The DOs of Magento 2 Extension Development

EXT DN

DO USE COMPOSER

Use composer packages to distribute (especially commercial) extensions. For a local environment, it is fine to develop your own code under `app/code`. However, once you distribute your module to other environments, it should be through composer as otherwise dependencies are left unmanaged. In a production environment, the `app/code` folder should therefore ideally be empty.

DO USE SERVICE CONTRACTS

Whenever a dependency is injected into a class try to abstract this dependency. If there is a preference for a specific class, in the form of a PHP interface, you should inject the interface instead. Inject only classes or interfaces (aka Service Contracts) that are either part of the Magento framework or the API of specific modules.

DO WRITE TESTS

Testing becomes a habit that pays off in the long run. Make sure functionality is tested using integration tests and functional tests. Remember that unit tests make more sense with isolated code, integration tests make more sense with code that touches the rest of Magento.

DO DOCUMENT YOUR DEPENDENCIES

If your module depends on other modules, make sure that both your composer file and your `module.xml` file reflect this. If your module only depends on the Magento framework, your module should likely be treated as a library, not a module. Your composer version constraints should respect the semantic versioning standards of Magento.

Collectively, ExtDN members are committed to advancing Magento as the world's leading ecommerce platform for merchants.

EXT DN

DO VERSION RELEASES

Whenever you have made a change, increase your composer package version number. Follow semantic versioning: Major for compatibility changes, minor for features, patch for fixes. Only increase the `setup_version` number in your `module.xml` file when changes to the database are needed.

DO PROVIDE A USER MANUAL

All users should get clear instructions on how to install, update and remove your extension as well as how to use the provided functionality. This could include a technical overview of your extension, documenting your customisation points like observable events and your public API definition.

DO USE EVENTS AND PLUGINS

Use events and plugins on `@api` marked methods as primary means of customising functionality. When intercepting functionality of non-API methods using plugins, which is less recommended, make sure to reflect this in your composer version constraints. Avoid redefining existing preferences where possible.

DO CHECK YOUR CODE

Use a static analysis tool like PHP CodeSniffer (with the ExtDN and MEQP rulesets). Check whether your extension works in Production Mode. Confirm your extension works under the Magento versions that you claim compatibility with. Have a colleague or friend review your code before releasing it.