

RНР

быстрый старт

КАЛЛУМ ХОПКИНС

Освойте RНР за выходные

УДК 004.45
ББК 32.973-018.2
П 22

© 2014 Eksmo Publishing Company

Authorized Russian translation of the English edition of Jump Start PHP, 1st Edition
(ISBN 9780987467409) © 2013 Sitepoint Pty. Ltd.

This translation is published and sold by permission of O'Reilly Media, Inc.,
which owns or controls all rights to publish and sell the same

PHP. Быстрый старт ; [пер. с англ. М. А. Райтман]. — Москва : Эксмо,
П 22 2014. — 160 с. — (Компьютер на 100%. Быстрый старт).

ISBN 978-5-699-72685-1

Язык PHP очень популярен, он обеспечивает функционирование 80% всех веб-сайтов, в том числе таких ресурсов, как Facebook, Wikipedia и WordPress. Этот язык прост в изучении и отлично подходит для начинающих, так что вы сможете довольно быстро начать его использовать. Вы узнаете, как шаг за шагом создать функционирующее блог-приложение, поймете, как PHP работает с данными, а также повысите безопасность своих PHP-приложений. Всего за несколько дней вы подготовите солидную базу для самостоятельного создания PHP-кода.

**УДК 004.45
ББК 32.973-018.2**

ISBN 978-5-699-72685-1

© Райтман М.А., перевод на русский язык, 2014
© Оформление. ООО «Издательство «Эксмо», 2014

ОГЛАВЛЕНИЕ

Предисловие	8
Для кого предназначена эта книга.....	8
Условные обозначения	9
Образцы кода	9
Советы, примечания и предупреждения.....	10
Об авторе.....	11
О серии «Интенсивный курс»	11
Глава 1. Быстрая установка сервера	12
Что такое PHP?.....	12
Настройка	13
Начало работы.....	14
Операционная система Windows.....	14
Операционная система OS X.....	17
Операционная система Linux.....	19
Конфигурация PHP	21
Привет, мир PHP.....	22
Переменные языка PHP	26
Массивы.....	29
Комментарии	31
Подготовка проекта	32
Резюме.....	33
Глава 2. PHP и данные	34
Операторы	34
Условные операторы	36
Оператор if.....	36
Оператор else.....	37
Оператор elseif.....	38
Оператор switch.....	38
Циклы.....	40
Цикл for.....	40
Цикл while	41
Цикл foreach.....	42
Базы данных, MySQL и PHP.....	43
Резюме.....	53

Глава 3. Объекты и парадигма ООП	54
Первые шаги в ООП.....	55
Расширение классов.....	58
Шаблоны	63
Файлы проекта.....	67
Резюме.....	75
Глава 4. Формы	76
Элементы форм.....	76
Методы POST и GET.....	78
Действия формы в PHP.....	84
Суперглобальные переменные и массив \$_REQUEST	86
Формы и базы данных.....	87
Усовершенствование нашей платформы.....	92
Резюме.....	108
Глава 5. Сеансы и файлы cookie.....	109
Файлы cookie: обзор.....	109
Сеансы: обзор.....	110
Сравнение сеансов и файлов cookies.....	111
Файлы cookie.....	111
Сеансы.....	112
Сеансы и файлы cookie в PHP.....	113
Файлы cookie в PHP.....	114
Сеансы в PHP	116
Работа над проектом	118
Резюме.....	145
Глава 6. Язык PHP и безопасность	146
Файл php.ini и безопасность.....	146
Параметр allow_url_include.....	147
Параметр open_basedir.....	148
Управление ошибками.....	148
Повышение безопасности сеанса.....	149
Проверка предоставленных данных	151
Резюме.....	153
Заключение	155
Предметный указатель.....	156

*Посвящается моей семье и друзьям,
спасибо за вашу поддержку.
Я люблю вас.*

ПРЕДИСЛОВИЕ

PHP считается одним из самых популярных языков для разработки веб-приложений. Он создан для того, чтобы сделать динамическим содержимое веб-страниц, однако за годы своего существования язык PHP превратился в нечто гораздо более полезное. Благодаря PHP разработчики легко могут создавать сложные приложения, например форумы, фотогалереи и многое другое.

Эта книга познакомит вас с процессом написания кода и разработки на языке PHP и проведет по пути от создания простой веб-страницы с динамическим контентом до построения интерактивных веб-приложений. Мы коснемся вопросов безопасности, взаимодействия с базой данных и создания среды разработки для ваших PHP-приложений.

По мере чтения книги вы будете работать над проектом небольшого, но надежного блог-приложения, чтобы применить теоретические знания, полученные в каждой главе, к реальному процессу разработки. Этот проект будет включать в себя некоторые полезные функции (внешний интерфейс для отображения сообщений, комментариев и инструментов администрирования) и, мы надеемся, позволит продемонстрировать рассмотренные в книге понятия.

Для кого предназначена эта книга

Эта книга предназначена для разработчиков, которым требуется быстрое ознакомление с языком PHP. Вам понадобится знание HTML и CSS, а также пригодится опыт работы с другими языками программирования.

Условные обозначения

Вы заметите, что в книге используются определенные стили оформления для обозначения различных типов информации. Обращайте внимание на следующие элементы.

Образцы кода

Код в этой книге отображается моноширинным шрифтом:

```
<h1>Прекрасный летний день</h1>
<p>Это был прекрасный день для прогулки в парке.
☞Птицы пели, а все дети вернулись в школу.</p>
```

Если код находится в архиве, прилагающемся к книге, то имя файла будет указано в верхней части листинга программы:

```
example.css
.footer {
background-color: #CCC;
border-top: 1px solid #333;
}
```

Если приводится только часть файла, на это указывает слово *фрагмент*:

```
example.css (фрагмент)
border-top: 1px solid #333;
```

При необходимости вставки в существующий пример дополнительного кода новый код будет выделен полужирным шрифтом:

```
function animate() {
new_variable = "Hello";
}
```

Кроме того, когда существующий код требуется для контекста, вместо его повторения используется символ ::

```
function animate() {  
:  
return new_variable;  
}
```

Некоторые фрагменты кода должны находиться на одной строке, но нам приходится перемещать их на следующую из-за ограничения ширины страницы. Символ ↵ указывает на перенос строки и присутствует только в целях форматирования, поэтому его следует игнорировать.

```
URL.open("http://www.sitepoint.com/responsive-web-  
↵design-real-user-testing/?responsive1");
```

Советы, примечания и предупреждения



СОВЕТ

Советы содержат небольшие полезные указания.



ПРИМЕЧАНИЕ

Примечания — это полезные отступления, которые имеют отношение к теме, но не являются критично важными. Считайте их дополнительными лакомыми кусочками информации.



ОБРАТИТЕ ВНИМАНИЕ

...на эти важные моменты.



ПРЕДУПРЕЖДЕНИЕ!

Предупреждения укажут на подводные камни, с которыми вы можете столкнуться.

Об авторе

Каллум — веб-разработчик по профессии и дизайнер по призванию. Обладая знаниями в области дизайна и разработки, он может влиять на обе стороны процесса при создании сайта. Страсть к сложным функциям, красивому дизайну и функциональности заставляет его искать новые пути построения, проектирования и оптимизации веб-решений для клиентов по всему миру.

О серии «Интенсивный курс»

Серия книг «Интенсивный курс» позволяет вам быстро и на практике познакомиться с языками и технологиями для веб-разработки. Обычно объем этих книг составляет около 150 страниц, вы можете прочитать их за выходные и получить базовые знания по теме, а также приобрести уверенность для самостоятельных экспериментов.

БЫСТРАЯ УСТАНОВКА СЕРВЕРА

Что такое PHP?

PHP является самым популярным серверным скриптовым языком, который используется в сфере веб-разработки и обеспечивает работу примерно 78,9% всех веб-сайтов.

Этот язык создан Расмусом Лердорфом в 1995 году и первоначально назывался Personal Home Page Tools («Инструменты для создания персональных веб-страниц»), однако в настоящее время он более известен как PHP: Hypertext Preprocessor («PHP: препроцессор гипертекста»). Его управлением, контролем и совершенствованием занимается группа разработчиков, известная под названием The PHP Group, которая продолжает бесплатно распространять этот скриптовый язык через официальный сайт¹.

Код PHP чаще всего интерпретируется, обрабатывается и отображается с помощью веб-сервера с установленным модулем PHP, что позволяет встраивать в HTML-разметку код, находящийся в файлах с расширением *.php*. Кроме того, язык PHP может бесплатно использоваться практически в любой операционной системе и на платформе, при этом наиболее популярно его сочетание с системами на базе Linux.

В настоящее время разработка на PHP в основном сосредоточена на создании серверных скриптов, а не на решении задач общего назначения, и соперничает с такими технологиями, как ASP.NET корпорации Microsoft, модуль `mod_perl` организации Apache Software Foundation

¹ php.net

и Node.js компании Joyent. Язык PHP используется в первую очередь для обработки сложных типов данных, позволяющей отображать на веб-страницах динамические данные, связанные с математическими расчетами, обработкой огромных числовых массивов и взаимодействием с базой данных. Это позволяет разработчикам заставить изначально статический HTML-контент или информацию, постоянно хранящуюся в базе данных, реагировать на запросы пользователей.

Язык PHP используется в основном для веб-разработки, что делает его очевидным выбором при создании веб-приложений или веб-сайтов. Пологая кривая обучения PHP позволяет разработчикам быстро приступить к его использованию, в то время как с помощью широкого спектра функций они развивают свои проекты, не прибегая к другим языкам программирования. Такие ресурсы, как Digg, Etsy, Yahoo, Facebook и Wikipedia, используют язык PHP не только для поддержания работы сайта, но и для обработки данных, касающихся посетителей.

Простым примером использования кода PHP на веб-странице является счетчик, отображающий количество посетителей. Количество людей, посетивших веб-страницу, хранится в базе данных, а PHP взаимодействует с этой базой данных и генерирует HTML-разметку для отображения текущего показателя. PHP также может использоваться для создания больших и сложных сайтов с многоуровневой навигацией, содержащих множество вложенных страниц, а также обычно применяется для поддержания работы сайтов электронной коммерции. Кроме того, PHP позволяет обеспечить уникальное взаимодействие с посетителем, используя собранную о нем информацию.

Популярность языка PHP способствовала образованию огромного сообщества разработчиков, которые готовы бесплатно оказать помощь тем, кто нуждается в совете, а также созданию постоянно расширяющейся библиотеки справочных материалов, доступных как во Всемирной паутине, так и в автономном режиме.

Настройка

Поддержка языка PHP входит практически в любой пакет хостинговых услуг. Кроме того, PHP может использоваться вместе с программным обеспечением Apache HTTP Server для создания локального веб-сервера на домашнем компьютере. PHP также может применяться на вашем собственном веб-сервере, доступ к которому предоставляется через Интернет.

Локальные серверы на домашних компьютерах часто создаются с помощью популярных комплектов программного обеспечения (стеков) LAMP (Linux, Apache, MySQL и PHP), MAMP (Mac OS X, Apache, MySQL и PHP) и WAMP (Windows, Apache, MySQL и PHP), доступных для бесплатной загрузки.

Эти стеки обычно содержат программу установки, позволяющую с помощью одного щелчка мышью установить сконфигурированный по умолчанию веб-сервер со стандартными настройками. Это позволяет веб-разработчикам настроить локальную среду, которая почти идентична той, что предоставляется их хостинговой компанией. Разработчики часто начинают создание сайтов и приложений, используя локальный сервер, что объясняется легкостью получения доступа к рабочим файлам и возможностью избежать хлопот и траты времени на загрузку файлов на веб-сервер.

Кроме того, данный метод позволяет не беспокоиться о возможном падении разрабатываемого кода на рабочий сайт и избежать ненужного трафика при передаче файлов.

Начало работы

Установка локального сервера на вашем домашнем компьютере может показаться сложной задачей, однако обычно это довольно просто. В зависимости от используемой вами операционной системы у вас есть несколько вариантов.

Операционная система Windows

В операционной системе Windows вы можете выбрать одну из двух популярных и мощных программ установки. Первая — это WAMP¹, которая устанавливает на ваш компьютер инструменты Apache, PHP, MySQL и phpMyAdmin (удобный веб-интерфейс для работы с базами данных MySQL). Другой программой является XAMMP² проекта Apache Friends, которая включает Apache, MySQL, PHP, Perl и phpMyAdmin.

В этой книге мы рассмотрим только настройку стека WAMP, однако, если вы решите выбрать пакет XAMMP, процесс установки будет аналогичным.

¹ www.wampserver.com/ru

² www.apachefriends.org/en/xampp.html

Первым шагом является загрузка пакета WAMP¹. Вам будет предложено несколько вариантов загрузки, поэтому выберите версию PHP 5.4 2.4, соответствующую процессору и операционной системе вашего компьютера.



КАКОЙ У МЕНЯ ПРОЦЕССОР?

Чтобы узнать, какой у вас процессор, найдите значок «Мой компьютер» или откройте окно **Панель управления** (Control Panel) и перейдите в раздел **Система** (System). Найдите пункт **Тип системы** (System Type), в котором указано, какую операционную систему использует ваш компьютер — 32- или 64-разрядную.



СРЕДА РАЗРАБОТКИ VISUAL C++

Вам также необходимо установить на свой компьютер среду разработки Visual C++ 2010 SP1 x86 или x64. На веб-странице вы увидите ссылки, соответствующие версии вашей операционной системы (32- или 64-разрядной). Настоятельно рекомендуется загрузить и установить этот пакет перед установкой стека WAMP.

После завершения загрузки запустите программу установки.

Как только установка WAMP будет закончена, вы увидите новый значок в области уведомлений панели задач операционной системы Windows. Цвет значка WAMP меняется в зависимости от текущего состояния служб:

- красный — службы Apache и MySQL работают в автономном режиме. Это может быть связано с тем, что они еще не запущены или их запуску мешает серьезная проблема в конфигурации;
- оранжевый — одну из служб запустить не удалось. Обычно это связано с незначительной проблемой в конфигурации, например с некорректной загрузкой библиотеки дополнений или с тем, что порт по умолчанию используется другой программой. Если это произойдет, обратитесь за помощью к службе поддержки или на форум;
- зеленый — все службы работают нормально, никаких ошибок не возникло.

¹ www.wampserver.com/ru/#download-wrapper

Если вы щелкнете левой кнопкой мыши по значку WAMP, откроется меню, содержащее различные варианты взаимодействия со службами, которые позволяют вам:

- управлять службами Apache и MySQL;
- переключаться между режимом онлайн и автономным режимом;
- устанавливать различные версии Apache, MySQL и PHP и переключаться между ними;
- получать доступ к файлам журнала;
- получать доступ к файлам с настройками;
- создавать псевдонимы.

Щелчок правой кнопкой мыши по значку позволяет выйти из программы, изменить ее язык и перейти на страницу поддержки веб-сайта WAMP. Если вы не уверены в чем-то, что касается стека WAMP, обратитесь к справочным файлам на официальном сайте, щелкнув по значку правой кнопкой мыши.

Чтобы проверить правильность установки и настройки локального сервера, откройте свой веб-браузер и в адресной строке введите **localhost** или **127.0.0.1**.

Вы должны попасть на главную страницу WAMP.

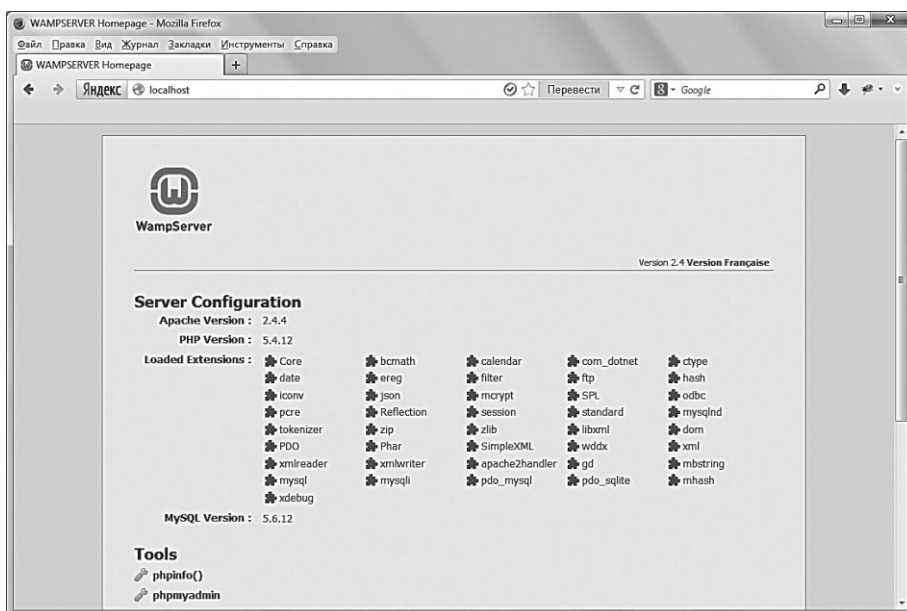


Рис. 1.1. Главная страница WAMP



ПРОБЛЕМЫ, СВЯЗАННЫЕ С ПОРТОМ 80

Наиболее распространенной проблемой, возникающей при работе с сервером WAMP, является то, что порт 80 (порт, используемый для подключения к серверу Apache) может быть занят другой программой. Например, если на вашем компьютере работает программа Skype, то вы можете столкнуться с этой проблемой при попытке запуска сервера WAMP, поскольку программа Skype также использует порт 80. Чтобы это исправить, рекомендуется закрыть приложение Skype и перезагрузить сервер WAMP. На сайте Stack Overflow¹ приведено решение этой проблемы.

Операционная система OS X

Пользователи компьютеров Mac могут настроить локальный сервер с PHP, используя простое приложение под названием MAMP. Как и в случае с операционной системой Windows, вы можете использовать пакет XAMPP, однако MAMP является более популярным вариантом, поскольку он обеспечивает настройки, которые идеально подходят для разработки веб-сайтов на PHP.

Во-первых, вам нужно скачать приложение с сайта MAMP², а затем, после завершения загрузки, открыть файл *.pkg*, который должен запустить процесс установки.



ПАПКА MAMP PRO

Программа установки MAMP может создать две папки: MAMP и MAMP Pro. Если на вашем компьютере создана папка MAMP Pro, удалите ее и соответствующее приложение. MAMP Pro — платная программа, в то время как MAMP — это ее бесплатный аналог.

Как только это будет сделано, вы можете запустить программу, щелкнув по значку MAMP, после чего должен появиться начальный экран данного приложения.

В левой части начального экрана, изображенного на рис. 1.2, вы увидите два пункта: **Apache Server** и **MySQL Server**, рядом с которыми должны быть красные точки. Эти красные точки означают, что данные службы

¹ stackoverflow.com/a/4705033

² www.mamp.info/en/index.html

не запущены. Чтобы запустить их, нужно нажать кнопку **Start Servers** (Запуск серверов), после чего точки должны стать зелеными. Вскоре после нажатия кнопки **Start Servers** (Запуск серверов) должен открыться браузер по умолчанию с главной страницей MAMP.

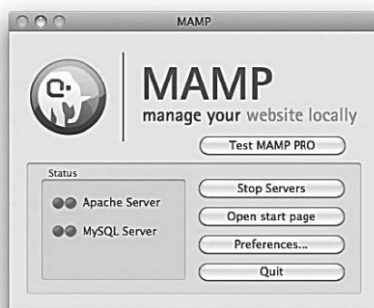


Рис. 1.2. Начальный экран MAMP

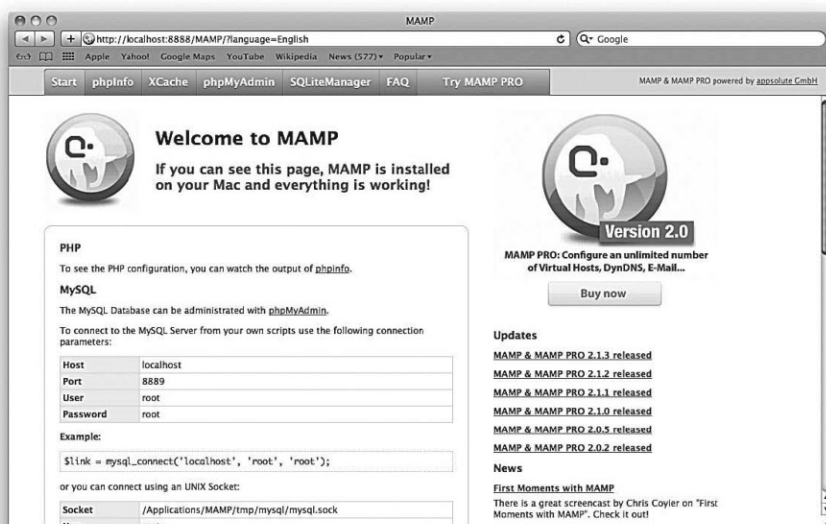


Рис. 1.3. Главная страница MAMP

Программа MAMP установлена. Тем не менее для доступа к серверу Apache используется порт 8888. Это менее удобно, чем получение доступа посредством ввода значения `localhost` в адресной строке браузера, как в случае с программой WAMP. К счастью, это легко исправить. Вернитесь к начальному экрану MAMP и щелкните по кнопке **Preferences**

(Параметры). В верхней части панели в небольшом навигационном меню выберите пункт **Ports** (Порты).

В данном окне вы должны увидеть поле ввода под названием **Apache Port** (Порт Apache). Если вы измените значение этого параметра с 8888 на 80, то сможете получить доступ к главной странице MAMP, введя значение **localhost/MAMP** в адресной строке браузера.

Операционная система Linux

Если вы работаете с операционной системой Linux, например Ubuntu или Debian, вы можете использовать терминал для установки пакета LAMP из репозитория системы. Процесс установки локального сервера в операционной системе Linux несколько отличается от того же процесса в операционной системе Windows или OS X. Он немного сложнее, однако предоставляет гораздо больше контроля при использовании и управлении сервером. На самом деле этот процесс практически идентичен процессу управления функционирующим веб-сервером в операционной системе Linux. В примере, который мы рассмотрим далее, описывается способ установки программы LAMP в операционной системе Ubuntu.

Сначала запустите терминал. Для этого найдите приложение Терминал (Terminal) в меню **Dash** (первый значок в боковом меню Ubuntu), или выберите команду меню **Приложения** ⇒ **Стандартные** ⇒ **Терминал** (Applications ⇒ Accessories ⇒ Terminal).

После открытия терминала вы можете начать процесс установки сервера Apache на свой компьютер. Для этого введите в терминале:

```
sudo apt-get install apache2
```

Нажмите клавишу **Enter**, после чего в окне терминала отобразится процесс установки.

Теперь у вас должна быть установлена программа Apache и настроены базовые параметры локального сервера. Чтобы убедиться в этом, откройте веб-браузер и введите значение `localhost` в адресной строке. Если установка прошла успешно, вы должны увидеть страницу, показанную на рис. 1.4.

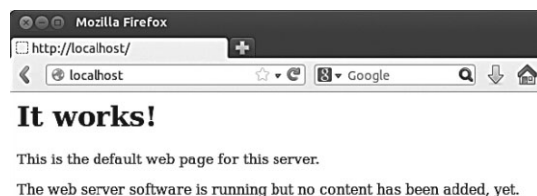


Рис. 1.4. Локальный хост в операционной системе Linux

Теперь вам нужно установить PHP, поэтому вернитесь в терминал и введите:

```
sudo apt-get install php5 libapache2-mod-php5*
```

Версия PHP 5 будет установлена наряду с базовыми библиотеками, необходимыми для обеспечения совместной работы с сервером Apache, который вы установили ранее. По окончании необходимо перезапустить программу Apache, чтобы в действие вступили изменения, связанные с установкой PHP. В терминале введите следующую команду:

```
sudo service apache2 restart
```

Ваш сервер Apache будет перезапущен. Убедитесь, что язык PHP установлен и все библиотеки, необходимые для его работы на локальном сервере, загружены.

Теперь у вас есть функционирующий локальный сервер с установленным на нем языком PHP. Для установки MySQL и phpMyAdmin я рекомендую следовать процедуре, описанной в руководстве Мхабуба Мамуна¹.

Наконец, у вас есть возможность включить модуль `mod_rewrite` сервера Apache, позволяющий разработчикам перенаправлять пользователей к различным разделам сайта путем переписывания запрашиваемых URL-адресов. Процедура включения и настройки модуля `mod_rewrite` подробно описана в руководстве *Nettuts+*².

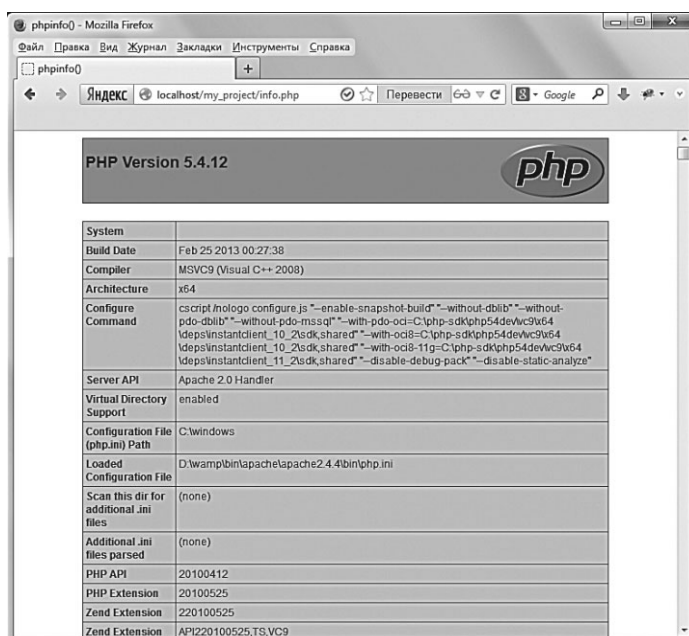
¹ www.developmentwall.com/install-apache-php-mysql-phpmyadmin-ubuntu
² net.tutsplus.com/tutorials/other/a-deeper-look-at-mod_rewrite-for-apache

Конфигурация PHP

Прежде чем мы примемся за написание кода на языке PHP, давайте настроим его, чтобы он лучше отвечал требованиям нашего проекта. Настройка осуществляется с помощью файла *php.ini*, который содержит все основные параметры, относящиеся к работе PHP на нашем сервере. Вы можете ознакомиться текущей конфигурацией PHP, используя функцию `phpinfo()` в сценарии PHP. Для этого создайте новый файл *info.php* в папке сервера, добавьте в этот файл следующий код и сохраните изменения:

```
<?php
phpinfo();
```

Теперь, открыв в браузере файл *info.php*, вы должны увидеть страницу с таблицей, содержащей полную информацию о текущей конфигурации PHP, как показано на рис. 1.5.



System	
Build Date	Feb 25 2013 00:27:38
Compiler	MSVC9 (Visual C++ 2008)
Architecture	x64
Configure Command	cmd /c "cd /d %~dp0\php\configure && "enable-snapshot-build" && "without-dblib" && "without-pdo-dblib" && "without-pdo-mssql" && "with-pdo-oci=C:\php-sdk\php54dev\vc9\w64\deps\instantclient_10_2\sdk,shared" && "with-oci8=C:\php-sdk\php54dev\vc9\w64\deps\instantclient_10_2\sdk,shared" && "with-oci8-11g=C:\php-sdk\php54dev\vc9\w64\deps\instantclient_11_2\sdk,shared" && "disable-debug-pack" && "disable-static-analyze"
Server API	Apache 2.0 Handler
Virtual Directory Support	enabled
Configuration File (php.ini) Path	C:\windows
Loaded Configuration File	D:\wamp\bin\apache\apache2.4.4\bin\php.ini
Scan this dir for additional .ini files	(none)
Additional .ini files parsed	(none)
PHP API	20100412
PHP Extension	20100525
Zend Extension	220100525
Zend Extension	API220100525,TS,VC9

Рис. 1.5. Результат работы функции `phpinfo()`

Здесь важно найти строку **Loaded Configuration File** (Загруженный файл конфигурации), которая точно укажет местоположение файла *php.ini*. На рис. 1.5 показано, что файл *php.ini* находится по адресу *D:\wamp*

bin\apache\apache2.4.4\bin\php.ini. В вашем случае местоположение этого файла, вероятно, будет другим, поэтому для его выяснения воспользуйтесь функцией `phpinfo()`.

Файл *php.ini* содержит большое количество параметров, многие из которых вам, вероятно, не понадобятся. Однако некоторые из них могут быть изменены, чтобы обеспечить лучшее соответствие PHP вашему стилю разработки и используемой среде. Описание файла *php.ini* вы можете найти в моей статье *Tour of php.ini*¹.

Привет, мир PHP

Теперь, когда ваш локальный сервер установлен и работает, давайте приступим к написанию кода на языке PHP! Сначала перейдем в веб-папку Apache, местоположение которой зависит от используемой операционной системы.

- Пользователям операционной системы Windows и стека WAMP нужно запустить программу Проводник (Explorer) или щелкнуть мышью по значку **Мой компьютер** (My Computer) и перейти в папку, расположенную по адресу *C:\Program Files\wamp\www*
- Пользователям операционной системы OS X и стека MAMP нужно запустить программу Finder и перейти в папку *Applications/MAMP/htdocs*
- В операционной системе Linux папка находится по адресу */var/www*. Многие разработчики, использующие операционную систему Linux, предпочитают изменять файл конфигурации своего сервера, чтобы сменить папку, в которой хранятся веб-файлы. Чтобы узнать, как это делается, обратитесь к решению на сайте Stack Overflow².

После того как вы нашли веб-каталог своего сервера, необходимо создать корневой каталог для вашего проекта, где будут расположены все ваши файлы. Создайте каталог с именем *my_project*. Я специально не стал использовать пробелы и прописные буквы в имени папки, поскольку оно будет являться частью URL-адреса для доступа к содержащимся в ней файлам.

Теперь создайте новый PHP-файл с именем *index.php* в папке *my_project*. После этого вам необходимо открыть его в текстовом редакторе, в котором можно редактировать PHP-файлы.

¹ www.sitepoint.com/a-tour-of-php-ini
² stackoverflow.com/a/5891858



ВЫБОР ТЕКСТОВОГО РЕДАКТОРА

Я рекомендую использовать редактор, который предусматривает подсветку синтаксиса PHP. Выбор редактора — это довольно личное решение, поскольку все программы имеют свои особенности. Некоторые могут соответствовать вашему способу написания кода, а некоторые — противоречить ему. Так что вам следует выбрать редактор, который сделает процесс кодирования проще и интереснее. Также стоит попробовать несколько различных редакторов, прежде чем сделать окончательный выбор. Далее перечислены некоторые из наиболее популярных платных и бесплатных редакторов.

- Adobe Dreamweaver¹ (операционная система Windows и OS X).
Один из самых широко известных редакторов. Он предусматривает множество команд «в один клик» и встроенный FTP-клиент.
- Sublime Text² (операционная система Windows, OS X и Linux).
Платный редактор, предусматривающий бесплатный пробный период, а также встроенные сочетания клавиш и макросы.
- Komodo Edit³ (операционная система Windows, OS X и Linux).
Мощный бесплатный редактор с превосходным интерфейсом и встроенным FTP-клиентом.
- Coda⁴ (операционная система OS X).
Очень удобный редактор с широким спектром функциональных возможностей.
- Notepad++⁵ (операционная система Windows).
Очень популярный бесплатный редактор.

Выбрав подходящий редактор, откройте файл *index.php*. Введите следующий код, а затем сохраните файл, используя кодировку UTF-8:

```
<?php header("Content-Type: text/html;  
↵ charset=utf-8"); ?>  
<meta http-equiv="Content-Type" content="text/html;  
↵ charset=utf-8"/>  
<?php echo "Привет, Мир"; ?>
```

¹ www.adobe.com/products/dreamweaver.html

² www.sublimetext.com

³ www.activestate.com/komodo-edit

⁴ panic.com/coda

⁵ www.notepad-plus-plus.org

Давайте протестируем этот код. Откройте веб-браузер и в адресной строке введите адрес **http://localhost/my_project/index.php**. Вы должны увидеть нечто похожее на рис. 1.6.

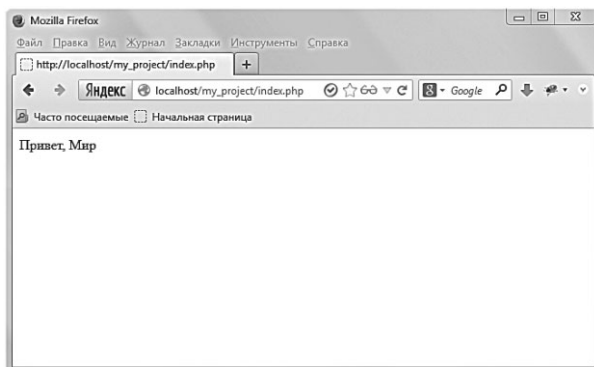


Рис. 1.6. Привет, мир!

ПРИМЕЧАНИЕ

Первые две строки приведенного выше кода отвечают за кодирование файла в разметке UTF-8. Эти строки нужны всегда, когда в коде используются символы, отличные от латиницы, в том числе и кириллические.

Поздравляем! Вы только что написали свой первый блок кода на языке PHP! Как видите, этот код отображает в веб-браузере фразу «Привет, Мир», но что именно вы ввели в своем PHP-файле? Давайте разберем нашу строку кода.

- `<?php`. Этот фрагмент кода называется открывающим тегом. Он сообщает вашему серверу о том, что следующий код следует интерпретировать как написанный на языке PHP и что для обработки этого кода сервер должен использовать движок PHP. Этот тег необходимо указывать каждый раз, когда вы хотите добавить в файл код на языке PHP.
- `echo "Привет, Мир";`. `echo` — это базовая конструкция PHP, означающая, что текст (содержимое), расположенный после команды `echo`, должен быть отображен в браузере. Текст, который необходимо отобразить на экране, заключается в кавычки. Кроме того, после текста вы добавляете точку с запятой, которая означает окончание выражения `echo`.

- `?>`. Это закрывающий тег, который приказывает серверу прекратить обработку кода PHP. Закрывающий тег используется не всегда. Оставление кода «незакрытым» может предотвратить непреднамеренную передачу контента браузеру. Опускание закрывающего тега, когда в нем нет необходимости, помогает избежать проблем, которые могут возникнуть при написании более сложного PHP-кода. Далее в этой книге вам будет встречаться код, в котором отсутствует закрывающий тег. Имейте в виду, что этот тег был пропущен специально.



ИСПОЛЬЗУЙТЕ ТОЧКУ С ЗАПЯТОЙ!

Важно не забывать добавлять точку с запятой в конце всех выражений PHP, поскольку в противном случае будет не ясно, где заканчивается одно выражение и начинается другое.

В PHP-файлах вы можете комбинировать PHP-код с обычным кодом HTML. Объединив коды PHP и HTML, вы можете создать действительно привлекательные веб-страницы, содержащие динамические и сложные данные, которые HTML не в состоянии обработать самостоятельно.

Ниже приведен небольшой пример совместной работы HTML и PHP.

```
<!DOCTYPE html>
<?php header("Content-Type: text/html;
↳ charset=utf-8");?>
<meta http-equiv="Content-Type" content="text/html;
↳ charset=utf-8"/>
<head>
<title>Совместная работа HTML и PHP</title>
</head>
<body>
<h1><?php echo 'Это тег H1 с данными PHP'; ?></h1>
<p><?php echo 'Это тег P с данными PHP'; ?></p>
</body>
</html>
```

Если вы замените содержимое файла *index.php* приведенным выше блоком кода и в адресной строке браузера введете адрес **http://localhost/my_project/index.php**, то убедитесь в правильном отображении кода, обработанного PHP.

Переменные языка PHP

Давайте переведем наш код на следующий уровень и исследуем мир переменных. Переменная представляет собой место в памяти компьютера для временного хранения битов информации. Мы можем поместить данные в переменную (в память) и использовать эту переменную в сценарии.

Каждый раз, когда интерпретатор PHP «видит» в нашем коде переменную, он знает, что ему нужно получить доступ к данным, которые она представляет. Данные, помещаемые в переменную, могут быть чем угодно — от простой строки текста, например «Привет, Мир», или чисел вроде «1234» до данных более сложной структуры, которые мы рассмотрим чуть позже.

Имена переменных должны начинаться с символа `$`, не содержать пробелов и состоять только из латинских букв (строчных и прописных), цифр и символов нижнего подчеркивания. Кроме того, первым символом в имени после `$` может быть только буква или символ подчеркивания. Приемлемым именем переменной может быть `$myAwesomeVariable` или даже `$my_Awesome_Variable_1`. Хорошим тоном считается присвоение переменным описательных имен, чтобы другой разработчик, читающий написанный вами код, понимал, какие данные представляет та или иная переменная. Например, если переменная используется для хранения чьего-то имени, то мы могли бы назвать ее `$personName`.



ИСПОЛЬЗУЙТЕ ГОРБАТЫЙ РЕГИСТР

Если имя переменной состоит более чем из одного слова, то обычно каждое слово начинается с заглавной буквы. Это улучшает читаемость кода при его беглом просмотре.

Давайте рассмотрим используемые переменные. Измените код в файле *index.php* следующим образом:


```
<?php header("Content-Type: text/html;  
↳ charset=utf-8");?>  
<meta http-equiv="Content-Type" content="text/html;  
↳ charset=utf-8"/>  
<?php  
$myVar = 'Привет, Мир! Используем переменную';  
echo $myVar;
```

Итак, что мы сделали в этом коде? Во-первых, мы присвоили значение (строку «Привет, Мир! Используем переменную») переменной `$myVar`. Поскольку здесь мы впервые используем эту переменную, она будет создана автоматически. Теперь, если мы напишем `$myVar` в любом месте сценария PHP, интерпретатор будет знать, что ему необходимо загрузить данные, которые мы поместили в эту переменную.

Если вы снова перезагрузите файл *index.php* в своем браузере, то на экране отобразится строка «Привет, Мир! Используем переменную».

Теперь изменим код в файле *index.php* так, чтобы он выглядел следующим образом:

```
<?php header("Content-Type: text/html;  
↳ charset=utf-8");?>  
<meta http-equiv="Content-Type" content="text/html;  
↳ charset=utf-8"/>  
<?php  
$myVar;  
$myVar = 'Привет, Мир! Используем переменную';  
echo $myVar;  
$myVar = 'Пока, Мир';  
echo $myVar;
```

После обновления страницы вы должны увидеть на экране обе фразы: «Привет, Мир! Используем переменную» и «Пока, Мир». Это связано с тем, что мы в любое время можем перезаписать данные в переменной.

В приведенном выше примере также показано, как интерпретатор PHP обрабатывает код строку за строкой сверху вниз. Наше первое выражение `echo` показывает первое предложение, поскольку в этот момент переменная `$myVar` содержит именно эту фразу. Тем не менее после первого выражения `echo` мы заменяем его вторым предложением. Важно помнить о правиле «сверху вниз» при планировании того, какие действия должен совершать сценарий PHP по отношению к определенным фрагментам данных в конкретных местах вашего сценария. Если вы перезапишете данные в переменной, прежде чем планируете ее использовать, то эти данные исчезнут и результат работы сценария будет неправильным.

В переменных PHP можно хранить не только строки и предложения. Давайте кратко рассмотрим различные типы данных, которые мы можем использовать:

```
<?php header("Content-Type: text/html;
↳ charset=utf-8");?>
<meta http-equiv="Content-Type" content="text/html;
↳ charset=utf-8"/>
<?php
$myVar = 0; //Целое число
$myVar = 3.14; //Число с плавающей точкой
$myVar = "Текущий год"; //Строка
$myVar = true; //Булева переменная
$myVar = array(250, 300, 325, 475); //Массив
```

В приведенном выше коде мы использовали широкий спектр типов переменных, поэтому давайте разберем их.

- **Integer** — целые числа. Они могут быть как положительными, так и отрицательными.
- **Float** — числа с плавающей точкой (числа, содержащие знаки после запятой). Они также могут быть положительными или отрицательными.
- **String** — строка (сочетание букв, цифр и символов). Строки заключаются в кавычки.

- Boolean — логическая переменная, принимающая одно из двух значений: `true` (истина) или `false` (ложь). Значение логической переменной не должно заключаться в кавычки, поскольку это превратит ее в строку.
- Array — массив. Многоуровневый тип переменной, подобный таблице.

Полное описание этих типов переменных, а также впечатляющие примеры их использования вы можете найти в следующих статьях и учебных пособиях:

- www.php.net/manual/ru/language.variables.basics.php
- www.sitepoint.com/variables/
- goldhat.ca/blog/php-beginner-lesson-using-variables

Массивы

Для более сложных способов хранения данных в памяти можно использовать массив, который представляет данные так же, как таблица.

Давайте начнем с таблицы, показанной на рис. 1.7.

Массив	Элемент1	Значение1
--------	----------	-----------

Рис. 1.7. Простой массив

Мы можем написать эту таблицу на языке PHP в виде массива следующим образом:

```
<?php header("Content-Type: text/html;  
↳ charset=utf-8");?>  
<meta http-equiv="Content-Type" content="text/html;  
↳ charset=utf-8"/>  
<?php  
$myArray = array('Элемент1' => 'Значение1');
```

В этом коде мы создали массив с именем `$myArray`, используя для присвоения значения конструкцию `array()`, а не простую строку или целое число. Затем мы добавили «ключ» с именем `Элемент1` и присвоили этому ключу значение `Значение1` с помощью символа `=>`. Символ `=>` используется в РНР для обозначения, что следующие за ним данные будут храниться в таком месте памяти, на которое можно сослаться из массива, используя данный ключ.

При определении массива мы можем добавить в него несколько пар ключ/данные, как показано далее:

```
<?php header("Content-Type: text/html;
↳ charset=utf-8");?>
<meta http-equiv="Content-Type" content="text/html;
↳ charset=utf-8"/>
<?php
$myArray = array('Элемент1' => 'Значение1', 'Элемент2'
↳ => 'Значение2');
```

Другим вариантом является создание пустого массива и дальнейшее добавление ключа и значения:

```
<?php header("Content-Type: text/html;
↳ charset=utf-8");?>
<meta http-equiv="Content-Type" content="text/html;
↳ charset=utf-8"/>
<?php
$myArray = array();
$myArray['Элемент1'] = 'Значение1';
$myArray['Элемент2'] = 'Значение2';
```

Начиная с версии РНР 5.4, появилась возможность использовать сокращенный (стенографический) вариант записи для создания массива вместо традиционной конструкции `array()`. Сокращенная запись выглядит следующим образом:

```
<?php header("Content-Type: text/html; charset=utf-8");?>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
<?php
$array = ['Элемент1' => 'Значение1', 'Элемент2' => 'Значение2'];
```

Массивы позволяют работать с очень сложными типами данных, и далее в книге мы к ним вернемся. Чтобы освоить основы работы с массивами, я рекомендую обратиться к следующим публикациям:

- www.php.net/manual/ru/language.types.array.php
- www.sitepoint.com/introduction-to-php-arrays/
- www.htmlandphp.com/beginner-php/207-introduction-to-arrays-in-php.html

Комментарии

Мы можем добавлять в PHP-код комментарии, которые не обрабатываются как выражения. Комментарий начинается с двух слешей (//). Все, что следует за ними на той же строке, рассматривается в качестве комментария и не обрабатывается. Например:

```
<?php
//это комментарий PHP только для данной строки
```

Мы также можем использовать символы /*, чтобы отметить начало комментария, расположенного на нескольких строках, и символы */, чтобы отметить его окончание. Например:

```
<?php
/*это начало нашего комментария PHP.
Он может занимать несколько строк.
Здесь он заканчивается.*/
```

С помощью комментариев разработчик может оставлять заметки, которые предназначены для любого человека, читающего код, позволяют понять, как он работает, и облегчают его сопровождение.

Комментарии также позволяют временно скрыть фрагменты кода (*закомментировать* их). Если бы мы поступили так с еще незавершенным кодом, это могло бы привести к возникновению ошибки при его обработке.

Подготовка проекта

В заключение этой главы давайте быстро создадим структуру каталогов для проекта, над которым будем работать на протяжении всей этой книги. Мы создадим простое блог-приложение, работа которого подразумевает использование только PHP и базы данных MySQL для хранения такой информации, как содержимое сообщения. Данное приложение будет состоять из внешнего интерфейса, отображающего список сообщений, и панели администратора для управления контентом нашего блога.

Сейчас нам нужно создать в веб-каталоге новую папку с именем *kickstart*. Внутри этой папки необходимо создать еще три каталога с именами *admin*, *frontend* и *includes*, как показано на рис. 1.8.

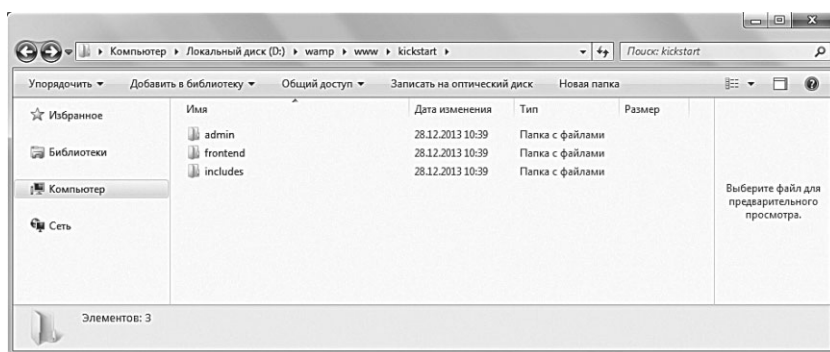


Рис. 1.8. Папки для нашего проекта

В каталоге *admin* будут храниться файлы, необходимые для функционирования панели администратора. Каталог *frontend* будет содержать файлы, обеспечивающие работу пользовательского интерфейса блога. Пользовательский интерфейс — это то, что увидят посетители при просмотре публикаций в блоге, и эти файлы будут отвечать за загрузку

данных блога для отображения в браузерах пользователей. В каталоге *includes* мы будем хранить сценарии, функциональность которых можно использовать в обоих разделах приложения.

В каталогах *admin* и *frontend* нам также нужно создать папки с именем *templates*. Данные папки будут содержать все файлы на языке PHP и HTML, которые станут принимать данные из сценариев и файлов, обеспечивающих функциональность приложения. Эти файлы шаблонов будут отвечать за форматирование данных и создание макетов, с которыми взаимодействуют посетители. Те же файлы шаблонов будут создавать макеты, которые мы станем использовать для управления блогом в разделе администрирования. Далее в книге мы более подробно поговорим о шаблонах и узнаем, почему мы используем их в работе над нашим проектом.

Такое разделение папок позволяет упростить верстку кода и улучшить его организацию.

Резюме

В этой главе мы усвоили основы настройки веб-сервера, познакомились с кодированием на языке PHP, отобразив на экране текст, и использовали переменные для хранения данных PHP. Все, что содержится в этой главе, является вводной информацией, и по мере чтения книги мы будем возвращаться к некоторым описанным здесь темам.

В следующей главе мы узнаем о том, как можно применять варианты постоянного хранения информации, такие как базы данных, с нашими сценариями, а также о том, как мы можем использовать браузер посетителя для хранения небольших объемов данных.

Глава 2

PHP И ДАННЫЕ

Данные — это источник жизненной силы PHP. В этой главе мы рассмотрим один из множества предусматриваемых PHP вариантов обработки и постоянного хранения данных — взаимодействие с базой данных MySQL. Кроме того, мы кратко рассмотрим некоторые важные понятия программирования, например условные операторы и циклы, которые необходимы при считывании информации из базы данных. Так что приготовьтесь и засучите рукава, поскольку мы собираемся погрузиться в мир обработки данных PHP!

Операторы

В PHP, как и практически во всех языках программирования, *операторы* используются для манипулирования или выполнения различных действий над переменными и значениями. Самым простым оператором является `=`, который, как мы видели в главе 1, используется для присвоения значения переменной. Однако существует множество других мощных операторов.

Для начала создадим новый файл с именем *op_experiments.php* в папке *my_project*, которую мы создали ранее. В этот файл нужно добавить следующий код:

```
<?php header("Content-Type: text/html;  
↳ charset=utf-8");?>  
<meta http-equiv="Content-Type" content="text/html;
```



```
↵charset=utf-8"/>
<?php
$firstName = "имя";
$lastName = "фамилия";
$fullName = $firstName . $lastName;
echo $fullName;
```

Хотя большая часть этого кода, вероятно, кажется довольно понятной, строка `$fullName = $firstName . $lastName;` может вызвать недоумение, особенно точка между двумя переменными.

Точка является оператором конкатенации в языке PHP, она соединяет вместе две строки, чтобы получилась одна длинная строка. Интерпретатор PHP берет строковое значение переменной `$firstName`, добавляет к нему значение переменной `$lastName` и сохраняет результат в переменной `$fullName`. При запуске сценария PHP выводит значение переменной `$fullName`, и в браузере отображается строка `myfirstnamemylastname`.

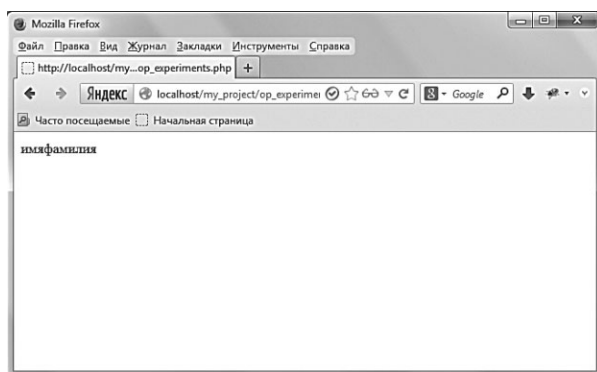


Рис. 2.1. Результат нашего эксперимента

Однако это значение читается с трудом, не так ли? Не волнуйтесь. Мы можем легко подправить код, чтобы улучшить восприятие результата. Внесите изменения, чтобы код выглядел следующим образом:

```
<?php header("Content-Type: text/html;
↵charset=utf-8");?>
```

```
<meta http-equiv="Content-Type" content="text/html;
↳ charset=utf-8"/>
<?php
$firstName = "Имя";
$lastName = "фамилия";
$fullName = $firstName . " - " . $lastName;
echo $fullName;
```

Теперь при просмотре в браузере мы видим, что два имени отделены друг от друга. Отлично.

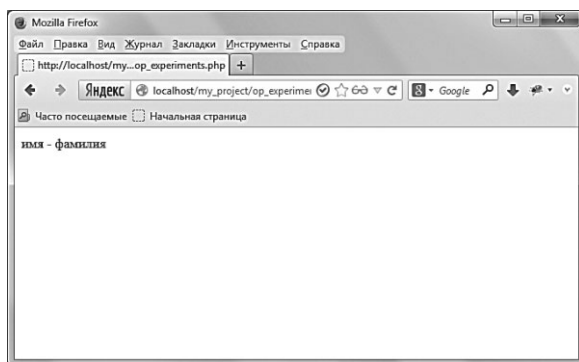


Рис. 2.2. Результат после добавления пробелов

Условные операторы

Язык PHP также предусматривает операторы, которые позволяют сравнивать значения. Например, оператор `==` проверяет равенство двух значений, а операторы `<` и `>` позволяют узнать, является ли одно значение меньшим или большим, чем другое. В большинстве случаев эти операторы сравнения используются при написании *условных операторов* (выражений, которые выполняются или не выполняются в зависимости от результата сравнения) и *циклов* (выражений, которые выполняются многократно).

Оператор `if`

Оператор `if` состоит из условия и одного или нескольких операторов, сгруппированных в виде блока. Если интерпретатор PHP оценивает вы-

ражение как истинное, то он выполняет блок операторов. Если условие ложно, то данный блок будет пропущен. В приведенном ниже примере интерпретатор PHP сравнивает значение переменных `$a` и `$b`. Если они равны, то он выводит на экран строку «А равно В»:

```
<?php header("Content-Type: text/html;  
↳ charset=utf-8");?>  
<meta http-equiv="Content-Type" content="text/html;  
↳ charset=utf-8"/>  
<?php  
if ($a == $b) {  
echo 'А равно В';  
}
```

Оператор `else`

Оператор `if` дополняет оператор `else`, который может следовать непосредственно после него. Интерпретатор PHP выполняет код, который составляет блок оператора `else`, только в том случае, если пропускается выполнение оператора `if`. В приведенном ниже примере интерпретатор PHP сравнивает значение переменных `$a` и `$b`. Если они равны, то он выводит на экран «А равно Б», а если не равны, то выполняется код в блоке, следующем за оператором `else`, и на экран выводится «А не равно Б».

```
<?php header("Content-Type: text/html;  
↳ charset=utf-8");?>  
<meta http-equiv="Content-Type" content="text/html;  
↳ charset=utf-8"/>  
<?php  
if ($a == $b) {  
echo 'А равно Б';  
} else {  
echo 'А не равно Б';  
}
```

Оператор elseif

Еще одним дополняющим оператором является `elseif`. Как следует из его имени, оператор `elseif` представляет собой сочетание операторов `if` и `else`. Он подразумевает наличие условия, и интерпретатор PHP проверяет это условие в том случае, если условие оператора `if` оказывается ложным.

Можно использовать несколько операторов `elseif`, чтобы проверить различные условия, интерпретатор PHP будет проверять каждое из них, пока одно не окажется истинным.

После определения истинного условия остальные операторы `elseif` в цепочке пропускаются.

```
<?php header("Content-Type: text/html;
↳ charset=utf-8");?>
<meta http-equiv="Content-Type" content="text/html;
↳ charset=utf-8"/>
<?php
if ($a == $b) {
echo 'A равно B';
} elseif ($a == $c) {
echo 'A равно C';
} elseif ($a == $d) {
echo 'A равно D';
} else {
echo 'A ничему не равно';
}
```

Оператор switch

Если вы внимательно рассмотрите наш пример с оператором `elseif`, вы заметите, что каждое условие, по сути, проверяет одно и то же — является ли значение некоторой переменной равным значению переменной `$a`. Оператор `switch` работает подобным образом, но выглядит более аккуратно.

```
<?php header("Content-Type: text/html;  
↳ charset=utf-8");?>  
<meta http-equiv="Content-Type" content="text/html;  
↳ charset=utf-8"/>  
switch ($a) {  
case $b:  
echo 'A равно B';  
break;  
case $c:  
echo 'A равно C';  
break;  
case $d:  
echo 'A равно D';  
break;  
default:  
echo 'A ничему не равно';  
break;  
}
```

В приведенном выше примере интерпретатор PHP берет значение, указанное в начале оператора `switch` (значение переменной `$a`) и сравнивает его со значением каждого оператора `case`. Выполняется фрагмент кода, содержащий соответствующее значение. Если совпадений не найдено, выполняется оператор `default`.

После нахождения соответствия интерпретатор PHP начинает выполнять расположенный далее код вплоть до ключевого слова `break`, после которого он переходит в конец оператора `switch` и приступает к выполнению остальной части сценария.



СЛЕДИТЕ ЗА КЛЮЧЕВЫМ СЛОВОМ `BREAK`

Интерпретатор PHP не воспринимает начало одного оператора `case` в качестве окончания предыдущего оператора `case`, именно поэтому необходимо ключевое слово `break`. Поначалу вы можете путаться и, вероятно, временами будете забывать использовать

в своих сценариях ключевое слово `break`, из-за чего они станут работать не так, как вы ожидаете, поэтому имейте это в виду.

Это также означает, что мы можем написать код, который выполняется для нескольких операторов `case`:

```
<?php header("Content-Type: text/html;
↳ charset=utf-8");?>
<meta http-equiv="Content-Type" content="text/html;
↳ charset=utf-8"/>
switch ($a) {
case $b:
case $c:
case $d:
echo 'A равно B, C, или D';
break;
default:
echo 'A ничему не равно';
}
```

Условные операторы позволяют создавать динамические сценарии, которые выполняются по-разному в зависимости от обстоятельств. Я рекомендую вам попрактиковаться в написании собственных условных операторов и прочитать официальную документацию PHP¹.

Циклы

Циклы PHP позволяют записать один набор выражений для многократного выполнения. Например, предположим, что нам нужно отобразить числа от 1 до 100. Конечно, мы могли бы написать 100 строк кода, в каждой из которых использовалась бы конструкция `echo` для вывода следующего числа. Однако удобнее использовать цикл, который многократно выполняет одну конструкцию `echo`, но по отношению к числу, которое каждый раз становится на единицу больше, пока не будут отображены все 100 чисел.

Цикл `for`

Цикл `for` — это общий механизм циклов, в котором переменная используется для отслеживания процесса его выполнения. Данный цикл состо-

¹ www.php.net/manual/ru/language.control-structures.php

ит из четырех основных частей: инициализации переменной (установка переменной для отслеживания, или счетчика, на известное начальное значение), условия (интерпретатор PHP продолжает выполнение цикла до тех пор, пока данное условие остается истинным), инкремента или декремента (который изменяет значение счетчика в соответствии с прогрессом цикла) и многократно выполняемого блока операторов.

Хотя это может показаться сложным, код цикла на самом деле довольно лаконичен, как показывает следующий пример:

```
for ($i = 1; $i < 101; $i = $i + 1) {  
echo $i . '<br>';  
}
```

Так как же интерпретатор PHP выполняет этот код? Сначала он присваивает переменной `$i` значение 1. Затем он оценивает условие, определяет, что 1 действительно меньше, чем 101, и выполняет блок операторов цикла.

В нашем примере этот блок содержит лишь конструкцию `echo`, которая выводит «1», за ней следует HTML-тег разрыва строки. После выполнения всех операторов в блоке интерпретатор PHP возвращается в начало цикла `for` для выполнения третьей части: `$i = $i + 1`. Это приводит к изменению значения нашей переменной для отслеживания на 2. Условие проверяется снова, 2 меньше 101, поэтому блок операторов выполняется повторно. На этот раз конструкция `echo` выводит значение «2».

Интерпретатор PHP продолжает выполнять цикл до тех пор, пока значение переменной `$i` не достигнет 101 (что сделает условие ложным, поскольку при этом значение переменной `$i` становится равно, а не меньше 101), а затем выходит из цикла.

Цикл `while`

Циклы `while` работают почти так же, как циклы `for`, за исключением того, что им требуется условие лишь в начале. Мы должны до начала цикла установить начальное значение переменной для отслеживания, которое будет корректироваться в блоке операторов.

```
$i = 1;
while ($i < 101){
echo $i . '<br>';
$i = $i + 1;
}
```

Сначала в нашем первом выражении интерпретатор PHP присваивает переменной `$i` значение 1, а затем запускает выполнение цикла `while`. Он проверяет условие (которое оказывается истинным) и приступает к выполнению двух операторов, составляющих тело цикла. Значение переменной `$i` выводится, а затем увеличивается, готовясь к следующему выполнению цикла.



ПРЕДУСМОТРИТЕ «ПЛАН ПОБЕГА»

Часто при работе с циклами разработчики забывают предусмотреть возможность изменить значение счетчика. Если значение не меняется, то условие всегда остается истинным и цикл никогда не перестает выполняться! При создании цикла следует удостовериться в том, что он в какой-то момент остановится.

Цикл `foreach`

В отличие от циклов `for` и `while`, в которых используется конкретное условие, цикл `foreach` берет переменную с несколькими ключами (например, массив) и применяет к каждому элементу набор операторов.

```
$myArray = array('Привет', 'Мир');
foreach ($myArray as $value) {
echo $value . '<br>';
}
```

Интерпретатор PHP присваивает значение каждого элемента массива `$myArray` переменной `$value`, что позволяет применить к нему операторы, составляющие тело цикла.

Кроме того, мы можем получить доступ к ключу текущего элемента и к его значению следующим образом:

```
$myArray = array(1 => 'Привет', 2 => 'Мир');  
foreach ($myArray as $key => $value) {  
    echo $key . ': ' . $value . '<br>';  
}
```

Более подробную информацию о циклах `while`, `for` и `foreach` вы можете получить, обратившись к официальной документации PHP¹, а также к следующим ресурсам:

- www.sitepoint.com/loops/
- webcheatsheet.com/PHP/loops.php

Базы данных, MySQL и PHP

При написании веб-приложения на языке PHP нам часто требуется, чтобы некоторые данные, созданные в нашем сценарии, где-то сохранялись для получения к ним доступа в будущем.

База данных позволяет хранить большие объемы различных типов данных: от строк, содержащих буквы и цифры, до сериализованных массивов.

MySQL является самой популярной базой данных с открытым исходным кодом, и она отлично подходит для хранения данных нашего приложения. Она очень легко настраивается и требует лишь нескольких функций для хранения и извлечения данных.

Такие традиционные базы данных, как MySQL, для организации данных используют таблицы, строки и столбцы.

Каждая таблица содержит набор записей (информация о пользователе, содержание заказов и т. д.), каждая строка таблицы хранит конкретную запись, а каждый столбец определяет другой фрагмент данных, составляющих эту запись.

Таблицу базы данных можно изобразить так, как показано на рис. 2.3.

¹ www.php.net/manual/ru/language.control-structures.php

Строка 1 Столбец 1	Строка 1 Столбец 2	Строка 1 Столбец 3
Строка 2 Столбец 1	Строка 2 Столбец 2	Строка 2 Столбец 3
Строка 3 Столбец 1	Строка 3 Столбец 2	Строка 3 Столбец 3

Рис. 2.3. Таблица базы данных



ПОДУМАЙТЕ О ТИПАХ ДАННЫХ

Поскольку каждый столбец таблицы может хранить данные только одного типа, который часто используется для определения столбца, важно заранее подумать о различных свойствах данных, которые вы собираетесь хранить.

Прежде чем продолжить, вам следует познакомиться с целями базы данных и основами MySQL. Если вы не разбираетесь в этих темах, пожалуйста, прочитайте статью о MySQL на сайте tizag.com¹.

Существует два расширения для работы с MySQL в PHP: MySQLi и PDO. Мы будем использовать расширение PDO для подключения и взаимодействия с нашей базой данных MySQL, поэтому убедитесь в том, что это расширение установлено и включено. Откройте файл *info.php*, который мы создали в главе 1, найдите раздел под названием **PDO** и убедитесь, что база данных MySQL указана в столбце **enabled** (включено).



ПРОБЛЕМЫ ПРИ УСТАНОВКЕ РАСШИРЕНИЯ PDO

Если у вас возникли проблемы с включением расширения, обратитесь к руководству по установке PDO².

Интерфейс phpMyAdmin — это очень популярное PHP-приложение, упрощающее управление несколькими базами данных. В связи с этим настоятельно рекомендуется установить phpMyAdmin на ваш сервер. Мы будем использовать это приложение для управления базой данных на протяжении всей книги.

Сначала нам нужно создать новую пустую базу данных с помощью phpMyAdmin. Для этого откройте phpMyAdmin в своем браузере и вой-

¹ www.tizag.com/mysqlTutorial

² www.php.net/manual/ru/pdo.installation.php

дите в систему. Вы должны увидеть страницу, похожую на изображенную на рис. 2.4:

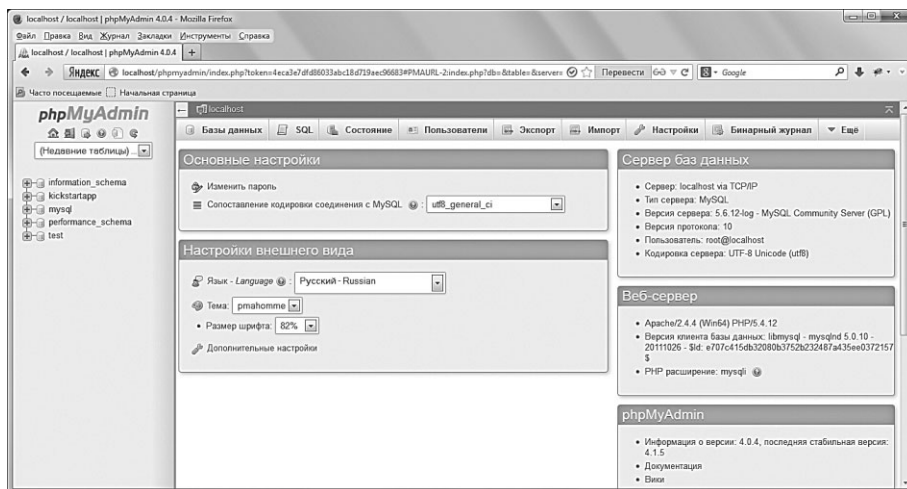


Рис. 2.4. Интерфейс phpMyAdmin

ПРИМЕЧАНИЕ

При первой авторизации в системе phpMyAdmin используйте логин root без пароля.

ПРИМЕЧАНИЕ

Для корректного отображения кириллических символов следует добавить строку `character-set-server = utf8` в раздел `[mysqld]` файла конфигурации `my.ini`, который можно открыть, щелкнув левой кнопкой мыши по значку **WAMP** в области уведомлений и выбрав пункт **my.ini** в меню **MySQL**. После сохранения изменений в файле конфигурации необходимо перезапустить службу **MySQL**, выбрав команду меню **MySQL** ⇒ **Service** ⇒ **Restart Service** (MySQL ⇒ Служба ⇒ Перезапустить службу).

Щелкните по ссылке **Базы данных** (Databases) в верхнем меню. На следующем экране мы можем создать новую базу данных, указав название, которое собираемся использовать.

Для целей нашего проекта введите «kickstartapp» в текстовое поле и нажмите кнопку **Создать** (Create).

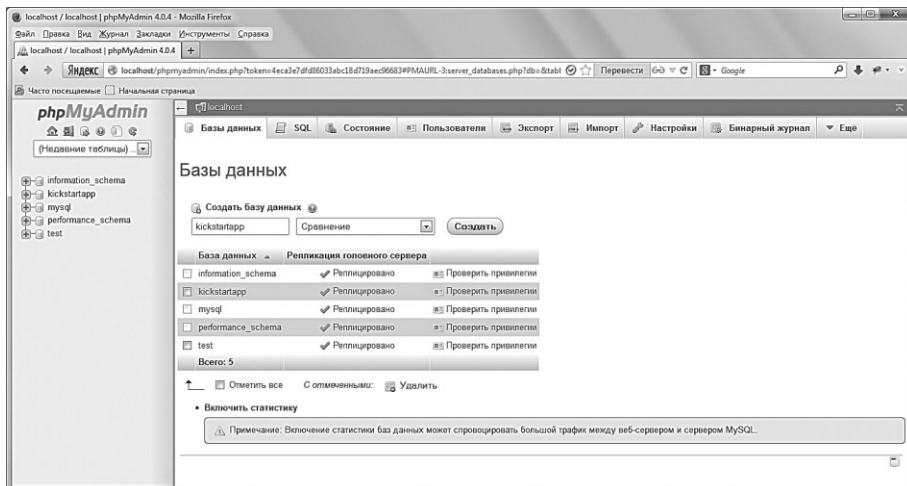


Рис. 2.5. Создание базы данных



СРАВНЕНИЕ

Не обращайте внимания на раскрывающийся список с выбранным пунктом **Сравнение** (Collation). Это более продвинутая функция, и для целей проекта мы будем использовать параметры по умолчанию. Если вы хотите подробнее узнать об этой функции, обратитесь к статьям:

- dev.mysql.com/doc/refman/5.0/en/charset-collation-implementations.html
- kb.mediatemple.net/questions/138/Default+MySQL+character+set+and+collation#gs



БУДЬТЕ ВНИМАТЕЛЬНЫ, ПРИСВАИВАЯ ИМЕНА БАЗАМ ДАННЫХ

Имена баз данных не должны содержать кириллицы, слешей, точек, пробелов или других символов, которые не допускаются в именах файлов.

Внимательный читатель, вероятно, заметил, что созданная нами база данных была добавлена в список доступных. Этот список в меню слева представляет собой коллекцию быстрых ссылок, которые мы можем использовать для того, чтобы выбрать базу данных для редактирования или управления. Найдите в списке новую базу данных `kickstartapp` и щелкните по ней. После этого вы попадете на новую довольно пустую страницу (см. рис. 2.6):

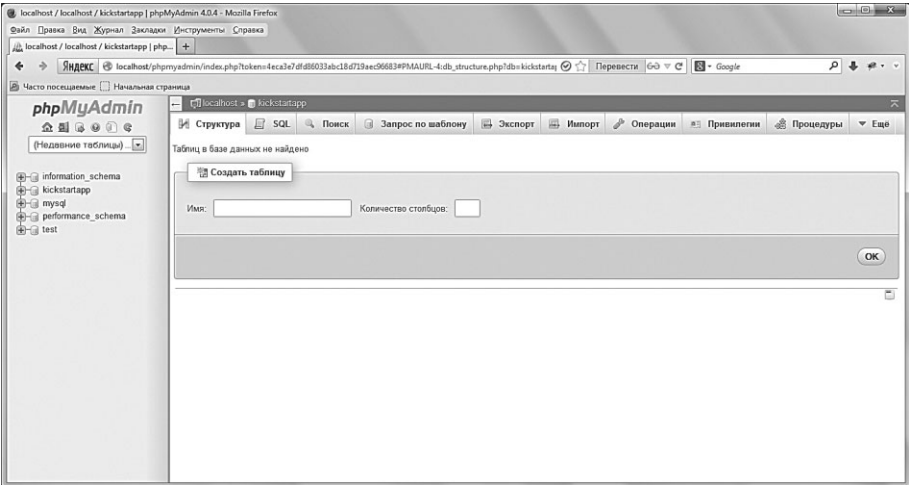


Рис. 2.6. База данных создана

Это главная рабочая область приложения phpMyAdmin, где мы можем создавать новые и редактировать существующие таблицы нашей базы данных. Однако прежде чем мы это сделаем, необходимо создать учетную запись, которая будет использоваться для подключения к базе данных, а также для хранения и извлечения из нее информации.

Перейдите на вкладку **Привилегии** (Privileges) в верхнем меню и на открывшейся странице щелкните по ссылке **Добавить пользователя** (Add User) как показано на рис. 2.7.

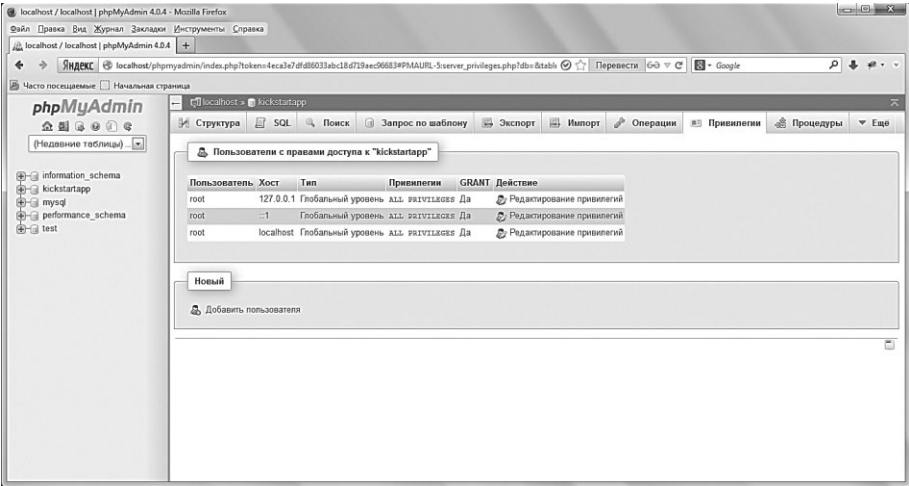


Рис. 2.7. Добавление нового пользователя

Теперь нам нужно ввести имя пользователя в поле **Имя пользователя** (User name) (убедитесь в том, что в раскрывающемся списке выбран пункт **Использовать текстовое поле** (Use text field)). Выберите вариант **Локальный** (Local) в меню **Хост** (Host), а затем введите пароль в поля **Пароль** (Password) (снова убедитесь в том, что в раскрывающемся списке выбран пункт **Использовать текстовое поле** (Use text field)). Затем прокрутите страницу и убедитесь в том, что в следующей области панели установлен флажок **Выставить полные привилегии на базу данных kickstartapp** (Grant all privileges on database kickstartapp). Наконец, выберите вариант **Отметить все** (Check all) в разделе **Глобальные привилегии** (Global privileges) в заключительной панели и нажмите кнопку **ОК**.

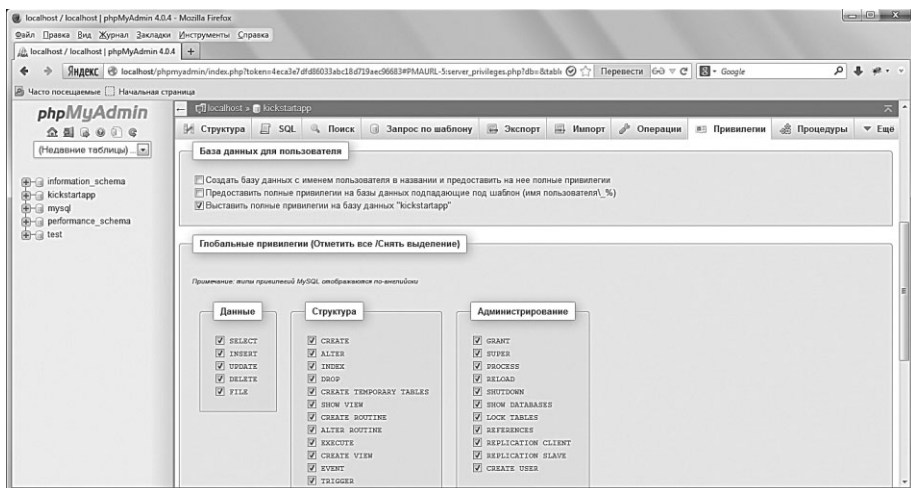


Рис. 2.8. Параметры для нового пользователя

Это может показаться немного сложным, однако мы всего лишь создали нового пользователя и затем предоставили ему право делать с базой данных kickstartapp все, что угодно. Такая комбинация параметров является наиболее распространенной среди пользователей баз данных, но не самой безопасной. При желании вы можете ограничить пользовательские привилегии. Для получения дополнительной информации о настройке параметров пользователя базы данных обратитесь к ресурсам:

- dev.mysql.com/doc/refman/5.1/en/adding-users.html
- kb.mediatemple.net/questions/788/HOWTO%3A+GRANT+privileges+in+MySQL#dv

Теперь, когда мы настроили учетную запись пользователя, давайте создадим таблицу в нашей базе данных. Создайте новый сценарий PHP с именем *setup.php* в папке проекта. Добавьте в этот файл следующий код, а затем сохраните изменения и откройте файл в браузере.



НАСТРОЙТЕ СВОЙ КОД

Замените текст `USERNAME` и `PASSWORD` в следующем фрагменте кода именем пользователя и паролем, которые вы указали при создании своей пользовательской базы данных.

```
<?php
$db = new PDO("mysql:host=localhost;dbname=
↳kickstartapp",
"USERNAME", "PASSWORD");
$db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_
↳EXCEPTION);
try {
$queryStr = "CREATE TABLE users (id INTEGER NOT NULL
AUTO_INCREMENT PRIMARY KEY, name VARCHAR(40), password
VARCHAR(100), email VARCHAR(150))";
$db->query($queryStr);
} catch (PDOException $e) {
echo $e->getMessage();
}
```

Сначала мы подключаемся к MySQL через PHP, создав объект PDO с четырьмя ключевыми фрагментами данных. Первый сообщает PHP о том, где находится наша база данных MySQL.

В нашем случае это `localhost`, поскольку сервер MySQL работает в той же системе, что и PHP. Второй фрагмент определяет, к какой базе данных мы хотим подключиться. Эти два фрагмента данных комбинируются с частью `mysql:` для создания того, что называется *DSN* или Data Source Name (Имя источника данных). Третий и четвертый аргументы представляют собой созданные нами имя пользователя и пароль для пользователя базы данных.

Далее мы указываем для объекта PDO вариант обратной связи, при котором любые ошибки в базе данных будут обрабатываться PHP в виде исключений. Это означает, что если что-то пойдет не так при попытке выполнения предоставленного нами SQL-кода базой данных MySQL, мы будем немедленно предупреждены о случившемся и сможем внести необходимые корректировки.

Далее мы определяем SQL-выражение, которое создаст таблицу и присвоит переменной `$queryStr` строковое значение. С помощью кода SQL мы даем MySQL инструкцию создать новую таблицу под названием `users`, которая будет использоваться для хранения всех данных, относящихся как к администраторам, так и к обычным пользователям нашего блог-приложения. Это делается с помощью команды `CREATE TABLE`, за которой следуют определения различных столбцов таблицы: `id`, `name`, `password` и `email`.

Определение столбца сообщает MySQL, данные какого типа будут находиться в этом столбце, а также содержит дополнительные параметры ячейки. Например, первый столбец (`id`) будет содержать только целые числа, и для конкретной записи никогда не будет разрешено его пустое значение (`NOT NULL`).

Далее следует тип условного оператора, который мы еще не обсуждали: блок `try...catch`. В блоке `try` мы передаем переменную `$queryStr` методу `query()` объекта PDO. Этот метод принимает нашу строку SQL и отправляет ее на сервер MySQL для выполнения.

PHP выполняет код в блоке `catch` только в том случае, если в блоке `try` возникла проблема. То есть PHP «выбросит» исключение, если MySQL не сможет выполнить наш запрос (мы определили такое поведение ранее с помощью фрагмента `PDO::ATTR_ERRMODE`). Наш блок `catch` будет «ловить» объект исключения, содержащий информацию, которую мы можем использовать для устранения неполадки.

Откройте файл *setup.php* в браузере. Если экран остается пустым, значит, запрос выполнен успешно (мы дали PHP инструкцию выводить предупреждения только в случае возникновения проблемы).

Теперь вернитесь в phpMyAdmin и щелкните по базе данных, чтобы убедиться в существовании вновь созданной таблицы.

Теперь давайте добавим данные в нашу таблицу! Создайте новый файл с именем *insert.php* в каталоге проекта и добавьте в него следующий код:


```
<?php
$db = new PDO("mysql:host=localhost;dbname=
↳kickstartapp",
"USERNAME", "PASSWORD");
$db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_
↳EXCEPTION);
try {
$queryStr = "INSERT INTO users (name, password, email)
VALUES ('admin', MD5('admin'), 'youremail@domain.
↳com') ";
$db->query($queryStr);
} catch (PDOException $e) {
echo $e->getMessage();
}
```

Большая часть кода остается такой же, как и раньше, за исключением того, что теперь мы посылаем базе данных MySQL команду INSERT INTO. Мы просим MySQL создать новую запись в таблице users с именем пользователя admin, хэшем пароля и адресом электронной почты.



ИСПОЛЬЗОВАНИЕ ФУНКЦИИ MD5 ()

Обратите внимание на то, что для столбца password мы использовали функцию MD5 (), чтобы хэшировать данные. Это сделано из соображений безопасности, чтобы в случае несанкционированного доступа к базе данных злоумышленники не могли легко получить пароли наших пользователей. Никогда не храните пароли в виде обычного текста!

Перейдите к файлу *insert.php*. Если запрос выполнен успешно, то вы снова увидите пустую страницу, в противном случае появится сообщение об ошибке с указанием того, что пошло не так. Вернитесь в phpMyAdmin и взгляните на таблицу users в базе данных kickstartapp, щелкнув по названию таблицы. Вы увидите, что в таблицу были добавлены данные.

Наконец, давайте извлечем данные из таблицы с помощью другого сценария PHP. Создайте файл *getdata.php* и добавьте в него следующий код:

```
<?php
$db = new PDO("mysql:host=localhost;dbname=
↳kickstartapp",
"USERNAME", "PASSWORD");
$db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_
↳EXCEPTION);
try {
$queryStr = "SELECT * FROM users";
$query = $db->prepare($queryStr);
$query->execute();
while ($row = $query->fetch()) {
echo $row['id'] . ' - ' . $row['name'] . ' - ' .
$row['email'] . ' - ' . $row['password'];
echo '<br>';
}
$query->closeCursor();
} catch (PDOException $e) {
echo $e->getMessage();
}
```

Опять же, большая часть кода должна казаться знакомой, однако на этот раз мы посылаем MySQL команду `SELECT`, это означает, что мы будем получать данные из базы данных. Эта команда, в том виде, как она написана, выделяет все столбцы всех строк в таблице `users`.

Метод `prepare()` подготавливает запрос для отправки в базу данных и возвращает объект запроса, который мы присвоили переменной `$query`. Этот объект запроса имеет методы, которые мы можем использовать для получения доступа к записям, возвращаемым MySQL. Затем метод `execute()` дает инструкцию расширению PDO отправить запрос MySQL.

При каждом вызове метод `fetch()` объекта запроса возвращает из базы данных одну запись в виде массива (каждый из столбцов становится элементом массива, а имена столбцов используются в качестве ключей для получения доступа к данным). Когда больше не остается записей, метод `fetch()` возвращает логическое значение `false`.

Каждый раз при выполнении цикла `while` мы присваиваем переменной `$row` запись базы данных, возвращаемую методом `fetch()`, и результат этого присвоения используется в качестве условия цикла. Когда запись возвращается, PHP оценивает условие как истинное и выполняет операторы цикла. Когда метод `fetch()` возвращает значение `false` в связи с отсутствием записей для обработки, PHP оценивает условие как ложное и прекращает выполнение цикла.

Наконец, мы вызываем метод `closeCursor()`, который освобождает соединение между PHP и MySQL для других запросов, и память, используемую PHP для получения набора результатов. Этот метод требуется не всегда, однако его применение является хорошей практикой, и я настоятельно рекомендую использовать его каждый раз, когда вы прекращаете прием данных от запроса с помощью расширения PDO.

Если мы откроем файл *getdata.php* в нашем браузере, то мы должны увидеть все данные, содержащиеся в таблице `users`. При появлении ошибки в коде PHP или в запросе SQL, отправленном в MySQL, вы увидите сообщение, которое подскажет, как решить возникшую проблему.

Резюме

В этой главе мы научились создавать базы данных с помощью приложения phpMyAdmin, таблицы в PHP с помощью команд MySQL, добавлять записи и извлекать данные из таблицы. Однако это лишь вершина айсберга. MySQL и PHP позволяют делать гораздо больше. Для получения дополнительной информации о MySQL и использовании MySQL с PHP обратитесь к следующим ресурсам:

- dev.mysql.com/doc/
- www.sitepoint.com/migrate-from-the-mysql-extension-to-pdo/
- www.php.net/manual/ru/book.pdo.php
- www.sitepoint.com/avoid-the-original-mysql-extension-1/

Глава 3

ОБЪЕКТЫ И ПАРАДИГМА ООП

В этой книге мы будем использовать парадигму *Объектно-ориентированное программирование* (ООП). ООП позволяет нам создавать «объекты», сочетающие данные и функции, применяемые к этим данным. Поскольку объекты в ООП собирают данные и применяемые к ним функции в одном месте, мы можем обеспечить хорошую организацию нашего кода и лучше справиться с растущей сложностью по мере разработки нашего приложения.

Код, который мы создаем для определения объекта, называется *классом*. Чтобы лучше понять взаимосвязь между классами и объектами, представьте себе строителя дома. Строитель (в нашем случае РНР) следует плану (класс), чтобы построить дом (объект).

Так же, как строитель может построить несколько домов, используя один и тот же план, так и РНР может создать несколько экземпляров объекта согласно одному классу. Все дома имеют одну и ту же планировку, но могут быть окрашены в разные цвета, и в них, очевидно, будут проживать разные семьи. В РНР каждый объект одного и того же класса имеет одинаковую функциональность, однако свойства и «живущие» в нем данные являются уникальными для конкретного экземпляра.

На первый взгляд парадигма ООП может показаться сложной, однако на самом деле она представляет собой очень простой подход к программированию. За дополнительной информацией, касающейся теории ООП, вы можете обратиться к вводу видеоуроку Лорны Митчелл на сайте sitepoint.com¹.

¹ www.sitepoint.com/object-oriented-php-lesson-1

Первые шаги в ООП

Давайте напишем небольшой пример ООП. Предположим, нам нужно представить собаку в качестве объекта в нашем PHP-коде. Нам потребуется создать определение объекта (класс), описывающее, что может делать собака.

Создайте новый файл с именем *Dog.php* в папке *experiments* со следующим содержимым:

```
<?php header("Content-Type: text/html;
↳ charset=utf-8");?>
<meta http-equiv="Content-Type" content="text/html;
↳ charset=utf-8"/>
<?php
class Dog
{
public $name;

public function __construct($name) {
$this->name = $name;
}

public function speak() {
return 'Гав! Гав!';
}
}
```

Давайте создадим еще один файл, с именем *DogTest.php*, в папке *experiments*.

```
<?php header("Content-Type: text/html;
↳ charset=utf-8");?>
<meta http-equiv="Content-Type" content="text/html;
↳ charset=utf-8"/>
```

```
<?php
require 'Dog.php';
$dog = new Dog('Фидо');
echo 'Собаку зовут: ' . $dog->name . '<br>';
echo 'Собака говорит: ' . $dog->speak() . '<br>';
```

Открыв файл *DogTest.php* в своем браузере, вы должны увидеть на экране имя собаки и ее «лай». Отлично.



ВКЛЮЧЕНИЕ КОДА ИЗ ДРУГИХ ФАЙЛОВ

Конструкция `require` в вышеприведенном примере, если говорить кратко, импортирует содержимое внешнего PHP-файла в наш файл *DogTest.php*. Это одна из четырех конструкций, используемых для включения кода из других файлов, каждая из которых имеет свои особенности.

- Конструкция `include` включает содержимое файла. Если файл не найден или к нему нет доступа, PHP выведет предупреждение, но продолжит выполнять код.
- Конструкция `include_once` аналогична конструкции `include`, однако PHP выполняет дополнительную проверку, чтобы убедиться в том, что файл не был импортирован ранее. Если это так, PHP не будет повторно включать содержимое.
- Конструкция `require` подобна конструкции `include`, однако если файл не будет найден, PHP остановит выполнение, сообщив о фатальной ошибке.
- Конструкция `require_once` представляет собой то же, что и `require`, однако осуществляет дополнительную проверку, чтобы не импортировать содержимое более одного раза.

Если мы посмотрим на наш файл *Dog.php*, то первое, что мы заметим, это ключевое слово `class`. Данное ключевое слово означает, что идущий следом код определяет класс, который будет известен под именем, следующим за ним (в нашем случае, `Dog`). После определения класса мы можем создать на его основе объект, используя фрагмент `new Dog`.

Методы `__construct()` и `speak()` являются частью определения класса. `__construct()` — это имя, которое имеет для PHP особое значение. При создании нового объекта на основе класса PHP выясняет, не

определен ли метод `__construct()`. Если определен, то PHP автоматически выполняет этот метод после создания экземпляра объекта. Это делает его подходящим местом для размещения кода, который отвечает за инициализацию нового объекта.



ФУНКЦИИ И МЕТОДЫ

Функции, которые принадлежат классу, называются *методами*. Функция не сильно отличается от метода, поэтому вы можете заменить слово «метод» словом «функция», если так вам понятнее.

Класс может иметь собственный набор переменных, которые помогают экземпляру объекта поддерживать свое состояние и доступны для методов. Строка `public $name;` в приведенном выше примере *Dog.php* определяет `$name` в качестве переменной класса или свойства.



ПЕРЕМЕННЫЕ И СВОЙСТВА

Переменные, которые принадлежат классу, называются *свойствами*. Как и в случае с функциями/методами, вы можете заменять слово «свойство» словом «переменная», пока не освоитесь в терминологии ООП.

Когда мы пишем код внутри класса, специальная переменная `$this` представляет экземпляр объекта и помогает нам правильно обрабатывать метод или свойство. Чтобы лучше понять, как это работает, давайте обновим файл *DogTest.php* следующим образом:

```
<?php header("Content-Type: text/html;
↳ charset=utf-8");?>
<meta http-equiv="Content-Type" content="text/html;
↳ charset=utf-8"/>
<?php
require 'Dog.php';
$fido = new Dog('Фидо');
echo 'Собаку зовут: ' . $fido->name . '<br>';
echo 'Собака говорит: ' . $fido->speak() . '<br>';
```

```
$fifi = new Dog('Фифи');  
echo 'Собаку зовут: ' . $fifi->name . '<br>';  
echo 'Собака говорит: ' . $fifi->speak() . '<br>';
```

Мы заменили переменную `$dog` на `$fido` и создали еще один экземпляр объекта `Dog` с именем «Фифи». При запуске кода вы должны увидеть на экране «лай» и Фидо, и Фифи. Однако давайте разберемся в том, чем это обеспечивается.

Каждый объект принадлежит одному и тому же классу (`Dog`), однако каждый из них является отдельным экземпляром с собственными данными, например у каждого есть свое свойство `$name`. При создании экземпляра `$fido` имя «Фидо» присваивается его переменной `$name`, в то время как имя «Фифи» присваивается в качестве имени экземпляра `$fifi`.



ОБРАЩАЙТЕ ВНИМАНИЕ НА СТРЕЛКИ

При вызове метода или свойства не забывайте использовать оператор `->`. Он сообщает PHP о том, что вы ссылаетесь на что-то, принадлежащее классу, а не на обычную переменную или функцию в области выполнения сценария. Имейте это в виду при вызове методов или переменных, поскольку вызов не того, что вам нужно, может привести к непредсказуемым результатам!

Расширение классов

Теперь, когда у вас есть общее представление о парадигме ООП, давайте кратко подведем итоги. Класс — это определение, которое группирует переменные и функции в логическую единицу, а конкретный экземпляр в памяти, созданный на основе этого класса, называется объектом. Разумеется, ООП не ограничивается только этим. Другим важным аспектом является возможность *расширения класса* для добавления или улучшения функциональных возможностей без необходимости заново создавать код для тех аспектов, которые остаются неизменными.

Чтобы разобраться в концепции *наследования*, то есть создания нового класса путем расширения существующего, давайте создадим файл *Pet.php* с определением класса `Pet`. Он будет служить в качестве *базового*

класса, который мы расширим с помощью других классов, чтобы указать определенные типы домашних животных, например собака, кошка, рыбка, ящерица и т. д.

```
<?php header("Content-Type: text/html;
↳ charset=utf-8");?>
<meta http-equiv="Content-Type" content="text/html;
↳ charset=utf-8"/>
<?php
class Pet
{
public $name;
public function __construct($name) {
$this->name = $name;
}
public function speak() {
return 'ничего';
}
}
```

Этот класс очень похож на класс Dog, поэтому здесь для вас не должно быть ничего удивительного. Теперь давайте перепишем класс Dog так, чтобы он расширил класс Pet и использовал его функциональность.

```
<?php header("Content-Type: text/html;
↳ charset=utf-8");?>
<meta http-equiv="Content-Type" content="text/html;
↳ charset=utf-8"/>
<?php
require_once 'Pet.php';
class Dog extends Pet
{
public function speak() {
```

```
return 'Гав! Гав!';
}
public function plays() {
return 'приносит что-нибудь';
}
}
```

И просто ради интереса давайте создадим еще несколько классов животных. Создайте файл *cat.php*:

```
<?php header("Content-Type: text/html;
↳ charset=utf-8");?>
<meta http-equiv="Content-Type" content="text/html;
↳ charset=utf-8"/>
<?php
require_once 'Pet.php';
class Cat extends Pet
{
public function speak() {
return 'Мяу!';
}

public function plays() {
return 'ЛОВИТ МЫШЬ';
}
}
```

...и файл *Fish.php*:

```
<?php header("Content-Type: text/html;
↳ charset=utf-8");?>
<meta http-equiv="Content-Type" content="text/html;
```

```
↳charset=utf-8"/>
<?php
require_once 'Pet.php';
class Fish extends Pet
{
}
```

Теперь давайте протестируем эти файлы. Создайте файл *PetTest.php*, содержащий следующий код:

```
<?php header("Content-Type: text/html;
↳charset=utf-8");?>
<meta http-equiv="Content-Type" content="text/html;
↳charset=utf-8"/>
<?php
require 'Dog.php';
require 'Cat.php';
require 'Fish.php';
$fido = new Dog('Фидо');
echo 'Собаку зовут: ' . $fido->name . '<br>';
echo 'Собака говорит: ' . $fido->speack() . '<br>';
echo 'Собака делает: ' . $fido->plays() . '<br>';
$mittens = new Cat('Миттен');
echo 'Кошку зовут: ' . $mittens->name . '<br>';
echo 'Кошка говорит: ' . $mittens->speack() . '<br>';
echo 'Кошка делает: ' . $mittens->plays() . '<br>';
$bubbles = new Fish('Баббл');
echo 'Рыбку зовут: ' . $bubbles->name . '<br>';
echo 'Рыбка говорит: ' . $bubbles->speack() . '<br>';
```

Теперь все становится интереснее! Создайте классы *Parrot* (попугай) и *Lizard* (ящерица), а затем дополните файл *PetTest.php*, чтобы создать по экземпляру для каждого класса, отобразить их имена и то, что они могут «сказать».

При расширении класса мы, по сути, создаем один класс из другого. Это обеспечивает связь «родитель — потомок». Например, мы можем сказать, что класс `Dog` является потомком класса `Pet`, а класс `Pet` — родителем класса `Dog`. Взяв за основу написанный ранее код, мы избавляемся от необходимости создавать его заново где-либо еще, и в результате наш код становится более организованным и эффективным.

Каждый класс, представляющий определенный тип домашних животных, наследует методы и свойства от родительского класса `Pet`. Рассмотрим класс `Fish`. Он не имеет собственного кода и наследует все свое поведение от класса `Pet`.

С другой стороны, классы `Dog` и `Cat` расширяют класс `Pet`, а также переопределяют методы `speak()` собственной версией и добавляют новую функциональность с помощью методов `plays()`.



ЧТО НАСЧЕТ ПУБЛИЧНОСТИ?

В классах мы определяем *область видимости* наших объектов и методов. Этот параметр может принимать три значения: `public`, `protected` или `private`. Значение `public` (общедоступный) означает, что доступ к объектам и методам можно получить отовсюду. Значение `protected` (защищенный) означает, что объекты и методы могут быть доступны только методам, принадлежащим тому же дереву классов, что и вызываемый метод или объект. Значение `private` (закрытый) означает, что методы и объекты могут быть доступны только классу, к которому принадлежат. Более подробную информацию об области видимости вы можете найти в следующих источниках:

- php.net/manual/ru/language.oop5.visibility.php
- aperiplus.sourceforge.net/visibility.php
- www.sitepoint.com/learn-object-oriented-php/

Мы познакомились с основами использования ООП в PHP. Хотя на первый взгляд данный подход может показаться очень сложным и изощренным, на самом деле он сродни езде на велосипеде — однажды научившись, вы никогда не потеряете навык. ООП не только упрощает разработку объемных и сложных приложений, но и является подходом, который используется почти в любом профессиональном языке программирования. Изучение и освоение этого подхода в PHP поможет вам в будущем разобраться в других языках программирования, поэтому этим навыком стоит овладеть.

С этого момента мы будем считать, что вы разбираетесь в ООП и знаете, как в языке PHP используются классы и объекты. Чтобы освоить темы, которые мы обсудили, обратитесь к следующим ресурсам:

- codular.com/introducing-php-classes
- www.php5-tutorial.com/classes/introduction/
- php.net/manual/en/keyword.extends.php
- jadendreamer.wordpress.com/2011/05/13/php-tutorial-learning-oop-class-basics-extending-classes/
- www.killerphp.com/tutorials/object-oriented-php/
- www.techotopia.com/index.php/PHP_Object_Oriented_Programming
- www.techflirt.com/tutorials/oop-in-php/index.html
- net.tutsplus.com/tutorials/php/object-oriented-php-for-beginners/

Шаблоны

Как уже говорилось в первой главе, файлы PHP могут содержать и код PHP, и код HTML, что делает их идеально подходящими для знакомства программистов, имеющих опыт работы с HTML, с процессом разработки на стороне сервера. Однако при создании сложных приложений комбинирование кода PHP и HTML может приводить к путанице. Кто-то из специалистов в больших командах разработчиков может заниматься разработкой пользовательских интерфейсов, а кто-то — разработкой серверной части приложения. Комбинирование кода в таких условиях способно привести к возникновению сложностей.

Использование шаблонов позволяет отделить логику пользовательского интерфейса от процессов обработки данных, которые происходят на стороне сервера. Существует несколько моделей программирования, построенных вокруг ООП, которые подразумевают использование шаблонов. Наиболее известной является архитектура MVC (Model-View-Controller, «модель-вид-контроллер»), которая реализована в таких известных программных каркасах, как CakePHP¹, Zend Framework² и CodeIgniter³. Архитектура MVC разделяет код на различные области, обеспечивая лучшую его организацию.

¹ cakephp.org

² framework.zend.com

³ ellislab.com/codeigniter

Мы можем применять шаблоны разными способами. Самый простой способ подразумевает загрузку файла РНР, который состоит в основном из HTML-кода. Цель любого фрагмента кода РНР, присутствующего в файле, заключается в отображении содержимого переменной. Мы помещаем переменную в определенном месте файла, содержащего только код РНР, а затем используем шаблон, который будет управлять отображением. Давайте рассмотрим пример.

Сначала файл шаблона будет выглядеть следующим образом:

```
<?php header("Content-Type: text/html;  
↳ charset=utf-8");?>  
<meta http-equiv="Content-Type" content="text/html;  
↳ charset=utf-8"/>  
<?php  
<html>  
<head>  
<title><?php echo $pageTitle; ?></title>  
</head>  
<body>  
<ul>  
<?php foreach ($array as $item) {?>  
<li><?php echo $item; ?></li>  
<?php } ?>  
</ul>  
</body>  
</html>
```

Затем сценарий РНР задаст переменные и включит шаблон:

```
<?php header("Content-Type: text/html;  
↳ charset=utf-8");?>  
<meta http-equiv="Content-Type" content="text/html;  
↳ charset=utf-8"/>
```

```
<?php
$pageTitle = 'Пример шаблона';
$array = array('one', 'two', 'three');
require 'path/to/template.php';
```

После включения шаблон наследует область видимости вызывающего файла и получает доступ ко всем переменным, функциям, классам и т. д., которые содержит вызывающий файл. У этого метода использования шаблонов есть следующие преимущества:

- его легко реализовать. Он не требует специальной сторонней библиотеки для визуализации шаблонов, чтобы обеспечить его доступность для приложения;
- шаблоны могут обрабатывать код PHP. Шаблоны могут обходить массивы, вызывать функции и т. д.

Тем не менее, у этого подхода есть и недостатки:

- разработчики пользовательских интерфейсов, применяющие шаблоны, должны знать язык PHP. Все разработчики, работающие с шаблонами, должны владеть PHP и обладать навыками разработки на этом языке;
- этот способ не подразумевает создания настоящих шаблонов. Строго говоря, эти «шаблоны» на самом деле являются не шаблонами, а дополнительными PHP-файлами, используемыми для определения структуры данных в рамках приложения.

Альтернативой является использование специальных библиотек шаблонов, предусматривающих собственный синтаксис. Они сильно отличаются от шаблонов, которые мы только что обсуждали, поскольку для того, чтобы наполнить шаблон данными, код PHP не применяется. Вместо этого используется специальный синтаксис, уникальный для конкретной библиотеки, который обычно представляет собой нечто среднее между PHP и обычным текстом. При визуализации шаблона библиотека заменяет специальный синтаксис данными, которые он представляет.

Этот подход используется в системе Expression Engine CMS¹, в которой все шаблоны содержат конкретные блоки кода для отображения различных фрагментов данных.

¹ ellislab.com/expressionengine

```
<?php header("Content-Type: text/html;  
↳ charset=utf-8");?>  
<meta http-equiv="Content-Type" content="text/html;  
↳ charset=utf-8"/>  
<?php  
<html>  
<head>  
<title>{% pageTitle %}</title>  
</head>  
<body>  
<ul>  
{% if array as element %}  
<li>{% element %}</li>  
{% /if %}  
</ul>  
</body>  
</html>
```

У этого метода есть следующие преимущества:

- отсутствует необходимость в изучении языка РНР. Разработчикам пользовательских интерфейсов, применяющим шаблоны, можно предоставить список блоков кода, которые они могут использовать для запроса данных от РНР, что снимает необходимость изучать процесс кодирования и синтаксис РНР;
- этот метод подразумевает использование стандартных HTML-файлов. Файлы шаблонов могут сохранять свое традиционное расширение HTML.

Тем не менее, этот подход также не лишен недостатков:

- для визуализации шаблонов требуется библиотека. Чтобы сделать шаблоны доступными для вашего приложения, понадобится сторонняя библиотека;
- каждая библиотека может иметь собственные специальный синтаксис и функциональность. Это может повлиять на структуру обработки данных вашего приложения или конфликтовать с вашим стилем кодирования.

Оба подхода имеют свои плюсы и минусы. Вам стоит ознакомиться с различными вариантами использования шаблонов, чтобы выбрать то, что лучше всего подходит для вашего приложения. Более подробную информацию о шаблонах вы можете найти на следующих ресурсах:

- www.broculos.net/2008/03/how-to-make-simple-html-template-engine.html
- coding.smashingmagazine.com/2011/10/17/getting-started-with-php-templating/
- www.sitepoint.com/smarty-php-template-engine/
- www.sitepoint.com/beyond-template-engine/

Файлы проекта

Теперь, когда мы познакомились с парадигмой ООП и концепцией использования шаблонов, пришло время применить эти мощные методы программирования к нашему блог-приложению для того, чтобы создать его программный каркас.

Нам нужно добавить несколько файлов. На рис. 3.1 показано, как должен выглядеть корневой каталог, на рис. 3.2 изображен каталог *admin*, на рис. 3.3 — каталог *includes*, а на рис. 3.4 — каталог *frontend*.

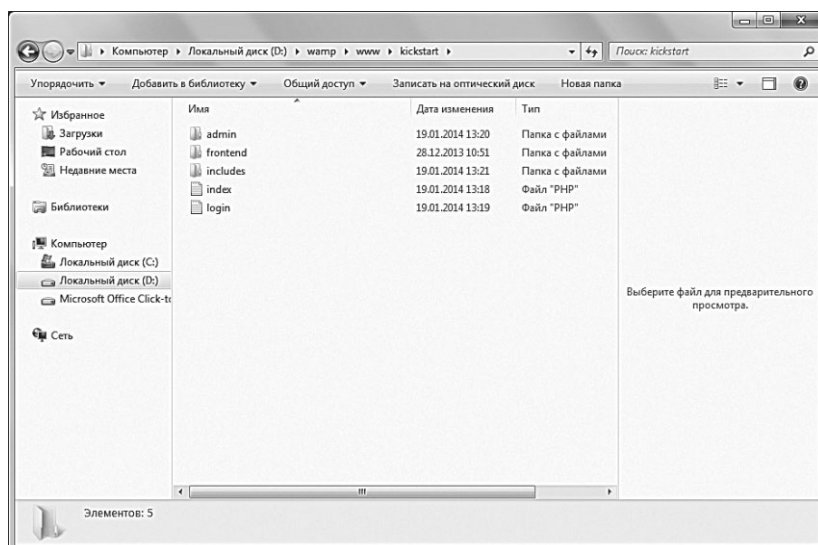
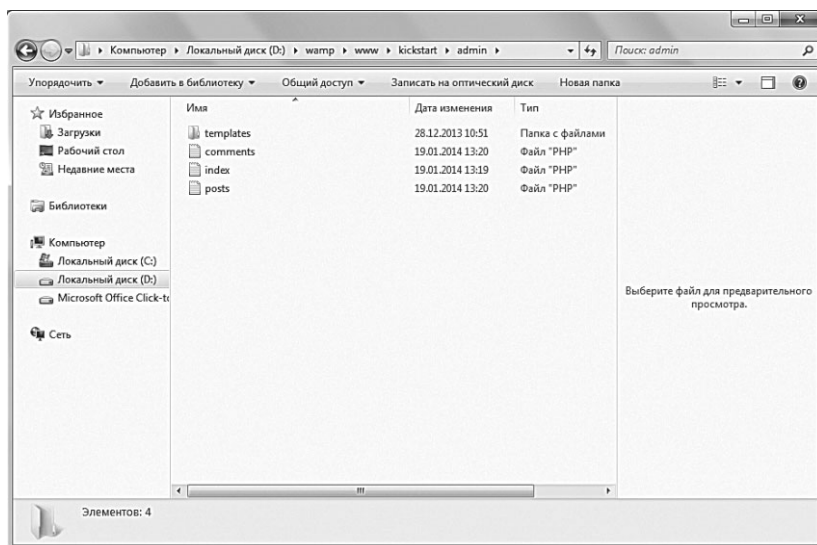
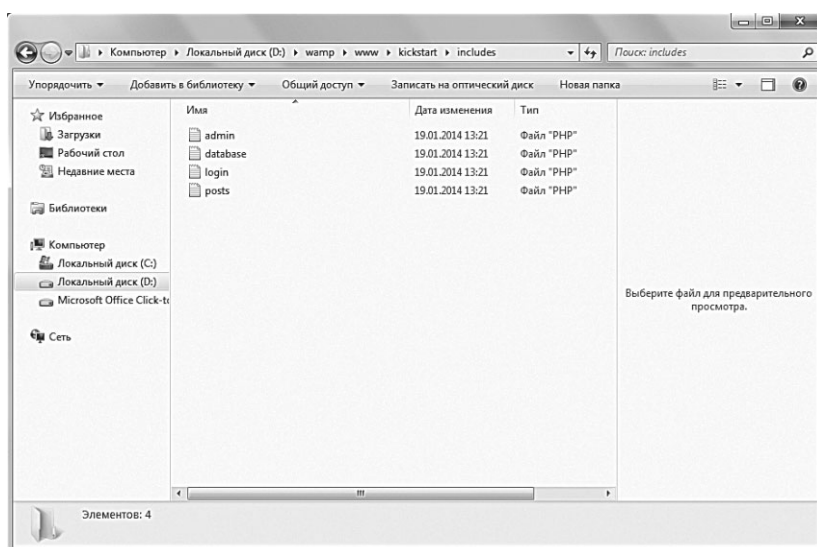
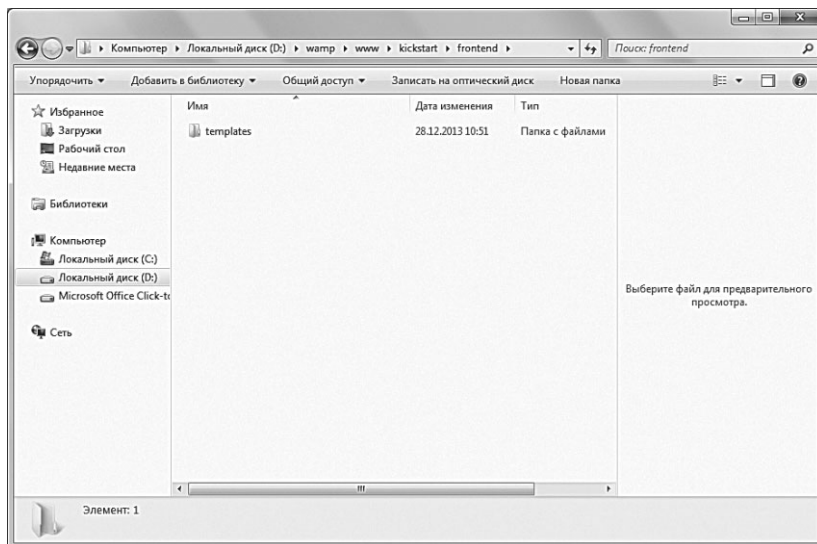


Рис. 3.1. Корневой каталог

Рис. 3.2. Каталог *admin*Рис. 3.3. Каталог *includes*

**Рис. 3.4.** Каталог *frontend*

Мы добавим код каркаса, чтобы в общих чертах показать принцип работы. Дополните перечисленные файлы следующим кодом и сохраните изменения.

index.php

```
<?php
require_once('includes/posts.php');
$blog = new Posts;
$admin = new Comments;
```

login.php

```
<?php
require_once('includes/login.php');
$login = new Login;
```

includes/database.php

```
<?php header("Content-Type: text/html;
↳charset=utf-8");?>
<meta http-equiv="Content-Type" content="text/html;
↳charset=utf-8"/>
<?php
class Database{

public $dbserver = '';
public $username = '';
public $password = '';
public $database = '';
public $db = '';
public function __construct(){
$this->dbserver = 'localhost';
$this->username = 'xxx'; //имя пользователя для доступа
↳к базе данных
$this->password = 'xxx'; //пароль для доступа к базе
↳данных
$this->database = 'xxx'; //имя базы данных
$this->db = new PDO("mysql:host=".$this->dbserver.";
dbname=".$this->database, $this->username, $this-
↳>password);
}

public function dbselect($table, $select, $where=NULL){

}

public function dbadd($tablename, $insert, $format){

}
}
```

```
public function dbupdate($tablename, $insert, $where){  
  
}  
}
```

includes/admin.php

```
<?php header("Content-Type: text/html;  
↳charset=utf-8");?>  
<meta http-equiv="Content-Type" content="text/html;  
↳charset=utf-8"/>  
<?php  
session_start();  
require_once('database.php');  
class Adminpanel{  
public function __construct(){  
}  
}  
class Posts extends Adminpanel{  
  
public function __construct(){  
parent::__construct();  
}  
  
public function listPosts(){  
  
}  
  
public function editPosts(){  
  
}  
  
public function addPost(){
```

```
}

public function savePost(){

}

public function deletePost(){

}

}

class Comments extends Adminpanel{

public function __construct(){
parent::__construct();
}

public function listComments(){

}

public function deletePost(){

}

}

$admin = new Adminpanel();
```

includes/login.php

```
<?php header("Content-Type: text/html;
↳ charset=utf-8");?>
<meta http-equiv="Content-Type" content="text/html;
```

```
↳ charset=utf-8"/>
<?php
require_once('database.php');
class Login{

public function __construct(){

}

public function index(){

}

public function loginSuccess(){

}

public function loginFail(){

}

private function validateDetails(){

}

}
```

includes/posts.php

```
<?php
require_once('database.php');
class Blog{
public $ksdb = '';
public $base = '';
public function __construct(){
```

```
$this->ksdb = new Database;
$this->base = new stdClass;
$this->base->url =
"http://".$_SERVER['SERVER_NAME'];
}
}

class Posts extends Blog{
public function __construct(){
parent::__construct();
}

public function getPosts(){

}

public function viewPost($postId){

}
}

class Comments extends Blog{

public function __construct(){
parent::__construct();
}

public function commentNumber($postId){

}

public function getComments($postId){

}
}
```



```
public function addComment() {  
  
}  
}
```

Все это может показаться сложным для понимания, однако не волнуйтесь. В следующей главе мы детально разберем каждый файл и содержащийся в нем код по мере создания функциональности, которая будет отвечать за работу нашего приложения.

Резюме

В этой главе мы познакомились с двумя методами, упрощающими организацию и сопровождение кода, — с ООП и шаблонами.

ООП позволяет разработчикам создавать организованный и допускающий многократное использование код. Вы можете обнаружить схемы ООП практически в каждом языке программирования, поэтому изучение и освоение ООП позволит вам повысить свой профессионализм в области разработки.

Использование шаблонов — это еще один метод, который обеспечивает лучшую организацию кода и помогает разработчикам, занимающимся созданием различных частей приложения, не становиться друг у друга на пути.

В следующей главе мы рассмотрим каждый из файлов, которые мы создали для нашего блог-приложения, и подробно обсудим содержащиеся в них методы, пока будем создавать основу нашего приложения.

Глава 4

ФОРМЫ

Формы являются жизненно важной частью любого веб-приложения. Обычно они используются как средство, позволяющее посетителям предоставлять некие данные веб-сайтам. Если вам приходилось заниматься веб-разработкой, то, скорее всего, вы имели дело с формами, вероятно, подключая их к сторонней системе валидации или обработки. PHP может работать с очень сложными формами, позволяя вашему приложению проверять и обрабатывать отправляемые пользователями данные.

В этой главе мы рассмотрим различные способы сбора данных из формы и обсудим, как проверить и подготовить их к размещению в базе данных MySQL. Позднее мы продолжим создание нашей блог-платформы и добавим рабочую панель администратора.

Элементы форм

Сначала давайте рассмотрим основные элементы, из которых состоит рабочая HTML-форма. Формы создаются с помощью различных HTML-элементов, например `<form>`, `<input>`, `<textarea>`, `<select>` и `<optgroup>`, отвечающих за ввод данных. Существуют также элементы `<label>` для маркировки входных данных, хотя присоединенные к этим меткам данные не передаются через форму при отправке.

Основа формы является важной частью процесса передачи данных из формы сценарию обработки, поэтому давайте рассмотрим важные атрибуты элемента `<form>`, о которых вам следует помнить.

- Атрибут `action` указывает место назначения, куда отправляются данные из формы. Значением этого атрибута может быть абсолютный или относительный URL-адрес.
- Атрибут `method` определяет, как данные передаются на URL-адрес, указанный в атрибуте `action`. Значением атрибута `method` может быть либо `POST`, либо `GET`, причем значение `GET` используется по умолчанию, если атрибуту значение не присвоено. Мы обсудим эти варианты далее в этой главе, поскольку они играют очень важную роль в процессе обработки данных формы в PHP.
- Атрибут `enctype` определяет, как кодируются данные формы при их отправке. Тем не менее этот атрибут оказывает влияние только при использовании значения `POST` для атрибута `method`. Атрибут `enctype` может принимать одно из трех значений: `application/x-www-form-urlencoded`, `multipart/form-data` и `text/plain`. Первый вариант кодирует данные формы так, что все пробелы заменяются на «+», а специальные символы преобразуются в значения ASCII Hex. Второй вариант не кодирует данные формы. Это важно, если в форме присутствует элемент, относящийся к загрузке файла или обработке. При использовании последнего варианта пробелы преобразуются в «+», а специальные символы не кодируются. Эти варианты следует рассмотреть при кодировании формы, поскольку они будут в значительной степени определять состояние данных в момент достижения ими URL-адреса, указанного в атрибуте `action`.

Существует два атрибута, которые можно присвоить любому элементу формы. Кроме того, они играют важную роль при обработке данных с помощью сценариев PHP.

- Значение атрибута `name` является идентификатором, используемым для сбора данных при их отправке на целевой URL-адрес. Например, если у нас есть элемент `<input type="text" name="data" />`, то значение, введенное для него пользователем, становится доступным в массиве на сервере, и на него можно сослаться с помощью ключа массива `'data'`.
- Атрибут `value` содержит информацию или данные о каждом элементе формы. Этот атрибут можно использовать, чтобы установить для элемента значение по умолчанию.

Ниже приведен пример формы, в которой используются некоторые из описанных выше атрибутов:

```
<form method="post">
<input name="myField" value="Hello World" />
<button type="submit">submit</button>
</form>
```

До сих пор мы рассматривали только важные части формы, использующиеся в PHP. Тем не менее вы можете использовать атрибуты и теги, которые не были здесь перечислены. Мы обсудили лишь основные элементы, поэтому, если вы хотите получить дополнительную информацию о формах, обратитесь к следующим ресурсам:

- reference.sitepoint.com/html/elements-form
- www.tizag.com/htmlT/forms.php
- www.htmlgoodies.com/tutorials/forms/article.php/3479121/So-You-Want-A-Form-Huh.htm

Методы POST и GET

Атрибут формы `method` может принимать одно из двух значений: `POST` или `GET`. Мы знаем, что в PHP существуют специальные predefined переменные, например, `$_SESSION`. Для сбора данных из HTML-форм в PHP предусмотрены переменные `$_POST` и `$_GET`, которые, тем не менее, могут использоваться и для других целей, о чем мы поговорим чуть позже.

Сейчас давайте на примере проиллюстрируем разницу между методами `POST` и `GET`.

Создайте два новых файла в папке *experiments*: *form.php* и *collect.php*. В файл *form.php* добавьте следующий код:

```
<?php header("Content-Type: text/html;
↳ charset=utf-8");?>
<meta http-equiv="Content-Type" content="text/html;
↳ charset=utf-8"/>
<html>
<body>
```

```
<form method="post" action="collect.php">
<h1>Форма №1</h1>
<input name="data" placeholder="введите строку"
type="text" />
<button type="submit">Отправить</button>
</form>
<form method="get" action="collect.php">
<h1>Форма №2</h1>
<input name="data" placeholder="введите строку"
type="text" />
<button type="submit">Отправить</button>
</form>
</body>
</html>
```

Добавили? Хорошо. Теперь добавьте следующий код в файл *collect.php*:

```
<?php header("Content-Type: text/html;
↳ charset=utf-8");?>
<meta http-equiv="Content-Type" content="text/html;
↳ charset=utf-8"/>
<?php
if (!empty($_POST['data'])) {
echo 'Следующая строка отправлена из формы №1: ' .
$_POST['data'];
} elseif (!empty($_GET['data'])) {
echo 'Следующая строка отправлена из формы №2: ' .
$_GET['data'];
}
```

Теперь откройте файл *form.php* в браузере, введите что-нибудь в поле ввода формы № 1 и нажмите кнопку отправки данных. Вы должны увидеть что-то похожее на рис. 4.2:

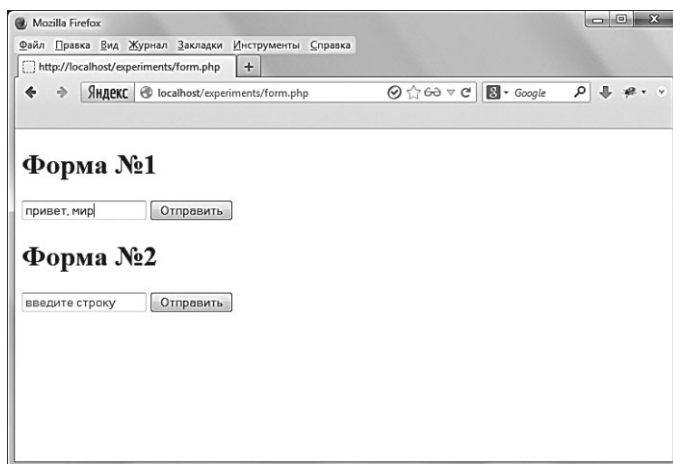


Рис. 4.1. Ввод данных в форму

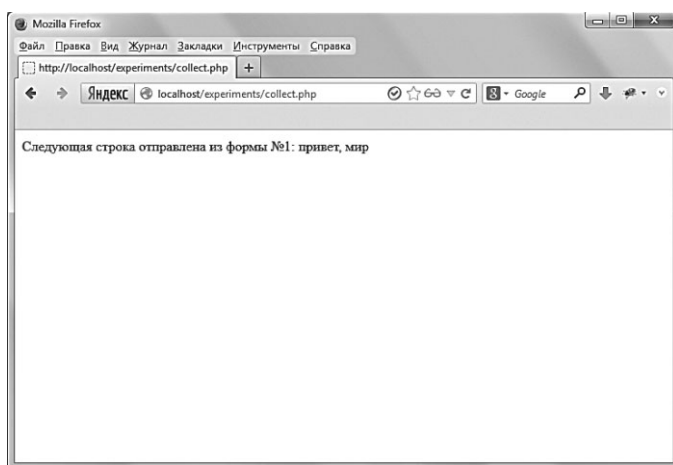


Рис. 4.2. Результат отправки данных формы

Если вы заполните форму № 2, как показано на рис. 4.3, и нажмете кнопку отправки данных, то вы должны будете увидеть нечто похожее на рис. 4.4:

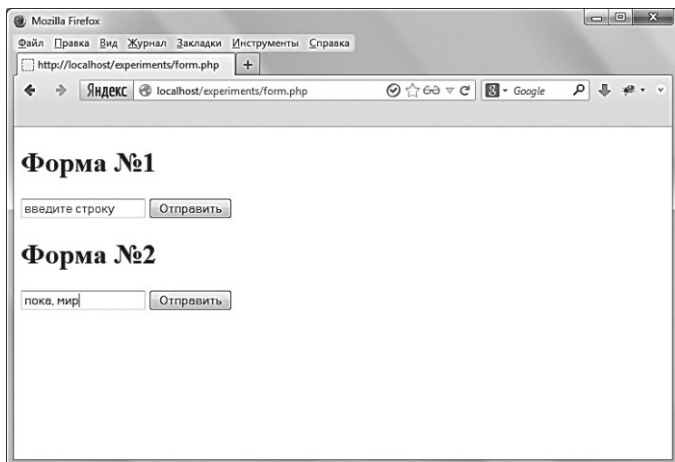


Рис. 4.3. Ввод данных в форму №2

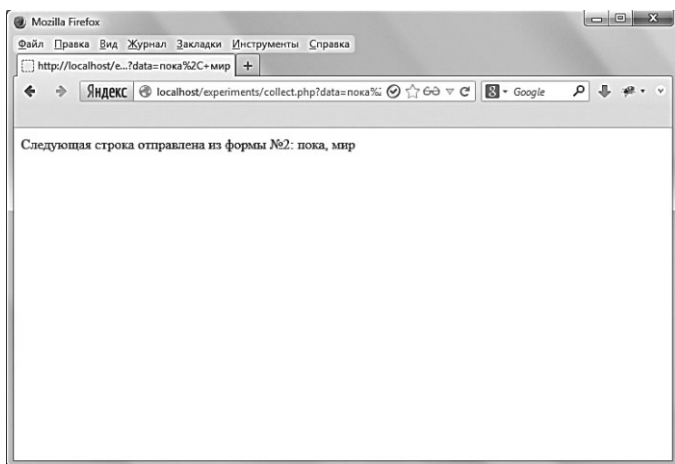


Рис. 4.4. Результат отправки данных формы

Основное различие между этими двумя формами заключается в URL-адресе, на который вы будете перенаправлены при отправке данных формы № 2, как показано на рис. 4.5.

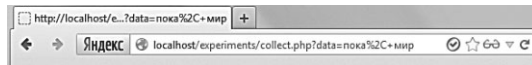


Рис. 4.5. URL-адрес, на который перенаправляется пользователь после отправки данных формы

Как видите, строка, введенная в форму № 2, была отправлена на URL-адрес вместе с некоторыми другими фрагментами кода. Это то, что мы называем *переменной URL*, которая является результатом отправки формы с помощью метода GET. Метод POST, с другой стороны, передает данные между страницами скрытно. При использовании этого метода пользователь не видит никаких доказательств отправки данных, кроме случаев, когда строка отображается в их браузере файлом *collect.php*.

Поскольку данные присутствуют в URL-адресе в качестве переменной до отправки сценарию *collect.php*, метод GET менее безопасен, чем метод POST. Таким образом, если в вашей форме есть скрытые поля ввода, например для данных идентификации или авторизации, вам следует использовать метод POST, а не метод GET. На самом деле в большинстве случаев при разработке веб-приложения вы будете выбирать для передачи данных метод POST, поскольку при использовании метода GET любые данные могут помещаться в переменную URL.

Например, если мы перейдем к файлу *collect.php* в браузере, мы должны будем увидеть пустой экран. Теперь в адресной строке после *collect.php* давайте добавим код `?data=привет%20мир%20я%20добавил%20код`. Если затем мы перейдем по этому новому URL-адресу, то увидим, что введенные в него данные отображаются на экране. Наш пример на первый взгляд не вызывает тревоги и даже может показаться довольно впечатляющим, однако он демонстрирует потенциально рискованную ситуацию.

Представьте, что у нас есть форма, которая выглядит, как на рис. 4.6.

При отправке данных этой формы содержащаяся в ней информация посылается в базу данных. Эта форма также включает в себя скрытый элемент ввода, который содержит данные, идентифицирующие пользователя в базе данных.

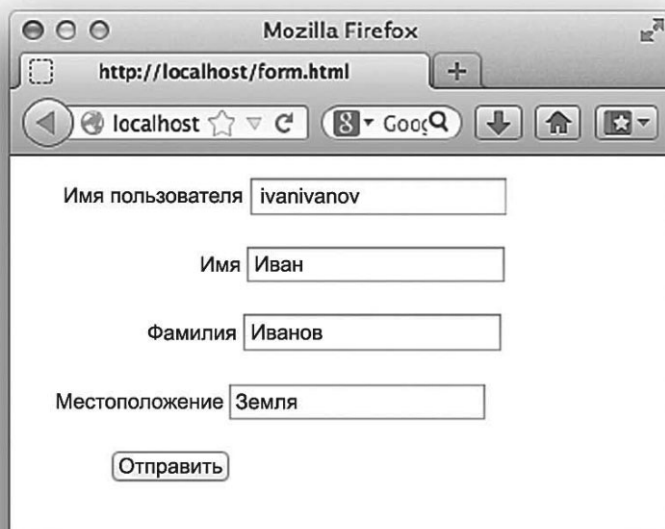


Рис. 4.6. Форма, подверженная риску утечки данных

Для этой формы мы использовали метод GET, поэтому при ее отправке в нашем браузере отобразится URL-адрес, подобный следующему:

```
http://www.ourdomain.com/savedata.  
php?userid=1&name=user&bio=hello%20world
```

Видите, в чем заключается потенциальный риск? Злоумышленник может изменить значение переменной URL `userid`, а затем — значения переменных URL `name` или `bio`.

В результате информация о пользователе будет изменена без его разрешения или даже без его ведома.

Действия формы в PHP

Мы рассмотрели несколько простых примеров работы с формами и познакомились с основами использования переменных `$_POST` и `$_GET`, однако обработка форм с помощью PHP этим не ограничивается.

Сначала кратко представим переменную `PHP_SELF`. Как уже говорилось, в PHP есть специальные предопределенные переменные, которые мы можем использовать в наших сценариях для решения некоторых задач. Однако данная группа переменных имеет еще больше особенностей, чем остальные.

Переменные этой группы расположены в массиве `$_SERVER`, который содержит информацию о сервере и среде выполнения, например заголовки, пути и расположение сценариев на сервере. Мы можем получить доступ к этим данным так же, как к любому другому массиву.

Одним из ключей в массиве `$_SERVER` является переменная `PHP_SELF`, содержащая информацию о PHP-файле, к которому в данный момент осуществляется доступ, по отношению к корневому каталогу документов.

Чтобы увидеть переменную `PHP_SELF` в действии, давайте создадим новый файл с именем *self.php* в папке *experiments*. После создания добавьте в этот файл следующий код:

```
<?php
echo $_SERVER['PHP_SELF'];
```

Теперь, если вы откроете этот файл в браузере, вы увидите URL-адрес файла *self.php* относительно корневого каталога документов вашего сервера.

В нашем случае он будет подобен адресу `localhost/experiments/self.php`. Мы можем применить значение `'PHP_SELF'` в атрибуте `action` нашей формы, чтобы данные из нее передавались сценарию *self.php*, где можно использовать переменные `$_POST` или `$_GET`, проверяя таким образом, отправила ли форма данные, а затем выполнить код для дальнейшей их обработки.

Примером этого может быть:

```
<?php header("Content-Type: text/html;
↳ charset=utf-8");?>
<meta http-equiv="Content-Type" content="text/html;
↳ charset=utf-8"/>
<?php
if (!empty($_POST['data'])) {
echo 'Форма отправлена';
// код обработки данных формы
}
?>
<html>
<body>
<form action="POST" action="<?php echo $_SERVER
['PHP_SELF']; ?>">
<input type="text" name="data" type="text"
placeholder="Введите строку" />
<button type="submit">Отправить</button>
</form>
</body>
</html>
```

При отправке формы PHP выполняет команду `echo`, которая показывает в нашем браузере строку «Форма отправлена».

Переменная `PHP_SELF` очень полезна для форм, данные из которых должны передаваться тому же сценарию, который их отображает и после отправки выполняет код для их обработки.

При использовании переменной `PHP_SELF` в качестве значения атрибута `action` вашей формы важно использовать функцию `htmlspecialchars()` для удаления из URL-адреса нежелательных символов, что позволяет предотвратить атаки типа XSS (Cross Site Scripting, меж-

сайтовый скриптинг). Для этого в приведенном выше примере нужно изменить `тег form` так, чтобы он выглядел следующим образом:

```
<form action="POST" action="<?php echo  
↳htmlspecialchars  
($_SERVER['PHP_SELF']); ?>">
```

Для получения дополнительной информации о том, как повысить уровень безопасности сценариев, и о защите от уязвимостей XSS обратитесь к следующим публикациям:

- seancoates.com/blogs/xss-woes
- blog.astrumfutura.com/tag/xss/

Суперглобальные переменные и массив `$_REQUEST`

Переменные `$_POST` и `$_GET` относятся к особой группе переменных, известной как *суперглобальные переменные*. Суперглобальные переменные — это специально определенные переменные, как правило, массивы, встроенные в PHP и доступные в любом сценарии в любой момент времени. Они называются суперглобальными потому, что к ним можно получить доступ в любом месте и в любое время. Переменная `$_SERVER`, о которой мы говорили в предыдущем разделе, наряду с некоторыми другими, которые мы еще не обсуждали, также является суперглобальной переменной.

К суперглобальным также относится переменная `$_REQUEST`. Она несколько отличается от других суперглобальных переменных, таких как `$_POST` и `$_GET`. Переменная `$_REQUEST` необычна тем, что ее ключи создаются всеми значениями, сгенерированными в текущем HTTP-запросе браузером пользователя. Это означает, что переменная `$_REQUEST` может получить доступ ко всем данным, хранящимся в переменных `$_GET` и `$_POST`.

Кроме того, любые данные, хранящиеся в файлах cookie браузера, также хранятся в переменной `$_REQUEST`. Это может показаться весьма полезным, однако на самом деле представляет значительный риск, который связан в частности с возможностью получения доступа к данным файлов cookie в PHP. Наличие суперглобальной переменной, собираю-

щей данные и доступной в любом месте в PHP, подобно бомбе замедленного действия в PHP-коде.

Я настоятельно рекомендую не использовать переменную `$_REQUEST`, а также убедиться в том, что она недоступна для вашего кода PHP. Это избавит вас от риска подвергнуться вредоносным атакам с целью получения доступа к суперглобальным переменным и к конфиденциальным данным ваших пользователей.

Файл *php.ini* по умолчанию настроен так, чтобы не включать данные файлов cookie в массив `$_REQUEST`, однако если файл конфигурации PHP *php.ini* был отредактирован, то настройки по умолчанию могли измениться. Настоятельно рекомендуется не получать доступ к данным файлов cookie с помощью суперглобальной переменной `$_REQUEST`. Самым безопасным способом является запрет доступа к данным файлов cookie в файле *php.ini*. Для получения дополнительной информации о суперглобальной переменной `$_REQUEST`, включая примеры ее использования, а также описание рисков, связанных с ее применением, обратитесь к руководству по PHP¹.

Формы и базы данных

После сказанного выше вам может показаться, что метод GET имеет одни недостатки, однако он способен быть весьма полезным. Как мы уже говорили, метод GET передает значения полей формы в качестве параметров в URL-адресе, который, в свою очередь, может использоваться PHP-сценарием для создания динамических данных.

До сих пор мы рассматривали данные, взятые из URL-адреса, созданного формой. Однако мы также можем использовать метод GET без формы для создания динамического URL-адреса. Это будет полезно во многих ситуациях. Например, если нам нужно загрузить сообщение в пользовательский интерфейс блог-приложения, то мы можем получить `id` сообщения из специального URL-адреса, используя переменную `$_GET`, а затем передать этот `id` нашему сценарию для получения конкретного сообщения из базы данных.

Чтобы опробовать этот метод, нам нужно открыть приложение phpMyAdmin и создать новую базу данных с именем `testposts`, выбрав в нашем списке пользователя, обладающего всеми привилегиями.

¹ www.php.net/manual/ru/reserved.variables.request.php

Далее мы создадим новую таблицу с именем `posts`, используя сценарий PDO MySQL или приложение phpMyAdmin.

Для целей данного примера наша таблица `posts` будет содержать только два столбца: `id` и `content`. Для столбца `id` выберите тип `INT`, индекс `PRIMARY` и установите флажок `A_I` (`AUTO_INCREMENT`). Для столбца `content` укажите тип `LONGTEXT`. В раскрывающемся списке **Сравнение** под таблицей выберите пункт `utf8_general_ci`.

После создания таблицы нужно поместить в нее тестовое сообщение. Перейдите на вкладку **Вставить** и добавьте какой-нибудь текст в поле **content**. При необходимости добавьте несколько HTML-тегов абзаца. Уникальный идентификатор для этой записи должен быть сгенерирован автоматически.



ПРИМЕНЯЙТЕ К СОДЕРЖИМОМУ ВАШЕЙ БАЗЫ ДАННЫХ ПРОСТОЕ HTML-ФОРМАТИРОВАНИЕ

В своей базе данных вы должны использовать только основные HTML-теги для форматирования. Использование сложного кода HTML или любого языка программирования, обрабатываемого браузером, может вызывать проблемы с безопасностью.

Теперь давайте создадим новый файл в веб-каталоге *experiments* с именем *posts.php*. Допустим, что у нас есть следующий URL-адрес: `localhost/experiments/posts.php?id=1`, где переменная URL `id` представляет собой идентификатор сообщения. В данном случае сообщение имеет идентификатор `1`, а *experiments* — это имя папки, в которой мы создадим этот пример.

Добавьте следующий код в файл *posts.php*. Вместо текста `dbuser` и `dbpass` укажите имя пользователя и пароль для доступа к базе данных:

```
<?php header("Content-Type: text/html;
↳ charset=utf-8");?>
<meta http-equiv="Content-Type" content="text/html;
↳ charset=utf-8"/>
<?php
class Posts {
```

```
public $db = '';
public function __construct() {
    $this->db = new
    PDO("mysql:host=localhost;dbname=testposts", "dbuser",
    ↵"dbpass");
    $this->db->setAttribute(PDO::ATTR_ERRMODE,
    PDO::ERRMODE_EXCEPTION);
    $this->index();
}
public function index() {
    $id = 0;
    $posts = array();
    $template = '';
    if (!empty($_GET['id'])) {
        $id = $_GET['id'];
    }
    try {
        if (!empty($id)) {
            $query = $this->db->prepare("SELECT *
            FROM posts WHERE id = ?");
            $params = array($id);
            $template = 'single-post.php';
        } else {
            $query = $this->db->prepare("SELECT *
            FROM posts");
            $params = array();
            $template = 'list-posts.php';
        }
        $query->execute($params);
        for ($i = 0; $row = $query->fetch(); $i++) {
            $posts[] = array('id' => $row['id'],
            'content' => $row['content']);
        }
    } catch (PDOException $e) {
```

```
echo $e->getMessage();  
}  
$query->closeCursor();  
$db = null;  
require_once($template);  
}  
}  
$posts = new Posts();  
?>
```

Здесь мы создали новый класс `Posts`. Вначале мы иницилируем соединение PDO MySQL. Затем переходим к методу `index`, в котором проверяем, содержит ли URL-адрес переменную `URL`, и если содержит, мы получаем ее из данного URL-адреса, а также пытаемся получить сообщение с этим же идентификатором из нашей базы данных. Если в URL-адресе нет переменной `URL`, мы просто загружаем все сообщения, содержащиеся в нашей базе. Наконец мы загружаем шаблон, относящийся к выполненному запросу.

Теперь мы должны использовать файлы шаблонов для отображения информации, извлеченной из базы данных. Создайте два новых файла: *list-posts.php* и *single-post.php*. В файл *list-posts.php* добавьте следующий код:

```
<?php header("Content-Type: text/html;  
↳ charset=utf-8"); ?>  
<meta http-equiv="Content-Type" content="text/html;  
↳ charset=utf-8"/>  
<h1>Список сообщений</h1>  
<?php foreach ($posts as $post): ?>  
<h3>Сообщение №<?php echo $post['id']; ?></h3>  
<?php echo $post['content']; ?>  
<a href="<?php echo "http://localhost/experiments/  
↳ post.php?id=". $post['id']; ?>">Подробнее</a>  
<hr/>  
<?php endforeach; ?>
```


Наконец, добавьте следующий код в файл *single-post.php*:

```
<?php header("Content-Type: text/html;  
↳ charset=utf-8"); ?>  
<meta http-equiv="Content-Type" content="text/html;  
↳ charset=utf-8"/>  
<?php foreach($posts as $post): ?>  
<h1>Сообщение №<?php echo $post['id']; ?></h1>  
<hr/>  
<?php echo $post['content']; ?>  
<a href="<?php echo "http://localhost/experiments/  
↳ post.php"; ?>">  
Вернуться к списку сообщений</a>  
<?php endforeach; ?>
```

Теперь, если мы перейдем к файлу *posts.php*, то увидим список сообщений или в данном случае одно сообщение, поскольку в настоящее время наша база данных содержит только его. Каждое сообщение содержит ссылку «Подробнее». Щелкнув по этой ссылке, мы переходим на страницу с конкретным сообщением, при этом отображается шаблон, отличный от того, который использовался для просмотра полного списка сообщений.

Это очень простой пример, показывающий, как переменная `$_GET` позволяет динамически загружать сообщения и взаимодействовать с нашей базой данных. Мы могли бы легко изменить этот пример и использовать переменную `$_POST`.

Однако в этом случае мы не могли бы применять URL-адреса, ссылающиеся на отдельные сообщения. Вместо этого URL-адрес только направлял бы посетителей к списку сообщений, где они должны были бы сами щелкать по ссылке «Подробнее».

Этот пример логически подводит нас к следующему разделу данной главы, где мы будем дорабатывать приведенные ранее фрагменты кода и приступим к обеспечению функциональности нашей блог-платформы.

Усовершенствование нашей платформы

В этом разделе мы дополним только что созданный пример, который отображает список сообщений со ссылками, ведущими к отдельным сообщениям, и обеспечим его функционирование в нашем приложении.

Давайте немного доработаем приведенный выше код. Дополните код конструктора в файле *posts.php* в каталоге *includes* следующим образом:

```
public function __construct() {
    parent::__construct();
    $this->comments = new Comments();
    if (!empty($_GET['id'])) {
        $this->viewPost($_GET['id']);
    } else {
        $this->getPosts();
    }
}
```

С помощью этого кода наш конструктор проверяет наличие переменной URL *id*. Если она присутствует, то он извлекает данные из этой переменной и передает их методу *viewPost()* в классе *Blog*. В случае отсутствия переменной URL *id* приложению дается инструкция загрузить все сообщения из базы данных. Это позволяет посетителям увидеть все сообщения, а затем щелкнуть по ссылке, чтобы просмотреть их по отдельности.

Далее мы дополним метод *getPosts()* в файле *posts.php*:

```
public function getPosts() {
    $id = 0;
    $posts = $return = array();
    $template = '';
    $query = $this->ksdb->db->prepare("SELECT * FROM
    ↳posts");
    try {
```

```
$query->execute();  
for ($i = 0; $row = $query->fetch(); $i++) {  
    $return[$i] = array();  
    foreach ($row as $key => $rowitem) {  
        $return[$i][$key] = $rowitem;  
    }  
}  
} catch (PDOException $e) {  
    echo $e->getMessage();  
}  
$posts = $return;  
$template = 'list-posts.php';  
include_once 'frontend/templates/' . $template;  
}
```

Метод `getPosts()` используется для сбора всех сообщений из базы данных и создания их списка для посетителей нашего блога. Он использует функциональность базы данных PDO из файла *database.php* в папке *includes*. Это позволяет нам написать наш собственный запрос к базе данных через PDO. С помощью этого метода мы хотим загрузить все строки из таблицы `posts`, используя запрос к базе данных `SELECT * FROM posts`. Затем при выполнении запроса мы используем процедуру обработки исключений `try` и `catch`. В случае успешного выполнения мы с помощью цикла обрабатываем массив, возвращенный из базы данных, и добавляем результат к новому ключу в массиве `$return`. В случае неудачи мы даем расширению PDO инструкцию отобразить сообщение об ошибке, которое показывает нам, что не так с запросом к базе данных. Наконец, мы загружаем шаблон *list-posts.php*.



ОТОБРАЖЕНИЕ СООБЩЕНИЙ ОБ ОШИБКАХ PDO

Отображение сообщений об ошибках PDO, как это сделано в вышеприведенном примере, может пригодиться в процессе разработки, поскольку помогает выявлять проблемы в запросах к базе данных. Тем не менее вам не следует отображать сообщения об ошибках PDO в функционирующих приложениях, поскольку это может представлять угрозу для безопасности.

Последнее изменение данного класса заключается в дополнении метода `viewPost()`:

```
public function viewPost($postId) {
    $id = $postId;
    $posts = $return = array();
    $template = '';
    $query = $this->ksdb->db->prepare("SELECT * FROM posts
    └WHERE id = ?");
    try {
        $query->execute(array($id));
        for ($i = 0; $row = $query->fetch(); $i++) {
            $return[$i] = array();
            foreach ($row as $key => $rowitem) {
                $return[$i][$key] = $rowitem;
            }
        }
    } catch (PDOException $e) {
        echo $e->getMessage();
    }
    $posts = $return;
    $posts[0]['content'] = $posts[0]['content'];
    $template = 'view-post.php';
    include_once 'frontend/templates/'.$template;
}
```

Если кратко, то метод `viewPost()` отображает отдельные сообщения для посетителей нашего блога. Наше приложение использует его для загрузки данных конкретного сообщения, основываясь на данных переменной URL. Переменную `$postId` необходимо передать данному методу при его вызове, что мы и сделали с конструктором класса. Этот метод использует функциональность базы данных PDO, чтобы загрузить из нее конкретное сообщение, исходя из его идентификатора (`id`). Кроме того, в нашем сценарии используется обработчик исключений `try ... catch`, который проверяет успешность выполнения запроса к базе данных. В случае успешного выполнения мы снова с помощью

цикла обрабатываем возвращенный из базы данных массив и добавляем полученные данные к ключу в массиве `$return`. В случае неудачи мы, как и раньше, отображаем сообщение об ошибке базы данных, сгенерированное расширением PDO. Наконец, мы загружаем шаблон `view-post.php`.

В описанных выше примерах мы загружаем некоторые файлы шаблонов, поэтому нам необходимо их создать. Мы могли бы создать эти файлы с нуля, однако для целей данного проекта создадим их с помощью набора инструментов Twitter Bootstrap Framework¹. Этот набор инструментов позволяет быстро создать и запустить элементы пользовательского интерфейса веб-приложений, позволяя вам сосредоточиться на создании серверной части приложения.

Если вы считаете, что можете самостоятельно настроить набор инструментов Bootstrap, встройте его в проект так, как считаете нужным. Если вы не разбираетесь в этом процессе, не волнуйтесь, — просто скопируйте все содержимое папки *includes*, находящейся в прилагающемся к этой книге архиве с кодом, и вы в кратчайшие сроки сможете приступить к работе с Twitter Bootstrap.

Вернитесь к нашему проекту и создайте два новых файла в папке *templates* в каталоге *frontend* и присвойте им имена *list-posts.php* и *view-post.php*. В файле *list-posts.php* добавьте следующий код:

```
<?php header("Content-Type: text/html;
↳ charset=utf-8"); ?>
<meta http-equiv="Content-Type" content="text/html;
↳ charset=utf-8"/>
<?php require_once 'includes/temps/header.php'; ?>
<?php foreach ($posts as $post): ?>
<h3>Сообщение №<?php echo $post['id']; ?></h3>
<p><?php echo implode(' ', array_slice(explode(' ',
↳ strip_tags($post['content'])), 0, 10)); ?> [...]</p>
<a href="<?php echo $this->base-
↳ >url."/&id=".$post['id']; ?>" class="btn btn-
↳ primary">Подробнее</a>
```

¹ twitter.github.io/bootstrap/index.html

```

<hr/>
<?php endforeach; ?>
<?php require_once('includes/temps/footer.php'); ?>
Затем добавьте следующий код в файл view-post.php:
<?php header("Content-Type: text/html;
↳ charset=utf-8"); ?>
<meta http-equiv="Content-Type" content="text/html;
↳ charset=utf-8"/>
<?php require_once 'includes/temps/header.php'; ?>
<br/>
<a href="<?php echo $this->base->url; ?>" class="btn
↳ btn-primary">Вернуться к списку сообщений</a>
<?php foreach ($posts as $post): ?>
<h3>Сообщение №<?php echo
↳ htmlspecialchars($post['id']);
?></h3>
<?php echo htmlspecialchars($post['content']); ?>
<hr/>
<?php endforeach; ?>
<?php require_once 'includes/temps/footer.php'; ?>

```

Файл *list-post.php* перебирает сообщения, полученные из нашей базы данных, создавая небольшой отрывок содержимого каждого из них, а также ссылки, перейдя по которым можно прочитать полный текст.

Файл *view-post.php* отображает название и содержимое отдельного сообщения, загруженного из базы данных.

Код в файлах *list-post.php* и *view-post.php* должен казаться довольно понятным. Тем не менее вы, вероятно, заметили необычный фрагмент в файле *list-posts.php*:

```

implode(' ', array_slice(explode(' ',
view-post.phpstrip_tags($post['content'])), 0, 10));

```

Данная строка кода объединяет несколько различных функций PHP для извлечения первых десяти слов сообщения и автоматического создания строки, которая будет отображена в нашем списке сообщений. Функция `strip_tags()` удаляет из сообщения любые HTML-теги, а функция `explode()` создает новый массив и ключ каждый раз, когда функция обнаруживает пробел в тексте содержимого. Далее, функция `array_slice()` используется для отбора первых десяти ключей в нашем массиве, созданном функцией `explode()`. Функции `array_slice()` передается значение 0, чтобы она начала отсчет с 0 и остановилась на 10. Кроме того, ей передается массив, созданный функцией `explode()`. Наконец мы используем функцию `implode()`, чтобы собрать эти десять значений массива в одной строке.

Для получения более полной информации об этих функциях PHP и подробного обзора возможностей, которые открывает их использование, обратитесь к следующим ресурсам:

- php.net/manual/ru/function.strip-tags.php
- php.net/manual/ru/function.explode.php
- php.net/manual/ru/function.array-slice.php
- php.net/manual/ru/function.implode.php

Следующим этапом является создание панелей администратора и авторизации для нашего приложения. Сначала создадим панель авторизации. Откройте файл *login.php*, находящийся в каталоге *includes*, который расположен в главном каталоге проекта. Начните с дополнения метода `__construct()`:

```
public function __construct() {  
    $this->ksdb = new Database;  
    $this->base = (object) '';  
    $this->base->url = "http://".$_SERVER['SERVER_NAME'];  
    $this->index();  
}
```

Наш конструктор создает объекты и переменные, которые мы будем использовать по всему классу. Теперь добавьте следующий код в метод `index()`:

```

public function index() {
    if ($_SERVER['REQUEST_METHOD'] === 'POST') {
        $this->validateDetails();
    } elseif (!empty($_GET['status']))
        ⤵ && $_GET['status'] ==
        'inactive') {
        $error = 'Сеанс завершен в связи с отсутствием
        ⤵ активности. Пожалуйста, авторизуйтесь снова.';
    }
}
require_once 'admin/templates/loginform.php';
}

```

С помощью метода `index()` мы проверяем, не завершен ли сеанс нашего пользователя в связи с его неактивностью. Это осуществляется путем проверки того, не была ли установлена переменная URL `inactive`. Данный метод также проверяет, не были ли данные формы переданы с помощью метода `POST`. Если это так, то сценарий предполагает, что пользователь пытается войти в систему, и запускает метод `validateDetails()`, чтобы проверить его учетные данные, мы рассмотрим это подробнее чуть позже.

Наконец, данный метод загружает сценарий *loginform.php* для загрузки формы, с помощью которой пользователи могут войти в систему.

Далее мы переходим к методам `loginSuccess()` и `loginFail()`:

```

public function loginSuccess() {
    header('Location: http://' . $_SERVER['SERVER_NAME'] .
    '/admin/posts.php');
    return;
}

public function loginFail() {
    return 'Неверное имя пользователя/пароль';
}

```


Метод `loginSuccess()` используется в случае успешного входа в систему и перенаправляет пользователя к панели администратора. Метод `loginFail()` выдает сообщение об ошибке, информируя пользователя о том, что вход в систему не удался. Более подробную информацию о функции `header()` вы можете найти по адресу: www.php.net/manual/ru/function.header.php. Наконец, мы переходим к методу `validateDetails()`:

```
private function validateDetails() {
    if (!empty($_POST['username']) && !empty($_
    ↪POST['password'])) {
        $salt = '$2a$07$R.gJb2U2N.FmZ4hPply2CN$';
        $password = crypt($_POST['password'], $salt);
        $return = array();
        $query = $this->ksdb->db->prepare("SELECT * FROM
        users WHERE username = ? AND password = ?");
        try {
            $query->execute(array($_POST['username'],
            $password));
            for ($i = 0; $row = $query->fetch(); $i++) {
                $return[$i] = array();
                foreach ($row as $key => $rowitem) {
                    $return[$i][$key] = $rowitem;
                }
            }
        } catch (PDOException $e) {
            echo $e->getMessage();
        }
        if (!empty($return) && !empty($return[0])) {
            $this->loginSuccess();
        } else {
            echo $error = $this->loginFail();
        }
    }
}
```

Метод `validateDetails()` является главным в классе `login`. С его помощью наш сценарий проверяет и оценивает попытку пользователя получить доступ к панели администратора блог-приложения.

Сначала этот метод удостоверяется в том, что сценарий получил данные формы с помощью метода `POST`, а также в том, что ключи, соответствующие имени пользователя и паролю, присутствуют в суперглобальной переменной `$_POST`, а не являются пустыми из-за использования функции `empty()`. Если в суперглобальной переменной `$_POST` присутствуют имя пользователя и пароль, мы создаем *соль*¹, состоящую из случайных символов для шифрования нашего пароля. Затем мы берем предоставленные пароли из формы авторизации, зашифровываем их и проверяем, соответствуют ли предоставленные имя пользователя и зашифрованный пароль информации, хранящейся в базе данных.

Теперь, когда мы создали класс `login`, нам нужно создать шаблон для формы авторизации. В каталоге *templates* в папке *admin* создайте новый файл с именем *loginform.php*.



ЗАМЕЧАНИЕ ПО ПОВОДУ СТРУКТУРЫ

Стоит потратить время на создание нового шаблона для заголовка и нижнего колонтитула раздела администрирования. Отделение шаблонов колонтитулов для пользовательского интерфейса позволяет легче отличить панель администратора от пользовательского интерфейса. За более подробной информацией обратитесь к структуре файлов проекта.

Добавьте следующий код в файл *loginform.php*:

```
<?php header("Content-Type: text/html;
↳ charset=utf-8");?>
<meta http-equiv="Content-Type" content="text/html;
↳ charset=utf-8"/>
<?php require_once 'includes/temps/header.php'; ?>
<br/>
<?php if (!empty($error)): ?>
<div class="alert alert-error">~<?php echo $error; ?>
```

¹ [ru.wikipedia.org/wiki/Соль_\(криптография\)](http://ru.wikipedia.org/wiki/Соль_(криптография))

```
</div>
<?php endif; ?>
<br/>
<form action="<?php echo htmlspecialchars($_SERVER
↳ ['PHP_SELF']); ?>" method="post" class="form-
↳ horizontal offset2">
<h3>Вход в админ-панель</h3>
<div class="control-group <?php echo
↳ (!empty($error)?
'error': ''); ?>">
<label class="control-label"
for="inputEmail">Имя пользователя</label>
<div class="controls">
<input type="text" name="username" id="inputEmail"
↳ placeholder="Имя пользователя">
</div>
</div>
<div class="control-group <?php echo (!empty($error)?
↳ 'error': ''); ?>">
<label class="control-label"
for="inputPassword">Пароль</label>
<div class="controls">
<input type="password" name="password"
id="inputPassword" placeholder="Пароль">
</div>
</div>
<div class="control-group">
<div class="controls">
<button type="submit"
class="btn">Войти</button>
</div>
</div>
</form>
<?php require_once 'includes/temps/footer.php'; ?>
```

В этом файле мы создали форму, которая запрашивает у посетителя имя пользователя и пароль для доступа к разделу администрирования нашего приложения. После отправки формы данные из нее посылаются тому же файлу, который наш класс `login` использует для обработки процесса авторизации. Затем этому файлу авторизации вместе с данными о пользователе он отправляет любые сообщения об ошибках.

Теперь, когда мы создали панель авторизации и класс `login`, нам нужно создать панель администратора и необходимые классы. Откройте файл *admin.php* из каталога *includes* и добавьте в него следующий код:

```
class Adminpanel {
public function __construct() {
$this->ksdb = new Database;
$this->base = (object) '';
$this->base->url =
"http://".$_SERVER['SERVER_NAME'];
}
}
```

Здесь мы добавили код в конструктор в классе `Adminpanel`, содержащем объекты и переменные, которые мы будем использовать в файле *admin.php*. Это основной класс, для расширения которого будут использоваться остальные классы в файле администратора.

Теперь дополним метод `listPosts()`:

```
public function listPosts() {
$postes = $return = array();
$query = $this->ksdb->db->prepare("SELECT * FROM
↳ posts");
try {
$query->execute();
for ($i = 0; $row = $query->fetch(); $i++) {
$return[$i] = array();
foreach ($row as $key => $rowitem) {
```

```
$return[$i][$key] = $rowitem;
}
}
} catch (PDOException $e) {
echo $e->getMessage();
}
$posts = $return;
require_once 'templates/manageposts.php';
}
```

Метод `listPosts()` почти в точности соответствует методу `getPosts()` из файла `posts.php`. Мы загружаем все сообщения из таблицы в базе данных и проверяем запрос. Если проверка прошла успешно, мы получаем данные из базы данных и добавляем их в массив `$return`. Если нет, мы отображаем сообщение об ошибке базы данных. Наконец, мы загружаем файл шаблона *manageposts.php*.

Теперь переходим к методу `addPost()`:

```
public function addPost() {
require_once 'templates/newpost.php';
}
```

Этот код просто загружает шаблон *newpost.php*.

Наконец, дополняем метод `savePost()`:

```
public function savePost() {
$array = $format = $return = array();
if (!empty($_POST['post'])) {
$post = $_POST['post'];
}
if (!empty($post['content'])) {
$array['content'] = $post['content'];
}
```

```
$format[] = ':content';
}
$cols = $values = '';
$i = 0;
foreach ($array as $col => $data) {
if ($i == 0) {
$cols .= $col;
$values .= $format[$i];
} else {
$cols .= ',' . $col;
$values .= ',' . $format[$i];
}
$i++;
}
try {
$query = $this->ksdb->db->prepare("INSERT INTO
posts(".$cols.") VALUES (". $values.")");
for($c=0;$c<$i;$c++){
$query->bindParam($format[$c], ${'var' . $c});
}
$z=0;
foreach($array as $col => $data){
${'var' . $z} = $data;
$z++;
}
$result = $query->execute();
$add = $query->rowCount();
} catch (PDOException $e) {
echo $e->getMessage();
}
$query->closeCursor();
$this->db = null;
if (!empty($add)) {
$status = array('success' => 'Ваше сообщение успешно
```

```
↳сохранено.');
```

```
} else {  
$status = array('error' => 'В процессе сохранения  
↳вашего сообщения возникла ошибка. Пожалуйста,  
↳повторите попытку позднее.');
```

```
}  
header("Location: http://localhost/kickstart/admin/  
↳posts.php");  
}
```

Метод `savePost()` использует функциональные возможности базы данных PDO, чтобы подготовить запрос INSERT SQL для сохранения данных в таблице `posts` в нашей базе данных. Затем мы возвращаем статус в зависимости от корректности сохранения информации в базе данных. Здесь мы используем метод вставки, о котором говорили в главе 2.

Теперь нам осталось создать еще пару шаблонов. Мы добавим два новых файла PHP в папку *templates* каталога *admin*. Первый файл *manageposts.php* будет содержать список всех сообщений, хранящихся в нашей базе данных, а файл *newpost.php* — очень простую форму, в которую мы можем вводить данные для создания содержимого нового сообщения. Добавьте следующий код в файл *manageposts.php*:

```
<?php header("Content-Type: text/html;  
↳charset=utf-8");?>  
<meta http-equiv="Content-Type" content="text/html;  
↳charset=utf-8"/>  
<?php require_once '_inc/header.php'; ?>  
<a href="<?php echo $this->base->url; ?>/posts.  
↳php?action=create" class="btn btn-info">Создать  
↳сообщение</a>  
<table>  
<thead>  
<tr>
```

```

<td>Заголовок </td>
<td>Содержимое</td>
<td>Действия</td>
</tr>
</thead>
<tbody>
<?php foreach($posts as $post): ?>
<tr>
<td><h3>Сообщение №<?php echo
htmlspecialchars($post['id']);?></h3></td>
<td><p><?php echo implode(' ',
array_slice(explode(' ',strip_tags($post['content'])),
↳0, 10)); ?> [...]</p></td>
<td><a href="<?php echo $this->base-
>url."/posts.php ?id=".$post['id']."&action=
↳edit"; ?>" class="btn btn-primary">Править
↳</a><a href="<?php echo $this->base->url."/?id=".
↳$post['id']."&action=delete"; ?>" class="btn btn-
↳primary">Удалить</a></td>
</tr>
<?php endforeach; ?>
</tbody>
</table>
<?php require_once '_inc/footer.php'; ?>

```

Здесь мы используем комбинацию методов `strip_tags()`, `explode()`, `array_slice()` и `implode()`, чтобы создать отрывок сообщения, а также цикл `foreach` для перебора всех сообщений, извлеченных из нашей базы данных.

Теперь откройте файл *newpost.php* и добавьте следующий код:

```

<?php header("Content-Type: text/html;
↳charset=utf-8");?>

```



```
<meta http-equiv="Content-Type" content="text/html;
↳ charset=utf-8"/>
<?php require_once '_inc/header.php'; ?>
<form action="<?php echo $this->base->url.'/posts.
↳ php?action=save'; ?>"method="POST">
<h3>
Новое сообщение
</h3>
<div class="control-group">

<label class="control-label" for="content">Сообщение
↳ </label>
<div class="controls">
<textarea name="post[content]" id="content">
↳ </textarea>
</div>
</div>
<div class="control-group">
<div class="controls">
<button type="submit" class="btn">
Сохранить сообщение
</button>
</div>
</div>
</form>
<?php require_once '_inc/footer.php'; ?>
```

На данный момент у нас есть только обычная текстовая область для ввода содержимого. Позднее мы дополним эту панель областью для заголовка сообщения, а также превратим текстовую область в динамический и полезный редактор контента.

После добавления в файл *newpost.php* вышеприведенного кода у вас должна быть функционирующая панель администратора, в которой вы можете создавать новые сообщения!

Мы также можем получить доступ к публичному разделу нашего проекта, то есть к пользовательскому интерфейсу, просмотреть список сообщений и щелкнуть по отдельному сообщению, чтобы прочитать его, как показано на рис. 4.7. Здорово, правда?

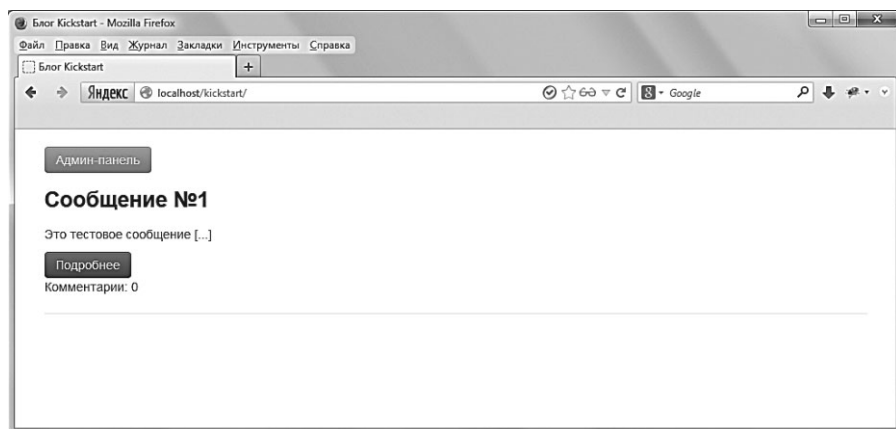


Рис. 4.7. Пользовательский интерфейс нашего проекта

Вы можете открыть основной файл проекта *index.php* и начать использовать систему.

ПРИМЕЧАНИЕ

Для авторизации используйте логин **admin** и пароль **admin**.

В следующей главе мы усовершенствуем панель администратора так, чтобы можно было обновлять имеющиеся в базе данных сообщения, а также удалять их.

Резюме

Формы являются жизненно важной частью любого веб-приложения, позволяя пользователям отправлять данные. Язык PHP располагает мощными инструментами для обработки, хранения и работы с данными, предоставленными с помощью форм. В этой главе мы обсудили, как PHP может собирать данные из формы, рассмотрели методы POST и GET, а также узнали о том, как проверить собранные данные, подготовить их к помещению в базу данных MySQL. Затем мы использовали эти методы для создания нашего блог-приложения.

Глава 5

СЕАНСЫ И ФАЙЛЫ СООКІЕ

Как мы уже обсуждали, базы данных идеально подходят для постоянного хранения информации, которую приложение может извлечь позднее. Тем не менее язык PHP предусматривает и способы временного хранения данных. *Файлы cookie* и *сеансы* предназначены, чтобы хранить данные меньшего размера, чем обычно содержатся в базе данных. Например, они часто используются для хранения учетных данных пользователей, а также для обеспечения быстрого доступа к пользовательским данным в профиле приложения.

Разница между файлами cookie и сеансами заключается в месте хранения данных. Файлы cookie сохраняют данные на клиентском компьютере, в то время как данные сеансов хранятся на веб-сервере. PHP может работать как с файлами cookie, так и с сеансами, и мы подробно рассмотрим каждый из этих вариантов далее в этой главе.

Файлы cookie: обзор

HTTP-запросы не сохраняют состояния, это означает, что каждый раз при запрашивании браузером новой страницы данные, относящиеся к предыдущему запросу пользователя, не сохраняются веб-сервером.

К счастью, у нас есть возможность использовать файлы cookie для хранения небольших фрагментов данных в браузере посетителя. Эта информация может быть загружена веб-сайтом и использована в сценариях для персонализации данных или для создания динамических данных на страницах сайта.

Файлы cookie также очень пригодятся, когда вам нужно будет сохранить данные между HTTP-запросами страницы. Каждый созданный в браузере файл cookie доступен только для браузера, который его создал, что позволяет обезопасить хранящуюся в нем информацию.



ФАЙЛЫ COOKIE И НОВОЕ ЕВРОПЕЙСКОЕ ЗАКОНОДАТЕЛЬСТВО

В последнее время файлы cookie стали поводом для сомнений у веб-разработчиков, живущих в Европе или занимающихся раз-вернутыми в ней проектами. Европейский союз считает, что файлы cookie используются для хранения слишком большого объема лич-ных данных о посетителях веб-сайтов, и целью нового законода-тельства¹ является повышение прозрачности их использования.

Разработчикам не запрещается использовать файлы cookie, од-нако они вынуждены использовать их более эффективно и только тогда, когда это действительно необходимо.

По правде говоря, большая часть используемых веб-сайтами файлов cookie, безвредны. В основном они применяются, чтобы улучшить предлагаемый посетителям контент, исходя из пере-данной ими личной информации.

Сеансы: обзор

Какими бы полезными они ни были, файлы cookies уязвимы. Поскольку они хранятся на компьютере пользователя, они, в теории, могут быть из-менены пользователем или, что еще хуже, вирусом на компьютере поль-зователя. Сеансы являются альтернативой файлам cookie и не представ-ляют риска для безопасности на стороне клиента.

Сеансы являются особой формой хранения данных о разных запросах страниц во время посещения вашего сайта пользователем. При входе пользователя в систему, редактировании содержимого и выходе из си-стемы чаще всего за кадром выполняется сеанс.

Стоит отметить, что сеансы полагаются на файлы cookie для успешного сопоставления данных сеанса, хранящихся на сервере веб-сайта, с кли-ентом. Иными словами, файл cookie на клиентском компьютере требу-ется для его идентификации при каждом посещении сайта.

¹ www.ico.org.uk/for_organisations/privacy_and_electronic_communications/the_guide/cookies

Сравнение сеансов и файлов cookies

Поскольку вы используете язык PHP, вы можете без проблем применять и файлы cookie, и сеансы. И к тем, и к другим можно получить доступ в любом сценарии с помощью очень простых методов. Однако прежде чем мы перейдем к коду, очень важно разобраться в уникальных особенностях файлов cookie и сеансов, чтобы лучше понять, когда вы можете предпочесть одно другому.

Файлы cookie

Первой уникальной особенностью файлов cookie, отличающей их от сеансов, является то, что мы определяем для них дату окончания действия. Когда посетители покидают наш веб-сайт или веб-приложение и переходят к другим интернет-ресурсам, существует возможность того, что они могут однажды вернуться. С помощью файлов cookie мы можем установить дату окончания действия, чтобы при повторном посещении, если срок действия не истек, информация, представленная или введенная пользователем, могла быть отображена так, как будто они не покидали сайт.

Хорошим примером этого является авторизация в социальных сетях, вроде Facebook. После входа в систему вы можете перейти на другой сайт. Однако, если вы вернетесь на Facebook в течение определенного времени, вы можете обнаружить, что вы по-прежнему авторизованы в системе. Файлы cookie предоставляют сервису Facebook фрагмент данных, который позволяет загрузить профиль пользователя, что значительно улучшает опыт его взаимодействия с сервисом. Определение срока действия для файлов cookie обеспечивает удобство пользователя и в то же время повышает уровень безопасности. Обязывая пользователей повторно авторизоваться после определенного периода бездействия, вы предотвращаете доступ к данным, хранящимся в файле cookie, путем взлома аппаратного обеспечения.

Еще одна уникальная особенность файлов cookie заключается в том, что при каждом посещении веб-сайта, создавшего файл cookie на компьютере пользователя, этот файл автоматически включается в последующие HTTP-запросы, отправляемые клиентом. Учитывая это, важно не позволять вашим файлам cookie хранить слишком много данных, поскольку они будут влиять на трафик веб-сервера.

Файлы cookie упрощают управление данными, поскольку не требуют какой-либо базы данных для хранения данных или самих файлов cookie.

Это, в свою очередь, означает, что для использования этих файлов вы не нуждаетесь в навыках или знаниях об управлении базами данных. Кроме того, во многих языках веб-программирования, таких как PHP, функциональность для создания, назначения данных и получения доступа к файлам cookie встроена, так что вам не придется беспокоиться о создании собственной библиотеки для этой цели.

В целом файлы cookie следует применять для хранения небольших фрагментов информации, которые могут использоваться для персонализации отдельных частей веб-страницы с целью улучшения пользовательского опыта. Их также можно применять, чтобы позволить пользователям возобновить предыдущее взаимодействие с вашим приложением, создавая иллюзию, что они все еще авторизованы.



ФАЙЛЫ СООКІЕ И БЕЗОПАСНОСТЬ

Важно помнить, что в файлах cookie нельзя хранить конфиденциальные данные, поскольку они отправляются на веб-сервер каждый раз при запросе страницы и могут быть доступны технически подкованным злоумышленникам.

Кроме того, поскольку файлы cookie хранятся на компьютере пользователя, мы как разработчики мало что можем сделать для их защиты. В связи с этим файлы cookie в идеале должны содержать только данные, которые могут быть использованы приложением для идентификации информации, хранящейся в базах данных или других защищенных хранилищах, в противоположность хранению данных в самих файлах cookie.

Сеансы

Сеансы объединяют удобство файлов cookie с безопасностью и управляемостью веб-сервера. В отличие от файлов cookie данные конкретной переменной и информация не хранятся непосредственно на компьютере пользователя. Сами данные на самом деле хранятся на веб-сервере, на котором работает веб-сайт или веб-приложение.

Сеансы значительно отличаются от любых других переменных, которые мы рассматривали до этого, поскольку они не передают данные непосредственно между страницами, запрашиваемыми браузером. Вместо этого данные извлекаются из сеанса, который мы иницилируем в начале просмотра каждой страницы. Как упоминалось ранее, сеансы использу-

ют файлы cookie, хранящиеся на компьютере клиента, для правильного сопоставления пользователя и данных сеанса. Этот файл cookie не содержит какой-либо личной информации, кроме того, данные не имеют смысла без их сопоставления с данными, хранящимися на веб-сервере. Обычно они представлены в форме ключа, например 350401be75bbb0fafd3d912a1a1d5e54. По этой причине данные сеансов защищены лучше, чем данные, хранящиеся в файлах cookie.

Хотя метод хранения данных на клиентском компьютере и на сервере может показаться несколько неудобным, он имеет определенные преимущества.

- Сеансы не ограничивают объем хранящихся в них данных. Это связано с тем, что сеансы хранятся на сервере, поэтому это не сказывается отрицательно на его пропускной способности.
- Поскольку данные сеанса хранятся на веб-сервере, а не на компьютере пользователя, вы можете сосредоточить свои усилия по обеспечению безопасности на своем веб-сервере, а не на применении методов шифрования, согласующихся с возможностями браузера пользователя.

При использовании сеансов важно помнить о том, что вы можете указать, как долго они остаются активными. Мы называем это «временем жизни», и, как правило, сеанс заканчивается после того, как пользователь закрывает браузер или вкладку, которая инициировала сеанс с веб-сервером. Так, если пользователь авторизуется в социальной сети, а затем прекращает ее использовать и закрывает свой браузер, сеанс, если он настроен соответствующим образом, автоматически удаляет данные этого пользователя, что является очень полезной функцией.

Сеансы и файлы cookie в PHP

Теперь, когда мы рассмотрели цели использования сеансов и файлов cookie, а также основные различия между ними, пришло время разобратся в том, как их можно применять в PHP.

И сеансы, и файлы cookie могут быть доступны и загружены с помощью переменных, предназначенных специально для этого. В PHP мы выражаем намерение использовать файлы cookie с помощью суперглобальной переменной `$_COOKIE`. Сеансы используются аналогичным способом с помощью суперглобальной переменной `$_SESSION`.

Сначала давайте разберем создание и использование файлов cookie.

Файлы cookie в PHP

Для назначения данных файлам cookie существуют специальные функции. Давайте создадим новый файл с именем *storage.php* в нашей папке *experiments*. После создания файла добавьте в него следующий код:

```
<?php header("Content-Type: text/html;
↳ charset=utf-8");?>
<meta http-equiv="Content-Type" content="text/html;
↳ charset=utf-8"/>
<?php
$expire = time() + 60 * 60 * 24 * 30;
//Эта дата соответствует дате через 30 дней
setcookie("user", "Дима", $expire);
if (isset($_COOKIE["user"])) {
echo "Добро пожаловать, " . $_COOKIE["user"] . "!";
} else {
echo "Добро пожаловать, гость!";
}
```



ФУНКЦИЯ `isset()`

Функция `isset()` проверяет, присвоено ли переменной или элементу значение. В приведенном выше коде функция `isset()` используется, чтобы удостовериться в том, что ключ в массиве `$_COOKIE` имеет значение, прежде чем выполнение оператора `if` будет продолжено.

Если вы сейчас откроете файл *storage.php* в своем браузере, то увидите, что наш сценарий не обнаружил имя Дима и на самом деле считает нас гостем. Это связано с тем, что файл cookie установлен с помощью функции `setcookie()`. Это дает PHP инструкцию установить файл cookie в нашем браузере вместе с данными, которые должны будут в нем храниться, а также время жизни этого файла после того, как движок PHP обработает весь сценарий. Когда мы впервые получаем в браузере доступ к сценарию с нашим кодом, PHP не обнаруживает файлов cookie, из которых можно было бы извлечь данные. Это связано с тем, что код

PHP обрабатывается на сервере. Однако файлы cookie создаются в нашем браузере после завершения обработки сценария PHP-сервером. Таким образом, при первой загрузке страницы мы не увидим данных, содержащихся в файлах cookie.

Обновите файл *storage.php*, и вы увидите, что из файла cookie было извлечено и теперь отображается имя «Дима», введенное нами в функцию `setcookie()`. Вы также можете заметить, что первый аргумент, переданный в эту функцию, — это аргумент, который задает имя файла cookie. Затем мы используем этот аргумент, чтобы указать PHP, какой конкретно файл cookie мы хотим загрузить для получения нужных нам данных. Второй аргумент, как мы уже говорили, — это данные, которые мы хотим сохранить в этом файле cookie. В приведенном выше примере — это строка «Дима». Последним аргументом функции является «время жизни» этого файла cookie. Как правило, для файлов cookie необходимо установить дату и время, когда браузер должен будет их уничтожить. Тем не менее стоит иметь в виду, что файлы cookie, используемые сеансами, не требуют указания времени жизни, поскольку в данном случае оно определяется временем жизни сеанса.

В нашем коде мы установили срок жизни файла cookie и сохранили его в переменной `$expire`, находящейся в первой строке нашего кода. Обратите внимание на то, что мы использовали функцию PHP `time()`. Это позволяет узнать дату и время посещения браузером нашего сценария, а также дату и время его обработки PHP. Функция `time()` возвращает в виде числа количество секунд, прошедших с полуночи по Гринвичу 1 января 1970. Таким образом, вместо возвращения даты в понятном формате, например «May 14 12:50» («14 мая 12:50»), функция `time()` возвращает такое число, как 1368553800. Для получения дополнительной информации о функциях `time()` и `date()` обратитесь к документации по PHP¹.

Важно также отметить, что данные, содержащиеся в файле cookie, будут отправляться на сервер при каждом запросе страницы. Если ваш сценарий отправляет код HTML до отправки вашим сценарием данных файла cookie или указывает конкретные данные файла cookie для отправки на сервер, PHP выдаст сообщение об ошибке. Важно удостовериться в том, что вы устанавливаете и отправляете данные файла cookie в начале вашего сценария. Тем не менее вы можете извлечь данные из файла cookie далее в сценарии после отправки браузеру кода HTML. Это не приведет

¹ www.php.net/manual/ru/book.datetime.php

к появлению сообщения об ошибке PHP (при условии, что файл cookie существует и содержит данные). Я настоятельно рекомендую прочитать мою статью на сайте SitePoint, чтобы подробнее узнать о файлах cookie в PHP¹.

Сеансы в PHP

Теперь, когда мы получили базовые знания о создании и использовании файлов cookie, мы узнаем о том, как создавать и использовать сеансы. Так же, как и в случае с файлами cookie, этот метод подразумевает использование специальной суперглобальной переменной, с помощью которой мы назначаем и извлекаем данные сеанса: `$_SESSION`.

Сеансы работают подобно файлам cookie, но характеризуются определенным процессом назначения данных, который заметно отличается от того, как устанавливаются данные файлов cookie. Сначала мы должны сообщить PHP о том, что мы хотим использовать сеанс. Для этого мы применяем функцию `session_start()`. У нас также есть возможность удалить переменные сеанса, как поодиночке, так и все сразу, с помощью таких функций, как `session_unset()` и `session_destroy()`. Далее приведен простой пример того, как мы начинаем сеанс, добавляем в него данные и извлекаем эти данные с помощью переменной `$_SESSION`, о которой говорили ранее:

```
<?php
session_start();
$_SESSION["username"] = "myusername";
echo "Username = " . htmlspecialchars($_
↳SESSION["username"]);
```

Как видите, мы иницилируем сеанс, создаем новую переменную с именем `username` и присваиваем этой переменной строковое значение `"myusername"`. Наконец, в браузере мы отображаем строку, хранящуюся в переменной сеанса. Это простейший пример. Если бы мы использовали этот код в реальном проекте, мы бы разделили его на два файла. Первый файл отвечал бы за инициацию сеанса и создание переменной

¹ www.sitepoint.com/php-sessions

username, а второй также инициировал бы сеанс, а затем отображал бы строку, хранящуюся в переменной сеанса.

Важно завершить сеанс после окончания его использования. Несмотря на то, что сеансы применяются для временного хранения данных, они могут подвергаться атакам. Завершение сеанса помогает обеспечить максимальную безопасность при работе с потенциально важной информацией. Для этого мы добавляем в наш код функцию `session_destroy()`:

```
<?php
session_start();
$_SESSION["username"] = "Username";
echo "Username = " . $_SESSION["username"];
session_destroy();
```

Чтобы предотвратить попадание в ваши сценарии вредоносного кода, рекомендуется завершить сеанс после окончания работы с ним.

Если мы хотим не завершить сеанс, а только удалить из него часть данных, мы можем использовать функцию `unset()`. Ниже приведен пример использования этой функции:

```
<?php
session_start();
unset($_SESSION["username"]);
```

Мы также можем сбросить все сеансы и их значения с помощью функции `session_unset()`. Таким образом, если изменить вышеприведенный пример, мы можем использовать функцию `session_unset()` следующим образом:

```
<?php
session_start();
session_unset();
```



НЕ ХРАНИТЕ В СЕАНСАХ ВАЖНЫЕ ДАННЫЕ

Ранее в этой главе мы говорили о том, что файлы cookie не следует использовать для хранения важных данных, даже временно. Этот же совет можно применить к сеансам, несмотря на то, что они считаются более безопасными. Хотя данные сеанса хранятся на веб-сервере, а не на компьютере пользователя, сеансы также могут подвергаться атакам хакеров. Таким образом, сеансы не рекомендуется использовать для хранения важных данных.

Работа над проектом

Теперь, когда мы рассмотрели способы использования сеансов и файлов cookie, пора приступить к наполнению оболочки нашего проекта, чтобы превратить его в полноценную блог-платформу. Начнем с добавления кода в файл *login.php* из папки *includes*. Сначала отредактируем метод `index()`:

```
public function index() {
    if (!empty($_GET['status']) && $_GET['status'] ==
        'logout') {
        session_unset();
        session_destroy();
        $error = 'Ваш сеанс завершен. Пожалуйста,
        авторизуйтесь снова.';
        require_once 'admin/templates/loginform.php';
    } elseif (!empty($_SESSION['kickstart_login']) &&
        $_SESSION['kickstart_login']) {
        header('Location: ' . $this->base->url .
            '/admin/posts.php');
        exit();
    } else {
        if ($_SERVER['REQUEST_METHOD'] === 'POST') {
            $this->validateDetails();
        } elseif (!empty($_GET['status'])) {
            if ($_GET['status'] == 'inactive') {
```

```
session_unset();
session_destroy();
$error = 'Сеанс завершен в связи с
отсутствием активности. Пожалуйста, авторизуйтесь
↪ снова.';
}
}
require_once 'admin/templates/loginform.php';
}
}
```

Изменения в методе `index()` сводятся к представлению сеанса нашему сценарию `login`. Такое использование сеансов позволяет отслеживать активность пользователей в панели администратора, и в случае их неактивности они будут выведены из системы в целях повышения уровня безопасности.

Мы также добавили процесс выхода из системы, который завершит любой активный сеанс при выходе пользователя из системы. Это опять же делается для того, чтобы повысить уровень безопасности, а также позволить пользователю контролировать доступ его сеанса к панели администратора. Наконец, мы добавили условный оператор `if`, чтобы проверить, возвращается ли пользователь к панели администратора и является ли его сеанс по-прежнему активным. Если это так, наше приложение не будет предлагать ему снова авторизоваться в системе, что положительно скажется на пользовательском опыте.

Далее мы кратко рассмотрим метод `loginSuccess()`:

```
public function loginSuccess() {
    $_SESSION['kickstart_login'] = true;
    $_SESSION["timeout"] = time();
    header('Location: ' . $this->base->url . '/admin/
    ↪ posts.php');
    return;
}
```

Здесь мы добавили две новые переменные сеанса и соответствующие им данные после входа пользователя в систему и получения им доступа к панели администратора. Наше приложение использует обе переменные сеанса, чтобы определить, является ли текущий пользователь авторизованным во время использования панели администратора приложения, и оценить уровень его активности.

Это единственные изменения, которые мы внесем в сценарий *login.php*. Теперь давайте обратимся к файлу *admin.php*, который также находится в папке *includes*. Начнем с добавления кода в начало нашего файла, а также в конструктор класса Adminpanel:

```
<?php
session_start();
require_once 'database.php';
class Adminpanel {
public function __construct() {
    $inactive = 600;
    if (isset($_SESSION["kickstart_login"])) {
        $sessionTTL = time() - $_SESSION["timeout"];
        if ($sessionTTL > $inactive) {
            session_unset();
            session_destroy();
            header("Location: http://" .
                $_SERVER['SERVER_NAME'] . "/login.
                ↵php?status=inactive");
        }
    }
    $_SESSION["timeout"] = time();
    $login = $_SESSION['kickstart_login'];
    if (empty($login)) {
        session_unset();
        session_destroy();
        header('Location:
            http://'. $_SERVER['SERVER_NAME'] . '/login.
            ↵php?status=loggedout');
```

```
} else {  
$this->ksdb = new Database;  
$this->base = (object) '';  
$this->base->url =  
'http://'.$_SERVER['SERVER_NAME'];  
}  
}  
}
```

Сначала мы добавили функцию `session_start()` в верхней части нашего файла, чтобы инициировать сеанс каждый раз, когда браузер пользователя запрашивает страницу в панели администратора. Затем код, добавленный в конструктор, заставляет класс `Adminpanel` проверить, насколько активным был наш пользователь при использовании панели администратора, и в случае его неактивности наше приложение, завершив сеанс, выведет пользователя из системы и направит к экрану авторизации.

Если он по-прежнему активен, класс проверяет, не отправлял ли пользователь запрос на выход из системы и не пытался ли он получить доступ к панели администратора напрямую без предварительной авторизации. В любом случае пользователь будет перенаправлен обратно к экрану авторизации.

Наконец, если пользователь уже вошел в систему и использует активный сеанс, класс приступит к загрузке необходимых данных для правильного отображения панели администратора.

Далее давайте рассмотрим класс `Posts`, который расширяет класс `Adminpanel`:

```
public function __construct() {  
parent::__construct();  
if (!empty($_GET['action'])) {  
switch ($_GET['action']) {  
case 'create':  
$this->addPost();  

```

```
break;
default:
$this->listPosts();
break;
case 'save':
$this->savePost();
break;
case 'delete':
$this->deletePost();
break;
}
} else {
$this->listPosts();
}
}
```

Здесь мы добавили новый случай в условный оператор `switch`. Он проверяет, содержит ли переменная `URL` action строку `delete`. Если содержит, наше приложение должно загрузить новый метод `deletePost()`.

Давайте добавим этот метод в нижнюю часть класса:

```
public function deletePost() {
if (!empty($_GET['id']) && is_numeric($_GET['id'])) {
$query = "DELETE FROM `posts` WHERE id = ?"
$stmt = $this->db->prepare($query);
$stmt->execute(array($_GET['id']));
$delete = $stmt->rowCount();
$this->db = null;
if (!empty($delete) && $delete > 0) {
header("Location: " . $this->base->url .
"/posts.php?delete=success");
} else {
```



```
header("Location: " . $this->base->url .  
"/posts.php?delete=error");  
}  
}  
}
```

Выше мы добавили функциональность, необходимую для того, чтобы мы могли удалять сообщения с помощью кнопки **Удалить**, которую мы видим, когда в панели администратора перечислены все наши сообщения. Для этого мы не только добавили необходимую функциональность в метод `deletePost()`, но также добавили в конструктор `Posts` код, необходимый для обнаружения метода `delete` при его запросе в URL-адресе. Мы сделали это, добавив новый случай `switch case` в конструктор `Posts`, чтобы по переменной URL определить, какое действие хотим совершить по отношению к выбранному в панели администратора сообщению.

Мы также добавили функциональность в конструкторе и методе `editPosts()`, которая позволяет нам редактировать уже созданные сообщения.

Наконец, нам необходимо обновить метод `savePost()`:

```
public function savePost() {  
    $array = $format = $return = array();  
    if (!empty($_POST['post'])) {  
        $post = $_POST['post'];  
    }  
    if (!empty($post['content'])) {  
        $array['content'] = $post['content'];  
        $format[] = ':content';  
    }  
    $cols = $values = '';  
    $i=0;  
    foreach ($array as $col => $data) {  
        if ($i == 0) {
```

```
$cols .= $col;
$values .= $format[$i];
} else {
    $cols .= ', ' . $col;
    $values .= ', ' . $format[$i];
}
$i++;
}
try {
    //Этот запрос подвержен риску внедрения SQL-кода
    $query = $this->ksdb->db->prepare("INSERT INTO
posts (". $cols .") VALUES (". $values .")");
    for ($c = 0; $c < $i; $c++) {
        $query->bindParam($format[$c], ${'var' . $c});
    }
    $z=0;
    foreach ($array as $col => $data) {
        ${'var' . $z} = $data;
        $z++;
    }
    $result = $query->execute();
    $add = $query->rowCount();
} catch (PDOException $e) {
    echo $e->getMessage();
}
$query->closeCursor();
$this->db = null;
if (!empty($add)) {
    $status = array('success' => 'Ваше сообщение успешно
↳ сохранено. ');
} else {
    $status = array('error' => 'В процессе сохранения
↳ вашего сообщения возникла ошибка. Пожалуйста,
↳ повторите попытку позднее. ');
```

```

}
header("Location: http://localhost/kickstart/admin/
↳posts.php");
}

```

Изменения, внесенные в метод `savePost()`, дают нашему приложению возможность обработать дополнительное поле при создании и редактировании наших сообщений. Это новое поле позволяет ввести название сообщения. До сих пор названия наших сообщений представляли собой просто уникальные идентификаторы. Тем не менее, поскольку мы создали новое поле для сообщения, это означает, что мы должны добавить новый столбец в таблицу `posts` в нашей базе данных `kickstartapp`. Вы можете сделать это либо с помощью запроса MySQL, либо с помощью пользовательского интерфейса phpMyAdmin. Рекомендуется установить для этого столбца тип `VARCHAR` с длиной около 100. Такое поле может вместить практически любое название. После добавления этого столбца ваша таблица должна выглядеть, как на рис. 5.1.

#	Имя	Тип	Сравнение	Атрибуты	Null	По умолчанию	Дополнительно	Действие
1	id	int(11)		Нет	Нет		AUTO_INCREMENT	Изменить Удалить Первичный Уникальный Индекс Ещё
2	content	longtext	utf8_general_ci	Нет	Нет			Изменить Удалить Первичный Уникальный Индекс Ещё
3	title	varchar(100)	utf8_general_ci	Нет	Нет			Изменить Удалить Первичный Уникальный Индекс Ещё

Рис. 5.1. Таблица `posts` в базе данных `kickstartapp`

Теперь нам нужно обновить шаблон `manageposts.php`:

```

<?php header("Content-Type: text/html;
↳charset=utf-8");?>
<meta http-equiv="Content-Type" content="text/html;
↳charset=utf-8"/>
<?php require_once '_inc/header.php';?>
<a href="<?php echo $this->base->url . '/posts.
↳php?action=create';?>" class="btn btn-info">
↳Создать сообщение</a>
<a href="<?php echo $this->base->url . '/comments.

```

```

<?php'; ?>" class="btn btn-info">Комментарии</a>
<table>
<thead>
<tr>
<td>Заголовок</td>
<td>Содержимое</td>
<td>Действия</td>
</tr>
</thead>
<tbody>
<?php foreach($posts as $post): ?>
<tr>
<td><h3><?php echo (!empty($post['title']) ?
htmlspecialchars($post['title']) : 'Сообщение №' .
<?php echo htmlspecialchars
($post['id'])); ?></h3></td>
<td><p><?php echo implode(' ',
array_slice(explode(' ',strip_tags($post['content'])),
<?php echo 0, 10)); ?> [...]</p></td>
<td><a href="<?php echo $this->base->url .
"/posts.php?id=" . $post['id'] . "&action=edit"; ?>"
class="btn btn-primary">Править</a><a href="<?php
<?php echo $this->base->url . "/posts.php?id=" . $post
['id'] . "&action=delete"; ?>" class="btn
btn-primary">Удалить</a></td>
</tr>
<?php endforeach; ?>
</tbody>
</table>
<?php require_once '_inc/footer.php'; ?>

```

Теперь в нашем шаблоне появилось дополнительное поле, в которое мы можем добавить названия наших сообщений. Оно будет связываться

с кодом, добавленным в метод `savePost()`, чтобы поместить информацию в базу данных.

Мы также должны добавить это новое поле в шаблон *newpost.php*, который мы используем для создания наших сообщений.

Теперь этот файл должен выглядеть следующим образом:

```
<?php header("Content-Type: text/html;
↳ charset=utf-8");?>
<meta http-equiv="Content-Type" content="text/html;
↳ charset=utf-8"/>
<?php require_once '_inc/header.php'; ?>
<form action="<?php echo $this->base->url . '/posts.
↳ php?action=save'; ?>" class="row" method="POST">
<section class="span7">
<h3>Новое сообщение</h3>
<div class="control-group">
<label class="control-label"
for="content">Заголовок</label>
<div class="controls">
<input type="text" name="post[title]"
id="title" placeholder="Заголовок вашего сообщения" />
</div>
</div>
<div class="control-group">
<label class="control-label" for="wmd-
input">Содержимое</label>
<div class="wmd-panel controls">
<div id="wmd-button-bar"></div>
<textarea class="wmd-input"
name="post[content]" id="wmd-input">*Вводите текст
↳ здесь*</textarea>
</div>
</div>
```

```

<div class="control-group">
<div class="controls">
<button type="submit"
class="btn">Сохранить сообщение</button>
</div>
</div>
</section>
<section class="span4" style="padding-left:20px;
↳border-left:1px solid #eee;">
<div id="wmd-preview" class="wmd-panel wmd-
preview"></div>
</section>
</form>
<?php require_once '_inc/footer.php'; ?>

```

Кроме добавления дополнительного поля, в котором пользователь может ввести название сообщения, мы также изменили текстовую область ввода содержимого. Теперь оно представляет собой поле ввода markdown. Синтаксис языка разметки markdown¹ был разработан, чтобы облегчить форматирование содержимого.

Движок markdown преобразует его, чтобы использовать соответствующее стандартам HTML-форматирование.

Для нашего проекта мы применим простой для реализации конвертер JavaScript Markdown и редактор Pagedown². Чтобы использовать библиотеку, вам необходимо загрузить три библиотеки Pagedown: Markdown.Converter, Markdown.Sanitizer и Markdown.Editor, именно в таком порядке. После загрузки вы инициализируете их, добавив следующий код в раздел заголовка вашего шаблона *header.php*:

```

<script>
(function () {
var converter = new Markdown.Converter();

```

¹ daringfireball.net/projects/markdown

² code.google.com/p/pagedown

```
var editor = new Markdown.Editor(converter);  
editor.run();  
}());  
</script>
```

Вам также необходимо убедиться в том, что у вас есть подходящие элементы, чтобы загрузить кнопки форматирования, а также сообщить текстовой области правильный класс, который поможет библиотекам решить, куда следует загружать код, относящийся к форматированию текста. Кроме того, вам нужно указать, где библиотеки должны отображать сообщение в режиме предварительного просмотра при вводе и форматировании содержимого.

Мы уже добавили оба эти элемента и необходимые им классы в вышеприведенный шаблон. Давайте разберем добавленный код.

Ниже приведен код для кнопок форматирования и ввода текста:

```
<div class="wmd-panel controls">  
<div id="wmd-button-bar"></div>  
<textarea class="wmd-input" name="post[content]"  
id="wmd-input">*Вводите текст здесь*</textarea>  
</div>  
А это код для области предварительного просмотра:  
<section class="span4" style="padding-left:20px;  
↳border-left:1px solid #eee;">  
<div id="wmd-preview" class="wmd-panel wmd-preview">  
↳</div>  
</section>
```

Вы можете определить, какие области требуются библиотекам Page-down, найдя элементы с префиксом `wmd`, указанным перед идентификаторами и/или классами. После загрузки этих библиотек вы должны получить возможность вводить содержимое сообщения, а также использовать некоторые возможности форматирования markdown в редакторе, как показано на рис. 5.2.

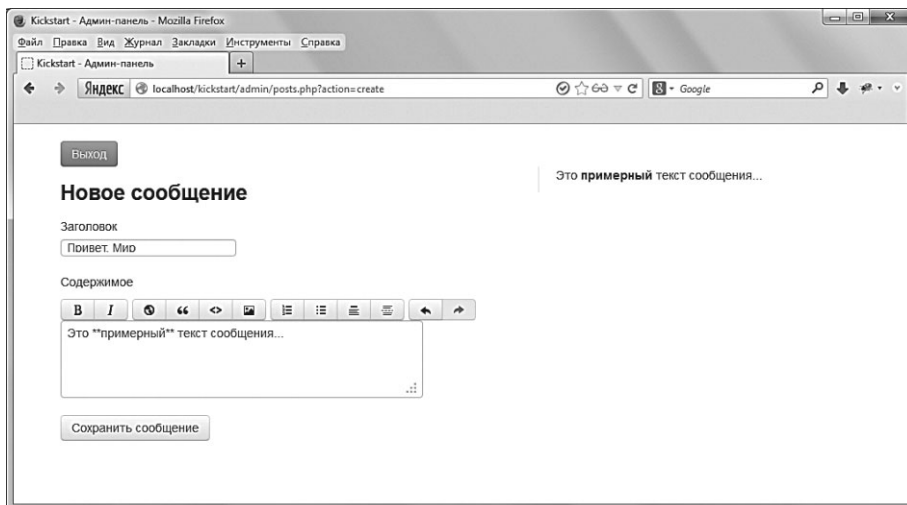


Рис. 5.2. Создание сообщения в текстовом редакторе

Вы также должны заметить, что в верхней части шаблона появилась новая кнопка **Комментарии**.

Щелкнув по этой кнопке, пользователь перейдет к панели управления комментариями, где он сможет просматривать все добавленные к сообщениям комментарии, а также удалять их.

Чтобы это обеспечить, в первую очередь нам нужно создать в базе данных новую таблицу, где будут храниться комментарии пользователей.

Настройте новую таблицу в базе данных в соответствии с рис. 5.3.

#	Имя	Тип	Сравнение	Атрибуты	Null	По умолчанию	Дополнительно	Действие
1	id	int(11)			Нет	Нет	AUTO_INCREMENT	Изменить Удалить Ещё
2	postid	int(11)			Нет	Нет		Изменить Удалить Ещё
3	email	varchar(200)	utf8_general_ci		Нет	Нет		Изменить Удалить Ещё
4	name	varchar(100)	utf8_general_ci		Нет	Нет		Изменить Удалить Ещё
5	comment	longtext	utf8_general_ci		Нет	Нет		Изменить Удалить Ещё

Рис. 5.3. Таблица `comments`

Чтобы обеспечить функционирование данного раздела панели администратора, откроем файл `admin.php` в папке `includes` и изменим код конструктора:


```
public function __construct() {
    parent::__construct();
    if (!empty($_GET['action']) && $_GET['action']
        == 'delete') {
        $this->deleteComment();
    } else {
        $this->listComments();
    }
}
```

Как мы видим, наш конструктор теперь проверяет, существует ли переменная URL `action` и содержит ли она строковое значение `delete`. Если да, то приложение загружает метод `deleteComment()`, а если нет, — то метод `listComments()`. Давайте сначала создадим метод `listComments()`:

```
public function listComments() {
    $comments = $return = array();
    $query = $this->ksdb->db->prepare("SELECT * FROM
        comments");
    try {
        $query->execute();
        for ($i = 0; $row = $query->fetch(); $i++) {
            $return[$i] = array();
            foreach ($row as $key => $rowitem) {
                $return[$i][$key] = $rowitem;
            }
        }
    } catch (PDOException $e) {
        echo $e->getMessage();
    }
    $comments = $return;
    require_once 'templates/managecomments.php';
}
```

В этом методе наше приложение загружает все комментарии из таблицы в базе данных, используя расширение PDO. После извлечения из базы данных комментарии загружаются в массив, содержащий уникальный ключ для каждого комментария. При возникновении проблем с подключением к базе данных или с запросом PDO наше приложение выдаст сообщение об ошибке. Наконец, метод загружает файл шаблона *managecomments.php*. Теперь нам нужно создать метод `deleteComment()`:

```
public function deleteComment() {
    if (!empty($_GET['id']) && is_numeric($_GET['id'])) {
        $query = "DELETE FROM `comments` WHERE id = ?";
        $stmt = $this->db->prepare($query);
        $stmt->execute(array($_GET['id']));
        $delete = $result->rowCount();
        $this->db = null;
        if(!empty($delete) && $delete > 0){
            header("Location: ".$this->base->url."/
            comments.php?delete=success");
        }else{
            header("Location: ".$this->base->url."/
            comments.php?delete=error");
        }
    }
}
```

Метод `deleteComment()` очень похож на метод `deletePost()` из класса `Posts` за исключением того, что теперь наше приложение ищет переменную URL `id`, содержащую идентификатор комментария относительно его строки в нашей базе данных. У нас также есть условный оператор `if`, который перенаправляет пользователя к панели управления комментариями с помощью сообщения об успешной или неудачной попытке удалить один из них.

Теперь нам нужно создать шаблон с именем *managecomments.php* для перечисления комментариев в нашей панели администратора. После создания этого файла мы должны добавить в него следующий код:

```

<?php header("Content-Type: text/html;
↳ charset=utf-8");?>
<meta http-equiv="Content-Type" content="text/html;
↳ charset=utf-8"/>
<?php require_once '_inc/header.php'; ?>
<a href="<?php echo $this->base->url . '/posts.
↳ php?action=create'; ?>" class="btn btn-info">Создать
↳ сообщение</a>
<a href="<?php echo $this->base->url . '/comments.
↳ php'; ?>"
class="btn btn-info">Комментарии</a>
<table cellpadding="10">
<thead>
<tr>
<td>Автор комментария</td>
<td>Идентификатор сообщения</td>
<td>Комментарий</td>
<td>Действия</td>
</tr>
</thead>
<tbody>
<?php foreach($comments as $comment): ?>
<tr>
<td><h4><?php echo
htmlspecialchars($comment['name']);?></h4></td>
<td><p><?php echo
htmlspecialchars($comment['email']);?></p></td>
<td><p><?php echo
htmlspecialchars($comment['comment']);?></p></td>
<td><a href="<?php echo $this->base->url .
"/comments. php?id=" . htmlspecialchars($comment
↳ ['id']) . "&action=delete"; ?>" class="btn
↳ btn-primary">Удалить комментарий</a></td>
</tr>

```

```
<?php endforeach; ?>
</tbody>
</table>
<?php require_once '_inc/footer.php'; ?>
```

После добавления этого кода у вас появится работающая панель администратора, показанная на рис. 5.4. Вы можете смело использовать ее для создания сообщений.

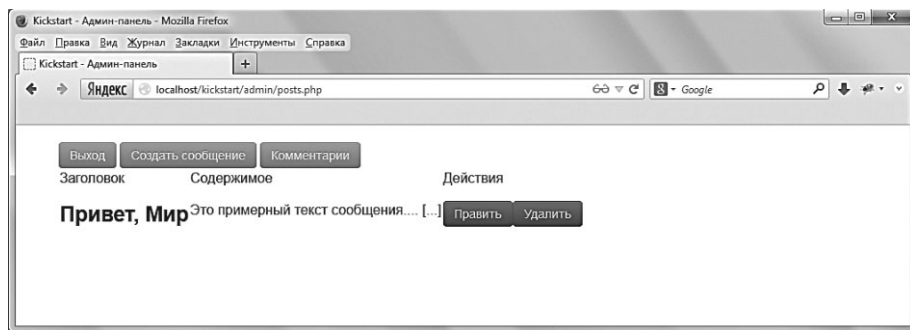


Рис. 5.4. Панель администратора в действии

По завершении тестирования панели администратора вам следует добавить последние штрихи к нашей блог-платформе и приступить к дальнейшему расширению функциональности для пользовательского интерфейса нашего приложения.

Во-первых, нам нужно дополнить пользовательский интерфейс новой библиотекой, чтобы публичная часть нашего блога могла преобразовать форматирование Markdown в соответствующий стандарт HTML-код. Для этого мы добавляем библиотеку Markdown¹ для РНР Мишеля Фортина. Ее очень легко установить, и она прекрасно обрабатывает синтаксис Markdown, кроме того, она написана исключительно на языке РНР, поэтому нам не придется устанавливать дополнительное программное обеспечение на сервере. Нам просто нужно вызвать библиотеку, когда потребуется преобразовать форматирование Markdown в код HTML.

Загрузите библиотеку и переместите файл *markdown.php* в каталог *includes*, находящийся в главной папке проекта. Затем откройте файл *posts.php* из каталога *includes* и добавьте в него следующий код:

¹ michelf.ca/projects/php-markdown

```
public function viewpost($postid) {
    $id = $postid;
    $posts = $this->ksdb->dbselect('posts', array('*'),
        ↳array('id' => $id));
    $markdown = new Michelf\Markdown();
    $posts[0]['content'] = $markdown->
        ↳defaultTransform($posts[0]['content']);
    $postcomments = $this->comments->getcomments($posts[0]
        ↳['id']);
    $template = 'view-post.php';
    include_once 'frontend/templates/'.$template;
}
```

Мы включили новую библиотеку Markdown в начало нашего сценария, подготовив ее к использованию в тот момент, когда приложению потребуется отобразить данные в формате Markdown. Мы изменили функциональность в нашем методе `getposts()`, а также в методе `viewpost()`, и теперь загружаем класс `Comments` в нашем конструкторе. Для начала подробно рассмотрим код, которым мы дополнили два метода. В методе `getposts()` мы добавили новый цикл `foreach`, который будет вызывать метод `commentnumber()` в нашем классе `Comments`. Нам по-прежнему нужно добавить этот метод в класс, поэтому, имея это в виду, рассмотрим метод `viewpost()`. Здесь мы загружаем класс `markdown` из библиотеки `Markdown` и, присваивая этот класс переменной `$markdown`, превращаем его в объект.

Вы можете заметить, что вызов библиотеки `Markdown` довольно сильно отличается от того, как мы вызывали классы ранее. Это связано с тем, что Мишель использовал в своей библиотеке *пространство имен*. Пространство имен помогает установить для класса уникальный идентификатор. Это позволяет избежать возникновения конфликта при загрузке двух одноименных классов.

Концепция пространства имен была реализована в версии PHP 5.3. Эта функция начинает набирать популярность, поскольку она помогает разрешить сложности, связанные с присвоением имен при создании библиотек для общественного использования. Пространства имен могут применяться к классам следующим образом:

```
<?php header("Content-Type: text/html;  
↳ charset=utf-8");?>  
<meta http-equiv="Content-Type" content="text/html;  
↳ charset=utf-8"/>  
<?php  
namespace Project;  
class myProject {  
function index() {  
echo 'Загрузка метода из класса с помощью пространства  
↳ имен!';  
}  
}
```

Несмотря на простоту реализации, пространства имен могут значительно повысить работоспособность кода. Например, РНР-библиотека Markdown использует пространство имен Michelf, поэтому для загрузки класса библиотеки и его методов в сценарии нам просто нужно, чтобы пространство имен находилось в нашем сценарии перед командой для вызова класса. В этом случае вместо того, чтобы писать:

```
$markdown = new Markdown();
```

Нам нужно написать:

```
$markdown = new Michelf\Markdown();
```



ПОДРОБНЕЕ О ПРОСТРАНСТВАХ ИМЕН

Для получения дополнительной информации о пространствах имен обратитесь к руководству по РНР¹ и к статье на сайте SitePoint².

Теперь, когда мы реализовали нашу библиотеку Markdown, мы можем использовать ее для преобразования сообщений, написанных с использованием синтаксиса Markdown, в полноценный код HTML.

¹ www.php.net/manual/ru/language.namespaces.php

² www.sitepoint.com/php-namespaces

Последний добавленный фрагмент кода отвечает за вызов другого метода в нашем классе `Comments`, код которого нам еще предстоит написать. Этим методом является `getcomments()`, он заставит наше приложение загрузить все комментарии, относящиеся к конкретному сообщению.

Откройте файл *comments.php*, находящийся в каталоге *frontend*. Измените код следующим образом:

```
<?php
require_once 'frontend/posts.php';
class Comments extends Blog {
public function __construct() {
parent::__construct();
if ($_SERVER['REQUEST_METHOD'] === 'POST' &&
!empty($_POST['comment'])) {
$this->addcomment();
}
}
```

Наш конструктор для этого класса загружает все, что нужно, из конструктора родительского класса (который является конструктором класса `Blog`), затем проверяет, есть ли запрос на добавление нового комментария. При его наличии приложение запускает метод `addcomment()`. В этом методе мы берем все данные из формы отправки комментариев и подготавливаем их к размещению в нашей базе данных, используя метод `dbadd()` из библиотеки базы данных PDO.

```
public function commentnumber($postid) {
$query = $this->ksdb->dbselect('comments',
array('*'), array('postid' => $postid));
$commentnum = count($query);
if ($commentnum <= 0) {
$commentnum = 0;
}
```

```
return $commentnum;
}
public function getcomments($postid) {
return $this->ksdb->dbselect('comments',
array('*'), array('postid' => $postid));
}
```

Как видите, у нас есть метод `getcomments()`, которому требуется получить переменную, содержащую идентификатор сообщения. Любые данные, которые необходимо передать методу, прежде чем он сможет быть корректно обработан, можно поместить в скобках после названия метода. Метод `getcomments()` собирает все комментарии, относящиеся к сообщению, исходя из переданного методу идентификатора, и возвращает их переменной, определенной при вызове этого метода. Это означает, что при передаче ему данных PHP выдает сообщение об ошибке:

```
<?php header("Content-Type: text/html;
↳ charset=utf-8");?>
<meta http-equiv="Content-Type" content="text/html;
↳ charset=utf-8"/>
public function addcomment() {
$status= '';
$array = $format = array();
if (!empty($_POST['comment'])) {
$comment = $_POST['comment'];
}
if (!empty($comment['fullname'])) {
$array['name'] = $comment['fullname'];
$format[] = ':fullname';
}
if (!empty($comment['email'])) {
$array['email'] = $comment['email'];
$format[] = ':email';
```



```

}
if (!empty($comment['context'])) {
$array['comment'] = $comment['context'];
$format[] = ':context';
}
if (!empty($comment['postid'])) {
$array['postid'] = $comment['postid'];
$format[] = ':postid';
}
$add = $this->ksdb->dbadd('comments', $array,
↳$format);
if (!empty($add) && $add > 0) {
$status = array('success' => 'Ваш комментарий
сохранен');
$key = 'success';
} else {
$status = array('error' => 'В процессе
сохранения вашего комментария возникла ошибка.
↳Пожалуйста, повторите попытку позднее.');
```

У нас также есть метод `commentnumber()`, который перед выполнением должен получить идентификатор сообщения. Этот метод пересчитывает все комментарии, относящиеся к идентификатору сообщения, который передается методу, и возвращает полученное число.

Теперь у нас есть работающий класс `Comments`. Осталось настроить шаблон *list-posts.php*, чтобы показать количество комментариев к нашим сообщениям, а также их новые заголовки, и добавить раздел в шаблон *view-post.php*, чтобы посетители могли оставлять комментарии (и опять же отображать заголовки сообщений).

Давайте начнем с файла *list-posts.php*. Откройте его и добавьте следующий код:

```
<?php header("Content-Type: text/html;
↳ charset=utf-8");?>
<meta http-equiv="Content-Type" content="text/html;
↳ charset=utf-8"/>
<?php require_once 'includes/temps/header.php'; ?>
<?php foreach ($posts as $post): ?>
<h3><?php echo (!empty($post['title'])) ?
↳ htmlspecialchars($post['title']) : 'Сообщение №' .
↳ htmlspecialchars($post['id'])); ?></h3>
<p><?php echo implode(' ', array_slice(explode('
', strip_tags($post['content'])), 0, 10)); ?>
↳ [...]</p>
<a href="<?php echo $this->base->url . "?id=" .
$post['id']; ?>" class="btn btn-primary">Подробнее</a>
↳ <p>Комментарии: <?php echo $post['comments']; ?></p>
<hr/>
<?php endforeach; ?>
<?php require_once 'includes/temps/footer.php'; ?>
```

Теперь у нас есть небольшой раздел, в котором указано количество комментариев для каждого сообщения, а также название сообщения и небольшая выдержка из него.

Наконец, добавим следующий код в шаблон *view-post.php*:

```
<?php header("Content-Type: text/html;
↳ charset=utf-8");?>
<meta http-equiv="Content-Type" content="text/html;
↳ charset=utf-8"/>
<?php require_once 'includes/temps/header.php'; ?>
<br/>
```

```

<a href="<?php echo $this->base->url; ?>" class="btn
↳btn-primary">Вернуться к списку сообщений</a>
<article>
<?php foreach($posts as $post): ?>
<h3><?php echo (!empty($post['title'])) ?
↳htmlspecialchars($post['title']) : 'Post #' .
↳htmlspecialchars($post['id'])); ?></h3>
<?php echo $post['content']; ?>
<hr/>
<?php endforeach; ?>
<h3>Комментарии</h3>
<?php foreach ($postcomments as $comment): ?>
<section class="span3">
<figure>

</figure>
<h4><?php echo htmlspecialchars($comment['name']);
?></h4>
<p><small><?php echo
htmlspecialchars($comment['email']); ?></small></p>
</section>
<section class="span8">
<p><?php echo
htmlspecialchars($comment['comment']); ?></p>
</section>
<hr style="clear:both;"/>
<?php endforeach; ?>
<br/>

```

В этом коде мы добавили раздел, который отображает в верхней части каждого сообщения его заголовок. Также мы создали область, отображающую каждый добавленный комментарий. Следует отметить, что при

добавлении раздела, где отображается заголовок сообщения, условный оператор проверяет, имеет ли оно заголовок. Если нет, наш сценарий отобразит номер сообщения, а также строку "Post #", чтобы обеспечить резервный заголовок на случай его отсутствия. Важно убедиться в том, что при вызове в сценариях данных, которые не всегда могут там присутствовать, у вас есть запасной вариант, чтобы в функционирующем приложении или на сайте не отображались сообщения об ошибке.

```
<?php header("Content-Type: text/html;
↳ charset=utf-8");?>
<meta http-equiv="Content-Type" content="text/html;
↳ charset=utf-8"/>
<h3>Оставьте комментарий</h3>
<form action="<?php echo htmlspecialchars($_SERVER
↳ ['PHP_SELF']); ?>" method="post" class="form-
↳ horizontal">
<input type="hidden" value="<?php echo $_GET['id'];
↳ ?>" name="comment[postid]" />
<div class="control-group <?php echo (!empty($error)?
↳ 'error':''); ?>">
<label class="control-label"
for="email">Email</label>
<div class="controls">
<input type="email" name="comment[email]"
id="email" placeholder="Адрес электронной почты"/>
</div>
</div>
<div class="control-group <?php echo (!empty($error) ?
↳ 'error' : ''); ?>">
<label class="control-label" for="name">Full
Name</label>
<div class="controls">
<input type="text" name="comment[fullname]"
id="name" placeholder="Ваше полное имя"/>
</div>
```

```
</div>
<div class="control-group <?php echo (!empty($error) ?
↳ 'error' : ''); ?>">
<label class="control-label"
for="comment">Комментарий</label>
<div class="controls">
<textarea id="comment"
name="comment[context]"></textarea>
</div>
</div>
<div class="control-group">
<div class="controls">
<button type="submit" class="btn">Отправить
комментарий</button>
</div>
</div>
</form>
</article>
<?php require_once('includes/temps/footer.php'); ?>
```

В приведенном выше коде мы добавили новую форму, где посетители блога могут оставить комментарий. С помощью метода POST форма отправляет данные самой себе, после чего наше приложение сможет обработать информацию и сохранить ее в базе данных, используя метод `addComment()`. Мы также добавили операторы `if` для каждого из элементов формы на случай, если пользователь забудет заполнить поле при отправке комментария. У нас также есть скрытое поле, содержащее идентификатор сообщения, требующийся нам для того, чтобы корректно сохранить комментарий в базе данных.



ИСПОЛЬЗОВАНИЕ СЕРВИСА GRAVATAR ДЛЯ АВАТАРОВ ПОЛЬЗОВАТЕЛЕЙ

Следует отметить, что мы добавили ссылку на сервис Gravatar¹ для того, чтобы поместить аватар пользователя рядом с коммен-

¹ ru.gravatar.com

тарием. Gravatar — это бесплатный сервис, который позволяет пользователям загрузить аватар на сервер и связать с ним свой адрес электронной почты. Благодаря ему разработчики могут запрограммировать запрос, отправляемый на сервер Gravatar для получения аватара, соответствующего предоставленному пользователем адресу электронной почты.

Вызов сервера Gravatar осуществляется бесплатно, это означает, что вам не нужно хранить аватары. Целью сервиса Gravatar является обеспечение актуальности и согласованности аватаров пользователей. Если пользователь не загрузил свой аватар на сервер Gravatar, то вам будет возвращен используемый по умолчанию пустой аватар. Каждый адрес электронной почты кодируется с помощью простого метода, поэтому мы должны закодировать адрес электронной почты пользователя, используя алгоритм шифрования md5. Мы сделали это в нашем коде для извлечения аватара с сервера Gravatar.

Вернемся к нашему коду. В нижней части нашего шаблона появилась новая область формы, где пользователи могут ввести данные о себе, а также оставить комментарий к записи в блоге.

Теперь все, что нужно сделать, — это сохранить данный шаблон, после чего у нас будет полностью функционирующая блог-платформа с комментариями, как показано на рис. 5.5.

Поздравляем!

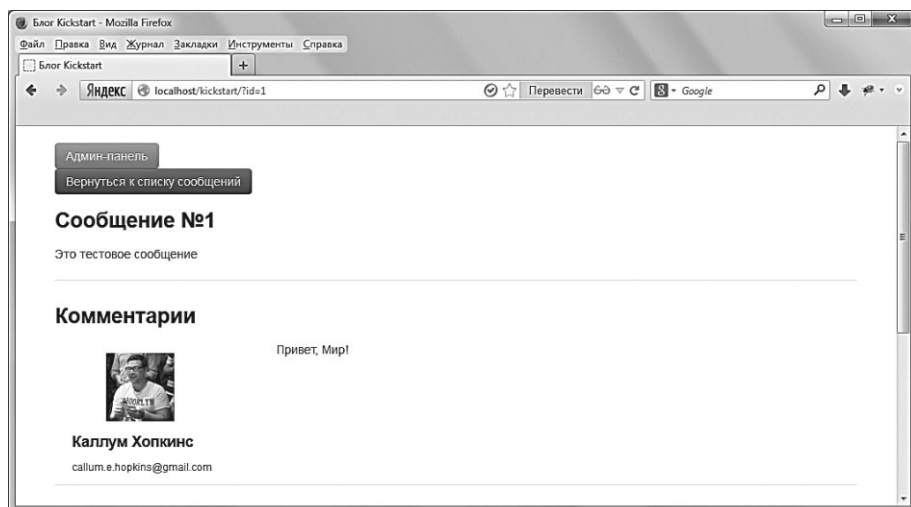


Рис. 5.5. Наш блог, теперь с комментариями!

Резюме

Сеансы и файлы cookie крайне важны при добавлении учетных записей пользователей в РНР-приложение. Они являются мощными инструментами для персонализации данных, отображаемых в браузере, поскольку позволяют быстро и эффективно идентифицировать пользователей на основании информации, хранящейся на их компьютерах. В случае с учетными записями пользователей в приложениях файлы cookie и сеансы жизненно необходимы для создания статуса авторизации, который мы реализовали для панели администратора нашего приложения. Тем не менее важно помнить о потенциальных угрозах безопасности, связанных с сеансами и файлами cookie, а также о создании кода, который позволяет снизить этот риск.

Для получения дополнительной информации об использовании сеансов и файлов cookie, обратитесь к следующим публикациям:

- www.sitepoint.com/php-sessions/
- www.sitepoint.com/baking-cookies-in-php/
- www.php.net/manual/ru/book.session.php
- php.net/manual/ru/features.cookies.php

Теперь, когда у нас есть функционирующая блог-платформа, переходим к заключительной главе. В ней мы рассмотрим различные аспекты РНР, в которых можно повысить безопасность и добавить дополнительные элементы блог-платформы на случай, если необходимо будет расширить функциональность приложения.

Глава 6

ЯЗЫК PHP И БЕЗОПАСНОСТЬ

Тестирование и обеспечение безопасности являются двумя наиболее важными аспектами разработки приложения. Вы можете создать самое впечатляющее приложение на свете, однако, если вы не обеспечите его безошибочное функционирование и безопасность, вы не завершите начатую работу.

В этой главе мы рассмотрим некоторые основные способы повысить безопасность вашего PHP-приложения. В этой небольшой книге нельзя осветить все возможные приемы, поэтому вам стоит обратиться к дополнительным источникам, чтобы подробнее изучить методы, описанные в этой главе.

Файл `php.ini` и безопасность

Мы можем начать заботиться о безопасности еще до начала создания PHP-кода. В начале книги мы кратко коснулись файла конфигурации *php.ini*. Он позволяет настроить множество поведений PHP, связанных, например, с загрузкой файлов, кэшированием сценариев и обеспечением безопасности, что представляет для нас в данный момент особый интерес.



ВЫ ДЕЙСТВУЕТЕ НА СВОЙ СТРАХ И РИСК!

Не изменяйте настройки, если вы на 100% не уверены в том, как эти изменения повлияют на работу PHP. Изменение настроек без соответствующих знаний может привести к неожиданным результатам.



НЕРЕДАКТИРУЕМЫЙ ФАЙЛ *PHP.INI*

Некоторые хостинг-провайдеры могут запретить доступ к файлу *php.ini* или его редактирование. Если вы не можете получить доступ к каталогу, в котором находится этот файл, вам следует обратиться к вашему провайдеру и попросить разрешения внести изменения.

Чтобы определить местонахождение файла *php.ini*, используйте функцию `phpinfo()`. Если вам требуется освежить свои знания, обратитесь к главе 1. Найдя нужный файл, откройте его в текстовом редакторе, и вы увидите огромное количество доступных параметров конфигурации. Некоторые из них, рассмотренные ниже, помогают повысить уровень безопасности вашего приложения.

Параметр `allow_url_include`

```
allow_url_include=Off
```

По умолчанию при стандартной установке PHP параметр `allow_url_include` отключен, однако некоторые хостинг-провайдеры могут активировать его, поэтому вам следует проверить свой файл *php.ini*.

При отключенном параметре `allow_url_include` PHP может загружать и выполнять только файлы, находящиеся в файловой системе веб-сервера. Тем не менее, если вы активируете параметр `allow_url_include`, вы сможете включить дополнительные файлы с помощью URL-адреса. Это может показаться интересным и полезным, однако при этом вашему приложению грозит опасность внедрения вредоносного кода. По этой причине рекомендуется оставлять этот параметр отключенным.



ВНЕДРЕНИЕ КОДА

Внедрение кода — это термин, используемый для описания ситуации, когда нежелательный код из внешнего источника без разрешения помещается в PHP-сценарий. Для получения дополнительной информации о внедрении кода обратитесь к следующим ресурсам:

- www.theserverpages.com/articles/webmasters/php/security/Code_Injection_Vulnerabilities_Explained.html
- www.owasp.org/index.php/Code_Injection

Связанный с параметром `allow_url_include` параметр `allow_url_fopen` также должен быть отключен.

Параметр `open_basedir`

```
open_basedir=/path/to/web/directory
```

Как и параметр `allow_url_include`, параметр `open_basedir` также позволяет ограничить круг файлов, которые сценарии могут загружать на сервер. С помощью параметра `open_basedir` вы можете указать, из какого каталога разрешается загружать файлы, и гарантировать то, что никакие файлы, находящиеся вне этого каталога, не будут загружены в сценарии.

Одним из методов, которыми могут воспользоваться потенциальные злоумышленники, является получение на веб-сервере, использующем операционную систему Linux, доступа к файлу `/etc/passwd`, в котором перечислены учетные записи пользователей, зарегистрированных на сервере. Если ваше PHP-приложение запускается из каталога `/var/www`, вы можете указать для параметра `open_basedir` значение `"/var/www/"`. После этого файлы, находящиеся вне каталога `www`, нельзя будет включить в ваши сценарии, что остановит злоумышленников. Активируйте этот параметр при запуске приложения в работу, чтобы обеспечить дополнительный уровень безопасности для вашего сервера.

Управление ошибками

На стадии разработки проекта полезно обеспечить получение от PHP обратной связи при возникновении ошибки или проблемы в коде. Тем не менее после развертывания приложения вам нужно, чтобы этих ошибок было как можно меньше. Скорее разработчик пожелает скрыть от пользователей сообщения об ошибках PHP, поскольку они могут вызвать недоверие.

Важно отметить, что сообщения об ошибках также могут предоставить злоумышленникам важные сведения о вашей системе, которые они могут использовать в своих интересах. Сообщения об ошибках PHP, используемые по умолчанию, могут содержать такую информацию, как путь установки приложения, подробные сведения о подключении к базе данных, названия столбцов базы данных и другие детальные сведения о сценарии, например имена классов, идентификаторы объектов и имена переменных.

Чтобы скрыть эту информацию от пользователей, отключите параметр `display_errors` в файле *php.ini*. Это даст PHP инструкцию не показывать сообщения об ошибках во время выполнения программы в случае их возникновения. Это не означает, что PHP проигнорирует ошибку, он просто не отобразит сообщение о ней в браузере.

Вы по-прежнему можете дать PHP инструкцию сохранять информацию об ошибках в специальном журнале, к которому вы, как разработчик, можете обратиться в частном порядке. Для этого в качестве значения параметра `error_log` укажите путь к нужному файлу с данными об ошибках.

Вы также можете активировать параметр `log_errors`. Это гарантирует то, что PHP будет регистрировать и сохранять данные об ошибках и избегать отображения сообщений об ошибках на экране.

После настройки этих параметров PHP создаст журналы с данными об ошибках, к которым вы можете обратиться в случае необходимости.

Повышение безопасности сеанса

Сеансы, как уже говорилось в предыдущей главе, являются средством временного хранения данных о статусе авторизации пользователя. Мы используем сеансы в нашем блог-приложении, и, как вы уже, наверное, заметили, они обеспечивают полезные функции. Сеансы часто содержат лишь небольшое количество информации, однако любые данные, связанные с получением доступа к приложению, могут использовать злоумышленники. По умолчанию сеансы хранят информацию о пользователях вашего приложения и используют файлы cookie для того, чтобы помочь таким языкам программирования, как PHP, правильно сопоставлять каждого пользователя с данными сеанса. К сожалению, сеансы поддерживают состояние авторизации, что играет на руку злоумышленникам и делает сеансы одной из наиболее привлекательных для них целей.

Одним из способов, которым могут воспользоваться хакеры, является получение доступа к данным сеанса. К счастью, файл *php.ini* снова приходит на помощь, позволяя изменить место хранения данных сеансов с помощью параметра `session.save_path`:

```
session.save_path = /var/lib/php
```

В приведенном выше примере демонстрируется параметр по умолчанию. Если вы измените местоположение, это помешает хакерам, поскольку им придется выяснить, куда вы переместили данные сеанса.

При изменении этого пути важно убедиться, что ваш веб-сервер способен считывать и записывать данные в новом месте, в противном случае PHP не сможет правильно использовать сеансы и будет выдавать сообщения об ошибках!

Хакеры также часто пытаются использовать ваше приложение и сервер с помощью техники Cross Site Scripting¹ (XSS, «межсайтовый скриптинг»), что позволяет им внедрять в приложение вредоносный код JavaScript. Сеансы часто подвергаются XSS-атакам, которые могут нанести различный вред — от слежения за клавиатурой, при котором сценарий сохраняет сведения обо всех нажатых клавишах, в том числе при вводе паролей и учетных данных, до несанкционированного рекламного обращения к пользователям с целью получения финансовой выгоды. Для предотвращения XSS-атак вы можете использовать следующий параметр:

```
session.cookie_httponly = 1
```

Этот параметр с установленным значением 1, которое используется по умолчанию в большинстве файлов *php.ini*, не позволяет коду JavaScript получить доступ к файлам cookie, созданным PHP или для использования вместе с сеансами. Настоятельно рекомендуется активировать этот параметр, если вы не хотите, чтобы код JavaScript использовал данные из файлов cookie, созданных в PHP, поскольку это действительно помогает контролировать поток данных от клиентского компьютера к вашему приложению и серверу, на котором оно работает.

Для получения дополнительной информации о параметрах файла *php.ini* и способах повышения безопасности PHP, обратитесь к следующим источникам:

- www.sitepoint.com/a-tour-of-php-ini/
- www.php.net/manual/ru/ini.php
- www.madirish.net/?article=229

¹ ru.wikipedia.org/wiki/Межсайтовый_скриптинг

Проверка предоставленных данных

Одна из самых серьезных угроз для PHP-приложений заключается не в недостаточной защите от действий злоумышленников, а в плохом качестве данных, предоставленных пользователем. Хотя часто трудно определить, предоставляются ли неправильные данные специально или случайно, они могут повредить вашему приложению.

Важно убедиться, что ваши пользователи знают, какой тип данных им необходимо предоставить. Используйте правильные элементы ввода для типа данных, которые вы запрашиваете, и убедитесь в том, что подписи явно указывают на то, что именно ожидается от пользователя.

Кроме того, даже при взаимодействии со специальными ограничительными полями ввода HTML-формы, пользователи могут предоставить данные в неправильном формате. При передаче данных в сценарий PHP для обработки важно использовать некий сценарий проверки.

Существует несколько функций PHP, которые мы можем применять для проверки. Следующий код был разработан для проверки предоставленных пользователем данных на корректность. Это помогает удостовериться в том, что формат данных позволяет приложению их использовать:

```
<?php
$input = array();
//здесь будут храниться все отфильтрованные данные
// поле ввода 'number'
that accepts only numeric input
if (isset($_POST['number']) && is_numeric
↳($_POST['input'])) {
$input['number'] = $_POST['number'];
}
//поле ввода 'date', принимающее даты в формате
↳дд/мм/гггг
if (isset($_POST['date'])) {
list ($dd, $mm, $yyyy) = explode('/', $input);
if (checkdate ($mm,$dd,$yyyy)) {
```

```
$input['date'] = $_POST['date'];  
}  
}  
// поле ввода 'content', принимающее произвольный  
↳ текст  
if (isset($_POST['content'])) {  
// отфильтровывает вредоносный HTML-код, который может  
↳ содержаться в строке  
$input['content'] = strip_tags($_POST['content']);  
}  
// значения в массиве $input можно без опасений  
↳ использовать в приложении
```

В приведенном выше примере мы проверяем каждое из входящих значений формы и помещаем их в новый массив, если они удовлетворяют базовым критериям проверки. Неудовлетворяющие им данные не добавляются в массив `$input`.

Функция `is_numeric()` проверяет содержимое строки и возвращает истинное значение только в случае, если она содержит символы, из которых состоит число. Числовые последовательности могут включать знак (+ или –), любое количество цифр, а также дробную или экспоненциальную часть.

Функция `checkdate()` проверяет корректность формата введенной даты и возвращает истинное значение, если аргументы составляют действительную дату.

Наконец, функция `strip_tags()` возвращает строку, из которой были удалены теги HTML и PHP, а оставлены только основные HTML-теги форматирования (например, тег разрыва строки `
`) и обычный текст.

После того как проверка проведена, значения в массиве `$input` можно спокойно использовать в остальной части нашего приложения, зная, что они представлены в правильном формате.

Тем не менее данные формы — это не единственный тип данных, корректность которых следует проверять. В созданном нами блог-прило-

жении мы использовали переменные URL для загрузки новых разделов приложения и выполнения конкретных сценариев. Переменные URL часто становятся мишенью хакеров, поскольку ими легко манипулировать, и они могут серьезно повредить приложениям, в которых не предусмотрены необходимые меры безопасности.

Мы уже реализовали несколько методов проверки, чтобы обеспечить безопасность нашего приложения, уделив особое внимание разделам, где переменные URL используются для указания того, какой метод или функцию необходимо выполнить:

```
...  
if (!empty($_GET['id']) && is_numeric($_GET['id'])) {  
...  
}
```

Этот фрагмент кода из файла *admin/posts.php* демонстрирует использование функции `is_numeric()` для гарантии того, что приложение продолжит обрабатывать метод внутри условного оператора `if`, только в том случае, если требуемые данные в URL-адресе будут представлять собой число.

Любые другие значения, передаваемые через URL-адрес, будут проигнорированы.

Резюме

В этой главе мы кратко рассмотрели тему безопасности PHP и некоторые простые способы защиты от наиболее распространенных угроз.

Безопасность PHP — это очень объемная тема, и в этой главе мы затронули лишь верхушку айсберга. Каждый раз, когда вы пишете новую функцию, постарайтесь представить себе, как кто-нибудь сможет:

- получить несанкционированный доступ к любым защищенным разделам;
- нарушить, исказить или изменить данные, обрабатываемые приложением;
- прервать, изменить или негативно повлиять на процессы и/или функциональность приложения.

В процессе разработки всегда стоит помнить о том, что пользователь может предоставить неправильные или некорректно отформатированные данные. Если при разработке приложения вы не забываете об этом, то сможете предотвратить возникновение многих проблем, связанных с безопасностью вашего приложения.

Более подробную информацию об использовании методов, функциональности и сценариев для повышения безопасности ваших PHP-приложений вы можете найти в следующих источниках:

- www.sitepoint.com/top-10-php-security-vulnerabilities/
- www.sitepoint.com/php-security-blunders/
- www.php-security.net

ЗАКЛЮЧЕНИЕ

В этой небольшой книге мы познакомились с процессом разработки на языке PHP и создали простую блог-платформу. Весь материал, который мы рассмотрели, был тщательно отобран, чтобы вдохновить вас на дальнейшее развитие приобретенных навыков.

Эта книга должна рассматриваться в качестве отправной точки в вашей деятельности PHP-разработчика. Для развития своих навыков вы можете использовать ресурс [**www.sitepoint.com/php/**](http://www.sitepoint.com/php/), содержащий множество статей, посвященных разработке на PHP.

«Функция хорошего программного обеспечения заключается в том, чтобы заставить сложное казаться простым». — Грэйди Буч

ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ

А

Apache Friends, проект, 14
Apache, сервер, 13, 14
 доступ к, 17, 18
 установка в ОС Ubuntu, 19

С

Cookie, файлы, 109
 время жизни, 115
 дата действия, 111

Д

DSN (Имя источника данных), 49

Е

Elseif, оператор, 38

Г

GET, метод, 82, 87

Н

HTML, код
 в комбинации с кодом PHP, 25

М

Markdown, язык разметки, 128
 конвертер, 128
Mod_rewrite, модуль, 20
MVC (Модель-вид-контроллер), архитектура, 63

Р

Pagedown, редактор, 128
PDO, расширение, 44
PhpMyAdmin, веб-интерфейс, 14, 44
PHP, код
 в комбинации с кодом HTML, 25
PHP, язык программирования
 автор, 12
 время создания, 12
 группа The PHP Group, 12
 комментарии, 31
 массивы, 29
 первоначальное название, 12
 файл конфигурации, 146

Т

Twitter Bootstrap Framework, 95

У

Ubuntu, ОС, 19

А

Алгоритм шифрования md5, 144
Архитектура MVC, 63

Б

База данных, 13
 MySQL, 14, 43
Базовый класс, 58
Булева переменная, 29

В

Веб-сервер
 локальный, 13

Д

Декремент, 41
Динамические данные, 13

З

Закрывающий тег, 25

И

Инкремент, 41

К

Кавычки, 24
Класс, 58
 базовый, 58
Ключевое слово
 break, 39
 class, 56
Ключ массива, 30
Конкатенация, 35
Конструкция PHP
 echo, 40, 41

Л

Локальный веб-сервер, 13
 LAMP, 14
 MAMP, 14
 WAMP, 14
 XAMMP, 14

М

Массив
 ключ, 30
 сокращенная запись, 30
Межсайтовый скриптинг (XSS), 85, 150
Метод, 57
 array_slice(), 97
 closeCursor(), 53
 empty(), 100
 execute(), 52
 explode(), 97
 fetch(), 52
 getPosts(), 93
 header(), 99

 htmlspecialchars(), 85
 implode(), 97
 index(), 98
 listPosts(), 103
 loginFail(), 99
 loginSuccess(), 99
 phpinfo(), 21
 POST, 82
 query(), 50
 session_destroy(), 116, 117
 session_start(), 116
 session_unset(), 116, 117
 setcookie(), 114
 strip_tags(), 97
 time(), 115
 unset(), 117
 validateDetails(), 98, 100
 viewPost(), 94
Модель-вид-контроллер (MVC),
 архитектура, 63

Н

Наследование, 58

О

Обработка исключений, 93
Объект, 54, 58
Объектно-ориентированное
 программирование (ООП), 54
 класс, 54
Оператор
 конкатенации, 35
Операционная система, 14
 Debian, 19
 Ubuntu, 19
Открывающий тег, 24

П

Переменная URL, 82
Переменные PHP
 инициализация, 41
 массив, 29
 строка, 28
 суперглобальные, 86
 типы, 28
 целые числа, 28
 числа с плавающей точкой, 28
Пользовательский интерфейс, 32
Порт, 17
Пространство имен, 135
Процессор, 15

Р

Расширение класса, 58

С

Свойство, 57

Сеанс, 109

 безопасность, 113

Скриптовый язык, 12

Сообщество разработчиков, 13

Суперглобальные переменные, 86

 \$_COOKIE, 113

 \$_REQUEST, 86

 \$_SESSION, 113, 116

Счетчик посетителей, 13

Т

Точка с запятой, 24

У

Условный оператор, 36

Ф

Функция, 57

Х

Хеширование данных, 51

Ц

Цикл, 36

 декремент, 41

 инкремент, 41

Э

Элемент <form>

 атрибуты, 76

Элементы формы, 76

Производственно-практическое издание

КОМПЬЮТЕР НА 100%. БЫСТРЫЙ СТАРТ

РНР
Быстрый старт
(орыс тілінде)

Ответственный редактор *В. Обручев*

ООО «Издательство «Эксмо»
123308, Москва, ул. Зорге, д. 1. Тел. 8 (495) 411-68-86, 8 (495) 956-39-21.
Home page: www.eksmo.ru E-mail: info@eksmo.ru

Өндіруші: «ЭКМО» АҚБ Баспасы, 123308, Мәскеу, Ресей, Зорге көшесі, 1 үй.
Тел. 8 (495) 411-68-86, 8 (495) 956-39-21
Home page: www.eksmo.ru E-mail: info@eksmo.ru

Тауар белгісі: «Эксмо»
Қазақстан Республикасында дистрибьютор және өнім бойынша
арыз-талаптарды қабылдаушының
өкілі «РДЦ-Алматы» ЖШС, Алматы қ., Домбровский көш., 3-а, литер Б, офис 1.
Тел.: 8 (727) 2 51 59 89, 90, 91, 92, факс: 8 (727) 251 58 12 вн. 107; E-mail: RDC-Almaty@eksmo.kz
Өнімнің жарамдылық мерзімі шектелмеген.
Сертификация туралы ақпарат сайтта: www.eksmo.ru/certification

Сведения о подтверждении соответствия издания согласно законодательству РФ
о техническом регулировании можно получить по адресу: <http://eksmo.ru/certification/>

Өндірген мемлекет: Ресей
Сертификация қарастырылмаған
Подписано в печать 27.08.2014. Формат 70х100¹/₁₆.
Печать офсетная. Усл. печ. л. 12,96.
Тираж экз. Заказ

ISBN 978-5-699-72685-1



Оптовая торговля книгами «Эксмо»:

ООО «ТД «Эксмо». 142700, Московская обл., Ленинский р-н, г. Видное,
Белокаменное ш., д. 1, многоканальный тел. 411-50-74.

E-mail: reception@eksmo-sale.ru

**По вопросам приобретения книг «Эксмо» зарубежными оптовыми
покупателями обращаться в отдел зарубежных продаж ТД «Эксмо»**

E-mail: international@eksmo-sale.ru

*International Sales: International wholesale customers should contact
Foreign Sales Department of Trading House «Eksmo» for their orders.*

international@eksmo-sale.ru

**По вопросам заказа книг корпоративным клиентам, в том числе в специальном
оформлении, обращаться по тел. +7 (495) 411-68-59, доб. 2261, 1257.**

E-mail: vipzakaz@eksmo.ru

**Оптовая торговля бумажно-беловыми и канцелярскими товарами для школы и офиса
«Канц-Эксмо»:** Компания «Канц-Эксмо»: 142702, Московская обл., Ленинский р-н, г. Видное-2,
Белокаменное ш., д. 1, а/я 5. Тел./факс +7 (495) 745-28-87 (многоканальный).

e-mail: kanc@eksmo-sale.ru, сайт: www.kanc-eksmo.ru

В Санкт-Петербурге: в магазине «Парк Культуры и Чтения БУКВОЕД», Невский пр-т, д. 46.

Тел.: +7(812)601-0-601, www.bookvoed.ru/

Полный ассортимент книг издательства «Эксмо» для оптовых покупателей:

В Санкт-Петербурге: ООО СЗКО, пр-т Обуховской Обороны, д. 84Е. Тел. (812) 365-46-03/04.

В Нижнем Новгороде: ООО ТД «Эксмо НН», 603094, г. Нижний Новгород, ул. Карпинского, д.
29, бизнес-парк «Грин Плаза». Тел. (831) 216-15-91 (92, 93, 94).

В Ростове-на-Дону: ООО «РДЦ-Ростов», пр. Стачки, 243А. Тел. (863) 220-19-34.

В Самаре: ООО «РДЦ-Самара», пр-т Кирова, д. 75/1, литера «Е». Тел. (846) 269-66-70.

В Екатеринбурге: ООО «РДЦ-Екатеринбург», ул. Прибалтийская, д. 24а.

Тел. +7 (343) 272-72-01/02/03/04/05/06/07/08.

В Новосибирске: ООО «РДЦ-Новосибирск», Комбинатский пер., д. 3.

Тел. +7 (383) 289-91-42.

E-mail: eksmo-nsk@yandex.ru

В Киеве: ООО «РДЦ Эксмо-Украина», Московский пр-т, д. 9. Тел./факс: (044) 495-79-80/81.

В Донецке: ул. Артема, д. 160. Тел. +38 (032) 381-81-05.

В Харькове: ул. Гвардейцев Железнодорожников, д. 8. Тел. +38 (057) 724-11-56.

Во Львове: ТП ООО «Эксмо-Запад», ул. Бузкова, д. 2. Тел./факс (032) 245-00-19.

В Симферополе: ООО «Эксмо-Крым», ул. Киевская, д. 153.

Тел./факс (0652) 22-90-03, 54-32-99.

В Казахстане: ТОО «РДЦ-Алматы», ул. Домбровского, д. 3а.

Тел./факс (727) 251-59-90/91. rdc-almaty@mail.ru

Интернет-магазин ООО «Издательство «Эксмо»

www.fiction.eksmo.ru

Розничная продажа книг с доставкой по всему миру.

Тел.: +7 (495) 745-89-14. E-mail: imarket@eksmo-sale.ru



Все, что нужно знать о PHP, в одной книге!

Язык PHP очень популярен, он обеспечивает функционирование 80% всех веб-сайтов, в том числе таких ресурсов, как Facebook, Wikipedia и WordPress. Этот язык прост в изучении и отлично подходит для начинающих, так что вы сможете довольно быстро начать его использовать!

Благодаря интенсивному курсу всего за пару дней вы сможете:

- Изучить основы PHP – синтаксис, операторы, циклы и функции
- Разобраться в теме объектно-ориентированного программирования
- Узнать, как PHP работает с формами и данными
- Повысить безопасность своих PHP-приложений

Всего за несколько дней вы подготовите солидную базу для самостоятельного создания PHP-кода!



Каллум Хопкинс – веб-разработчик по профессии и дизайнер по призванию. Обладая знаниями в области дизайна и разработки, он может влиять на обе стороны процесса создания сайта. Страсть к сложным функциям, красивому дизайну и функциональности заставляет его искать новые пути построения, проектирования и оптимизации веб-решений для клиентов по всему миру.

ISBN 978-5-699-72685-1

