ZF2Guard

Dzen Framework

Методичка по организации контроля доступа от Dolphin

Время пришло.

Оглавление

I. Используемый инструментарий

Используемый инструментарий:

ОС – Windows XP

Я думаю, что пользователи Unix-систем вполне смогут применить полученную здесь информацию.

Web - сервер – XAMPP (http://ru.wikipedia.org/wiki/XAMPP)

Все примеры выполнены на XAMPP. Если раньше вы использовали Denwer, не расстраивайтесь – Denwer может прекрасно работать даже с флешки. Поэтому вы можете установить XAMPP на свой компьютер без опасений, главное – не пытайтесь запускать XAMPP и Denwer одновременно.

Чтобы установить XAMPP перейдите по адресу http://www.apachefriends.org/en/xampp-windows.html и прокрутите страницу до надписи Installer. По установке и настройке XAMPP Google выдает **огромное** количество ресурсов на русском языке. Фактически, на сегодняшний день это наиболее популярная сборка веб-сервера.

Файловый менеджер – Far версии 2.0 и выше

Пользователи Unix-систем наверняка пользуются аналогичной программой – mc. Если вы забыли, как использовать Far, хочу напомнить:

1) Alt + F1 – выбор диска в левой панели. Alt + F2 – выбор диска в правой панели. Переключение между панелями с помощью Tab. Меню – по F9, но многие предпочитают, чтобы оно не скрывалось автоматически.
2) По умолчанию Far при запуске открывает собственную директорию. Перейдите в каталог C:\xampp\htdocs и выберите Параметры – > Сохранить параметры (или комбинация клавиш Shift + F9). Когда вы в следующий раз откроете Far, будут отображены заданные каталоги. Вы можете сделать то же самое перед завершением работы с Far, чтобы продолжить следующий сеанс работы с теми же каталогами.

3) И самое главное – при нажатии Ctrl + O окошки Far скрываются, и перед нами обычный командный интерпретатор, причем мы уже находимся в нужном нам каталоге.

Довольно полезная информация по Far:

http://www.fcenter.ru/online.shtml?articles/software/utilities/7082

Zend Tool

Вообще-то я использую `zftool.phar` только для создания новых модулей. Но, как показывает практика, это не всегда работает. То есть, недавно я скачал свежую версию, и… она просто не заработала. Хотя та версия, что я скачивал месяц назад – создает новые модули без всяких проблем. Поэтому если вы столкнетесь с ситуацией, когда на этой неделе Zend Tool не работает ☺, воспользуйтесь следующей возможностью. Существует не только ZendSkeletonApplication, но и ZendSkeletonModule, т.е. базовый модуль, который можно также создать и с помощью `zftool.phar`, его можно скачать по этой ссылке

https://github.com/zendframework/ZendSkeletonModule/archive/master.zip

II. Простое управление доступом с помощью ACL

Организовать простое управление доступом только с помощью компонентов фреймворка можно очень легко. Параллельно мы рассмотрим основы использования менеджера Far для быстрой разработки приложений.

Начнем с того, что такое ACL. ACL в Zend Framework 2 представляет собой гибкую систему контроля доступа на основе четырех базовых понятий:

- `роли (любое произвольное имя)`
- `ресурсы (любое произвольное имя)`
- `привилегии (не обязательно, любое произвольное имя)`
- `разрешения (allow или deny)`

Человеку, не знакомому с предметом, объяснение вроде "ресурсом может быть все, что угодно" не даст ровным счетом ничего. Поэтому мы зайдем с другой стороны. Фактически, объект Zend\Permissions\Acl содержит некоторые списки. Во-первых, это список ролей, зарегистрированных в системе. Здесь все понятно, это может быть "гость", "зарегистрированный пользователь", "администратор" и т.д. Во-вторых, это список ресурсов. Здесь несколько сложнее. В качестве ресурса для ACL мы задаем произвольную

абстрактную строку, вроде "контроллер администратора Василия". Как же ACL определяет что это значит? А он и не определяет. Что на самом деле значит это название ресурса, знаем только мы сами. Задача ACL довольно проста – он должен просто зарегистрировать имя ресурса и разрешения на доступ к этому ресурсу для определенной роли (guest, user, admin, и т.д.). Разрешения бывают всего двух типов – "разрешить" (allow) или "запретить" (deny). Конечно, мы можем назначить в качестве имени ресурса любую абракадабру, но нам ведь нужно совсем не это, не так ли? Все будет зависеть от того, как именно вы будете организовывать проверку доступа в своей системе. Например, в качестве ресурсов вы можете задавать в своей системе маршруты (routes). Или имена контроллеров. А затем вы назначите для ресурса "AdminController" разрешение `allow` для администратора и `deny` для всех остальных. Но ACL не будет определять, обратились ли вы к какому-либо контроллеру, или нет. Более того, ACL совершенно не в курсе, пользователь с какой именно ролью сейчас обращается к системе. Задача ACL сводится к тому, чтобы содержать в себе все эти списки разрешений, ресурсов и ролей и при обращении к методу

```
$access = $acl->isAllowed($roleName, $resourceName);
```

возвращать `false` или `true`, в зависимости от того, что вы перед этим внесли в ACL самостоятельно (вручную, или с помощью конфигурационных файлов, или из базы данных и т.д.). Т.е., наш код, управляющий доступом, должен самостоятельно определить, доступ к какому ресурсу сейчас осуществляется и пользователь с какой ролью хочет получить доступ к этому ресурсу. И только тогда наш код, управляющий доступом, обращается к методу ACL `isAllowed()`, чтобы узнать, можно ли пользователю предоставлять доступ, или нет. И именно наш код, а не ACL должен принять соответствующие меры, если в доступе к ресурсу пользователю нужно отказать.

Если вы все еще не поняли, что такое ACL, не переживайте. Мы рассмотрим этот механизм очень подробно. На самом деле реализация этого механизма с помощью ZF2 занимает не более 5-7 минут.

Создание приложения

Даже если вы знаете, как это делается, но работаете под Windows, эти главы обязательны к прочтению!

Создание приложения на Zend Framework 2 начинается с установки каркаса. Каркас содержит начальный код для нашего приложения и называется ZendSkeletonApplication. Его можно скачать с GitHub.

Перейдите по адресу https://github.com/zendframework/ZendSkeletonApplication. В правой части веб-страницы GitHub вы увидите колонку со списком опций. Внизу этой колонки находится кнопка "Download ZIP". Еще недавно она называлась "ZIP" и находилась вверху слева. Нажмите на эту кнопку и загрузите архив. Он будет называться ZendSkeletonApplication-master.zip. Распакуйте архив. Мы неоднократно будем использовать полученный каталог, поэтому нам следует работать только с его копией.

Скопируйте приложение с помощью менеджера Far (по F5) в каталог C:\xampp\htdocs. После нажатия F5 в появившемся окне копирование щелкните на поле "Копировать в:" и после последней косой черты допишите новое имя каталога – "SimpleAclApp". Так мы сразу переименуем каталог и назовем наше приложение "SimpleAclApp".

Теперь у нас есть так называемый "код инициализации", но для его работы необходимо еще установить саму библиотеку Zend Framework 2. Перейдите в Far в корень каталога SimpleAclApp и выполните команду:

```
1 $ php composer.phar self-update && php composer.phar install
```

Хочу напомнить, что в Far переключение в режим командного интерпретатора выполняется комбинацией клавиш Ctrl + O.

Composer установит библиотеку Zend Framework 2 в каталог vendor\zendframework. SimpleAclApp уже содержит один модуль "из коробки", это модуль Application. Но мы создадим собственный новый модуль. Скачайте инструментарий Zend Tool, файл zftool.phar, по адресу:

https://packages.zendframework.com/zftool.phar

и переместите его в корень каталога SimpleAclApp. Мы будем использовать Zend Tool для создания новых модулей.

Убедитесь, что вы находитесь в каталоге SimpleAclApp и перейдите в режим интерпретатора (Ctrl + O). Выполните команду:

```
1 $ php zftool.phar create module TestAclModule
```

Мы создали и подключили новый модуль TestAclModule.

Установка и настройка виртуального хоста

Откройте каталог C:\windows\system32\drivers\etc. Far позволяет создавать ссылки на папки с помощью комбинации клавиш Ctrl + Shift + {любая цифровая клавиша вверху клавиатуры} либо с помощью Команды - > Ссылки на папки. Создайте ссылку на этот каталог. Переходить по ссылке затем можно будет с помощью **правого Ctrl** + {любая цифровая клавиша вверху клавиатуры}. Не забывайте, что после каждой настройки Far необходимо сохранять параметры по Shift + F9.

Нам необходимо отредактировать файл hosts. Для редактирования в Far перейдите к этому файлу в списке файлов и нажмите F4. В файл hosts допишем следующие строки:

```
127.0.0.1 localhost.simpleacl
```

Так мы переадресовываем все запросы из браузера к адресу localhost.simpleacl на так называемую "локальную петлю" – 127.0.0.1, где этот запрос уже будет ждать наш локальный сервер на 80 порту.

Сохраните изменения (F2) и выйдите из режима редактирования (снова F4).

Теперь нам нужно добавить виртуальный хост. Нужно сказать, что все эти операции выполняются практически одинаково как под Window, так и под Unix, разница, в основном, в путях к файлам и названиям.

> *Примечание*
>
> *Пользователям Debian/Ubuntu, например, нужно будет сначала создать файл конфигурации виртуального хоста, например, `localhost.simpleacl.conf`, в каталоге `/etc/apache2/sites-available`, а также создать символическую ссылку на него в каталоге `/etc/apache2/sites-enabled`:*
>
> ```
> # ln -L /etc/apache2/sites-available/localhost.simpleacl.conf /etc/apache2/sites-enabled/localhost.simpleacl.conf
> ```

Перейдите в каталог `C:\xampp\apache\conf\extra`. Создайте в Far ссылку на этот каталог. Откройте для редактирования файл `httpd-vhosts.conf`. Убедитесь, что строка `NameVirtualHost *:80` в этом файле раскомментирована. Добавьте конфигурацию нового виртуального хоста:

```
<VirtualHost *:80>
   DocumentRoot "C:/xampp/htdocs/SimpleAclApp/public"
   ServerName localhost.simpleacl
   <Directory "C:/xampp/htdoc/SimpleAclApp/public>
      AllowOverride All
      Order deny,allow
      Deny from all
      Allow from localhost
   </Directory>
   DirectoryIndex index.html index.php
</VirtualHost>
```

Откройте панель управления XAMPP и нажмите кнопу "Start" напротив надписи "Apache". Так вы запустите web-сервер Apache. Теперь при обращении из браузера к адресу `http://localhost.simpleacl` мы увидим страницу приветствия Zend Framework 2.

Базовая функциональность модуля

Поскольку мы создали наш новый модуль TestAclModule с помощью Zend Tool, он уже подключен к остальным модулям нашего приложения. В этом можно убедиться, открыв файл `SimpleAclApp\config\application.config.php`. В разделе `modules` мы увидим сейчас два модуля – модуль "из коробки" Application и созданный нами модуль TestAclModule. Zend Tool выполнил эту работу за нас.

В корне каждого модуля находится файл Module.php, класс Module которого отвечает за конфигурацию модуля. Помимо необходимой структуры каталогов, Zend Tool создал этот

файл и указал в нем базовую конфигурацию для автозагрузки в методе `getAutoloaderConfig()` и подключил конфигурацию самого модуля в методе `getConfig()`. Подобное подключение файлов с отдельными элементами конфигурации методом `include` позволяет сохранять читаемость класса Module при наличии в нем множества методов и параметров. Таким образом, метод getConfig() просто возвращает содержимое файла `TestAclModule\config\module.config.php` Пока что содержимым этого файла является пустой массив:

```
1 <?php
2 return array();
```

но по мере работы мы будем заполнять конфигурацию модуля с помощью этого массива.

Создание контроллеров

Для демонстрации работы ACL нам понадобятся два контроллера – Frontend для гостей сайта и Backend для администратора. Действие по умолчанию для этих контроллеров, `indexAction`, будет просто возвращать экземпляр ViewModel для того, чтобы вернуть в браузер простейший html-шаблон.

Создадим в каталоге

`SimpleAclApp\module\TestAclModule\src\TestAclModule\Controller`

два контроллера – `FrontendController` и `BackendController`:

```
1  <?php
2
3  namespace TestAclModule\Controller;
4
5  use Zend\Mvc\Controller\AbstractActionController;
6  use Zend\View\Model\ViewModel;
7
8  class FrontendController extends AbstractActionController
9  {
10    public function indexAction()
11    {
12        return new ViewModel();
13    }
14 }
```

```
1  <?php
2
3  namespace TestAclModule\Controller;
4
5  use Zend\Mvc\Controller\AbstractActionController;
6  use Zend\View\Model\ViewModel;
```

```
7
8  class BackendController extends AbstractActionController
9  {
10    public function indexAction()
11    {
12        return new ViewModel();
13    }
14 }
```

Кроме того, нам необходимо указать конфигурацию контроллеров в классе Module нашего модуля. Для этого необходимо создать метод `getControllerConfig()`:

```
1 public function getControllerConfig()
2 {
3    return include __DIR__ . '/config/controllers.config.php';
4 }
```

а также сам файл `TestAclModule\config\controllers.config.php` со следующим содержимым:

```
1 return array(
2    'invokables' => array(
3        'TestAclModule\Controller\Frontend'  => ' TestAclModule\Controller\FrontendController',
4        ' TestAclModule\Controller\Backend'  => ' TestAclModule\Controller\BackendController',
5    ),
6 );
```

Раздел "invokables" указывает сервис-менеджеру фреймворка (ServiceManagar), экземпляр какого класса создавать в случае необходимости.

Создание шаблонов

Создадим для каждого контроллера собственный шаблон представления в каталоге `SimpleAclApp\module\TestAclModule\view\test-acl-module`. Как видим, последний каталог, `test-acl-module`, соответствует пространству имен модуля, но "ВерблюжьяНотация" преобразуется для шаблонов представления в "нотацию-через-дефис". Традиционно в Zend Framework расширением для шаблонов представления является *.phtml, что говорит о том, что шаблон может содержать как html-разметку, так и php-код. Поскольку в одном контроллере может быть несколько действий, нужно создать каталог, соответствующий имени контроллера, а в нем будут находиться шаблоны с именем, соответствующим имени действия. Таким образом, в каталоге `SimpleAclApp\module\TestAclModule\view` у нас будет следующая структура:

```
view/
    test-acl-module/
        frontend/
            index.phtml
```

```
backend/
    index.phtml
```

Содержимое файла `frontend\index.phtml`:

```
1 <h1>Hello, Guest!</h1>
```

Содержимое файла backend\index.phtml:

```
1 <h1>Hello, Admin!</h1>
```

Теперь нам нужно указать конфигурацию `view_manager`, чтобы фреймворк знал, где искать шаблоны для нашего модуля (в каталоге `view`, конечно же). Параметры `view_manager` и маршруты указываются в классе Module нашего модуля, в методе `getConfig()`. Но поскольку мы в этом методе просто подключаем содержимое файла `TestAclModule\config\module.config.php`, то указывать конфигурацию `view_manager` мы будем именно в этом файле:

```
1 <?php
2 return array(
3    'view_manager' => array(
4        'template_path_stack' => array(
5            __DIR__ . '/../view',
6        ),
7    ),
8 );
```

Создание маршрутов

Давайте сейчас немного отвлечемся от нашего модуля TestAclModule, и рассмотрим модуль Application, который поставляется вместе с ZendSkeletonApplication. Сейчас нас интересует файл Application\config\module.config.php, в котором содержится основная конфигурация Application, в том числе и определения маршрутов для этого модуля. В этом файле конфигурации мы можем видеть следующее:

```
1  'router' => array(
2      'routes' => array(
3          'home' => array(
4              'type' => 'Zend\Mvc\Router\Http\Literal',
5              'options' => array(
6                  'route'    => '/',
7                  'defaults' => array(
8                      'controller' => 'Application\Controller\Index',
9                      'action'     => 'index',
10                 ),
11             ),
12         ),
13 // [..]
```

Раздел routes содержит несколько маршрутов для приложения, но нам интересен сейчас маршрут home. Это маршрут типа Literal, который используется для буквального соответствия набранного в браузере Url конкретному шаблону route (а в данном случае это '/'), который указан в параметрах (options) маршрута (в данном случае маршрута home). Конечно же, маршрут можно было назвать не home, а как-нибудь еще, скажем, frontend. Имя маршрута выбирается произвольно и нужно только для того, чтобы мы могли впоследствии при необходимости обратиться к данному маршруту по имени и получить его параметры из того же, скажем, сценария представления index.phtml.

Почему нас заинтересовал данный маршрут? Дело в том, что маршрут '/' означает обращение к корню сайта, поскольку маршруты не должны зависеть от конкретного имени сайта, то они представляют собой относительный адрес, т.е. все то, что мы набираем в браузере после абсолютного адреса http://localhost.simpleacl/. Но мы хотели бы, чтобы обращение к корню сайта вызывало не IndexController модуля Application, а FrontendController нашего модуля TestAclModule, точнее, его действие indexAction. Мы можем воспользоваться здесь тем фактом, что конфигурации модулей при загрузке приложений на Zend Framework 2 считываются последовательно, в том порядке, в каком они указаны в файле SimpleAclApp\config\application.config.php, в разделе modules. Поэтому нам нужно всегда учитывать этот порядок. Поскольку наш модуль указан в этом разделе после модуля Application, мы можем перезаписать конфигурацию для маршрута home своими собственными значениями, что мы, собственно, и сделаем для контроллера FrontendController. Для контроллера BackendController создадим маршрут admin. Укажем в файле TestAclModule\config\module.config.php конфигурацию маршрутов:

```
1  <?php
2
3  return array(
4      'router' => array(
5          'routes' => array(
6              'home' => array(
7                  'type' => 'Zend\Mvc\Router\Http\Literal',
8                  'options' => array(
9                      'route'    => '/',
10                     'defaults' => array(
11                         'controller' => 'TestAclModule\Controller\Frontend',
12                         'action'     => 'index',
13                     ),
14                 ),
15             ),
16             'admin' => array(
17                 'type' => 'Zend\Mvc\Router\Http\Literal',
18                 'options' => array(
19                     'route' => '/admin',
20                     'defaults' => array(
21                         'controller' => 'TestAclModule\Controller\Backend',
```

```
22                        'action' => 'index',
23                    ),
24                ),
25            ),
26        ),
27    ),
28    'view_manager' => array(
29        'template_path_stack' => array(
30            __DIR__ . '/../view',
31        ),
32    ),
33 );
```

Проверка работоспособности модуля

Если вы все сделали правильно, то теперь при обращении из браузера к адресу:

```
http://localhost.simpleacl
```

вы должны увидеть вместо приветствия Zend Framework 2 надпись:

Hello, Guest!

При обращении к адресу:

```
http://localhost.simpleacl/admin
```

вы должны увидеть надпись:

Hello, Admin!

Учим модуль работать с событиями

Хотелось бы напомнить, что в основе Zend Framework 2 лежит событийно-ориентированная архитектура. Для понимания того, что это такое, давайте рассмотрим этот вопрос несколько иначе. Поставим вопрос так: зачем это нужно? В ходе работы приложения на Zend Framework 2 иногда может понадобиться изменить поведение приложения, добавив в него пользовательский код. Например, сейчас перед нами стоит вполне конкретная задача – нам нужно проверить, имеет ли право пользователь обратиться по маршруту http://localhost.simpleacl/admin. Естественно, что такая проверка должна быть выполнена до того, как сработает контроллер, заданный для этого маршрута. Как это сделать, не переписывая код библиотеки Zend Framework 2 и не добавляя туда свой собственный код? Вполне очевидно, что работу всего приложения можно было бы разбить на какие-то именованные этапы и создать некоторый менеджер, который может присоединить наш произвольно выбранный код к такому логическому этапу работы приложения. Давайте назовем эти этапы работы событиями, а некий абстрактный менеджер – менеджером событий (EventManager), и нам сразу станет понятно, что это такое и зачем оно нужно. Конечно же, такое объяснение не является адекватным описанием принципов

событийно-ориентированной архитектуры, но зато значительно облегчает практическое использование этих принципов.

Таким образом, в коде фреймворка "зашиты" конкретные именованные этапы работы приложения, и фреймворк просто в определенной последовательности инициирует эти события, чтобы поэтапно выполнить всю работу. Чтобы подключить свой код к какому-либо событию (иначе говоря, добавить его в список обработчиков или, по-другому, слушателей какого-либо события), нужно в метод `attach` менеджера событий передать первым параметром имя события (т.е. того этапа работы приложения, на котором должен быть выполнен наш произвольный код), а вторым параметром указать, какой именно код должен быть выполнен.

Давайте рассмотрим все это на примере. Добавим в класс Module нашего модуля метод `onBootstrap()`. Если в классе Module любого модуля присутствует этот метод, он всегда автоматически вызывается фреймворком для установки параметров начальной загрузки модуля. В системе событий фреймворка это произойдет до того, как сработает какой-либо контроллер. Давайте вспомним, какие именно события (или этапы обработки ☺) существуют в целом для любого MVC-приложения на Zend Framework 2 (не вчитывайтесь в детали, просто смотрите на названия и порядок событий ☺):

- Zend\Mvc\Application: bootstrap (начальная загрузка) – выполняется при запуске приложения.
- Zend\Mvc\Application: route (маршрут) – происходит при определении маршрута, заданного пользователем из браузера и сопоставлении этому маршруту реально запрограммированных в приложении маршрутов и их параметров.
- Zend\Mvc\Application: dispatch (диспетчеризация) – запуск событий Zend\Mvc для выполнения диспетчеризации.
    - Zend\Mvc\Controller\ActionController: dispatch (диспетчеризация контроллера) – событие происходит для контроллера, наследующего классу ActionController. Выполняется вызов контроллера и его действия, которые были определены на этапе маршрутизации.

- Zend\Mvc\Application: render (визуализация) – запуск событий Zend\View для выполнения визуализации.
    - Zend\View\View: renderer – визуализация макетов и шаблонов.
    - Zend\View\View: response – внесение результатов визуализации в объект Response.
- Zend\Mvc\Application: finish (завершение) – завершение работы приложения, в результате которого пользователю возвращается экземпляр ответа (Response).

Как видим, существует основная "событийная линия" mvc-приложения Zend\Mvc\Application и для некоторых событий производная, как бы дочерняя. Нам остается всего лишь выбрать точку, к которой мы хотим подключить наш обработчик-

слушатель. Очевидно, что перед событием dispatch, в результате которого будет вызвано действие какого-либо контроллера произойдет событие route. После выполнения всех обработчиков-слушателей, зарегистрированных для события route приложение будет уже располагать данными о том, какой маршрут был запрошен пользователем и какой контроллер должен быть сопоставлен этому маршруту. Но чтобы еще более упростить наш пример, давайте добавим в класс Module нашего модуля следующий код:

```
1  use Zend\Debug\Debug;
2  public function onBootstrap($e)
3  {
4      $e->getApplication()
5         ->getEventManager()
6         ->attach(
7             'route', array($this, 'checkAccess')
8         );
9  }
10
11 public function checkAccess($e)
12 {
13     Debug::dump("Произошло событие route");
14 }
```

В метод onBootstrap(), как и в каждый слушатель (в дальнейшем мы будем использовать этот термин вместо "обработчик") события передается экземпляр события $e, с помощью которого мы можем получить менеджер событий Event Manager и снова назначить какой-либо обработчик для последующих событий с помощью метода attach. Поскольку метод onBootstrap() класса Module нашего модуля будет вызван фреймворком автоматически, мы указали менеджеру событий запустить слушатель checkAccess() при наступлении события route.

Теперь перед запуском любого контроллера произойдет вызов метода checkAccess() класса Module нашего модуля, поскольку этот метод зарегистрирован, как слушатель для события route, а это событие предшествует диспетчеризации.

Если мы обратимся в браузере к адресам:

http://localhost.simpleacl

http://localhost.simpleacl/admin

то мы увидим отладочную информацию, выведенную с помощью Zend\Debug.

Объект RouteMatch

Согласно официальной документации Zend Framework 2, под маршрутизацией (роутингом) понимается процесс сопоставления запроса с заданным контроллером. Обычно в

результате маршрутизации определяются один или несколько параметров. Чтобы различные объекты приложения могли легко запрашивать эти параметры, все они инкапсулируются в объект RouteMatch.

Давайте посмотрим, что представляет из себя объект RouteMatch. Измените метод `checkAccess()` класса Module следующим образом:

```
1 public function checkAccess($e)
2 {
3     $route = $e->getRouteMatch();
4     Debug::dump($route,'Объект RouteMatch:');
5 }
```

В результате для адреса http://localhost.simpleacl мы увидим следующее:

```
object(Zend\Mvc\Router\Http\RouteMatch)[154]
  protected 'length' => int 1
  protected 'params' =>
    array (size=2)
      'controller' => string 'TestAclModule\Controller\Frontend' (length=33)
      'action' => string 'index' (length=5)
  protected 'matchedRouteName' => string 'home' (length=4)
```

Как видим, объект RouteMatch содержит в свойстве `matchedRouteName` имя маршрута, которое было сопоставлено заданному адресу и в свойстве `params` содержатся имена контроллера и действия, которые мы указали для данного маршрута.

Имя маршрута мы можем получить с помощью метода `getMatchedRouteName()`, а имя контроллера или действия с помощью метода `getParam(string $name, mixed $default)`, задав в качестве `$name` `'controller'` или `'action'` соответственно.

Стандартное создание списков контроля доступа (ACL)

На этом этапе мы уже должны для себя определить, что же будет выступать у нас в роли ресурса. Мы можем легко определить, по какому маршруту обратился пользователь и можем определить, какой контроллер нужно запустить, чтобы выполнить запрос. Что именно мы будем рассматривать в качестве ресурса для нашего приложения, зависит только от нас.

Допустим, мы будем рассматривать в качестве ресурса контроллер. Но ведь у контроллера могут быть совершенно различные методы, к одним из которых мы захотим разрешить доступ пользователю, а к другим запретить. Например, в контроллере FrontendController можно создать помимо метода indexAction() для гостей метод userAction() для зарегистрированных пользователей. Вполне естественно, что хотя эти два метода принадлежат одному контроллеру, мы не хотели бы открывать гостям доступ к методу userAction().

Давайте еще раз вспомним, какими понятиями оперирует ACL:

- роли (любое произвольное имя)
- ресурсы (любое произвольное имя)
- привилегии (не обязательно, любое произвольное имя)
- разрешения (allow или deny)

Мы уже знаем, что такое роли, понимаем, что такое ресурсы, можем догадаться, что такое разрешения. Но вот что такое привилегии? Перед тем, как ответить на этот вопрос, давайте рассмотрим, как стандартно назначить какие-либо правила для ACL:

```
1 use Zend\Permissions\Acl\Acl;
2
3 $acl = new Acl();
4 $acl->addRole('guest')
5     ->addRole('user')
6     ->addRole('admin');
7
8 $acl->addResource('TestAclModule\Controller\Frontend')
9     ->addResource('TestAclModule\Controller\Backend');

10 // $acl->deny('guest', 'TestAclModule\Controller\Backend');

11 $acl->allow('guest', 'TestAclModule\Controller\Frontend', 'index')
12     ->allow('user',  'TestAclModule\Controller\Frontend')
13     ->allow('admin');
```

Как можно видеть из строк 3-10, Zend\Permissions\Acl позволяет задавать роли и ресурсы в виде строковых значений. Преобразование в объекты соответствующих типов будут выполнены вышеуказанными методами автоматически. Для того, чтобы разрешить или запретить доступ для какой-либо роли к какому-либо ресурсу используются методы `allow()` и `deny()` соответственно. Именно с помощью этих методов назначаются разрешения.

В строке 11 приведен пример разрешения привилегий. Первым параметром в метод `allow()` передается имя роли, для которой назначаются разрешения, в данном случае это роль `guest`. Вторым параметром указывается имя ресурса. Поскольку мы решили, что при проверке прав доступа в качестве ресурсов будут выступать имена контроллеров, мы передаем имя контроллера TestAclModule\Controller\Frontend в качестве ресурса. Третьим параметром указывается привилегии. Если несколько отвлечься от нашего примера, то в какой-либо другой системе в качестве ресурса может выступить, скажем, страница Page, а в качестве привилегий – read (чтение), edit (редактирование), delete (удаление) и т.д. В нашем случае мы можем в качестве привилегий указать действе контроллера (в данном случае это `index`), к которому разрешен доступ для роли `guest`.

В строке 12 мы разрешаем доступ зарегистрированному пользователю к ресурсу TestAclModule\Controller\Frontend. Поскольку мы не указываем какие-либо конкретные привилегии, это означает, что пользователь будут доступны любые привилегии для указанного ресурса. В нашем конкретном случае, когда в качестве привилегий мы используем названия действий, это будет означать, что пользователю `user` будет разрешено выполнение любых действий в контроллере Frontend.

В строке 13 мы указываем разрешения для пользователя `admin`. Поскольку мы не указываем никаких ресурсов и привилегий, это будет означать, что пользователь с ролью admin должен получить беспрепятственный доступ к *любому ресурсу* приложения.

Пример в строке 10 является абстрактным примером того, как можно запретить доступ к какому-либо ресурсу.

Zend\Permissions\Acl\Acl поддерживает наследование ролей. Например, мы хотим добавить в систему пользователя `author`, которому разрешено все то же, что и пользователю `user`, плюс к этому создание и редактирование тематических статей. Мы можем наследовать эту роль от пользователя `user`, чтобы наследовались также и все разрешения пользователя `user`, а затем добавить пользователю `author` необходимые для работы с тематическими статьями разрешения. Для того, чтобы это осуществить, нужно при добавлении роли в ACL вторым параметром указать роль, от которой будет наследоваться наша новая роль author:

```
1 $acl->addRole('author','user');
```

Мало того, Zend\Permissions\Acl\Acl позволяет нам наследовать несколько ролей! Для этого вторым параметром в метод addRole() передается не строка, а массив, содержащий список ролей.

Более подробно эта тема освещена в переводах официальной документации по адресу: Тисовая Аллея, 4… Шучу, конечно: http://zf2.com.ua/doc/63

Добавляем в модуль инициализацию ACL

В предыдущем примере мы условно рассмотрели добавление к функциональности нашего модуля действия `userAction()` в контроллер FrontendController, которое могло бы расширить возможности модуля, предоставив доступ зарегистрированным пользователям. Но на практике мы не будем этого делать, чтобы сэкономить время и силы ☺.

Мы уже разобрались, как инициализировать ACL начальными параметрами стандартным образом, добавляя роли, ресурсы, привилегии и разрешения, давайте теперь попробуем автоматизировать этот процесс и передать в ACL параметры с помощью файла конфигурации.

Для начала создадим файл конфигурации. Воспользуемся возможностью автозагрузки фреймворком конфигураций, находящихся в каталоге любого приложения `\config\autoload\`. Создадим в этом каталоге файл с именем `testaclmodule.global.php`. Имя файла отвечает требованиям ZF2. Первая часть имени файла соответствует имени

модуля, для которого эта конфигурация предназначена. Вторая часть имени, `global`, указывает, что это глобальная конфигурация. Здесь возможны, собственно, два варианта – либо `global`, либо `local`. Значения, указанные в файлах `local` имеют больший приоритет, поскольку считываются фреймворком только после того, как считаны все файлы `global`, и, соответственно, все указанные в `local` значения могут переопределить те, что указаны в `global`. Поэтому если какому-нибудь программисту, использующему наш модуль, понадобится конкретизировать какие-либо значения, он сможет переопределить их в файле `local`. Если же не понадобится – наличие файла `global` со значениями по умолчанию гарантирует работоспособность нашего модуля. Итак, создадим файл `testaclmodule.global.php` со следующим содержимым:

```
1 <?php
2 return array('testaclmodule' => array(
3     'allow' => array(
4         array(array('guest'), 'TestAclModule\Controller\Frontend'),
5         array(array('admin'), 'TestAclModule\Controller\Backend'),
5         array(array('admin'), null),
6     ),
7     'deny' => array(),
8 ));
```

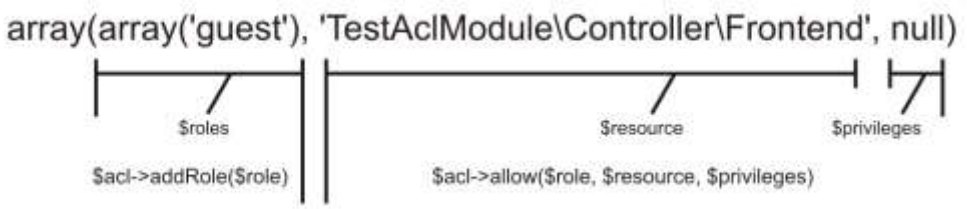
Как это работает? В глобальную конфигурацию добавляется ключ с именем testaclmodule, который содержит все наши параметры. Конфигурация разделена на две секции – для метода ACL `allow()` и для метода ACL `deny()`. Рассмотрим строку параметров:

```
array(array('guest'), 'TestAclModule\Controller\Frontend')
```

Эту строку можно представить для наглядности несколько по-другому:

```
array(array('guest'), 'TestAclModule\Controller\Frontend', null)
```

Схема, по которой это должно работать вполне очевидна:



Для чего я подробно останавливаюсь на этой схеме? Дело в том, что эта схема понадобится нам, когда мы будем рассматривать стандартный ZF2-модуль BjyAuthorize, конфигурация правил для которого организована подобным образом.

Но сейчас давайте вернемся к нашему модулю и добавим к классу Module метод `initAcl()` со следующим содержимым:

```
1  public function initAcl($e)
2  {
```

```
3       $this->acl = new Acl();
4       $config = $e->getApplication()
5                   ->getServiceManager()
6                   ->get('Config');
7       $params = $config['testaclmodule'];
8       if(array_key_exists('allow', $params)) {
9           $allow = $params['allow'];
10          foreach($allow as $rule) {
11              $roles      = $rule[0];
12              $resources  = isset($rule[1]) ? $rule[1] : null;
13              $privileges = isset($rule[2]) ? $rule[2] : null;
14              foreach($roles as $role) {
15                  if(!$this->acl->hasRole($role)) {
16                      $this->acl->addRole($role);
17                  }
18                  if(is_array($resources)) {
19                      foreach($resources as $resource) {
20                          if(!$this->acl->hasResource($resource)) {
21                              $this->acl->addResource($resource);
22                          }
23                      }
24                  } elseif(is_string($resources)) {
25                      if(!$this->acl->hasResource($resources)) {
26                          $this->acl->addResource($resources);
27                      }
28                  }
29                  $this->acl->allow($role, $resources, $privileges);
30              }
31          }
32      }
33  }
```

Чтобы упростить задачу, мы устанавливаем в этом методе только конфигурацию, указанную в секции `allow` нашего конфигурационного файла. При желании вы можете добавить код для установки правил из секции `deny` самостоятельно.

Как видим, в строке 3 метод `initAcl()` создает экземпляр класса Zend\Permissions\Acl\Acl и устанавливает его во внутреннюю переменную класса Module `$this->acl`. Кроме того, сам метод инициализации `initAcl()` необходимо вызвать из метода `onBootstrap()` нашего модуля, чтобы настройка ACL произошла в рамках начальной загрузки модуля. Таким образом, наш класс Module приобретает следующий вид:

```
1  <?php
2  namespace TestAclModule;
3
4  use Zend\Permissions\Acl\Acl;
5  use Zend\Debug\Debug;
6
7  class Module
```

```
8  {
9      protected $acl;
10
11     public function onBootstrap($e)
12     {
13         $this->initAcl($e);
14
15         $e->getApplication()
16           ->getEventManager()
17           ->attach(
18              'route', array($this,'checkAccess')
19           );
20     }
21     // [..]
22 }
```

Авторизация и аутентификация

Мы уже реализовали метод, который выполняет инициализацию ACL нужными правилами доступа, но необходимо еще реализовать метод `checkAccess()`, чтобы проверить, имеет ли пользователь право доступа к запрошенному ресурсу.

Проверка с помощью ACL выглядит следующим образом:

```
1 $access = $acl->isAllowed($role, $resource, $privilege);
```

Вполне очевидно, что при проверке нам необходимо указать роль пользователя, без которой вся эта проверка теряет смысл. Мы указали в конфигурации ACL, какие права установлены для ролей `guest` и `admin`, но у нас пока что нет никакого пользовательского интерфейса, с помощью которого администратор мог бы войти в систему, и система бы запомнила нашего администратора.

И здесь необходимо четко понимать, что проверка доступа всегда состоит из двух обязательных компонентов – аутентификации и авторизации.

Авторизация – это проверка, которую выполняет наш метод checkAccess() – имеет ли право пользователь с определенной ролью получать доступ к какому-либо ресурсу.

Аутентификация позволяет пользователю ввести логин и пароль, чтобы подтвердить свою роль в системе. Если пользователь укажет правильный пароль, механизм аутентификации определит надлежащую пользователю роль.

Один из этих механизмов теряет смысл при отсутствии другого. В дальнейшем мы рассмотрим два стандартных ZF2-модуля – ZfcUser и BjyAuthorize.

Модуль ZfcUser является модулем аутентификации. Он позволяет пользователю войти в систему с помощью своих учетных данных и пароля. С помощью механизма сессий ZF2

этот модуль определяет роль пользователя при каждом обращении к системе, т.е. идентифицирует пользователя, позволяя не вводить пароль каждый раз, когда пользователь хочет обратиться к защищенной странице нашего сайта. Но этот модуль не отвечает за авторизацию.

Модуль BjyAuthorize является модулем авторизации. В конфигурации данного модуля мы должны указать, какие роли, ресурсы и разрешения существуют в нашей системе и назначить необходимые нам права доступа. После этого модуль BjyAuthorize возьмет на себя всю работу по авторизации пользователей – при обращении к какому-либо ресурсу на основании указанной нами конфигурации этот модуль будет разрешать либо блокировать доступ к указанному ресурсу. Фактически, BjyAuthorize будет выполнять ту работу, которую (хотя и примитивно) уже на данный момент способен выполнять наш собственный модуль.

Но необходимо понимать, что каждый из этих модулей выполняет свою задачу, и пользуясь только одним из этих модулей, по определению невозможно организовать защиту нашей системы.

Имитируем аутентификацию с помощью сессий

Конечно же, нам понадобится какой-либо механизм аутентификации, но сейчас мы не будем реализовывать этот механизм, поскольку потребуется создание отдельных форм аутентификации, сущностей, работа с базой данных и т.д.

Вместо этого мы сымитируем этот механизм следующим образом: мы создадим в нашей системе два маршрута `/login` и `/logout`, при обращении к первому пользователь будет считаться администратором, как будто бы он уже ввел правильный пароль. При обращении к logout пользователь снова будет считаться гостем.

Для этого сначала давайте коротко рассмотрим механизм сессий. Php предоставляет механизм инициализации сессии `session_start()` и глобальный массив `$_SESSION`, при использовании которого можно сохранять данные в промежутке между запросами. Но этот механизм несколько несовершенен. Например, при активном использовании сторонних модулей мы можем столкнуться с ситуацией, когда некоторые значения, сохраненные в сессии одним модулем под определенным ключом, будут перезаписаны другим модулем. Что предлагает нам Zend Framework 2? Работа с сессиями происходит с помощью так называемых контейнеров:

```
use Zend\Session\Container;

$session = new Container('SomeKeyName');
$session->test = "This is test.";
```

Здесь мы создаем новый контейнер с именем SomeKeyName. Если мы уже создавали его ранее при предыдущих обращениях к нашей веб-странице, он будет содержать все внесенные нами ранее данные. В этом примере мы присваиваем некоторой переменной этого контейнера test строковое значение. Если переменная test до этого не существовала, она будет создана.

У такого подхода несколько преимуществ. Во-первых, если на момент создания контейнера сессия еще не стартовала, то фреймворк запустит сессию автоматически. Во-вторых, налицо объектно-ориентированный подход при работе с данными сессии. И самое главное, контейнер представляет собой отдельное пространство имен, создающееся прозрачно для разработчика в массиве $_SESSION, пространство имен, соответствующее имени контейнера.

Таким образом, нам остается только создать контроллер AuthenticationController с двумя действиями, loginAction() и logoutAction(), которые будут в контейнер сессии user заносить переменную role со значением admin, либо удалять эту переменную:

```
1  <?php
2
3  namespace TestAclModule\Controller;
4
5  use Zend\Mvc\Controller\AbstractActionController;
6  use Zend\Session\Container;
7
8  class AuthenticationController extends AbstractActionController
9  {
10     public function loginAction()
11     {
12         $session = new Container('user');
13         $session->role = 'admin';
14         $this->redirect()->toRoute('admin');
15     }
16
17     public function logoutAction()
18     {
19         $session = new Container('user');
20         if(isset($session->role)) {
21             unset($session->role);
22         }
23         $this->redirect()->toRoute('home');
24     }
25 }
```

В действии loginAction мы переадресуем пользователя на контроллер администратора, а в действии logoutAction – на главную страницу сайта.

Кроме того, нам необходимо еще указать данный контроллер в `controllers.config.php`, как `invokable`:

```
1  <?php
2
3  return array(
4      'invokables' => array(
5          'TestAclModule\Controller\Frontend' => 'TestAclModule\Controller\FrontendController',
6          'TestAclModule\Controller\Backend'  => 'TestAclModule\Controller\BackendController',
7          'TestAclModule\Controller\Authentication'  =>
8              'TestAclModule\Controller\AuthenticationController',
9      ),
10 );
```

и создать маршруты /login и /logout в module.config.php:

```
1  // [..]
2  'login' => array(
3      'type' => 'Zend\Mvc\Router\Http\Literal',
4      'options' => array(
5          'route' => '/login',
6          'defaults' => array(
7              'controller' => 'TestAclModule\Controller\Authentication',
8              'action' => 'login',
9          ),
10     ),
11 ),
12 'logout' => array(
13     'type' => 'Zend\Mvc\Router\Http\Literal',
14     'options' => array(
15         'route' => '/logout',
16         'defaults' => array(
17             'controller' => 'TestAclModule\Controller\Authentication',
18             'action' => 'logout',
19         ),
20     ),
21 ),
22 // [..]
```

Выполняем авторизацию

Давайте несколько изменим метод `checkAccess()` класса Module нашего модуля:

```
use Zend\Session\Container;
public function checkAccess($e)
{
    $route = $e->getRouteMatch();
    $session = new Container('user');
```

```
    $role = isset($session->role) ? $session->role : 'guest';
    Debug::dump($role,'Роль пользователя:');
}
```

Таким образом, к какому бы контроллеру мы не обращались, мы сможем в отладочной информации увидеть текущую роль пользователя. Вообще-то, следует оговориться, что в реальном проекте нельзя хранить роль пользователя в сессии хотя бы из соображений безопасности. Попробую пояснить.

Допустим, мы выложили наш сайт на некий хостинг. Допустим также, что найдется злоумышленник, который захочет наш сайт взломать. Поскольку все сессии PHP хранит в системе в одном и том же каталоге для всех хостов, то все, что остается сделать злоумышленнику – это создать свой сайт на том же хостинге, что и вы. Он обратится к вашему сайту, получит идентификатор сессии, изменит для cookie домен на свой собственный, а затем обратится к своему собственному скрипту, выполнив дамп сессии:

```
1 session_start();
2 var_dump($_SESSION);
```

После этого он будет располагать всеми переменными, находящимися в сессии. Если в этой переменной вдруг хранится роль пользователя, злоумышленнику ничего не стоит изменить это значение, а затем вновь обратиться к вашему сайту уже с повышенными привилегиями. Хотя, строго говоря, вам не поможет даже то, что вы будете хранить в сессии только идентификатор пользователя, поскольку, скорее всего, идентификатор администратора в вашей системе будет равен 1 ☺. Есть несколько вариантов решения данной проблемы. Например, Wordpress не использует сессии вообще и хранит всю необходимую информацию в cookie. Joomla хранит данные сессии в базе данных. В любом случае хранение сессии стандартным образом крайне небезопасно. Вы, конечно же, можете просто изменить путь для хранения сессии на произвольный каталог своего приложения. Давайте я попробую догадаться – это будет, скажем, каталог sessions в корне приложения или в каталоге data ☺. Т.е., это тоже не самое безопасное решение.

Но вернемся к нашему приложению. Давайте изменим метод `checkAccess()` следующим образом:

```
1  public function checkAccess($e)
2  {
3      $route = $e->getRouteMatch();
4      $session = new Container('user');
5      $role = isset($session->role) ? $session->role : 'guest';
6      if(!$this->acl->hasResource($route->getParam('controller'))) {
7          $access = false;
8      } else {
9          $access = $this->acl->isAllowed(
10             $role,
11             $route->getParam('controller'),
12             $route->getParam('action')
13         );
```

```
14      }
15      Debug::dump($access,'Assess:');
16      // if(!$access) { exit('Restricted Access!'); }
17  }
```

Как видим, в строках 6-8 мы предусмотрели ситуацию, когда пользователь обратится к контроллеру, который существует в системе, но не указан в правилах доступа в файле `testaclmodule.global.php`. Например, сейчас там не указан наш контроллер AuthenticationController, и если мы раскомментируем строку 16 и в браузере обратимся к адресу `http://localhost.simpleacl/login`, то произойдет экстренное завершение приложения с сообщением "Restricted Access!".

Как видим, нужно предусмотреть множество вариантов поведения системы в различных ситуациях. И здесь нам существенно облегчат задачу стандартные модули ZF2, которые мы рассмотрим в следующих разделах. Здесь же мы рассмотрели только общие принципы организации защиты доступа. Если бы мы создавали собственный модуль, то наверняка сделали бы конфигурацию более гибкой, логику авторизации вынесли бы из класса Module, где ей совсем не место, в отдельный сервис и т.д. Но мы не будем этого делать по нескольким причинам, и дело даже не в том, что уже существуют готовые решения. Дело в том, что, возможно, вы захотите предоставить свои модули другим программистам. И тогда им также придется перестраивать код своих модулей под вашу систему защиты доступа. Как уже где-то было сказано, одна из проблем ZF1 была в том, что на сегодняшний день существуют десятки тысяч несовместимых реализаций защиты доступа, которые нельзя повторно использовать в сторонних приложениях. Так давайте попытаемся избежать этой проблемы хотя бы в ZF2. С другой стороны, использование событийной модели в таком подходе обладает определенной универсальностью, поскольку остальной код приложения даже не догадывается о том, что где-то происходит проверка доступа. Решайте сами. Если вы чувствуете в себе силы создать что-то более гибкое, более мощное, универсальное и производительное, чем то, что предоставляют стандартные ZF2-модули – то все в ваших руках.

III. Стандартные модули – быстрый старт

Тестовое приложение

Давайте оставим наше предыдущее приложение без изменений, а для работы со стандартными модулями ZF2 создадим новое приложение Zf2Guard. Скопируйте каталог ZendSkeletonApplication в каталог C:\xampp\htdocs под новым именем Zf2Guard. Перейдите в этот каталог и выполните команду:

```
1 php composer.phar self-update && php.composer.phar install
```

чтобы установить библиотеку Zend Framework 2.

Скопируйте zftool.phar в каталог Zf2Guard, перейдите в этот каталог и создайте новый модуль TestGuardModule:

```
1 $ php zftool.phar create module TestGuardModule
```

Мы создали новый модуль, теперь создадим три контроллера – GuestController, UserController и AdminController. Их действия `indexAction()` будут просто отображать приветствия для гостя, зарегистрированного пользователя и администратора системы:

```
1  <?php
2  namespace TestGuardModule\Controller;
3  use Zend\Mvc\Controller\AbstractActionController;
4  use Zend\View\Model\ViewModel;
5
6  class GuestController extends AbstractActionController
7  {
8      public function initAction()
9      {
10         return new ViewModel();
11     }
12 }
```

```
1  <?php
2  namespace TestGuardModule\Controller;
3  use Zend\Mvc\Controller\AbstractActionController;
4  use Zend\View\Model\ViewModel;
5
6  class UserController extends AbstractActionController
7  {
8      public function initAction()
9      {
10         return new ViewModel();
11     }
```

```
12 }
```

```
1  <?php
2  namespace TestGuardModule\Controller;
3  use Zend\Mvc\Controller\AbstractActionController;
4  use Zend\View\Model\ViewModel;
5
6  class AdminController extends AbstractActionController
7  {
8      public function initAction()
9      {
10         return new ViewModel();
11     }
12 }
```

Мы должны также указать конфигурацию этих контроллеров. Добавим к классу Module нашего модуля метод getControllerConfig():

```
1 // [..]
2 public function getControllerConfig()
3 {
4     return include __DIR__ . '/config/controllers.config.php';
5 }
6 // [..]
```

а также создадим файл, подключаемый этим методом, со следующим содержимым:

```
1  <?php
2
3  return array(
4      'invokables' => array(
5          'TestGuardModule\Controller\Guest'
6              => 'TestGuardModule\Controller\GuestController',
7          'TestGuardModule\Controller\User'
8              => 'TestGuardModule\Controller\UserController',
9          'TestGuardModule\Controller\Admin'
10             => 'TestGuardModule\Controller\AdminController',
11     ),
12 );
```

Пропишем также маршруты `home` для гостя, `user` для пользователя и `admin` для администратора в файле `\TestGuardModule\config\module.config.php`. Сразу же в этом файле укажем и конфигурацию `view_manager` для шаблонов представления, чтобы в ближайшее время не возвращаться к этому файлу:

```
1  <?php
2  return array(
3      'router' => array(
4          'routes' => array(
5              'home' => array(
```

```
6                 'type' => 'Zend\Mvc\Router\Http\Literal',
7                 'options' => array(
8                     'route' => '/',
9                     'defaults' => array(
10                        'controller' => 'TestGuardModule\Controller\Guest',
11                        'action' => 'index',
12                    ),
13                ),
14            ),
15            'user' => array(
16                'type' => 'Zend\Mvc\Router\Http\Literal',
17                'options' => array(
18                    'route' => '/user',
19                    'defaults' => array(
20                        'controller' => 'TestGuardModule\Controller\User',
21                        'action' => 'index',
22                    ),
23                ),
24            ),
25            'admin' => array(
26                'type' => 'Zend\Mvc\Router\Http\Literal',
27                'options' => array(
28                    'route' => '/admin',
29                    'defaults' => array(
30                        'controller' => 'TestGuardModule\Controller\Admin',
31                        'action' => 'index',
32                    ),
33                ),
34            ),
35        ),
36    ),
37    'view_manager' => array(
38        'template_path_stack' => array(
39            __DIR__ . '/../view',
40        ),
41    ),
42 );
```

Создадим шаблоны представления для наших контроллеров. В каталоге \TestGuardModule\view создадим каталог, соответствующий пространству имен модуля в "нотации-через-дефис", а именно – каталог `test-guard-module` и в нем:

\guest\index.phtml

```
1 <h1>Hello, Guest!</h1>
```

\user\index.phtml

```
1 <h1>Hello, User!</h1>
```

\admin\index.phtml

```
1 <h1>Hello, Admin!</h1>
```

Добавим в файл hosts (как вы помните, в Far у нас уже есть ссылка на содержащую его папку) новый адрес:

```
127.0.0.1 localhost.zf2guard
```

и в файл `httpd-vhosts.conf` добавим виртуальный хост:

```
<VirtualHost *:80>
   DocumentRoot "C:/xampp/htdocs/Zf2Guard/public"
   ServerName localhost.zf2guard
   <Directory "C:/xampp/htdoc/Zf2Guard/public>
      AllowOverride All
      Order deny,allow
      Deny from all
      Allow from localhost
   </Directory>
   DirectoryIndex index.html index.php
</VirtualHost>
```

Теперь при обращении к адресам

- http://localhost.zf2guard
- http://localhost.zf2guard/user
- http://localhost.zf2guard/admin

мы должны увидеть приветствия для пользователей guest, user и admin соответственно.

ZfcUser

Как вы помните из предыдущего раздела, главы "Авторизация и аутентификация", модуль ZfcUser отвечает за аутентификацию (а по большому счету и за идентификацию, поскольку при каждом повторном обращении пользователя к нашему сайту идентифицирует его с помощью механизма сессий).

Загрузка модуля

Перейдите в Far в корневой каталог нашего приложения. Выполните команду

```
1 $ cls
```

под Windows или

```
1 $ clear
```

под Unix

для того, чтобы очистить вывод командного интерпретатора. Затем выполните команду:

```
1 $ php composer.phar require zf-commons/zfc-user:0.*
```

Мы увидим примерно следующее:

```
1 composer.json has been updated
2 Loading composer repositories with package information
3 Updated dependencies (including require-dev)
4    - Installing zf-commons/zfc-base (v0.1.2)
5      Loading from cache
6    - Installing zf-commons/zfc-user (0.1.2)
7      Loading from cache
8 Writing lock file
9 Generating autoload files
```

В строке 1 мы видим, что composer обновил файл `composer.json`. Вы можете убедиться в этом, выйдя в Far из режима командного интерпретатора, выбрав в списке файлов `composer.json` и нажав F3 (открытие файла только для чтения). Теперь, если мы через какое-то время решим обновить проект свежими версиями библиотек с помощью команды

```
1 $ php composer.phar self-update && php composer.phar update
```

при наличии более новых версий модуля ZfcUser, этот модуль будет обновлен.

В строке 2 composer сообщает о том, что он получает информацию о модуле из репозитория. В зависимостях модуля ZfcUser указан другой стандартный ZF2-модуль, ZfcBase. Как мы можем видеть в строке 4, предварительно выполняется загрузка этого модуля.

Если вы уже выполняли загрузку какого-либо компонента с помощью composer, и его версия в репозитории GitHub осталась прежней, composer не будет загружать данные из репозитория, а возьмет их из локального кеша. В Windows XP, например, этот кэш находится в папке `%userprofile%\Application Data\Composer`.

Подключение модуля

После того, как модуль будет загружен, нам необходимо внести модуль в конфигурацию приложения. Найдите в файле `Zf2Guard\config\application.config.php` раздел `modules` и измените его следующим образом:

```
'modules' => array(
    'Application',
    'TestGuardModule',
    'ZfcBase',
    'ZfcUser',
),
```

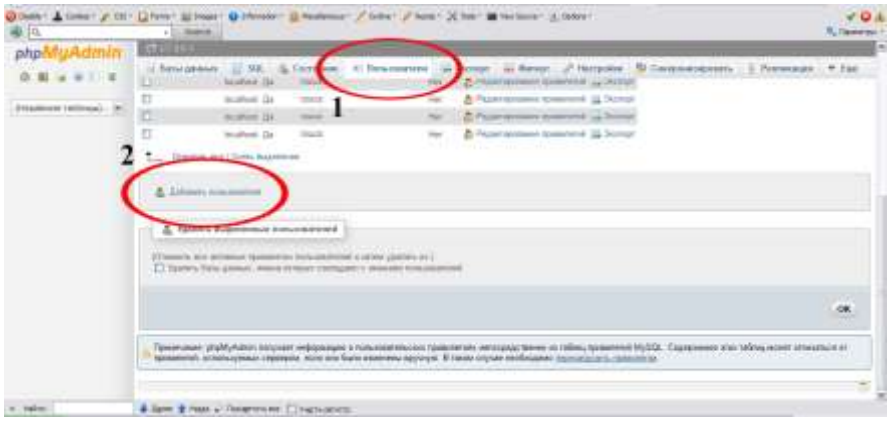
Мы добавляем в список модулей также и ZfcBase, чтобы выполнить зависимости модуля ZfcUser.

Создание базы данных и миграция модуля в БД

Следующим шагом будет создание базы данных для нашего проекта, поскольку ZfcUser хранит данные о пользователях в базе данных. Давайте сначала создадим базу данных для нашего проекта в MySql с помощью phpMyAdmin.

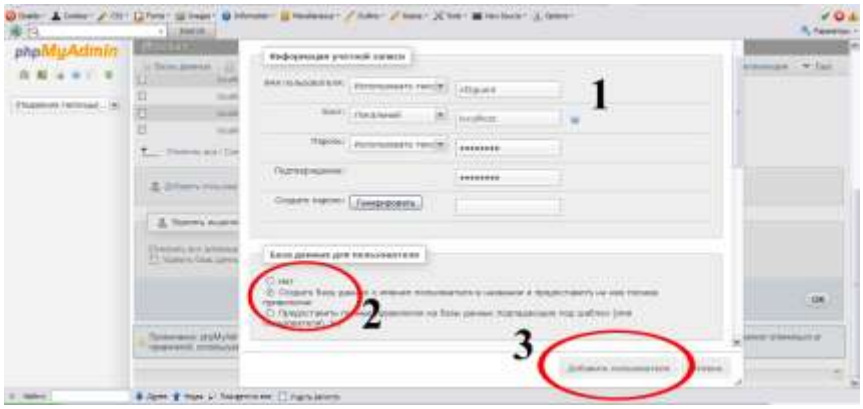
Откройте панель управления XAMPP и нажмите кнопку "Start" напротив надписи "MySQL". Так мы запустим MySql. Нажмите на кнопку "Admin" напротив надписи "MySQL". В браузере откроется phpMyAdmin.

В phpMyAdmin выберите вверху вкладку "Пользователи" (в более ранних версиях – "Привелегии"). Нажмите кнопку "Добавить пользователя".



- В поле "Имя пользователя" укажите zf2guard.
- Из выпадающего списка "Хост" выберите "Локальный".
- В качестве пароля я буду использовать zf2guard, поскольку предыдущий пункт все равно разрешает вход пользователя только с локального компьютера, а при активной работе с базами данных сложные пароли легко забываются.
- Подтвердите пароль в текстовом поле ниже.

Установите переключатель в положение "Создать базу данных с именем пользователя в названии и предоставить на нее полные привилегии". Подтвердите выбранные параметры, нажав на кнопку "Добавить пользователя".



В каталоге Zf2Guard\vendor\zf-commons\zfc-user\data находятся файлы для миграции модуля ZfcUser в базу данных. Файл можно установить либо с помощью phpMyAdmin, выбрав созданную нами базу данных zf2guard и перейдя на вкладку "Импорт", либо из командного интерпретатора.

Откройте в Far каталог Zf2Guard\vendor\zf-commons\zfc-user\data и перейдите в режим командного интерпретатора.

Запустите клиент mysql от имени пользователя zf2guard:

```
1 $ mysql -u zf2guard -pzf2guard
```

Не забывайте, что после ключа -p, в котором указывается пароль пользователя, не должно быть пробелов.

Выберите нашу базу данных и импортируйте данные:

```
1 $ USE zf2guard
2 $ source schema.mysql.sql
```

Теперь давайте посмотрим, что сейчас представляет из себя наша база данных. Выполните команду:

```
1 $ SHOW TABLES;
mysql> SHOW TABLES;
+--------------------+
| Tables_in_zf2guard |
+--------------------+
| user               |
+--------------------+
3 rows in set (0.00 sec)
```

Как видим, была создана таблица user. Посмотрим, что она из себя представляет. Выполните:

```
1 $ DESCRIBE user;
mysql> DESCRIBE user;
+--------------+----------------------+------+-----+---------+----------------+
| Field        | Type                 | Null | Key | Default | Extra          |
+--------------+----------------------+------+-----+---------+----------------+
| user_id      | int(10) unsigned     | NO   | PRI | NULL    | auto_increment |
| username     | varchar(255)         | YES  | UNI | NULL    |                |
| email        | varchar(255)         | YES  | UNI | NULL    |                |
| display_name | varchar(50)          | YES  |     | NULL    |                |
| password     | varchar(128)         | NO   |     | NULL    |                |
| state        | smallint(5) unsigned | YES  |     | NULL    |                |
+--------------+----------------------+------+-----+---------+----------------+
6 rows in set (0.06 sec)
```

В качестве первичного ключа задан столбец user_id с атрибутом AUTO_INCREMENT. Для столбцов `username` и `email` также существует индекс, что обеспечит быстрый поиск и выборку по этим столбцам при наличии большого количества данных в этой таблице.

Столбец `state` по умолчанию не используется, но, тем не менее, присутствует. Его можно использовать для установки к-л произвольного состояния, например, когда пользователь не подтвердил аккаунт посредством e-mail, или требует одобрения администратором, или если нужно не удалять пользователя из базы данных, а просто "забанить" и т.д. Это может вам не понадобиться вообще, в таком случае можно спокойно игнорировать этот параметр.

В столбце `password` сохраняется пароль пользователя в зашифрованном виде, поэтому ни в коем случае нельзя редактировать его вручную. Для шифрования используется Zend\Crypt\Password\Bcrypt, внутренне использующий нативную функцию PHP crypt(), которая обеспечивает необратимое шифрование пароля (http://php.net/manual/ru/function.crypt.php). С одной стороны это довольно медленный алгоритм шифрования, но, с другой стороны, эта медлительность может защитить от атаки с помощью перебора (брутфорс). Кроме того, разработчики модуля утверждают, что параметры шифрования подобраны таким образом, чтобы найти оптимальное сочетание скорости и надежности и не создавать повышенную нагрузку на сервер, поэтому я не рекомендовал бы изменять параметры шифрования, установленные по умолчанию.

Учим приложение работать с базой данных

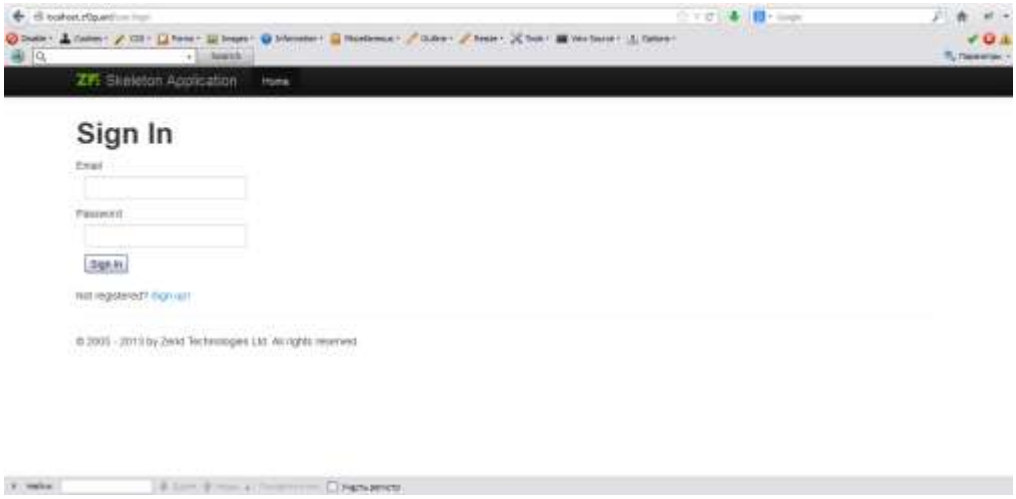
Все необходимые действия по установке модуля уже выполнены, но нужно еще настроить наше приложение, чтобы оно могло работать с базой данных.

Для этого необходимо выполнить два требования – указать сервис-менеджеру фреймворка, какой адаптер использовать для работы с базой данных и указать параметры соединения с базой данных, такие как имя пользователя, пароль и т.д

В каталоге `Zf2Guard\config\autoload` находится файл `local.php.dist`. Переименуем его в `local.php` и укажем в нем конфигурацию параметров соединения с базой данных:

```
<?php
return array(
    'service_manager' => array(
        'factories' => array(
            'Zend\Db\Adapter\Adapter' => 'Zend\Db\Adapter\AdapterServiceFactory',
        ),
    ),
    'db' => array(
        'driver'    => 'pdo',
        'dsn'       => 'mysql:dbname=zf2guard;host=localhost',
        'username'  => 'zf2guard',
        'password'  => 'zf2guard',
    ),
);
```

Базовая функциональность модуля



Теперь при обращении к адресу: `http://localhost.zf2guard/user` мы будем переадресованы на адрес `http://localhost.zf2guard/user/login`, где нам предложат либо ввести свои учетные данные, либо зарегистрироваться в системе.

По умолчанию для аутентификации ZfcUser использует e-mail пользователя и пароль, хотя у нас есть возможность использовать, например, логин для аутентификации вместо e-mail с помощью дополнительного конфигурирования модуля.

После установки модуля у нас доступно несколько дополнительных маршрутов. Начнем с того, что при обращении по адресу `http://localhost.zf2guard/user` уже не вызывается действие index контроллера UserController нашего собственного модуля TestGuardModule. Для пользователя, который прошел аутентификацию, т.е. ввел правильные учетные данные обращение по маршруту `/user` приводит к срабатыванию действия index контроллера UserController самого модуля ZfcUser. Причем, если даже в конфигурации приложения `ZF2Guard\config\application.config.php` мы укажем в разделе modules наш собственный модуль после ZfcUser, это никак не повлияет на поведение приложения. Это обусловлено тем, что в конфигурации модуля ZfcUser для маршрута `/user` указан очень высокий приоритет (строка 5):

```
1  'router' => array(
2      'routes' => array(
3          'zfcuser' => array(
4              'type' => 'Literal',
5              'priority' => 1000,
6              'options' => array(
7                  'route' => '/user',
8                  'defaults' => array(
9                      'controller' => 'zfcuser',
10                     'action'     => 'index',
11                 ),
12             ),
13 // [..]
```

Однако это никак не повлияет на маршруты, дочерние от маршрута /user. То есть, если мы в контроллере нашего собственного модуля TestGuardModule создадим действие someAction() для маршрута /user/some, то при обращении к адресу http://localhost.zf2guard/user/some все произойдет согласно обычной схеме – будет вызвано действие someAction() контроллера нашего модуля. Я подробно остановился на этом моменте, поскольку в официальной документации на сегодняшний день практически ничего не сказано о параметре priority для маршрутов.

На первый взгляд может показаться, что ZfcUser выполняет также авторизацию пользователей. При обращении к адресу http://localhost.zf2guard/user для пользователя, не прошедшего аутентификацию или вышедшего (logout) из системы будет запрошен пароль, что говорит о том, что система проверяет, имеет ли право пользователь на доступ к запрошенному ресурсу. Хотя с академической точки зрения это действительно элемент авторизации, а не аутентификации, смысл его несколько иной. Если мы обратились бы сейчас к некоему дочернему от /user маршруту /user/some, то могли бы увидеть, что в данном случае пароль не запрашивается. Т.е. подобное поведение справедливо только для одного маршрута – маршрута /user. Проверка реализована в методе indexAction контроллера UserController модуля ZfcUser:

```
1 public function indexAction()
2 {
3     if (!$this->zfcUserAuthentication()->hasIdentity()) {
4         return $this->redirect()->toRoute(static::ROUTE_LOGIN);
5     }
6    return new ViewModel();
7 }
```

Как видим, в строке 3 происходит попытка идентификации пользователя. Контроллер обращается к плагину контроллера ZfcUserAuthentication, который в свою очередь использует компонент Zend Framework 2 Zend\Authentication. Если пользователя не удалось идентифицировать, выполняется его переадресация по маршруту user/login.

Нас такое поведение более, чем устраивает, однако мы хотели бы, чтобы после успешного ввода аутентификационных данных пользователь был бы перенаправлен на контроллер именно нашего модуля. ZfcUser предоставляет такую возможность, мы рассмотрим дополнительное конфигурирование модуля чуть позже.

После установки модуля ZfcUser у нас появляются дополнительно в системе следующие маршруты:

Имя маршрута в системе	Путь в браузере	Функциональность
zfcuser	/user	Родительский маршрут для работы с функциональностью пользователя. Пользователя, не прошедшего аутентификацию,

		переадресует на /user/login. Приоритет маршрута – 1000.
zfcuser/login	/user/login	Позволяет пользователю войти в систему, предоставляя форму входа, а также выполняет валидацию формы.
zfcuser/authenticate	/user/authenticate	Основная функция аутентификации, используется, в основном, самим модулем. Если необходим повторный вход с другими учетными данными, используйте /user/login.
zfcuser/logout	/user/logout	Позволяет вошедшему в систему пользователю выполнить выход из системы.
zfcuser/register	/user/register	Позволяет выполнить регистрацию новых пользователей.
zfcuser/changepassword	/user/change-password	Позволяет зарегистрированному пользователю изменить свой пароль.
zfcuser/changeemail	/user/change-email	Позволяет зарегистрированному пользователю изменить свой e-mail.

Дополнительное конфигурирование модуля

Хотя на данный момент установленный модуль уже обладает базовой функциональностью, существует возможность дополнительного конфигурирования.

У модуля ZfcUser есть несколько параметров, позволяющих быстро настроить основную функциональность. После установки ZfcUser, скопируйте `./vendor/zf-commons/zfc-user/config/zfcuser.global.php.dist` в `./config/autoload/zfcuser.global.php` [обратите внимание на то, что необходимо изменить расширение с `*.php.dist` на `*.php`] и измените значения по собственному усмотрению:

- `zend_db_adapter` – определяет DI-псевдоним (alias) для конфигурации адаптера к базе данных. По умолчанию – Zend\Db\Adapter\Adapter. [тема будет раскрыта в расширенном использовании]
- `user_entity_class` – имя используемого класса сущности. Позволяет использовать ваш собственный класс сущности вместо предоставляемого по умолчанию. Пользовательский класс должен реализовывать интерфейс

ZfcUser\Entity\UserInterface. По умолчанию – ZfcUser\Entity\User. [тема будет раскрыта в расширенном использовании]

- `enable_registration` – значение логического типа (boolean), определяет, разрешена ли регистрация новых пользователей. По умолчанию – `true`.
- `enable_username` – значение логического типа (boolean), включающее поле с именем пользователя в форму регистрации. Если для параметра `auth_identity_fields` будет установлено значение `array('email','username')`, это позволит пользователю войти в систему, используя свое имя пользователя ИЛИ адрес e-mail. По умолчанию – `false`.
- `auth_adapters` – адаптеры аутентификации. Определяет адаптеры, которые будут использованы для аутентификации пользователей. Допустимые значения – массив сервисов, реализующих интерфейс
  `ZfcUser\Authentication\Adapter\ChainableAdapter`
  Значение по умолчанию – `ZfcUser\Authentication\Adapter\Db` с приоритетом `100`. [тема будет раскрыта в расширенном использовании]
- `enable_display_name` – значение логического типа, определяющее, должно ли присутствовать поле display name (отображаемое имя пользователя, не путайте с полем username) в форме регистрации пользователей. По умолчанию – false.
- `auth_identity_fields` – массив значений (одно или несколько), определяющих, какое поле (или поля) будут использованы для идентификации пользователя при входе в систему. Значения будут проверены плагином Authentication в соответствующем указанному порядке. Допустимые значения – массив с одним или несколькими из: `email`, `username`. Значение по умолчанию – `email`.
- `login_form_timeout` – целое число, определяющее время ожидания в секундах для поля защиты от CSRF (подделки межсайтовых запросов) в форме входа пользователей в систему (login form). По умолчанию – 300 секунд. [На самом деле в той версии ZfcUser, которая, например, установлена в моем приложении это поле не добавляется в форму вообще. В классе ZfcUser\Form\Login следующие строки закомментированы:

```
// @todo: Fix this
// 1) getValidator() is a protected method
// 2) i don't believe the login form is actually being validated by the login action
// (but keep in mind we don't want to show invalid username vs invalid password or
// anything like that, it should just say "login failed" without any additional info)
//$csrf = new Element\Csrf('csrf');
//$csrf->getValidator()->setTimeout($options->getLoginFormTimeout());
//$this->add($csrf);
```

  Подобная же история и с формой ZfcUser\Form\Registration, наследующей класс ZfcUser\Form\Base. Рискну предположить, что данная функциональность находится еще в стадии разработки.]

- user_form_timeout – целое число, определяющее время ожидания в секундах для поля защиты от CSRF (подделки межсайтовых запросов) в форме регистрации пользователей. По умолчанию – 300 секунд. [см. пункт login_form_timeout]
- login_after_registration – значение логического типа (boolean), автоматически выполняет вход пользователя в систему (login) после регистрации. Значение по умолчанию – false.
- use_registration_form_captcha – значение логического типа, определяет, должна ли использоваться captcha в форме регистрации пользователя. Значение по умолчанию – false.
- form_captcha_options – в настоящее время используется в регистрационной форме, но может быть повторно использовано в любом месте. Используйте этот параметр, чтобы указать, какой адаптер Zend\Captcha использовать, а также передаваемые ему параметры. По умолчанию используется figlet:

```
'form_captcha_options' => array(
    'class'   => 'figlet',
    'options' => array(
        'wordLen'    => 5,
        'expiration' => 300,
        'timeout'    => 300,
    ),
),
```

- use_redirect_parameter_if_present – значение логического типа, проверяет параметр redirect, на который пользователь будет перенаправлен методом POST или GET после успешной аутентификации. Если этот параметр задан, он может переопределить параметр login_redirect_route, указанный ниже.
- user_login_widget_view_template – устанавливает заданный шаблон представления для виджета входа в систему (login). Виджет реализован с помощью помощников представления, поэтому можно не только установить необходимый шаблон виджета для модуля ZfcUser, но и использовать его в любом своем сценарии представления следующим образом:

```
<?php echo $this->ZfcUserLoginWidget(); ?>
```

  Значение по умолчанию – 'zfc-user/user/login.phtml'.
- login_redirect_route – строковое значение, имя маршрута в приложении, на который пользователь будет перенаправлен после успешной аутентификации. Значение по умолчанию – 'zfcuser'.
- logout_redirect_route – строковое значение, имя маршрута в приложении, на который пользователь будет перенаправлен после выхода (logout) из системы. По умолчанию – 'zfcuser/login'.
- password_cost – НЕ ИЗМЕНЯЙТЕ НАСТРОЙКИ ХЭША ПАРОЛЕЙ ПО УМОЛЧАНИЮ, если только а) вы не провели достаточно исследований и полностью понимаете, что именно вы хотите изменить. б) у вас есть очень веские причины отклониться от параметров по умолчанию и вы знаете, что делаете.

Значение должно быть целым числом от 4 до 31. Число представляет собой логарифм количества итераций по основанию 2, использующихся для хеширования По умолчанию это 14 (около 10 хешей в секунду на процессоре i5).

- `enable_user_state` – значение логического типа, включает использование параметра `state` для пользователя. Указывает, должно ли использоваться состояние (`state`) пользователя при регистрации и входе в систему (login). Смысл параметра описывался выше, при рассмотрении миграции модуля в базу данных.
- `default_user_state` – целое число, определяет состояние (state) пользователя по умолчанию после регистрации.
- `allowed_login_states` – массив значений, указывающих допустимые состояния (state), при которых пользователю будет разрешен вход в систему (login). Укажите null, если хотите позволить вход в систему для пользователей без состояния: 'allowed_login_states' => array(null, 1),
- `table_name` – имя таблицы пользователей в базе данных. По умолчанию – '`user`'.

ZendDeveloperTools

Как вы помните, вначале мы создали наш собственный модуль TestGuardModule и в нем несколько контроллеров. Если мы сейчас обратимся в браузере по заданным нами маршрутам для этих контроллеров, мы можем увидеть, как результат работы наших контроллеров, приветствия для гостя, пользователя и администратора. Но как нам определять при разработке, какая роль у нас сейчас в системе? Даже сейчас, когда в системе всего три контроллера? Давайте воспользуемся инструментом ZendDeveloperTools, который легко может решить эту задачу.

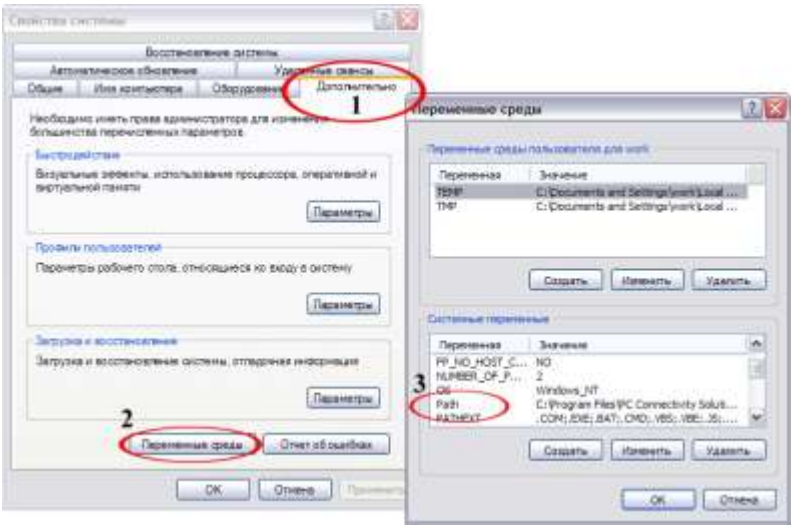
Однако прежде чем выполнить простую установку модуля, пользователям Windows придется выполнить еще ряд дополнительных действий.

Установка git и настройка Windows

Дело в том, что если мы попытаемся загрузить с помощью composer какой-либо пакет без тегов (а теги, в принципе, для git-пакетов не обязательны), то composer для установки этого пакета запросит git. В отличии от пользователей Unix и MacOS, на Windows git по умолчанию не установлен. Поэтому если вы работаете под Windows и еще не установили git на свой компьютер, то сейчас самое время это сделать, чтобы продолжить изучение данного материала. Сам git под Windows вы можете скачать по адресу: http://git-scm.com/download/win. Кроме того, чтобы composer мог вызывать git при необходимости, необходимо добавить в системную переменную Path следующий путь:

- для XP - C:\Program Files\Git\cmd
- для Windows 7 - C:\Program Files (x86)\Git\cmd

Напомню, что самый простой способ сделать это – открыть окно свойств компьютера (либо правой кнопкой мыши на "Мой компьютер" -> Свойства, либо Win + R и выполнить sysdm.cpl) и на вкладке "Дополнительно" нажать "Переменные среды".



Прокрутить список "Системные переменные" и дважды щелкнуть на "Path". Хотелось бы еще напомнить, что хотя значения переменных разделяются символом "точка с запятой" (;), но после последнего значения "точку с запятой" ставить нельзя.

Загрузка модулей ZendDeveloperTools и BjyProfiler

После установки git и настройки системных переменных, перейдите в Far в каталог нашего проекта Zf2Guard и переключитесь в режим командного интерпретатора (Ctrl + O). Для установки модуля ZendDeveloperTools выполните команду:

```
1 $ php composer.phar require zendframework/zend-developer-tools:dev-master
```

Если внимательно посмотреть на вывод результата выполнения команды, можно увидеть:

```
zendframework/zend-developer-tools suggest installing bjyoungblood/bjy-profiler …
```

Если мы поинтересуемся, что такого интересного предлагает (suggest) нам установить модуль ZendDeveloperTools, то увидим, что это модуль BjyProfiler, позволяющий нам профилировать запросы к базе данных. Давайте загрузим и этот модуль:

```
1 $ php composer.phar require bjyoungblood/bjy-profiler:dev-master
```

Подключение модулей ZendDeveloperTools и BjyProfiler

После того, как модули будут загружены, нам необходимо внести их в конфигурацию приложения. Найдите в файле `Zf2Guard\config\application.config.php` раздел `modules` и измените его следующим образом:

```
'modules' => array(
    'Application',
    'TestGuardModule',
    'ZendDeveloperTools',
    'BjyProfiler',
    'ZfcBase',
    'ZfcUser',
),
```

Кроме того, нужно скопировать файл конфигурации ZendDeveloperTools, который находится по следующему пути:

`vendor\zendframework\zend-developer-tools\config\zenddevelopertools.local.php.dist`

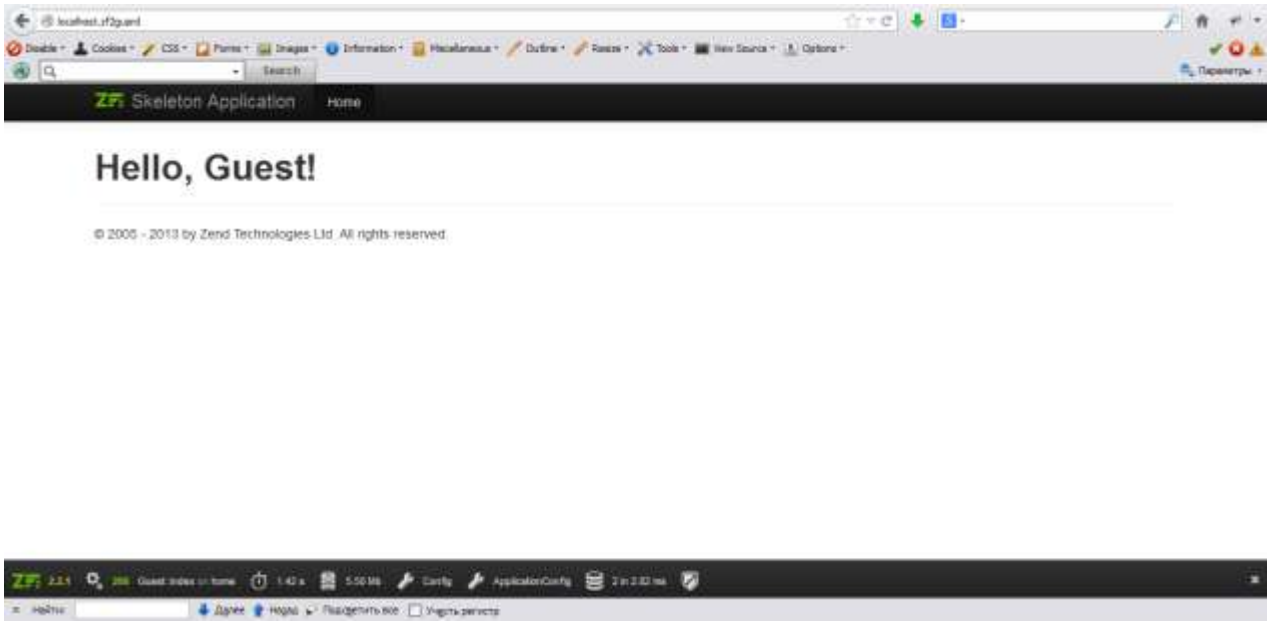
в каталог автозагрузки приложения `Zf2Guard\config\autoload`, изменив расширение `*.php.dist` на `*.php`.

В том же каталоге автозагрузки необходимо создать и файл конфигурации для модуля BjyProfiler `Zf2Guard\config\autoload\bjyauthorize.local.php` со следующим содержимым (для других проектов нам нужно будет менять только параметры подключения к базе данных):

```
<?php

$dbParams = array(
    'database'  => 'zf2guard',
    'username'  => 'zf2guard',
    'password'  => 'zf2guard',
    'hostname'  => 'localhost',
    // buffer_results - only for mysqli buffered queries, skip for others
    // 'options' => array('buffer_results' => true)
);

return array(
    'service_manager' => array(
        'factories' => array(
            'Zend\Db\Adapter\Adapter' => function ($sm) use ($dbParams) {
                $adapter = new BjyProfiler\Db\Adapter\ProfilingAdapter(array(
                    'driver'    => 'pdo',
                    'dsn'       => 'mysql:dbname='.$dbParams['database']
                                   .';host='.$dbParams['hostname'],
                    'database'  => $dbParams['database'],
                    'username'  => $dbParams['username'],
                    'password'  => $dbParams['password'],
                    'hostname'  => $dbParams['hostname'],
                ));

                if (php_sapi_name() == 'cli') {
                    $logger = new Zend\Log\Logger();
                    // write queries profiling info to stdout in CLI mode
                    $writer = new Zend\Log\Writer\Stream('php://output');
                    $logger->addWriter($writer, Zend\Log\Logger::DEBUG);
                    $adapter->setProfiler(new BjyProfiler\Db\Profiler\LoggingProfiler($logger));
                } else {
                    $adapter->setProfiler(new BjyProfiler\Db\Profiler\Profiler());
                }
                if (isset($dbParams['options']) && is_array($dbParams['options'])) {
                    $options = $dbParams['options'];
```

```
                } else {
                    $options = array();
                }
                $adapter->injectProfilingStatementPrototype($options);
                return $adapter;
            },
        ),
    ),
);
```

Теперь, при обращении к нашему приложению из браузера, мы должны увидеть внизу страницы панель ZendDeveloperTools.



К сожалению, пока мы не подключим модуль авторизации BjyAuthorize, значок щита (в правой части панели), отображающий текущую роль пользователя, не появится.

BjyAuthorize

На данный момент все существующие версии BjyAuthorize содержат один занимательный баг, который нам придется подкорректировать своими силами. Исправление этого бага слегка зависло в воздухе и возможно даже выйдет вскоре после написания этой методички. Но, поскольку на данный момент решения нет, нам придется скорректировать эту ошибку своими силами.

В состав модулей ZF-Commons (https://github.com/ZF-Commons) обычно входят рекомендованные зендом модули, но если мы обратимся к странице модуля ZfcAcl (https://github.com/ZF-Commons/ZfcAcl), мы сможем увидеть:

> "Этот модуль устарел и больше не поддерживается. Пожалуйста, используйте вместо него https://github.com/bjyoungblood/BjyAuthorize."

Это как раз тот случай, когда авторский модуль оказался на порядок эффективнее рекомендуемого.

Модуль BjyAuthorize реализует шаблон проектирования "фасад" для предоставления доступа к Zend\Permissions\Acl, чтобы упростить интеграцию ACL при работе с модулями и приложениями. По умолчанию он предоставляет простую установку с помощью файлов конфигурации, либо с помощью Zend\Db или Doctrine ORM/ODM.

Чтобы напомнить, что такое шаблон "фасад" и не отсылать к сторонним источникам, приведу здесь определения и примеры, взятые из таких источников. Согласно определению, данному в википедии, шаблон фасад (англ. *Facade*) — шаблон проектирования, позволяющий скрыть сложность системы путем сведения всех возможных внешних вызовов к одному объекту, делегирующему их соответствующим объектам системы. Но для наглядности приведем выдержку из книги Бейтс, Сьерра, Фримен "Паттерны проектирования":

… Домашний кинотеатр, просмотр фильма (сложный способ)

Выберите диск, расслабьтесь и приготовьтесь встретиться с киноволшебством. Да, и еще одно… Чтобы посмотреть кино, нужно выполнить несколько операций:

- Включить аппарат для попкорна
- Запустить приготовление попкорна.
- Выключить свет.
- Опустить экран.
- Включить проектор.
- …

… честное слово, я не хочу писать все то, что перечислено в этой книге ☺, при выключении все происходит в обратном порядке ☺ …

Мы реализуем все это с помощью пульта управления кинотеатром, когда нажатие на одну кнопку приводит к выполнению ряда методов в заданной последовательности. Фасад определяет высокоуровневый интерфейс для работы с подсистемами. Он предоставляет унифицированный интерфейс (как пульт управления кинотеатром) к группе интерфейсов подсистемы. Он обеспечивает логическую изоляцию клиента от подсистемы, состоящей из многих компонентов. Если вам понадобится расширенная функциональность классов подсистемы – обращайтесь к ним напрямую.

Загрузка модуля

Для установки модуля перейдите в Far в каталог нашего приложения Zf2Guard, переключитесь в режим командного интерпретатора (Ctrl + O) и введите команду:

```
1 $ php composer.phar require bjyoungblood/bjy-authorize:1.4.*
```

Composer установит модуль и добавит его в composer.json

Подключение модуля

После того, как модуль будет загружен, нам необходимо внести модуль в конфигурацию приложения. Найдите в файле `Zf2Guard\config\application.config.php` раздел `modules` и измените его следующим образом:

```
'modules' => array(
    'Application',
    'TestGuardModule',
    'ZendDeveloperTools',
    'BjyProfiler',
    'ZfcBase',
    'ZfcUser',
    'BjyAuthorize',
),
```

Миграция модуля в БД

Поскольку мы установили модуль с помощью composer, схема для миграции в MySql должна находиться у нас по следующему пути:

```
Zf2Guard\vendor\bjyoungblood\bjy-authorize\data\schema.sql
```

Перейдите в Far в каталог `Zf2Guard\vendor\bjyoungblood\bjy-authorize\data`. Как мы можем видеть, в указанном каталоге находится также и ряд других файлов, их назначение мы рассмотрим в главе о расширенном использовании модуля совместно с Doctrine.

Переключитесь в Far в режим командного интерпретатора и вызовите клиент mysql для пользователя mysql zf2guard:

```
1 $ mysql -u zf2guard -pzf2guard
```

Выберите нашу базу данных и импортируйте данные:

```
1 $ USE zf2guard
2 $ source schema.sql
```

Давайте посмотрим, что мы импортировали в базу, выполните в командном интерпретаторе:

```
1 $ SHOW TABLES;
mysql> SHOW TABLES;
+--------------------+
| Tables_in_zf2guard |
+--------------------+
| user               |
| user_role          |
| user_role_linker   |
+--------------------+
3 rows in set (0.00 sec)
```

Как видим, сейчас у нас в базе три таблицы. Таблица user была установлена при миграции модуля ZfcUser. Таким образом, у нас добавились две таблицы – user_role, содержащая список ролей в системе и user_role_linker, которая связывает конкретного пользователя из таблицы user с конкретной ролью из таблицы user_role_linker.

Давайте посмотрим, что из себя представляет таблица user_role. Выполните в командном интерпретаторе:

```
1 $ DESCRIBE user_role;
```

```
mysql> DESCRIBE user_role;
+------------+--------------+------+-----+---------+----------------+
| Field      | Type         | Null | Key | Default | Extra          |
+------------+--------------+------+-----+---------+----------------+
| id         | int(11)      | NO   | PRI | NULL    | auto_increment |
| roleId     | varchar(255) | NO   |     | NULL    |                |
| is_default | tinyint(1)   | NO   |     | NULL    |                |
| parent_id  | varchar(255) | YES  |     | NULL    |                |
+------------+--------------+------+-----+---------+----------------+
4 rows in set (0.03 sec)
```

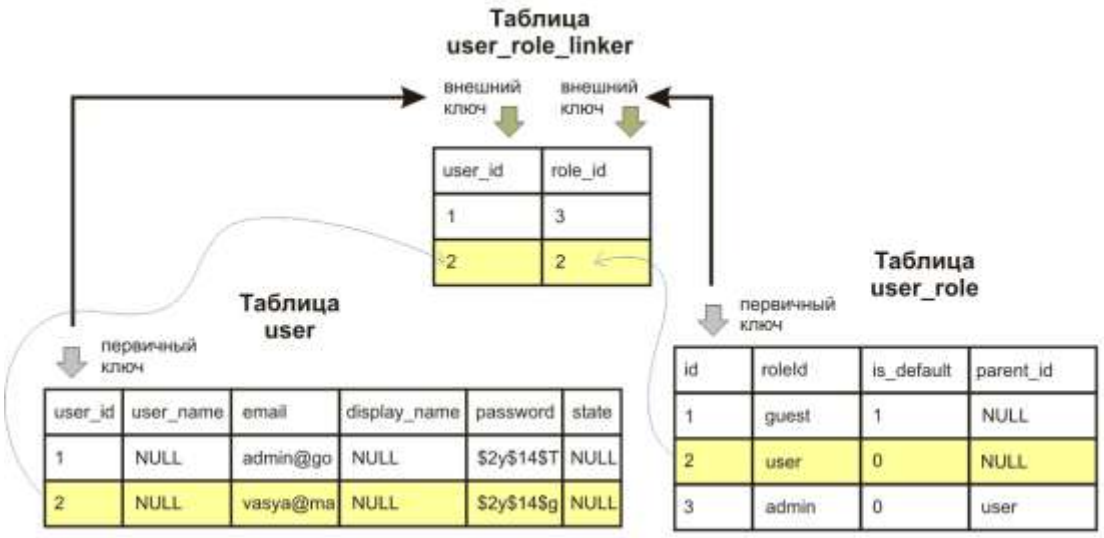
Как видим, столбец id является первичным ключом с атрибутом AUTO_INCREMENT. Несколько озадачивает имя второго столбца – roleId. В текущей версии модуля имя этого столбца отличается от остальных имен по стилю, поскольку использует верблюжью нотацию вместо нижнего подчеркивания. Трудно сказать, с чем это связано, но в предыдущих версиях модуля этот столбец назывался не roleId, а role_id, так что будьте внимательны при настройке параметров модуля, указывая существующие имена столбцов. Обратите внимание также на следующий момент – для столбца is_default значение по умолчанию (Default) – NULL, хотя тип столбца tinyint(1) (так в MySql представлен тип BOOLEAN). Если для команды INSERT мы явно не укажем 0 или 1 (что будет означать TRUE или FALSE) в качестве значения для этого столбца, это вызовет ошибку. Гораздо логичнее было бы объявить этот столбец, как NOT NULL DEFAULT '0', автор предполагает исправить это в следующей версии. Вторым интересным моментом является то, что значение поля parent_id четко привязано к текущей роли, и, следовательно, при такой схеме одна роль сможет иметь только одного родителя.

Давайте рассмотрим таблицу user_role_linker. Выполните команду:

```
1 $ DESCRIBE user_role_linker;
```

```
mysql> DESCRIBE user_role_linker;
+---------+------------------+------+-----+---------+-------+
| Field   | Type             | Null | Key | Default | Extra |
+---------+------------------+------+-----+---------+-------+
| user_id | int(11) unsigned | NO   | PRI | NULL    |       |
| role_id | int(11)          | NO   | PRI | NULL    |       |
+---------+------------------+------+-----+---------+-------+
2 rows in set (0.08 sec)
```

Как видим, в качестве первичного ключа объявлены оба столбца таблицы. То есть, первичный ключ будет состоять из комбинации обоих столбцов. Такой подход позволит не вводить дополнительный третий столбец, который служил бы исключительно в качестве первичного ключа. Оба столбца являются внешними ключами. Более наглядно взаимосвязь между таблицами представлена на цветной схеме.



Базовое конфигурирование модуля

В отличие от модуля ZfcUser, который работает и с настройками по умолчанию, настраивать модуль BjyAuthorize необходимо обязательно. Модуль обеспечивает автоматическую защиту двух видов – защиту маршрутов и защиту контроллеров. Мы применяли подобную методику, когда рассматривали организацию контроля доступа с помощью ACL фреймворка. Поэтому мы уже должны понимать, что для того, чтобы организовать подобную защиту, нам необходимо указать модулю, какие контроллеры и маршруты существуют в нашей системе. Вполне естественно, что модуль BjyAuthorize не может работать с настройками по умолчанию, поскольку он попросту не обладает всей этой информацией.

Пока что мы лишь ознакомимся с общей конфигурацией, а собственную будем выполнять поэтапно. Полностью конфигурация модуля выглядит следующим образом:

```
<?php

return array(
    'bjyauthorize' => array(

        // Установите роль 'guest', как роль по умолчанию
        // (она должна быть также задана в секции role provider).
        'default_role' => 'guest',
```

```
/* Модуль использует мета-роли, которые могут наследоваться от любых ролей,
 * которые должны быть применены к активному пользователю.
 * Провайдер идентификации (identity provider) предоставляет информацию о том,
 * какие роли "с собственной идентичностью" (т.е., не guest) унаследованы
 * (текущей ролью пользователя).
 *
 * для ZfcUser это должен быть провайдер идентификации по умолчанию:
 */
'identity_provider' => 'BjyAuthorize\Provider\Identity\ZfcUserZendDb',

/* Если у вас есть только роль по умолчанию (гость, не требующая
 * аутентификации) и роль, требующая аутентификации, вы можете использовать
 * 'AuthenticationIdentityProvider', чтобы предоставлять или разрешать
 * доступ на основании того, выполнял ли пользователь вход в систему (login),
 * или нет. (Т.е. в этом случае может быть всего две роли в системе, скажем,
 * гость и админ, или же гость и пользователь. Если вас это не устраивает, и
 * необходимо более, чем две роли, используйте провайдер идентификации по
 * умолчанию, как показано выше для ZfcUser.)
 *
 * 'default_role'       => 'guest',         // not authenticated
 * 'authenticated_role' => 'user',          // authenticated
 * 'identity_provider'  => 'BjyAuthorize\Provider\Identity\AuthenticationIdentityProvider',
 */

/* Провайдеры ролей (role providers) просто предоставляют список ролей,
 * которые должны быть в системе, в экземпляр Zend\Acl. Модуль предоставляет
 * два провайдера, один используется для указания ролей с помощью файла
 * конфигурации (который мы сейчас и заполняем), а другой загружает роли
 * из базы данных, используя адаптер Zend\Db. Для второго провайдера
 * необходимо самостоятельно создать в базе данных необходимую вам
 * конфигурацию ролей. В этом вам поможет цветная схема выше в этой главе.
 */
'role_providers' => array(

    /* здесь 'guest' и 'user' определены, как роли верхнего уровня, а
     * 'admin' наследуется от 'user'
     *
     * 'BjyAuthorize\Provider\Role\Config' => array(
     *     'guest' => array(),
     *     'user'  => array('children' => array(
     *         'admin' => array(),
     *     )),
     * ),
     */

    // здесь роли загружаются из таблицы user_role базы данных,
    // вы можете указать собственную таблицу, главное, чтобы соблюдались
    // типы данных для столбцов: user_role(role_id(varchar), parent(varchar))
```

```
    // Я смотрю, чтобы программисты не скучали, автор иногда меняет
    // имена столбцов в schema.sql модуля, так что будьте внимательны.
    'BjyAuthorize\Provider\Role\ZendDb' => array(
        'table'            => 'user_role',
        'role_id_field'    => 'roleId',
        'parent_role_field' => 'parent_id',
    ),

    // При использовании Doctrine это позволит загружать роли
    // из сервиса 'BjyAuthorize\Provider\Role\Doctrine'
    'BjyAuthorize\Provider\Role\Doctrine' => array(),
),

// провайдер ресурсов (resource provider) предоставляет список ресурсов
// в ACL, как и роли, они могут быть иерархическими.
// Заметьте, что ресурсы на сегодняшний день не хранятся в базе данных, и
// и считываются только из этой конфигурации.
'resource_providers' => array(
    'BjyAuthorize\Provider\Resource\Config' => array(
        'pants' => array(), // pants - брюки, читай пример ниже
    ),
),

/* здесь могут быть указаны правила в формате:
 * array(roles (array), resource, [privilege (array|string), assertion])
 *
 * Утверждения (assertions) должны быть загружены с помощью сервис-менеджера
 * и должны реализовывать интерфейс Zend\Acl\Assertion\AssertionInterface.
 * *если вы используете утверждения, определите их в сервис-менеджере!*
 * Мы еще не рассматривали утверждения, эта тема будет раскрыта
 * в расширенном использовании.
 */
'rule_providers' => array(
    'BjyAuthorize\Provider\Rule\Config' => array(
        'allow' => array(
            // позволяет гостям и пользователям (и администраторам, по наследству)
            // привелегию "носить" (wear) для ресурса "брюки" (pants)
            array(array('guest', 'user'), 'pants', 'wear')
        ),

        // Не смешивайте правила разрешения/запрещения (allow/deny), если вы
        // используете наследование для ролей.
        // это может привести к непредсказуемым ошибкам.
        'deny' => array(
            // ...
        ),
    ),
),
```

```
/* В настоящее время защита (guards) существует только для контроллеров и маршрутов:
 *
 * Обращаю ваше внимание, если вы не поняли этот момент, поскольку это реализация
 * шаблона "фасад", то ресурсы, привилегии и разрешения, которые мы задавали ранее,
 * просто передаются в ACL. Реальная защита контроллеров и маршрутов указывается
 * здесь. Поскольку защите подлежат только маршруты и контроллеры, возможно, скажем,
 * где-то в шаблоне представления нам нужно будет проверить разрешения на доступ к
 * ресурсу и в зависимости от того, что вернет isAllowed(), показывать это
 * пользователю, или нет. И эти правила мы можем как раз указать в предыдущих
 * пунктах. В этом же разделе указываются параметры автоматической защиты.
 *
 * Рассмотрите возможности включения защиты для контроллеров или маршрутов,
 * в зависимости от того, что именно вам необходимо.
 */
'guards' => array(
    /* Если этот защитник (guard) указан здесь (т.е., если он включен), он
     * будет блокировать доступ ко всем контроллерам и действиям, которые
     * здесь не указаны. Вы можете опустить ключ 'action', чтобы предоставить
     * доступ ко всему контроллеру.
     */
    'BjyAuthorize\Guard\Controller' => array(
        array('controller' => 'index', 'action' => 'index',
              'roles' => array('guest','user')),
        array('controller' => 'index', 'action' => 'stuff',
             'roles' => array('user')),
        // Вы можете также задать массив действий или массив контроллеров (или то и
        // другое), чтобы позволить для "guest" и "admin" получить доступ к действиям
        // "list" и "manage" для контроллеров "index", "static" и "console"
        array(
            'controller' => array('index', 'static', 'console'),
            'action' => array('list', 'manage'),
            'roles' => array('guest', 'admin')
        ),
        array(
            'controller' => array('search', 'administration'),
            'roles' => array('staffer', 'admin') // staffer (англ.) - сотрудник
        ),
        array('controller' => 'zfcuser', 'roles' => array()),
        // Ниже - действие по умолчанию index для ZendSkeletonApplication
        // array('controller' => 'Application\Controller\Index',
        //     'roles' => array('guest', 'user')),
    ),

    /* Если этот защитник указан здесь (т.е., если он включен), он будет
     * блокировать доступ ко всем маршрутам, которые здесь не указаны.
     */
    'BjyAuthorize\Guard\Route' => array(
```

```
                array('route' => 'zfcuser', 'roles' => array('user')),
                array('route' => 'zfcuser/logout', 'roles' => array('user')),
                array('route' => 'zfcuser/login', 'roles' => array('guest')),
                array('route' => 'zfcuser/register', 'roles' => array('guest')),
                // Ниже - действие по умолчанию index для ZendSkeletonApplication
                array('route' => 'home', 'roles' => array('guest', 'user')),
            ),
        ),
    ),
);
```

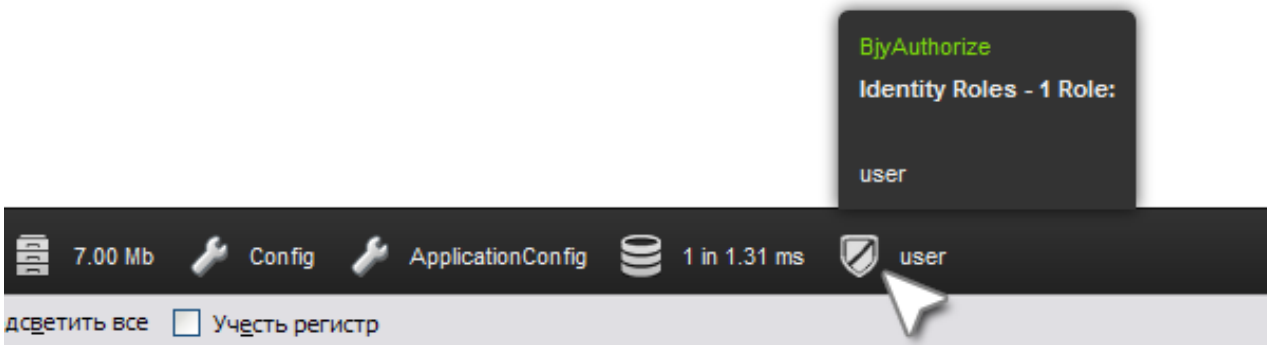
Для того, чтобы установить конфигурацию модуля, нам нужно указать необходимые параметры в файле `Zf2Guard\config\autoload\bjyauthorize.local.php`. Если же мы сейчас обратимся к нашему приложению из браузера и наведем курсор на значок щитка в панели инструментов Developer Tools, то увидим, что BjyAuthorize никак не идентифицировал нашу роль в системе:

```
BjyAuthorize
Identity Roles - No Role:
```

Давайте исправим ситуацию, создав простейшую конфигурацию для модуля BjyAuthorize в файле `Zf2Guard\config\autoload\bjyauthorize.global.php`:

```
<?php

return array('bjyauthorize' => array(
    'default_role'       => 'guest',  // неавторизированный пользователь
    'authenticated_role' => 'user',   // авторизированный пользователь
    'identity_provider'  => 'BjyAuthorize\Provider\Identity\AuthenticationIdentityProvider',
));
```

Если мы теперь перегрузим в браузере страницу и наведем курсор на значок щита, модуль BjyAuthorize определит нашу роль в зависимости от того, проходили ли мы уже аутентификацию с помощью модуля ZfcUser, или нет.



Как вы уже могли узнать, если прочитали комментарии к файлу конфигурации BjyAuthorize, если мы будем использовать в качестве провайдера идентификации (identity provider) простой провайдер BjyAuthorize\Provider\Identity\AuthenticationIdentityProvider, предоставляемый модулем, мы сможем использовать в системе только две роли. Эти роли

могут иметь произвольное название, но одна из них (в данном случае, guest) представляет собой роль для пользователя, не прошедшего аутентификацию, а вторая (назовите ее как угодно, скажем user, или admin, или еще как-нибудь) – для пользователей, прошедших аутентификацию.

Если подобная система доступа покажется вам достаточной, то просто сконфигурируйте раздел `guards`, и на этом ваша настройка доступа закончится. Если ролью для аутентифицированного пользователя вы хотите назначить `admin`, вы можете указать в модуле ZfcUser, что регистрация новых пользователей запрещена, и система будет прекрасно работать. Дальнейшее конфигурирование базы данных в этом случае не нужно.

Если же вы все-таки хотите воспользоваться более гибкой системой доступа, то в качестве провайдера идентификации необходимо будет выбрать

'identity_provider' => 'BjyAuthorize\Provider\Identity\ZfcUserZendDb'

При этом конфигурация ролей будет браться из базы данных, из таблицы user_role. Если вы внимательно читали комментарии к файлу конфигурации, то наверняка заметили, что конфигурацию ролей можно задать и напрямую в файле конфигурации, не используя базу данных. Сразу хотелось бы пояснить, что эта возможность снова обусловлена только тем, что BjyAuthorize является фасадом для ACL и поэтому позволяет внести в ACL роли напрямую из файла конфигурации, однако если мы это сделаем, модуль не сможет определить, какая именно из указанных в конфигурации ролей соответствует пользователю, вошедшему в систему (login). Кроме того, если в базе для роли guest в столбце is_default указано 1 (т.е. TRUE), то в конфигурации это можно не указывать.

По сути, файлы схемы БД (schema.sql) для модуля BjyAuthorize позволяют только создать необходимые таблицы. Но реально для каждого нашего приложения нам придется писать собственную схему миграции. При этом файл `my.schema.sql` должен будет содержать данные как для модуля ZfcUser, так и для модуля BjyAuthorize, поскольку данные пользователя, который первым будет добавлен в систему, т.е., администратора, должны быть взаимосвязаны с той системой ролей, которая также должна находиться в базе данных.

Давайте изменим конфигурационный файл `bjyauthorize.local.php`, чтобы он работал с базой данных, следующим образом:

```
<?php

return array('bjyauthorize' => array(
    //'default_role'      => 'guest',
    'identity_provider' => 'BjyAuthorize\Provider\Identity\ZfcUserZendDb',
    'role_providers' => array(
        'BjyAuthorize\Provider\Role\ZendDb' => array(
            'table'            => 'user_role',
            'role_id_field'    => 'roleId',
            'parent_role_field' => 'parent_id',
```

```
        ),
    ),
));
```

Чтобы это работало, нам нужно создать в таблицах модуля BjyAuthorize конфигурацию, приведенную на цветной схеме. Можно сделать это с помощью к-л инструментов, например из командного интерпретатора с помощью клиента `mysql`, или с помощью phpMyAdmin, но более оптимальным способом является составление собственного файла схемы БД.

Но не спешите устанавливать параметры защиты в разделе конфигурации guards. Если сейчас мы обратимся по адресу `localhost.zf2guard/user` и пройдем аутентификацию, мы увидим, что наша роль в системе не была правильно идентифицирована. На панели ZendDeveloperTools вместо имени роли мы увидим ее идентификатор, в данном случае "3" (для администратора, если вы настроили роли в соответствии с цветной схемой). Этот баг содержится во всех версиях модуля, и сейчас авторы работают над его исправлением. Возможно, в то время, как вы читаете эту инструкцию, баг уже будет исправлен и мне придется корректировать эту методичку. Но на данный момент дела обстоят именно так. Итак, что же делать? Правильным решением, конечно же, будет создать собственный файл провайдера идентификации и указать этот файл в конфигурации модуля BjyAuthorize:

```
'identity_provider' => 'TestGuardModule\Provider\Identity\ZfcUserZendDb',
```

но, поскольку авторы обещали со дня на день исправить эту ошибку, я предлагаю просто открыть файл

```
BjyAuthorize\Provider\Identity\ZfcUserZendDb.php
```

и напрямую заменить содержимое метода getIdentityRoles() следующим образом:

```
public function getIdentityRoles()
{
    $authService = $this->userService->getAuthService();

    if (! $authService->hasIdentity()) {
        return array($this->getDefaultRole());
    }

    // get roles associated with the logged in user
    $sql    = new Sql($this->adapter);

    $select = $sql->select()
        ->columns(array('role_id' => 'roleId'))
        ->from(array('roles' => 'user_role'))
        ->join(array('linker' => $this->tableName), 'roles.id = linker.role_id', array())
        ->where(array('linker.user_id = ?' => $authService->getIdentity()->getId()));

    $results = $sql->prepareStatementForSqlObject($select)->execute();
    $roles     = array();

    foreach ($results as $i) {
```

```
            $roles[] = $i['role_id'];
        }

        return $roles;
    }
```

После того, как вы это сделаете, вы сможете в панели ZendDeveloperTools увидеть ту реальную роль, с которой вы вошли в систему и спокойно создать конфигурацию защиты в разделе guards, которая вам необходима.

Вопросы расширенной конфигурации будут рассмотрены после обновления модуля BjyAuthorize.