Oxwall Documentation

Release latest

Oxwall Foundation

November 24, 2016

Contents

Subject:

CHAPTER 1

Application Lifecycle

in progress

CHAPTER 2

Main Application Service

in progress

CHAPTER 3

Routing and Controllers

in progress

CHAPTER 4

View and Components

in progress

CHAPTER 5

Database and Models

in progress

CHAPTER 6

Authorization

in progress

CHAPTER 7

Cron (Task Scheduler)

in progress

CHAPTER 8

Translation files

in progress

CHAPTER 9

Widgets

in progress

CHAPTER 10

Plugin Structure

Oxwall is one of the most simple platforms that uses MVC architectural pattern for implementing user interfaces. It helps web developers to manage the complex tasks and understand the web frameworks written in PHP.

Oxwall plugins are developed basing on the MVC pattern which's principle is to separate the application into 3 main parts, known as the Model, the View, and the Controller - MVC (Model View Controller). Below you can find the specific information about the MVC architectural pattern components and Oxwall plugin structure:

10.1 Model

The Model handles the direct interaction with database or other data sources. Model mainly consists of queries to database and works on the data sending to database and retrieving it by converting it into a particular format. Oxwall uses ORM programming technique. However, it also provides the possibility to write your own queries to database omitting ORM.

10.2 View

The View is responsible for the presentation of the information in the appropriate format. It should mainly contain presentational code, such as HTML, and simple PHP code and all business logic should be moved out, where feasible, to Model, Controller or Component. That implies plain and simple templates for the data presentation. Oxwall's templates have .html extension.

Component - this is a part of encapsulated logic of View. i.e. the Component is some kind of the View helper. The Component is used in case it's required to indicate the business logic which is mostly should be represented as the part of View than as the part of Controller. Also, Components are used during the View logic reusing. For example, the form or menu representation. Components can have its own template file (.html), and can be called via Ajax. More information about the Components can be found in the *View and Components* section.

10.3 Controller

Controller handles the data received from different 'clients' and outputs it using the View or Component files. It's a kind of glue that binds models, views and other components together into a runnable application.

More information about the Controllers can be found in the *Routing and Controllers* section.

10.4 The plugin directory structure:

/
- - **bol/**
- - **classes/**
- - **components/**
- - **controllers/**
- - **mobile/**
- - - - **classes/**
- - - - **components/**
- - - - **controllers/**
- - - - **views/**
- - - - *init.php*
- - **static/**
- - - - **css/**
- - - - **js/**
- - - - **images/**
- - **views/**
- - - - **components/**
- - - - **controllers/**
- - **update/**
- - *init.php*
- - *cron.php*
- - *activate.php*
- - *deactivate.php*
- - *install.php*
- - *uninstall.php*
- - *langs.zip*
- - *plugin.xml*

1. **bol** - contains the model files, which work with the database or any other data sources. Find more information on how to work with the models here: *Database and Models*
2. **classes** - contains various classes of the plugin, which are not related to the model and/or controller, such as classes of the system events or downloaded libraries.
3. **controllers** - contains controllers classes, which are directly deal with the end user requests by the registered routes. Find more information about the routing and controllers here: *Routing and Controllers*.
4. **mobile** - contains classes of controllers, models and components which are required for the mobile context operations. i.e. it's used to represent the mobile content when a user clicks the 'Mobile version' link or when the system automatically detects if the content is requested from the mobile clients.
5. **static** - contains .js, .css, and image files needed for the correct plugin functionality. It should be noted that this directory is not accessible for the end users requests and all .js, .css or image files which are included in the plugin controller - are copied automatically to the directory called /ow_static that's available through http and located in the root of the software.
6. **views** - contains files of all plugin controllers and components representation. Take into account that the files of the representation will be called automatically for all controller or component methods until you force the

method to stop execution (for instance, it's required for the ajax requests which can be executed without representation files).

7. **update** - contains the files of the plugin updates. Find more information about the plugin update here: *Plugin Update*.
8. **init.php** - this file contains the script that runs initially. This script is called all the time during the plugin initialization. The main task of this file is registering the plugin routes (more information can be found here: *Routing and Controllers*). Also, it handles additional functionality, such as collects event subscribers or other system events.
9. **cron.php** - this file runs background tasks by the time-based job scheduler called CRON. Find more information about the CRON here: *Cron (Task Scheduler)*. This file should be created only in case the plugin has some time-based functionality. For example, send automatic letters. If this file exists in the plugin - it will be automatically included and used by the system.
10. **activate.php** - this file contains the plugin logic that should be executed when the administrator activates plugin in the Admin Panel. For example, the methods which add plugin widgets (more information can be found here: *Widgets*) or menu items onto the certain positions. This file is optional, i.e. if there is no specific functionality that should be run on the plugin activation, this file can be omitted. If this file exists in the plugin - it will be automatically included and used by the system.
11. **deactivate.php** - this file contains the plugin logic that should be executed when the administrator deactivates plugin in the Admin Panel. For example, the methods which remove plugin widgets (more information can be found here: *Widgets*) or menu items from their positions. This file is optional, i.e. if there is no specific functionality that should be run on the plugin deactivation, this file can be omitted. If this file exists in the plugin - it will be automatically included and used by the system.
12. **install.php** - this file runs only during the plugin installation. It can be used to run SQL queries which, for example, create tables in the database, import language file (more information can be found here: *Translation files*), register new plugin settings or authorization groups/actions (more information can be found here: *Authorization*). This file is optional, i.e. if there is no specific functionality that should be run on the plugin installation, this file can be omitted. If this file exists in the plugin - it will be automatically included and used by the system.
13. **uninstall.php** - this file runs only during the plugin uninstallation. It can be used to run SQL queries which, for example, remove tables which were created during the installation. There is no need to include methods which will remove the plugin settings, authorization groups/actions or language translation because such entries are removed automatically by the system. This file is optional, i.e. if there is no specific functionality that should be run on the plugin uninstallation, this file can be omitted. If this file exists in the plugin - it will be automatically included and used by the system.
14. **plugin.xml** - this file contains all the necessary service information about the plugin. Below you can find the file structure.

10.5 plugin.xml file structure:

```
<?xml version="1.0" encoding="utf-8"?>
<plugin>
    <name>My Super Plugin</name>
    <key>superplugin</key>
    <description>My super plugin.</description>
    <author>Me</author>
    <authorEmail>me@oxwall.org</authorEmail>
    <authorUrl>http://www.me.com</authorUrl>
    <developerKey>MY_DEV_KEY</developerKey>
    <build>1</build>
    <copyright>(C) 2015 My. All rights reserved.</copyright>
```

```
    <license>OSCL</license>
    <licenseUrl>http://www.oxwall.org/store/oscl</licenseUrl>
</plugin>
```

1. **name** - plugin name.
2. **key** - plugin name. It's required to use lower case and latin letters only [a-z]. Before choosing the name, you should make sure that there is no plugin with this name. To do so, please go to the Developer Tools page at Oxwall.org: http://www.oxwall.org/store/dev-tools.
3. **description** - short description of the plugin functionality.
4. **author** - the name of the plugin developer.
5. **authorEmail** - the plugin developer email.
6. **developerKey** - the plugin developer key. It's needed for the further plugin updates. This key can be found on the Developer Tools page at Oxwall.org: http://www.oxwall.org/store/dev-tools. Find the detailed instructions on how to get the key and sell your plugins in Oxwall Store here: *Getting developer and plugin keys*.
7. **build** - number of the plugin build. It's needed for the further plugin updates.
8. **copyright** - the information about the plugin copyright.
9. **license** - the type of the license thats used for the plugin.
10. **licenseUrl** - URL of the page with detailed description of the chosen license.

CHAPTER 11

Plugin Update

in progress

CHAPTER 12

Getting developer and plugin keys

in progress