Magento 2 
Developer's Guide

Magento is one of the most exciting, flexible, and customizable e-commerce system. After years of development, Magento 2 introduces itself with a strong emphasis on modularity, web APIs, automated testing, and a new technology stack platform. This book is your one-stop guide to building and customizing a quality e-commerce website from the latest version of the most popular e-commerce platform—Magento 2.

We start off with an introduction to the fundamental concepts of Magento to give you a foundation to work from. After this, you’ll get to grips with the major concepts and conventions that are new to the Magento 2 platform. You will also be guided through building a full-blown helpdesk module from scratch.

Who this book is written for
This book is intended primarily for intermediate to professional-level PHP developers who are interested in Magento development. For backend developers, several topics are covered that will enable you to modify and extend your Magento 2 store. Frontend developers will also find some coverage of how to customize the look of the Web Store.

What you will learn from this book
- Set up the development and production environment of Magento 2
- Understand the new major concepts and conventions used in Magento 2
- Build a miniature yet fully-functional module from scratch to manage your e-commerce platform efficiently
- Write models and collections to manage and search your entity data
- Dive into backend development, such as creating events, observers, cron jobs, logging, profiling, and messaging features
- Get to the core of frontend development, such as blocks, templates, layouts, and the themes of Magento 2
- Use token, session, and OAuth token-based authentication via various flavors of API calls, as well as creating your own APIs
- Get to grips with testing Magento modules and custom themes, which forms an integral part of development

In this package, you will find:

- The author biography
- A preview chapter from the book, Chapter 1 'Understanding the Platform Architecture'
- A synopsis of the book’s content
- More information on Magento 2 Developer’s Guide
About the Author

Branko Ajzele is a husband, father of two, son, brother, author, and a software developer.

He has a degree in electrical engineering. A lover of all things digital, he makes a living from software development. He hopes to find enough quality time some day to seriously dive into hobby electronics; he has his eye on Arduino and Raspberry Pi.

He has years of hands-on experience with full-time software development and team management, and has specializing in e-commerce platforms. He has been working with Magento since 2008; he has been knee-deep in it since its very first beta version. Branko is regularly in touch with everything related to PHP, databases (MySQL/ MongoDB), search/analytics (Solr/Elasticsearch), Node.js, and related technologies.

He has a strong technical knowledge with an ability to communicate those technicalities frequently and clearly with a strong direction. He feels comfortable proposing alternatives to demands which he feels can be improved, even when this means pulling a late shift to meet the deadlines.

He holds several respected IT certifications, such as Zend Certified Engineer (ZCE PHP), Magento Certified Developer (MCD), Magento Certified Developer Plus (MCD+), Magento Certified Solution Specialist (MCSS), and JavaScript Certified Developer.

*Instant E-Commerce with Magento: Build a Shop, Packt Publishing,* was his first Magento-related book that was oriented towards Magento newcomers. After writing this book, he wrote *Getting Started with Magento Extension Development* for developers.

Currently, he works as a full-time contractor for Lab Lateral Ltd, an award-winning team of innovative thinkers, artists, and developers who specialize in customer-centric websites, digital consultancy, and marketing. He is the Lead Magento Developer and Head of Lab’s Croatia office.

He was awarded the *E-Commerce Developer of the Year* by Digital Entrepreneur Awards in October 2014 for his excellent knowledge and expertise in e-commerce development. His work is second to none. He is truly dedicated to helping the Lab Lateral Ltd team and his fellow developers across the world.
Preface

Building Magento-powered stores can be a challenging task. It requires a great range of technical skills that are related to the PHP/JavaScript programming language, development and production environments, and numerous Magento-specific features. This book will provide necessary insights into the building blocks of Magento.

By the end of this book, you should be familiar with configuration files, the dependency injection, models, collections, blocks, controllers, events, observers, plugins, cron jobs, shipping methods, payment methods, and a few other things. All of these should form a solid foundation for your development journey later on.

What this book covers

Chapter 1, *Understanding the Platform Architecture*, gives a high-level overview of the technology stack, architectural layers, top-level system structure, and individual module structure.

Chapter 2, *Managing the Environment*, gives an introduction to VirtualBox, Vagrant, and Amazon AWS as platforms to set up development and production environments. It further provides hands-on examples to set up/script Vagrant and Amazon EC2 boxes.

Chapter 3, *Programming Concepts and Conventions*, introduces readers to a few seemingly unrelated but important parts of Magento, such as composer, service contracts, code generation, the var directory, and finally, coding standards.

Chapter 4, *Models and Collections*, takes a look into models, resources, collections, schemas, and data scripts. It also shows the practical CRUD actions that are applied to an entity alongside filtering collections.
Preface

Chapter 5, Using the Dependency Injection, guides readers through the dependency injection mechanism. It explains the role of an object manager, how to configure class preferences, and how to use virtual types.

Chapter 6, Plugins, gives a detailed insight into the powerful new concept called plugins. It shows how easy it is to extend, or add to, an existing functionality using the before/after/around listeners.

Chapter 7, Backend Development, takes readers through a hands-on approach to what is mostly considered backend-related development bits. These involve cron jobs, notification messages, sessions, cookies, logging, profiler, events, cache, widgets, and so on.

Chapter 8, Frontend Development, uses a higher-level approach to guide the reader through what is mostly considered frontend-related development. It touches on rendering the flow, view elements, blocks, templates, layouts, themes, CSS, and JavaScript in Magento.

Chapter 9, The Web API, takes up a detailed approach to the powerful Web API provided by Magento. It gives hands-on practical examples to create and use both REST and SOAP, either through the PHP cURL library, or from the console.

Chapter 10, The Major Functional Areas, adopts a high-level approach towards introducing readers with some of the most common sections of Magento. These include CMS, catalog and customer management, and products and customer import. It even shows how to create a custom product type and a shipping and payment method.

Chapter 11, Testing, gives an overview of the types of test that are available in Magento. It further shows how to write and execute a custom test.

Chapter 12, Building a Module from Scratch, shows the entire process of developing a module, which uses most of the features introduced in the previous chapters. The final result is a module that has admin and storefront interface, an admin configuration area, e-mail templates, installed schema scripts, tests, and so on.
Magento is a powerful, highly scalable, and highly customizable e-commerce platform that can be used to build web shops and, if needed, some non-e-commerce sites. It provides a large number of e-commerce features out of the box.

Features such as product inventory, shopping cart, support for numerous payment and shipment methods, promotion rules, content management, multiple currencies, multiple languages, multiple websites, and so on make it a great choice for merchants. On the other hand, developers enjoy the full set of merchant-relevant features plus all the things related to actual development. This chapter will touch upon the topic of robust Web API support, extensible administration interface, modules, theming, embedded testing frameworks, and much more.

In this chapter, a high-level overview of Magento is provided in the following sections:

- The technology stack
- The architectural layers
- The top-level filesystem structure
- The module filesystem structure
The technology stack

Magento's highly modular structure is a result of several open source technologies embedded into a stack. These open source technologies are composed of the following components:

- **PHP**: PHP is a server-side scripting language. This book assumes that you have advanced knowledge of the object-oriented aspects of PHP, which is often referred to as PHP OOP.
- **Coding standards**: Magento puts a lot of emphasis on coding standards. These include PSR-0 (the autoloading standard), PSR-1 (the basic coding standards), PSR-2 (the coding style guide), PSR-3, and PSR-4.
- **Composer**: Composer is a dependency management package for PHP. It is used to pull in all the vendor library requirements.
- **HTML**: HTML5 is supported out of the box.
- **CSS**: Magento supports CSS3 via its in-built LESS CSS preprocessor.
- **jQuery**: jQuery is a mature cross-platform JavaScript library that was designed to simplify the DOM manipulation. It is one of the most popular JavaScript frameworks today.
- **RequireJS**: RequireJS is a JavaScript file and module loader. Using a modular script loader such as RequireJS helps improve the speed and quality of code.
- **Third-party libraries**: Magento comes packed with a lot of third-party libraries, with the most notable ones being Zend Framework and Symfony. It is worth noting that Zend Framework comes in two different major versions, namely version 1.x and version 2.x. Magento uses both of these versions internally.
- **Apache or Nginx**: Both Apache and Nginx are HTTP servers. Each has its distinct advantages and disadvantages. It would be unfair to say one is better than another, as their performance widely depends on the entire system’s setup and usage. Magento works with Apache 2.2 and 2.4 and Nginx 1.7.
- **MySQL**: MySQL is a mature and widely used relational database management system (RDBMS) that uses structured query language (SQL). There are both free community versions and commercial versions of MySQL. Magento requires at least the of MySQL Community Edition version 5.6.
- **MTF**: Magento Testing Framework (MTF) delivers an automated testing suite. It covers various types of tests, such as performance, functional, and unit testing. The entire MTF is available on GitHub, which can be viewed by visiting https://github.com/magento/mtf as an isolated project.
Different pieces of technology can be glued into various architectures. There are different ways to look at the Magento architecture—from the perspective of a module developer, system integrator, or a merchant, or from some other angle.

The architectural layers
From top to bottom, Magento can be divided into four architectural layers, namely presentation, service, domain, and persistence.

The presentation layer is the one that we directly interact with through the browser. It contains layouts, blocks, templates, and even controllers, which process commands to and from the user interface. Client-side technologies such as jQuery, RequireJS, CSS, and LESS are also a part of this layer. Usually, three types of users interact with this layer, namely web users, system administrators, and those making the Web API calls. Since the Web API calls can be made via HTTP in a manner that is the same as how a user uses a browser, there's a thin line between the two. While web users and Web API calls consume the presentation layer as it is, the system administrators have the power to change it. This change manifests in the form of setting the active theme and changing the content of the CMS (short for content management system) pages, blocks, and the products themselves.

When the components of a presentation layer are being interacted with, they usually make calls to the underlying service layer.

The service layer is the bridge between the presentation and domain layer. It contains the service contracts, which define the implementation behavior. A service contract is basically a fancy name for a PHP interface. This layer is where we can find the REST/SOAP APIs. Most user interaction on the storefront is routed through the service layer. Similarly, the external applications that make the REST/SOAP API calls also interact with this layer.

When the components of a service layer are being interacted with, they usually make calls to the underlying domain layer.

The domain layer is really the business logic of Magento. This layer is all about generic data objects and models that compose the business logic. The domain layer models themselves do not contribute to data persistence, but they do contain a reference to a resource model that is used to retrieve and persist the data to a MySQL database. A domain layer code from one module can interact with a domain module code from another module via the use of event observers, plugins, and the di.xml definitions. We will look into the details of these later on in other chapters. Given the power of plugins and di.xml, it’s important to note that this interaction is best established using service contracts (the PHP interface).
When the components of the domain layer are being interacted with, they usually make calls to the underlying persistence layer.

The persistence layer is where the data gets persisted. This layer is in charge of all the CRUD (short for create, read, update, and delete) requests. Magento uses an active record pattern strategy for the persistence layer. The model object contains a resource model that maps an object to one or more database rows. Here, it is important to differentiate the cases of simple resource model and the Entity-Attribute-Value (EAV) resource models. A simple resource model maps to a single table, while the EAV resource models have their attributes spread out over a number of MySQL tables. As an example, the Customer and Catalog resource models use EAV resource models, while the newsletter's Subscriber resource model uses a simple resource model.

**The top-level filesystem structure**

The following list depicts the root Magento filesystem structure:

- .htaccess
- .htaccess.sample
- .php_cs
- .travis.yml
- CHAGELOG.md
- CONTRIBUTING.md
- CONTRIBUTOR_LICENSE AGREEMENT.html
- COPYING.txt
- Gruntfile.js
- LICENSE.txt
- LICENSE_AFL.txt
- app
- bin
- composer.json
- composer.lock
- dev
- index.php
The **app/etc/di.xml** file is one of the most important files that we might often look into during development. It contains various class mappings or preferences for individual interfaces.

The **var/magento/language-*** directories is where the registered languages reside. Though each module can declare its own translations under app/code/{VendorName}/{ModuleName}/i18n/, Magento will eventually fall back to its own individual module named i18n in case translations are not found in the custom module or within the theme directory.

The **bin** directory is where we can find the magento file. The magento file is a script that is intended to be run from a console. Once triggered via the php bin/magento command, it runs an instance of the Magento\Framework\Console\Cli application, presenting us with quite a number of console options. We can use the magento script to enable/disable cache, enable/disable modules, run an indexer, and do many other things.

The **dev** directory is where we can find the Magento test scripts. We will have a look at more of those in later chapters.

The **lib** directory comprises two major subdirectories, namely the server-side PHP library code and fonts found under lib/internal and the client-side JavaScript libraries found in lib/web.

The **pub** directory is where the publicly exposed files are located. This is the directory that we should set as root when setting up Apache or Nginx. The pub/index.php file is what gets triggered when the storefront is opened in a browser.
The **var** directory is where the dynamically generated group type of files such as cache, log, and a few others get created in. We should be able to delete the content of this folder at any time and have Magento automatically recreate it.

The **vendor** directory is where most of the code is located. This is where we can find various third-party vendor code, Magento modules, themes, and language packs. Looking further into the **vendor** directory, you will see the following structure:

- .htaccess
- autoload.php
- bin
- braintree
- composer
- doctrine
- fabpot
- justinrainbow
- league
- lusitanian
- magento
- monolog
- oyejorge
- pdepend
- pelago
- phpmd
- phpseclib
- phpunit
- psr
- sebastian
- seld
- sjparkinson
- squizlabs
- symfony
- tedivm
- tubalmartin
- zendframework
Within the vendor directory, we can find code from various vendors, such as `phpunit`, `phpseclib`, `monolog`, `symfony`, and so on. Magento itself can be found here. The Magento code is located under `vendor/magento` directory, listed (partially) as follows:

- `composer`
- `framework`
- `language-en_us`
- `magento-composer-installer`
- `magento2-base`
- `module-authorization`
- `module-backend`
- `module-catalog`
- `module-customer`
- `module-theme`
- `module-translation`
- `module-ui`
- `module-url-rewrite`
- `module-user`
- `module-version`
- `module-webapi`
- `module-widget`
- `theme-adminhtml-backend`
- `theme-frontend-blank`
- `theme-frontend-luma`

You will see that the further structuring of directories follows a certain naming schema, whereas the `theme-*` directory stores themes, the `module-*` directory stores modules, and the `language-*` directory stores registered languages.
The module filesystem structure

Magento identifies itself as a highly modular platform. What this means is that there is literally a directory location where modules are placed. Let's take a peak at the individual module structure now. The following structure belongs to one of the simpler core Magento modules—the Contact module that can be found in vendor/magento/module-contact:

- Block
- composer.json
- Controller
- etc
  - acl.xml
  - adminhtml
    - system.xml
  - config.xml
  - email_templates.xml
  - frontend
    - di.xml
    - page_types.xml
    - routes.xml
  - module.xml
- Helper
- i18n
- LICENSE_AFL.txt
- LICENSE.txt
- Model
- README.md
- registration.php
- Test
  - Unit
    - Block
    - Controller
    - Helper
    - Model
- view
  - adminhtml
  - frontend
    - layout
Even though the preceding structure is for one of the simpler modules, you can see that it is still quite extensive.

The Block directory is where the view-related block PHP classes are located.

The Controller directory is where the controller-related PHP classes are stored. This is the code that responds to the storefront POST and GET HTTP actions.

The etc directory is where the module configuration files are present. Here, we can see files such as module.xml, di.xml, acl.xml, system.xml, config.xml, email_templates.xml, page_types.xml, routes.xml, and so on. The module.xml file is an actual module declaration file. We will look into the contents of some of these files in the later chapters.

The Helper directory is where various helper classes are located. These classes are usually used to abstract various store configuration values into the getter methods.

The i18n directory is where the module translation package CSV files are stored.

The Module directory is where the entities, resource entities, collections, and various other business classes can be found.

The Test directory stores the module unit tests.

The view directory contains all the module administrator and storefront template files (.phtml and .html) and static files (.js and .css).

Finally, the registration.php is a module registration file.

Summary

In this chapter, we took a quick look at the technology stack used in Magento. We discussed how Magento, being an open source product, takes extensive use of other open source projects and libraries such as MySQL, Apache, Nginx, Zend Framework, Symfony, jQuery, and so on. We then learned how these libraries are arranged into directories. Finally, we explored one of the existing core modules and briefly took a look at an example of a module's structure.

In the next chapter, we are going to tackle the environment setup so that we can get Magento installed and ready for development.
Where to buy this book


Click here for ordering and shipping details.