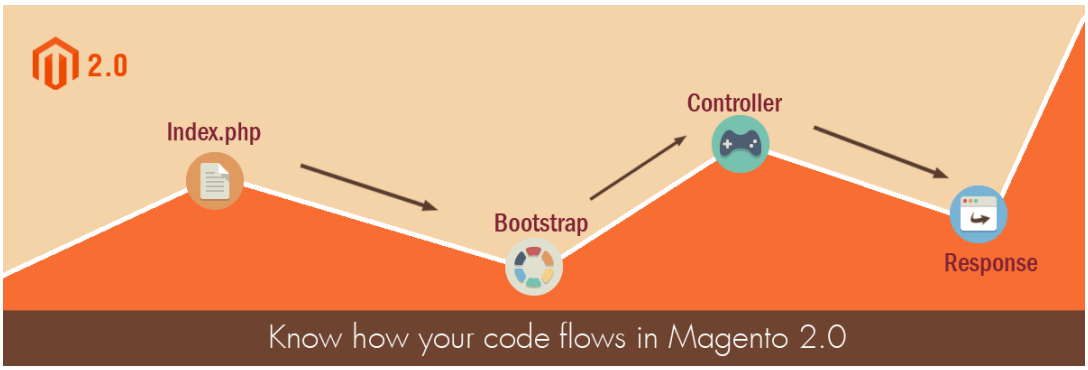
Magento 2.0 request Flow



Magento 2.0 request Flow : This blog illustrates provides a full description of the Magento 2.0 code flow from index.php to .phtml file which are responsible for rendering the html. In previous articles, we have gone through simple Hello World module creation for Magento 2.0. If curious about how the output is generated or wondering what's the code flow, then keep reading the blog till the end.

Here, will cover the main aspects of Magento's request flow.

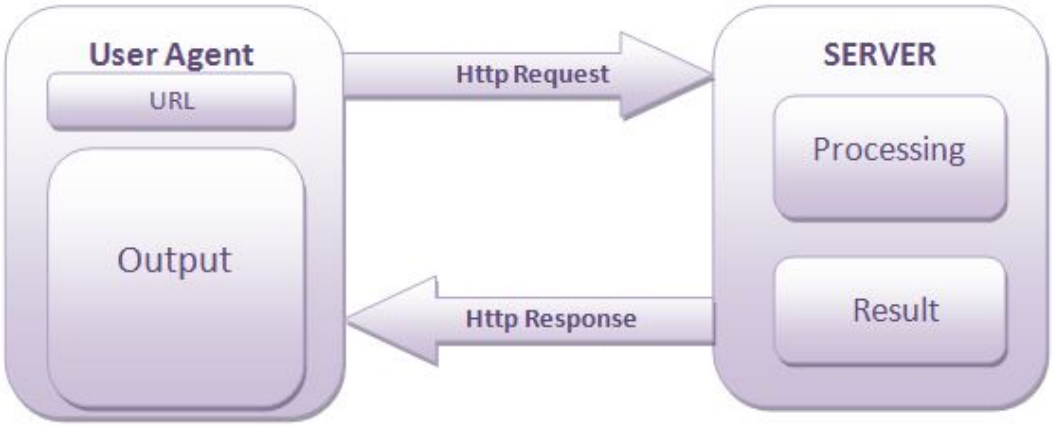


Figure 1 – Request flow block diagram

A quick view on how a typical web server works:

User Agent (Web browser):

- Sends HTTP Request
- Receives HTTP Response and generates HTML Output

Server:

- Receives HTTP Request from User Agent, prepares information related to request
- Returns information as Http Response to User Agent

How Magento fetches output (broad view) :

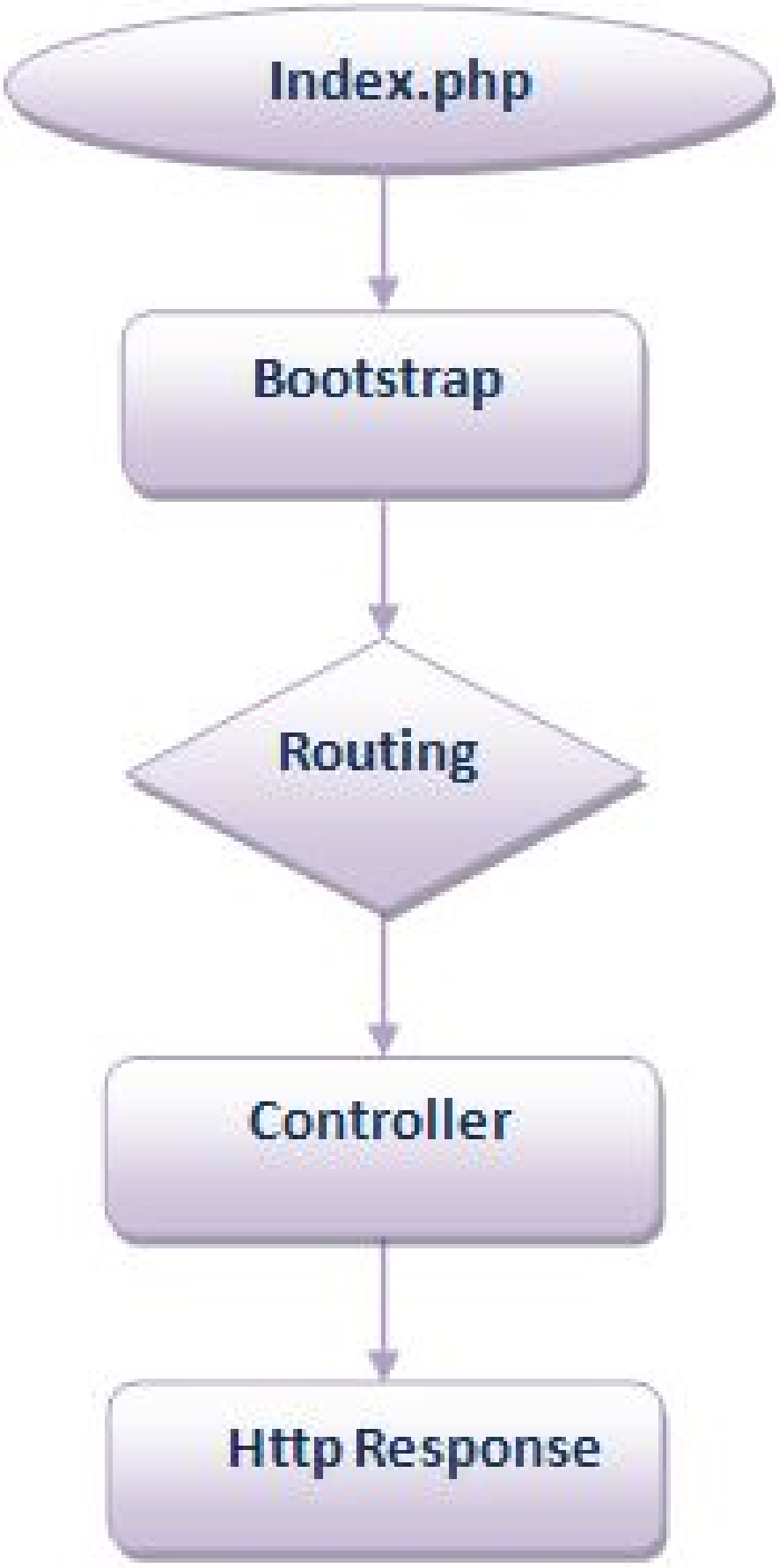


Figure 2 – Request flow inner diagram

Magento Request Url format :

baseurl/module front name/controller/action/parameter/value

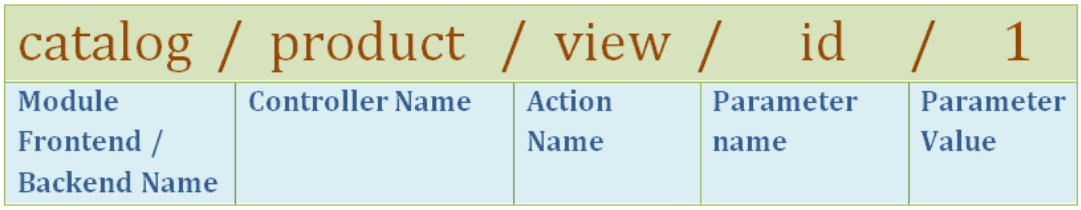


Figure 3 – URL breakthrough

index.php is the initial file where HTTP request to server hits, index.php further initiates Magento environment

```
try {
    require __DIR__ . '/app/bootstrap.php';
} catch (\Exception $e) {
    echo <<<HTML
<div style="font:12px/1.35em arial, helvetica, sans-serif;">
    <div style="margin:0 0 25px 0; border-bottom:1px solid #ccc;">
        <h3 style="margin:0;font-size:1.7em;font-weight:normal;text-transform:none;tex
        Autoload error</h3>
    </div>
    <p>{$e->getMessage()}</p>
</div>
HTML;
    exit(1);
}
$bootstrap = \Magento\Framework\App\Bootstrap::create(BP, $_SERVER);
/** @var \Magento\Framework\App\Http $app */
$app = $bootstrap->createApplication('Magento\Framework\App\Http');
$bootstrap->run($app);
```

let's see how it goes inside index.php line by line

1. Include Bootstrap

- Line number 22 inside index.php includes bootstrap.php
- Bootstrap provides complete environment for execution of Magento application .
- Set error reporting config to **E_ALL**

```
/**
 * Environment initialization
 */
error_reporting(E_ALL);#ini_set('display_errors', 1);
```

- Check **PHP Version**

```
/* PHP version validation */
if (version_compare(phpversion(), '5.5.0', '<') === true) {
```

```
    if (PHP_SAPI == 'cli') {
        echo 'Magento supports PHP 5.5.0 or later. ' .
            'Please read https://devdocs.magento.com/guides/v1.0/install-gde/system-re
    } else {
        echo <<<HTML
<div style="font:12px/1.35em arial, helvetica, sans-serif;">
    <p>Magento supports PHP 5.5.0 or later. Please read
    <a target="_blank" href="https://devdocs.magento.com/guides/v1.0/install-gde/syste
    Magento System Requirements</a>.
</div>
HTML;
    }
    exit(1);}
```

As clearly visible, function **version_compare** checks for php version. If version found is less than 5.5.0 then it is going to throw error message and stop further execution.

- Define **base path** of root directory

```
/**
 * Shortcut constant for the root directory
 */define('BP', dirname(__DIR__));
```

- Initialize **autoloader**

```
$vendorDir = require BP . '/app/etc/vendor_path.php';
$vendorAutoload = BP . "/{$vendorDir}/autoload.php";
/* 'composer install' validation */
if (file_exists($vendorAutoload)) {
    $composerAutoloader = include $vendorAutoload;
} else {
    throw new \Exception(
        'Vendor autoload is not found. Please run \'composer install\' under applicati
    );
}
AutoloaderRegistry::registerAutoloader(new ClassLoaderWrapper($composerAutoloader));
// Sets default autoload mappings, may be overridden in Bootstrap::create
\Magento\Framework\App\Bootstrap::populateAutoloader(BP, []);
```

- **line 16 :** includes vendor file located at **app/etc/vendor_path.php** which in turn returns 'vendor'.
- **line 17 :** includes autoload file (**vendor/autoload.php**) created by composer
- **Line 21 :** includes autoload file which in turn returns object of **classLoader** file located at " **vendor/composer/ClassLoader.php**"
- **Line 28 :** registers autoloader
- **Line 31 :** populates **Autoloader** and maps a namespace prefix to directories for searching the corresponding class. Populate autoloader creates two arrays one for each parser i.e prefixLengths and prefixDirs

  Lets see how autoloader is loading files:

  Lets take an example for Psr4:

  Populate autoloader will create two arrays for "Psr4" i.e prefixLengthsPsr4 and prefixDirsPsr4 as shown below .

```

```

```
prefixLengthsPsr4=Array(
                                        )
                                                                         [Magento\Framewor
                                        )
                                        )
                       )
prefixDirsPsr4=Array(
                                            (
                                            )
                                            (
                                            )
                                            (
                                            )
                                            (
                                            )
                                            (
                                            )
                                            (
                                            )
                                            (
                                            )
                                            (
                                            )                    )
```

So how the Populate autoloader is finally set in action?

for example, if Magento requires instance of "**Magento\Framework\Autoload\AutoloaderRegistry**", initially it is going to take first letter of the class **Magento\Framework\Autoload\AutoloaderRegistry.** The first letter is 'M' so it will check if prefixLengthsPsr4['M'] is set, in case it is not found, it will throw exception

and terminate execution. Here as it's already set then it is going to search through each entry.

Array(
[Monolog\] = 8,
[Magento\Setup\] = 14,
[Magento\Framework\] = 18,
[Magento\] = 8
)

NOTE : Don't mind the directory separator if working in WAMP environment.

1. initially, it will search for required directory in array prefixDirsPsr4[Monolog\]]
prefixDirsPsr4 [Monolog\] = Array(
[0] = C:\xampp\htdocs\dev\m2\vendor/monolog/monolog/src/Monolog
)
it will search for file '**AutoloaderRegistry.php**' in directory "**C:\xampp\htdocs\dev\m2\vendor/monolog/monolog/src/Monolog**", if not found it will continue searching in the next key.
2. similarly in prefixDirsPsr4[Magento\Setup\], if is not found searches in next key.
3. searches in prefixDirsPsr4[Magento\Framework\]
prefixDirsPsr4 [Magento\Framework\] = Array(
[0] = C:\xampp\htdocs\dev\m2/lib/internal/Magento/Framework
)
if file is found in the above directory, same is returned i.e.
"**C:\xampp\htdocs\dev\m2/lib/internal/Magento/Framework\Autoload\AutoloaderRegistry.php**"

- Enables profiler if environment variable "**MAGE_PROFILER**" is set

```
if (!empty($_SERVER['MAGE_PROFILER'])) {
    \Magento\Framework\Profiler::applyConfig($_SERVER['MAGE_PROFILER'], BP, !empty($_
}
```

- Sets default time zone to UTC

```
if (ini_get('date.timezone') == '') {
    date_default_timezone_set('UTC');}
```

2. Create HTTP App

```
/** @var \Magento\Framework\App\Http $app */
$app = $bootstrap->createApplication('Magento\Framework\App\Http');
```

- **Function createApplication()**

```
    /**
     * Factory method for creating application instances
     *
```

```
     * @param string $type
     * @param array $arguments
     * @return \Magento\Framework\AppInterface
     * @throws \InvalidArgumentException
     */
    public function createApplication($type, $arguments = [])
    {
        try {
            $this->initObjectManager();
            $application = $this->objectManager->create($type, $arguments);
            if (!($application instanceof AppInterface)) {
                throw new \InvalidArgumentException("The provided class doesn't implem
            }
            return $application;
        } catch (\Exception $e) {
            $this->terminate($e);
        }    }
```

- **Line 219:** Initialize Object Manager
- As clear from its name that this function is initializing object manager and setting its instance to bootstrap variable "**objectManager**" for future use.
- Further it retrieves Main configuration from **baseDirectory/app/etc/config.php** by calling function below

```
    /**
                                                 *          Loads          the
     *
                                                                *              @


     */
                                public        function        load($configFile        =
    {

this->dirList->getPath(DirectoryList::CONFIG)$configFile;

this->dirList->getPath(DirectoryList::CONFIG)$this->file;
            }

            return $result ?: [];    }
```

Here $configFile is set to null, hence from the else condition i.e $this->file="config.php" it will include /app/etc/config.php in line no- 79.

- **Line 220 :** Create App

  Create function of object (Magento\Framework\ObjectManager\ObjectManager) will intialize HTTP App object which in turn will return instance of '**Magento\Framework\App\Http'** .

3. Run App

Now **run($app) function** of bootstrap instance will be called.

```

```

```
/** @var \Magento\Framework\App\Http $app */
$app = $bootstrap->createApplication('Magento\Framework\App\Http');
$bootstrap->run($app);
```

- **Run Function**

```
    /**
     * Runs an application
     *
     * @param \Magento\Framework\AppInterface $application
     * @return void
     */
    public function run(AppInterface $application)
    {
        try {
            try {
                \Magento\Framework\Profiler::start('magento');
                $this->initErrorHandler();
                $this->initObjectManager();
                $this->assertMaintenance();
                $this->assertInstalled();
                $response = $application->launch();
                $response->sendResponse();
                \Magento\Framework\Profiler::stop('magento');
            } catch (\Exception $e) {
                \Magento\Framework\Profiler::stop('magento');
                if (!$application->catchException($this, $e)) {
                    throw $e;
                }
            }
        } catch (\Exception $e) {
            $this->terminate($e);
        }
    }
```

- **Line 241 : *initErrorHandler()*** initializes error Handler.
- **Line 242 : *initObjectManager()*** initializes object manager if it's not initialized.
- **Line 243 : *assertMaintenance()*** inspects maintenance flag status. If set 'yes', maintenance page will display at forntend.
- **Line 244 : *assertInstalled()*** check if installation is completed, if not then will throw exception and exception handler will redirect it to setup directory.
- **Line 245 : *launch()***

```
    /**

     *


     */

    {
```

*

rontName());

```
                $this->_state->setAreaCode($areaCode);
                $this->_objectManager->configure($this->_configLoader->load($areaCode));
\Magento\Framework\App\FrontControllerInterface $frontController
                                                        $result = $frontController->disp
Temporary solution till all controllers are returning ResultInterface (MAGETWO-28359)
                                                    if ($result instanceof
    $this->registry->register('use_page_cache_plugin', true, true);
                    $result->renderResult($this->_response);
                                        } elseif ($result instanceof HttpI


                    throw new \InvalidArgumentException('Invalid return type');
                }
    This event gives possibility to launch something before sending output (allow cookie setting)
    $eventParams = ['request' => $this->_request, response => $this->_response];

                return $this->_response;    }
```

- **Line 110 :** retrieves area code (**frontend/adminhtml**)
- **Line 111 :** set up area code
- **Line 112 : Load Config and Configure**

  configuration is loaded according to area and by calling configure function, this merges all loaded configuration.
- **Line 114 : Get Front Controller**
  - here, get function will retrieve main front controller instance according to interface “**Magento\Framework\App\FrontControllerInterface**” with help of **di.xml** (dependency injection)

```
/vendor/magento/framework/ObjectManager/etc/config.xsd">
Psr\Log\LoggerInterface" type="Magento\Framework\Logger\Monolog" />
\Template\Html\Minifier" />
Dynamic\Developer" />

esolver" />
tion\Status" />
pp\RequestType\Http" type="Magento\Framework\App\Request\Http" />
quest\PathInfoProcessor" />
Response\Http" type="Magento\Framework\App\Response\Http" />
\RouterList" type="Magento\Framework\App\RouterList" />
\Framework\App\FrontController" />
ork\App\Cache\Proxy" type="Magento\Framework\App\Cache" />
che\State" type="Magento\Framework\App\Cache\State" />
\Framework\App\Cache\TypeList" />
toreManager" type="Magento\Store\Model\StoreManager" />
esign\Proxy" type="Magento\Theme\Model\View\Design" />
ork\View\Design\Theme" type="Magento\Theme\Model\Theme" />
\Framework\Theme\Model\Theme\Resolver" />
mework\View\Config" type="Magento\Framework\View\Config" />
ork\View\Asset\Bundle\Config" />
ork\Locale\Lists" type="Magento\Framework\Locale\Lists" />
tributeTypeResolver" />

                <arguments>
```

```
            </arguments>
        </type>
\Locale\Config" type="Magento\Framework\Locale\Config" />
ework\Notification\NotifierPool" />
    <preference for="Magento\Framework\UrlInterface" type="Magento\Framework\Url" />
mework\Url\EncoderInterface" type="Magento\Framework\Url\Encoder" />
mework\Url\DecoderInterface" type="Magento\Framework\Url\Decoder" />
y\Query" />
ork\Config\ScopeInterface" type="Magento\Framework\Config\Scope" />
er" type="Magento\Framework\App\Config\FileResolver" />
```

line number 19 : “Magento\Framework\App\FrontController” type is defined for interface “**Magento\Framework\App\FrontControllerInterface**“

- After retrieving front controller it will call factory create function which creates instance of Interceptor class and returns it to **Magento\Framework\App\FrontController\Interceptor**

- **Line 115 : dispatch()** function
- Dispatch function of interceptor will be called, which in turn calls dispatch function of main front controller i.e. **Magento\Framework\App\FrontController**, further dispatch function of FrontController will call **processRequest()** for **Routing.**

ROUTING

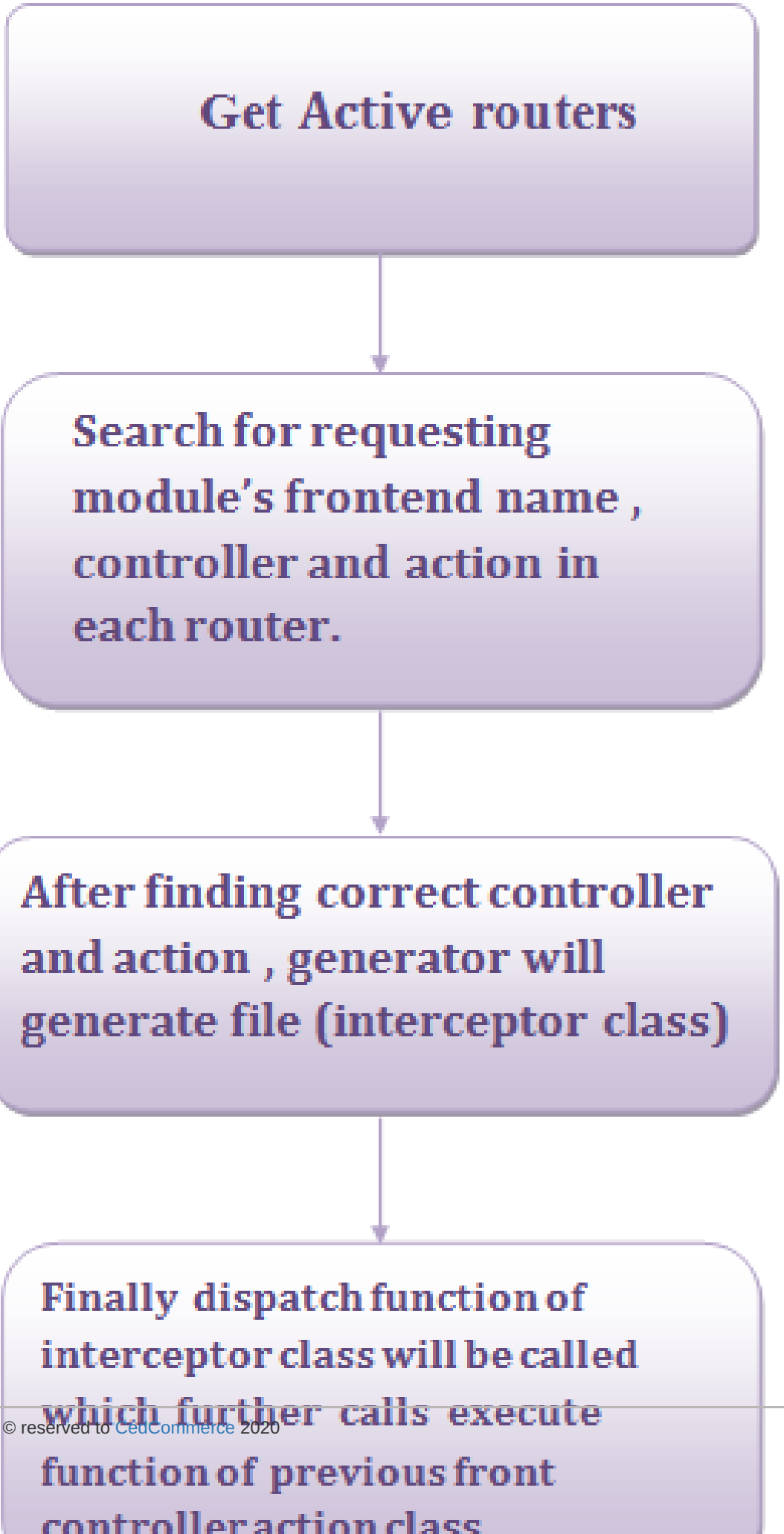


Figure 4 – Match routers

Routing has an algorithm to find a matching controller, determined by request.

- **ProcessRequest function of frontController**

```
    /**
     * Route request and dispatch it
     *
     * @param RequestInterface $request
     * @return ResponseInterface|\Magento\Framework\Controller\ResultInterface|null
     */
    protected function processRequest(RequestInterface $request)
    {
        $result = null;
        /** @var \Magento\Framework\App\RouterInterface $router */
        foreach ($this->_routerList as $router) {
            try {
                $actionInstance = $router->match($request);
                if ($actionInstance) {
                    $request->setDispatched(true);
                    $actionInstance->getResponse()->setNoCacheHeaders();
                    try {
                        $result = $actionInstance->dispatch($request);
                    } catch (\Magento\Framework\Exception\NotFoundException $e) {
                        throw $e;
                    } catch (\Exception $e) {
                        $this->handleException($e);
                        $result = $actionInstance->getDefaultResult();
                    }
                    break;
                }
            } catch (\Magento\Framework\Exception\NotFoundException $e) {
                $request->initForward();
                $request->setActionName('noroute');
                $request->setDispatched(false);
                break;
            }
        }
        return $result;    }
```

line 106 : is looping routerList and extracting the required one.

- **Get Active routers list**
- Get routers list from modules **di.xml**
- Lets take an example of frontend Magento Store module's **di.xml**

```
                  <type     name="Magento\Framework\App\RouterList"     shared="true">
        <arguments>



            <item name="disable" xsi:type="boolean">false</item>
                                 <item  name="sortOrder"  xsi:type="string">20</item>
                <item>
```

```
            <item name="disable" xsi:type="boolean">false</item>
                        <item name="sortOrder" xsi:type="string">100</item>
                    </item>
                </argument>
            </arguments>    </type>
```

Above config is adding two routers (standard and default) in routerList.

It can check Line number 20 and 25. In both, disable parameter is set to false means both are active.

Similarly you can use same method in your custom module for creating your own router.

- **Line 108: match()**
  - It is looping through each router and calling match function for every router.
  - Match function will search for module's frontend name, controller and action.
  - Lets take router "Magento\Framework\App\Router\Base" so line 108 will call **match** function of **Magento\Framework\App\Router\Base.**
  - Then match function will further call **matchAction** of same instance.
  - If match is found then it will call create function of factory which will create interceptor class file of found controller and then return instance of it.
- **Line 113 :** is calling **dispatch()** function of iterceptor class
  - Dispatch function of interceptor will call main controller's dispatch function and finally **execute** function of controller will be called.

CONTROLLER

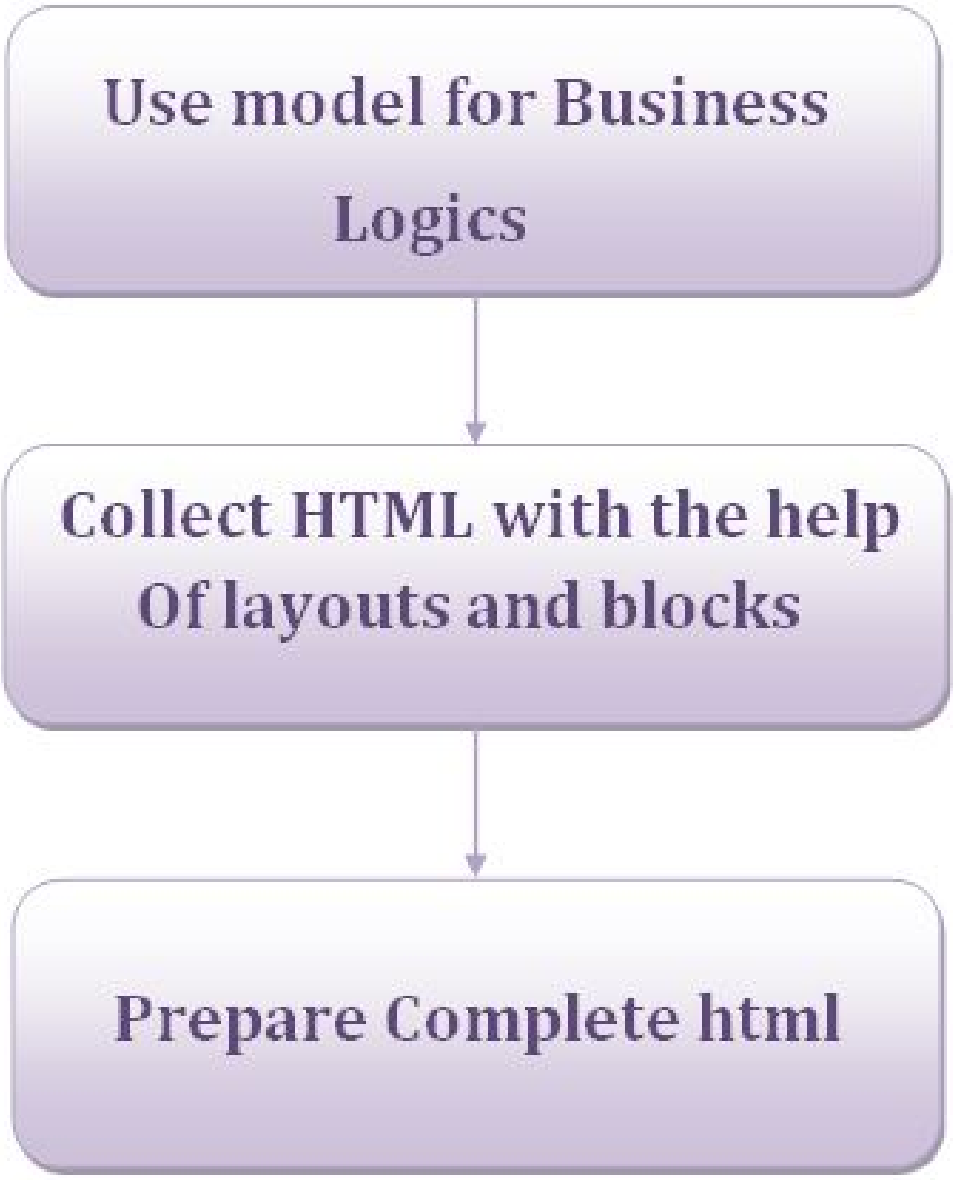


Figure 5 – Prepare HTML

- LoadLayout
  - this function is responsible for adding default handlers as well as handlers defined by current controller.
  - after that it loads layout configuration of all handlers added in previous step.
  - prepares collection of all blocks associated with handlers.
- RenderLayout
  - it renders all handler's layout added previously and generate html output and appends it to a output variable.

HTTP Response

This is the end phase of magento execution where prepared output will be finally sent to requesting agent as HTTP response.

- Send Response line 246

```
    public function run(AppInterface $application)
    {
        try {
            try {
                \Magento\Framework\Profiler::start('magento');
                $this->initErrorHandler();
                $this->initObjectManager();
                $this->assertMaintenance();
                $this->assertInstalled();
                $response = $application->launch();
                $response->sendResponse();
                \Magento\Framework\Profiler::stop('magento');
            } catch (\Exception $e) {
                \Magento\Framework\Profiler::stop('magento');
                if (!$application->catchException($this, $e)) {
                    throw $e;
                }
            }
        } catch (\Exception $e) {
            $this->terminate($e);
        }    }
```

  - sendContent

```
    /**
     * Send content
     *
     * @return Response
     */
    public function sendContent()
    {
        if ($this->contentSent()) {
            return $this;
        }
        echo $this->getContent();
        $this->contentSent = true;
        return $this;    }
```

Line 116 : $this->getContent() finally echo/print/output the content prepared by roaming inside magento application .

If any issues / suggestion , comments are most appreciated . Next in the journey will create Products Featured Slider module in magento 2 .

Contact Us

Have a query? Let us know more about it.

Here’s how you can reach out to us.

Connect with the Sales & Support Team

support@cedcommerce.com

Connect with the Marketing Team

marketing@cedcommerce.com

Want us to call you ?

Raise a Ticket (24*7)

India HeadQuarter

(+91) -7234976892

USA Sales and Marketing

888-882-0953

Add us on Skype

live:support_35785

Visit Us At

USA

Portland, Oregon

CedCommerce Inc, 1812 N Columbia Blvd, Suite C15-653026, Portland, Oregon, 97217

UK

Leicester, Leicestershire

3rd Floor, St. George' s House, 6 St George' s Way, Leicester, Leicestershire, LE1 1QZ

India

Lucknow, UP

3/460, First Floor, Vishwas Khand, Gomti Nagar, Lucknow, Uttar Pradesh 226010

Malaysia

Selangor Darul Ehsan

Suite 20-01 & 20-02B, Level 20, The Persiaran Lagoon Bandar Sunway, Subang Jaya Selangor Darul Ehsan 47500