

5 Concepts

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

Plan

Object Manager Factory

- Replaces Mage::getModel, Mage::getSingleton, ... of M1

Object Manager Factory

- Replaces Mage::getModel, Mage::getSingleton, ... of M1
- Method **create**: returns a new instance

Object Manager Factory

- Replaces Mage::getModel, Mage::getSingleton, ... of M1
- Method **create**: returns a new instance
- Method **get**: returns a singleton

Object Manager Factory

- Replaces Mage::getModel, Mage::getSingleton, ... of M1
- Method **create**: returns a new instance
- Method **get**: returns a singleton
- Never use the global Object Manager Factory

Object Manager Factory

- Replaces Mage::getModel, Mage::getSingleton, ... of M1
- Method **create**: returns a new instance
- Method **get**: returns a singleton
- Never use the global Object Manager Factory
- Use only the factory of the object you want to create

Object Manager Factory

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

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

  `\Magento\Catalog\Block\Adminhtml\Category\Widget\Chooser::96`

Object Manager Factory

- Replaces Mage::getModel, Mage::getSingleton, ... of M1
- Method **create**: returns a new instance
- Method **get**: returns a singleton
- Never use the global Object Manager Factory
- Use only the factory of the object you want to create
- 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, in the **generated** directory

Plan

Dependency Injection

- Methods must only use the objects that have been injected in the class

Dependency Injection

- Methods must only use the objects that have been injected in the class
- Two injection methods
  - Constructor injection: add the required object/factory to the class constructor

Dependency Injection

- Methods must only use the objects that have been injected in the class
- Two injection methods
  - Constructor injection: add the required object/factory to the class constructor
  - Method injection: add the required object to the method parameters

Dependency Injection

- Methods must only use the objects that have been injected in the class
- Two injection methods
  - Constructor injection: add the required object/factory to the class constructor
  - Method injection: add the required object to the method parameters
- 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

di.xml files

- You can configure the object manager to load a specific class instead of the specified class/interface:

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

di.xml files

- You can configure the object manager to load a specific class instead of the specified class/interface:

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

- You can also specify the value of constructor parameters

```
<type name="Magento\Framework\App\RouterList">
    <arguments>
        <argument name="routerList" xsi:type="array">
            <item name="cms" xsi:type="array">
                <item name="class" xsi:type="string">Magento\Cms\Controller\Router</item>
                <item name="disable" xsi:type="boolean">false</item>
                <item name="sortOrder" xsi:type="string">60</item>
            </item>
        </argument>
    </arguments>
</type>
```

Dependency Injection

To instanciate a requested object, the Magento 2 Object Manager:

Dependency Injection

To instanciate a requested object, the Magento 2 Object Manager:

- Parses the constructor parameters of the object

Dependency Injection

To instanciate a requested object, the Magento 2 Object Manager:

- Parses the constructor parameters of the object
- Uses **./etc/di.xml** and **./etc/[area]/di.xml** files to prepare the parameter values

Dependency Injection

To instanciate a requested object, the Magento 2 Object Manager:

- Parses the constructor parameters of the object
- Uses **./etc/di.xml** and **./etc/[area]/di.xml** files to prepare the parameter values
- Creates the object by giving all the parameters to the constructor

Dependency Injection

To instanciate a requested object, the Magento 2 Object Manager:

- Parses the constructor parameters of the object
- Uses **./etc/di.xml** and **./etc/[area]/di.xml** files to prepare the parameter values
- Creates the object by giving all the parameters to the constructor
- Returns the object

Dependency Injection

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

Dependency Injection

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

- **Injectable**: An object (typically a singleton) that can be instantiated by the object manager

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**

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

Dependency Injection

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

Dependency Injection

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

- **Injectable** can request for other **Injectable** objects in the constructor

Dependency Injection

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

- **Injectable** can request for other **Injectable** objects in the constructor
- **Injectable** must not request for **Non-injectable** objects in the constructor

Dependency Injection

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

- **Injectable** can request for other **Injectable** objects in the constructor
- **Injectable** must 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

Dependency Injection

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

- **Injectable** can request for other **Injectable** objects in the constructor
- **Injectable** must 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 receive 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)

Compiler tool

- Reads all the class definition using reflection
- Generates all the required factories
- Generates interceptors for all classes that have plugins (see after)
- Run the compiler tool:

```
sudo -u www-data bin/magento setup:di:compile
```

Compiler tool

- Reads all the class definition using reflection
- Generates all the required factories
- Generates interceptors for all classes that have plugins (see after)
- Run the compiler tool:

```
sudo -u www-data bin/magento setup:di:compile
```

- See the result in **./generated** folder

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

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

- In the previous module **Training/Helloworld**

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`

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 associated product

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 associated product
- Display the name of the product

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 associated product
- Display the name of the product
- Display the 404 page if the product does not exist

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

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

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

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;

use Magento\Framework\App\Action\Action;

/**
 * Action: Product/Index
 *
 * @author    Laurent MINGUET <lamin@smile.fr>
 * @copyright 2018 Smile
 */
class Index extends Action
{

}
```

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

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

- Ask for the Product Factory in the constructor

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

- Ask for the Product Factory in the constructor

```php
<?php
...
use Magento\Framework\App\Action\Context;
use Magento\Catalog\Model\ProductFactory;
...

    /**
     * @var ProductFactory
     */
    protected $productFactory;

    /**
     * @param Context $context
     * @param ProductFactory $productFactory
     */
    public function __construct(Context $context, ProductFactory $productFactory)
    {
        parent::__construct($context);

        $this->productFactory = $productFactory;
    }

...
```

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

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

- Load the asked product

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

- Load the asked product

```php
<?php
...
use Magento\Catalog\Model\Product;
...

    /**
     * Get the requested product
     *
     * @return Product|null
     */
    protected function getProduct()
    {
        // get the requested id
        $productId = (int) $this->getRequest()->getParam('id');
        if (!$productId) {
            return null;
        }

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

        return $product;
    }

...
```

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

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

- Display the product name

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

- Display the product name

```php
<?php
...
use Magento\Framework\Controller\Result\Raw as ResultRaw;
use Magento\Framework\Exception\NotFoundException;
...
    /**
     * {@inheritdoc}
     */
    public function execute()
    {
        $product = $this->getProduct();
        if ($product === null) {
            throw new NotFoundException(__('product not found'));
        }

        /** @var ResultRaw $result */
        $result = $this->resultFactory->create(ResultFactory::TYPE_RAW);
        $result->setContents('Product: ' . $product->getName());

        return $result;
    }
...
```

Plan

Events and Observers

Event Design Pattern

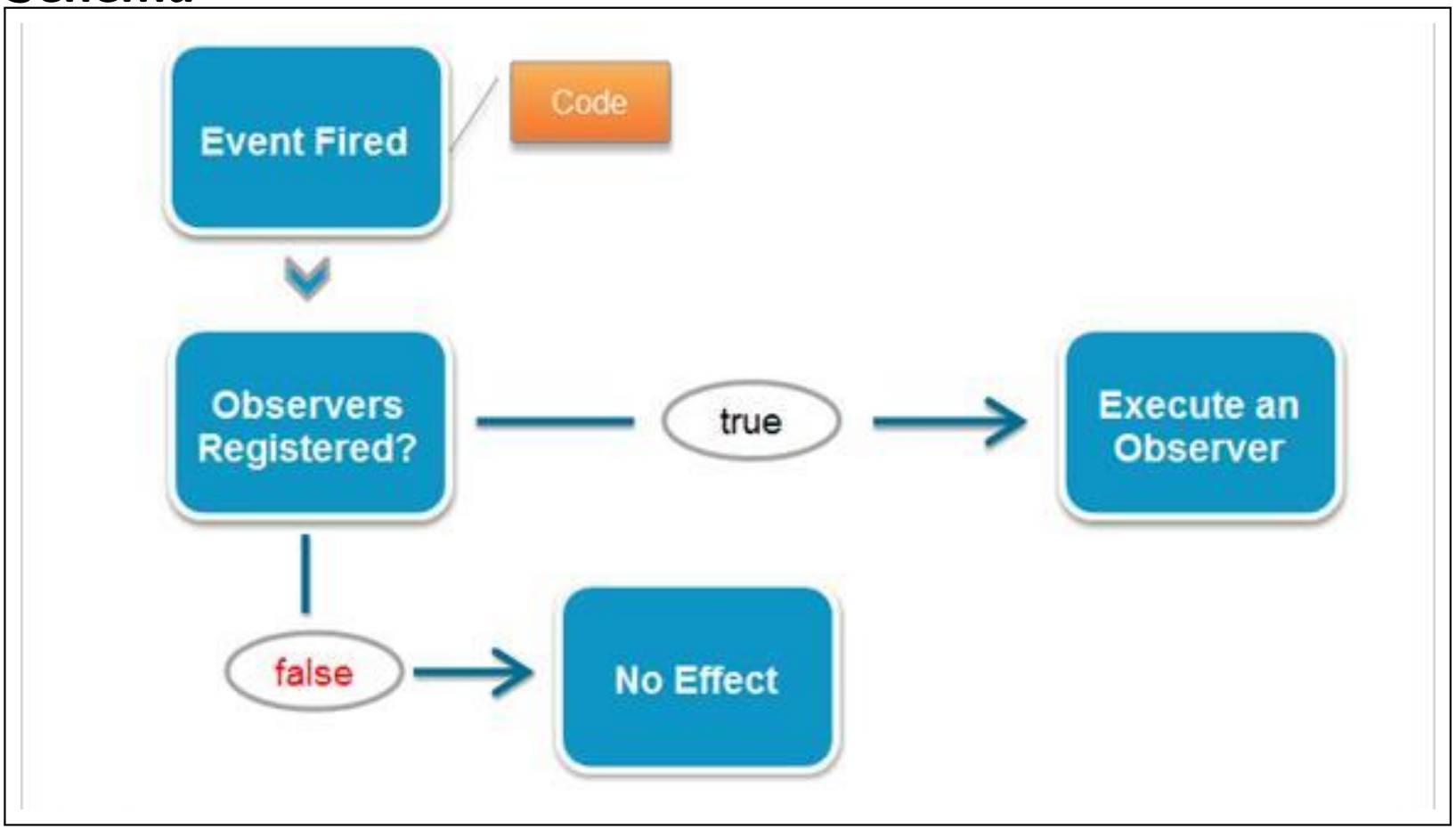
- One of the main ways to extend Magento functionality
- Events are dispatched by modules when certain actions are triggered
- Events can be dispatched by using the class

  `Magento\Framework\Event\Manager`
- When an event is dispatched, it will pass data to any observer configured to watch that event

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;
    }
}
```

Schema


Event Fired
Code
Observers Registered?
true
Execute an Observer
false
No Effect

Events and Observers

Observer

- Class that implements **\Magento\Framework\Event\ObserverInterface**

Events and Observers

Observer

- Class that implements **\Magento\Framework\Event\ObserverInterface**
- One required method **execute**
- With one required parameter: \Magento\Framework\Event\Observer **$observer**

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 areas
- Registered in **./etc/[area]/events.xml** to execute it only in a specific area

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 areas
- Registered in **./etc/[area]/events.xml** to execute it only in a specific area
- Good practice: create the class in the **Observer** folder of your module

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>
```

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

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

- In the previous module **Training/Helloworld**

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

- In the previous module **Training/Helloworld**
- Prepare a new observer **PredispatchLogUrl**

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**

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** with the **info** level

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

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

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

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 2018 Smile
 */
class PredispatchLogUrl implements ObserverInterface
{

}
```

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

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

- Method: **execute**

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

- Method: **execute**

```
<?php
...
use Magento\Framework\Event\Observer;
...

    /**
     * {@inheritdoc}
     */
    public function execute(Observer $observer)
    {
        // @todo
    }

...
```

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

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

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

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

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

- Inject the logger in the observer

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

- Inject the logger in the observer

```
<?php
...
use Psr\Log\LoggerInterface;
...

    /**
     * @var LoggerInterface
     */
    protected $logger;

    /**
     * @param LoggerInterface $logger
     */
    public function __construct(LoggerInterface $logger)
    {
        $this->logger = $logger;
    }

...
```

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

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

- Log the pathinfo

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

- Log the pathinfo

```php
<?php
...
use Magento\Framework\HTTP\PhpEnvironment\Request;
...

    /**
     * {@inheritdoc}
     */
    public function execute(Observer $observer)
    {
        /** @var Request $request */
        $request = $observer->getEvent()->getData('request');
        $url = $request->getPathInfo();
        $this->logger->info('Current Url : '.$url);
    }

...
```

Plan

Definition

Plugins

Definition

- Used to extend/change the behavior of any public method within a Magento class

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

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

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

Definition

Plugins

Definition

- One original method can have lots of plugins, executed in a specific order

Plugins

Definition

- One original method can have lots of plugins, executed in a specific order
- A plugin class does not implement any interface or extend any class

Plugins

Definition

- One original method can have lots of plugins, executed in a specific order
- A plugin class does not implement any interface or extend any class
- Declared in the **di.xml** files

Plugins

Definition

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

Plugins

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

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

```xml
<?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="Namespace\Module\Original\Class">
        <plugin
            name="Foo-Bar-Original-Class-Plugin"
            type="Foo\Bar\Plugin\Original\Class"
            sortOrder="10"
            disabled="false"
        />
    </type>

</config>
```

Before-Listener method

Before-Listener method

```php
<?php
    /**
     * Original Method - Before
     *
     * @param \Namespace\Module\Original\Class $subject
     * @param string                           $firstParameter
     * @param string                           $secondParameter
     *
     * @return mixed
     */
    public function beforeOriginalMethod(
        \Namespace\Module\Original\Class $subject,
        $firstParameter,
        $secondParameter
    ) {
        $subject->setSomeThing(true);
        $firstParameter  = mb_strtolower($firstParameter);
        $secondParameter = mb_strtolower($secondParameter);

        return [$firstParameter, $secondParameter];
    }
```

After-Listener method

After-Listener method

```php
<?php
    /**
     * Original Method - After
     *
     * @param \Namespace\Module\Original\Class $subject
     * @param string                           $result
     * @param string                           $firstParameter
     * @param string                           $secondParameter
     *
     * @return string
     */
    public function afterOriginalMethod(
        \Namespace\Module\Original\Class $subject,
        $result,
        $firstParameter,
        $secondParameter
    ) {
        if ($firstParameter) {
            $result = 'My Name Is ' . $result;
        }

        return $result;
    }
```

Around-Listener method

Around-Listener method

```php
<?php
    /**
     * Original Method - Around
     *
     * @param \Namespace\Module\Original\Class $subject
     * @param \Closure                         $proceed
     * @param string                           $firstParameter
     * @param string                           $secondParameter
     *
     * @return string
     */
    public function aroundOriginalMethod(
        \Namespace\Module\Original\Class $subject,
        \Closure $proceed,
        $firstParameter,
        $secondParameter
    ) {
        // something before
        $firstParameter  = mb_strtoupper($firstParameter);
        $secondParameter = mb_strtoupper($secondParameter);

        $result = $proceed($firstParameter, $secondParameter);

        if ($firstParameter) {
            $result = 'My Name Is ' . $result;
        }

        return $result;
    }
```

Plugin practice (see 05-plugin) 1/4

Plugin practice (see 05-plugin) 1/4

- In the previous module **Training/Helloworld**

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**

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 frontend plugin before the **setFirstname** method to transform the value in Title Case

Plugin practice (see 05-plugin) 2/4

Plugin practice (see 05-plugin) 2/4

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

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;

use Magento\Customer\Model\Data\Customer as ModelCustomer;

/**
 * Plugin Customer
 *
 * @author    Laurent MINGUET <lamin@smile.fr>
 * @copyright 2018 Smile
 */
class Customer
{
    // @todo
}
```

Plugin practice (see 05-plugin) 3/4

Plugin practice (see 05-plugin) 3/4

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

Plugin practice (see 05-plugin) 3/4

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

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

        return [$value];
    }
```

Plugin practice (see 05-plugin) 4/4

Plugin practice (see 05-plugin) 4/4

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

Plugin practice (see 05-plugin) 4/4

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

```xml
<?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

Definition

Rewrites

Definition

- Replace a Magento class by a specific one, to modify its behavior

Rewrites

Definition

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

Rewrites

How to **Rewrite** a class

Rewrites

How to **Rewrite** a class

- In the **./etc/module.xml**, the new module must depend on the original module to rewrite

Rewrites

How to **Rewrite** a class

- In the **./etc/module.xml**, the new module must depend 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

Rewrites

How to **Rewrite** a class

- In the **./etc/module.xml**, the new module must depend 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 extend the original rewrited class or implement the original interface

Rewrites

How to **Rewrite** a class

- In the **./etc/module.xml**, the new module must depend 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 extend the original rewrited class or implement the original interface
- Good practice: create the class in the **Rewrite** folder of your module

Rewrites

How to **Rewrite** a class

- In the **./etc/module.xml**, the new module must depend 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 extend the original rewrited class or implement the original interface
- Good practice: create the class in the **Rewrite** folder of your module
- Good practice: always prefer using plugins instead of rewrites

Rewrite practice (see 06-rewrite) 1/5

Rewrite practice (see 06-rewrite) 1/5

- In the previous module **Training/Helloworld**

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 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.

Rewrite practice (see 06-rewrite) 2/5

Rewrite practice (see 06-rewrite) 2/5

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

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;

use Magento\Catalog\Model\Product as BaseProduct;

/**
 * Rewrite \Magento\Catalog\Model\Product
 *
 * @author    Laurent MINGUET <lamin@smile.fr>
 * @copyright 2018 Smile
 */
class Product extends BaseProduct
{

}
```

Rewrite practice (see 06-rewrite) 3/5

Rewrite practice (see 06-rewrite) 3/5

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

Rewrite practice (see 06-rewrite) 3/5

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

```xml
<?xml version="1.0"?>
<config
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="urn:magento:framework:Module/etc/module.xsd"
>
    <module name="Training_Helloworld" setup_version="1.0.0">
        <sequence>
            <module name="Magento_Catalog"/>
        </sequence>
    </module>
</config>
```

Rewrite practice (see 06-rewrite) 4/5

Rewrite practice (see 06-rewrite) 4/5

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

Rewrite practice (see 06-rewrite) 4/5

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

```xml
<?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>
```

Rewrite practice (see 06-rewrite) 5/5

Rewrite practice (see 06-rewrite) 5/5

- Rewrite the **getName** method

Rewrite practice (see 06-rewrite) 5/5

- Rewrite the **getName** method

```php
<?php
/**
 * {@inheritdoc}
 */
public function getName()
{
    return parent::getName() . ' (Hello World)';
}
```

Plan

6 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

6 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

Model, Resource, Collection, and Entity Manager

Magento **Object Relational Mapping** elements

- **Models**: data + behavior; entities

Model, Resource, Collection, and Entity Manager

Magento **Object Relational Mapping** elements

- **Models**: data + behavior; entities
- **Resource Models**: data mappers for storage structure (legacy, logic is now deferred to an entity manager)

Model, Resource, Collection, and Entity Manager

Magento **Object Relational Mapping** elements

- **Models**: data + behavior; entities
- **Resource Models**: data mappers for storage structure (legacy, logic is now deferred to an entity manager)
- **Collections**: model sets & related functionality, such as filtering, sorting, and paging

Model, Resource, Collection, and Entity Manager

Magento **Object Relational Mapping** elements

- **Models**: data + behavior; entities
- **Resource Models**: data mappers for storage structure (legacy, logic is now deferred to an entity manager)
- **Collections**: model sets & related functionality, such as filtering, sorting, and paging
- **Entity Manager**: load, save, delete entities

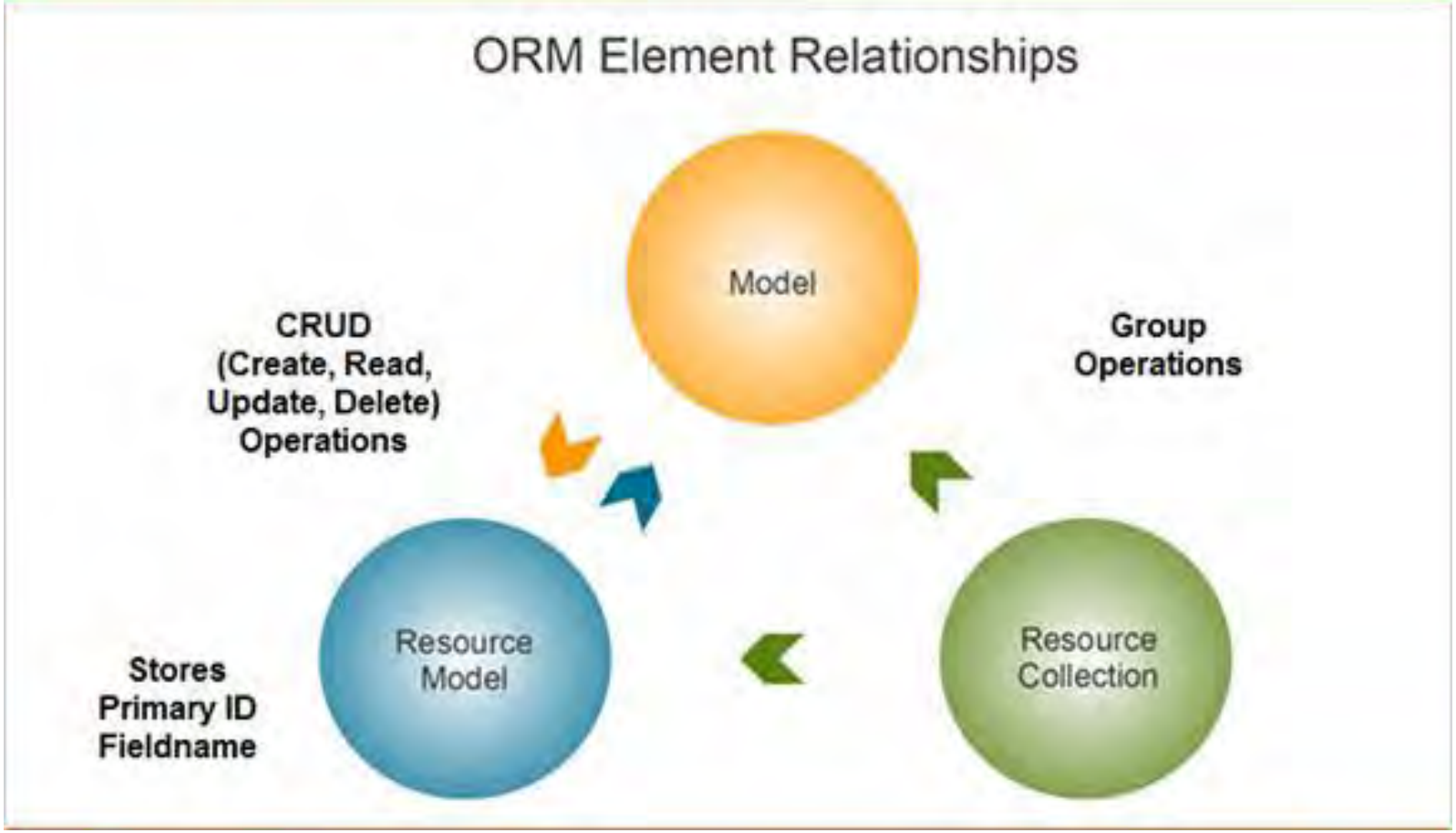
Model, Resource, Collection, and Entity Manager

Magento **Object Relational Mapping** elements

- **Models**: data + behavior; entities
- **Resource Models**: data mappers for storage structure (legacy, logic is now deferred to an 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



Model, Resource, Collection, and Entity Manager

- Not all **Models** are ORM entities

Model, Resource, Collection, and Entity Manager

- Not all **Models** are ORM entities
- ORM entities extends **AbstractModel**

Model, Resource, Collection, and Entity Manager

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

Model, Resource, Collection, and Entity Manager

Model

- **AbstractModel** provides generic behaviors

Model, Resource, Collection, and Entity Manager

Model

- **AbstractModel** provides generic behaviors
- It also provides old legacy CRUD operations (via the Resource Model)
  - load(): Read
  - save(): Create & Update
  - delete(): Delete

Model, Resource, Collection, and Entity Manager

Model

- **AbstractModel** provides generic behaviors
- It also provides old legacy CRUD operations (via the Resource Model)
  - load(): Read
  - save(): Create & Update
  - delete(): Delete
  - These methods are deprecated and must not be used
    Instead, use the entity manager, or the methods declared in the resource model

Model, Resource, Collection, and Entity Manager

Link the **Model** to the **Resource Model**

Link the **Model** to the **Resource Model**

```php
<?php
namespace Magento\Cms\Model;

use Magento\Cms\Api\Data\BlockInterface;

class Block extends \Magento\Framework\Model\AbstractModel implements BlockInterface
{
    ...

    protected function _construct()
    {
        $this->_init(
            \Magento\Cms\Model\ResourceModel\Block::class
        );
    }

    ...
}
```

The method **_init** must be provided with the name of the **Resource Model** class

Resource Model

- Extends
  \Magento\Framework\Model\ResourceModel\Db\AbstractDb

Resource Model

- Extends **\Magento\Framework\Model\ResourceModel\Db\AbstractDb**
- Has legacy save, delete, load methods, but you must not use them because it does not use the new entity manager

Resource Model

- Extends **\Magento\Framework\Model\ResourceModel\Db\AbstractDb**
- Has legacy save, delete, load methods, but you must not use them because it does not use the new entity manager
- You must redefine those methods to use the entity manager

Resource Model

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

Model, Resource, Collection, and Entity Manager

Link the **Resource Model** to the **Database**

Link the **Resource Model** to the **Database**

```php
<?php
namespace Magento\Cms\Model\ResourceModel;

class Block extends \Magento\Framework\Model\ResourceModel\Db\AbstractDb
{
    ...

    protected function _construct()
    {
        $this->_init(
            \Magento\Cms\Api\Data\BlockInterface::TABLE_NAME,
            \Magento\Cms\Api\Data\BlockInterface::FIELD_BLOCK_ID
        );
    }

    ...
}
```

The method **_init** must be provided with the names of the **Database** table and primary key.

Entity Manager

- Class **\Magento\Framework\EntityManager\EntityManager**

Entity Manager

- Class **\Magento\Framework\EntityManager\EntityManager**
- Provides the CRUD methods (save, delete, load)

Entity Manager

- Class **\Magento\Framework\EntityManager\EntityManager**
- Provides the CRUD methods (save, delete, load)
- Uses the metadata tool defined via dependency injection

Model, Resource, Collection, and Entity Manager

Entity Manager

- Class **\Magento\Framework\EntityManager\EntityManager**
- Provides the CRUD methods (save, delete, load)
- Uses the metadata tool defined via dependency injection
- Has access to the database

Entity Manager

- It provides automatic events

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

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

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
- Search for ->**dispatchEntityEvent** to find them

Link the **Collection Model** to the **Model** and **Resource Model**

Link the **Collection Model** to the **Model** and **Resource Model**

```php
<?php
namespace Magento\Cms\Model\ResourceModel\Block;

use \Magento\Framework\Model\ResourceModel\Db\Collection\AbstractCollection;

class Collection extends AbstractCollection
{
    ...

    protected function _construct()
    {
        $this->_init(
            \Magento\Cms\Model\Block::class,
            \Magento\Cms\Model\ResourceModel\Block::class
        );
        ...
    }

    ...
}
```

The method **_init** must be provided with the names of the **Model** and the **Resource Model**.

Collection filtering with methods:

- **addFieldToFilter** (for flat models)
- **addAttributeToFilter** (for eav models)

Model, Resource, Collection, and Entity Manager

Collection filtering with methods:

- **addFieldToFilter** (for flat models)
- **addAttributeToFilter** (for eav models)
- Example:

```
$blockCollection->addFieldToFilter('block_id', ['in' => $neededIds]);
```

Collection filtering with methods:

- **addFieldToFilter** (for flat models)
- **addAttributeToFilter** (for eav models)
- Example:

```
$blockCollection->addFieldToFilter('block_id', ['in' => $neededIds]);
```

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

Models and Cache

Models and Cache

- When saving / deleting a model, Magento can automatically purge the cache of the blocks and pages that use this model

Models and Cache

- When saving / deleting a model, Magento can automatically purge the cache of the blocks and pages that use this model
- The model must:
  - implement \Magento\Framework\DataObject\IdentityInterface

Models and Cache

- When saving / deleting a model, Magento can automatically purge the cache of the blocks and pages that use this model
- The model must:
  - implement \Magento\Framework\DataObject\IdentityInterface
  - have a constant **CACHE_TAG** that defines the prefix of the cache tags

Models and Cache

- When saving / deleting a model, Magento can automatically purge the cache of the blocks and pages that use this model
- The model must:
  - implement \Magento\Framework\DataObject\IdentityInterface
  - have a constant **CACHE_TAG** that defines the prefix of the cache tags
  - have the method **getIdentities** that returns the list of the cache tags

Plan

Entity Attribute Value Schema

- EAV = Entity Attribute Value

Entity Attribute Value Schema

- EAV = Entity Attribute Value
  - "Flat" mode (common):
    - 1 object type <=>1 table
    - object attributes <=>columns of the tables

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

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

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

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
    - ...

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
    - ...

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)**

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**

EAV in Magento

- EAV used for the most important objects in Magento
  - Product
  - Category
  - Customer
  - Customer address
  - ...

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

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

Attributes in Magento

- Definition
  - Characteristics of a model
  - Each model has a specific list of attributes (product attributes, customer attributes...)

Attributes in Magento

- Definition
  - Characteristics of a model
  - Each model has a specific list of attributes (product attributes, customer attributes...)
- Type and values
  - An attribute has a type, close to MySQL
  - Types are:
    - static (directly in the main entity table. Ex: product's sku)
    - int
    - decimal
    - varchar
    - textarea (can contain HTML code)
    - datetime
    - select (uses an association table of ID ->value[s])
    - multiselect (uses the same association table)

Product attributes

- Product attributes are stored using EAV

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

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

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%';
```

Product attribute sets

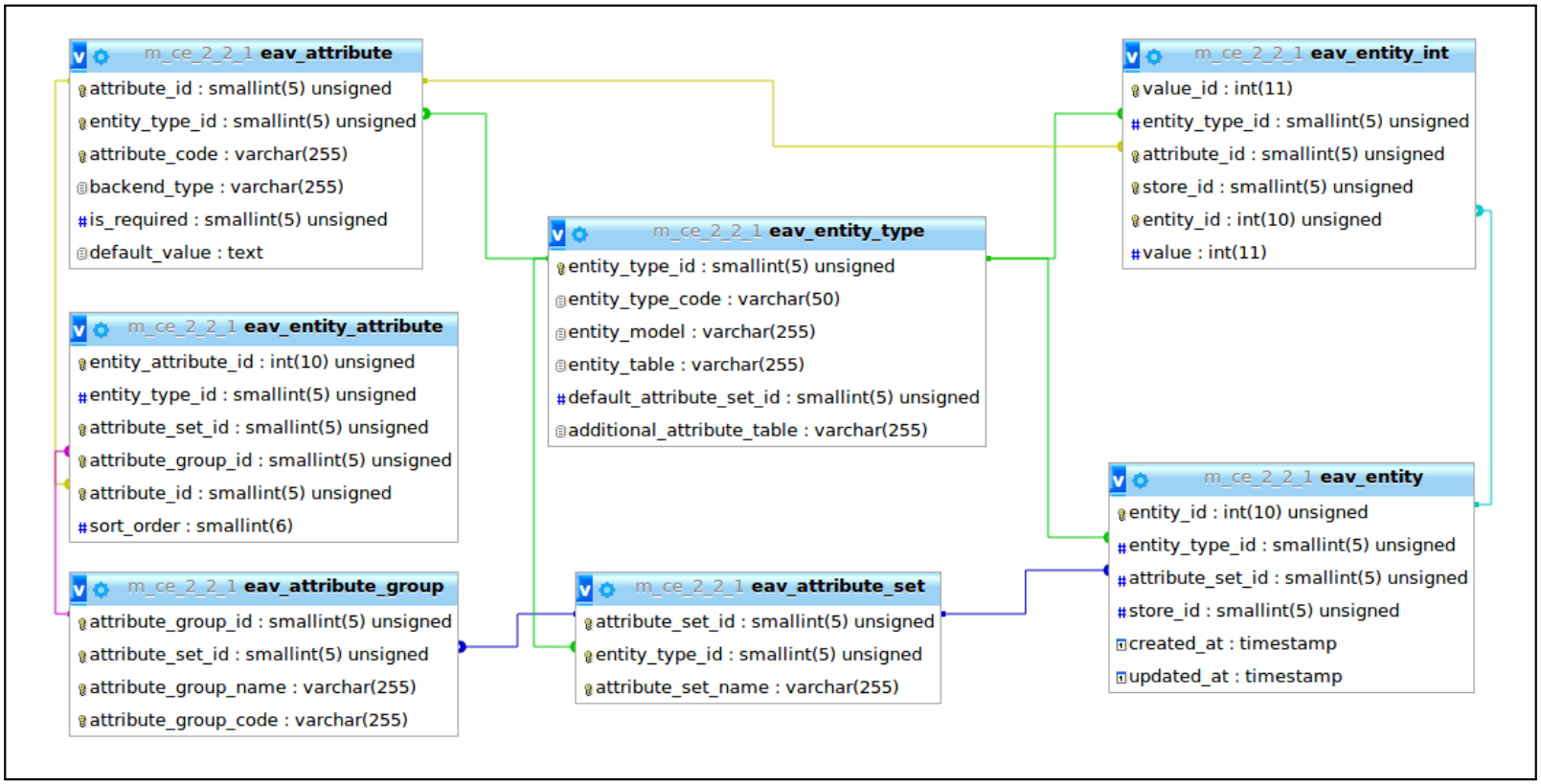
- An **attribute set** represents a list of attributes

Product attribute sets

- An **attribute set** represents a list of attributes
- Each EAV entity is mapped to an attribute set, and will use only the attributes defined in the set

Product attribute sets

- An **attribute set** represents a list of attributes
- Each EAV entity is mapped to an attribute set, and will use only the attributes defined in the set
- Example (products):
  - T-shirt attribute set uses a color and a size attribute
  - Book attribute set has no color nor size, but uses a page_nb and an author attribute


m_ce_2_2_1 eav_attribute
attribute_id : smallint(5) unsigned
entity_type_id : smallint(5) unsigned
attribute_code : varchar(255)
backend_type : varchar(255)
is_required : smallint(5) unsigned
default_value : text
m_ce_2_2_1 eav_entity_int
value_id : int(11)
entity_type_id : smallint(5) unsigned
attribute_id : smallint(5) unsigned
store_id : smallint(5) unsigned
entity_id : int(10) unsigned
value : int(11)
m_ce_2_2_1 eav_entity_type
entity_type_id : smallint(5) unsigned
entity_type_code : varchar(50)
entity_model : varchar(255)
entity_table : varchar(255)
default_attribute_set_id : smallint(5) unsigned
additional_attribute_table : varchar(255)
m_ce_2_2_1 eav_entity_attribute
entity_attribute_id : int(10) unsigned
entity_type_id : smallint(5) unsigned
attribute_set_id : smallint(5) unsigned
attribute_group_id : smallint(5) unsigned
attribute_id : smallint(5) unsigned
sort_order : smallint(6)
m_ce_2_2_1 eav_entity
entity_id : int(10) unsigned
entity_type_id : smallint(5) unsigned
attribute_set_id : smallint(5) unsigned
store_id : smallint(5) unsigned
created_at : timestamp
updated_at : timestamp
m_ce_2_2_1 eav_attribute_group
attribute_group_id : smallint(5) unsigned
attribute_set_id : smallint(5) unsigned
attribute_group_name : varchar(255)
attribute_group_code : varchar(255)
m_ce_2_2_1 eav_attribute_set
attribute_set_id : smallint(5) unsigned
entity_type_id : smallint(5) unsigned
attribute_set_name : varchar(255)

Plan

Model - Practice

Model practice (see 07-model) 1/6

Model - Practice

Model practice (see 07-model) 1/6

- In the previous module **Training/Helloworld**

Model - Practice

Model practice (see 07-model) 1/6

- In the previous module **Training/Helloworld**
- Create a new frontend controller product/categories

Model - Practice

Model practice (see 07-model) 1/6

- In the previous module **Training/Helloworld**
- Create a new frontend controller product/categories
- Ask for the Product and Category Collection factories in the constructor

Model - Practice

Model practice (see 07-model) 1/6

- In the previous module **Training/Helloworld**
- Create a new frontend controller product/categories
- Ask for the Product and Category Collection factories in the constructor
- Get all the products where the name contains "bag"

Model - Practice

Model practice (see 07-model) 1/6

- In the previous module **Training/Helloworld**
- Create a new frontend controller 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 load)

Model - Practice

Model practice (see 07-model) 1/6

- In the previous module **Training/Helloworld**
- Create a new frontend controller 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 load)
- For each product, display its name and the list the name of its associated categories

Model - Practice

Model practice (see 07-model) 2/6

Model - Practice

Model practice (see 07-model) 2/6

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

Model practice (see 07-model) 2/6

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

```php
<?php
/**
 * Magento 2 Training Project
 * Module Training/Helloworld
 */
namespace Training\Helloworld\Controller\Product;
use Magento\Framework\App\Action\Action;
use Magento\Framework\Controller\Result\Raw as ResultRaw;
use Magento\Framework\Controller\ResultFactory;
/**
 * Action: Product/Categories
 *
 * @author    Laurent MINGUET <lamin@smile.fr>
 * @copyright 2018 Smile
 */
class Categories extends Action
{
    /**
     * {@inheritdoc}
     */
    public function execute()
    {
        /** @var ResultRaw $result */
        $result = $this->resultFactory->create(ResultFactory::TYPE_RAW);
        $result->setContents('@todo');
        return $result;
    }
}
```

Model - Practice

Model practice (see 07-model) 3/6

Model - Practice

Model practice (see 07-model) 3/6

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

Model practice (see 07-model) 3/6

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

```php
<?php
...
use Magento\Catalog\Model\ResourceModel\Product\CollectionFactory  as ProductCollectionFactory;
use Magento\Catalog\Model\ResourceModel\Category\CollectionFactory as CategoryCollectionFactory;
...
    /** @var ProductCollectionFactory */
    protected $productCollectionFactory;

    /** @var CategoryCollectionFactory */
    protected $categoryCollectionFactory;

    /**
     * @param Context                    $context
     * @param ProductCollectionFactory   $productCollectionFactory
     * @param CategoryCollectionFactory  $categoryCollectionFactory
     */
    public function __construct(
        Context $context,
        ProductCollectionFactory  $productCollectionFactory,
        CategoryCollectionFactory $categoryCollectionFactory
    ) {
        parent::__construct($context);
        $this->productCollectionFactory  = $productCollectionFactory;
        $this->categoryCollectionFactory = $categoryCollectionFactory;
    }
```

Model - Practice

Model practice (see 07-model) 4/6

Model - Practice

Model practice (see 07-model) 4/6

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

Model - Practice

Model practice (see 07-model) 4/6

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

```php
<?php
...
use Magento\Catalog\Model\ResourceModel\Product\Collection  as ProductCollection;
...

    public function execute()
    {
        /** @var ProductCollection $productCollection */
        $productCollection = $this->productCollectionFactory->create();
        $productCollection
            ->addAttributeToFilter('name', array('like' => '%bag%'))
            ->addCategoryIds()
            ->load();

        ...
    }
```

Model - Practice

Model practice (see 07-model) 5/6

Model - Practice

Model practice (see 07-model) 5/6

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

Model practice (see 07-model) 5/6

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

```php
<?php
...
use Magento\Catalog\Model\Product as ProductModel;
use Magento\Catalog\Model\ResourceModel\Category\Collection as CategoryCollection;
...

    public function execute()
    {
        ...

        $categoryIds = [];
        foreach ($productCollection->getItems() as $product) {
            /** @var ProductModel $product */
            $categoryIds = array_merge($categoryIds, $product->getCategoryIds());
        }
        $categoryIds = array_unique($categoryIds);

        /** @var CategoryCollection $catCollection */
        $catCollection = $this->categoryCollectionFactory->create();
        $catCollection
            ->addAttributeToFilter('entity_id', array('in' => $categoryIds))
            ->addAttributeToSelect('name')
            ->load();

        ...
    }
```

Model - Practice

Model practice (see 07-model) 6/6

Model - Practice

Model practice (see 07-model) 6/6

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

Model practice (see 07-model) 6/6

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

```php
<?php
...
use Magento\Catalog\Model\Category as CategoryModel;
...

    public function execute()
    {
        ...
        $html = '<ul>';
        foreach ($productCollection->getItems() as $product) {
            $html.= '<li>';
            $html.= $product->getId() . ' => ' . $product->getSku() . ' => ' . $product->getName();
            $html.= '<ul>';
            foreach ($product->getCategoryIds() as $categoryId) {
                /** @var CategoryModel $category */
                $category = $catCollection->getItemById($categoryId);
                $html.= '<li>' . $categoryId . ' => ' . $category->getName() . '</li>';
            }
            $html.= '</ul>';
            $html.= '</li>';
        }
        $html.= '</ul>';

        /** @var ResultRaw $result */
        $result = $this->resultFactory->create(ResultFactory::TYPE_RAW);
        $result->setContents($html);
        return $result;
    }
```

Plan

Api, Data, and Repository

- API = all the interfaces in the api folder of a module, or classes with the @api phpdoc tag

Api, Data, and Repository

- API = all the interfaces in the api folder of a module, or classes with the @api phpdoc tag
- Modules should only communicate through the API

Api, Data, and Repository

- API = all the interfaces in the api folder of a module, or classes with the @api phpdoc tag
- Modules should only communicate through the API
- Never use a no-api class of an external module.

Api, Data, and Repository

- API = all the interfaces in the api folder of a module, or classes with the @api phpdoc tag
- Modules should only communicate through the API
- Never use a no-api class of an external module.
- Use only the API interfaces and its declared methods

Api, Data, and Repository

- API = all the interfaces in the api folder of a module, or classes with the @api phpdoc tag
- Modules should only communicate through the API
- Never use a no-api 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 its api interface / class

Api, Data, and Repository

3 main **API groups**

- Data Api
- Repository API
- Operational API

Api, Data, and Repository

Data Api

- In the **./Api/Data/** folder

Api, Data, and Repository

Data Api

- In the **./Api/Data/** folder
- Interfaces named **XxxxInterface**

Api, Data, and Repository

Data Api

- In the **./Api/Data/** folder
- Interfaces named **XxxxInterface**
- Only access to the data of an object via setter and getter

Api, Data, and Repository

Data Api

- In the **./Api/Data/** folder
- Interfaces named **XxxxInterface**
- Only access to the data of an object via setter and getter
- No CRUD operations

Api, Data, and Repository

Data Api

- In the **./Api/Data/** folder
- Interfaces named **XxxxInterface**
- Only access to the data of an 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

Api, Data, and Repository

Repository Api

- In the **./Api/** folder

Api, Data, and Repository

Repository Api

- In the **./Api/** folder
- Interfaces named **XxxxRepositoryInterface**

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

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 Results Interface**
  Example:
  \Magento\Customer\Api\Data\CustomerSearchResultsInterface

Api, Data, and Repository

Operational Api

- In the **./Api/** folder

Api, Data, and Repository

Operational Api

- In the **./Api/** folder
- Interfaces named **XxxxManagementInterface**

Api, Data, and Repository

Operational Api

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

  Example: \Magento\Customer\Api\AccountManagementInterface

Api, Data, and Repository

- Ability to customize based on the documentation

Api, Data, and Repository

- Ability to customize based on the documentation
- Better decoupling

Api, Data, and Repository

- Ability to customize based on the documentation
- Better decoupling
- Minimizing conflicts

Api, Data, and Repository

- Ability to customize based on the documentation
- Better decoupling
- Minimizing conflicts
- Ability to rely on the interface, not on implementation

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 upgrades are much safer to execute without anything breaking

Repository Implementation Example - Load by Id

```php
<?php
    /**
     * Retrieve a entity by its ID
     *
     * @param int $objectId
     *
     * @return AbstractModel
     * @throws NoSuchEntityException
     */
    public function getEntityById($objectId)
    {
        if (!isset($this->cacheById[$objectId])) {
            $object = $this->objectFactory->create();
            $this->objectResource->load($object, $objectId);

            if (!$object->getId()) {
                throw NoSuchEntityException::singleField('objectId', $objectId);
            }

            $this->cacheById[$objectId] = $object;
            $this->cacheByIdentifier[$object->getIdentifier()] = $object;
        }

        return $this->cacheById[$objectId];
    }
```

Repository Implementation Example - Load by Identifier

```php
<?php
    /**
     * Retrieve a entity by its identifier
     *
     * @param string $objectIdentifier
     *
     * @return AbstractModel
     * @throws NoSuchEntityException
     */
    public function getEntityByIdentifier($objectIdentifier)
    {
        if (!isset($this->cacheByIdentifier[$objectIdentifier])) {
            $object = $this->objectFactory->create();
            $this->objectResource->load($object, $objectIdentifier, 'identifier');

            if (!$object->getId()) {
                throw NoSuchEntityException::singleField('objectIdentifier', $objectIdentifier);
            }

            $this->cacheById[$object->getId()] = $object;
            $this->cacheByIdentifier[$objectIdentifier] = $object;
        }

        return $this->cacheByIdentifier[$objectIdentifier];
    }
```

Repository Implementation Example - Save

```php
<?php
    /**
     * Save entity
     *
     * @param AbstractModel $object
     *
     * @return AbstractModel
     * @throws CouldNotSaveException
     */
    public function saveEntity(AbstractModel $object)
    {
        try {
            $this->objectResource->save($object);

            unset($this->cacheById[$object->getId()]);
            unset($this->cacheByIdentifier[$object->getIdentifier()]);
        } catch (\Exception $e) {
            $msg = new Phrase($e->getMessage());
            throw new CouldNotSaveException($msg);
        }

        return $object;
    }
```

Repository Implementation Example - Delete

```php
<?php
    /**
     * Delete entity
     *
     * @param AbstractModel $object
     *
     * @return boolean
     * @throws CouldNotDeleteException
     */
    public function deleteEntity(AbstractModel $object)
    {
        try {
            $this->objectResource->delete($object);

            unset($this->cacheById[$object->getId()]);
            unset($this->cacheByIdentifier[$object->getIdentifier()]);
        } catch (\Exception $e) {
            $msg = new Phrase($e->getMessage());
            throw new CouldNotDeleteException($msg);
        }

        return true;
    }
```

Repository Implementation Example: Search and Get a List

```php
<?php
    /**
     * Retrieve  entities which match a specified criteria.
     *
     * @param SearchCriteriaInterface $searchCriteria search criteria
     *
     * @return SearchResults
     */
    public function getEntities(SearchCriteriaInterface $searchCriteria = null)
    {
        $collection = $this->objectCollectionFactory->create();

        $searchResults = $this->searchResultsFactory->create();

        if ($searchCriteria) {
            $searchResults->setSearchCriteria($searchCriteria);
            $this->collectionProcessor->process($searchCriteria, $collection);
        }

        // load the collection
        $collection->load();

        // build the result
        $searchResults->setTotalCount($collection->getSize());
        $searchResults->setItems($collection->getItems());

        return $searchResults;
    }
```

Repository Implementation Example: Collection Processor

- API: \Magento\Framework\Api\SearchCriteria\CollectionProcessorInterface

Repository Implementation Example: Collection Processor

- API: \Magento\Framework\Api\SearchCriteria\CollectionProcessorInterface
- Class: \Magento\Framework\Api\SearchCriteria\CollectionProcessor

Repository Implementation Example: Collection Processor

- API: \Magento\Framework\Api\SearchCriteria\CollectionProcessorInterface
- Class: \Magento\Framework\Api\SearchCriteria\CollectionProcessor
- Processors list: ./app/etc/di.xml line 1243

Repository Implementation Example: Collection Processor

- API: \Magento\Framework\Api\SearchCriteria\CollectionProcessorInterface
- Class: \Magento\Framework\Api\SearchCriteria\CollectionProcessor
- Processors list: ./app/etc/di.xml line 1243
- \Magento\Framework\Api\SearchCriteria\CollectionProcessor\FilterProcessor
- \Magento\Framework\Api\SearchCriteria\CollectionProcessor\SortingProcessor
- \Magento\Framework\Api\SearchCriteria\CollectionProcessor\PaginationProcessor

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

Order Builder methods

- setField
- setDescendingDirection
- setAscendingDirection
- 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

Alternative (and more concise) way to build a filter

```php
<?php
$this->searchCriteriaBuilder->addFilter('name', '%book%', 'like');
```

As defined in

\Magento\Framework\Api\SearchCriteriaBuilder

```php
<?php
public function addFilter($field, $value, $conditionType = 'eq')
{
    $this->addFilters([
        $this->filterBuilder->setField($field)
            ->setValue($value)
            ->setConditionType($conditionType)
            ->create()
    ]);
    return $this;
}
```

API practice (see 08-api) 1/5

API practice (see 08-api) 1/5

- In the previous module **Training/Helloworld**

API practice (see 08-api) 1/5

- In the previous module **Training/Helloworld**
- Create a new frontend controller product/search

API practice (see 08-api) 1/5

- In the previous module **Training/Helloworld**
- Create a new frontend controller product/search
- Ask for the following object in the constructor
  - ProductRepositoryInterface
  - SearchCriteriaBuilder
  - FilterBuilder
  - SortOrderBuilder

API practice (see 08-api) 1/5

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

API practice (see 08-api) 2/5

API practice (see 08-api) 2/5

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

API practice (see 08-api) 2/5

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

```php
<?php
/**
 * Magento 2 Training Project
 * Module Training/Helloworld
 */
namespace Training\Helloworld\Controller\Product;
use Magento\Framework\App\Action\Action;
use Magento\Framework\Controller\Result\Raw as ResultRaw;
use Magento\Framework\Controller\ResultFactory;
/**
 * Product Controller, action Search
 *
 * @author    Laurent MINGUET <lamin@smile.fr>
 * @copyright 2018 Smile
 */
class Search extends Action
{
    /**
     * {@inheritdoc}
     */
    public function execute()
    {
        /** @var ResultRaw $result */
        $result = $this->resultFactory->create(ResultFactory::TYPE_RAW);
        $result->setContents('@todo');
        return $result;
    }
}
```

API practice (see 08-api) 3/5

Api, Data, and Repository

API practice (see 08-api) 3/5

- Ask for objects in the constructor

API practice (see 08-api) 3/5

- Ask for objects in the constructor

```php
<?php
...
use Magento\Catalog\Api\ProductRepositoryInterface;
use Magento\Framework\Api\FilterBuilder;
use Magento\Framework\Api\SearchCriteriaBuilder;
use Magento\Framework\Api\SortOrderBuilder;
use Magento\Framework\App\Action\Context;
...

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

    public function __construct(
        Context                     $context,
        ProductRepositoryInterface $productRepository,
        SearchCriteriaBuilder       $searchCriteriaBuilder,
        FilterBuilder               $filterBuilder,
        SortOrderBuilder            $sortOrderBuilder
    ) {
        parent::__construct($context);

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

API practice (see 08-api) 4/5

API practice (see 08-api) 4/5

- Get the products list

API practice (see 08-api) 4/5

- Get the products list

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

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

        $sortOrder = $this->sortOrderBuilder
            ->setField('name')->setDescendingDirection()
            ->create();

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

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

API practice (see 08-api) 5/5

API practice (see 08-api) 5/5

- Display the result

API practice (see 08-api) 5/5

- Display the result

```php
<?php
    /**
     * {@inheritdoc}
     */
    public function execute()
    {
        $products = $this->getProductList();

        $html = '<ul>';
        foreach ($products as $product) {
            $html.= '<li>' . $product->getSku() . ' => ' . $product->getName() . '</li>';
        }
        $html.= '</ul>';

        /** @var ResultRaw $result */
        $result = $this->resultFactory->create(ResultFactory::TYPE_RAW);
        $result->setContents($html);

        return $result;
    }
```

Plan

Web Api

- Allows exposure of the **Module API** through the Web API

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

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

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
- Steps 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

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 Web API URL to use for REST
- **route[method]**:
  the HTTP method to use for REST
- **service[class]**:
  the API interface that to use for this URL and HTTP method
- **service[method]**:
  the API method to call for this URL and HTTP method
- **resources**:
  the list of the required ACL roles to access this URL

REST Webservice API

Web Api

REST Webservice API

- Create an admin user with the ACLs you want to use

Web Api

REST Webservice API

- Create an 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

Web Api

REST Webservice API

- Create an 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
  $\Rightarrow$ it will return a token that must be used in all following requests using the header "Authorization: Bearer TOKEN"

Web Api

REST Webservice API

- Create an 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
  $\Rightarrow$ it will return a token that must be used in all following requests using the header "Authorization: Bearer TOKEN"
- Then make the request(s) you want

REST Webservice API

- Create an 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 following requests using the header "Authorization: Bearer TOKEN"
- Then make the request(s) you want

See the example file
./09-apiweb/_extra/scripts/testModules/catalog/rest.php

SOAP Webservice API

Web Api

SOAP Webservice API

- Init a SOAP token in System >Integration

SOAP Webservice API

- 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

SOAP Webservice API

- 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
- 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

SOAP Webservice API

- 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
- 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

See the example file
./09-apiweb/_extra/scripts/testModules/catalog/soap.php

Plan

6 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

Setup: install and upgrade

- In the **Setup** folder of each module

Setup: install and upgrade

- In the **Setup** folder of each module
- 4 setup files:
  - **InstallSchema.php**
  - **UpgradeSchema.php**
  - **InstallData.php**
  - **UpgradeData.php**

Setup: install and upgrade

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

```xml
<?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>
```

Setup: install and upgrade

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

```xml
<?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

Setup: install and upgrade

class **InstallSchema**

Setup: install and upgrade

class **InstallSchema**

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

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();
    }
}
```

Setup: install and upgrade

class **UpgradeSchema**

Setup: install and upgrade

class **UpgradeSchema**

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

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();
    }
}
```

Setup: install and upgrade

class **InstallData**

Setup: install and upgrade

class **InstallData**

- Implements **InstallDataInterface**
- Must contain only insertions or modifications of data
- Executed after Schema setups
- Executed only once during the first install of the module

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)
    {
        ...
    }
}
```

Setup: install and upgrade

class **UpgradeData**

Setup: install and upgrade

class **UpgradeData**

- Implements **UpgradeDataInterface**
- Must contain only insertions or modifications of data
- Executed after Schema setups
- Executed after an install and upon subsequent upgrades
- One class for all the version updates, with test on the current version of the module

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', '<')) {
            ...
        }

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

Plan

6 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

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

Seller Module (see 10-seller-part1)

- In a new module **Training/Seller**

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

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**

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 (unsigned int, primary key, autoincrement)
  - identifier (varchar 64, required)
  - name (varchar 255, required)
  - created_at (timestamp, automatic init)
  - updated_at (timestamp, automatic init and update)

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**

- Describes 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/Data/SellerSearchResultsInterface.php**

- Describes 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[]

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

File **Api/SellerRepositoryInterface.php**

- Describes what the repository will expose
- getById will take an integer and will return a SellerInterface
- getByIdentifier will take a string and will return a SellerInterface
- getList will take a search criteria, and will return a Seller Search Result
- save will take a SellerInterface and will return the saved SellerInterface
- deleteById will take an integer and will return true if the seller is deleted
- deleteByIdentifier will take a string and will return true if the seller is deleted
- Do not forget the exceptions NoSuchEntityException, CouldNotSaveException, CouldNotDeleteException

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

File **Model/Seller.php**

- Must extend Magento\Framework\Model\AbstractModel
- Must implement Magento\Framework\DataObject\IdentityInterface
- Must implement 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, using the constant CACHE_TAG
- The public method getSellerId must use the native method getId
- The public method setSellerId must use the native method setId
- The public method populateFromArray can be implemented

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

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

- Must extend Magento\Framework\Model\ResourceModel\Db\AbstractDb
- Must use the new entity manager and Metadata Pool
- The protected method _construct must call the method _init to link the resource model to the database
- The method **getConnection** allows to get the good mysql connection linked to the object threw the metadata pool
- The method **load** allows to load an object threw the entity manager
- The method **save** allows to save an object threw the entity manager
- The method **delete** allows to delete an object threw the entity manager

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 as a source for select/multiselect attributes

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

- Implements generic behavior for repositories
- It can be used in any repository that has to manage a flat model
- The following parameters will be injected manually:
  - The **Object Factory**, to generate the object
  - The **Object ResourceModel**, to make the CRUD operations
  - The **Object Collection Factory**, to load set of objects
  - The **Object SearchResultFactory**, to return the good search result
  - The **Object Identifier FieldName**, to know the identifier field name (sku, email, ...)

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

- The method **getEntityById** allows to load an entity by its id
- The method **getEntityByIdentifier** allows to load an entity by its identifier
- The method **saveEntity** allows to save an entity
- The method **deleteEntity** allows to delete an entity
- The method **deleteEntityById** allows to delete an entity by its id
- The method **deleteEntityByIdentifier** allows to delete an entity by its identifier
- The method **getEntities** allows to load a list of entities, regarding to a search criteria, using the Collection Processor

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

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

- Must implement the **SellerRepositoryInterface**
- Must use the new **repository manager**
- **Warning**, the repository manager is not injectable, use its factory
- Must implement all the methods of the interface

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 **Setup/InstallSchema.php**

- Defines the new table training_seller, with:
  - an auto-increment primary key on the seller_id field
  - a unique index on the identifier field
  - an automatic init and update of the fields created_at and updated_at

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

File **Setup/InstallData.php**

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

File **etc/acl.xml**

- Defines the new resource "Training_Seller::manage" that a user must have to be able to use the 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 **_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

7 Controller and View

- Routing
- Controller
- Practice - Seller - Part 2 - Router / Controller
- View and Layout
- Practice - Seller - Part 3 - Layout / Block / Template
- Practice - Seller - Part 4 - Layout Update
- Practice - Seller - Part 5 - Admin

Plan

Routing

- **Routing** converts a request URL into a format Magento can handle, and then finds the class that will be able to process it

Routing

- **Routing** converts a request URL into a format 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

Routing

- **Routing** converts a request URL into a format 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)

Routing

- **Routing** converts a request URL into a format 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)

Routing

- **Routing** converts a request URL into a format 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/)

Routing

- **Routing** converts a request URL into a format 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

Routing

The **routing** process:

- Managed by \Magento\Framework\App\FrontController::dispatch

Routing

The **routing** process:

- Managed by \Magento\Framework\App\FrontController::dispatch
- Iterates on a list of routers defined in \Magento\Framework\App\RouterList

Routing

The **routing** process:

- Managed by \Magento\Framework\App\FrontController::dispatch
- Iterates on a list of routers defined in \Magento\Framework\App\RouterList
- The front controller searches for a router that will
  - Understand the requested URL (e.g. /about-us)

Routing

The **routing** process:

- Managed by \Magento\Framework\App\FrontController::dispatch
- Iterates on a list of routers defined in \Magento\Framework\App\RouterList
- The front controller searches for a router that will
  - Understand the requested URL (e.g. /about-us)
  - Convert the URL to a Magento-style URL (e.g. /cms/page/view/id/5)

Routing

The **routing** process:

- Managed by \Magento\Framework\App\FrontController::dispatch
- Iterates on a list of routers defined in \Magento\Framework\App\RouterList
- The front controller searches for a router that will
  - Understand the requested URL (e.g. /about-us)
  - Convert the URL to a Magento-style URL (e.g. /cms/page/view/id/5)
  - Identify the controller class that will process the URL

Routing

The **routing** process:

- Managed by \Magento\Framework\App\FrontController::dispatch
- Iterates on a list of routers defined in \Magento\Framework\App\RouterList
- The front controller searches for a router that will
  - Understand the requested URL (e.g. /about-us)
  - Convert the URL to a Magento-style URL (e.g. /cms/page/view/id/5)
  - Identify the controller class that will process the URL
- The front controller will then execute the identified controller class

Routing

Routers:

- Implements **\Magento\Framework\App\RouterInterface** with method **match**

Routing

Routers:

- Implements **\Magento\Framework\App\RouterInterface**
  with method **match**
- Magento\Backend\App\Router
  Matches a **Magento-style URL** with a real backend controller class

Routing

Routers:

- Implements **\Magento\Framework\App\RouterInterface**
  with method **match**
- Magento\Backend\App\Router
  Matches a **Magento-style URL** with a real backend controller class
- Magento\Framework\App\Router\Base
  Matches a **Magento-style URL** with a real frontend controller class

Routing

Routers:

- Implements **\Magento\Framework\App\RouterInterface** with method **match**
- Magento\Backend\App\Router
  Matches a **Magento-style URL** with a real backend controller class
- Magento\Framework\App\Router\Base
  Matches a **Magento-style URL** with a real frontend controller class
- Magento\UrlRewrite\Controller\Router
  Uses the **url_rewrite** mysql table to match an URL with a Magento-style url.

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

Routing

Routers:

- Implements **\Magento\Framework\App\RouterInterface** with method **match**
- Magento\Backend\App\Router
  Matches a **Magento-style URL** with a real backend controller class
- Magento\Framework\App\Router\Base
  Matches a **Magento-style URL** with a real frontend controller class
- Magento\UrlRewrite\Controller\Router
  Uses the **url_rewrite** mysql table to match an URL with a Magento-style url.

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

- Magento\Cms\Controller\Router
  Matches a URL with a CMS page

Routing

Routers:

- Implements **\Magento\Framework\App\RouterInterface**
  with method **match**
- Magento\Backend\App\Router
  Matches a **Magento-style URL** with a real backend controller class
- Magento\Framework\App\Router\Base
  Matches a **Magento-style URL** with a real frontend controller class
- Magento\UrlRewrite\Controller\Router
  Uses the **url_rewrite** mysql table to match an URL with a Magento-style url.

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

- Magento\Cms\Controller\Router
  Matches a URL with a CMS page
- Magento\Robots\Controller\Router
  Matches a URL with the robots.txt file

Routing

Routers:

- Implements **\Magento\Framework\App\RouterInterface**
  with method **match**
- Magento\Backend\App\Router
  Matches a **Magento-style URL** with a real backend controller class
- Magento\Framework\App\Router\Base
  Matches a **Magento-style URL** with a real frontend controller class
- Magento\UrlRewrite\Controller\Router
  Uses the **url_rewrite** mysql table to match an URL with a Magento-style url.

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

- Magento\Cms\Controller\Router
  Matches a URL with a CMS page
- Magento\Robots\Controller\Router
  Matches a URL with the robots.txt file
- Magento\Framework\App\Router\DefaultRouter
  Executed after all the others, to manage forward and notfound (404)

Plan

Controller

- A controller class can only process a **single action**

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

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 manage ACLs (only users with the specified resource have access to the action)

Controller

Backend controller

- Extends **\Magento\Backend\App\Action**
  - Empty class, not useful...

Controller

Backend controller

- Extends **\Magento\Backend\App\Action**
  - Empty class, not useful...
- Extends **\Magento\Backend\App\AbstractAction**
  - Asks for the objects required to manage a backend Controller, like Autorization, Auth, Current Helper, Backend Url Manager, Form Key validator, Admin Session, ...

Controller

Backend controller

- Extends **\Magento\Backend\App\Action**
  - Empty class, not useful...
- Extends **\Magento\Backend\App\AbstractAction**
  - Asks for the objects required to manage a backend Controller, like Autorization, Auth, Current Helper, Backend Url Manager, Form Key validator, Admin Session, ...
- Extends **\Magento\Framework\App\Action\Action**
  - Asks for the objects required to manage a frontend Controller, like Object Manager, Event Manager, Frontend Url Manager, View Manager, Redirect Manager, Message Manager, ...

Controller

Backend controller

- Extends **\Magento\Backend\App\Action**
  - Empty class, not useful...
- Extends **\Magento\Backend\App\AbstractAction**
  - Asks for the objects required to manage a backend Controller, like Autorization, Auth, Current Helper, Backend Url Manager, Form Key validator, Admin Session, ...
- Extends **\Magento\Framework\App\Action\Action**
  - Asks for the objects required to manage a frontend Controller, like Object Manager, Event Manager, Frontend Url Manager, View Manager, Redirect Manager, Message Manager, ...
- Extends **\Magento\Framework\App\Action\AbstractAction**
  - Asks for the objects required to manage a standard Controller, like Request, Response, Result Redirect Factory, Result Factory, ...

Controller

Backend controller

- Extends **\Magento\Backend\App\Action**
  - Empty class, not useful...
- Extends **\Magento\Backend\App\AbstractAction**
  - Asks for the objects required to manage a backend Controller, like Autorization, Auth, Current Helper, Backend Url Manager, Form Key validator, Admin Session, ...
- Extends **\Magento\Framework\App\Action\Action**
  - Asks for the objects required to manage a frontend Controller, like Object Manager, Event Manager, Frontend Url Manager, View Manager, Redirect Manager, Message Manager, ...
- Extends **\Magento\Framework\App\Action\AbstractAction**
  - Asks for the objects required to manage a standard Controller, like Request, Response, Result Redirect Factory, Result Factory, ...
- Implements **\Magento\Framework\App\ActionInterface**
  - Has only one method: **execute()**

Plan

Practice - Seller - Part 2 - Router / Controller

Seller Module (see 11-seller-part2)

- In the module **Training/Seller**

Practice - Seller - Part 2 - Router / 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

Practice - Seller - Part 2 - Router / 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

Practice - Seller - Part 2 - Router / 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 a new router to match the URLs /sellers.html and /seller/[identifier].html

Practice - Seller - Part 2 - Router / 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 a new router to match the URLs /sellers.html and /seller/[identifier].html
- Create new backend action training_seller/seller/index to display the name of the main seller

Practice - Seller - Part 2 - Router / Controller

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

- Defines the frontname to use for the frontend controllers of the module : seller
- The router id to use is **standard**

Practice - Seller - Part 2 - Router / Controller

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

- Generic behavior for the seller actions
- Asks for the **seller repository**
- Asks for the **search criteria builder**

Practice - Seller - Part 2 - Router / Controller

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

- Displays the list of the sellers, using the repository

Practice - Seller - Part 2 - Router / Controller

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

- Displays a specific seller, using the repository
- Throws a **NotFoundException** exception if the asked seller does not exist

Practice - Seller - Part 2 - Router / Controller

File **Controller/Router.php**

- New router to manage the URLs /sellers.html and /seller/[identifier].html
- Asks for the action factory in the constructor
- Must implement the public method match
  - Returns an action if the router can handle the URL
  - Returns NULL otherwise

Practice - Seller - Part 2 - Router / Controller

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

- Add the new router to the router list

Practice - Seller - Part 2 - Router / Controller

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

- Defines the frontname to use for the backend controllers of the module: training_seller
- The router id to use is **admin**

Practice - Seller - Part 2 - Router / Controller

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

- Defines the new entry Training Seller in the admin menu

Practice - Seller - Part 2 - Router / Controller

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

- Generic behaviors for the seller actions
- Asks for the **seller repository** in the constructor
- Overrides the constant **ADMIN_RESOURCE** with the ACL resource **Training_Seller::manage**

Practice - Seller - Part 2 - Router / Controller

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

- Generic behaviors for the seller actions
- Asks for the **seller repository** in the constructor
- Overrides the constant **ADMIN_RESOURCE** with the ACL resource **Training_Seller::manage**

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

- Displays the name of the main seller

Plan

View and Layout

Templates (phtml)

View and Layout

Templates (phtml)

- Handle HTML, JavaScript, and some PHP

View and Layout

Templates (phtml)

- Handle HTML, JavaScript, and some PHP
- Each template is rendered by a Block (PHP class)

View and Layout

Templates (phtml)

- Handle HTML, JavaScript, and some PHP
- Each template is rendered by a Block (PHP class)
- In the folder **./view/[AREA]/templates/** of each module

Blocks (PHP)

Blocks (PHP)

- Allow you to move reusable functionality from template files into classes
- The same block can be assigned to multiple templates

Blocks (PHP)

- Allow you to move reusable functionality from template files into classes
- The same block can be assigned to multiple templates
- Extend the class **\Magento\Framework\View\Element\AbstractBlock**
- Implement the interface **\Magento\Framework\View\Element\BlockInterface**

Blocks (PHP)

- Allow you to move reusable functionality from template files into classes
- The same block can be assigned to multiple templates
- Extend the class **\Magento\Framework\View\Element\AbstractBlock**
- Implement the interface **\Magento\Framework\View\Element\BlockInterface**
- In the folder **./Block/** of each module

Blocks (PHP)

- Allow you to move reusable functionality from template files into classes
- The same block can be assigned to multiple templates
- Extend the class **\Magento\Framework\View\Element\AbstractBlock**
- Implement the interface **\Magento\Framework\View\Element\BlockInterface**
- In the folder **./Block/** of each module
- Be careful: specific blocks for frontend or backend
  - \Magento\Framework\View\Element\Template
  - \Magento\Backend\Block\Template

View and Layout

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

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;
    }
```

Blocks and Cache

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**: an array that contains the tags of the models used by this block (deprecated, use getIdentities instead)

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**: an array that contains the tags of the models used by this block (deprecated, use getIdentities instead)
- FPC Cache
  - The block must implement \Magento\Framework\DataObject\IdentityInterface
  - The block must implement the method **getIdentities** that returns the tags of the models used by this block

Layouts (xml)

View and Layout

Layouts (xml)

- Define how a page will be rendered, by specifying the blocks and the templates to render

View and Layout

Layouts (xml)

- Define how a page will be rendered, by specifying the blocks and the templates to render
- Magento loads all layout files named after the handles used by the layout engine

View and Layout

Layouts (xml)

- Define how a page will be rendered, by specifying the blocks and the templates to render
- Magento loads all layout files named after the handles used by the layout engine
- Default handles:
  - Current action, e.g. **catalog_product_view.xml**

View and Layout

Layouts (xml)

- Define how a page will be rendered, by specifying the blocks and the templates to render
- Magento loads all layout files named after the handles used by the layout engine
- Default handles:
  - Current action, e.g. **catalog_product_view.xml**
  - All actions: **default.xml**

View and Layout

Layouts (xml)

- Define how a page will be rendered, by specifying the blocks and the templates to render
- Magento loads all layout files named after the handles used by the layout engine
- Default handles:
  - Current action, e.g. **catalog_product_view.xml**
  - All actions: **default.xml**
  - Specific handles, e.g **catalog_product_view_type_bundle.xml**

View and Layout

Layouts (xml)

- Define how a page will be rendered, by specifying the blocks and the templates to render
- Magento loads all layout files named after the handles used by the layout engine
- Default handles:
  - Current action, e.g. **catalog_product_view.xml**
  - All actions: **default.xml**
  - Specific handles, e.g **catalog_product_view_type_bundle.xml**
- In the folder **./view/[AREA]/layout/** of each module

Layouts (xml)

View and Layout

Layouts (xml)

- Two possible root nodes:
  - **page**: renders a complete HTML page
  - **layout**: renders only a section of HTML

View and Layout

Layouts (xml)

- Two possible root nodes:
  - **page**: renders a complete HTML page
  - **layout**: renders only a section of HTML
- Two possible elements:
  - **block**
  - **container**:
    - contains other blocks and containers
    - renders all its children
    - does not display anything if no children

View and Layout

Layouts / Root node **page**
Can have 4 different child nodes:

- **html**, with "name" and "value" attributes

Layouts / Root node **page**

Can have 4 different child nodes:

- **html**, with "name" and "value" attributes
  - Allows to add HTML attributes to the <html> tag

Layouts / Root node **page**

Can have 4 different child nodes:

- **html**, with "name" and "value" attributes
  - Allows to add HTML attributes to the <html> tag
  - Example: `<html name="class" value="training"/>` results in `<html class="training">` in the DOM

View and Layout

Layouts / Root node **page**
Can have 4 different child nodes:

- **head**

View and Layout

Layouts / Root node **page**
Can have 4 different child nodes:

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

View and Layout

Layouts / Root node **page**

Can have 4 different child nodes:

- **head**
  - sub node **attribute**,
    with "name" and "value" attributes
  - sub node **css**,
    with "src" attribute

View and Layout

Layouts / Root node **page**

Can have 4 different child nodes:

- **head**
  - sub node **attribute**,
    with "name" and "value" attributes
  - sub node **css**,
    with "src" attribute
  - sub node **script**,
    with "src" attribute

Layouts / Root node **page**

Can have 4 different child 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

Layouts / Root node **page**

Can have 4 different child 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

View and Layout

Layouts / Root node **page**

Can have 4 different child 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

View and Layout

Layouts / Root node **page**

Can have 4 different child 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**

View and Layout

Layouts / Root node **page**
Can have 4 different child nodes:

- **body**

View and Layout

Layouts / Root node **page**

Can have 4 different child nodes:

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

View and Layout

Layouts / Root node **page**

Can have 4 different child nodes:

- **body**
  - sub node **attribute**,
    with "name" and "value" attributes
  - sub node **container**,
    with "name", "htmlTag", "htmlClass", "htmlId", "label" attributes

View and Layout

Layouts / Root node **page**

Can have 4 different child 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

Layouts / Root node **page**

Can have 4 different child 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

Layouts / Root node **page**

Can have 4 different child 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

Layouts / Root node **page**

Can have 4 different child 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

View and Layout

Layouts / Root node **page**
Can have 4 different child nodes:

- **update**, with "handle" attribute

View and Layout

Layouts / Root node **page**
Can have 4 different child nodes:

- **update**, with "handle" attribute
  - Allows to use additional layout files, "handle" attribute contains the name of the layout file to include

Layouts / Root node **page**
Can have 4 different child nodes:

- **update**, with "handle" attribute
  - Allows to use additional layout files, "handle" attribute contains the name of the layout file to include
  - Example: Magento_Checkout::checkout_cart_configure.xml includes Magento_Catalog::catalog_product_view.xml

Page **Layouts**

View and Layout

Page **Layouts**

- Defines how the page will be defined globally
- Uses only containers

View and Layout

Page **Layouts**

- Defines how the page will be defined globally
- Uses only containers
- Defined in the module **Magento_Theme**

Frontend Page **Layouts**

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

Backend Page **Layouts**

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

Layout example:
Module **Magento_Customer**, frontend action **forgotpassword**

View and Layout

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

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="Magento_Customer::form/forgotpassword.phtml">
                <container name="form.additional.info" as="form_additional_info"/>
            </block>
        </referenceContainer>
    </body>
</page>
```

Plan

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

Seller Module (see 12-seller-part3)

- In the module **Training/Seller**

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

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

Seller Module (see 12-seller-part3)

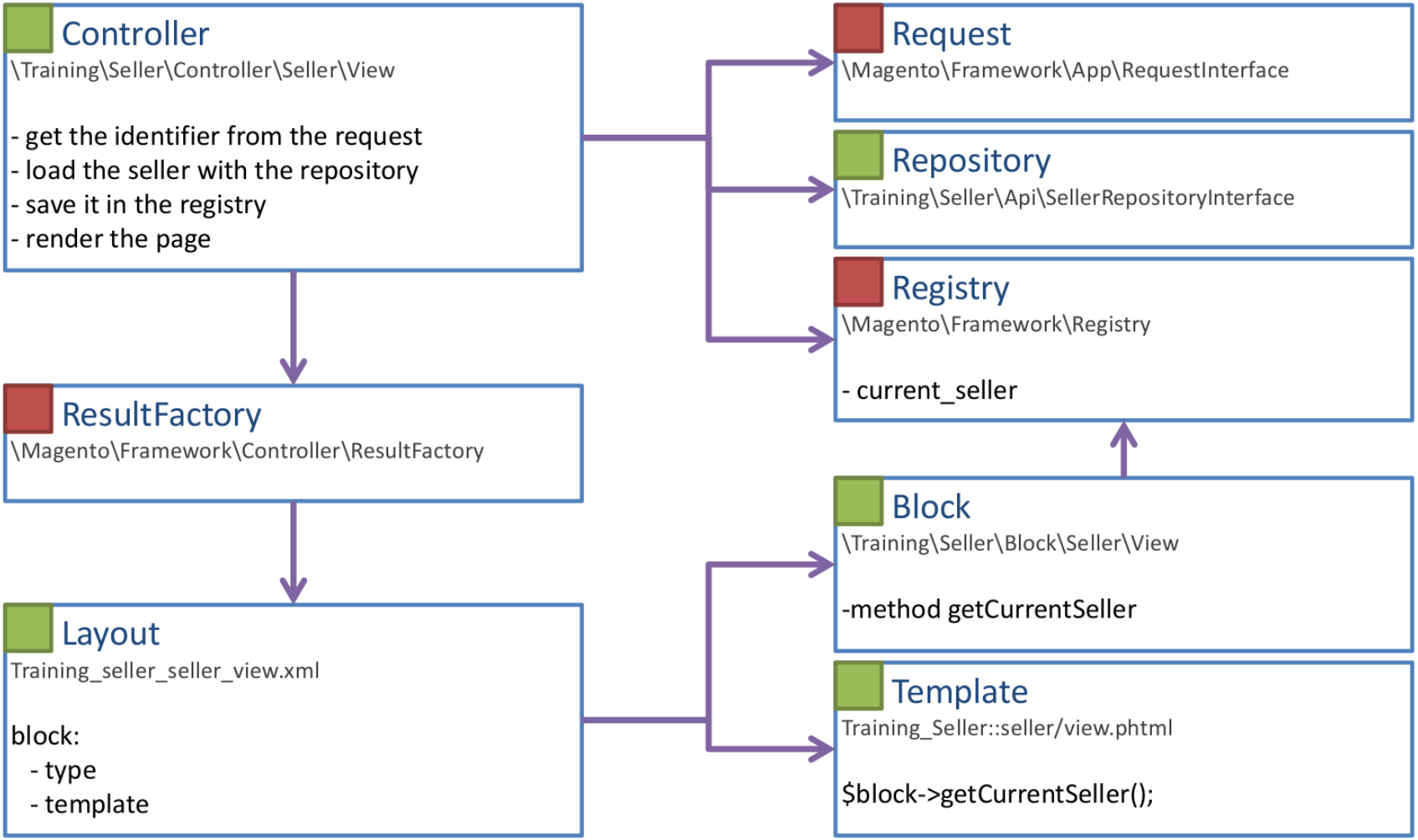
- In the module **Training/Seller**
- Use layouts, templates, and blocks for the frontend actions index and view
- The index action must allow to filter sellers by name, and to choose the sort order (asc/desc), without paging

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 allow to filter sellers by name, and to choose the sort order (asc/desc), without paging
- First, start with the view action

Seller Module (see 12-seller-part3)



File **Helper/Url.php**

- Generic helper to manage the frontend URLs
- getSellersUrl: returns the URL to display the list of the sellers
- getSellerUrl: returns the URL to display one seller, from its identifier

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

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

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

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

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

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

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

- Asks for generic useful tools: Url Helper and Magento Registry
- Provides two shortcuts for the URL methods

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**

- Uses the 1column layout to display the content of the seller
- New block "seller.view" in the "content" container
- Good practice: always add the module prefix for the phtml template files

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

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

- Uses the $block object to access to its public methods
- Good practice: always use the escapeHtml method to add protection
- Good practice: always use the ___ function to be i18n ready
- Good practice: add /* @escapeNotVerified */ for already secured output

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

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

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

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 sellers found
- 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 **view/frontend/layout/training_seller_seller_index.xml**

- Uses the 2columns-left layout to display the filter in the left sidebar
- New block "seller.list" in the "content" container
- New block "seller.list.filter" in the "sidebar.main" container
- These two blocks use the same PHP block class
- Good practice: always add the module prefix for the phtml template files

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

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

- Uses the $block object to access to its public methods
- Good practice: always use the escapeHtml method to add protection
- Good practice: always use the ___ function to be i18n ready
- Good practice: add /* @escapeNotVerified */ for already secured output

Plan

Practice - Seller - Part 4 - Layout Update

Seller Module (see 13-seller-part4)

Practice - Seller - Part 4 - Layout Update

Seller Module (see 13-seller-part4)

- In the previous module **Training/Seller**

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

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 a block that displays a link to the sellers page at the top of the content section, on all pages
  - type: template
  - phtml to use : new header.phtml file
  - parameters: border_color and background_color

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 a block that displays a link to the sellers page at the top of the content section, on all pages
  - type: template
  - phtml to use : new header.phtml file
  - parameters: border_color and background_color
- The header.phtml file:
  - displays a div that uses border_color and background_color
  - contains only a link "[Sellers list]"

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 a block that displays a link to the sellers page at the top of the content section, on all pages
  - type: template
  - phtml to use : new header.phtml file
  - parameters: border_color and background_color
- The header.phtml file:
  - displays a div that uses border_color and background_color
  - contains only a link "[Sellers list]"
- On the Category page, move this new block to the top of the sidebar

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 a block that displays a link to the sellers page at the top of the content section, on all pages
  - type: template
  - phtml to use : new header.phtml file
  - parameters: border_color and background_color
- The header.phtml file:
  - displays a div that uses border_color and background_color
  - contains only a link "[Sellers list]"
- On the Category page, move this new block to the top of the sidebar
- On the Product page, use different border and background colors

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 a block that displays a link to the sellers page at the top of the content section, on all pages
  - type: template
  - phtml to use : new header.phtml file
  - parameters: border_color and background_color
- The header.phtml file:
  - displays a div that uses border_color and background_color
  - contains only a link "[Sellers list]"
- On the Category page, move this new block to the top of the sidebar
- On the Product page, use different border and background colors
- On the Sellers pages, remove this block

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 forget 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 a new argument "background_color"
  - Set a new argument "border_color"

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 get the color values

Practice - Seller - Part 4 - Layout Update

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

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

Practice - Seller - Part 4 - Layout Update

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

- Move the block "training.seller.content.top" to 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"

Practice - Seller - Part 4 - Layout Update

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

- Move the block "training.seller.content.top" to 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 layout to remove the "training.seller.content.top"

Practice - Seller - Part 4 - Layout Update

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

- Move the block "training.seller.content.top" to 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 layout to remove the "training.seller.content.top"

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

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

Plan

Practice - Seller - Part 5 - Admin

Seller Module (see 14-seller-part5)

Practice - Seller - Part 5 - Admin

Seller Module (see 14-seller-part5)

- In the previous module **Training/Seller**

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 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

Practice - Seller - Part 5 - Admin

File **etc/di.xml**

- Needed by the advanced listing UI component
- Adds the new Seller Grid Collection to the DataProvider Collection factory
- Adds the parameters of the Seller Grid Collection constructor

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

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

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

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

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 be used by the js

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 be used by the js

Sub Node **settings**

- The item **buttons** defines the buttons to display on the top of the listing
- The item **spinner** defines the columns to use (spinner)
- The item **deps** defines the dependencies with the data source

Practice - Seller - Part 5 - Admin

File **view/adminhtml/ui_component/training_seller_seller_listing.xml** 3/5
Sub Node **dataSource**

- The item **settings** defines the storage config for the id field, ans the renderer to use to update the grid
- The item **acl** defines the acl resource that can access this datasource
- The item **data provider** defines the data provider to use, with its settings

Practice - Seller - Part 5 - Admin

File **view/adminhtml/ui_component/training_seller_seller_listing.xml** 4/5
Sub Node **listingToolbar**

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

File **view/adminhtml/ui_component/training_seller_seller_listing.xml** 5/5
Sub Node **columns**

- The item **settings** allows to define the inline edit config
- The item **selectionsColumn** allows to define the field to use for mass actions
- The item **column** allows to define the columns of the list
  - 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 (text by default)
  - The item **editor** defines the validator for inline edit
  - The item **sorting** defines the default sort
- 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

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**

- Provides a constructor to ask for useful tools like the Magento registry

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**

- Provides a constructor to ask for useful tools like the Magento registry

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

- Renders the page using the page result factory

Practice - Seller - Part 5 - Admin

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

- Method execute: delete the ids and redirect to the list
- Uses the messageManager property to display a success message

Practice - Seller - Part 5 - Admin

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

- Method execute: delete the ids and redirect to the list
- Uses the messageManager property to display a success message

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

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

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

- Needed by the advanced form UI component
- Defines how to get the data to display in the form
- Must extend 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
  - Overrides the data with the data in session (if they exist)

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 returns the button info
- The public method getObjectId returns the current seller_id, if valid

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 returns the button info
- The public method getObjectId returns the current seller_id, if valid

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

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

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 returns the button info
- The public method getObjectId returns the current seller_id, if valid

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

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

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

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

Practice - Seller - Part 5 - Admin

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

- Displays the "Save" button, to submit the edit form and redirect to the list

Practice - Seller - Part 5 - Admin

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

- Displays the "Save" button, to submit the edit form and redirect to the list

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

- Displays the "Save and Continue" button, to submit the edit form and redirect to the form

Practice - Seller - Part 5 - Admin

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

- Displays the "Save" button, to submit the edit form and redirect to the list

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

- Displays the "Save and Continue" button, to submit the edit form and redirect to the form

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

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

File **view/adminhtml/ui_component/training_seller_seller_form.xml** 1/4

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

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 be used by the JS
- The item **label** defines the name to use
- The item **template** defines the form template to use

File **view/adminhtml/ui_component/training_seller_seller_form.xml** 2/4

Sub Node **argument**

- The item **js_config** defines the data source to be used by the JS
- The item **label** defines the name to use
- The item **template** defines the form template to use

Sub Node **settings**

- The item **buttons** defines the buttons to display on the top of the form
- The item **namespace** defines the namespace of the form
- The item **dataScope** defines the key of the data
- The item **deps** defines the dependencies with the data source

Practice - Seller - Part 5 - Admin

File **view/adminhtml/ui_component/training_seller_seller_form.xml** 3/4
Sub Node **dataSource**

- The item **js_config** defines the JS component to use
- The item **setting** defines the submit url to use
- The item **acl** defines the ACL resource that can access this UI component
- The property **dataProvider/class** defines the data provider to use
- The property **dataProvider/name** defines the name of the data source
- The item **dataProvider/primaryFieldName** defines the name of the db primary key
- The item **dataProvider/requestFieldName** defines the name of the request field for the primary key

File **view/adminhtml/ui_component/training_seller_seller_form.xml** 4/4
Sub Node **fieldset**

- The item **settings** allows to define the label of the fieldset
- The items **field** are the HTML inputs of the form
  - The property **formElement** defines the type of form element
  - The item **visible** defines the visibility
  - The item **dataType** defines the type of data
  - The item **label** defines the label to display
  - 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

Practice - Seller - Part 5 - Admin

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

- The page contains only the new seller form UI component

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**

- Uses the result factory to generate the page

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**

- Uses the result factory to generate the page

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

- Uses the seller repository to load the current seller and save the values
- Uses the dataPersistor to save the values in the session
- Uses the messageManager to save a message in the session
- Uses the result factory to redirect to the index page

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**

- Uses the result factory to generate the page

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

- Uses the seller repository to load the current seller and save the values
- Uses the dataPersistor to save the values in the session
- Uses the messageManager to save a message in the session
- Uses the result 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 result factory to redirect to the index page

Plan

8 Others

- Practice - Seller - Part 6 - Upgrade
- Practice - Seller - Part 7 - Customer Attribute
- Practice - Seller - Part 8 - Product Attribute
- Practice - Seller - Part 9 - Extension Attribute
- Practice - Seller - Part 10 - Console Command
- Practice - Seller - Part 11 - i18n
- Practice - Seller - Part 12 - Cron
- Practice - Seller - Part 13 - Unit Test
- Create a new type of xml config file
- Create a new type of xml config file - Practice

Plan

8 Others

- Practice - Seller - Part 6 - Upgrade
- Practice - Seller - Part 7 - Customer Attribute
- Practice - Seller - Part 8 - Product Attribute
- Practice - Seller - Part 9 - Extension Attribute
- Practice - Seller - Part 10 - Console Command
- Practice - Seller - Part 11 - i18n
- Practice - Seller - Part 12 - Cron
- Practice - Seller - Part 13 - Unit Test
- 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)

Practice - Seller - Part 6 - Upgrade

Seller Module (see 15-seller-part6)

- In the previous module **Training/Seller**

Practice - Seller - Part 6 - Upgrade

Seller Module (see 15-seller-part6)

- In the previous module **Training/Seller**
- Add an optional text field **description** to the seller entity

Practice - Seller - Part 6 - Upgrade

Seller Module (see 15-seller-part6)

- In the previous module **Training/Seller**
- Add an optional text field **description** to the seller entity
- Add it to the edit form, using a WYSIWYG field

Practice - Seller - Part 6 - Upgrade

Seller Module (see 15-seller-part6)

- In the previous module **Training/Seller**
- Add an optional text field **description** to 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

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 $< 1.0.1$

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

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

Practice - Seller - Part 7 - Customer Attribute

Seller Module (see 16-seller-part7)

Practice - Seller - Part 7 - Customer Attribute

Seller Module (see 16-seller-part7)

- In the previous module **Training/Seller**

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 used as a source by the attribute to fetch its values
- Extends Magento\Eav\Model\Entity\Attribute\Source\AbstractSource
- Asks for the Seller Collection Factory in the constructor
- The public method getAllOptions returns the values, and uses a local cache

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
- Use the new Option Seller class as the attribute source
- The new attribute must be added to the adminhtml_customer form
- The EAV config cache must be cleared after each modification

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
- Use the new Option Seller class as the attribute source
- 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

Plan

Practice - Seller - Part 8 - Product Attribute

Seller Module (see 17-seller-part8)

Practice - Seller - Part 8 - Product Attribute

Seller Module (see 17-seller-part8)

- In the previous module **Training/Seller**

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

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"

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

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

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 in the varchar table, using the ArrayBackend class
- The form field will be a multiselect field
- Use the new Option Seller class as the attribute source
- 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

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 in the varchar table, using the ArrayBackend class
- The form field will be a multiselect field
- Use the new Option Seller class as the attribute source
- 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

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**

- Use the search criteria builder classes, and the seller repository
- The public method getProductSellerIds returns the seller ids linked to a product
- The public method getSearchCriteriaOnSellerIds builds a search criteria, filtered on a list of seller ids
- The public method getProductSellers uses 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**

- Use the registry, and the new data helper
- The public method getCurrentProduct returns the current product (saved in the registry)
- The public method getProductSellers uses the data helper to get the list of the sellers linked to the current product

Practice - Seller - Part 8 - Product Attribute

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

- Use the registry, and the new data helper
- The public method getCurrentProduct returns the current product (saved in the registry)
- The public method getProductSellers uses 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

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

Practice - Seller - Part 9 - Extension Attribute

Seller Module (see 18-seller-part9)

Practice - Seller - Part 9 - Extension Attribute

Seller Module (see 18-seller-part9)

- In the previous module **Training/Seller**

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

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** module, for the **website_ids** extension attribute

Practice - Seller - Part 9 - Extension Attribute

File **etc/extension_attributes.xml**

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

Practice - Seller - Part 9 - Extension Attribute

File **etc/extension_attributes.xml**

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

File **etc/di.xml**

- Adds a new ReadHandler on the product interface, to load the sellers linked to a product automatically during product loading

Practice - Seller - Part 9 - Extension Attribute

File **Model/Product/Seller/ReadHandler.php**

- Uses the Data Helper to load the sellers linked to a product
- The public method **execute**:
  - Gets the extension attributes from the product
  - Gets the list of the sellers linked to the current product
  - Adds the list to the extension attributes
  - Saves them to the product

Practice - Seller - Part 9 - Extension Attribute

File **Model/Product/Seller/ReadHandler.php**

- Uses the Data Helper to load the sellers linked to a product
- The public method **execute**:
  - Gets the extension attributes from the product
  - Gets the list of the sellers linked to the current product
  - Adds the list to the extension attributes
  - Saves them to the product

Don't forget to modify the block and the product sellers tabs to use this new extension attribute

Plan

8 Others

- Practice - Seller - Part 6 - Upgrade
- Practice - Seller - Part 7 - Customer Attribute
- Practice - Seller - Part 8 - Product Attribute
- Practice - Seller - Part 9 - Extension Attribute
- Practice - Seller - Part 10 - Console Command
- Practice - Seller - Part 11 - i18n
- Practice - Seller - Part 12 - Cron
- Practice - Seller - Part 13 - Unit Test
- Create a new type of xml config file
- Create a new type of xml config file - Practice

Practice - Seller - Part 10 - Console Command

Seller Module (see 19-seller-part10)

Practice - Seller - Part 10 - Console Command

Seller Module (see 19-seller-part10)

- In the previous module **Training/Seller**

Practice - Seller - Part 10 - Console Command

Seller Module (see 19-seller-part10)

- In the previous module **Training/Seller**
- Create a GetCommand that receives a seller ID as an argument and displays the name of this seller on the terminal output

Practice - Seller - Part 10 - Console Command

Seller Module (see 19-seller-part10)

- In the previous module **Training/Seller**
- Create a GetCommand that receives a seller ID as an argument and displays the name of this seller on the terminal output
- Help: look for the CronCommand in the Magento cron module

Practice - Seller - Part 10 - Console Command

File **etc/di.xml**

- Declare the GetCommand in the CommandList entry

Practice - Seller - Part 10 - Console Command

File **etc/di.xml**

- Declare the GetCommand in the CommandList entry

File **Console/Command/GetCommand.php**

- Inject sellerRepository in constructor
- Declare the definition of the command in configure method
- Fetch the seller and displays its name in the execute method:
  - Get the id parameter from $input
  - Write on standard output with $output

Plan

Practice - Seller - Part 11 - i18n

- It is possible to add translations, using CSV files
- The CSV files must be stored in the **i18n** folder of each module
- In the PHTML template files, the function **__('...')** can be used to translate text

Practice - Seller - Part 11 - i18n

- It is possible to add translations, using CSV files
- The CSV files must be stored in the **i18n** folder of each module
- In the PHTML template files, the function **___('...')** can be used to translate text
- The CSV files must:
  - use the language code as filename (en_US, fr_FR, ...)
  - use the **,** char as column separator
  - use the **"** char as value protector (use """ to escape a double quote)
  - use the **n** char as line separator
  - be encoded in **UTF8**

Practice - Seller - Part 11 - i18n

- It is possible to add translations, using CSV files
- The CSV files must be stored in the **i18n** folder of each module
- In the PHTML template files, the function **___('...')** can be used to translate text
- The CSV files must:
  - use the language code as filename (en_US, fr_FR, ...)
  - use the **,** char as column separator
  - use the **"** char as value protector
    (use """ to escape a double quote)
  - use the **n** char as line separator
  - be encoded in **UTF8**
- You can use %1, %2, ... to manage variables

Practice - Seller - Part 11 - i18n

Seller Module (see 20-seller-part11)

Practice - Seller - Part 11 - i18n

Seller Module (see 20-seller-part11)

- In the previous module **Training/Seller**
- Add i18n file for English for all the phrases of the module
- Test it!

Practice - Seller - Part 11 - i18n

Seller Module (see 20-seller-part11)

- In the previous module **Training/Seller**
- Add i18n file for English for all the phrases of the module
- Test it!
- Configure your Magento to use **French (France)** as the locale for the Default Store View
- Add i18n file for French for all the phrases of by the module
- Test it!

Plan

Practice - Seller - Part 12 - Cron

- It is possible to add Crons managed by Magento
- The cron must be injectable
- The cron must be declared in **./etc/crontab.xml**

Practice - Seller - Part 12 - Cron

- It is possible to add Crons managed by Magento
- The cron must be injectable
- The cron must be declared in **./etc/crontab.xml**
- Good practices:
  - Create the cron class in the **./Cron/** folder
  - Call the main method **execute**
  - Add a command to launch the cron manually if needed

Practice - Seller - Part 12 - Cron

- It is possible to add Crons managed by Magento
- The cron must be injectable
- The cron must be declared in **./etc/crontab.xml**
- Good practices:
  - Create the cron class in the **./Cron/** folder
  - Call the main method **execute**
  - Add a command to launch the cron manually if needed
- Look at the **cron_schedule** table to see all the scheduled crons

Practice - Seller - Part 12 - Cron

Seller Module (see 21-seller-part12)

Practice - Seller - Part 12 - Cron

Seller Module (see 21-seller-part12)

- In the previous module **Training/Seller**
- Create a Cron that logs the total number of sellers every 5 minutes
- Follow all the good pratices

Plan

Practice - Seller - Part 13 - Unit Test

- It is possible to add Unit Test
- It uses **PhpUnit** version **6.2**
- The PHP files must be stored in the **Test/Unit** folder of each module

Practice - Seller - Part 13 - Unit Test

The file **phpunit.xml** must be added in the root folder

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
    backupGlobals attribute is very important
    or you will have "Serialization of 'Closure' is not allowed" error
-->
<phpunit xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:noNamespaceSchemaLocation="http://schema.phpunit.de/4.1/phpunit.xsd"
     colors="true"
     bootstrap="./app/bootstrap.php"
     backupGlobals="false"
>
    <testsuite name="Training Unit Tests">
        <directory suffix="Test.php">app/code/Training/*/Test/Unit</directory>
    </testsuite>
    <filter>
        <whitelist addUncoveredFilesFromWhiteList="true">
            <directory suffix=".php">app/code/Training/*</directory>
            <exclude>
                <directory>app/code/Training/*/Test</directory>
                <directory suffix="registration.php">app/code/Training</directory>
            </exclude>
        </whitelist>
    </filter>
    <logging>
        <log type="coverage-clover" target="build/logs/clover.xml"/>
        <log type="coverage-html"   target="build/logs/coverage"
            charset="UTF-8" yui="true" highlight="true" />
    </logging>
</phpunit>
```

Add the **/build/** folder in your **.gitignore** file.

Practice - Seller - Part 13 - Unit Test

PhpUnit can be executed with the following command:

```
./bin/phpunit
```

Practice - Seller - Part 13 - Unit Test

PhpUnit can be executed with the following command:

```
./bin/phpunit
```

It will display the following output:

```
No tests executed!
```

Practice - Seller - Part 13 - Unit Test

PhpUnit can be executed with the following command:

```
./bin/phpunit
```

It will display the following output:

```
No tests executed!
```

You can visualize the coverage report with the following command:

```
firefox build/logs/coverage/index.html
```

First Unit Test

./Training/Seller/Test/Unit/Model/SellerTest.php

```php
<?php
namespace Training\Seller\Test\Unit\Model;

use \PHPUnit\Framework\TestCase;
use \Magento\Framework\TestFramework\Unit\Helper\ObjectManager;
use \Training\Seller\Model\Seller as ModelSeller;

class SellerTest extends TestCase
{
    protected $objectManager;

    protected function setUp()
    {
        $this->objectManager   = new ObjectManager($this);
    }

    /**
     * Test the name methods
     */
    public function testName()
    {
        $value = 'test';
        /** @var ModelSeller $model */
        $model = $this->objectManager->getObject(ModelSeller::class);
        $model->setName($value);
        $this->assertEquals($value, $model->getName());
    }
}
```

Practice - Seller - Part 13 - Unit Test

Seller Module (see 22-seller-part13)

Practice - Seller - Part 13 - Unit Test

Seller Module (see 22-seller-part13)

- In the previous module **Training/Seller**
- Create Unit Test for all the methods of the Seller model
- Launch the Unit Test

Practice - Seller - Part 13 - Unit Test

Seller Module (see 22-seller-part13)

- In the previous module **Training/Seller**
- Create Unit Test for all the methods of the Seller model
- Launch the Unit Test
- Create Unit Test for all the methods of the Url Helper
- Launch the Unit Test

Plan

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

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 provides the path to the XSD schema files

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 provides the path to the XSD schema files
- Converter: PHP class that converts XML to PHP array

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 provides the path to the XSD schema files
- Converter: PHP class that converts XML to PHP array
- Schema: XSD schema file

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 provides the path to the XSD schema files
- Converter: PHP class that converts XML to PHP array
- Schema: XSD schema file
- Interface: PHP Interface that specifies how the data can be accessed from another module

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 provides the path to the XSD schema files
- Converter: PHP class that converts 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, with getters for config values

Plan

8 Others

- Practice - Seller - Part 6 - Upgrade
- Practice - Seller - Part 7 - Customer Attribute
- Practice - Seller - Part 8 - Product Attribute
- Practice - Seller - Part 9 - Extension Attribute
- Practice - Seller - Part 10 - Console Command
- Practice - Seller - Part 11 - i18n
- Practice - Seller - Part 12 - Cron
- Practice - Seller - Part 13 - Unit Test
- Create a new type of xml config file
- Create a new type of xml config file - Practice

Create a new type of xml config file - Practice

Practice (see 23-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 type of xml config file - Practice

Practice (see 23-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 config file **etc/shops.xml** that uses it

Create a new type of xml config file - Practice

Practice (see 23-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 config file **etc/shops.xml** that uses it
- Create all the required PHP files to use this new XML config file

Create a new type of xml config file - Practice

Practice (see 23-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 config file **etc/shops.xml** that uses it
- Create all the required PHP files to use this new XML config file
- Create a basic frontend action to test them (without layout/block)

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

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 to a PHP Array

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 to a PHP Array

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

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

Create a new type of xml config file - Practice

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

- Defines how the config values will be accessed by others modules

Create a new type of xml config file - Practice

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

- Defines how the config values will be accessed by others modules

File **Config/Shop.php**

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

Create a new type of xml config file - Practice

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

- Defines how the config values will be accessed by others modules

File **Config/Shop.php**

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

File **etc/di.xml**

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

Create a new type of xml config file - Practice

Create action **Index/Index**

- Display the list of all the shops
- Display a specific shop

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

Plan