# **Magento2 Quickies**

## Magento2 – Registry

Magento Registry lưu trữ dữ liệu vào bộ nhớ cụ thể cho yêu cầu đó (thay vì người dùng hoặc bất kỳ thứ gì khác) và chỉ tồn tại trong thời gian yêu cầu đó. Nguyên tắc rất đơn giản, lớp Mage được khởi tạo như một đối tượng đơn cho mọi yêu cầu và đối tượng Mage tức thời vẫn còn trong bộ nhớ, và có thể truy cập được trong tất cả các lớp (và các khuôn mẫu) cho đến khi yêu cầu hoàn thành và phản hồi được gửi đi.

Khi đối tượng là một singleton, bất cứ khi nào bạn truy cập nó, bạn sẽ có cùng một đối tượng. Tất cả những gì đang xảy ra là bạn đang lưu trữ các giá trị cho đối tượng này, vì vậy khi một lớp lưu trữ một giá trị và một lớp khác truy cập nó, cả hai đều làm việc trên cùng một đối tượng và lớp thứ hai có thể lấy giá trị của lớp đầu tiên.

Việc thực hiện hơi khác một chút. Ở đây chúng ta sử dụng class Magento \ Framework \ Registry, và chúng ta có thể tiêm nó vào bất kỳ constructor nào như sau:

namespace Excellence\Hello\Controller\Index;

class Index extends \Magento\Framework\App\Action\Action

{

    public function \_\_construct(

        \Magento\Framework\App\Action\Context $context,

        \Magento\Framework\Registry $registry)

    {

        $this->registry = $registry;

        return parent::\_\_construct($context);

    }

    public function execute()

    {

        #some code here

        $this->registry->register('test\_var', 'this is a test!');

        #some code here

    }

}

Để truy cập biến, chúng tôi sẽ sử dụng đoạn mã sau:

$ this-> registry-> registry ('test\_var');

Về cơ bản, bạn có thể thiết lập một biến trong registry trong bộ điều khiển, chặn và truy cập nó ở bất cứ đâu như helper, black, model etc nhưng chỉ trong cùng một yêu cầu.

## Magento2 - Chuyển hướng URL và Trình tạo URL

Có 2 cách chuyển hướng

* Chuyển về trang trước
* Chuyển hướng đến trang khác

namespace Excellence\Hello\Controller\Index;

use Magento\Framework\Controller\ResultFactory;

class Actionname name extends \Magento\Framework\App\Action\Action

{

      public function execute()

      {

**// tạo ra resultFactory**

           $resultRedirect = $this->resultFactory->create(ResultFactory::TYPE\_REDIRECT);

**// sử dụng resultFactory để chuyển hướng đến trang khác**

**// tham số** $ this -> \_ redirect-> getRefererUrl ()  dùng để chuyển hướng về trang trước

           $resultRedirect->setUrl($this->\_redirect->getRefererUrl());

           return $resultRedirect;

      }

}

Ví dụ:

return $ this-> resultRedirectFactory-> create () -> setPath(excellence

/index/add/', ['\_current' => true]);

 ['\_current' => true] thì các đối tượng được truyền qua chuỗi truy vấn sẽ vẫn nằm trong URL được chuyển hướng nhưng nếu chúng ta không sử dụng ['\_current' => true] hoặc sử dụng ['\_current' => false] thì đối tượng sẽ không được chuyển trong URL được chuyển hướng.

* Giả sử chúng ta muốn chuyển hướng từ trang này sang trang khác bằng cách sử dụng thẻ neo hoặc tương tự, thì chúng ta cần đặt thuộc tính href cho URL mà nó cần chuyển hướng trang hiện tại. Ở đây chúng ta không nên sử dụng mã cứng như http://127.0.0.1/magento2/index.php/excellence/index/add/. Magento 2 đi kèm với khái niệm về trình tạo URL. nó xây dựng URL thực tế của hành động điều khiển sẽ được sử dụng. Chúng ta phải sử dụng:
* $this->\_urlBuilder->getUrl("excellence/index/add/")

## Magento 2 – Request Response JSON

Trong nhiều trường hợp, chúng ta cần đọc dữ liệu được truyền bằng phương thức GET hoặc POST chẳng hạn như dữ liệu được truyền từ biểu mẫu. Để xử lý dữ liệu này, trước tiên chúng ta cần đọc dữ liệu.

Để đọc dữ liệu được truyền qua phương thức get, chúng ta chỉ cần sử dụng hai phương thức sau:

**$this->getRequest()->getParams()**

điều này sẽ đọc tất cả dữ liệu nhận được nhưng để đọc bất kỳ dữ liệu cụ thể nào nên sử dụng

**$this->getRequest()->getParam('data');**

Để đọc dữ liệu được truyền qua phương thức POST, chúng tôi sử dụng phương thức sau:

**$this->request->getPost()**

điều này sẽ đọc tất cả dữ liệu được truyền qua bài đăng. Nhưng nếu chúng ta muốn đọc dữ liệu cụ thể thì chúng ta sẽ sử dụng

**$this->getRequest()->getPost('data');**

Nhưng, nếu bạn đi qua các tập tin lõi magento2, thì bạn sẽ tìm thấy phương thức getPostvalue () thay vì getPost (). Nó được định nghĩa trong lib \ internal \ Magento \ Framework \ HTTP \ PhpEnvironment \ Request.php:

public function getPostValue($name = null, $default = null)

{

    $post = $this->getPost($name, $default);

    if ($post instanceof ParametersInterface) {

        return $post->toArray();

    }

    return $post;

}

Bạn có thể truy cập dữ liệu bài đăng bằng cách sử dụng:

$post = $this->getRequest()->getPostValue();

Mặc dù cả hai phương thức, getPost () & getPostValue () có chức năng gần như tương tự nhưng có sự khác biệt nhỏ trong định dạng mà chúng trả về dữ liệu. Trong Magento2, nó được cho là thực hành tốt để sử dụng getPostValue ().

Ví dụ sau sẽ giúp bạn hiểu rõ hơn:

Khi chúng ta sử dụng getPost (), nó trả về một thể hiện như Zend \ Stdlib \ Parameters Object ([lưu trữ: ArrayObject: private] => Array ([dữ liệu] => Array ([title] => abc [status] => 1 [submit] => Lưu)))

Nhưng khi chúng ta sử dụng getPostValue () thì chúng ta sẽ có một mảng được cấu trúc tốt như: Array ([data] => Array ([title] => abc [status] => 1 [submit] => Save)) dễ thao tác . Đây là lý do, getPostValue được ưa thích hơn phương thức getPost ().

sau đó nó nhận giá trị getPost từ nhà cung cấp \ zendframework \ zend-http \ src \ Request.php:

public function getPost($name = null, $default = null)

{

    if ($this->postParams === null) {

        $this->postParams = new Parameters();

    }

    if ($name === null) {

        return $this->postParams;

    }

    return $this->postParams->get($name, $default);

}

Các phương thức trên sẽ hoạt động nếu bạn đang thử điều này từ một bộ điều khiển mở rộng Magento \ Framework \ App \ Action \ Action, bạn có thể nhận được yêu cầu. Trong các trường hợp khác, bạn cần tiêm yêu cầu trong hàm tạo như sau:

class ClassName

{

     protected $request;

     public function \_\_construct(

     \Magento\Framework\App\Request\Http $request,

     ....//rest of parameters here

     ) {

          $this->request = $request;

          ...//rest of constructor here

     }

     public function getPost()

     {

          return $this->request->getPost();

     }

}

**Sending Custom Header/Response from Controller**:

từ một phương thức thực hiện điều khiển, để thao tác yêu cầu gửi một trang tiêu đề và trang lỗi tùy chỉnh.

Trước hết, ở action controller \ Magento \ Framework \ App \ ActionInterface :: execute (), hành động trả về một thể hiện của \ Magento \ Framework \ Controller \ ResultInterface (\ Magento \ Framework \ App \ ResponseInterface cũng là được hỗ trợ, nhưng là di sản và sẽ bị loại bỏ trong các bản phát hành tương lai của M2, khi tất cả các tập quán cốt lõi được tái cấu trúc).

Vì vậy, hãy chọn các triển khai có sẵn của \ Magento \ Framework \ Controller \ ResultInterface. Phù hợp nhất cho REST API tùy chỉnh (giả sử nó hoạt động với JSON) có vẻ là \ Magento \ Framework \ Controller \ Result \ Json. Tuy nhiên, nếu bạn cần một cái gì đó thậm chí nhiều hơn tùy chỉnh, xem xét \ Magento \ Framework \ Controller \ Result \ Raw.

Ví dụ

namespace VendorName\ModuleName\Controller;

/\*\*

 \* Demo of authorization error for custom REST API

 \*/

class RestAuthorizationDemo extends \Magento\Framework\App\Action\Action

{

    /\*\* @var \Magento\Framework\Controller\Result\JsonFactory \*/

    protected $jsonResultFactory;

    public function \_\_construct(

        \Magento\Framework\App\Action\Context $context,

        \Magento\Framework\Controller\Result\JsonFactory $jsonResultFactory

    ) {

        parent::\_\_construct($context);

        $this->jsonResultFactory = $jsonResultFactory;

    }

    public function execute()

    {

        /\*\* @var \Magento\Framework\Controller\Result\Json $result \*/

        $result = $this->jsonResultFactory->create();

        /\*\* You may introduce your own constants for this custom REST API \*/

        $result->setHttpResponseCode(\Magento\Framework\Webapi\Exception::HTTP\_FORBIDDEN);

        $result->setData(['error\_message' => \_\_('What are you doing here?')]);

        return $result;

    }

}

## Magento2 – Form Validation

Có nhiều cách

* Thường

<form class="form" id="**custom-form**" method="post" autocomplete="off">

<fieldset class="fieldset">

       <legend class="legend"><span><?php echo \_\_('Personal Information') ?></span></legend><br>

        <div class="field required">

            <label for="email\_address" class="label"><span><?php echo \_\_('Email') ?></span></label>

            <div class="control">

                <input type="email" name="email" id="email\_address" value="" title="<?php echo \_\_('Email') ?>" class="input-text" data-validate="{required:true, 'validate-email':true}">

            </div>

        </div>

   </fieldset>

   <div class="actions-toolbar">

        <div class="primary">

            <button type="submit" class="action submit primary" title="<?php  echo \_\_('Submit') ?>"><span><?php echo \_\_('Submit') ?></span></button>

        </div>

    </div>

</form>

Để xác định validate bằng js có 2 cách

* Viết trực tiếp js trên file đó

<form data-mage-init='**{"validation": {}}'** class="form" id="custom-form" method="post" autocomplete="off">

* Hoặc viết js riêng

<script type="text/javascript">

require([

    'jquery',

    'mage/mage'

], function($){

   var dataForm = $('#custom-form');

   dataForm.mage('validation', {});

});

</script>

Thường để lib/web/mage/validation.js

ở đây bạn có thể thấy các loại xác nhận khác nhau có thể được sử dụng như 'validate-cc-cvn', 'validate-length', 'validate-one-required', v.v ...

cũng là widget “$ .widget (“ mage.validation ”)” được định nghĩa ở đó bạn có thể thấy các tùy chọn

## Magento2 – Events – Module Development

**Setup event**

Để thêm event ta cần thiết lập :

**Event frontend**

Excellence/Hello/etc/frontend/events.xml

<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="excellence\_hello\_test\_observer" instance="Excellence\Hello\Observer\Predispatch" />  
</event>  
</config>

* name (bắt buộc) - Tên của người quan sát cho định nghĩa sự kiện.
* instance (bắt buộc) - Tên class quan sát.
* disabled- Xác định xem người quan sát này có đang hoạt động hay không. Giá trị mặc định là sai.
* shared- Xác định [lối sống](https://devdocs.magento.com/guides/v2.2/extension-dev-guide/build/di-xml-file.html#object-lifestyle-configuration) của lớp học. Mặc định là sai.

Tiếp theo tạo class quan sat

Excellence\Hello\Observer\Predispatch.php

<?php

 namespace Excellence\Hello\Observer;

use \Psr\Log\LoggerInterface;

use Magento\Framework\Event\Observer;

use Magento\Framework\Event\ObserverInterface;

class Predispatch implements ObserverInterface

{

    protected $logger;

    public function \_\_construct(LoggerInterface $logger)

    {

        $this->logger = $logger;

    }

    public function execute(Observer $observer)

    {

        $this->logger->warn('Observer Works');

        //exit; un commet this to make sure event works

    }

}

Vì chúng ta đã sử dụng sự kiện 'controller\_action\_predispatch', nên người quan sát này nên làm việc trên mọi trang. Nếu bạn mở bất kỳ trang giao diện nào, bạn sẽ thấy ‘Observer Works' trong thư mục **var / log / system.log.**

Hãy xem một ví dụ khác về việc sử dụng các sự kiện phức tạp hơn một chút.

Để tìm sự kiện được phát ra thông qua Magento, bạn cần mở các tệp lõi và tìm sự kiện. Hãy viết mã để đăng nhập tên khách hàng khi anh ta đăng nhập.   
Nhìn vào mô hình 'Magento \ Customer \ Model \ Customer', chúng ta tìm thấy sự kiện liên quan

$this->\_eventManager->dispatch(

            'customer\_customer\_authenticated',

            ['model' => $this, 'password' => $password]

        );

Hãy viết người quan sát cho sự kiện này trong tệp events.xml của chúng ta

<event name="customer\_customer\_authenticated">

        <observer name="excellence\_hello\_customer\_customer\_authenticated" instance="Excellence\Hello\Observer\Customer\Authenticated" />

    </event>

Tạo class Excellence\Hello\Observer\Customer\Authenticated.php

<?php

namespace Excellence\Hello\Observer\Customer;

use \Psr\Log\LoggerInterface;

use Magento\Framework\Event\Observer;

use Magento\Framework\Event\ObserverInterface;

class Authenticated implements ObserverInterface

{

    protected $logger;

    public function \_\_construct(LoggerInterface $logger)

    {

        $this->logger = $logger;

    }

    public function execute(Observer $observer)

    {

        $customer = $observer->getModel();

        //print\_r($customer->getData());exit;

        $this->logger->warn('Customer Logged IN:' . $customer->getFirstname());

    }

}

**Event backend**

Excellence/Hello/etc/adminhtml/events.xml