

Создание web-приложений, исполняемых на стороне сервера при помощи языка программирования PHP и технологии AJAX

Урок №2

Сохранение состояния пользователя и файлы как хранилище данных

Содержание

Обработка форм.....	3
Элемент form	4
Передача данных на сервер методом POST	8
Форма и обработчик в одном файле	10
Сохранение состояния пользователя.....	16
Куки.....	18
Сессии	23
Разработка сайта	30
Разметка.....	30
Страница регистрации	40
Функция register()	44
Страница Upload	49
Страница gallery.....	56
Домашнее задание	61

Материалы к уроку прикреплены к данному PDF-файлу.

Для доступа к материалам необходимо открыть урок в программе
[Adobe Acrobat Reader](#).

Обработка форм

На прошлом уроке вы узнали, что пользователь веб-приложения может создать запрос к серверу тремя способами:

1. Ввести адрес требуемого ресурса в адресную строку браузера и нажать Enter.
2. Кликнуть по какой-либо ссылке на странице.
3. Нажать специальную кнопку в какой-либо форме.

Рассмотрим внимательно третий способ – использование формы. Вы уже знаете, что форма – это набор элементов управления (текстовых полей, списков и т.п.), в которые пользователь может заносить данные. Теперь вы должны дополнить свои знания тем, что форма позволяет создавать запрос к веб-серверу. Когда вы нажимаете в форме кнопку с типом `submit`, браузер создает Request к веб-серверу. Вы уже знаете, что кнопки могут быть разными, так вот, запомните: запрос создается только при нажатии на кнопку типа `submit`. Нажатия на другие кнопки не приводят к созданию запроса. И последнее: запросы, созданные с помощью формы, отличаются от других запросов тем, что позволяют передавать веб-серверу данные. Вы помните из первого урока, что данные можно передавать через адресную строку и без помощи формы. Однако форма позволяет делать это намного эффективнее.

После этого краткого вступления давайте научимся использовать формы и пополним свой багаж очень

мощным и гибким инструментом, необходимым для создания веб-приложений.

Элемент form

Продолжим работать в нашей папке Test. Создайте в ней два файла с именами form1.php и handler1.php. В первом файле мы создадим форму, а во втором – обработчик этой формы. Обработчик – это файл, который получает данные, присланные формой, и обрабатывает их.

Занесем в form1.php такую разметку, создающую условную форму:

```
<form action="handler1.php" method="GET">
<p>Enter your name: <input type="text"
name="username"></p>
<p>Enter your password: <input type="password"
name="userpass"></p>
<p><input type="submit" name="submit"
value="Login"></p>
</form>
```

Не обращайте внимания на то, что расширение файла php, а мы указали в нем только HTML разметку. Это обычная практика при работе в PHP. Первая строка выведет в браузер открывающий тег <form>. У этого тега надо указывать два обязательных атрибута: action и method.

Атрибут action указывает «обработчик формы». Вы же помните, что нажатие на кнопку submit приведет к созданию Request? В каждом Request указывается тот ресурс (файл на сервере), который пользователь просит

прислать ему. При создании запроса формой в этом запросе содержится просьба прислать файл, указанный в атрибуте `action`. Обработчик формы – это файл на сервере, которому будут отправлены данные формы. Он обработает эти данные, затем будет прислан сервером обратно клиенту в `Response` и принесет результаты обработки данных формы.

Атрибут `method` указывает, каким способом мы хотим пересылать данные формы на сервер. В нашем случае мы указали, что хотим использовать метод `GET` (тот самый, с которым познакомились в первом уроке). Посмотрим, как он работает в этой ситуации.

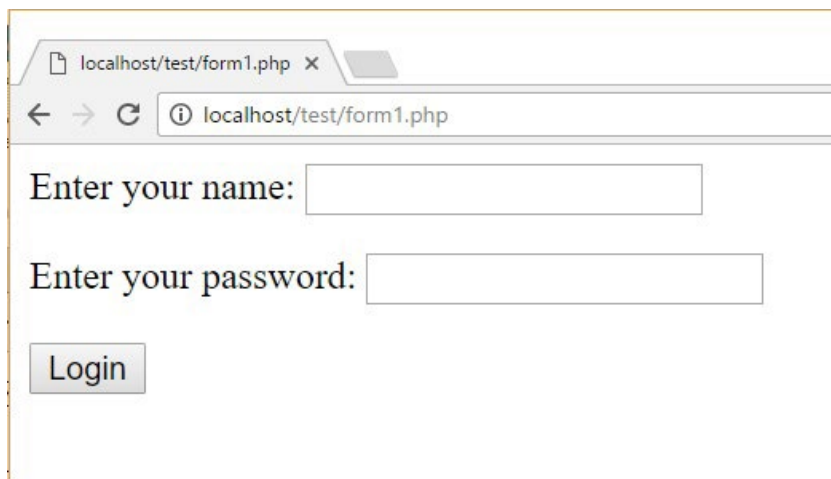
Мы создали очень простую форму, содержащую текстовое поле для ввода имени пользователя, поле для ввода пароля и кнопку `submit`. Обратите внимание, что каждый `input` содержит атрибут `name` с уникальными значениями.

Запомните: PHP получает доступ к элементам HTML-страницы по атрибутам `name`. Другими словами: если вы хотите в PHP коде работать с каким-либо элементом (тегом) на странице – обязательно добавьте этому элементу атрибут `name` с уникальным для страницы значением. В нашей форме поле для ввода имени имеет имя `username`, поле для ввода пароля – `userpass`, а кнопка получила имя `submit`.

Попросите веб-сервер прислать вам этот файл, для чего введите в адресную строку браузера такой адрес:

<http://localhost/test/form1.php>

Вы получите в браузере следующую форму:



The screenshot shows a web browser window with the address bar displaying 'localhost/test/form1.php'. The page content includes a form with the following elements:

- A text input field preceded by the label 'Enter your name:'.
- A text input field preceded by the label 'Enter your password:'.
- A button labeled 'Login'.

Рис. 1. Форма входа

Форма выглядит не очень привлекательно, но на данном этапе оформление будет отвлекать нас от предмета изучения. Вот когда мы начнем создавать сайты, то обязательно будем использовать для их оформления Bootstrap.

Давайте немного потестируем созданную форму. Оставьте все поля в форме пустыми и нажмите кнопку Login. Первое, что бросится в глаза – вы получили в браузере пустую страницу. Так и должно быть. Сервер прислал нам в ответ на нажатие кнопки submit файл handler1.php. А этот файл у нас пока пуст. Вот мы и видим пустую страницу. Но это еще не все. Посмотрите внимательно на адресную строку в браузере:

<http://localhost/test/handler1.php?username=&userpass=&submit=Login>

Узнаете? Это результат работы метода `get`. Метод `get` создал пары ключ=значение для каждого элемента формы. В качестве ключа используется значение атрибута `name` элемента, а в качестве значения – данные элемента. Поскольку вы в форму ничего не заносили, то видите пустые данные для `username` и `userpass`. Вернитесь в браузере к нашей форме и занесите в нее какие-нибудь значения. Например, «Bill» для имени и «123» для пароля. Затем снова нажмите `Login` и посмотрите в адресную строку. Теперь она выглядит так:

<http://localhost/test/handler1.php?username=Bill&userpass=123&submit=Login>

Метод `get` обработал ваши данные. Обратите внимание, что пароль передается в открытом виде.

Перейдем к обработчику для нашей формы. Метод `get` не только занес наши данные в адресную строку. Он также создал соответствующие элементы в глобальном массиве `$_GET[]`. Поэтому обработчик может получить присланные ему данные из этого массива. Откройте в блокноте файл `handler1.php` и занесите в него такой код:

```
<?php
if (isset($_GET["submit"]))
{
    echo 'You are welcome, ' . $_GET["username"] .
    '!' . 'Your password is ' . $_GET["userpass"] . '<br/>';
}
else
{
    echo 'No data were received!<br/>';
}
?>
```

Наш обработчик очень простой: он проверяет, была ли нажата в форме кнопка `submit` и приветствует пользователя, демонстрируя полученные данные. Подробное объяснение использования функции `isset()` вы найдете немного ниже в этом уроке. Вызовите теперь нашу форму снова, занесите в нее данные и нажмите `Login`. Вы получите в браузере такой результат:

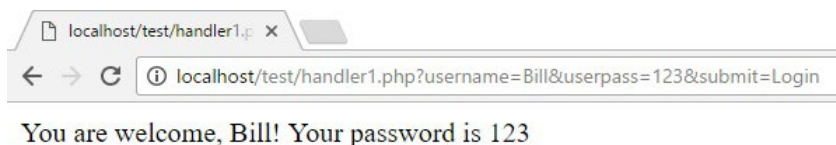


Рис. 2. Результат обработки формы

Подведем итог проделанной работе. Вы увидели полный цикл работы с формой как инструментом для создания запросов к серверу. У тега `form` надо указывать два атрибута: `action` и `method`. У каждого элемента управления в форме надо указывать атрибут `name` с уникальным для страницы значением. В форме обязательно должна присутствовать кнопка типа `submit`.

Кстати, можете провести такой эксперимент. В элементе `input`, создающем кнопку, поменяйте значение атрибута `type` с `submit` на `button`. Внешне ничего не изменится. Кнопка останется кнопкой. Но когда вы будете кликать по ней, никакой запрос к серверу создаваться не будет. Запрос в форме создается только кнопкой типа `submit`.

Передача данных на сервер методом POST

Посмотрим теперь на атрибут `method`. Мы указали для него значение `get`, и вы увидели, как это работает.

Замените в этом атрибуте значение `get` на `post` и снова вызовите нашу форму, занесите в нее данные и нажмите Login. Вы получите в браузере такой результат:

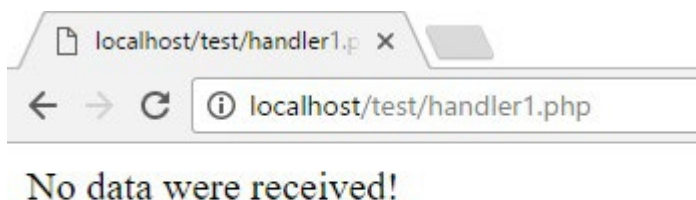


Рис.3. Результат обработки формы

Похоже, что-то пошло не так. Наш обработчик не получил данные, присланные ему формой? Нет, данные он получил. Просто неправильно ими воспользовался. Дело в том, что при использовании метода `post` PHP не заносит данные формы в массив `$_GET[]`. Он заносит данные в другой глобальный массив с именем `$_POST[]`. Поэтому наш обработчик должен вынимать данные уже из этого массива. Измените код в `handler1.php` на такой:

```
<?php
if(isset($_POST["submit"]))
{
    echo 'You are welcome, ' . $_POST["username"] .
    '! Your password is ' . $_POST["userpass"] . '<br/>';
}
else
{
    echo 'No data were received!<br/>';
}
?>
```

Выполните обработку формы снова. Теперь все в порядке – браузер отобразил присланные данные:

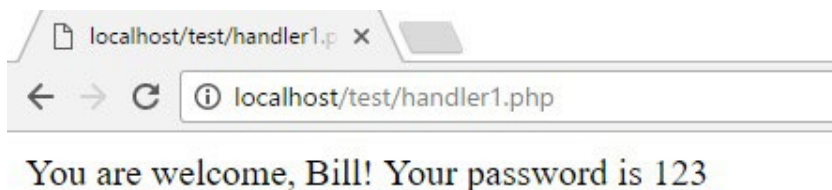


Рис. 4. Результат обработки формы методом POST

Посмотрите в адресную строку. Сейчас вы не видите там символа «?» и каких-либо данных. Это потому, что метод post пересылает данные не в адресной строке, а в теле Request. Размер данных пересылаемых по методу post намного больше, чем при методе get, и эти данные не выставляются на всеобщее обозрение. Метод post считается стандартным для пересылки данных форм.

Форма и обработчик в одном файле

В примере, с которым мы работали, за обработку формы отвечали два файла: form1.php и handler1.php. Если в создаваемом приложении требуется много форм, то количество файлов может быть очень большим, и разбираться, в каком файле какой обработчик, может быть не просто. Поэтому чаще применяют создание формы и ее обработчика в одном файле. Давайте посмотрим, как это можно сделать.

Создайте в папке Test файл form2.php. В этом файле будет форма. Второй файл с обработчиком для этой формы создавать не надо, т.к. обработчик тоже будет создан в этом же файле. Мы хотим сделать так, чтобы при обращении пользователя к серверу с просьбой при-

слать ему файл form2.php сервер присылал этот файл с формой. А после ввода в эту форму данных и после нажатия на кнопку submit сервер снова присылал бы этот же файл, но уже с результатами обработки формы.

Какое действие пользователя позволяет серверу понять, что присылать: форму или результат ее обработки? Конечно же, нажатие на кнопку submit. Если пользователь нажал на эту кнопку – значит, он уже занес в форму данные и хочет получить результат их обработки. Если же пользователь еще не нажимал кнопку submit, значит, он еще не видел форму, и именно форму надо ему прислать.

А что происходит внутри PHP, когда пользователь нажимает кнопку submit? Как понять, нажимал он эту кнопку или нет? Происходит вот что. При нажатии на кнопку submit в массиве \$_GET[] или \$_POST[] (в зависимости от используемого метода get или post), создается новый элемент с индексом, равным значению атрибута name кнопки.

Повторю то же самое другими словами. У вас есть такая кнопка:

```
echo '<p><input type="submit" name="btn1" value="OK"></p>';
```

Если вы ее нажмете, то при использовании в форме атрибута method="get", в массиве \$_GET[] будет создан элемент \$_GET["btn1"]. Если же в форме используется атрибут method="post", то будет создан элемент \$_POST["btn1"] в массиве \$_POST[]. В любом случае, индекс созданного элемента будет совпадать со значением атрибута name нажатой кнопки.

Значит, нам надо проверить, есть ли в соответствующем массиве элемент с индексом, равным имени кнопки. В PHP есть одна замечательная функция – `isset()`. Это булевская функция, она принимает имя какой-либо переменной, и, если такая переменная существует, `isset()` возвращает `true`, в противном случае `isset()` возвращает `false`. Обратите внимание, эта функция ничего не может сказать о значении переменной. Она проверяет только факт существования указанной переменной.

Полезно будет запомнить, что функцию `isset()` можно вызывать с несколькими параметрами:

```
iIf( isset($x, $y, $z))
{
    echo "All three variables exist! <br/>";
}
```

В таком случае функция возвращает `true`, если все указанные переменные существуют.

Мы применим функцию `isset()` в нашем файле `form2.php` и в дальнейшем будем использовать ее очень часто. Введите в `form2.php` такой код:

```
<?php
    if( !isset($_POST['submit']))
    {
        echo '<form action="form2.php" method="POST">';
        echo '<p>Enter your name: <input type="text"
                                name="username"></p>';
        echo '<p>Enter your password:
<input type="password" name="userpass"></p>';
        echo '<p><input type="submit" name="submit"
                                value="Login"></p>';
        echo '</form>';
    }
```

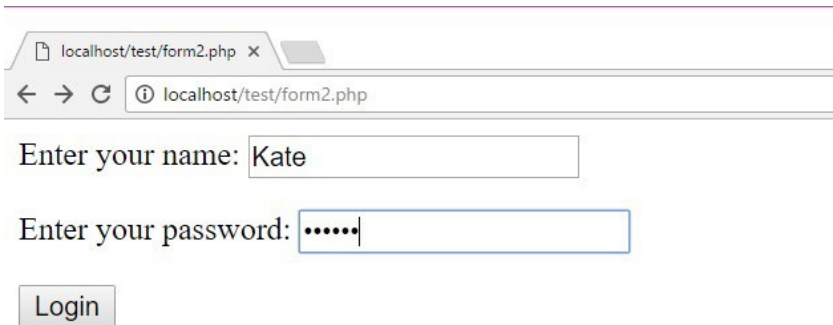
```

else
{
    echo 'You are wellcome, '.$_POST["username"].'!
Your password is '.
        $_POST["userpass"].'<br/>';
}
?>

```

Наша форма использует метод post, поэтому мы проверяем, существует ли переменная `$_POST["submit"]`. Если такой переменной нет, выполняется ветка `if` и пользователь получает форму. В противном случае, пользователь получает результат обработки нашей формы. Обратите внимание, что в качестве обработчика нашей формы указан этот же файл. Обратитесь в браузере к файлу `form2.php` и убедитесь, что все работает.

При первом обращении к файлу вы увидите форму и сможете занести в нее данные:



localhost/test/form2.php x

localhost/test/form2.php

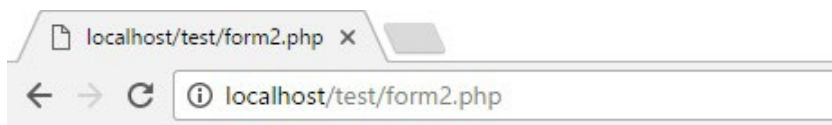
Enter your name:

Enter your password:

Login

Рис. 5. Вызов формы

После ввода данных и нажатия на кнопку Login вы получите результат обработки формы:



You are welcome, Kate! Your password is 123456

Рис. 6. Обработка формы

Рассмотрим еще несколько приемов, позволяющих упростить работу с формой. Согласитесь, что создавать HTML разметку в коде PHP с помощью функции `echo` сложнее, чем создавать такую же разметку просто на HTML странице. Особенно, если надо создать много разметки и если надо использовать кавычки и следить за тем, какие кавычки внешние, какие внутренние. Поэтому часто разрывают код PHP, чтобы вклинить в него чистую HTML разметку. Например, нашу форму можно было бы создать другим способом. В этом примере показано, как это можно сделать:

```
<?php
if( !isset($_POST['submit']))
{
?>

    <form action="form2.php" method="POST">
    <p>Enter your name: <input type="text"
                                name="username"></p>
    <p>Enter your password: <input type="password"
                                name="userpass"></p>
    <p><input type="submit" name="submit"
                                value="Login"></p>

    </form>
<?php
}
```

```
else
{
    echo 'You are welcome, '.$_POST["username"].'!  

    Your password is '.$_POST["userpass"].'<br/>';
}
?>
```

Мы закрыли PHP скрипт после открывающей скобки if и дальше создали форму на чистом HTML. А после создания формы снова открыли PHP и продолжили наш PHP скрипт. Результат работы этого файла будет таким же, как и до модификации.

Сохранение состояния пользователя

Мы уже говорили о том, что веб-сервер, обработав какой-либо Request пользователя, отправляет пользователю Response и тут же отключается от этого пользователя. Если этот же пользователь пришлет веб-серверу через полсекунды новый Request, веб сервер будет вести себя с этим пользователем так, как будто видит его в первый раз. Так ведут себя все веб-серверы. Это намеренно сделано для того, чтобы максимально разгрузить веб-сервер, позволяя ему обрабатывать больше клиентов. Вы должны понять, к чему приводит такое поведение сервера. Для этого мы проведем еще один эксперимент. Создайте все в той же папке Test два файла: session1.php и session2.php.

В файл session1.php занесите такой код:

```
<?php
    $num=100;
    echo 'From first file num='.$num.'<br/>';
?>
<a href='session2.php'>Forward</a>
```

А в файл session2.php – такой:

```
<?php
    echo 'From second file num='.$num.'<br/>';
?>
<a href='session1.php'>Back</a>
```


Вызовите первый файл, указав в браузере путь к нему: <http://localhost/test/session1.php>

Вы увидите результат обработки этого файла. Все ожидаемо: значение переменной \$num и ссылка на второй файл.



Рис. 7. Результат выполнения session1.php

Теперь кликните в браузере по ссылке Forward. Вы увидите такой вывод:



Рис. 8. Результат перехода по ссылке

Обсудим полученный результат. В файле session1.php мы создали переменную \$num и присвоили ей некоторое значение. Это значение мы увидели в браузере. Затем мы кликнули по ссылке и перешли в файл session2.php. И теперь почему-то мы не увидели преды-

дущего значения у этой переменной. Эта переменная пуста. Почему это произошло?

Помните, каким образом пользователь может создавать запросы к веб-серверу? Там был клик по ссылке. Клик по любой ссылке отправляет на сервер новый Request, и сервер присылает Response на этот Request. Результаты предыдущего Request удаляются. Наша переменная \$num со значением 100 осталась в старом запросе. В новом запросе, созданном кликом по ссылке, создается новая переменная \$num. Эта новая переменная пуста, так как мы в нее ничего не заносим в новом запросе на странице session2.php.

Клиент работает с сервером дискретно, от запроса к запросу. И каждый новый запрос начинается с чистой страницы. Все значения, прочитанные или вычисленные в предыдущем запросе – затираются. Это принцип работы веб-сервера по умолчанию. Но такое умолчание очень часто не устраивает пользователя. Для таких случаев существуют два механизма, позволяющие изменить поведение веб-сервера. Рассмотрим эти механизмы.

Куки

Для сохранения состояния пользователя можно использовать куки. Куки – это текстовые файлы на клиентском компьютере, поэтому значения, занесенные в них, могут оставаться активными независимо от того, включен компьютер или выключен, хоть 100 лет. В куки-файлах обычно хранятся индивидуальные пользовательские настройки, разные идентификаторы и т.п., в форме пар «ключ-значение». Если вы создали куки, то

они автоматически передаются на сервер с каждым вашим запросом в строках-заголовках. Поэтому их надо создавать до какого-либо вывода в браузер. Учтите, что куки имеют ограниченный размер. Суммарный размер всех куки для одного сайта – 4 килобайта. Обратите внимание на то, что данные в куках хранятся в виде обычных строк. Поэтому, хранить конфиденциальные данные в куки-файлах не стоит. Если же вы решили хранить там важные данные, то используйте шифрование и ограничение доступа к куки-файлам. В этом разделе вы увидите, как это можно делать.

Рассмотрим пример создания куки-файлов, их использования и удаления.

Для создания куки надо вызвать функцию `setcookie()`. Эта функция содержит шесть параметров, но только два из них обязательные.

Полный перечень параметров выглядит так:

`setcookie(name, value, expire, path, domain, security);`

где:

`name` – имя куки;

`value` – значение, хранимое в куки;

`expire` – время, после которого куки уничтожается;

`path` – указывает папки, для которых куки доступен;

`domain` – задает домены, для которых куки доступен;

`security` – указывает, надо ли шифровать значение куки.

После вызова функции `setcookie()` пользователю становится доступным для записи и чтения суперглобальный массив `$_COOKIE[]`, работа с которым идентична

использованию других суперглобальных массивов: мы можем добавлять в этот массив элементы с уникальными ключами и читать такие элементы.

Создайте в папке Test файл cookie.php и добавьте в него такой код:

```
<?php
if(!isset($_COOKIE['name']))
{
    setcookie("name", "Johnnie Walker",
              time()+60*60*24, "/", "", 0);
    setcookie("volume", "1", time()+60*60*24,
              "/", "", 0);
}
else
{
    $_COOKIE['volume']=$_COOKIE['volume']+1;
    setcookie("volume", $_COOKIE['volume']);
}
echo 'Name: '.$_COOKIE['name'].'<br/>';
echo 'Volume: '.$_COOKIE['volume'].'<br/>';
?>
<a href="cookie.php">Increase</a>
```

Этот скрипт проверяет, существует ли куки-файл с именем name. Если такого файла нет, создается пара куки с именами name и volume. Если же куки name существует, то значение другой куки, с именем volume, увеличивается на 1. Затем значения обоих куки-файлов выводятся в браузер. Ссылка на эту же страницу существует для перегрузки страницы (создания нового запроса). При первом обращении к этому файлу вы увидите:

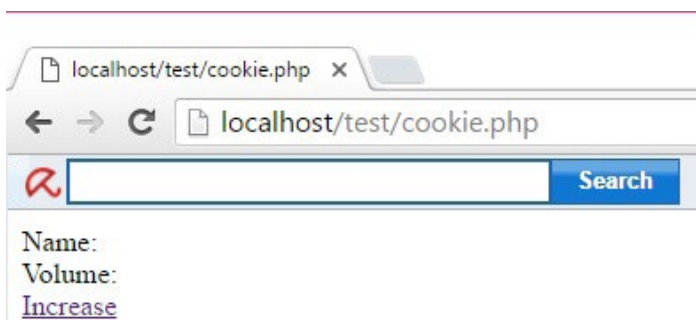


Рис. 9. Результат первого вызова cookie.php

Как видите, значения куки еще пусты. Вы не можете в одном и том же запросе создать куки и увидеть занесенные в них значения. Так как куки будут созданы на клиентском компьютере только после получения Response на этот запрос, в котором куки создавались. Обновите теперь эту страницу, либо кликнув по ссылке, либо нажав F5 в браузере. Теперь вы увидите оба значения, и значение volume будет увеличиваться при каждом обновлении:

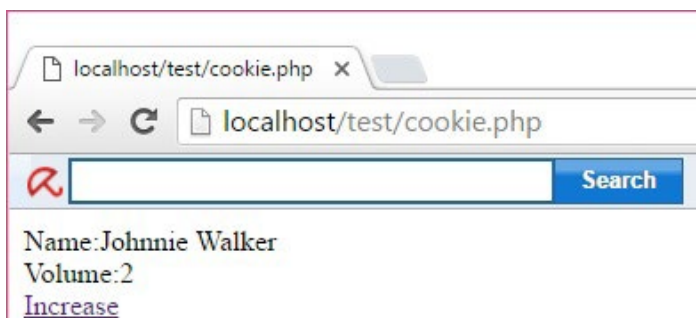


Рис. 10. Обновление cookie.php

Обратите внимание, что для обновления значения куки надо снова вызывать функцию `setcookie()`, указы-

вая имя куки, значение которой надо изменить, и само новое значение. Если же вы хотите удалить куки, то надо снова вызвать `setcookie()`, указав для параметра `expire` момент времени в прошлом – например, 10 секунд назад: `time() - 10`. Третий параметр функции `setcookie()` задает не продолжительность существования куки с момента создания, а момент времени, когда куки будет удален браузером.

Как можно обезопасить данные в куки от нежелательного доступа? Если вы хотите ограничить доступ к куки для всех страниц, кроме страниц, расположенных, например, в папке `pages`, используйте параметр `path` функции `setcookie()`:

```
setcookie("volume", "1", time()+60*60*24, "/pages/", "", 0);
```

Более того, в этом параметре вы можете указать конкретный файл, которому будут доступны значения из куки:

```
setcookie("volume", "1", time()+60*60*24, "/pages/register.php", "", 0);
```

Для дополнительной безопасности вы можете указать конкретный домен, которому будут доступны значения из куки:

```
setcookie("volume", "1", time()+60*24, "/pages/register.php", "mysite.com", 0);
```

А если вы укажете последний параметр равным 1, то данные в куках будут автоматически шифроваться, в то время, как по умолчанию они хранятся в виде открытого текста:

```
setcookie("volume", "1", time()+60*24, "/pages/register.php", "mysite.com", 1);
```

Кроме этого старайтесь избегать использования в имени куки символа точка «.» и пробела. Дело в том, что эти два символа в имени куки-файла автоматически заменяются символом подчеркивания «_», что может вызвать проблемы, если забыть об этом.

Если вам не понравился способ указания времени существования куки с помощью `time()+n`, можете использовать функцию `mktime()`, позволяющую задать момент времени с точностью до секунды. Почитайте описание этой функции здесь:

<http://php.net/manual/en/function.mktime.php>

Ко всему сказанному о куки, остается добавить, что пользователь может отключить у себя в браузере использование куки. В таком случае вам надо будет использовать другие средства для сохранения состояния пользователя.

Сессии

Сессии, как и куки, создают возможность сохранять какие-либо пользовательские данные на протяжении многих запросов. Как надо пользоваться сессиями? Предположим, что на какой-то странице вы получили определенное значение (например, имя вошедшего пользователя) и хотите в дальнейшем использовать это значение на других страницах. Вы понимаете, что переход на другую страницу заставит вас создать новый запрос, в котором это значение будет затерто.

Как сделать это значение доступным в других запросах? На странице, где вы получаете важные данные, надо вызвать функцию `session_start()`.

Запомните, что эту функцию обязательно надо вызывать до любого вывода в браузер. Другими словами, на странице, где вы вызываете `session_start()` до этого вызова не должно быть ни одного вывода конструкцией `echo` или `print_r()`, или другой подобной функцией. При вызове `session_start()` сервер создает уникальный символьный идентификатор, например, такой: `3dg9tlbe5047rnimqlae4kk190`. В специальной папке, заданной в настройечном файле `php.ini` в атрибуте `session.save_path`, сервер создает файл, имя которого является конкатенацией префикса `sess_` и созданного идентификатора. В нашем примере имя файла будет таким: `sess_3dg9tlbe5047rnimqlae4kk190`. Теперь пользователь может особым образом сохранять свои важные данные, и они будут храниться на сервере в этом файле.

Кроме этого, создается куки с именем `PHPSESSID`. В этот куки записывается созданный идентификатор, который таким образом передается в Request браузеру клиента. Затем клиент будет автоматически передавать этот идентификатор серверу с каждым своим последующим запросом. Сервер по этому идентификатору будет отличать запросы такого конкретного пользователя, и будет разрешать этому пользователю использовать значения, занесенные в файл `sess_*`, до тех пор, пока сессия будет активна. Т.е. идентификатор сессии, который позволяет серверу «узнавать» клиента, передается через куки.

А как будет передаваться идентификатор сессии, если пользователь отключил куки в своем браузере? В этом случае, снова же автоматически, без участия пользователя, этот идентификатор будет передаваться в каждом GET и POST запросе.

За это отвечают такие параметры в `php.ini`, как `session.use_cookies` и `session.use_trans_sid`. Если параметр `session.use_cookies` равен 1, то PHP создает куки с именем `PHPSESSID` и передает идентификатор сессии браузеру в этом куки. Если же этот параметр равен 0 – то такой куки не создается; Если параметр `session.use_trans_sid` равен 1, то происходит вот что.

После того, как PHP обрабатывает всю страницу и выполнит на ней все скрипты, он допишет к каждой ссылке на странице идентификатор созданной сессии, а также вставит этот идентификатор в каждую форму на этой странице. Это приведет к тому, что ссылка:

```
<a href="/index.php">Index</a>
```

превратится в

```
<a href="/index.php?PHPSESSID=3dg9t1be5047rnimqlae4kk190">
Index</a>
```

В каждую же форму будет вставлен такой элемент:

```
<input type="hidden" name="PHPSESSID"
      value="3dg9t1be5047rnimqlae4kk190"/>
```

Теперь, когда пользователь получит такую страницу, и кликнет в ней по любой ссылке, или нажмет на кнопку `submit` в форме, он отправит на сервер идентификатор своей сессии и сервер его «узнает» по этому идентификатору.

Каким образом можно сохранять данные в сессии?

После вызова функции `session_start()` пользователь получает доступ к еще одному глобальному ассоциатив-

ному массиву с именем `$_SESSION[]`. Данные, добавленные в этот массив, запоминаются в файле на сервере и являются доступными на других страницах из других запросов. При этом запомните, что если вам нужен доступ к массиву `$_SESSION[]` на какой-либо странице, вы должны на этой странице также вызвать `session_start()`. Эта функция при первом вызове создает сессию, а при последующих вызовах (на других страницах) предоставляет доступ к уже созданной сессии.

Функция `session_id()` возвращает идентификатор сессии.

Давайте изменим код в наших файлах `session1.php` и `session2.php`. Приведем `session1.php` к такому виду:

```
<?php
session_start();
echo 'id='.session_id().'\n';

$_SESSION['num']=100;
echo 'From first file num='.$_SESSION['num'].'\n';
?>
<a href='session2.php'>Forward</a>
```

Что здесь происходит? Открываем сессию, выводим в браузер идентификатор созданной сессии. Теперь мы имеем доступ к массиву `$_SESSION[]`. Создаем в этом массиве элемент с индексом `'num'` (название индекса совершенно произвольное) и заносим в этот элемент значение, которое хотим использовать на других страницах.

А файл `session2.php` приводим к такому виду:

```
<?php
session_start();
echo 'id='.session_id().'\n';

echo 'From second file num='.$_SESSION['num'].'\n';
?>
<a href='session1.php'>Back</a
```

В этом файле мы снова вызываем `session_start()`, этот вызов выполняется после вызова `session_start()` в первом файле. Поэтому сейчас мы не создаем новую сессию, а подключаемся к сессии, созданной при обработке `session1.php`. Выводим в браузер идентификатор сессии, чтобы продемонстрировать, что мы подключились к той же сессии. Теперь мы снова имеем доступ к массиву `$_SESSION[]`. Выводим оттуда значение по индексу `'num'`.

Попросите веб-сервер прислать вам файл `session1.php`. Вы получите в браузере такой вывод:

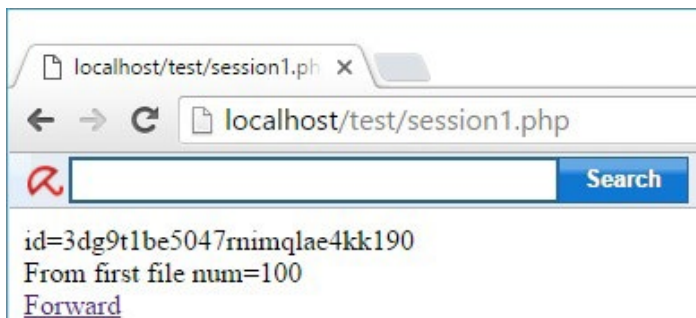


Рис. 11. Результат обработки `session1.php`

Теперь перейдите по ссылке `Forward` на страницу `session2.php`:

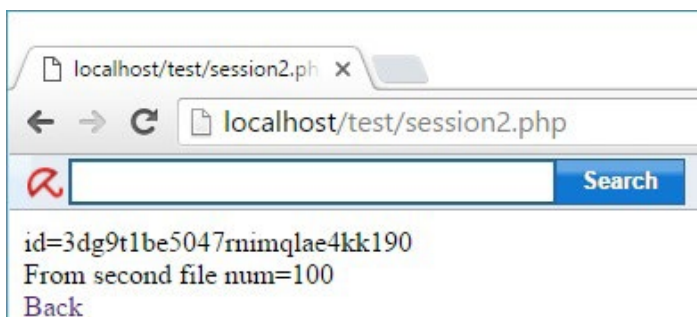


Рис. 12. Результат обработки session2.php

Убедитесь в том, что идентификатор сессии совпадает в обоих случаях. Выходит, мы работаем в пределах одной сессии. А это значит, что значения, добавленные в сессию на любой из страниц, будут доступны, пока сессия активна. Вы понимаете, что в одной сессии можно сохранять много значений, конечно же, с разными индексами:

```
$_SESSION['name']=$username;
$_SESSION['pass']=$userpassword;
$_SESSION['rate']=$score;
```

Если вам надо закрыть сессию и затереть все хранящиеся в ней значения, вызовите функцию `session_destroy()`. Если же вам надо удалить какое-то одно значение из сессии, используйте функцию `unset()`:

```
unset($_SESSION['num']);
```

В дальнейшем мы поговорим о сессиях более подробно. Сейчас только добавим, что сессия остается активной, пока пользователь работает в браузере. Закрытие браузера удаляет сессию.

Кроме того, сервер может удалить сессию пользователя, который не проявляет активности в браузере на протяжении некоторого времени, заданного в файле настройки `php.ini`. В этом файле есть два атрибута: `session.gc_maxlifetime` и `session.cookie_lifetime`, в которых в секундах указывается время существования сессии при отсутствии активности пользователя. Но вы должны учитывать, что если в параметре `session.gc_maxlifetime` вы укажете значение 600, это вовсе не гарантирует вам, что через 600 секунд сборщик мусора удалит вашу сессию. Это просто гарантирует вам, что через 600 секунд появится отличная от нуля вероятность того, что сборщик мусора сессию удалит. Но вероятность эта будет намного меньше 100%. Если же вам понадобится задать точное время существования сессии, это придется делать самостоятельно в коде.

Разработка сайта

Давайте продолжим дальнейшее изучение PHP не путем создания маленьких скриптов, без всякой пользовательской логики, а другим способом. PHP – это инструмент для создания сайтов. Поэтому продолжим обучения на примере создания разных сайтов. В ходе наших уроков мы создадим четыре разных сайта, каждый из которых будет демонстрировать применение тех или других возможностей PHP.

Сейчас мы приступим к созданию сайта, который продемонстрирует такие приемы, как:

- работа с формами;
- авторизация и аутентификация;
- upload;
- работа с файлами, строками и папками;
- создание на странице динамических элементов управления.

В отличие от наших следующих сайтов, особой прикладной логики у этого сайта не будет. Просто он продемонстрирует основные приемы языка PHP.

Разметка

Создайте рядом с папкой Test новую папку с именем Site1. В этой папке создайте такие вложенные папки: pages, images, css и js. Рядом с этими вложенными папками создайте файл index.php – это будет главная страница нашего сайта.

css	30.10.2016 14:59
images	30.10.2016 14:59
js	30.10.2016 15:00
pages	30.10.2016 14:59
index.php	30.10.2016 15:00

Рис. 13. Результат обработки session1.php

Будет полезно использовать в нашей работе Bootstrap. Bootstrap – это фреймворк, использующий HTML, CSS и Javascript и предназначенный для создания структуры и оформления HTML страниц. Сегодня этот фреймворк фактически является стандартом для оформления. Знакомство с ним позволяет быстро создавать современные, привлекательные сайты, даже без участия дизайнера. Хотя, участие дизайнера в оформлении сайта всегда желательно и полезно.

Чтобы применять на странице Bootstrap, надо создать ссылки на него. Использовать Bootstrap можно как из локальной загрузки, так и из CDN. Я предлагаю нам загрузить Bootstrap и работать с ним локально. Загрузите этот продукт со страницы:

<http://getbootstrap.com/getting-started/>

Разархивируйте полученный архив. В нем вы увидите три папки: js, css и fonts. Содержимое папок js и css занесите в одноименные папки нашего сайта, а папку fonts перенесите в нашу папку Site1 целиком.

Рассмотрим основные принципы применения Bootstrap. С одной стороны, можно рассматривать этот фреймворк, как набор CSS стилей. Применяя эти стили,

вы будете получать на своих страницах красивые элементы. Но только оформлением элементов применение Bootstrap не ограничивается.

Сегодня очень важным является умение создавать так называемый *responsive* или адаптивный дизайн страниц. Такой дизайн позволяет странице правильно отображать содержимое на устройствах с разными размерами экранов, автоматически определяя размер экрана и изменяя свой контент под конкретный экран. Так вот, Bootstrap позволяет создавать также и адаптивный дизайн страниц. Раньше иногда приходилось создавать несколько версий одной и той же страницы для разных экранов. Затем с помощью Javascript определялся размер экрана и пользователь переадресовывался на ту версию страницы, которая наилучше подходила для его экрана. Адаптивный дизайн освобождает разработчика от необходимости писать отдельную страницу для каждого типа экранов. Одна и та же адаптивная страница сама перераспределит свой контент и для мобильного устройства и для планшета и для обычного компьютера. Обратите внимание, страница не будет выглядеть одинаково на всех трех устройствах, он будет выглядеть адекватно в максимально подходящем для конкретного устройства виде.

Чтобы понять принципы применения Bootstrap надо запомнить, что он рассматривает страницу, как таблицу, состоящую из 12 колонок. Ширина каждого элемента на странице задается в количестве таких колонок (или же в долях $1/12$ от ширины всей страницы). Другими словами, если ширина какого-либо элемента равна 6 колонкам, это значит, что этот элемент занима-

ет половину ширины страницы. Если ширина элемента равна 3 колонкам – такой элемент занимает четверть ширины страницы. Обратите внимание, что при таком подходе ширина не указывается в пикселах. Кроме 12 колонок, страница может разбиваться на любое количество строк. В пределах каждой строки можно располагать любые элементы, снова же указывая их ширину «в колонках».

К этой информации надо добавить понимание того, как Bootstrap использует свои классы. Здесь уже надо рассматривать конкретные названия стилевых классов.

Строка создается с помощью класса `.row` и обязательно должна располагаться в контейнере с классом `.container`. Элементами, создающими строки и контейнеры чаще всего являются `div`. Вот пример контейнера с двумя строками.

```
<div class="container" >
  <div class="row">

    </div>
    <div class="row">

      </div>
    </div>
  </div>
```

Давайте в первой строке создадим три элемента с ширинами 25%, 50% и 25% ширины страницы. А во второй строке создадим два элемента шириной по 50% каждый.

Для этого вам надо познакомиться с некоторыми префиксами названий полезных классов для колонок:

- .col-xs- (extra small) для очень маленьких экранов с шириной меньше 768px;
- .col-sm- (small) для маленьких экранов с шириной больше 768px и меньше 992px;
- .col-md- (medium) для средних экранов с шириной больше 992px и меньше 1200px;
- .col-lg- (large) для больших экранов с шириной больше 1200px.

Директивы CSS приведут к тому, что активными будут только стили с одним префиксом, в зависимости от конкретной ширины экрана. Если пользователь будет работать на планшете, то для него активными будут классы .col-sm-*, а остальные классы .col-*- будут пустыми. Если пользователь будет работать на большом экране, то для него активными будут классы .col-lg-*, а остальные классы .col-*- будут пустыми. Это позволяет создавать такую разметку:

```
<div class="col-xs-12 col-sm-6 col-md-4 col-lg-2">  
</div>
```

Что это значит? Это значит, что:

- на мобильном устройстве этот div будет занимать всю ширину экрана, так как будет применен класс col-xs-12, присваивающий этому div ширину в 12 колонок;
- на планшете этот div будет занимать половину ширины экрана, так как будет применен класс col-sm-12, присваивающий этому div ширину в 6 колонок из 12;

- на настольном устройстве этот `div` будет занимать треть ширины экрана, так как будет применен класс `col-md-4`, присваивающий этому `div` ширину в 4 колонки из 4;
- на устройстве с большим экраном этот `div` будет занимать одну шестую ширины экрана, так как будет применен класс `col-lg-2`, присваивающий этому `div` ширину в 2 колонки из 12.

Теперь вернемся к нашему примеру и создадим три элемента в первой строке и два во второй:

```
<div class="container" >
  <div class="row">
    <!-- 25% page's width element -->
    <div class="col-sm-3 col-md-3 col-lg-3">
    </div>
    <!-- 50% page's width element -->
    <div class="col-sm-6 col-md-6 col-lg-6">
    </div>
    <!-- 25% page's width element -->
    <div class="col-sm-3 col-md-3 col-lg-3">
    </div>
  </div>
  <div class="row">
    <!-- 50% page's width element -->
    <div class="col-sm-6 col-md-6 col-lg-6">
    </div>
    <!-- 50% page's width element -->
    <div class="col-sm-6 col-md-6 col-lg-6">
    </div>
  </div>
</div>
```

Эта разметка не предусматривает варианта для sx экрана и не предусматривает изменения ширины элементов, для разных экранов. В этом случае для всех устройств ширины трех элементов в первой строке и двух во второй будут оставаться в такой же пропорции относительно ширины страницы:



Рис. 14. Пример разметки двух строк

Вооружившись этими начальными знаниями, перейдем к созданию нашего сайта. Вы же обязательно должны приучаться работать с Bootstrap. Для этого можно использовать много полезных ресурсов в сети.

Например, этот: <http://getbootstrap.com/2.3.2/>

Откройте в блокноте файл index.php, создайте в нем структуру html-страницы. Добавьте туда link и script для ссылки на Bootstrap. У вас должна получиться такая разметка:

```

1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>Site 1</title>
6   <!-- Bootstrap -->
7   <link href="css/bootstrap.min.css" rel="stylesheet">
8 </head>
9 <body>
10
11
12
13
14
15 <!-- jQuery (necessary for Bootstrap's JavaScript plugins) -->
16 <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js"></script>
17 <!-- Include all compiled plugins (below), or include individual files as needed -->
18 <script src="js/bootstrap.min.js"></script>
19 </body>
20 </html>

```

Рис. 15. Заготовка главной страницы для нового сайта

Разметка страниц нашего сайта будет простой: вверху – заголовок, под ним – горизонтальное меню, под меню – контент. Для создания такой структуры, вставьте в `<body>` следующую разметку:

```

10 <body>
11 <div class="container" >
12   <div class="row">
13     <header class="col-sm-12 col-md-12 col-lg-12">
14
15     </header>
16   </div>
17
18   <div class="row">
19     <nav class="col-sm-12 col-md-12 col-lg-12">
20
21     </nav>
22   </div>
23
24   <div class="row">
25     <section class="col-sm-12 col-md-12 col-lg-12">
26
27     </section>
28   </div>
29 </div>
30 <!-- jQuery (necessary for Bootstrap's JavaScript plugins) -->
31 <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js"></script>
32 <!-- Include all compiled plugins (below), or include individual files as needed -->
33 <script src="js/bootstrap.min.js"></script>
34 </body>
35 </html>

```

Рис. 16. Разметка главной страницы нового сайта

Создадим навигацию для нашего сайта. Для этого в папке `pages` создадим файл `menu.php` с таким содержанием:

```
<ul class="nav nav-pills">
  <li role="presentation" class="active">
    <a href="index.php?page=1">Home</a>
  </li>
  <li role="presentation">
    <a href="index.php?page=2">Upload</a>
  </li>
  <li role="presentation">
    <a href="index.php?page=3">Gallery</a>
  </li>
  <li role="presentation">
    <a href="index.php?page=4">Registration</a>
  </li>
</ul>
```

Этот файл создает четыре пункта меню с названиями Home, Upload, Gallery и Registration.

Для каждого из пунктов меню создадим в папке `pages` отдельный файл, который будет подгружаться на главную страницу при выборе соответствующего пункта меню. Поэтому создадим в папке `pages`, рядом с файлом `menu.php`, еще четыре таких файла: `home.php`, `upload.php`, `gallery.php` и `registration.php`. В каждый из этих файлов пока что занесем только название соответствующего пункта меню.

Обратите внимание, что переход по каждому пункту меню осуществляется ссылкой, которая ведет на главную страницу нашего сайта и при этом, с помощью метода `get`, передает на страницу переменную с именем `page`. Значение этой переменной будет использоваться

для загрузки на главную страницу файла из папки pages соответствующего выбранному пункту меню.

Теперь с помощью `include_once()` включим в элемент `nav` файл с нашим меню. А в элемент `section` вставим код, определяющий, какую страницу надо загрузить при активации какого-либо пункта меню.

```

18 <div class="row">
19 <nav class="col-sm-12 col-md-12 col-lg-12">
20 <?php include_once('pages/menu.php'); ?>
21 </nav>
22 </div>
23
24 <div class="row">
25 <section class="col-sm-12 col-md-12 col-lg-12">
26 <?php
27     if(isset($_GET['page']))
28     {
29         $page=$_GET['page'];
30         if($page == 1)include_once('pages/home.php');
31         if($page == 2)include_once('pages/upload.php');
32         if($page == 3)include_once('pages/gallery.php');
33         if($page == 4)include_once('pages/registration.php');
34     }
35 ?>
36 </section>
37 </div>

```

Рис. 17. Создание меню

Запустите в браузере наш сайт, введя такой адрес:

<http://localhost/site1/index.php>

Затем покликайте по разным пунктам меню и убедитесь, что в области контента появляется имя активированного пункта меню. Это говорит о том, что в элемент `section` с помощью функции `include_once()` загружается

файл из папки pages, соответствующий выбранному пункту меню.

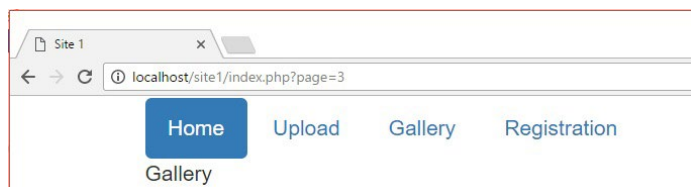


Рис. 18. Проверка работы меню

Теперь нам надо создать каждую из страниц: home.php, upload.php, gallery.php и registration.php.

Страница регистрации

Начнем со страницы регистрации registration.php. Будем делить посетителей нашего сайта на зарегистрированных пользователей и гостей. У гостей не будет доступа к меню Upload, а у зарегистрированных пользователей – будет.

На этой странице создадим форму регистрации. Для регистрации будем требовать логин, пароль и email. Зарегистрированных пользователей будем хранить в файле.

Занесите в файл registration.php такой код:

```
<h3>Registration Form</h3>
<?php
if(!isset($_POST['regbtn']))
{
?>
<form action="index.php?page=4" method="post">
<div class="form-group">
    <label for="login">Login:</label>
```



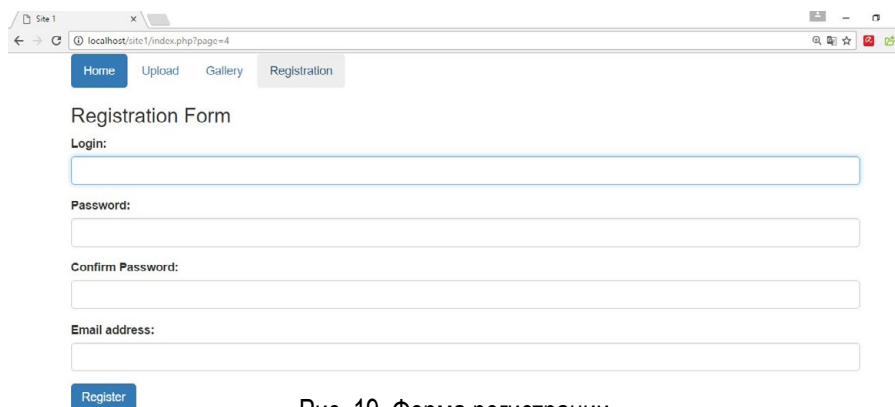
```

        <input type="text" class="form-control"
name="login">
</div>
<div class="form-group">
    <label for="pass1">Password:</label>
    <input type="password" class="form-control"
name="pass1">
</div>
<div class="form-group">
    <label for="pass2">Confirm Password:</label>
    <input type="password" class="form-control"
name="pass2">
</div>
<div class="form-group">
    <label for="email">Email address:</label>
    <input type="email" class="form-control"
name="email">
</div>
<button type="submit" class="btn btn-primary"
name="regbtn">Register</button>
</form>
<?php
}
else
{
}
?>

```

Ничего нового для вас в этом коде нет. Вы должны узнать структуру файла, содержащего и форму и ее обработчик. Об этом говорит проверка нажатия кнопки submit нашей формы. Если эта кнопка не нажималась, мы выводим форму. В нашем случае форма создана на чистом HTML. В противном случае, если кнопка submit уже нажималась, в else будет вставлен обработчик.

Форма содержит понятные поля ввода и оформлена с помощью Bootstrap. Данные, введенные в эту форму, будут передаваться на сервер методом post. Каждый элемент формы, конечно же, имеет атрибут name с уникальным значением. Откройте в браузере главную страницу нашего сайта, а затем кликните по меню Registraton. Вы должны увидеть:



Registration Form

Login:

Password:

Confirm Password:

Email address:

Register

Рис. 19. Форма регистрации

Конечно же, оформление этой формы каждый из вас может сделать по своему вкусу.

И еще одно очень важное замечание. Запомните, всякий раз, когда вы предоставляете пользователю возможность заносить что-то в форму, вы подвергаете свое приложение опасности. Пользователи специально или нечаянно могут занести в форму совсем не то, чего вы от них ожидаете. Поэтому валидация данных формы в приложениях очень важна. Более того, валидация должна выполняться два раза: на клиентской стороне, перед отправкой данных на сервер, и на серверной стороне, сразу после получения и перед началом использования полученных данных.

На клиентской стороне данные можно проверять самыми разными способами. Можно использовать чистый Javascript или jQuery, или AngularJS. Но мы в нашем примере не будем это делать. Мы изучаем РНР, а не Javascript. Поэтому мы рассмотрим валидацию только на серверной стороне, средствами РНР.

Мы предусмотрели в форме повторный ввод пароля, но сейчас мы не будем выполнять проверку на совпадение введенных паролей. Для этого есть причина.

Приступим к созданию обработчика формы регистрации. Задача обработчика заключается в проверке правильности занесенных данных и в добавлении в специальный текстовый файл новых зарегистрированных пользователей. При этом будем выполнять проверку на соблюдение уникальности login. У нас на сайте не может быть двух пользователей с одинаковыми login. Кроме этого, будем требовать, чтобы длина login и пароля была в пределах от 3 до 30 символов.

Для выполнения регистрации, для проверки входа зарегистрированного пользователя и для выполнения других действий на сайте, полезно будет создавать собственные функции. И все такие функции удобно держать в одном файле. Поэтому мы создадим в папке pages файл functions.php и в нем будем объявлять свои функции. Давайте подумаем, как нам подключаться к этому файлу с функциями. Конечно, можно использовать include_once() в каждом файле, где нужна будет какая-либо функция. Например, как сейчас, в файле registration.php. А можно поступить иначе.

Вы уже увидели, что все наши файлы `home.php`, `upload.php`, `gallery.php` и `registration.php` подключаются функцией `include_once()` в файл `index.php`. Поэтому, если мы подключим в `index.php` и файл с функциями `functions.php`, то все функции из этого файла будут доступными в других файлах, и нам не надо будет заботиться о том, куда подключать файл с функциями, а куда нет. Так и сделаем. Добавим ссылку на файл с функциями в `index.php`, например, здесь:

```
<nav class="col-sm-12 col-md-12 col-lg-12">
<?php
    include_once('pages/menu.php');
    include_once('pages/functions.php');
?>
</nav>
```

Теперь мы сможем вызывать функции, определенные в файле `functions.php`, на любой странице нашего сайта.

Функция `register()`

Открываем в блокноте `functions.php` и начинаем создавать функцию для регистрации новых пользователей. Эта функция будет принимать введенные в форму `login`, пароль и `email`. Затем будет проверять, чтобы все три значения были заполнены и чтобы длина их находилась в заданном диапазоне. Еще наша функция будет выполнять экранирование потенциально опасных символов, чтобы исключить `SQL injection`. После валидации функция проверит, уникален ли переданный `login`, и если все в порядке, добавит в файл описание нового пользователя.

В папке pages будет создан файл users.txt, в котором будут храниться зарегистрированные пользователи. Каждый пользователь в файле будет записан в отдельной строке в таком формате – имя:пароль:адрес. Пароль будет храниться в хешированном виде.

Наша функция может выглядеть таким образом:

```
$users = 'pages/users.txt';
function register($name, $pass, $email)
{
    //data validation block
    $name=trim(htmlspecialchars($name));
    $pass=trim(htmlspecialchars($pass));
    $email=trim(htmlspecialchars($email));
    if($name =='' || $pass =='' || $email =='')
    {
        echo "<h3/><span style='color:red;*>
            Fill All Required Fields!</
            span><h3/>";
        return false;
    }
    if(strlen($name) < 3 || strlen($name) > 30 ||
        strlen($pass) < 3 || strlen($pass) > 30)
    {
        echo "<h3/><span style='color:red;*>
            Values Length Must Be Between 3 And
            30!</span><h3/>";
        return false;
    }
    //login uniqueness check block
    global $users;
    $file=fopen($users,'a+');
    while($line=fgets($file, 128))
    {
        $readname=substr($line,0,strpos($line,':'));
        if($readname == $name)
        {
```

```

                echo "<h3/><span
style='color:red;*>
Such Login Name Is Already Used!</span><h3/>";
                return false;
            }
        }
        //new user adding block
        $line=$name.':'.md5($pass).':'. $email."<h3/>";
        fputs($file,$line);
        fclose($file);
        return true;
    }

```

Обсудим этот код. Во внешней переменной \$users указан путь к файлу с пользователями. Путь указан с позиции файла index.php, в котором будет находиться объявленная функция. В самой функции ссылка на эту переменную указана со спецификатором global. Параметры при вызове функции будут передаваться прямо из формы, поэтому валидацию выполняем здесь. Функция trim() обрезает пробелы в начале и в конце строки. Функция htmlspecialchars() выполняет экранирование активных символов, чтобы исключить передачу какого-либо скрипта. Если валидация не проходит, функция выводит сообщение об ошибке и возвращает false. В PHP сложилась хорошая традиция: если какая-либо функция не может успешно завершить работу, она возвращает false. Наша функция ведет себя так же.

После успешно пройденной валидации функция выполняет проверку уникальности введенного в форму login. Для этого мы открываем файл с пользователями (если его еще нет – то создаем его) и читаем построчно функцией fgets(\$file, 128). Все функции, работающие с

файлом, вы должны помнить со времен изучения C++ – они здесь идентичны. Мы читаем по 128 символов, так как с нашими ограничениями все строки будут короче этого значения. Чтобы вырезать из прочитанной строки формата «имя:пароль:адрес» имя пользователя, мы применяем функции `substr()` и `strpos()`. Подробное описание этих и других функций можете посмотреть на странице: <http://php.net/manual/en/function.substr.php>

Занеся прочитанное из файла имя в переменную `$readname`, мы сравниваем его с именем, занесенным в форму (переменная `$name`). Если есть совпадение – функция сообщает об этом и возвращает `false`. Если после прочтения всего файла совпадений нет, переходим к записи нового пользователя. Указатель в файле как раз находится в конце, поскольку мы только что завершили чтение всего файла. Готовим строку вида «имя:пароль:адрес» для записи. Пароль в эту строку вставляем функцией `md5()`, чтобы хранить в файле хешированную строку. Выполняем запись и закрываем файл. На приведенных ниже рисунках показана работа функции в разных ситуациях. Но сначала надо занести вызов нашей функции в раздел `else` файла `registration.php`:

```
else
{
    if(register($_POST['login'],$_POST['pass1'],$_POST['email']))
    {
        echo "<h3/><span style='color:green;'>
            New User Added!</span><h3/>";
    }
}
```

Вот как выглядят результаты использования нашей функции:

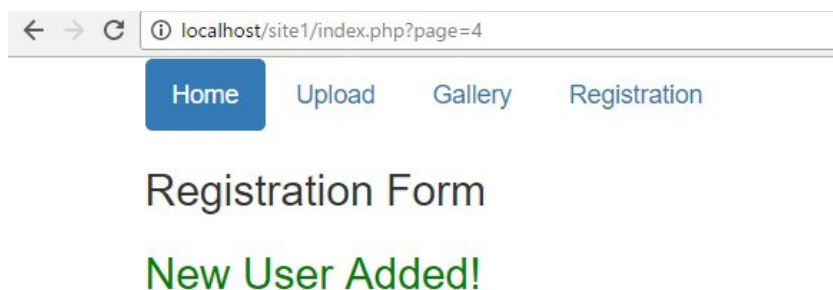


Рис. 20. Успешная регистрация нового пользователя

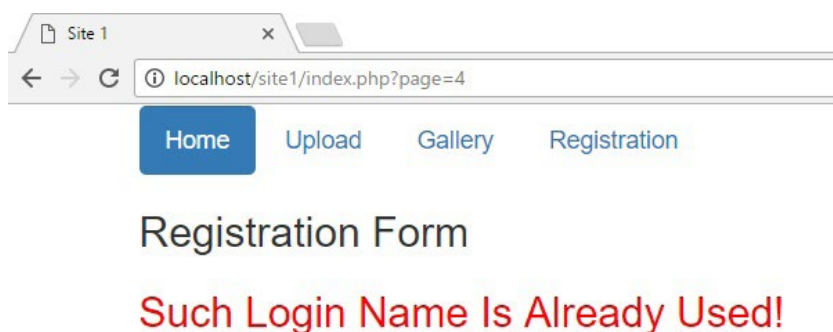


Рис. 21. Попытка регистрации с неуникальным именем

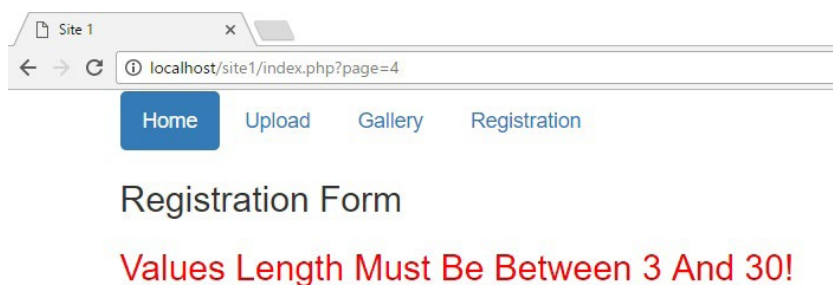


Рис. 22. Нарушение допустимой длины значения

А так выглядит содержимое файла `users.txt` после добавления двух пользователей:

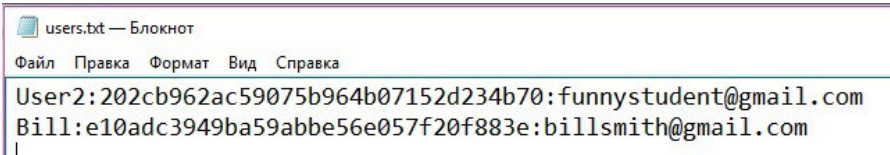


Рис. 23. Содержимое файла с пользователями

Теперь нам понадобится функция `login`, обрабатывающая вход пользователей на сайт (выполняющая аутентификацию). Эта функция должна будет принимать из формы имя и пароль входящего пользователя и проверять, есть такой пользователь в файле `users.txt`. Если такой пользователь есть, его имя будет заноситься в сессию, и функция будет возвращать `true`. Если аутентификация не пройдет – функция будет пересылать пользователя на страницу регистрации.

К разговору об этой функции и о форме входа вернемся немного позже. Сейчас же перейдем к обработке пункта меню `upload`.

Страница Upload

Термин `upload` означает пересылку файла с клиентского компьютера на сервер. Это действие противоположно действию `download`, когда пользователь загружает какой-либо файл с сервера на свой компьютер. Для выполнения этого действия в PHP используется еще один глобальный массив с именем `$_FILES`. Давайте разберем подробно, как выполняется `upload` и какую роль при этом играет массив `$_FILES`.

Все начинается с формы. Форма, в которой вы хотите выполнять upload, отличается двумя особенностями от других форм. Во-первых, в такой форме должен быть элемент `<input type='file' name='myfile'>`. Именно этот элемент создает стандартное окно для выбора файла. Во-вторых, у такой формы должен быть указан еще один атрибут: `enctype='multipart/form-data'`.

Этот атрибут всегда указывается с таким значением при выполнении upload. Запомните, если вы забудете указать в форме этот атрибут или допустите при его указании синтаксическую ошибку, PHP не скажет вам об этом. Но upload выполняться не будет.

Когда пользователь выберет пересылаемый файл и нажмет в такой форме кнопку submit, будут выполняться такие действия: выбранный файл будет отправлен на сервер, во временную папку, которая указана в файле `php.ini` в атрибуте `upload_tmp_dir`. Если в этом атрибуте ничего не указано, то используется временная папка `Windows\Temp`. Во временной папке файлу присваивается уникальное временное имя, чтобы избежать возможного конфликта имен. Пересланный файл будет находиться в этой папке до завершения текущего запроса, после чего сервер удалит его в целях безопасности. Поэтому необходимо сразу забирать выгруженный файл из временной папки и переносить его в требуемую папку. Мы будем складывать такие файлы в папку `images` нашего сайта.

Для обработки выгруженного файла мы будем пользоваться массивом `$_FILES`. Этот массив отличается от других глобальных массивов тем, что он двумерный. У

его элементов пара индексов. Первый индекс – это name элемента input с type='file'. Второй индекс имеет предопределенное имя. Для каждого выгружаемого файла в массиве `$_FILES` создаются пять элементов. Для нашего примера это будут элементы с такими индексами:

<code>\$_FILES['myfile']['name']</code>	– исходное имя файла, каким его выбрал пользователь;
<code>\$_FILES['myfile']['tmp_name']</code>	– временное имя, присвоенное файлу в папке temp;
<code>\$_FILES['myfile']['size']</code>	– размер файла в байтах;
<code>\$_FILES['myfile']['type']</code>	– тип файла в формате MIME;
<code>\$_FILES['myfile']['error']</code>	– код завершения.

Значения элемента `$_FILES['myfile']['error']` могут быть такими:

- 0 – файл успешно загружен на сервер, ошибок нет
- 1 – размер загруженного файла больше, чем предел, установленный параметром `upload_max_filesize` в `php.ini`
- 2 – размер загруженного файла больше, чем предел, установленный параметром `MAX_FILE_SIZE` в форме
- 3 – файл загружен не полностью
- 4 – файл не был загружен, т.к. указан неверный путь к файлу

Кроме этого, запомните, что можно создать многофайловую выгрузку, когда пользователь в одном запросе может передать на сервер сразу несколько файлов. Мы обязательно научимся это делать, но сначала рассмотрим выгрузку файлов по одному.

Несколько слов о размерах выгружаемого файла. Размер файла обязательно надо контролировать. Это можно делать еще до отправки, если добавить в форму, **непосредственно перед input с type='file'**, такой скрытый элемент:

```
<input type="hidden" name="MAX_FILE_SIZE" value="1024*1024*3" />.
```

Обязательно с таким значением name. Значение value этого элемента указывает максимальный размер файла в байтах, который примет эта форма. В моем примере – 3 МБ. Однако такое ограничение размера не является достаточным. Также размер надо проверять уже на стороне сервера. В файле php.ini есть атрибут upload_max_filesize, который содержит предельный размер, разрешенный для выгрузки на конкретном сервере. Например, upload_max_filesize = 2М устанавливает максимальный допустимый размер в 2 МБ.

Открываем страницу upload.php и в ней создаем форму для выгрузки файла и обработчик этой формы:

```
<h3>Upload Form</h3>
<?php
if(!isset($_POST['upload']))
{
?>
<form action="index.php?page=2" method="post"
      enctype="multipart/form-data">
<div class="form-group">
  <label for="myfile">Select file for upload:</label>
  <input type="hidden" name="MAX_FILE_SIZE"
        value="1024*1024*3" />
  <input type="file" class="form-control"
        name="myfile" accept="image/*">
</div>
```

```

<button type="submit" class="btn btn-primary"
name="uppbtn">Send File</button>
</form>
<?php
}
else
{
    if(isset($_POST['uppbtn']))
    {
        //errors handling
        if($_FILES['myfile']['error'] != 0)
        {
            echo "<h3/><span style='color:red;'>Upload
                                error code: " .
                                $_FILES['myfile']['error'] .
                                "</span><h3/>";
            exit();
        }
        //does the file exist on server in temp folder?
        //if(is_uploaded_file($_FILES['myfile']
        //['tmp_name']))
        {
            //remove the file from temp folder into images
            //folder with origin name
            //move_uploaded_file($_FILES['myfile']
            //['tmp_name'],
            //    "./images/".$_FILES['myfile']['name']);
        }
        echo "<h3/><span style='color:green;'>File
                                Uploaded Successfully!
</span><h3/>";
    }
}
?>

```

В форме выделены атрибут `enctype` и элемент `input` с `type='file'`. У этого элемента указан атрибут `accept='image/*'`. Этот атрибут позволит выбирать только графические файлы. Все остальное в форме вам уже знакомо.

Поговорим об обработчике. Обратите внимание, что первый индекс каждого элемента массива `$_FILES` – это значение `name` нашего `input` с `type='file'`. Сначала проверяем, возникли ли при выполнении `upload` ошибки. Если да – выводим сообщение об ошибке и прекращаем дальнейшую обработку `upload`. В противном случае, проверяем, появился ли наш файл во временной папке. Обратите внимание, что нам не надо знать, где эта папка расположена фактически. Функция `is_uploaded_file()` все знает сама, потому что мы передаем ей элемент `$_FILES['myfile']['tmp_name']`. Если файл на месте, переносим его в `images` с помощью функции `move_uploaded_file()`. Вот и все.

Теперь надо проверить, как этот код работает. Обратите внимание, что для успешного выполнения `upload` требуется полный доступ к временной папке. Это может быть актуальным, если политики на вашем компьютере ограничивают ваш доступ к папке `Windows` и ее содержимому, а временной папкой является как раз `Windows\Temp`.

Форма для выгрузки файла может выглядеть таким образом:

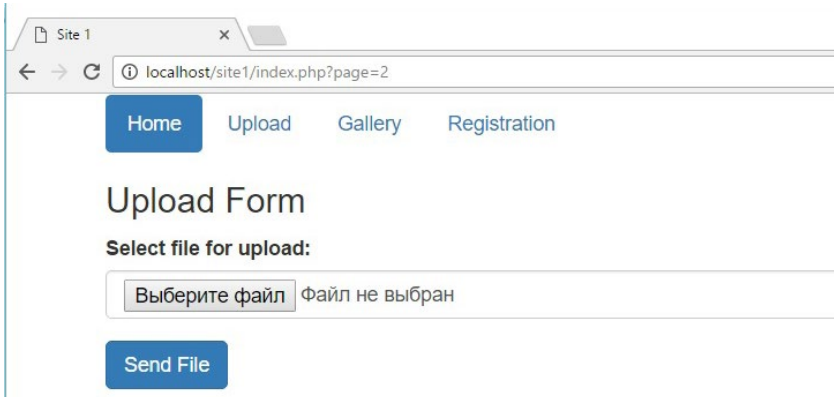


Рис. 24. Форма для выгрузки файла

После выбора и отправки файла на сервер страница выглядит так:

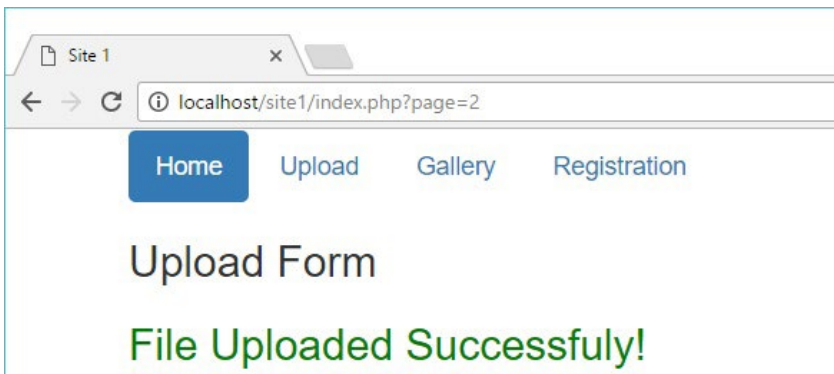


Рис. 25. Успешная выгрузка файла

Выполните выгрузку нескольких файлов. Перейдите в папку `images` нашего сайта и убедитесь, что выгруженные файлы находятся там.

Страница gallery

Перейдем к созданию самой красивой страницы нашего сайта – страницы галереи. На этой странице мы будем отображать выбранные графические файлы из папки `images`. То что мы создадим, галереей можно называть условно. Но наша цель – рассмотреть несколько полезных действий при создании этой страницы, а не создать суперкрасивую галерею.

Страница будет выглядеть так. Пользователю будет предложено выбрать из выпадающего списка одно из приведенных там графических расширений. Затем для всех файлов с выбранным расширением, расположенных в папке `images`, будет построен набор миниатюр. Каждая миниатюра будет кликабельной. При клике в соседней вкладке будет открываться кликнутая картинка в исходном размере.

Расширения файлов в указанном выпадающем списке будут создаваться в зависимости от актуального контента папки `images`. Понятно, что если в папке `images` будет сто файлов с расширением `jpg`, в выпадающем списке это расширение будет указано только один раз.

Работа над этой страницей продемонстрирует создание динамических элементов управления (список с расширениями и `img`), работу с папками и еще несколько полезных РНР функций.

Откройте страницу `gallery.php` и создайте в ней такой код:


```

<h2>Gallery</h2>
<form action='index.php?page=3' method='post'>
<p>Select graphics extension to display:</p>
<select name='ext'>
<?php
    $path = 'images/';
    if($dir = opendir($path))
    {
        $ar = array();
        while (($file = readdir($dir)) !== false)
        {
            $fullname = $path . $file;
            $pos = strrpos($fullname, '.');
            $ext = substr($fullname, $pos+1);
            $ext= strtolower($ext);
            if( !in_array($ext, $ar) )
            {
                $ar[] = $ext;
                echo "<option>" . $ext . "</option>";
            }
        }
        closedir($dir);
    }
?>
</select>
<input type="submit" name="submit" value="Show
                                   Pictures"
class="btn btn-primary"/>
</form>
<br/>
<?php
    if(isset($_POST['submit']))
    {
        $ext = $_POST['ext'];
        $ar = glob($path . "*. " . $ext);
        echo "<div class='panel panel-
                                   primary'>";
        echo '<div class="panel-heading">';
        echo '<h3 class="panel-title">Gallery
                                   content</h3></div>';
    }

```

```

        foreach ($ar as $a)
        {
            echo "<a href='" . $a . "'
                    target='_blank'>
                <img src='" . $a . "'
                    height='100px' border='0'
                    alt='picture'
                    class='img-polaroid' />
                </a>";
        }
        echo "</div>";
    }
?>

```

Сначала создаем стандартную форму, в которой размещаем элемент `select` с именем `"ext"`. В этом элементе пользователь будет выбирать желаемый тип файлов. Теперь надо создать набор вложенных элементов `option` для этого `select`. Каждый `option` будет отображать одно расширение. Количество и содержимое элементов `option` зависит от того, какие файлы в данный момент располагаются в папке `images`. Поэтому для создания этих `option` нам надо войти в папку `images`, пересмотреть там все файлы и создать массив из расширений этих файлов. Мы предполагаем, что в папке `images` находятся только графические файлы, поскольку при выборе файлов для `upload` мы использовали атрибут `ассерт`, позволяющий выбирать только графические файлы. Поэтому дополнительную фильтрацию сейчас выполнять не будем.

Для работы с папками используем функции `opendir()` и `readdir()`. Почитать о них подробнее можно здесь: <http://php.net/manual/en/function.opendir.php>

Первая функция пытается найти требуемую папку и возвращает указатель на нее. Вторая функция возвра-

щает информацию о файлах, расположенных в заданной папке. Эта функция вызывается в цикле и прекращает работу, когда вернет значение `===false`. Это признак того, что все файлы в папке уже прочитаны. Для каждого прочитанного в папке файла мы в цикле достаем его расширение с помощью функций `substr()` и `strrpos()`. Обратите внимание, сейчас мы используем не функцию `strpos()`, а `strrpos()`. В отличие от первой, функция `strrpos()` ищет заданную подстроку с конца строки. Чтобы не создать несколько `option` для одного и того же расширения, мы будем запоминать каждое обработанное расширение в массиве `$ar`. Чтобы исключить недоразумения с `jpg` и `JPG`, мы с помощью функции `strtolower()` переводим расширения в нижний регистр. После обработки всей папки вызываем функцию `closedir()`, чтобы освободить занятые ресурсы.

В обработчике получаем из формы значение, выбранное в `select`, и строим шаблон вида `"images/*.EXT"`, где `EXT` – выбранное расширение. Передаем этот шаблон функции `glob` (смотреть на странице <http://php.net/manual/en/function.glob.php>) и получаем в ответ массив, каждый элемент которого – это один из файлов с выбранным расширением по указанному пути. Для каждого такого файла создаем кликабельную миниатюру высотой 100px.

Проверяем, как это работает. Переходим на страницу `upload`, выбираем в списке какое-либо расширение и жмем кнопку `submit`. Кликните по каким-либо миниатюрам, чтобы проверить, как они открываются в новых вкладках. У меня это выглядит так:

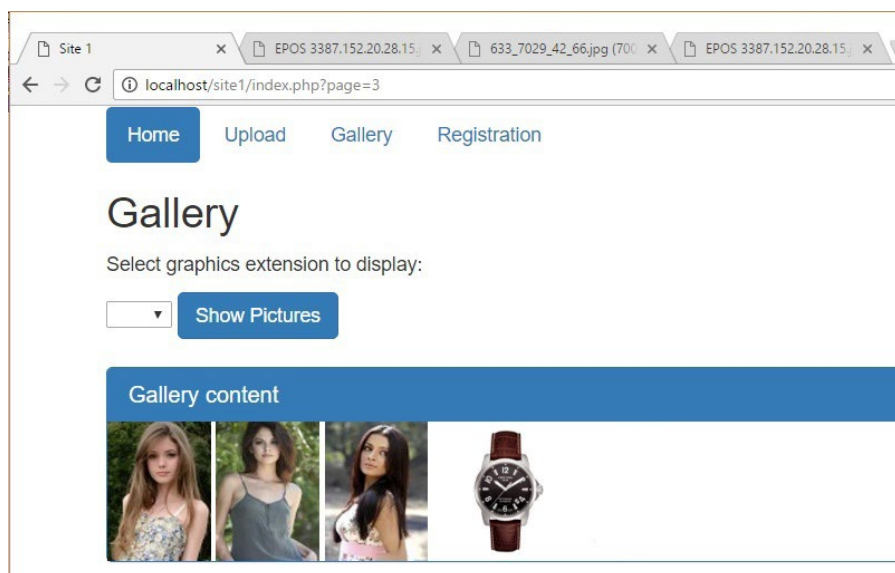


Рис. 25. Внешний вид галереи

Домашнее задание

Перейдем к обсуждению домашнего задания. Оно будет заключаться в доработке нашего сайта. Вам надо выполнить следующую работу:

- в разделе header нашего сайта создать небольшую форму входа на наш сайт, состоящую из полей для ввода login и пароля и кнопки submit;
- создать обработчик этой формы, который будет использовать для аутентификации пользователей функцию login();
- в файле function.php создать функцию login();
- запретить незарегистрированному пользователю доступ к меню upload.

Сделаю небольшую подсказку относительно функции login(). Она должна принимать два параметра: имя и пароль входящего пользователя. Эти значения надо брать непосредственно из формы входа, подобно тому, как мы делали это в форме регистрации. Полученные из формы значения обязательно надо проверить в функции.

Затем надо начать чтение файла users.txt и проверять, есть ли там пользователь с введенными именем и паролем. Не забывайте, что пароль в файле хранится в виде хешированной строки, поэтому перед сравнением пароль из формы входа тоже надо хешировать и сравнивать между собой две строки хеша. Если в файле будет найден входящий пользователь, функция должна создать

сессию и занести в нее имя вошедшего пользователя. Например, так: `$_SESSION['registered_user'] = $name`.

Если входящий пользователь не будет аутентифицирован, функция `login()` должна переадресовать его на форму регистрации. Переадресацию можно выполнять разными способами. Например, можно использовать PHP функцию `header()`, которая умеет пересылать пользователю строки-заголовки. Почитайте об этой функции здесь: <http://php.net/manual/en/function.header.php>. Например, если вызвать эту функцию с таким параметром:

```
header('Location: index.php?page=4');
```

то будет выполнена переадресация на указанную страницу. Однако, у этой функции есть одна нехорошая особенность – ее надо вызывать до любого вывода в браузер. Поэтому очень часто переадресация с этой функцией не будет работать там, где нам это надо. В таких случаях переадресацию можно сделать так:

```
echo ' <script>window.location = "index.  
php?page=4";</script>';
```

Чтобы запретить незарегистрированному пользователю доступ к меню `upload`, можно проверять наличие/значение переменной `$_SESSION['registered_user']` при входе на страницу `upload.php`.

Конечно же, каждый из вас может выполнить это задание как-нибудь иначе. Главная цель работы – предоставить доступ к меню `upload` только зарегистрированным пользователям. Творите.

