Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное
учреждение высшего образования
«Севастопольский государственный университет»

Методические указания к лабораторным работам по дисциплине «Веб-технологии» для студентов дневной и заочной форм обучения направлений 09.03.02 – «Информационные системы и технологии» и 09.03.03 – «Прикладная информатика»

Часть 2

Севастополь
2020

УДК 004.42 (076.5)
ББК 32.973-018я73
М54

Методические указания к лабораторным работам по дисциплине «Веб-технологии» для студентов дневной и заочной форм обучения направлений 09.03.02 – «Информационные системы и технологии» и 09.03.03 – «Прикладная информатика», часть 2 / Сост. А.Л. Овчинников, А.К. Забаштанский. – Севастополь: Изд-во СевГУ, 2020. – 109 с.

Методические указания предназначены для проведения лабораторных работ и обеспечения самостоятельной подготовки студентов по дисциплине «Веб-технологии». Целью методических указаний является облегчение изучения студентами основ веб-технологий и приобретения практических навыков создания веб-сайтов и веб-ориентированных систем.

Методические указания составлены в соответствии с требованиями программы дисциплины «Веб-технологии» для студентов направлений 09.03.02 – «Информационные системы и технологии» и 09.03.03 – «Прикладная информатика» и утверждены на заседании кафедры информационных систем протокол № ___ от _____ 20__ года.

Допущено научно-методической комиссией института информационных технологий и управления в технических системах в качестве методических указаний.

Рецензент: Альчаков В.В., к.т.н., доцент кафедры ИиУТС, СевГУ

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цели и задачи лабораторных работ

Основная цель выполнения настоящих лабораторных работ: приобретение практических навыков создания программного обеспечения для генерации WEB-страниц и обработки данных HTML-форм на языке PHP с использованием механизма сессий PHP, асинхронных обменов с сервером, а также с возможностью построения веб-приложений с использованием MVC фреймворка Laravel.

Выбор вариантов и график выполнения лабораторных работ

В лабораторных работах студент решает заданную индивидуальным вариантом задачу. Варианты заданий приведены к каждой лабораторной работе и уточняются преподавателем.

Лабораторная работа выполняется в два этапа. На первом этапе – этапе самостоятельной подготовки – студент должен выполнить следующее:

- проработать по конспекту и рекомендованной литературе, приведенной в конце настоящих методических указаний, основные теоретические положения лабораторной работы и подготовить ответы на контрольные вопросы;
- написать программный код, реализующий поставленную перед студентом задачу;
- оформить результаты первого этапа в виде заготовки отчета по лабораторной работе (**студенты-заочники** первый этап выполнения лабораторных работ оформляют в виде контрольной работы, которую сдают на кафедру до начала зачетно-экзаменационной сессии).

На втором этапе, выполняемом в лабораториях кафедры, студент должен:

- выполнить отладку кода программы;
- разработать и выполнить тестовый пример.

Выполнение лабораторной работы завершается защитой отчета. Студенты должны выполнять и защищать работы **строго по графику**. График защиты лабораторных работ студентами дневного отделения:

- лабораторная работа №8 – 6-ая неделя весеннего семестра;
- лабораторная работа №9 – 8-ая неделя весеннего семестра;
- лабораторная работа №10 – 10-ая неделя весеннего семестра;
- лабораторная работа №11 – 12-ая неделя осеннего семестра.
- лабораторная работа №12 – 16-ая неделя весеннего семестра;

График уточняется преподавателем, ведущим лабораторные занятия.

Описание лабораторной установки

Для выполнения лабораторных работ студенту потребуется персональный компьютер с установленной операционной системой (Windows, Linux или MacOS) и следующее программное обеспечение:

- один из современных браузеров (Google Chrome, Mozilla Firefox, Opera и тп);
- редактор программного кода (например, Sublime Text [1], Visual Studio Code [2] и тп);
- веб-сервер (например, Apache [3]);
- СУБД MySQL [4];
- Модуль вебсервера для интерпритации PHP [5].

Для ускорения установки и настройки необходимого программного обеспечения рекомендуется исползование одной из «сборок» ПО, включающих веб-сервер, СУБД и модуль PHP. В частности, может использоваться один из следующих пакетов ПО:

- Open Server [6] (только для Windows);
- WampServer [7] (только для Windows);
- XAMPP [8] (кроссплатформенный набор для Windows, Mac OS и Linux);
- MAMP [9] (кроссплатформенный набор для Windows, Mac OS и Android).

Для выполненя лабораторной работы №12 потребуется установка дополнительного программного обеспечения, которое является необходимым для работы фреймворка Laravel. Для его установки возможно использование менеджера зависимостей PHP – Composer [10]. Для упрощения установки всего необходимого для разработки с Laravel ПО возможно использование виртуальной машины Laravel Homestead [11].

Требования к оформлению отчета

Отчёт по лабораторной работе оформляются каждым студентом индивидуально. Отчёт должен включать: название и номер лабораторной работы; цель работы; вариант задания и постановку задачи; тексты разработанных программных модулей с комментариями; графическое отображение результатов проделанной работы; выводы по работе; приложения.

Постановка задачи представляет собой изложение содержания задачи и цели её решения. Также на этом этапе должны быть полностью

определены исходные данные, выбраны методы решения и дана характеристика предполагаемых результатов.

В приложении могут приводиться тексты программных модулей, результаты работы программных модулей с пояснениями, тестовые примеры и наборы данных.

Содержание отчета указано в методических указаниях к каждой лабораторной работе.

1 ЛАБОРАТОРНЯ РАБОТА №8

«Исследование архитектуры MVC приложения и возможностей обработки данных HTML-форм на стороне сервера с использованием языка PHP»

1.1 Цель работы

Исследовать особенности построения приложений с использованием архитектуры MVC. Изучить основы синтаксиса PHP и принципы функционирования MVC приложения на стороне сервера. Приобрести практические навыки использования языка PHP для генерации HTML-кода и обработки HTML-форм в MVC приложении.

1.2 Краткие теоретические сведения

1.2.1 Язык PHP

PHP [5] (Personal Home Page, а позднее Hypertext Preprocessor) – язык программирования, ориентированный на Web разработку, при помощи которого можно создавать Web-сайты и Web-ориентированные системы различных уровней сложности. В этом случае PHP-интерпретатор может работать как модуль web-сервера или как CGI-программа на web-сервере. Также можно выделить еще две возможные области использования PHP: создание скриптов, выполняющихся в командной строке (вне зависимости от Web-сервера), и создание десктопных приложений с использованием расширения PHP-GTK [12].

К преимуществам PHP относят: ориентацию на Web-разработку, кроссплатформенность, бесплатность и открытость, низкий порог вхождения (доступность и простоту изучения).

Среди недостатков выделяют: низкую производительность (по сравнению с компилируемыми языками), непоследовательный синтаксис (особенно старой части библиотеки функций), разрозненность сообщества разработчиков.

В 2009 году разработчики нескольких PHP-фреймворков договорились о создании сообщества PHP Framework Interop Group (PHP FIG), которое бы вырабатывало рекомендации для разработчиков. В литературе и статьях, посвященных PHP, на сформированные PHP FIG стандарты ссылаются, используя аббревиатуру PSR, которая расшифровывается как PHP standards recommendation. При выполнении настоящей и последующих лабораторных работ рекомендуется придерживаться PHP PSR [13].

1.2.2 Основной синтаксис и использование PHP для создания Web-страниц

Существует несколько способов создания Web-страниц с использованием языка PHP: вставка фрагментов PHP кода в HTML

страницу; вставка фрагментов HTML в PHP-скрипт; разделение бизнес логики(кода программы на PHP) и отображения данных(HTML-кода или кода с использованием различных шаблонизаторов) посредством использования паттерна MVC(см ниже).

В первом случае, HTML-страница имеет вид:

```
<html>
  <head>
  <title>Пример</title>
  </head>
  <body>
  <?php
 echo "<p> Строка, выводимая PHP!</p>";
  ?>
</body>
</html>
```

Как видно из примера выше, для размещения кода PHP внутри HTML страницы используется пара специальных тегов <?php ?> (или сокращенный вариант <? ?>). В данном примере фрагмент HTML " Строка, выводимая PHP! ", будет выведен в тело HTML-документа с помощью встроенного оператора PHP echo. При этом в браузер клиента будет передан уже только результат обработки этого документа PHP-интерпретатором, то есть, в данном случае:

```
<html>
   <head>
   <title>Пример</title>
   </head>
   <body>
   <p> Строка, выводимая PHP!</p>
   </body>
</html>
```

Обрабатывая файл (обычно это файл с расширением php, который тем не менее содержит в основном HTML код), PHP-интерпретатор пропускает текст(выдавая его без изменения на выход), пока не встретит специальный тег <?php, который сообщает ему о необходимости начать интерпретацию. Интерпретация продолжается до закрывающего тега или до конца файла(в конце файла закрывающий тег ставить не следует).

При использовании 2 способа фрагменты HTML выводятся в тексте PHP-скрипта (обычно с использованием PHP оператора echo):

```
<?php
   $f = fopen("../news.txt", "r");
   echo "<html><body>";
   echo "<h1>Последние новости:</h1>";
   for ($i=1; !feof($f) && $i <= 5; $i++) {
     echo "<li>$i-я новость: ".fgets($f, 1024);
   }
```

```
echo "</body></html>";
```

Использование 3 способа (MVC) будет рассмотрено ниже.

1.2.2.1 Разделение инструкций

Программа на PHP – это набор команд (инструкций). Как видно из примера выше, в PHP инструкции разделяются так же, как и в Си или Perl, – каждое выражение заканчивается точкой с запятой.

Закрывающий тег «?>» также подразумевает конец инструкции, поэтому перед ним точку с запятой ставить не обязательно.

1.2.2.2 Комментарии

PHP поддерживает несколько видов комментариев: в стиле Си, C++ и оболочки Unix. Символы // и # обозначают начало однострочных комментариев, /* и */ – соответственно начало и конец многострочных комментариев.

```
<?php
echo "Выводимая строка";
// Это однострочный комментарий
// в стиле C++
echo "Вторая строка";
/* Это многострочный комментарий.
Здесь можно написать несколько строк.
*/
echo"И еще одна строка";
  # Это комментарий в стиле
  # оболочки Unix
?>
```

Комментарий # является устаревшим и не рекомендуется к использованию.

1.2.2.3 Переменные, константы и операторы

Важным элементом каждого языка являются переменные, константы и операторы, применяемые к этим переменным и константам. Рассмотрим, как выделяются и обрабатываются эти элементы в PHP.

Переменные

Переменная в PHP обозначается знаком доллара, за которым следует ее имя. Например: $my_var. Имя переменной чувствительно к регистру, т.е. переменные $my_var и $My_var различны. Имена переменных соответствуют тем же правилам, что и остальные наименования в PHP: правильное имя переменной должно начинаться с буквы или символа подчеркивания с последующими в любом количестве буквами, цифрами или символами подчеркивания.

По умолчанию в PHP переменные присваиваваются по значению (кроме объектов, которые по умолчанию присваиваются по ссылке). Это

означает, к примеру, что после присвоения одной переменной значения другой, изменение одной из них не влияет на значение другой. Например:

```
<?php
$first  = ' Text ';      // Присваиваем $first значение  ' Text '
$second = $first;        // Присваиваем $second значение  переменной $first
$first  = ' New text '; // Изменяем  значение $first  на ' New text '

// выводим значение $first
echo "Переменая с именем first равна $first <br>";

echo "Переменая с именем second равна $second";
// выводим значение $second
?>
```

Результат работы этого скрипта будет следующим:

Переменная с именем first равна New text
Переменная с именем second равна Text

PHP предлагает еще один способ присвоения значений переменным: присвоение по ссылке. Для того, чтобы присвоить значение переменной по ссылке, это значение должно иметь имя, т.е. оно должно быть представлено какой-либо переменной. Чтобы указать, что значение одной переменной присваивается другой переменной по ссылке, нужно перед именем первой переменной поставить знак амперсант &.

Рассмотрим тот же пример, что и выше, только будем присваивать значение переменной first переменной second по ссылке:

```
<?php
// Присваиваем $first значение ' Text '
$first = ' Text ';

// Делаем ссылку на $first через $second.
// Теперь значения этих переменных будут всегда совпадать
$second = &$first;

// Изменим значение $first на ' New text '
$first = ' New text ';

// выведем значения обеих переменных
echo "Переменная с именем first равна $first <br>";
echo "Переменная с именем second равна $second";
?>
```

Этот скрипт выведет следующее:

Переменная с именем first равна New text.
Переменная с именем second равна New text.

То есть вместе с переменной $first изменилась и переменная $second.

Константы

Для хранения постоянных величин, т.е. таких величин, значение которых не меняется в ходе выполнения скрипта, используются константы. У константы нет приставки в виде знака доллара и ее нельзя определить простым присваиванием значения. Для определения констант в PHP может быть использована специальная функция define() или ключевое слово const вне объявления класса (с PHP 5.3.0). Синтаксис define:

```
define(
    "Имя_константы",
   "Значение_константы",
    [Нечувствительность_к_регистру]
)
```

По умолчанию имена констант чувствительны к регистру. Для каждой константы это свойство можно изменить, указав в качестве значения аргумента *Нечувствительность_к_регистру* значение True. В соответствии с PHP PSR имена констант классов всегда пишутся в верхнем регистре, при этом составные слова разделяются подчеркиванием(например, POOL_OF_CONNECTIONS).

Получить значение константы можно, указав ее имя. В отличие от переменных, не нужно предварять имя константы символом $. Кроме того, для получения значения константы можно использовать функцию constant() с именем константы в качестве параметра.

```
<?php
// определяем константу PASSWORD
define("PASSWORD","qwerty");

// определяем регистронезависимую константу PI со значением 3.14
define("PI","3.14", True);

// выведет значение константы PASSWORD, т.е. qwerty
echo (PASSWORD);

// тоже выведет qwerty
echo constant("PASSWORD");

// выведет password и предупреждение, поскольку мы
// ввели регистрозависимую константу PASSWORD
echo (password);

// выведет 3.14,
// поскольку константа PI регистронезависима по определению
echo pi;
?>
```

В PHP существует ряд констант, определяемых самим интерпретатором. Например, константа __FILE__ хранит имя файла программы (и путь к нему), которая выполняется в данный момент,

__FUNCTION__ содержит имя функции, __CLASS__ – имя класса, PHP_VERSION – версия интерпретатора PHP. Полный список предопределенных констант можно получить в руководстве по PHP [5, раздел manual].

Операторы

Операторы позволяют выполнять различные действия с переменными, константами и выражениями. Выражение это все, что имеет значение. Переменные и константы – это основные и наиболее простые формы выражений. Существует множество операций (и соответствующих им операторов), которые можно производить с выражениями. Основные из них представлены в приложении А (таблицы А.1-А.6). Полный список операторов представлен в справочном разделе сайта PHP [5].

1.2.2.4 Типы данных

PHP поддерживает десять простых типов данных.

Четыре скалярных типа:

- boolean (логический);
- integer (целый);
- float (с плавающей точкой);
- string (строковый).

Два смешанных типа:

- array (массив);
- object (объект);
- callable
- iterable.

И два специальных типа:

- resource (ресурс);
- NULL.

В PHP не принято явное объявление типов переменных. Обычно это делает сам интерпретатор во время выполнения программы в зависимости от контекста, в котором используется переменная.

Рассмотрим основные типы данных PHP

Тип boolean (булевый или логический тип)

Этот простейший тип выражает истинность значения, то есть переменная этого типа может иметь только два значения – истина TRUE или ложь FALSE. (хотя константы true и false регистронезависимы, PHP PSR-2 требует их использования в нижнем регистре).

Логические переменные используются в различных управляющих конструкциях (циклах, условиях и т.п.). Иметь логический тип могут также и некоторые операторы (например, оператор равенства).

Тип integer (целые)

Этот тип задает число из множества целых чисел Z = {..., -2, -1, 0, 1, 2, ...}. Целые могут быть указаны в десятичной, шестнадцатеричной или восьмеричной системе счисления, по желанию с предшествующим знаком «-» или «+». Для использования восьмеричной системы счисления перед числом ставится 0 (ноль), для использования шестнадцатеричной системы нужно поставить перед числом 0x. Например:

```
<?php
$a = 1234; // десятичное число
$a = -123; // отрицательное число
$a = 0123; // восьмеричное число (83 в десятичной системе)
$a = 0x1A; // шестнадцатеричное число (26 в десятичной системе)
?>
```

Размер типа integer зависит от платформы, хотя, как правило, максимальное значение равно 231 (это 32-битное знаковое). С версии PHP 5.0.5 размер integer может быть определен с помощью константы PHP_INT_SIZE, а максимальное значение - с помощью константы PHP_INT_MAX. С версии PHP 7.0.0 также введена константа PHP_INT_MIN, с помощью которой можно определить минимальное значение.

Если PHP обнаружил, что число превышает размер типа integer, он будет интерпретировать его как float. Аналогично, если результат операции лежит за границами типа integer, он будет преобразован в float.

В PHP не существует оператора деления целых. Результатом 1/2 будет число с плавающей точкой 0.5. Вы можете привести значение к целому, что всегда округляет его в меньшую сторону, либо использовать функцию round(), округляющую значение по стандартным правилам.

Для преобразования переменной к конкретному типу нужно перед переменной указать в скобках нужный тип. Например, для преобразования переменной $a=0.5 к целому типу необходимо написать (integer)($a) или а или использовать сокращенную запись (int)($a). Возможность явного приведения типов по такому принципу существует не для всех типов данных (не всегда значение одного типа можно перевести в другой тип).

Тип float (числа с плавающей точкой)

Числа с плавающей точкой (они же числа двойной точности или действительные числа) могут быть определены при помощи любого из следующих синтаксисов:

```
<?php
$a = 1.234;
$b = 1.2e3;
$c = 7E-10; ?>
```

Размер числа с плавающей точкой зависит от платформы, хотя максимум, как правило, составляет 1.8e308 с точностью около 14 десятичных цифр.

Тип string (строки)

Строка – это набор символов. В PHP 7.0.0 на 64-битных платформах нет каких-либо достижимых ограничений для длины строки, в 32-битных системах и в более ранних версиях PHP, длина строки не может быть больше 2 ГБ.

Строка в PHP может быть определена четырьмя различными способами:

- с помощью одинарных кавычек;
- с помощью двойных кавычек;
- heredoc-синтаксисом;
- nowdoc- синтаксисом.

Простейший способ определить строку – это заключить ее в одинарные кавычки «'». Чтобы использовать одинарную кавычку внутри строки, как и во многих других языках, перед ней необходимо поставить символ обратной косой черты «\», т. е. экранировать ее. Если обратная косая черта должна идти перед одинарной кавычкой либо быть в конце строки, необходимо продублировать ее «\\'».

Если внутри строки, заключенной в одинарные кавычки, обратный слэш «\» встречается перед любым другим символом (отличным от «\» и «'»), то он рассматривается как обычный символ и выводится, как и все остальные. Поэтому обратную косую черту необходимо экранировать, только если она находится в конце строки, перед закрывающей кавычкой.

В PHP существует ряд комбинаций символов, начинающихся с символа обратной косой черты. Их называют управляющими последовательностями, и они имеют специальные значения(например переход на новую строку или табуляция). Переменные и управляющие последовательности для специальных символов, встречающиеся в строках, заключенных в одинарные кавычки, не обрабатываются.

```
<?php
echo 'Также вы можете вставлять в строки
    символ новой строки таким образом,
    поскольку это нормально';

echo 'Чтобы вывести \' надо перед ней поставить \\'
// Выведет: Чтобы вывести ' надо перед ней поставить \

echo 'Вы хотите удалить C:\\*.*?';
// Выведет: Вы хотите удалить C:\*.*?

echo 'Это не вставит: \n новую строку';
// Выведет: Это не вставит: \n новую строку

echo 'Переменные $expand , $either не подставляются';
// Выведет: Переменные $expand, $either не подставляются
```

Если строка заключена в двойные кавычки «"», PHP распознает большее количество управляющих последовательностей для специальных символов. Некоторые из них приведены в таблице 1.1.

Таблица 1.1 – Управляющие последовательности

Последовательность	Значение
\n	Новая строка (LF или 0x0A (10) в ASCII)
\r	Возврат каретки (CR или 0x0D (13) в ASCII)
\t	Горизонтальная табуляция (HT или 0x09 (9) в ASCII)
\\	Обратная косая черта
\$	Знак доллара
\"	Двойная кавычка

Самым важным свойством строк в двойных кавычках является обработка переменных, т.е. вместо имени переменной будет выведено ее значение:

```
<?php
$expand=1;
$either=’abc’;
echo "Переменные $expand , {$either}  подставляются";
        // Выведет: Переменные 1, abc подставляются
```

Другой способ определения строк – это использование heredoc-синтаксиса. В этом случае строка должна начинаться с символа <<<, после которого идет идентификатор. Заканчивается строка этим же идентификатором. Закрывающий идентификатор должен начинаться в первом столбце строки. Кроме того, идентификатор должен содержать только буквенно-цифровые символы и знак подчеркивания и начинаться не с цифры или знака подчеркивания.

Heredoc-текст обрабатывается так же, как и строка в двойных кавычках. Это означает, что нет необходимости экранировать двойные кавычки в heredoc, что может улучшить читабельность кода:

```
<?php
$str = <<<EOD
Пример строки, охватывающей несколько
строчек, с использованием
heredoc-синтаксиса
EOD;
// Здесь идентификатор – EOD. Ниже
// идентификатор EOT
$name = 'Вася';
echo <<<EOT
Меня зовут "$name".
EOT;                    // это выведет: Меня зовут "Вася".
?>
```

Nowdoc-синтаксис - это то же самое для строк в одинарных кавычках, что и heredoc для строк в двойных кавычках. Nowdoc похож на heredoc, но внутри него не осуществляется никаких подстановок. Эта конструкция идеальна для внедрения PHP-кода или других больших блоков текста без необходимости его экранирования:

```
<?php
echo <<<'EOD'
Пример текста,
занимающего несколько строк,
с помощью синтаксиса nowdoc.
Обратные слеши всегда обрабатываются
буквально, например, \\ и \'.
EOD;
```

Тип array (массив)

Массив в PHP представляет собой упорядоченную хеш-таблицу – тип, который оперирует со значениями и ключами. Этот тип оптимизирован в нескольких направлениях, поэтому может быть использован как, собственно массив, список (вектор), хеш-таблица, стэк, очередь и т.д. Определить массив можно с помощью конструкции array() или непосредственно задавая значения его элементам.

```
array (
    [key1]  => value1,
    [key2]  => value2,
    ...
)
```

Языковая конструкция array() принимает в качестве параметров пары *ключ => значение*, разделенные запятыми. Символ => устанавливает соответствие между значением и его ключом. Ключ может быть как целым числом, так и строкой, а значение может быть любого имеющегося в PHP типа. Числовой ключ массива часто называют индексом. Индексирование массива в PHP начинается с нуля.

В PHP, начиная с версии 5.4, была добавлена поддержка короткого синтаксиса массивов. Например:

```
$arr = [
    "foo" => "bar",
    "bar" => "foo",
];
```

Значение элемента массива можно получить, указав после имени массива в квадратных скобках ключ искомого элемента. Если ключ массива представляет собой стандартную запись целого числа, то он рассматривается как число, в противном случае – как строка. Поэтому запись $a["1"] равносильна записи $a[1], так же как и $a["-1"] равносильно $a[-1].

```
<?php
```

```
$books = ["php" => "PHP users guide", 12 => true);
echo $books["php"]; // выведет "PHP users guide"
echo $books[12]; // выведет 1(true при выводе преобразуется в 1)
?>
```

Если для элемента ключ не задан, то в качестве ключа берется максимальный числовой ключ, увеличенный на единицу. Если указать ключ, которому уже было присвоено какое-то значение, то это значение будет перезаписано.

```
<?php
// массивы $arr и $arr1 эквиваленты
$arr  = [5 => 43, 32, 56, "b" => 12];
$arr1 = [5 => 43, 6 => 32,  7 => 56, "b" => 12];
?>
```

Если использовать в качестве ключа true или false, то его значение переводится соответственно в единицу и ноль типа integer. Если использовать NULL, то вместо ключа получим пустую строку. Можно использовать и саму пустую строку в качестве ключа, при этом ее надо брать в кавычки. Нельзя использовать в качестве ключа массивы и объекты.

Создать массив можно, просто записывая в него значения. Как уже отмечалось, значение элемента массива можно получить с помощью квадратных скобок, внутри которых нужно указать его ключ, например, $book["php"]. Если указать новый ключ и новое значение, например, $book["new_key"]="new_value", то в массив добавится новый элемент. Если не указать ключ, а только присвоить значение $book[]="new_value", то новый элемент массива будет иметь числовой ключ, на единицу больший максимального существующего. Если массив, в который мы добавляем значения, еще не существует, то он будет создан.

```
<?
// добавили в массив $books значение value с ключом ‘key’
$books[‘key’]= value;

// добавили в массив значение value1 с  ключом 13, поскольку
// максимальный ключ у нас был 12(см пример выше)
$books[] = value1;
```

Для того чтобы изменить конкретный элемент массива, нужно просто присвоить ему с его ключом новое значение. Изменить ключ элемента нельзя, можно только удалить элемент (пару ключ/значение) и добавить новую. Чтобы удалить элемент массива, нужно использовать функцию unset().

```
<?php
$books = [php" =>  "PHP users guide",  12 => true];
// добавили элемент с ключом (индексом) 13 это эквивалентно
```

```
// $books[13] = "Book about Perl";
$books[] = "Book about Perl";

// Это добавляет к массиву новый элемент с ключом "lisp" и
// значением 123456
$books["lisp"] = 123456;

// Это удаляет элемент с ключом 12 из массива

unset($books[12]);

// удаляет массив полностью
unset ($books);
```

Заметим, что, когда используются пустые квадратные скобки, максимальный числовой ключ ищется среди ключей, существующих в массиве с момента последнего переиндексирования.

```
<?php
// Создаем массив  со значениями  "a", "b" и "c". Поскольку ключи
// не указаны, они   будут 0,1,2 соответственно
$arr = array ("a","b","c");

print_r($arr);    // Выводим массив (и ключи, и значения).
unset($arr[0]);  // Удаляем из него все значения.
unset($arr[1]);
unset($arr[2]);
print_r($arr);    // Выводим массив (и ключи, и значения).

$arr[] = "aa";    // Добавляем новый элемент в массив.
print_r($arr);    // Его индексом (ключом) будет 3, а не 0
```

Переиндексировать массив можно с помощью функции array_values():

```
$arr = array_values($arr); // ключом “aa” будет 0
$arr[] = "bb";    // ключом этого элемента будет 1.
print_r($arr);
```

Результатом работы этого скрипта будет:

```
Array ( [0] => a [1] => b [2] => c )
Array ( )
Array ( [3] => aa )
Array ( [0] => aa [1] => bb )
```

1.2.2.5 Управляющие конструкции PHP

PHP содержит такие управляющие конструкции как условные операторы, циклы, операторы включения (include, require).

PHP предлагает как традиционный (C-подобный) так и альтернативный синтаксис для некоторых своих управляющих структур, а именно для if, while, for, foreach и switch. При использовании

альтернативного синтаксиса открывающую скобку нужно заменить на двоеточие “:”, а закрывающую – на endif;, endwhile; и т.д. соответственно.

Условные операторы

Оператор if

Обобщенный вид оператора if представлен ниже:

```
if (выражение) блок_выполнения
elseif(выражение1) блок_выполнения1
...
else блок_выполненияN
```

Здесь выражение есть любое правильное PHP-выражение (т.е. все, что имеет значение). В процессе обработки скрипта выражение преобразуется к логическому типу. Если в результате преобразования значение выражения истинно (true), то выполняется блок_выполнения. В противном случае блок_выполнения игнорируется. Если блок_выполнения содержит несколько команд, то он должен быть заключен в фигурные скобки { }.

Правила преобразования выражения к логическому типу:

1. В false преобразуются следующие значения:
   - логическое false;
   - целый ноль (0);
   - действительный ноль (0.0);
   - пустая строка и строка "0";
   - массив без элементов;
   - объект без переменных;
   - специальный тип null.
2. Все остальные значения преобразуются в true.

Пример использования конструкций else и elseif, с альтернативным синтаксисом:

```
<?php
if ($a == 5):
    print "a равно 5";
    print "...";
elseif ($a == 6):
    print "a равно 6";
    print "!!!";
else:
    print "a не равно ни 5, ни 6";
endif;
?>
```

Оператор switch

Синтаксис оператора switch соответствует языку C:

```
switch (выражение или переменная){
case значение1:
   блок_действий1
```

```
break;
case значение2:
   блок_действий2
break;
...
default:
   блок_действий_по_умолчанию
}
```

Для конструкции switch, как и для if, возможен альтернативный синтаксис, где открывающая switch фигурная скобка заменяется двоеточием, а закрывающая – endswitch; соответственно.

Циклы

Цикл while.

Структура:

```
while (выражение) { блок_выполнения }
либо
while (выражение): блок_выполнения endwhile;
```

Цикл do... while

Так как истинность выражения цикла do...while проверяется в конце цикла - блок_выполнения гарантированно выполняется хотя бы один раз. Структура:

```
do {блок_выполнения} while (выражение);
```

Цикл for

Структура:

```
for (выражение1; выражение2; выражение3) {блок_выполнения}
```

Цикл foreach

Конструкция foreach предназначена исключительно для работы с массивами или объектами поддерживающими интерфейс Iterable. Синтаксис:

```
foreach ($array as $value) {блок_выполнения}
foreach ($array as $key => $value)  {блок_выполнения}
```

В первом случае формируется цикл по всем элементам массива, заданного переменной $array. На каждом шаге цикла значение текущего элемента массива записывается в переменную $value, и внутренний счетчик массива передвигается на единицу (так что на следующем шаге будет доступен следующий элемент массива). Внутри блока_выполнения значение текущего элемента массива может быть получено с помощью переменной $value. Выполнение блока_выполнения происходит столько раз, сколько элементов в массиве $array. Вторая форма записи, в

дополнение ко всему перечисленному выше, на каждом шаге цикла записывает ключ текущего элемента массива в переменную $key, которую тоже можно использовать в блоке_выполнения. Когда foreach начинает исполнение, внутренний указатель массива автоматически устанавливается на первый элемент.

Операторы передачи управления.

В случае если требуется перейти к следующей итерации цикла до окончания блока выполнения или немедленно завершить работу цикла, в PHP как и в C используют операторы continue и break соответственно.

Операторы включения

Оператор *include* позволяет включать код, содержащийся в указанном файле, и выполнять его столько раз, сколько программа встречает этот оператор. Включение может производиться любым из перечисленных способов:

```
include 'имя_файла';
include $file_name;
include ("имя_файла");
```

Следует отметить, что в момент вставки файла происходит переключение из режима обработки PHP в режим HTML. Поэтому код внутри включаемого файла, который нужно обработать как PHP-скрипт, должен быть заключен в теги PHP(<? ?>).

Если файл включен с помощью include, то содержащийся в нем код наследует область видимости переменных строки, где появился include. Любые переменные вызванного файла будут доступны в вызывающем файле с этой строки и далее. Соответственно, если include появляется внутри функции вызывающего файла, то код, содержащийся в вызываемом файле, будет вести себя так, как будто он был определен внутри функции. Таким образом, он унаследует область видимости этой функции.

Оператор *require* действует почти так же, как и #include в C++. Основное отличие require и include заключается в том, как они реагируют на ситуацию, когда подключаемый файл отсутствует. Как уже говорилось, include выдает предупреждение, и работа скрипта продолжится, тогда как, require вызывает фатальную ошибку работы скрипта и прекращает его выполнение.

1.2.3 Функции в PHP

Функция, определяемая пользователем, в PHP может быть описана с помощью следующего синтаксиса:

```
function имяФункции (параметр1, параметр2,  ... параметрN){
  Блок операторов
  return "значение возвращаемое функцией";
}
```

В соответствии с PHP PSR для именования функций и методов класса используют CamelCase стиль, но начиная со строчной буквы (например, sendByHttp()). Хотя функции стандартной библиотеки PHP зачастую используют snake_case стиль (например, mysql_query()) или даже не используют подчеркивания между словами(например, strpos()).

В PHP4 функция может вызываться и до ее определения. Единственное исключение составляют функции, определяемые условно (внутри условных операторов или других функций). Когда функция определяется таким образом, ее определение должно предшествовать ее вызову.

```
<?
$make = true;
/* здесь нельзя вызвать Make_event(),
потому что она еще не существует, но можно вызвать Show_info() */
Show_info("Вася","Иванов",   "У Вас очень информативный сайт");

if ($make){
// определение функции Make_event()
  function Make_event(){
    echo "<p>Хочу изучать PHP<br>";
  }
}
// теперь можно вызывать Make_event()
Make_event();

function Show_info($first, $last, $message){ // определение функции Show_info
  echo "<br>$message<br>";
  echo "Имя: ". $first . " ". $last . "<br>";
}
// Функцию Show_info можно вызывать и здесь
Show_info("Федя","Федоров",  "На мой взгляд Вашему сайту не хватает графики");
```

Если функция однажды определена в программе, то переопределить или удалить ее позже нельзя. Не смотря на то, что имена функций нечувствительны к регистру, лучше вызывать функцию по тому же имени, каким она была задана в определении.

Аргументы функций

У каждой функции может быть список аргументов. Каждый аргумент представляет собой переменную или константу.

С помощью аргументов данные в функцию можно передавать тремя различными способами. Это передача аргументов по значению (используется по умолчанию), по ссылке и задание значения аргументов по умолчанию. Когда аргумент передается в функцию по значению,

изменение значения аргумента внутри функции не влияет на его значение вне функции. Чтобы позволить функции изменять ее аргументы, их нужно передавать по ссылке. Для этого в определении функции перед именем аргумента следует написать знак амперсант «&».

```
<?php
// напишем функцию, которая бы добавляла к строке слово checked
function add_label(&$data_str){
  $data_str .= "checked";
}
$str = "<input type=radio name=article ";  // пусть имеется такая строка
echo $str .="><br>";  // выведет элемент формы –  не отмеченную радио кнопку
add_label($str);    // вызовем функцию
echo $str .="><br>";//выведет уже отмеченную радио кнопку(с атрибутом checked)
```

В функции можно определять значения аргументов, используемые по умолчанию. Само значение по умолчанию должно быть константным выражением, а не переменной и не представителем класса или вызовом другой функции. Например:

```
<?php
function Message($firstname="Вася",$secondname="Петров"){
  echo "$firstname $secondname";
}
Message();  // вызываем функцию без параметра.
    // В этом случае функция выведет – Вася Петров
Message("Петя","Иванов"); // В этом случае функция выведет:  Петя Иванов
```

Если у функции несколько параметров, то те аргументы, для которых задаются значения по умолчанию, должны быть записаны после всех остальных аргументов в определении функции. В противном случае появится ошибка, если эти аргументы будут опущены при вызове функции.

Списки аргументов переменной длины

В PHP (как и в JavaScript) можно создавать функции с переменным числом аргументов. В этом случае в PHP для работы с аргументами используются встроенные функции: func_num_args(), func_get_arg(), func_get_args().

Функция func_num_args() возвращает число аргументов, переданных в текущую функцию. Эта функция может использоваться только внутри определения пользовательской функции. Если она появится вне функции, то интерпретатор выдаст предупреждение.

```
<?php
function DataCheck(){
  $n = func_num_args();
```

```
  echo "Число аргументов функции $n";
}
DataCheck();
   // Выведет: "Число аргументов функции 0"
DataCheck(1,2,3);
   // Выведет: "Число аргументов функции 3"
```

Функция *func_get_arg (номер_аргумента)* возвращает аргумент из списка переданных в функцию аргументов, порядковый номер которого задан параметром номер_аргумента. Аргументы функции считаются, начиная с нуля. Как и func_num_args(), эта функция может использоваться только внутри определения какой-либо функции. Номер_аргумента не может превышать число аргументов, переданных в функцию. Иначе будет сгенерировано предупреждение, и функция func_get_arg() возвратит False.

Функция *func_get_args()* возвращает массив, состоящий из списка аргументов, переданных функции. Каждый элемент массива соответствует аргументу, переданному функции. Если функция используется вне определения пользовательской функции, то генерируется предупреждение.

Глобальные переменные

Чтобы использовать внутри функции переменные, заданные вне нее, эти переменные нужно объявить как глобальные. Для этого в теле функции следует перечислить их имена после ключевого слова *global*:

```
<?
$a=1;
function Test_g(){
global $a;
  $a = $a*2;
  echo 'в результате работы функции $a=',$a
}
echo 'вне функции $a='.$a;//выведет - вне функции $a=1
echo "<br>";
Test_g();//выведет - в результате работы функции $a=2
echo "<br>";
echo 'вне функции $a='.$a;//выведет - вне функции $a=2
echo "<br>";
Test_g();//выведет - в результате работы функции $a=4
```

В PHP существует массив $GLOBALS, который содержит все переменные, определенные в глобальной области видимости. Поэтому можно использовать глобальную переменную $a внутри функции без ее предварительного описания следующим образом:

```
<?
$GLOBALS["a"];
```

Статические переменные

Чтобы использовать переменные только внутри функции, при этом сохраняя их значения и после выхода из функции, нужно объявить эти переменные как статические. Статические переменные видны только внутри функции и не теряют своего значения, если выполнение программы выходит за пределы функции. Объявление таких переменных производится с помощью ключевого слова static. Статической переменной может быть присвоено любое значение, но не ссылка.

```
<?
function Test_s(){
  static $a = 1;
  // нельзя присваивать выражение или ссылку
  $a = $a*2;
  echo $a;
}
Test_s(); // выведет 2
echo $a;  // ничего не выведет, так как  $a доступна только  внутри функции
Test_s(); // внутри функции $a=2, поэтому  результатом  будет число 4
```

Возвращаемые значения

Кроме действий, любая функция может возвращать как результат своей работы какое-нибудь значение. Это делается с помощью утверждения return. Возвращаемое значение может быть любого типа, включая массивы и объекты. Когда интерпретатор встречает команду return в теле функции, он немедленно прекращает ее исполнение и переходит на ту строку, из которой была вызвана функция.

В результате работы функции может быть возвращено только одно значение. Несколько значений можно получить, если возвращать список значений (одномерный массив). Допустим, мы хотим получить полный возраст человека с точностью до дня по дате рождения:

```
<?php
function Full_age($b_day, $b_month, $b_year){
  if ((date("m") > $b_month) && (date("d")>$b_day)) {
    $day = date ("d") – $b_day;
    $month = date("m") – $b_month;
    $year = date("Y") – $b_year;
  }
 else {
    $year = date("Y") – $b_year – 1;
    $day = 31 – $b_day – date ("d");
    $month = 12 – ($b_month – date("m"));
  }
  return [$day,$month, $year];
}
$age = Full_age("07","08","1998");
echo "Вам $age[2] лет, $age[1] месяцев  и $age[0] дней";
```

// выведет возраст на текущую дату

1.2.4 Обработка строк в PHP

Для работы со строковыми (текстовыми) данными в PHP предусмотрен очень богатый набор функций. Для использования этих функций не требуется выполнения каких-либо дополнительных действий (подключения библиотек и др.), поскольку они являются частью ядра PHP. Наиболее часто используемые из них представлены ниже.

string strtok(string arg1, string arg2) – функция возвращает строку по частям. При первом вызове функция принимает два аргумента: исходную строку arg1 и разделитель arg2. Она возвращает часть строки arg1 до разделителя arg2. При последующих вызовах функции возвращается следующая часть до следующего разделителя, и так до конца строки.

array explode(string arg, string str [, int maxlimit]) – функция производит разделение строки в массив. Она возвращает массив строк, каждая из которых соответствует фрагменту исходной строки str, находящемуся между разделителями, указанными аргументом arg. Необязательный параметр maxlimit указывает максимальное количество элементов в массиве. Оставшаяся неразделенная часть будет содержаться в последнем элементе.

string implode(string glue, array param) – функция является обратной функции explode() и производит объединение массива в строку. Функция возвращает строку, которая последовательно содержит все элементы массива, заданного в параметре param, между которыми вставляется значение, указанное в параметре glue.

string substr(string string, int start[, int length]) – функция возвращает часть строки. Первый аргумент – исходная строка; второй – индекс символа в строке (отсчет начинается с нуля), начиная с которого необходимо вернуть подстроку; третий – длина строки в символах, которую надо вернуть. Если третий аргумент не указан, то возвращается вся оставшаяся часть строки.

string strpos(string haystack, string needle[, int offset]) – функция возвращает позицию в строке haystack, в которой найдена переданная ей подстрока needle. Необязательный параметр offset позволяет указать в строке позицию, с которой надо начинать поиск.

string strrpos(string haystack, string needle) – функция ищет в строке haystack последнюю позицию, где встречается символ needle.

string strstr(string haystack, string needle) – функция возвращает участок строки, заданной в параметре haystack, начиная с первого фрагмента, указанного в параметре needle и до конца строки. В случае неудачи функция возвращает false. Чувствительна к регистру.

string str_replace(string from, string to, string str) – функция заменяет в строке str все вхождения подстроки from на to и возвращает результат –

обработанную строку. Если и *search*, и *replace* - массивы, то **str_replace()** использует все значения массива *search* и соответствующие значения массива *replace* для поиска и замены в *subject*. Если в массиве *replace* меньше элементов, чем в *search*, в качестве строки замены для оставшихся значений будет использована пустая строка. Если *search* - массив, а *replace* - строка, то *replace* будет использована как строка замены для каждого элемента массива *search*.

int strcmp(string str1, string str2) – функция сравнивает две строки (с учетом регистра) и возвращает:

- 0 - если строки полностью совпадают;
- >0 - если, строка str1 лексикографически больше str2;
- <0 - если, наоборот, строка str1 лексикографически меньше str2.

int strncmp(string str1, string str2, int len) – функция отличается от strcmp() тем, что сравнивает начала строк, а точнее первые len байтов. В остальном функция ведет себя аналогично предыдущей.

int strlen (string $string) – функция возвращает длину строки, которую принимает в качестве аргумента.

trim(string $string) – функция принимает в качестве своего единственного аргумента строку, и удаляет из нее пробелы слева и справа.

htmlspecialchars(string str [, int quote_style [, string charset]]) – функция производит преобразование спецсимволов в их HTML эквиваленты (эскейп-последовательности). Это гарантирует, что введенный пользователем в форме текст отобразится браузером так же, как был введен (в противном случае, если, например, пользователь введет фрагмент JavaScript, то отображение этого текста приведет к выполнению скрипта). Первый аргумент – строка, в которой надо выполнить преобразование. В качестве второго необязательного аргумента принимается константа, задающая режим преобразования кавычек. По умолчанию, используется ENT_COMPAT, преобразующая двойные кавычки, при этом одиночные остаются без изменений. В режиме ENT_QUOTES преобразуются и двойные, и одиночные кавычки, а в режиме ENT_NOQUOTES и двойные, и одиночные кавычки остаются без изменений. Третий необязательный аргумент принимает строку, представляющую набор символов, используемых в преобразовании (по умолчанию ISO-8859-1). К примеру, если Вы обрабатываете какое-то сообщение msg формы, и выводите его без htmlspecialchars:

```
<?
$msg = $_POST["msg"];
echo $msg;
```

то в этом случае, если пользователь введет в поле msg фрагмент:

```
<Script Language="JavaScript">
  alert("Привет!"); // функция вывода в JavaScript
</Script>
```

При выводе будет выполнен JavaScript и выведено окно alert. Чтобы избежать этого перед выводом msg достаточно добавить:

```
$msg = htmlspecialchars($msg);
```

array parse_url(string url) – функция возвращает ассоциативный массив, включающий множество различных существующих компонентов URL: "scheme", "host", "port", "user", "pass", "path", "query" и "fragment". Пример:

```
<?
$url = "http://www.google.com/search?hl=ru&ie=UTF-8&oe=UTF-8&q=php&lr=";
$arr = parse_url($url);
print_r($arr);
```

результат следующий:

```
Array ( [scheme] => http [host] => www.google.com [path] => /search [query] => hl=ru&ie=UTF-8&oe=UTF-8&q=PHP&lr= )
```

1.2.5 Обработка HTML-форм с помощью PHP

Внутри PHP-скрипта для получения доступа к данным, переданным *клиентом* по протоколу *HTTP*, используются суперглобальные массивы *$_POST* и *$_GET*, в зависимости от метода передачи данных.

Допустим, нам нужно обработать форму, содержащую элементы ввода с именами first_name, last_name. Данные были переданы методом *POST*. Приведем пример использования массива $_POST:

```
<?php
$str = "Здравствуйте,".$_POST ["first_name"]." ".$_POST ["last_name"] ."!<br>";
echo $str;
```

Иногда возникает необходимость узнать значение какой-либо *переменной окружения*, например метод, использовавшийся при передаче запроса или IP-адрес компьютера, отправившего запрос. Получить такую информацию можно с помощью суперглобального массива $_SERVER. Он содержит множество переменных окружения Web-сервера.

Пусть необходимо получать данные (ФИО, E-mail) и сообщение от пользователя, заполнившего форму и вывести ему полученные данные. Текст PHP скрипта, реализующего данную задачу, представлен ниже:

```
<?
//если использован метод POST – надо выводить сообщение пользователю
if ($_SERVER["REQUEST_METHOD"]=="POST") {
  //формируем строку с именем пользователя
  $usr_name=$_POST["first_name"]." ".$_POST["last_name"];
  echo "Уважаемый, ".$usr_name."!<br> Ваше сообщение:<br> ".$_POST["message"];
}
//иначе просто выводим форму
else {
```

```
?>
 <html>
 <head>
        <title>Форма</title>
 </head>
 <body>
 <h1>Форма обратной связи</h1>
  <form action="<? echo $_SERVER["PHP_SELF"]?>" method=POST>
                <?//используем $_SERVER["PHP_SELF"] для отправки данных в этот же самый скрипт?>
 Имя <br><input type=text name="first_name"><br>
 Фамилия <br><input type=text name="last_name"><br>
 E-mail <br><input type=text name="email"><br>
 Сообщение <br><textarea name=message cols=50 rows=5></textarea><br>
 <input type=submit value="Отправить">
 <input type=reset value="Отменить">
  </form>
 </body>
 </html>
<?
}?>
```

1.2.6 Объектно-ориентированное программирование в PHP

Начиная с пятой версии, PHP обладает полноценной поддержкой ООП. PHP поддерживает все три основных механизма ООП — *инкапсуляцию*, *полиморфизм* и *наследование*. Множественное наследование классов не поддерживается, однако класс может реализовывать несколько *интерфейсов*, а также использовать готовую функциональность от нескольких *трейтов*.

Рассмотрим подробнее основные особенности ООП в PHP.

Как было отмечено выше, в PHP (начиная с версии 5) ***присвоение объекта или его передача в качестве параметра функции происходит по умолчанию по ссылке***, а не по значению. Представленный ниже код, может продемонстрировать данную особенность.

```
<?php
    class MyClass {
        var $property;
    }
    $obj1 = new MyClass;
    $obj1->property = 1;
    $obj2 = $obj1;
    $obj2->property = 2;
    echo $obj1->property; // Выводит 1 в PHP 4 и 2 в PHP 5
    echo $obj2->property; // Выводит 2
?>
```

В данном случае $obj1 и $obj2 указывают на один и тот же объект, так как оператор $obj2 = $obj1 копирует не сам объект, а только его идентификатор. В том случае, если необходимо провести именно копирование объекта, придется явно использовать новый метод __clone(). При этом объект копируется со всеми своими методами, свойствами и их значениями:

```
<?php
    class MyClass{
        var $property;
    }
    $obj1 = new MyClass;
    $obj1->property = 1;
    $obj2 = clone $obj1;
    echo $obj1->property; // Выводит 1
    echo $obj2->property; // Выводит 1
    $obj2->property = 2;
    echo $obj2->property; // Выводит 2
?>
```

Следует обратить внимание на то, что к методу __clone() нельзя обратиться непосредственно и для копирования объекта используется ключевое слово clone. Метод __clone() необязательно описывать в самом классе, однако его явное определение, т.е. *перегрузка*, позволяет, например, изменить значения свойств копируемого объекта:

```
<?php
    class MyClass{
        var $property;
        function __clone() {
            $this->property = 2;
        }
    }
    $obj1 = new MyClass;
    $obj1->property = 1;
    $obj2 = clone $obj1;
    echo $obj1->property; // Выводит 1
    echo $obj2->property; // Выводит 2
?>
```

Метод __clone() не принимает аргументов, однако позволяет обратиться к получаемому в результате объекту через указатель $this.

В PHP существуют спецификаторы доступа ***public***, ***protected*** и ***private***, которые позволяют указать степень доступа к свойствам и методам класса. К общедоступным (public) свойствам и методам можно получить доступ без каких либо ограничений. Защищенные (protected) элементы класса доступны внутри класса, в котором они объявлены, и в производных от него классах. Частные (private) элементы доступны только в классе, в котором они объявлены.

```
<?php
    class MyClass {
        public $public_value = "общедоступный элемент";
        protected $protected_value = "защищенный элемент";
        private $private_value = "частный элемент";
        public function printPrivate() {
            echo $this->private_value.'</br>';
        }
    }
    $obj1 = new MyClass;
    echo $obj1->public_value.'</br>'; //Выводит "общедоступный элемент"
    class MyClass1 extends myClass {
        public function printProtected() {
            echo $this->protected_value.'</br>';
        }
    }
    $obj2 = new MyClass1();
    $obj2->printProtected(); // Выводит "защищенный элемент"
    $obj1->printPrivate(); //Выводит "частный элемент"
    echo $obj1->protected_value.'</br>'; // Вызывает ошибку доступа
    echo $obj1->private_value.'</br>'; // Вызывает ошибку доступа
?>
```

В PHP существуют ***конструкторы*** и ***деструкторы***. Метод-конструктор вызывается автоматически при каждом создании объекта. В PHP существует схема именования конструктора - метод **__construct()** является конструктором класса. Аналогично, при уничтожении объекта вызывается специальный метод **__destruct()** – деструктор класса.

```
<?php
    class MyClass {
        function __construct() {
            echo "Запущен конструктор";
        }
        function __destruct() {
            echo "Запущен деструктор";
        }
    }
    $obj = new MyClass(); // Выводит "Запущен конструктор"
    unset($obj); // Выводит "Запущен деструктор"
?>
```

Если необходимо вызвать конструктор или деструктор базового класса, то необходимо это делать явно, через указатель parent.

```
<?php
    class MyClass {
        function __construct() {
            echo "Запущен конструктор базового класса";
        }
        function __destruct() {
            echo "Запущен деструктор базового класса";
```

```
        }
    }
    class MyClass1 extends MyClass {
      function __construct() {
         echo "Запущен конструктор дочернего класса";
         parent::__construct();
      }
      function __destruct() {
        echo "Запущен деструктор дочернего класса";
         parent::__destruct();
      }
   }
   $obj = new MyClass1();
   unset($obj);
?>
```

В PHP существуют ***абстрактные*** (abstract) классы и методы. Абстрактные методы имеют только объявление и не имеют реализации. Класс, который содержит такие методы, должен быть обязательно объявлен как абстрактный.

```
<?php
    abstract class MyClass {
       abstract public function abstrFunc();
    }
    class MyClass1 extends MyClass {
       public function abstrFunc() {
          echo 1;
       }
    }
    $obj = new MyClass1;
    $obj->abstrFunc(); // Выводит 1
?>
```

При этом невозможно создать объект абстрактного класса, можно только определять новые классы от базового абстрактного класса и создавать объекты уже от производных классов. Стоит отметить, что абстрактные классы также могут содержать и обычные (не абстрактные) элементы.

В PHP 5 появилось понятие ***интерфейса*** (interface) . Интерфейсами являются абстрактные классы, содержащие только абстрактные методы и не имеющие никаких свойств. Основное отличие интерфейсов от абстрактных классов заключается в том, что класс не может быть порожден от нескольких классов, в том числе и абстрактных, но зато может быть создан на основе любого числа интерфейсов. При этом в интерфейсе методы объявляются ключевым словом function без указания каких-либо спецификаторов, в том числе и abstract.

```
<?php
     interface Int1 {
```

```
        function func1();
    }
    interface Int2 {
        function func2();
    }
    class MyClass implements Int1, Int2 {
        public function func1() {
            echo 1;
        }
        public function func2() {
            echo 2;
        }
    }
    $obj = new MyClass;
    $obj->func1(); // Выводит 1
    $obj->func2(); // Выводит 2
?>
```

Начиная с версии 5.4, в PHP введен дополнительный инструмент для повторного использования кода в классах - ***трейты***. В отличие от интерфейсов, трейты содержат не абстрактные методы, а общие фрагменты классов.

Если мы объявляем какие-либо интерфейсы и заставляем все классы реализовывать методы этих интерфейсов, наверняка среди реализаций будет довольно много повторяющегося кода. Для решения этой проблемы как раз и предназначены трейты. Реализовав один раз определенную функциональность, ее можно “подмешивать” в любой класс, в котором данная функциональность может потребоваться.

Объявляются трейты при помощи ключевого слова trait, после которого следует название трейта и в фигурных скобках его содержимое. Для включения одного или нескольких трейтов в класс используется ключевое слово use. Например:

```
<?php ## Использование трейтов
trait Seo
{
    private $keyword;
    private $description;
    private $ogs;
    public function keywords()
    {
        // $query = "SELECT keywords FROM seo WHERE id = :id LIMIT 1"
        echo "Seo::keywords<br />";
    }
    public function description()
    {
        // $query = "SELECT description FROM seo WHERE id = :id LIMIT 1"
        echo "Seo::description<br />";
    }
```

```
    public function ogs()
    {
        // $query = "SELECT ogs FROM seo WHERE id = :id LIMIT 1"
        echo "Seo::ogs<br />";
    }
}

class News
{
    // Новости снабжаются SEO-информацией
    use Seo;
    private $id;
}

$news = new News();
$news->keywords();
```

Следует отметить, что трейты перегружают методы базового класса, но методы текущего класса перегружают методы трейтов.

Метод, при определении которого используется ключевое слово ***final***, не может быть переопределен в классах, производных от данного класса.

```
<?php
    class MyClass {
        final public function func() {
            // Код метода
        }
    }
    class MyClass1 extends MyClass {
        // Следующий код вызывает ошибку
        // переопределения финального метода
        // базового класса MyClass
        public function func() {
            // Код метода
        }
    }
?>
```

Кроме этого, если final используется при определении самого класса, то порождение от него других классов становится невозможным.

```
<?php
    final class MyClass {
        // Код описания класса
    }
    // Следующий код вызывает ошибку
    // порождения от финального класса
    class MyClass1 extends MyClass {
        // Код описания класса
    }
```

```
?>
```

Если класс определен как final, то и все методы данного класса автоматически становятся финальными, таким образом, определять их явно как final уже нет необходимости.

В PHP существует особый элемент класса – ***константа***.

```
<?php
    class MyClass {
        const CONSTANT = "константа класса";
    }
    echo MyClass::CONSTANT; // Выводит "константа класса"
?>
```

Также в PHP возможно объявление ***статических*** свойств класса.

```
<?php
     class MyClass {
         static $static = 1;
     }
     echo MyClass::$static; // Выводит 1
?>
```

Статические свойства едины для всего класса и не могут принадлежать ни одному из объектов класса. Изменение такого свойства в одном из методов любого объекта приводит к его изменению для всех остальных объектов данного класса. Кроме этого, становится возможным обратиться к такому свойству вне контекста объекта.

Статические методы классов в PHP, также как и статические свойства, принадлежат всему классу в целом. Это позволяет использовать такой метод без создания какого-либо объекта такого класса.

```
<?php
     class MyClass {
         static function statFunc() {
             echo "статический метод";
         }
     }
     MyClass::statFunc(); // Выводит "статический метод"
?>
```

Однако в статическом методе становится невозможным использовать указатель *$this*, так как при вызове статического метода или неизвестно в контексте какого объекта он вызывается, или такого объекта может и не существовать вовсе.

Статические свойства могут использоваться, например, для счетчиков количества экземпляров объектов одного класса.

```
<?php
class Stats {
```

```
  static $counter;
  function __construct($name, $authors, $category,
    Stats::$counter++;
  }
  // ...
  function __destruct() {
    parent::__destruct();
    Stats::$counter--;
  }
}
?>
```

Таким образом, обратившись к статическому свойству, можно определить количество инициализированных в данный момент экземпляров классов: echo Stats::$counter;.

Специальное ключевое слово ***instanceof*** в PHP позволяет определять является ли объект экземпляром определенного класса, или же экземпляром класса производного от какого-либо класса.

```
<?php
    class MyClass { }
    $obj1 = new MyClass();
    if ($obj1 instanceof MyClass) {
        echo "\$obj1 - объект класса MyClass";
    }
     class MyClass1 extends MyClass { }
    $obj2 = new MyClass1();
    if ($obj2 instanceof MyClass) {
        echo "\$obj2 - объект класса, производного от MyClass";
    }
?>
```

Также с помощью instanceof можно определить является ли объект экземпляром класса, созданного на основе определенного интерфейса.

```
<?php
    interface Int { }
    class MyClass2 implements Int { }
    $obj3 = new MyClass2();
    if ($obj3 instanceof Int) {
        echo "\$obj3 – объект класса, созданного на основе интерфейса Int";
    }
?>
```

Специальная функция **__*autoload*()** будет вызываться в случае попытки создания объекта неопределенного класса.

```
<?php
   function __autoload($class) {
      echo "попытка создать объект неопределенного класса ", $class;
```

```
    }
    $obj = new MyClass;
?>
```

В качестве параметра функции __autoload() передается имя неопределенного класса.

В PHP возможна ***перегрузка*** доступа к свойствам объектов.

```
<?php
    class MyClass {
        private $properties;
        function __set($name, $value) {
            echo "задание нового свойства $name = $value";
            $this->properties[$name]=$value;
        }
        function __get($name) {
            echo "чтение значения свойства ", $name;
            return $this->properties[$name];
        }
    }
    $obj = new MyClass;
    $obj->property = 1; // Выводит "задание нового свойства property=1"
    $a = $obj->property; // Выводит "чтение значения свойства property"
    echo $a; // выводит 1;
?>
```

Методы доступа ***__get()*** *и* ***__set()*** позволяют легко проводить динамическое назначение свойств объектам. В качестве параметров этим методам передаются имена свойств. Метод __set() также получает и значение, которое устанавливается для свойства.

При вызове в PHP несуществующего метода объекта автоматически вызывается специальный метод **__call()**.

```
<?php
    class MyClass {
        function __call($name, $params) {
            echo "Вызван метод $name с параметром $params[0]";
        }
    }
    $obj = new MyClass;
    echo $obj->method(1); // Выводит "Вызван метод method
                          // с параметром 1"
?>
```

В качестве параметров __call() принимает имя вызываемого метода и передаваемые этому методу параметры.

В PHP псевдо-константа ***__METHOD__*** возвращает имя класса и вызываемый метод. При обращении к функции вне класса __METHOD__ возвращает только имя функции.

```
<?php
    class MyClass {
        public function myMethod() {
            echo "вызов метода ", __METHOD__;
        }
    }
    $obj = new MyClass;
    $obj->myMethod(); // Выводит "вызов метода MyClass::myMethod"
    function myFunction() {
        echo "вызов функции ", __METHOD__;
    }
    myFunction(); // Выводит "вызов функции myFunction"
?>
```

В PHP введен еще один специальный метод класса – ***__toString()***.Данный метод позволяет выполнить перегрузку преобразования объекта в строку.

```
<?php
    class MyClass {
        function __toString() {
            return "вызван метод __toString()";
        }
    }
    $obj = new MyClass;
    echo $obj; // Выводит "вызван метод __toString()"
?>
```

В PHP 5 введена возможность ***разыменования*** (dereferencing) объектов, которые возвращаются функциями.

```
<?php
    class MyClass1 {
        public function showClassName() {
            echo "объект класса MyClass1";
        }
    }
    class MyClass2 {
        public function showClassName() {
            echo "объект класса MyClass2";
        }
    }
    function deref($obj) {
        switch ($obj) {
            case "MyClass1":
                return new MyClass1();
            case "MyClass2":
                return new MyClass2();
```

```
        }
    }
    deref("MyClass1")->showClassName(); // Выводит "объект  класса My-
Class1"
    deref("MyClass2")->showClassName(); // Выводит "объект класса My-
Class2"
?>
```

Данный механизм позволяет вызывать методы объектов, экземпляры классов которых возвращаются пользовательскими функциями.

1.2.7 Шаблон проектирования MVC

Шаблон проектирования *MVC*(от англ. Model-View-Controller, Модель-Представление-Контроллер) – позволяет разделить код приложения на 3 взаимосвязанные части: бизнес логику, представление и обработку действий пользователя.

Модель (Model) – предоставляет собой объектную модель некой предметной области. Может включать в себя данные (или обеспечивает доступ к хранилищу данных, например, к БД) и методы работы с этими данными, реагирует на запросы контроллера, возвращая данные и/или изменяя своё состояние. При этом модель не содержит в себе информации, каким образом можно визуализировать данные.

Представление (View) – отвечает за отображение (визуализацию) информации. При этом одни и те же данные могут представляться различными способами: например, коллекцию объектов при помощи разных “шаблонов” отбражения можно представить как в табличном виде, так и в виде списка, или же для различных устройств отображения (мобильная платформа или десктоп).

Контроллер (Controller) – обеспечивает связь между пользователем и системой. Использует модель для проверки (валидации) и обработки поступивших данных (например, сохранения в БД или извлечения необходимых данных на основании поступивших параметров). Как правило, на уровне контроллера осуществляется так же и авторизация (проверяются права пользователя на выполнение действий или получение информации). Контроллер использует представление для отображения данных полученных от можели в требуемом виде.

***Примечание**: Для лучшего понимания работы MVC приложений рекомендуется ознакомиться с существующими PHP фреймворками, такими как: YII[14], Laravel[11, 15], Zend[16] и т.д.*

1.2.8 Рекомендации по реализации MVC приложения на PHP

Перед тем как приступить к реализации MVC приложения рассмотрим обобщенный алгоритм его работы.

Как было отмечено выше, обработкой запросов от пользователя занимается контроллер. Однако, MVC приложение редко имеет только одну сущность в предметной области, поэтому для каждой сущности реализуют собственный контроллер. Например, *TestController* – контроллер для обработки запросов со страницы “Тест по дисциплине”, а *AlbumController* – для обработки запросов со страницы “Фотоальбом”.

Зачастую, для того чтобы объединить все действия по обработке запросов в одном месте, все запросы пользователя направляют на единую точку входа приложения – обычно это файл index.php. В файле index.php размещают вызов *роутера* – кода, который на основании параметров запроса принимает решение о том, какой контроллер и какой медод контроллера (*Action*) должен быть вызван и вызывает его. Адресная строка входящего запроса в этом случае может иметь следующий вид: *http://mysite.ru/index.php?controller=test&action=index* или так как файл index.php в настройках Web-сервера устанавливается файлом, вызываемым по умолчанию – *http://mysite.ru/?controller=test&action=index*.

Для увеличения читабельности URL, реализуют так называемые ЧПУ(человекопонятные URL). В этом случае URL может иметь следующий вид: *http://mysite.ru/test/index*. Для того чтобы преобразовать URL из такого формата в формат с параметрами, может быть использован специальный модуль Web-сервера. В состав Web-сервера Apache для преобразования входящих URL входит модуль *mod_rewrite* [3]. Он позволяет с использованием набора регулярных выражений задать правила преобразования URL.

Чтобы использовать *mod_rewrite* для преобразования URL вида:

http://mysite.ru/test/index

в URL:

http://mysite.ru/index.php?controller=test&action=index

необходимо в корневой директории сайта разместить файл *.htaccess*, содержащий следующий набор директив:

```
RewriteEngine On
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteRule ^(.*)/(.*) index.php?controller=$1&action=$2 [QSA]
```

Таким образом, обработка запроса в том случае, когда пользователь открывает страницу “Тест по дисциплине”, будет включать следующие действия:

1. GET-запрос *http://mysite.ru/test/index* поступает на “вход” Web-сервера;

2. Модуль *mod_rewrite* преобразует URL в *http://mysite.ru/index.php?controller=test&action=index*;
3. Код роутера в файле index.php проверяет наличие контроллера *TestController* и вызывает его метод *index*;
4. В методе *index* контроллера *TestController* вызывается метод *render* представления, который подставляет указанную HTML форму теста по дисциплине в основной шаблон(*layout*) сайта и выводит полученный HTML код.

В том случае, если пользователь заполнит форму и нажмет кнопку отправить, получим следующую последовательность действий:

1. POST-запрос *http://mysite.ru/test/validate* с данными формы в теле запроса поступает на “вход” Web-сервера;
2. Модуль *mod_rewrite* преобразует URL в *http://mysite.ru/index.php?controller=test&action=validate*;
3. Код роутера в файле index.php проверяет наличие контроллера *TestController* и вызывает его метод *validate*;
4. В методе *validate* контроллера *TestController* вызывается метол модели *TestModel*, осуществляющий проверку данных(*валидацию*) и формирующий список ошибок *Errors*, если таковые имеются.
5. В методе *validate* контроллера *TestController* вызовом метода *render* в основной шаблон(*layout*) сайта подставляется либо список ошибок *Errors*, либо сообщение об успешном сохранении данных.

Для реализации MVC приложения рекомендуется организовать следующую структуру файлов и папок (см. таблицу 1.2):

Таблица 1.2 – Рекомендуемая структура файлов и папок

Каталог/файл	**Описание**
/.htaccess	Файл содержит настройки Web-сервера, в часности правила преобразования URL
/index.php	Точка входа подключение необходимых библиотек и вызов роутера
/app/core/router.php	Файл содержит класс роутера приложения
/app/core/controller.php	Файл содержит базовый класс контроллера приложения
/app/core/model.php	Файл содержит базовый класс модели приложения
/app/core/view.php	Файл содержит базовый класс представления приложения
/app/controllers/	Каталог содержит классы контроллеров
/app/models/	Каталог содержит классы моделей
/app/models/validators	Каталог содержит классы валидаторов
/app/views/	Каталог содержит файлы представлений
/public/assets/js/	Каталог содержит все необходимые JavaScript файлы приложения
/public/assets/css/	Каталог содержит все необходимые CSS файлы приложения
/public/assets/img/	Каталог содержит все изображения приложения

Реализация роутера может выглядеть следующим образом:

```
<?
class Router
{
  static function route()
  {
    //Получаем имя контроллера или "page" по умолчанию
    $controller_name = $_REQUEST["controller"] ? $_REQUEST["controller"] : "page";
    //Получаем имя экшена или "index" по умолчанию
    $action_name = $_REQUEST['action'] ? $_REQUEST['action'] : "index";
    //Путь и имя файла контроллера
    $controller_file = "app/controllers/".$controller_name.'_controller.php';
    //Проверяем наличие файла контроллера и завершаем работу в случае его
отсутствия
    if(file_exists($controller_file)){
      include $controller_file;
    }  else{
      die "ОШИБКА! Файл контроллера $controller_file не найден!";
    }
    //Создаем экземпляр контроллера
    $controller_class_name = ucfirst($controller_name).'Controller';
    $controller = new $controller_class_name;

    //Получаем имя модели и имя файла модели
    $model_name = $controller_name.'_model';
    $model_file = "app/models/".$model_name.'.php';
    //Проверяем наличие файла модели и завершаем работу в случае его
отсутствия
    if(file_exists($model_file)) {
      include $model_file;
    }  else {
      die "ОШИБКА! Файл модели $model_file не найден";
    }
    //Создаем экземпляр модели
    $model_class_name = ucfirst($controller_name).'Model';
    $model = new $model_class_name;
    //Присваиваем экземпляр модели соответствующему полю контроллера
    $controller->model = $model;

    //Вызываем экшн конроллера
    if(method_exists($controller, $action_name)) {
      $controller->$action_name();
    } else {
      die "ОШИБКА! Отсутствует метод $action_name контроллера $control-
ler_class_name";
    }
  }
}
```

В базовом классе контроллера достаточно определить поля для хранения экземпляров модели и представления и создать экземпляра представления в конструкторе:

```
<?php
class Controller
 {
  public $model;
  public $view;

  function __construct()
  {
    $this->view = new View();
  }
```

При этом в базовом классе представления метод render может влючать основной шаблон(layout) приложения:

```
<?php
class View
{
  function render($content_view, $title, $model = NULL, $layout = 'layout.php')
  {
    include 'app/views/'.$layout;
  }
}
```

Для того чтобы внутри основного шаблона заголовок страницы опредлялся параметром $title и отобразилось содержимое необходимого представления, передаваемого с использованием параметра $content_view, достаточно выполнить вывод значения переменной $title(в данном случае используется сокращенная форма оператора вывода ‘=’ сразу после открывающего тега php) и include для включения файла представления внутри шаблона:

```
<!DOCTYPE html>
<html>
  <head>
    <title><?=$title?></title>
    <script type="text/javascript" src="/public/assets/js/lib.js"></script>
          …
  </head>
  <body>
    <header>
      <h1>Добро пожаловать на мой персональный сайт</h1>
      <nav>
          …
      </nav>
    </header>
    <?
```

```
    include 'app/views/'.$content_view;
  ?>
   <footer>
       …
   </footer>
  </body>
 </html>
```

При этом внутри представленния можно будет использовать переменную *$model,* если она была передана при вызове render. Например, для страницы “Тест по дисциплине”:

```
<section class='testic'>
<? if ($model && count($model->validator->Errors)==0) { ?>
  <h2>Тест пройден успешно!!!</h2>
<? } else { ?>
…//отбражение сообщения об ошибках или вывод формы для заполнения
<?}  ?>
```

Базовый класс модели может создавать экземпляр валидатора в конструкторе, а также определить метод validate, вызывающий соответствующий метод валидатора:

```
<?php
require 'app/models/validators/form_validation.php';
class Model
{
  public $validator;

  function __construct()
  {
    $this->validator = new FormValidation();
  }
  function validate($post_data)
  {
    $this->validator->validate($post_data);
  }
}
```

При необходимости класс модели может переопределить валидатор, созданный в базовом классе, например:

```
<?php
class TestModel extends Model
{
  function __construct()
  {
    parent::__construct();
    $this->validator = new ResultVerification();
```

```
    $this->validator->SetRule('fio', 'IsNotEmpty');
    …
  }
}
```

1.3 Варианты заданий

Работа выполняется на основе персонального сайта, полученого в ходе выполнения лабораторных работ №1-4. Необходимо выполнить все рекомендации из п.1.4.

1.4 Порядок выполнения работы

1. В соответствии с рекомендациями, представленными в п. 1.2.8 реструктурировать код пресонального сайта на основе шаблона MVC.

Внимание! *Запрещается использовать готовые решения для реализации MVC архитектуры***.**

2. Для страницы «Фотоальбом» реализовать хранение данных фото(подписей и имен файлов) в соответствующих константах модели Photo. Вывод таблицы, содержащей фото, реализовать в представлении с использованием операторов циклов.

3. Для страницы «Мои интересы» реализовать хранение данных (название категорий, интересов, описание интересов) в соответствующих константах модели Interest. Вывод меню ссылок и интересов реализовать в представлении с использованием операторов циклов.

4. Реализовать класс FormValidation, выполняющий валидацию данных форм, передаваемых на сторону сервера. Рекомендуемая структура класса:

- Rules – поле(массив), содержащее набор правил для проверки валидности данных;
- Errors – поле(массив), содержащее тексты ошибок возникших при проверке валидности данных;
- isNotEmpty(data) – метод проверки является ли значение data не пустым – возвращает сообщение об ошибке, если таковая имеется;
- isInteger(data) – метод проверки является ли значение data строковым представлением целого числа – возвращает сообщение об ошибке, если таковая имеется;
- isLess(data, value) – метод проверки является ли значение data строковым представлением целого числа и не меньшим, чем value – возвращает сообщение об ошибке, если таковая имеется;
- isGreater(data, value) – метод проверки является ли значение data строковым представлением целого числа и не большим, чем value – возвращает сообщение об ошибке, если таковая имеется;
- isEmail(data) – метод проверки является ли значение data строковым представлением email – возвращает сообщение об ошибке, если таковая имеется;
- SetRule(field_name, validator_name) – метод, добавляющий в массив Rules проверку для поля field_name типа validator_name;
- Validate(post_array) – метод выполняющий проверку элементов в массиве post_array, в соответствии с правилами Rules и сохраняющий сообщения об ошибках в поле Errors;

- ShowErrors() – метод, выводящий все сообщения об ошибках из поля Errors в формате HTML.

5. С использованием разработанного класса реализовать валидацию форм «Контакт» и «Тест по дисциплине “…”».

6. Реализовать дочерний класс CustomFormValidation от класса FormValidation, дополнив его возможностью выполнения специализированной проверки формы «Тест по дисциплине» на стороне сервера.

7. Реализовать дочерний класс ResultsVerification от класса Custom-FormValidation, дополнив его возможностью проверки правильности ответов, введенных пользователем на странице "Тест по дисциплине" (реализовать проверку правильности для вопросов с элементами ввода типа RadioButton, ComboBox или однострочный текст) и вывода результатов проверки пользователю.

1.5 Содержание отчета

Цель работы, порядок выполнения работы, тексты переработанных HTML-документов и разработанных PHP-скриптов, внешний вид переработанных Web-страниц и результаты работы PHP-скриптов, выводы по результатам работы.

1.6 Контрольные вопросы

1.6.1. Каковы основные области применения PHP?
1.6.2. Как размещается PHP код в HTML-странице?
1.6.3. Расскажите о переменных и о типах данных PHP?
1.6.4. В чем отличие строковых констант в одинарных и двойных кавычках в PHP?
1.6.5. Как определить массив в PHP?
1.6.6. Приведите синтаксис условного оператора PHP?
1.6.7. Какие операторы циклов определены в PHP?
1.6.8. Для чего могут быть использованы операторы включения в PHP и в чем их отличие?
1.6.9. Каков синтаксис определения пользовательских функций в PHP?
1.6.10. Какие типы аргументов функций поддерживаются в PHP?
1.6.11. Как реализуются функции с переменным числом аргументов в PHP?
1.6.12. Как использовать глобальные переменные в PHP-функциях?
1.6.13. Приведите пример использования статических переменных в PHP функциях?
1.6.14. Как функции PHP возвращают результат своей работы?
1.6.15. Как в PHP получать данные форм HTML?
1.6.16. Каким образом в PHP обрабатываются строковые данные?
1.6.17. Приведите функции, с помощью которых возможно провести поиск подстроки в строке?

1.6.18. Приведите функции с помощью которых возможно разбиение строк на подстроки?
1.6.19. Как происходит присвоение объектов в PHP 5?
1.6.20. Какие спецификаторы доступа определены свойствам и методам класса в PHP 5?
1.6.21. Каким образом определить конструктор класса в PHP 5?
1.6.22. Как в PHP 5 реализовать наследование?
1.6.23. Как в PHP 5 можно обратиться к методу родительского класса?
1.6.24. Для чего в PHP 5 используется зарезервированное слово abstract?
1.6.25. Что такое статические свойства и методы класса?
1.6.26. Для чего используются финальные методы класса?
1.6.27. Приведите пример перегрузки доступа к свойствам класса?
1.6.28. Как в PHP 5 преобразовать объект в строку?
1.6.29. Что такое шаблон проектирования MVC? Из каких частей он стоит?
1.6.30. Что такое ***модель***, за что она отвечает?
1.6.31. Что такое ***представление***, что делает и за что отвечат?
1.6.32. Что делает ***контроллер***? Каковы его основные задачи?

2 ЛАБОРАТОРНАЯ РАБОТА №9

«Исследование возможностей хранения данных на стороне сервера. Работа с файлами. Работа с реляционными СУБД»

2.1 Цель работы

Исследовать возможности хранения данных на стороне сервера, изучить работу с файлами и СУБД MySQL из PHP, приобрести практические навыки организации хранения данных на стороне сервера в файлах, в базах данных MySQL, а также постраничного вывода данных.

2.2 Краткие теоретические сведения

2.2.1 Хранение данных на стороне сервера. Работа с файлами в PHP

2.2.1.1 Открытие файла

Функция *fopen()* используется для открытия файла. Синтаксис этой функции:

resource fopen (имя_файла, тип_доступа [, use_include_path])

В результате работы эта функция возвращает указатель (типа ресурс) на открытый ею файл. В качестве параметров этой функции передаются: имя файла, который нужно открыть; тип доступа к файлу (определяется тем, что мы собираемся делать с ним); параметр, определяющий, искать ли указанный файл в include_path.

Параметр *имя_файла* должен быть строкой, содержащей правильное локальное имя файла или URL-адрес файла в сети. Если имя файла начинается с указания протокола доступа (например, http://... или ftp://...), то интерпретатор считает это имя адресом URL и ищет обработчик указанного в URL протокола. Если обработчик найден, то PHP проверяет, разрешено ли работать с объектами URL как с обычными файлами (директива allow_url_fopen в настройках PHP). Если allow_url_fopen=off, то функция fopen вызывает ошибку и генерируется предупреждение. Если имя файла не начинается с протокола, то считается, что указано имя локального файла. Чтобы открыть локальный файл, нужно, чтобы PHP имел соответствующие права доступа к этому файлу.

Параметр *use_include_path*, установленный в значение 1 или TRUE, заставляет интерпретатор искать указанный в fopen() файл в include_path (include_path – это директива из файла настроек PHP, задающая список директорий, в которых могут находиться файлы для включения). Кроме функции fopen() она используется функциями include() и require().

Параметр *тип_доступа* может принимать одно из следующих значений (см. таб. 2.1).

Таблица 2.1 – Значения, принимаемые параметром тип_доступа

Тип доступа	Описание
r	Открывает файл только для чтения; устанавливает указатель позиции в файле на начало файла.
r+	Открывает файл для чтения и записи; устанавливает указатель файла на его начало.
w	Открывает файл только для записи; устанавливает указатель файла на его начало и усекает файл до нулевой длины. Если файл не существует, то пытается создать его.
w+	Открывает файл для записи и для чтения; устанавливает указатель файла на его начало и усекает файл до нулевой длины. Если файл не существует, то пытается создать его.
a	Открывает файл только для записи; устанавливает указатель файла в его конец. Если файл не существует, то пытается создать его.
a+	Открывает файл для записи и для чтения; устанавливает указатель файла в его конец. Если файл не существует, то пытается создать его.
x	Создает и открывает файл только для записи; помещает указатель файла на его начало. Если файл уже существует, то fopen() возвращает false и генерируется предупреждение. Если файл не существует, то делается попытка создать его.
x+	Создает и открывает файл для записи и для чтения; помещает указатель файла на его начало. Если файл уже существует, то fopen() возвращает false и генерируется предупреждение. Если файл не существует, то делается попытка создать его.

Таким образом, чтобы создать файл, можно открыть несуществующий файл на запись:

```
// открывает на запись файл my_file.html,если он существует,
// или создает пустой файл с таким именем, если его еще нет
$h = fopen("my_file.html","w");

// открывает на запись и чтение или создает файл
// another_file.txt в директории dir
$h = fopen("dir/another_file.txt","w+");

// открывает на чтение файл, находящийся по указанному адресу
$h = fopen( "http://www.server.ru/dir/file.php","r");
```

Что происходит, если открыть или создать файл с помощью fopen не удается? В этом случае PHP генерирует предупреждение, а функция fopen

возвращает как результат своей работы значение false. Такого рода предупреждения можно «подавить» (запретить) с помощью символа @.

Например, такая команда не выведет предупреждения, даже если открыть файл не удалось:

```
$h = @fopen(“dir/another_file.txt”,”w+”);
```

Таким образом, функция fopen() позволяет создать только лишь пустой файл и сделать его доступным для записи.

Однако при программировании на PHP считается «дурным тоном» подавлять ошибки, т.к. отслеживание подобных участков кода является довольно проблематичным. В связи с этим, для исключения подобных ситуаций рекомендуется перед вызовом fopen проверять, что файл/папка доступны для записи (используя метод is_writable) или что файл существует, если он открывается только для чтения (используя метод is_file).

2.2.1.2 Закрытие соединения с файлом

После выполнения необходимых действий с файлом, будь то чтение или запись данных или что-либо другое, соединение, установленное с этим файлом функцией fopen(), нужно закрыть. Для этого используют функцию fclose(). Синтаксис у нее следующий:

```
fclose (указатель на файл)
```

Эта функция возвращает TRUE, если соединение успешно закрыто, и FALSE – в противном случае. Параметр этой функции должен указывать на файл, успешно открытый, например, с помощью функции fopen().

```
$h = fopen("my_file.html","w");
fclose($h);
```

2.2.1.3 Запись данных в файл

Для того чтобы записать данные в файл, доступ к которому открыт функцией fopen(), можно использовать функцию fwrite(). Синтаксис у нее следующий:

```
int fwrite ( указатель на файл,  строка [, длина])
```

Эта функция записывает содержимое строки строка в файл, на который указывает указатель на файл. Если указан дополнительный аргумент длина, то запись заканчивается после того, как записано количество символов, равное значению этого аргумента, или тогда, когда будет достигнут конец строки. В результате своей работы функция fwrite() возвращает число записанных байтов или false, в случае ошибки.

```
$h      = fopen("my_file.html","w");
$text  = "Этот текст запишем в файл.";
if (fwrite($h, $text)) {
  echo "Запись прошла успешно";
}
else {
  echo "Произошла ошибка при записи данных";
}
```

```
fclose($h);
```

В результате работы этого скрипта в браузере мы увидим сообщение о том, что запись прошла успешно, а в файле my_file.html появится строка “Этот текст запишем в файл.”. Если бы этот файл существовал до того, как мы выполнили этот скрипт, все находящиеся в нем данные были бы удалены.

2.2.1.4 Чтение данных из файла

Чтобы прочитать данные из файла, нужно воспользоваться одной из специальных функций: file, readfile, file_get_contents, fread, fgets и т.п.

Функция fread

Эта функция осуществляет чтение данных из файла. Ее можно использовать и для чтения данных из бинарных файлов, не опасаясь их повреждения. Синтаксис fread() следующий:

string fread (указатель на файл, длина)

При вызове этой функции происходит чтение данных длины (в байтах), определенной параметром длина, из файла, на который указывает указатель на файл. Параметр указатель на файл должен быть реально существующей переменной типа ресурс, содержащей в себе связь с файлом, открытую, например, с помощью функции fopen(). Чтение данных происходит до тех пор, пока не встретится конец файла или пока не будет прочитано указанное параметром длина число байтов.

В результате работы функция fread() возвращает строку со считанной из файла информацией.

Параметр длина в этой функции – обязательный. Следовательно, если мы хотим считать весь файл в строку, нужно знать его длину. PHP может самостоятельно вычислить длину указанного файла. Для этого нужно воспользоваться функцией filesize(имя файла). В случае ошибки эта функция вернет false. К сожалению, ее можно использовать только для получения размера локальных файлов.

В примере ниже прочитаем содержимое файла my_file.html:

```
// отрываем файл на запись и чтение
$h = fopen("my_file.html","r+");

// считываем содержимое файла в строку
$content = fread($h, filesize(“my_file.html”));

// закрываем соединение с файлом
fclose($h);

// выводим содержимое файла на экран браузера
echo $content;
```

Для того чтобы считать содержимое бинарного файла, например изображения, рекомендуется открывать файл с помощью флага rb или ему подобных, содержащих символ b в конце.

Функция filesize() кэширует результаты своей работы. Если изменить содержимое файла my_file.html и снова запустить приведенный выше скрипт, то результат его работы не изменится. Более того, если запустить скрипт, считывающий данные из этого файла с помощью другой функции (например, fgetss), то результат может оказаться таким, как если бы файл не изменился. Чтобы этого избежать, нужно очистить статический кэш, добавив в код программы команду clearstatcache();

Функция fgets

С помощью функции fgets() можно считать из файла строку текста. Синтаксис этой функции практически такой же, как и у fread(), за исключением того, что длину считываемой строки указывать необязательно:

string fgets (указатель на файл [, длина])

В результате работы функция fgets() возвращает строку длиной (длина–1) байт из файла, на который указывает указатель на файл. Чтение заканчивается, если прочитано (длина–1) символов и встретился символ перевода строки или конец файла. Если параметр длина не задан, считывается строка целиком. В случае ошибки функция fgets() возвращает false.

```
$h = fopen("my_file.html","r+");

// считает первый символ из первой строки файла my_file.html
$content = fgets($h,2);

fclose($h);
echo $content;
```

Обе функции, fread() и fgets(), прекращают считывание данных из файла, если встречают конец файла. В PHP есть специальная функция, проверяющая, смотрит ли указатель позиции файла на конец файла. Это булева функция feof(), в качестве параметра которой передается указатель на соединение с файлом.

Например, вот так можно считать все строки файла my_file.html:

```
$h = fopen("my_file.html","r");
while (!feof ($h)) {
   $content = fgets($h);
   echo $content,"<br>";
}
fclose($h);
```

Функция fgetss

Существует разновидность функции fgets() – функция fgetss(). Она тоже позволяет считывать строку из указанного файла, но при этом удаляет из него все встретившиеся html-теги, за исключением некоторых указанных. Синтаксис fgetss():

string fgetss(указатель на файл, длина [, допустимые теги])

Следует обратить внимание, что здесь аргумент длина обязательный. Пусть имеется файл my_file.html следующего содержания:

```
<h1>Lorem ipsum dolor sit amet</h1>
<b>consectetur adipiscing elit</b>
Integer et tortor mollis, porta ex a, <i>pellentesque lacus</i>.
```

Выведем на экран все строки файла my_file.html, удалив из них все теги, кроме <b> и <i>:

```
$h = fopen("my_file.html","r");
while (!feof ($h)) {
  $content = fgetss($h,1024,'<b><i>');
  echo $content,"<br>";
}
fclose($h);
```

В результате работы этого скрипта будет выведен отформатированный HTML с жирным текстом и курсивом.

Функция fgetc

Естественно, если можно считывать информацию из файла построчно, то можно считывать ее и посимвольно. Для этого предназначена функция fgetc(). Синтаксис:

string fgetc (указатель на файл)

Эта функция возвращает символ из файла, на который ссылается указатель на файл, и значение, вычисляемое как FALSE, если встречен конец строки. Вот так, например, можно считать файл по одному символу:

```
$h = fopen("my_file.html","r");
while (!feof ($h)) {
   $content = fgetc($h);
   echo $content,"<br>";
}
fclose($h);
```

На самом деле, для того чтобы прочитать содержимое файла, открывать соединение с ним посредством функции fopen() совсем не обязательно. В PHP есть функции, которые позволяют делать это, используя лишь имя файла. Это функции readfile( ), file( ) и file_get_contents( ). Рассмотрим каждую их них подробнее.

Функция readfile

Синтаксис:

int readfile (имя_файла [, use_include_path])

Функция readfile() считывает файл, имя которого передано ей в качестве параметра имя_файла, и выводит его содержимое на экран. Если дополнительный аргумент use_include_path имеет значение TRUE, то поиск файла с заданным именем производится и по директориям, входящим в include_path. В программу эта функция возвращает число считанных байтов (символов) файла, а в случае ошибки – FALSE.

Следующий скрипт выведет на экран содержимое файла my_file1.html и размер этого файла, если он существует. В противном случае выведется наше сообщение об ошибке – строка "Error in readfile":

```
// выводит на экран содержимое файла и записывает
// его размер в переменную $n
$n = readfile ("my_file1.html");

// если функция readfile() выполнилась с ошибкой, то $n=false и
// выводим сообщение об ошибке.Если ошибки не было, то
// выводим число считанных символов
if (!$n) {
   echo "Error in readfile";
} else  {
   echo $n;
}
```

С помощью функции readfile() можно читать содержимое удаленных файлов, указывая их URL-адрес в качестве имени файла, если эта опция не отключена в настройках сервера.

Сразу же выводить содержимое файла на экран не всегда удобно. Порой нужно записать информацию из файла в переменную, чтобы в дальнейшем произвести с ней какие-либо действия. Для этого можно использовать функцию file() или file_get_contents().

Функция file

Функция file() предназначена для считывания информации из файла в переменную типа массив. Синтаксис у нее такой же, как и у функции readfile(), за исключением того, что в результате работы она возвращает массив:

array file(имя_файла [, use_include_path])

Каждый элемент массива *array* является строкой в файле, информацию из которого мы считываем (его имя задано аргументом имя_файла). Символ новой строки тоже включается в каждый из элементов массива. В случае ошибки функция file(), как и все уже рассмотренные, возвращает false. Дополнительный аргумент use_include_path опять же определяет, искать или нет данный файл в

директориях include_path. Открывать удаленные файлы с помощью этой функции тоже можно, если не запрещено настройками.

2.2.1.5 Загрузка файлов на сервер

Ниже представлена HTML-форма, позволяющая выполнить загрузку файла на сервер:

```
<html>
 <head>
   <title> Загрузка файлов на сервер </title>
 </head>
<body>
 <h2> Форма для загрузки файлов </h2>
 <form action="upload.php" method="post" enctype="multipart/form-data">
  <input type="file" name="messages" <br>
  <input type="submit" value="Загрузить"
</form>
</body>
</html>
```

Форма содержит поле типа *file* для загрузки файла и кнопку для отправки параметров формы обработчику, расположенному в файле upload.php. Атрибут *enctype* формы определяет вид кодировки, которую браузер применяет к параметрам формы. Для того чтобы отправка файлов на сервер действовала, атрибуту *entype* необходимо присвоить значение *multipart/formdata* (по умолчанию этот атрибут имеет значение application/x-www-formurlencoded).

После того, как на сервере получен HTTP-запрос POST, содержимое загруженного файла записывается во временный файл, который создается в каталоге сервера, заданном по умолчанию для временных файлов.

В PHP характеристики загруженного файла доступны через двумерный суперглобальный массив $_FILES. При этом переменная со значениями этого массива может иметь следующий вид:

- $_FILES["filename"]["name"] - содержит исходное имя файла на клиентской машине;
- $_FILES["filename"]["size"] - содержит размер загруженного файла в байтах;
- $_FILES["filename"]["type"] - содержит MIME-тип файла;
- $_FILES["filename"]["tmp_file"] - содержит имя временного файла, в который сохраняется загруженный файл.

В качестве имени *filename* должно выступать значение параметра *name* поля *file* HTML-формы.

В следующем примере файл, отправленный из формы, перемещается в каталог messages:

```
<html>
<head>
<title> Результат загрузки файла </title>
</head>
<body>
```

```
<?php
  $source = $_FILES["messages "]["tmp_name"];
  $dest     = "messages/".$_FILES["messages "]["name"];
  If (move_uploaded_file($source, $dest)) {
    echo("Файл успешно загружен");
  } else {
    echo("Ошибка загрузки файла");
  }
?>
</body>
</html>
```

После выполнения данного фрагмента, загруженный файл будет помещен в каталог messages, а браузер выдаст строку: «Файл успешно загружен».

2.2.2 Работа PHP с базой данных MySQL

Одним из возможных способов работы с реляционными базами данных в PHP является использование интерфейса PDO - PHP Data Objects (PHP объекты данных).

Преимущество PDO заключается в обеспечении слоя абстракции при доступе к данным. Это значит, что вне зависимости от того, какая конкретная СУБД используется (например, MySQL или PostgreSQL), вы можете пользоваться одними и теми же функциями для взаимодействия с ней(выполнения запросов, модификации данных и тп).

Расширение PDO внедрено в PHP 5.1, но также доступно в 5.0 в виде PECL расширения; PDO использует новый OO функционал из ядра PHP 5, соответственно он не будет работать с ранними версиями PHP.

2.2.2.1 Соединение с сервером

Прежде чем начать работать с базой данных, необходимо установить с ней сетевое соединение, а также провести авторизацию пользователя. Для этого необходимо создать объект PDO. Конструктор объекта выглядит следующим образом:

```
public PDO::__construct (
  string $dsn [, string $username [, string $password [, array $options ]]]
)
```

Здесь:

dsn - имя источника данных или DSN, содержащее информацию, необходимую для подключения к базе данных (в общем, DSN состоит из имени драйвера PDO, за которым следует двоеточие и специфический синтаксис подключения драйвера PDO);

username - имя пользователя для строки DSN;

password - пароль для строки DSN;

options - массив специфичных для драйвера настроек подключения вида ключ=>значение.

Ниже приведен пример вызова данного конструктора для СУБД MySQL:

```
// Подключение к MySQL базе с именем test_db
$dsn      = 'mysql:dbname=test_db;host=127.0.0.1';
$username = 'username';
$password  = 'password';
try {
   $pdo = new PDO(
      $dsn, $username, $password,
     [PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION]
   );
} catch (PDOException $e) {
   echo 'Подключение не удалось: ' . $e->getMessage();
}
```

В данном случае $pdo – это объект, рализующий подключение к СУБД MySQL, расположенной на сервере 127.0.0.1. Вся дальнейшая работа ведется именно с этим объектом. Все другие функции, вызываемые у данного объекта, будут однозначно определять выбранную базу данных.

2.2.2.2 Разрыв соединения с сервером

Соединение остается активным на протяжении всего времени жизни объекта PDO. Чтобы закрыть соединение, необходимо уничтожить объект путем удаления всех ссылок на него (этого можно добиться, присваивая NULL всем переменным, указывающим на объект):

```
$pdo = null;
```

PHP автоматически закроет соединение по окончании работы скрипта, поэтому явно закрывать соединение с базой данных **не обязательно**.

2.2.2.3 Обработка ошибок

Если в процессе работы с MySQL возникают ошибки (например, в запросе не сбалансированы скобки или же не хватает параметров), то по умолчанию функции, выполняющие запрос, будут возвращать false. Однако для удобства отладки и перехвата ошибок рекомендуется включать режим генерации исключений в конструкторе PDO:

```
[PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION]
```

Если данный режим включен, то при возникновении ошибки будет сгенерировано исключение PDOException, в котором будет содержаться детальная информация об ошибке.

2.2.2.4 Выполнение запросов к базе данных

Для отправки SQL-запроса серверу СУБД используется функция query:

```
public PDOStatement PDO::query (
  string $statement
)
public PDOStatement PDO::query (
  string $statement , int $PDO::FETCH_COLUMN , int $colno
)
public PDOStatement PDO::query (
  string $statement , int $PDO::FETCH_CLASS , string $class , array $ctorargs
)
public PDOStatement PDO::query (
 string $statement , int $PDO::FETCH_INTO , object $object
)
```

Данная функция выполняет SQL запрос без подготовки и возвращает результирующий набор (если есть) в виде объекта PDOStatement. Ниже приведен пример выполнения простейшего SQL запроса с помощью функции query:

```
$sql = 'SELECT name, color FROM cars ORDER BY name';
foreach ($pdo->query($sql) as $car) {
  print $car['name'] . "\t";
  print $car['color'] . "<br/>";
}
```

В данном примере PDO возвращает объект-итератор, который может быть использован в цикле foreach для последовательного доступа к возвращенным записям.

В случае если необхдмосразу получить массив всех возвращенных записей, можно использовать функцию iterator_to_array. Воспользуемся данной функцией, чтобы написать метод findAllCars, который вернет все автомобили из таблицы cars в виде массива:

```
function findAllCars($pdo) {
  $sql = 'SELECT * FROM cars ORDER BY name';
  return iterator_to_array($pdo->query($sql));
}
print_r(findAllCars($pdo));
```

2.2.3 Организация постраничного вывода данных из БД

Довольно часто при работе с базами данных возникает необходимость в отображении большого объема однотипной информации, при этом, для удобства восприятия, её следует разбивать на части, т.е. реализовать постраничный просмотр этой информации.

Для решения задачи частичной выборки записей из таблицы базы данных в синтаксисе оператора SELECT языка SQL в СУБД MySQL предусмотрен оператор LIMIT:

```
LIMIT [offset,] rows
```

Как видно, LIMIT принимает один или два числовых аргумента. Эти аргументы должны быть целочисленными константами. Если заданы два

аргумента, то первый указывает на индекс первой возвращаемой строки, а второй задает максимальное количество возвращаемых строк. При этом индекс первой строки 0:

```
# возвращает строки 6-15
SELECT * FROM cars LIMIT 5,10;
```

Если задан один аргумент, то он показывает максимальное количество возвращаемых строк (*LIMIT n* эквивалентно *LIMIT 0,n*):

```
# возвращает первых 5 строк
SELECT * FROM cars LIMIT 5;
```

При реализации постраничного вывода информации из БД, кроме частичной выборки, необходимо также решить задачу организации строки ссылок на страницы, для обеспечения пользователя возможностью «перелистывания» страниц. При формировании строки ссылок на страницы необходимо знать общее количество записей. Для получения общего количества отображаемых записей возможно использование дополнительного SQL-запроса, вычисляющего и возвращающего общее количество записей. Например:

```
SELECT COUNT(*) FROM cars
```

Однако при использовании СУБД MySQL выполнение дополнительного запроса можно избежать благодаря появившемуся в MySQL 4.0.0 параметру *SQL_CALC_FOUND_ROWS* оператора SELECT. Например, выполняя запрос:

```
SELECT SQL_CALC_FOUND_ROWS * FROM cars LIMIT 40, 20
```

MySQL подсчитает полное число строк, подходящих под условие запроса ***без учета ограничений, вызванных оператором LIMIT***, и сохранит это число в памяти. Сразу после выполнения запроса на выборку для получения количества записей нужно выполнить SELECT-запрос:

```
SELECT FOUND_ROWS ()
```

В результате MySQL без дополнительных вычислений вернет сохраненный в памяти параметр – полное количество строк в результате запроса.

Таким образом, для формирования постраничного вывода записей таблицы ***table*** может быть использован примерно следующий фрагмент кода:

```
// количество записей, выводимых на странице
$per_page=10;

// получаем номер страницы
$page= (int)(isset($_GET['page'])  ? ($_GET['page']-1) : 0);

// вычисляем первый операнд для LIMIT
$start=abs($page*$per_page);
```

```
// выполняем запрос и выводим записи
$query = "SELECT * FROM cars ORDER BY name LIMIT $start, $per_page";
foreach ($pdo->query($query) as $i => $car) {
  echo ($i+$start).". ".$car['name']."<br>\n";
}

// выводим ссылки на страницы:
$query      = "SELECT count(*) FROM cars";
$total_rows = $pdo->query($query)->fetchColumn();

// Определяем общее количество страниц
$num_pages = ceil($total_rows/$per_page);

for($i=1;$i <= $num_pages; $i++) {
  // текущую страницу выводим без ссылки
  if ($i-1 == $page) {
    echo $i." ";
  } else {
    echo '<a href="'.$_SERVER['PHP_SELF'].'?page='.$i.'">'.$i."</a> ";
  }
}
```

2.2.4 Подготавливаемые запросы

PDO поддерживает *подготавливаемые запросы*. Подготавливаемые (или параметризованные) запросы используются для повышения эффективности, когда один запрос выполняется многократно с различными параметрами.

Выполнение работы с БД в данном режиме можно разбить на следующие этапы:

- подготовка;
- выполнение;
- повторное выполнение запроса (N раз).

На этапе подготовки на сервер посылается шаблон запроса. Сервер выполняет синтаксическую проверку этого шаблона, строит план выполнения запроса и выделяет под него ресурсы.

В примере ниже выражение INSERT один раз подготавливается, а затем многократно выполняется:

```
// подготавливаемый запрос, первая стадия: подготовка
$s = $pdo->prepare('INSERT INTO cars(name, color) VALUES (:name, :color)');

// подготавливаемый запрос, вторая стадия: привязка и выполнение
$name = 'audi';
$color = 'audi';
$s->bindParam(':name', $name);
$s->bindParam(':color', $color);
$s->execute();
```

```
// подготавливаемый запрос: повторные выполнения,
// на сервер передаются только значения переменных
for ($i = 1; $i < 5; $i++) {
    $name  = $name.$i;
    $color   = $color.$i;
    $s->bindParam(':name', $name);
    $s->bindParam(':color', $color);
    $s->execute();
}

// обычный запрос
$sql = 'SELECT * FROM cars ORDER BY name';
print_r(iterator_to_array($pdo->query($sql)));
```

Следует обратить внимание на методы bindParam, которые подставляют значения переменных в SQL запрос. Это является самым безопасным способом для работы с базой данных. Дело в том, что если вставлять параметры в запрос напрямую, например как это сделано в коде ниже:

```
$q  = "SELECT * FROM cars ORDER BY field LIMIT ".$_GET['page'];
```

то возникает риск возникновения атаки на сервер типа SQL-инъекция. Злоумышленник может передать в переменную $_GET[‘page’] специальные символы (кавычки, % и т.п.) и выполнить непредусмотренный системой SQL запрос. В свою очередь метод bind-Param экранирует все служебные символы СУБД, предотвращая данный вид атаки.

Для извлечения результатов подготовленных запросов используются методы:

public mixed PDOStatement::fetch ([int $fetch_style [, int $cursor_orientation = PDO::FETCH_ORI_NEXT [, int $cursor_offset = 0]]])

public array PDOStatement::fetchAll ([int $fetch_style [, mixed $fetch_argument [, array $ctor_args = array()]]])

Метод fetch возвращает массив, содержащий одну строку результирующего набора. Метод fetchAll возвращает все строки результирующего набора. Параметр fetch_style определяет, в каком виде следующая строка будет возвращена в вызывающий метод. Это может быть одна из констант PDO::FETCH_*. По умолчанию PDO::ATTR_DEFAULT_FETCH_MODE (что равносильно PDO::FETCH_BOTH). Более подробно с данными константами можно ознакомиться в источнике [5], в разделе документации, посвященном PDO.

2.2.5 Рекомендации по реализации паттерна ActiveRecord

ActiveRecord (AR) – шаблон проектирования приложений, описанный Мартином Фаулером в своей книге «Шаблоны корпоративных приложений» [17]. AR это способ доступа к данным реляционных баз данных в объектно-ориентированном программировании. При использовании AR программист не взаимодействует с БД напрямую, а использует объекты, которые реализуют доступ к БД и логику обращения с данными.

В соответствии с AR для таблицы БД создаётся специальный класс, являющийся отражением (представлением) таблицы таким образом, что:

- каждый экземпляр данного класса соответствует одной записи таблицы;
- при создании нового экземпляра класса (и заполнении соответствующих полей) в таблицу добавляется новая запись;
- при чтении полей объекта считываются соответствующие значения записи таблицы баз данных;
- при изменении (удалении) какого-либо объекта изменяется (удаляется) соответствующая ему запись.

Реализация концепции AR существует во многих фреймфорках для различных языков программирования.

Например, если для таблицы cars с полями name и color, в соответствии с шаблоном Active Record может быть реализован класс Car, с помощью которого возможно:

```
$car          = new Car();
$car->name  = "audi";
$car->color   = “red”;
$car->save()
```

Код класса Car создаст новую запись в таблице cars с данными значениями, для чего должен будет выполнить примерно следующий SQL запрос:

```
INSERT INTO cars (name, color) VALUES (‘audi’, ‘red’);
```

Также AR реализовывает и остальные методы, необходимые для управления данными СУБД, например, delete (вызывает DELETE запрос), find (поиск записи по ID) и findAll (поиск коллекции записей по указанным критериям) и тп. Таким образом, код:

```
// ищет запись с id = 1
$car  = Car::find(1);

// обновляет поле color
$car->color = “blue”;
$car->save()

// уничтожает запись в БД
$car->delete();
```

должен выполнять следующие запросы:

*SELECT * FROM cars WHERE id = 1;*
UPDATE cars SET color = ‘blue’ WHERE id = 1;
DELETE FROM cars WHERE id = 1;

Рассмотрим фрагмент реализации базового класса BaseActiveRecord:

```
class BaseActiveRecord{

  public static $pdo;

  protected static $tablename;
  protected static $dbfields = array();

  public function __construct() {
    if (!static::$tablename){
      return ;
    }

    static::setupConnection();
    static::getFields();
  }

  public static function getFields() {
    $stmt = static::$pdo->query("SHOW FIELDS FROM ".static::$tablename);
    while ($row = $stmt->fetch()) {
      static::$dbfields[$row['Field']] = $row['Type'];
    }
  }

  public static function setupConnection() {
    if (!isset(static::$pdo)) {
      $eror = false;
      try {
        static::$pdo = new PDO("mysql:dbname=lrdb; host=localhost; char-
set=utf8", "username", "password");
      } catch (PDOException $ex) {
        die("Ошибка подключения к БД: $ex");
      }
    }
  }

  public static function find($id){
    $sql = "SELECT * FROM ".static::$tablename." WHERE id=$id";
    $stmt = static::$pdo->query($sql);

    $row = $stmt->fetch(PDO::FETCH_ASSOC);

    if (!$row) {
        return null;
    }
```

```
        $ar_obj = new static();
        foreach ($row as $key => $value) {
          $ar_obj ->$key = $value;
         }
        return $ar_obj;
      }

     public static function findAll() { … }

    public function save() { … }

    public function delete(){
        $sql = "DELETE FROM ".static::$tablename." WHERE ID=".$this->id;
        $stmt = static::$pdo->query($sql);
        if($stmt){
          return $stmt->fetch(PDO::FETCH_ASSOC);
        }else{
          print_r(static::$pdo->errorInfo());
        }
      }
  }
```

В примере выше в конструкторе AR объекта проверяется наличие имени связанной таблицы(*tablename*), выполняется соединение с БД, а так же с использованием метода *getFields* реализуется получение существующих в БД полей таблицы и их типов (для этого использован SQL запрос «*SHOW FIELDS FROM TABLENAME*»). Информация о типах данных полей может потребоваться, например, для конструирования запроса сохранения данных (в отличие от числовых типов данных, строковые значения необходимо заключать в кавычки):

```
        $fields_list = array();
        foreach (static::$dbfields as $field => $field_type) {
           $value = $this->$field;
           if (strpos($field_type,'int')===false) $value = "'$value'";
          $fields_list[] = "$field = $value";
         }
         $sql = "UPDATE ".static::$tablename." SET ".join(', ',$fiels_list)." WHERE
ID=".$this->id;
```

В статическом методе *find($id)* реализуется поиск записи в таблице по ее индентификатору и в случае если запись найдена возвращается экземпляр класса с полями, проинициализированными значениями записи.

В методе экземпляра *delete* выполняется удаление текущей записи с использованием ее идентификатора(*$this->id*).

Рассмотрим пример дочернего класса:

```
class BlogModel extends Model{
  protected static $tablename = 'blogs';
```

```
  public $name;
  public $text;
  public $img;
  public $data;

  function __construct()
  {
    parent::__construct();
    $this->validator = new ResultVerification();
    $this->validator->SetRule('name', 'IsNotEmpty');
    $this->validator->SetRule('text', 'IsNotEmpty');
  }

  function validate($post_data){
    $this->validator->Validate($post_data);
  }
}
```

Как видно из примера выше, в дочернем классе необходимо определить имя таблицы, а также используемые поля. Так как в конструкторе базового класса контроллера создается экземпляр соответсвующей модели(см ЛР №5), в контроллере можно использовать экземпляр класса модели. Ниже представлен пример реализации метода save контроллера BlogController, выполняющий для сохранения в БД полей новой записи блога вызов метода save базового класса BaseActiveRecord:

```
function save(){
    $this->model->validate($_POST);
    if (count($this->model->validator->Errors)==0){
      $this->model->name = $_POST['name'];
      $this->model->data = $_POST['data'];
      $this->model->text = $_POST['text'];
      $this->model->save();
    }
    $this->view->render('blog.php', 'Мой блог', $this->model);
  }
```

2.3 Варианты заданий

Формат постраничного вывода определяется из таблицы 2.2.

Таблица 2.2 – Варианты задания

№	Формат строки ссылок на страницы	Количество записей на странице
1	Страницы: 1 … 5 6 7 … 24	Постоянное, определяется константой
2	Записи: 1-10 11-20 21-30 31-40 41-54	Переменное, зависит от общего количества записей – максимальное количество страниц 5
3	Страницы: <-Предыдущая Следующая -> 1 2 …8… 19 20	Переменное, выбирается пользователем из выпадающего

		меню
4	Всего страниц: 31 Предыдущая 21 22 23 24 25 Следующая	Постоянное, определяется константой
5	Всего записей: 55 Страницы: 1 2 3 4 5 6 7 8 9 10	Переменное, зависит от общего количества записей – максимальное количество страниц 10
6	Записи: 1-10 11-20 21-30 … 91-92	Переменное, выбирается пользователем из выпадающего меню

2.4 Порядок выполнения работы

1. Разработать базовый класс BaseActiveRecord для работы с базой данных, который реализует паттерн ActiveRecord (данный класс разместить в папке /my_site/app/core). Данный класс должен реализовать следующие функции:

Таблица 2.3 – Методы класса BaseActiveRecord

Метод	Возвращаемое значение	Описание
save()	Текущий AR (т.е. this)	Создает или обновляет соответствующую запись в БД. Создание(запрос INSERT) происходит только для новых записей. Если у записи уже есть $id, то происходит ее изменение (выполнение SQL запроса ‘UPDATE’).
delete()	true	Удаляет запись из БД к которой привязан данный ActiveRecord объект.
find($id)	AR объект	Ищет в таблице, привязанной к текущему AR строку с id = $id и возвращает объект текущего класса, у которого все поля заполнены значениями из соответствующих полей таблицы БД.
findAll()	Массив AR объектов	Возвращает список всех строк таблицы. Каждая строка – это AR объект.

2. Для всех моделей, которые будут использоваться при выполнении данной лабораторной работы, создать классы, наследующие BaseActiveRecord. Для каждого из классов определить все поля и названия таблиц (данные классы необходимо разместить в папке /my_site/app/models).

3. Создать новую страницу "Гостевая книга". Страница должна содержать форму ввода (Фамилия, Имя, Отчество, E –mail, Текст отзыва), а также таблицу сообщений, оставленных пользователями. Сообщения в таблице должны располагаться в порядке убывания даты добавления сообщения. Для хранения сообщений пользователей использовать текстовый файл messages.inc, содержащий разделенные символом «;» данные: Дату сообщения, ФИО, E–mail и Текст отзыва (для получения текущей даты сервера возможно использовать PHP функцию *date('d.m.y')).*

4. Реализовать страницу "Загрузка сообщений гостевой книги", содержащую форму загрузки подготовленного заранее файла messages.inc на сервер.

5. Реализовать на странице "Тест по дисциплине" сохранение ответов пользователей и правильности ответов в разработанную таблицу базы данных MySQL, с возможностью просмотра сохраненных данных (дата, ФИО, ответы, верно/неверно).

6. Разработать страницу «Редактор Блога», позволяющую добавлять записи Блога. Страница должна содержать форму добавления записи Блога и список выдаваемых постранично записей отсортированных в порядке убывания даты. Форма добавления должна содержать поля ввода:

- Тема сообщения – поле ввода однострочного текста (заполнение обязательно);
- Изображение – поле ввода файла (заполнение не обязательно);
- Текст сообщения – поле ввода многострочного текста.

Данные хранить в разработанной таблице базы данных MySQL. Валидацию данных осуществлять с использованием класса FormValidation, разработанного при выполнении ЛР №8.

7. Разработать страницу «Мой Блог», содержащую упорядоченные в порядке убывания даты добавления, выдаваемые постранично данные:

- Дата и время сообщения;
- Тема сообщения;
- Изображение;
- Текст сообщения.

Данные извлекать из таблицы базы данных MySQL. Формат постраничного вывода определяется из таблицы 2.2 в соответствии с вариантом задания. Вариант задания определяется как (ПЦЗК mod 6)+1 – (остаток от деления последней цифры зачетной книжки на 6 плюс 1).

8. Реализовать возможность добавления записей на страницу «Мой Блог» из файла формата CSV, содержащего следующие поля: title, message, author, created_at. Например: "тема 1","сообщение 1","Vasiliy","2019-01-01 14:00". Для этого необходимо разработать страницу «Загрузка сообщений блога», содержащую форму загрузки файла формата CSV. Добавление записей из файла в БД осуществлять с использованием подготавливаемых запросов (см. п. 2.2.4). Валидацию данных осуществлять с использованием класса FormValidation, разработанного при выполнении ЛР №8.

2.5 Содержание отчета

Цель работы, порядок выполнения работы, тексты переработанных HTML-документов и разработанных PHP-скриптов, DDL запросов для создания таблиц базы данных, внешний вид переработанных Web-страниц

и результаты работы PHP-скриптов, UML диаграмма классов моделей и базового класса AR, выводы по результатам работы.

2.6 Контрольные вопросы

2.6.1. Расскажите о работе с файлами в PHP?
2.6.2. Какие функции могут быть использованы в PHP для чтения данных из файлов?
2.6.3. Какие функции могут быть использованы в PHP для записи данных в файл?
2.6.4. Как реализовать загрузку файла на сервер?
2.6.5. Расскажите о работе с базой данных MySQL в PHP?
2.6.6. Как в PHP выполняется подключение к базе данных с помощью PDO?
2.6.8. Какие функции PHP используются для выполнения запросов к базе данных MySQL?
2.6.9. Как обработать результаты выполнения запросов, которые возвращают множество записей?
2.6.10. Что такое подготавливаемые запросы и как их реализовать с использованием PDO?
2.6.11. Как организовать постраничный вывод данных MySQL в PHP?
2.6.12. Поясните суть паттерна ActiveRecord?
2.6.13. Что такое позднее статическое связывание?
2.6.14. Чем отличаются ключевые слова self и static в PHP?

3 ЛАБОРАТОРНАЯ РАБОТА №10
«Исследование механизма сессий в PHP»

3.1 Цель работы

Исследовать механизмы сохранения данных в процессе работы пользователя с сайтом, изучить возможности авторизации пользователей с использованием механизма сессий PHP, приобрести практические навыки использования переменных, сохраняющих свое значение при переходе от одной страницы сайта к другой.

3.2 Краткие теоретические сведения

3.2.1 Механизм сессий

Сессии – это механизм, который позволяет создавать и использовать переменные, сохраняющие свое значение при переходе от одной страницы сайта к другой в течение всего времени работы пользователя с сайтом.

Задача идентификации пользователя в течение сеанса (или сессии) работы с сайтом решается путем присвоения каждому пользователю уникального номера, так называемого идентификатора сессии (SID, Session IDentifier). Он генерируется PHP в тот момент, когда пользователь заходит на сайт, и уничтожается, когда пользователь уходит с сайта. Идентификатор сессии представляет собой строку из 32 символов (например, ac4f4a45bdc893434c95dcaffb1c1811). Этот идентификатор передается на сервер вместе с каждым запросом клиента и возвращается обратно вместе с ответом сервера.

Существует несколько способов передачи идентификатора сессии:

1. С помощью cookies. Cookies были созданы специально как метод однозначной идентификации клиентов и представляют собой расширение протокола HTTP. В этом случае идентификатор сессии сохраняется во временном файле на компьютере клиента, пославшего запрос. Многие пользователи отключают поддержку cookies на своем компьютере из-за проблем с безопасностью.

2. С помощью параметров командной строки. В этом случае идентификатор сессии автоматически встраивается во все запросы (URL), передаваемые серверу, и хранится на стороне сервера. Например: адрес http://is.sevntu.sebastopol.ua/test.php превращается в адрес: is.sevntu.sebastopol.ua/test.php?PHPSESSID=ac4f4a45bdc893434c95dcaffb1c1811.Этот способ передачи идентификатора используется автоматически, если у браузера, отправившего запрос, выключены cookies. Он достаточно надежен – передавать параметры в командной строке можно независимо от настроек клиента.

Кроме перечисленных вариантов передачи идентификатора сессии, существует еще несколько, но они используются гораздо реже.

3.2.2 Настройка сессий

Работа с сессиями в PHP поддерживается по умолчанию. Это значит, что устанавливать никаких дополнительных элементов не нужно. Однако полезными могут оказаться сведения о настройках PHP для работы с сессиями.

Настройки PHP, в том числе и для работы с сессиями, определяются в файле php.ini. Обратимся к этому файлу.

Как было отмечено ранее, идентификатор сессии (число, по которому можно уникально идентифицировать клиента, пославшего запрос) сохраняется на компьютере-сервере, или на компьютере-клиенте, или на обеих сторонах.

Параметр session.save_path в php.ini, определяет, где на сервере будут храниться данные сессии. Из-за него чаще всего возникают проблемы для Windows-серверов, потому что по умолчанию значение session.save_path установлено в /tmp. И если в корневой директории сервера такой папки нет, то при запуске сессий будет выдаваться ошибка.

Сервер может обрабатывать большое количество сессий одновременно, и все их временные файлы будут храниться в директории, заданной параметром session.save_path. Если система плохо работает с папками большого размера, то удобно использовать поддиректории. Для этого, кроме названия папки, в значение параметра добавляют еще и число, определяющее глубину вложенности поддиректорий в этой папке: N;/dir. Это значение нужно обязательно взять в кавычки, поскольку точка с запятой является одним из символов комментариев в файле настроек PHP. Все директории и поддиректории для хранения данных сессии нужно создать самостоятельно.

Например: 2;/Temp определяет, что переменные сессий будут храниться в папках вида c:\Temp\0\a\, c:\Temp\0\b\ и т.п.

Хранение данных на стороне клиента осуществляется с помощью cookies. Работу PHP с cookies можно настроить, в частности, с помощью параметров session.use_cookies, session.cookie_lifetime и т.п.

Параметр session.use_cookies определяет, использовать ли cookies при работе с сессиями. По умолчанию эта опция включена (т.е. принимает значение "1").

Параметр session.cookie_lifetime задает длительность жизни cookies в секундах. По умолчанию это "0", т.е. данные в cookies считаются правильными до закрытия окна браузера.

Кроме этих параметров, полезными могут оказаться session.name, определяющий имя сессии, session.auto_start, позволяющий автоматически запускать сессии, session.serialize_handler, задающий способ кодировки

данных сессии, и параметр session.cache_expire, определяющий, через сколько минут устаревает документ в кэше.

Имя сессии session.name по умолчанию устанавливается как PHPSESSID и используется в cookies как имя переменной, в которой хранится идентификатор сессии. Автоматический запуск сессий по умолчанию отключен, но его можно задать, сделав значение session.auto_start равным "1". Для кодирования данных сессии по умолчанию используется php. Устаревание данных, сохраненных в кэше, происходит через 180 минут.

Существует еще множество настроек, с которыми можно познакомиться в документации или непосредственно в файле настроек php.ini.

3.2.3 Создание сессии

Первое, что нужно сделать для работы с сессиями (если они уже настроены администратором Web-сервера), это запустить механизм сессий. Если в настройках сервера переменная session.auto_start установлена в значение "0" (если session.auto_start=1, то сессии запускаются автоматически), то любой скрипт, в котором нужно использовать данные сессии, должен начинаться с вызова функции session_start();

Выполняя данную функцию, сервер создает новую сессию или восстанавливает текущую, основываясь на идентификаторе сессии, переданном по запросу. Как это делается? Интерпретатор PHP ищет переменную, в которой хранится идентификатор сессии (по умолчанию это PHPSESSID) сначала в cookies, потом в переменных, переданных с помощью POST- или GET-запроса. Если идентификатор найден, то пользователь считается идентифицированным, производится замена всех URL и выставление cookies. В противном случае пользователь считается новым, для него генерируется новый уникальный идентификатор, затем производится замена URL и выставление cookies.

Команду session_start() нужно вызывать во всех скриптах, в которых предстоит использовать переменные сессии, причем до вывода каких-либо данных в браузер. Это связано с тем, что cookies выставляются только до вывода информации на экран.

Получить идентификатор текущей сессии можно с помощью функции session_id().

Для наглядности сессии можно задать имя с помощью функции session_name([имя_сессии]). Делать это нужно еще до инициализации сессии. Получить имя текущей сессии можно с помощью этой же функции, вызванной без параметров: session_name();

В примере ниже показано создание сессии в файле index.php:

```
<?
// создаем новую сессию или  восстанавливаем текущую
```

```
session_start();
echo session_id();    // выводим идентификатор сессии
?>
<html>
<head><title>My home page</title></head>
... // домашняя страничка
</html>
<?
// выводим имя текущей сессии.    В данном случае это PHPSESSID
echo session_name();
?>
```

Если проделать то же самое на другой странице сайта, то значения выводимых переменных (id сессии и ее имя) будут такими же, если перейти на нее с index.php и не закрывать перед этим окно браузера. В случае открытия нового окна браузера идентификатор сессии изменится.

3.2.4 Регистрация переменных сессии

От самих идентификатора и имени сессии пользы для решения практических задач немного. Необходимо передавать и сохранять в течение сессии какие-либо переменные (например, логин и пароль). Для того чтобы этого добиться, нужно просто зарегистрировать переменные в сессии. Для этого необходимо просто записать ее значение в ассоциативный массив $_SESSION, т.е.:

```
$_SESSION['имя_переменной'] =  'значение_переменой';
```

В этом массиве хранятся все зарегистрированные переменные сессии. Доступ к таким переменным осуществляется с помощью массива $_SESSION['имя_переменной'].

Рассмотрим пример регистрации логина и пароля, вводимых пользователем на странице авторизации (login.php). В случае если введенный логин – pit, а пароль – 123, то пользователь переадресовывается на секретную страничку secret_info.php:

```
<?
// создаем новую сессию или  восстанавливаем текущую
session_start();
if (!isset($_GET['go'])) {
  echo '<form>
    Login: <input type="text" name="login">
    Password: <input type="password" name="passwd">
    <input type=submit name=go value=Go>
  </form>';
} else {
   // регистрируем переменную login
   $_SESSION['login']=$_GET['login'];
   // регистрируем переменную passwd
   $_SESSION['passwd']=$_GET['passwd'];
   // теперь логин и пароль - глобальные переменные для этой сессии
   if ($_GET['login']=="pit" &&   $_GET['passwd']=="123") {
```

```
        // перенаправляем на страницу  secret_info.php
        header("Location: secret_info.php");
    } else {
      echo "Неверный ввод,   попробуйте еще раз<br>";
    }
} ?>
```

Теперь, попав на страничку secret_info.php (как и на любую другую страницу сайта), мы сможем работать с введенными пользователем логином и паролем, которые будут храниться в массиве $_SESSION. Таким образом, если код секретной странички:

```
<?php
session_start();
?>
<html>
<head><title>Secret info</title></head>
<body>
<?
// выводим все переменные сессии
print_r($_SESSION);
?>
<p>Здесь размещена секретная информация.
</body>
</html>
```

То мы получим в окне браузера после ввода на странице login.php логина pit и пароля 123 (рисунок 3.1):

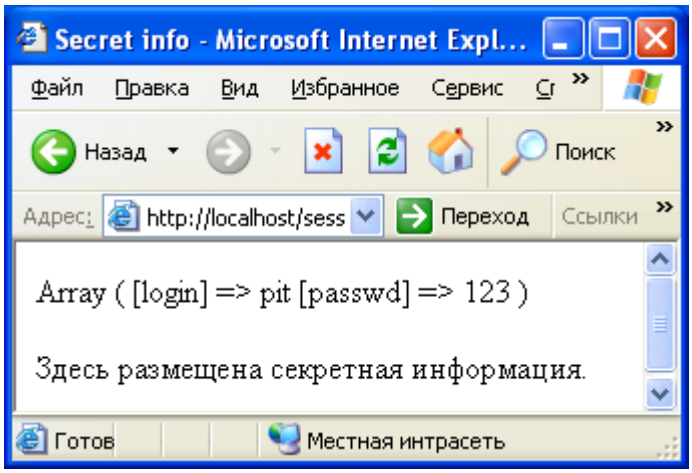

Рисунок 3.1 – Содержимое страницы secret_info

Т.е. получим список переменных, зарегистрированных на login.php (список ключей массива и их значения можно вывести, используя php-функцию *print_r*) и, собственно, саму секретную информацию.

Чтобы избежать проникновения на данную страницу не авторизированного пользователя (который просто наберет в строке браузера адрес секретной странички secret_info.php), нужно дописать всего пару строк в код нашей секретной странички:

```
<?php
session_start();
// проверяем правильность  пароля-логина если авторизация
// неудачна, то переадресовываем пользователя на страницу авторизации
if (!($_SESSION['login']==pit && $_SESSION['passwd']==123)) {
   Header("Location: authorize.php");
}
//Если авторизация прошла успешно выведется содержимое страницы
?>
<html>
<head><title>Secret info</title></head>
<body>
<?
print_r($_SESSION);  // выводим все переменные сессии
?>
<p>Здесь размещена секретная информация.
</body>
</html>.
```

3.2.5 Удаление переменных сессии

Кроме возможности регистрировать переменные сессии (т.е. делать их глобальными на протяжении всего сеанса работы), полезно также иметь возможность удалять такие переменные и сессию в целом.

Функция unset удаляет переменную из текущей сессии (т.е. удаляет ее из списка зарегистрированных переменных). Она не возвращает никакого значения, а просто уничтожает указанные переменные:

```
unset($_SESSION['login']).
```

Для чего может понадобиться возможность удаления переменных сессии? Например, для уничтожения данных о посетителе (в частности, логина и пароля для нашего примера) после его ухода с сайта. Ведь если не закрыть окно браузера после посещения сайта, то правильные логин и пароль сохранятся в параметрах сессии и любой другой пользователь этого компьютера сможет получить доступ к странице secret_info.php и прочитать закрытую информацию.

Для того чтобы уничтожить логин и пароль, введенные ранее, и убедиться, что они уничтожены, можно использовать следующий код:

```
<?
session_start();
unset($_SESSION[' passwd']);     // уничтожаем пароль
unset($_SESSION['login']);         // уничтожаем логин
print_r($_SESSION);               // выводим глобальные переменные сессии
```

Уничтожить текущую сессию целиком можно с помощью функции session_destroy(); Она не сбрасывает значения глобальных переменных

сессии и не удаляет cookies, а уничтожает все данные, ассоциируемые с текущей сессией.

```
<?
// инициализируем сессию
session_start();
$test = "Переменная сессии";

// регистрируем переменную $test.
$_SESSION['test']= $test;

// выводим все глобальные переменные сессии
print_r($_SESSION);

// выводим идентификатор сессии
echo session_id();
echo "<hr>";

// уничтожаем сессию
session_destroy();
print_r($_SESSION);
echo session_id();
?>
```

В результате работы этого скрипта будут выведены две строки: в первой - массив с элементом test и его значением, а также идентификатор сессии, во второй - пустой массив. Таким образом, видно, что после уничтожения сессии уничтожается и ее идентификатор, и мы больше не можем ни регистрировать переменные, ни вообще производить какие-либо действия с сессией.

3.2.6 Получание информации о посетителе сайта

Для получения информации о текущем пользователе в PHP могут быть использованы переменные суперглобального ассоциативного массива *$_SERVER*. Например, модель реализующая сбор и сохранение статистики в БД может выглядеть следующим образом:

```
 <?php
class StatisticModel extends Model{
  protected static $tablename = 'statistics';
  protected static $fields = array();

  public $time_statistic;
  public $web_page;
  public $ip_address;
  public $host_name;
  public $browser_name;

  function save_statistic($page){
    $this->time_statistic = date('Y-m-d h:m:s');
```

```
        $this->web_page = $page;
        $this->ip_address = $_SERVER['REMOTE_ADDR'];
        $this->host_name =  gethostbyaddr($_SERVER['REMOTE_ADDR']);
        $this->browser_name = $_SERVER['HTTP_USER_AGENT'];
        $this->save();
      }
    }
```

3.2.7 Рекомендации по разработке раздела администратора сайта

Для создания раздела администратора сайта, рекомендуется выполнить следующие действия:

- в директории app Web-сайта создать папку «admin», в которой разместить папки «controllers» и «vievs»;
- создать макет главной страницы раздела администратора(app/admin/views/admin_layout.php), по аналогии с макетом пользовательской части сайта, содержащий меню и включение основного содержимого. Например:

```
<!DOCTYPE html>
<html>
  <head>
…
  </head>
<body>
    <header>
      <h1>Администратор</h1>
      <nav>
        <ul>
          <li><?=      (strpos($_SERVER['REQUEST_URI'],'admin/blog/edit')      ?
'Редактор блога' : '<a href="/admin/blog/edit">Редактор блога</a>') ?></li>
          …
          <li><a href="/authorization/log_out">Выход</a></li>
          <li> <span id='thistime'></span> </li>
        </ul>
      </nav>
    </header>

    <div class='main_content'>
      <?
        include $content_view;
      ?>
    </div>
…
```

- модифицировать файл .htaccess добавив правило преобразования для url, содержащих admin/:

```
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
```

RewriteRule ^admin/(.)(.*) index.php?controller=$1&action=$2&**admin_area**=1 [QSA]*

• Модифицировать файл роутера, добавив префиксы *admin* в случае наличия соответствующего параметра в запросе:

```
if($_REQUEST['admin_area']){
    $admin_path = 'admin/';
    $admin_file_prefix = 'admin_';
    $admin_class_prefix = 'Admin';
  }else{
    $admin_path = '';
    $admin_file_prefix = '';
    $admin_class_prefix = '';
  }
…
$controller_file = "app/${admin_path}controllers/". $admin_file_prefix.
$controller_name.'_controller.php';
…
$controller_class_name = $admin_class_prefix .ucfirst($controller_name).
'Controller';
```

Теперь при переходе, например, на страницу */admin/blog/edit* будет создан экземпляр контроллера AdminBlogController из файла /app/admin/controller/admin_blog_controller и запущен его метод *edit*, который может выглядеть следующим образом:

```
<?php
class AdminBlogController extends AdminController{

  function edit(){
    $this-> authenticate();
    $this->view->render('admin_blog_edit.php', 'Редактор блога', $this-
>model, 'admin_layout.php', 'admin/');
  }
```

3.3 Варианты заданий

Работа выполняется на основе персонального сайта, полученого в ходе выполнения предыдущих лабораторных работ. Небходимо доработать проект в соответствии с указаними, данными в п.3.4 данной лабораторной работы.

3.4 Порядок выполнения работы

1. В соответствии с рекомендациями п. 3.2.7 реализовать зону администрирования сайта и переместить в нее страницы «Редактор блога» и «Загрузка сообщений гостевой книги».

2. Реализовать накопление информации о посетителях (см п. 3.2.6) страниц пользовательского раздела сайта в разработанной для этого таблице базы данных. Структура информации:

- Дата и время посещения;
- Web-страница посещения;
- IP-адрес компьютера посетителя;
- Имя хоста компьютера посетителя;
- Название браузера, который использовал посетитель.

3. Добавить в зону администрирования сайта страницу «Статистика посещений», отображающую информацию о посетителях сайта. Реализовать постраничное отображение информации в порядке убывания даты и времени посещения.
4. Добавить в зону администратора форму входа, содержащую поля логин и пароль. Логин и пароль администратора хранить к коде программы. Проверку пароля осуществлять сравнением зашифрованного пароля с вычисленным заранее хешем верного пароля. Метод контроллера, обрабатывающий форму ввода может содержать:

```
if ($_POST['login']=='admin@gmail.com') && (md5($_POST['password'])==' d8578edf8458ce06fbc5bb76a58c5ca4')) {
  $_SESSION['isAdmin']=1;
  return true;
}
$_SESSION['isAdmin']=0;
return false;
```

5. На всех страницах зоны администратора реализовать проверку авторизации и в случае, если посетитель не авторизован как администратор –перенаправлять пользователя на страницу авторизации). Для этой цели возможно добавить в базовый контроллер раздела администратора метод аутентификации, который будет проверять установленную контроллером переменную сессии *isAdmin*

```
class AdminController extends Controller{
  function authenticate(){
    if (!isset($_SESSION['isAdmin'])) {
      header('Location:/authorization/index');
      exit;
    }
  }
}
```

Добавив вызов данного метода вначале каждого метода контроллеров раздела администратора, обеспечим защиту раздела администратора от несанкционированного доступа.

6. В пользовательском разделе сайта разработать страницу «Регистрация пользователя», содержащую форму для ввода (все поля обязательны к заполнению):
   - ФИО пользователя;
   - e-mail пользователя;
   - логин пользователя;
   - пароль пользователя.

Добавлять введенную пользователем информацию в разработанную таблицу базы данных. Предусмотреть проверку на невозможность добавления двух одинаковых логинов.

7. Добавить в меню пользовательского раздела сайта ссылку «Войти» на форму авторизации. Форма должна содержать строку ввода логина, строку ввода пароля, кнопку "Войти", а также ссылку «Регистрация» (на страницу «Регистрация пользователя»).

После успешной авторизации в меню сайта вместо ссылки «Войти» должна отображаться ссылка "Выйти", при нажатии которой происходит выход авторизированного пользователя (завершение текущей сессии).

После успешной авторизации на каждой странице сайта (возможно в области меню) должно отображаться "Пользователь: ФИО" (ФИО – Фамилия Имя Отчество пользователя).

8. Реализовать отображение результатов тестирования на странице «Тест по дисциплине» только авторизованным пользователям.

3.5 Содержание отчета

Цель работы, порядок выполнения работы, тексты переработанных HTML-документов и разработанных PHP-скриптов, внешний вид переработанных Web-страниц и результаты работы PHP-скриптов, выводы по результатам работы.

3.6 Контрольные вопросы

3.6.1. Для чего используется механизм сессий?
3.6.2. Какие переменные файла php.ini влияют на работу механизма сессий?
3.6.3. Как идентифицируются пользователи в течение сессии работы с сайтом?
3.6.4. Как создается новая сессия в PHP?
3.6.5. Каким образом можно передавать значения PHP-переменных от одной страницы сайта к другой?
3.6.6. Как удаляются переменные сессии?
3.6.7. Каким образом можно уничтожить сессию?
3.6.8. Как в получить информацию о текущем пользователе(ip адрес, браузер и тп)?

4 ЛАБОРАТОРНАЯ РАБОТА №11

«Исследование возможностей асинхронного взаимодействия с сервером. Технология AJAX»

4.1 Цель работы

Исследовать возможности асинхронного обмена данными с сервером, изучить принципы асинхронной работы с сервером, приобрести практические навыки использования технологии AJAX для организации отправки данных на сервер и получения данных без полной перезагрузки страницы.

4.2 Краткие теоретические сведения

4.2.1 Технология AJAX

AJAX (Asynchronous Javascript and XML – асинхронный JavaScript и XML) – подход к построению интерактивных пользовательских интерфейсов web-приложений, заключающийся в асинхронном («фоновом») обмене данными между браузером и web-сервером без необходимости полной перезагрузки web-страницы. В результате web-приложения становятся более быстрыми и удобными.

AJAX не является самостоятельной технологией, а представляет собой методику использования нескольких смежных технологий: JavaScript, XMLHttpRequest; HTML, XML или JSON (JavaScript Object Notation); DOM.

AJAX базируется на двух основных принципах:

1. Использование одного из возможных вариантов динамического обращения к серверу без перезагрузки всей страницы. Среди нативных средств браузера можно отметить:
   - использование ***XMLHttpRequest*** (API, используемый в языках JavaScript, JScript, VBScript для пересылки различных данных (XML, JSON и т.д.) по HTTP-протоколу между браузером и веб-сервером);
   - использование ***Fetch API*** (предоставляет глобальный метод fetch, который даёт более современный и гибкий по сравнению с XMLHttpRequest способ для ассинхронной отправки и получения данных);
   - через динамическое создание ***дочерних плавающих фреймов(iFrame)***;
   - через динамическое создание тега ***<script>***.

Также возможно использование библиотечных функций, например, метода ***Ajax*** библиотеки jQuery.

2. Использование JavaScript для динамического изменения содержания страницы - информирования пользователя о ходе обмена данными с сервером и отображения полученных данных.

В качестве формата передачи данных в AJAX можно использовать HTML, XML или JSON (JavaScript Object Notation).

4.2.2 Технология динамического обращения к серверу XMLHttpRequest

Впервые XMLHttpRequest был разработан компанией Microsoft, появившись в компоненте Outlook Web Access программы Microsoft Exchange Server 2000. Он был назван IXMLHTTPRequest. Позднее, наработки были включены в состав MSXML 2.0 (Microsoft XML) в виде объекта ActiveX, доступного через JScript, VBScript, или другие скриптовые языки, поддерживающиеся браузером. MSXML 2.0 был включён в состав браузера Internet Explorer 5.

Программисты проекта Mozilla разработали совместимую версию, называющуюся nsIXMLHttpRequest в Mozilla 0.6. Доступ к компоненту был реализован через JavaScript-объект, названный XMLHttpRequest. Однако полная функциональность появилась в Mozilla 1.0. В дальнейшем этот объект был реализован компаниями Apple, начиная с Safari 1.2, родственным браузером Konqueror, компанией Opera Software начиная с Opera 8.01. Методы объекта XMLHttpRequest представлены в таблице 4.1.

Таблица 4.1 – Методы объекта XMLHttpRequest

Метод	Описание
abort()	Отменяет текущий запрос, удаляет все заголовки, ставит текст ответа сервера в null.
getAllResponseHeaders()	Возвращает полный список HTTP-заголовков в виде строки. Заголовки разделяются знаками переноса (CR+LF). Если флаг ошибки равен true, возвращает пустую строку. Если статус 0 или 1, вызывает ошибку INVALID_STATE_ERR.
getResponseHeader(headerName)	Возвращает значение указанного заголовка. Если флаг ошибки равен true, возвращает null. Если заголовок не найден, возвращает null. Если статус 0 или 1, вызывает ошибку INVALID_STATE_ERR.
open(method, URL, async, userName, password)	Определяет метод, URL и другие не обязательные параметры запроса; параметр async определяет, происходит ли работа в асинхронном режиме. Последние три параметра необязательны.
send(content)	Отправляет запрос на сервер.
setRequestHeader(label, value)	Добавляет HTTP-заголовок к запросу.
overrideMimeType(mimeType)	Позволяет указать mime-type документа, если сервер его не передал или передал неправильно. **Внимание**: метод отсутствует в Internet Explorer!

Свойства объекта XMLHttpRequest представлены в таблице 4.2.

Таблица 4.2 – Свойства объекта XMLHttpRequest

Свойство	Тип	Описание
onreadystatechange	EventListener	Обработчик события, которое происходит при каждой смене состояния объекта. Имя должно быть записано в нижнем регистре.
readyState	unsigned short	Текущее состояние объекта (0 – не инициализирован, 1 – открыт, 2 – отправка данных, 3 – получение данных и 4 – данные загружены)
responseText	DOMString	Текст ответа на запрос. Если статус не 3 или 4, возвращает пустую строку.
responseXML	Document	Текст ответа на запрос в виде XML, который затем может быть обработан посредством DOM. Если статус не 4, возвращает null.
Status	unsigned short	HTTP-статус в виде числа (404 – «Not Found», 200 – «OK» и т. д.)
statusText	DOMString	Статус в виде строки («Not Found», «OK» и т. д.). Если статус не распознан, браузер пользователя должен вызвать ошибку INVALID_STATE_ERR.

Использование объекта XMLHttpRequest включает следующие этапы:

1) создание экземпляра объекта XMLHttpRequest;
2) установка обработчика события;
3) открытие соединения и отправка запроса.

На стадии создания экземпляра объекта XMLHttpRequest для старых версий браузеров необходима отдельная реализация(создание объекта отличается: в IE5-IE6 она реализована через ActiveXObject, а в остальных браузерах (IE7 и выше, Mozilla, Opera, Chrome, Netscape и Safari) – как встроенный объект типа XMLHttpRequest.

В общем случае, вызов объекта выглядит так:

```
var req = new XMLHttpRequest();
```

Установка обработчика событий, открытие соединения и отправка запросов выглядят так:

```
req.open(<"GET"|"POST"|...>, <url>, <asyncFlag>);
req.onreadystatechange = processReqChange;
```

После определения всех параметров запроса его остается только отправить. Делается это функцией send(). Если необходимо передать на сервер POST-данные, их надо подставить в качестве параметра для этой функции. POST-данные должны быть свернуты в URL-escaped строку (кодировка UTF-8) и добавлен заголовок req.setRequestHeader ("Content-Type", "application/x-www-form-urlencoded") (либо "text/xml" в случае

передачи xml данных). При отправке GET-запроса для версии без ActiveX необходимо указать параметр null, в остальных случаях можно не указывать никаких параметров. Не будет ошибкой, если при отправке методом GET всегда будет указан параметр null:

```
req.send(null);
```

В случае отправки методом POST в аргументе метода send должна быть передана URL-escaped строка с данными. После этого начинает работать вышеуказанный обработчик событий. Он – фактически основная часть программы. В обработчике обычно происходит перехват всех возможных кодов состояния запроса и вызов соответствующих действий, а также перехват возможных ошибок. Пример кода с этими двумя функциями:

```
var req;

function loadXMLDoc(url)
{
    req = new XMLHttpRequest();
    req.open("GET", url, true);
    req.onreadystatechange = processReqChange;
    req.send(null);
}

function processReqChange()
{
  try {
     // только при состоянии "complete"
     if (req.readyState == 4) {
        // для статуса "OK"
        if (req.status == 200) {
           // отображение полученных данных
           alert('Данные успешно получены');
        } else {
            alert("Не удалось получить данные:\n" + req.statusText);
        }
     }
  }
  catch( e ) {
    alert('Caught Exception: ' + e.description);
  }
}
```

4.2.3 Использование Fetch API

Fetch API [18] предоставляет более современный, чем XmlHttpRequest программный интерфейс для выполнения ассинхронных обменов. Fetch API определяет объекты Request, Response, Headers (и другие, связанные с сетевыми запросами). Кроме того, определяет метод ***fetch()***, доступный глобально.

Метод fetch() принимает один обязательный аргумет — путь к данным, которые необходимо получить. Он возвращает ***Promise*** [18], который разрешается в Response независимо от того, был ли запрос удачным. Во втором аргументе может быть передан необязательный объект с указанием параметров вызова:

- method – метод запроса,
- headers – заголовки запроса (объект),
- body – тело запроса: FormData, Blob, строка и т.п.
- mode – одно из: «same-origin», «no-cors», «cors», указывает, в каком режиме кросс-доменности предполагается делать запрос.
- credentials – одно из: «omit», «same-origin», «include», указывает, пересылать ли куки и заголовки авторизации вместе с запросом.
- cache – одно из «default», «no-store», «reload», «no-cache», «force-cache», «only-if-cached», указывает, как кешировать запрос.
- redirect – можно поставить «follow» для обычного поведения при коде 30x (следовать редиректу) или «error» для интерпретации редиректа как ошибки.

Как только Response выполнится успешно, становятся доступными несколько методов для определения тела контента.

Пример использования:

```
fetch('/article/fetch/user.json')
    // выполнится когда будет получен отклик на запрос
  .then(function(response) {
    alert(response.headers.get('Content-Type')); //выведет application/json charset=utf-8
    alert(response.status); //выведет 200
    return response.json();
  })
    // выполнится только когда завершится предыдущий then
  .then(function(user) {
    alert(user.name);
  })
    // выполнится только если при выполнении запроса произойдет ошибка
  .catch( alert('Error!') );
```

4.2.4 Создание плавающего фрейма iFrame

Альтернативный вариант обмена данными с сервером – использование невидимого iFrame (тег HTML <iFrame> – плавающий фрейм – позволяет встраивать в любое место документа фрейм без необходимости использования тега <frameset>). Динамическое изменение URL iFrame инициирует GET-запрос по указанному URL, таким образом, iFrame может использоваться для отправки запроса серверу.

Кроме того, iFrame может использоваться для отправки файлов на сервер. Для этого необходимо реализовать POST-запрос для формы, содержащей элемент ввода файла, предварительно установив в атрибут target формы имя iFrame (атрибут name). При этом iFrame выполнит POST-запрос по URL, указанному в форме.

Изменение URL iFrame для выполнения запроса к серверу рекомендуется осуществлять вызовом:

iframeDocument.location.src.replace(newURL).

Такой синтаксис, в отличие от iframeDocument.location.src=..., не отражается на истории посещений (объект History) в большинстве браузеров, что делает запрос невидимым для посетителя.

Получение ссылки на документ, содержащийся в iFrame, может зависеть от типа браузера: iframe.contentDocument, iframe.contentWindow.document, либо iframe.document:

```
// получить окно по тегу iFrame
function getIframeDocument(iframeNode) {
  if (iframeNode.contentDocument) return iframeNode.contentDocument
  if (iframeNode.contentWindow) return iframeNode.contentWindow.document
return iframeNode.document
}
```

Момент окончания получения ответа от сервера в iFrame для большинства браузеров может быть установлен с использованием обработчика события onLoad, вызывающегося после загрузки iFrame. В браузерах IE для этой цели следует использовать событие onreadystatechange, которое имеют все загружающиеся элементы (document, frame, frameSet, iframe, img, link, object, script):

```
if (navigator.userAgent.indexOf("MSIE") > -1 && !window.opera){
    iframe.onreadystatechange = function(){
        if (iframe.readyState == "complete"){
            alert("iframe загружен.");
        }
    };
} else {
    iframe.onload = function(){
        alert("iframe загружен.");
    };
}
```

Для доступа к основному окну из обработчика окончания загрузки iFrame, как и в случае обычных фреймов, может быть использован объект parent.

В примере ниже для отправки файла на сервер динамически создается невидимый iFrame. После получения файла, если его размер не превышает 100 Кб, скрипт сервера размещает его в директории Uploaded и

формирует сообщение о успешной загрузке, в противном случае формирует сообщение об ошибке:

```
//обработка полученого файла
<? if( isset($_FILES['attach']) )
{
        $file = (object) $_FILES['attach'];
        if( $file->size && $file->size <= 102400 )
        {
                $savepath =  './'.$_FILES['attach']['name'];
                move_uploaded_file( $file->tmp_name, $savepath );
                echo "Файл ".$_FILES['attach']['name']." размером ".$file->size."
байт успешно загружен на сервер!";
        }
   else echo 'Файл не был загружен, т.к его размер превышает
допустимый!';
    exit;
}
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=win1251" />
<title> AJAX отправка файла </title>

<script type="text/javascript">
function fileUpload(myform)
{//создание iFrame
        iframe = document.createElement('iframe');
//формирование уникального имени фрейма
   iframe.name = myform.target= Date.parse(new Date);
     /i/frame  невидимый
     iframe.style.display = 'none';
//обработчик ответа сервера - окончания загрузки файла
     iframe.onload = iframe.onreadystatechange = function(){
       parent.document.getElementById('results').innerHTML =
this.contentWindow.document.body.innerHTML;
     }
    //добавление фрейма в документ
     document.body.appendChild(iframe);
    //отображение начала процесса загрузки
     document.getElementById('results').innerHTML = 'Загрузка...'
     //отправка данных в фрейм
     myform.submit();
}
</script>

</head>
<body>
<div class="border block">
```

```
Загрузка файла на сервер при помощи AJAX. <br />
Размер файла не должен превышать 100 Кб
<form action='<?=$_SERVER["PHP_SELF"]?>'
  method="post" target="upload" enctype="multipart/form-data">
      <input type="file" size="50" name="attach" />
      <input type="button" name="send" value="Загрузить файл" on-
click="fileUpload(document.forms[0])" />
</form>

</div>
<div class="border" style="float:left; width:350px;" id="results">
      Тут будут отображаться результаты
</div>
</body>
</html>
```

4.2.4 Создание <script>

Тот факт, что с использованием DOM возможно создавать динамические теги, дает возможность использовать для обмена данными с сервером динамическое создание тега <script>.

Добавление нового <script> в документ заставляет браузер немедленно запрашивать скрипт по URL, указанному в его свойстве src.

Следующая функция добавляет в head документа тег <script> с заданными id и src:

```
function attachScript(id, src){
  var element = document.createElement("script")
  element.type = "text/javascript"
  element.src = src
  element.id = id
  document.getElementsByTagName("head")[0].appendChild(element)
}
```

Вызов:

```
attachScript('script1','translate.php?word=apple')
```

добавит в head документа тег:

```
<script src='translate.php?word=apple'></script>
```

Браузер тут же обработает тег: запросит translate.php и попытается выполнить возвращенный сервером код как JavaScript-программу.

После полной загрузки кода, для отображения полученных данных, в конце скрипта, возвращаемого сервером, обычно выполняется вызов специально подготовленной функции – функции возврата.

Так как браузер получает готовый JavaScript код, то этот метод зачастую является самым лаконичным на стороне клиента.

Кроме того, по сравнению с методами XMLHttpRequest/iFrame, он имеет одно важное преимущество: в данном случае AJAX-запросы (URL

тега script) можно направлять не только к собственному, но и к любому другому серверу.

В примере ниже с использованием динамического тега <script> содержимое текстового поля отправляется на сервер, где переворачивается (записывается в обратном порядке). В случае если размер строки менее 5 символов, то возвращается сообщение об ошибке:

```
<?php
if (isset($_REQUEST ['text'])){
 //указываем, что тип возвращаемого документа javascript
 header ('Content-Type: text/javascript');
  $text = isset($_REQUEST ['text'])?$_REQUEST ['text']:'';
 if (strlen($text)< 5){
  echo 'var js_ErrCode = true;' . "\n";
  echo 'var js_ErrMsg = "Ошибка, не переворачиваю";' . "\n";
 }
 else{
  echo 'var js_ErrCode = false;' . "\n";
  echo 'var js_ErrMsg = "Ошибок нет, переворачиваю";' . "\n";
 }
 echo 'var js_Result = "'.strrev ($text) .'"; ' . "\n";
 echo 'makeLoadComplete ();' . "\n";
 exit;
}
?><html>
 <head>
 <script type="text/javascript" language="javascript">
 // функция возврата, вызываемая  когда php-скрипт вернет данные браузеру
 function makeLoadComplete(){
  var dv_Result = document.getElementById ('dv_Result');
  // проверяем код завершения переворота строки
  if (js_ErrCode){
    // если ошибка, то выводим текст сообщения об ошибке
    dv_Result.innerHTML = js_ErrMsg;
    dv_Result.style.color = 'red';
  }
  else{  // если все удачно, то выводим результат в подготовленный блок div
    dv_Result.innerHTML = js_Result;
    dv_Result.style.color = 'green';
  }
 }

 function create_script() {
  var elScript = document.createElement( 'script' );
  var str_for_rev = document.getElementById('str_for_rev');
  // и отправляем запрос
  elScript.src='<?=$_SERVER["PHP_SELF"]?>?text='+str_for_rev.value;
  document.body.appendChild( elScript );
```

```
} </script>
</head>

<body>
 <!–блок для вывода результата -->
 <div id="dv_Result" style="">Result ...</div>
 Введите в поле некоторый текст:
 <input type="text" id="str_for_rev" value="text value" /> <br />
 И нажмите на кнопку, чтобы выполнить его переворачивание:
 <input type="button" onclick=" create_script()" value="Revert !"/>
</body>
</html>
```

4.2.5 Форматы передачи данных в AJAX

Формат HTML

В самом простом случае – ответом на AJAX-запрос является фрагмент HTML. Например:

```
<p class="notification">Файл успешно загружен на сервер</p>
```

Такой фрагмент может быть сразу отображен с использованием свойства innerHTML какого-либо узла DOM:

```
domObject.innerHTML = data
```

К преимуществам такого подхода относится отсутствие необходимости дополнительной обработки полученных данных.

В качестве недостатка можно отметить возможную избыточность при передаче – вместе с данными передается HTML-разметка.

Формат XML

В том случае, когда скрипт сервера получает и отправляет XML-данные, необходимо перед отправкой данных на клиенте преобразовать их в формат XML, а при получении XML-документа с сервера его необходимо обработать с использованием JavaScript и отобразить нужным образом.

Например, если пользователь набирает свое имя и адрес в какой-либо форме на своей Web-странице, вы можете получить из формы примерно такие данные:

```
firstName=’Иван’
lastName=’Иванов’
street=’Ленина, 22’
city=’Севастополь’
```

В формате XML эти данные могут выглядеть следующим образом:

```
<profile>
  <firstName>Иван</firstName>
  <lastName>Иванов</lastName>
  <street>Ленина, 22</street>
  <city>Севастополь</city>
</ profile>
```

Таким образом, необходимо преобразовать данные формы в формат XML с использованием JavaScript, после чего они могут быть отправлены. В примере ниже отправка происходит с использованием объекта xmlHtttp типа XMLHttpRequest, проинициализированного ранее:

```
var firstName = document.getElementById("firstName").value;
var lastName = document.getElementById("lastName").value;
var street      = document.getElementById("street").value;
var city         = document.getElementById("city").value;

 var xmlString = "<profile>" +
    "  <firstName>" + escape(firstName) + "</firstName>" +
    "  <lastName>" + escape(lastName) + "</lastName>" +
    "  <street>" + escape(street) + "</street>" +
    "  <city>" + escape(city) + "</city>" +
    "</profile>";

  // URL для отправки
  var url = "saveAddress.php";

// Откроем соединение с сервером
xmlHttp.open("POST", url, true);

// Сообщим серверу, что посылаются данные в формате XML
xmlHttp.setRequestHeader("Content-Type", "text/xml");

// Установим функцию уведомления пользователя о
// завершении операции на сервере
xmlHttp.onreadystatechange = confirmUpdate;

// Отправим данные
xmlHttp.send(xmlString);
```

Доступ к данным, отправленным методом POST не из формы, а с использованием объекта XMLHttpRequest, возможен на сервере с помощью чтения входного потока PHP: *php://input.*

```
$post = file_get_contents('php://input');
```

Для обработки полученных XML-данных на сервере возможно использование расширения PHP SimpleXML или расширения XML Parser Functions, встроенного в PHP по умолчанию. В простейшем случае возможно использование функций работы со строками для выделения данных из полученной XML строки.

Обработка сформированных сервером XML-данных на стороне клиента возможна с использованием XML-анализаторов, встроенных во все современные браузеры.

Имеются некоторые отличия между анализаторами в Microsoft и в других браузерах. Первый поддерживает как загрузку XML файлов, так и

текстовых строк, содержащих XML код, в то время как в других браузерах используются раздельные анализаторы. При этом все анализаторы имеют функции для перемещения по дереву(DOM) XML документа, доступа, вставки и удаления узлов в дереве.

Формат JSON

В простейшем случае JSON позволяет преобразовывать данные, представленные в объектах JavaScript, в строку, которую можно легко передавать от одной функции к другой или – в случае асинхронного приложения – от Web-клиента к серверной программе.

Преимущество использования JSON состоит в том, что этот формат можно легко интерпретировать как на стороне клиента так и на стороне сервера.

По сути, JSON – это JavaScript-код, определяющий некую структуру данных. В нем используются две основные синтаксические конструкции:

```
// объявление массива:
var array = [ v1, v2, ... ];
// объявление ассоциативного массива:
var hash = { "key1" : v1, "key2" : v2, ... };
```

С их помощью можно описать структуру данных произвольной сложности. Например:

```
{
   "firstName": "Иван",
   "lastName": "Иванов",
   "address": {
      "street": "Ленина, 22",
      "city": "Севастополь",
   },
   "phoneNumbers": [
      "978-765-1234",
      "978-123-4567"
   ]
}
```

Если предположить, что вышеприведенный текст находится в переменной json_var, то для его преобразования в JavaScript объект достаточно всего лишь вызвать функцию JSON.parse, передав в нее переменную, json_var как показано ниже:

```
var p = JSON.parse(json_var);
div.innerHTML = p.firstName+" "+p.lastName+" живет в городе
"+p.address.city;
```

Для работы на сервере с форматом JSON в PHP встроены функции json_encode, json_decode:

```
string json_encode ( mixed $value [, int $options = 0 ] )
```

```
mixed json_decode ( string $json [, bool $assoc = false [, int $depth = 512 ]] )
```

Например, для преобразования массива в JSON -строку:

```
<?php
$arr = ['a'=>1, 'b'=>2, 'c'=>3, 'd'=>4,'e'=>5];
echo json_encode($arr);

// выведет {"a":1,"b":2,"c":3,"d":4,"e":5}
?>
Для обратного преобразования в объект:
<?php
$json = '{"a":1,"b":2,"c":3,"d":4,"e":5}';
var_dump(json_decode($json));
// выведет
// object(stdClass)#1 (5) {
//    ["a"] => int(1)
//    ["b"] => int(2)
//    ["c"] => int(3)
//    ["d"] => int(4)
//    ["e"] => int(5)
//}
?>
```

А для преобразования в ассоциативный массив:

```
<?php
$json = '{"a":1,"b":2,"c":3,"d":4,"e":5}';
var_dump(json_decode($json,true));
// выведет
// array(5) {
//    ["a"] => int(1)
//    ["b"] => int(2)
//    ["c"] => int(3)
//    ["d"] => int(4)
//    ["e"] => int(5)
//}
```

В простейшем случае в PHP возможно использование функций работы со строками для выделения данных из полученной JSON-строки.

4.2.6 Рекомендации по реализации AJAX запросов в MVC приложении

Рассмотрим фрагмент JavaScrip кода, реализующий с использованием jQuery отображение окна добавления комментария записи блога, отправляющий данные введенные пользователем на сервер с использованием iFrame и отображающий сам комментарий и время его добавления, полученное с сервера в качестве отклика, после успешного сохранения коментария в базу данных:

```
function commentWindow(post_id,personal_fio) {
  var form = document.createElement("form");
```

```
        var div = document.createElement("div");
        //формируем и отображаем окно с формой для ввода комментария
        $(div).append($(form));
        $(div).addClass("comment");
        $(form).attr("action", "/comment/save")
              .attr("method", "POST")
              .attr("target", "myFrame");
        $(form).html('<p>Пожалуйста, оставьте ваш комментарий!</p><textarea
id="message_ta" name="message" rows="10" cols="100"></textarea>'
          +'<input type="hidden" name="blog_id" value="' + post_id + '"></input>'
          +'<br><input type="button" id="addCommentButton" value="Сохранить
комментарий"/>'
        );
        document.body.appendChild(div);
        //отправляем введенные данные и отображаем комментарий в случае
отсутствия ошибок
        $("# addCommentButton").click(function(){
          var iframe = document.createElement("iframe");
          $(iframe).attr('name', 'myFrame')
            .on('load', function() {
              if(this.contentWindow.document.body.innerHTML){
                var data = JSON.parse(this.contentWindow.document.body.innerHTML),
                  time = data.time,
                  errors = data.errors,
                  message = $('#message_ta').val();
                if(data.errors){
                  window.alert('Ошибка сохранения комментария:'+ data.errors);
                } else{
                  //добавляем текст комментария и серверное время его
добавления
                  $('<tr><td> '+ personal_fio + '</td><td>' + time + '</td><td>' + message
+ '</td></tr>').prependTo('#comments' + post_id + '>tbody');
                }
                //удаляем использованные модальное окно и iframe
                $(div).remove();
                $(iframe).remove();
              }
            });

          document.body.appendChild(iframe);
          $(form).submit();
        });
      }
```

Как видно из приведенного кода, данные формы, размещенной в iFrame, будут отправлены в экшн save котроллера CommentsController(адрес action формы *"/comment/save")*. Так как в данном случае формат отправляемых данных application/x-www-form-urlencoded (формат данных, отправляемых из HTML формы по умолчанию), а

использованный метод – POST, экшн может получить отправленные данные из массива $_POST и должен сохранить их в БД, а в качестве результата вернуть текущие дату и время сервера; а в случае ошибки – соответствующее сообщение. При этом форматом ответа будет – JSON:

```
<?php
class CommentController extends Controller{

 function save(){
  $this->model->blog_id = $_POST['blog_id'] ;
  $this->model->user_id = $_SESSION['user_id'];
  $this->model->message = $_POST['message'];
  $this->model->save()
  echo "{time: '"+ date("Y-m-d h:m:s")+"', errors: '"+$this->model->errors+"}";
 }
```

4.3 Варианты заданий

Работа выполняется на основе проекта, полученого в ходе выполнения предыдущих лабораторных работ. Небходимо доработать проект в соответствии с указаниями, данными в п.4.4 данной лабораторной работы с учетом вариантов заданий из таблицы 4.3.

Таблица 4.3 – Варианты задания

№	Задание №1	Задание №2	Задание №3
1	XMLHttpRequest, Plaintext	Fetch, HTML	iFrame, JSON
2	Fetch, HTML	iFrame, JSON	Script, XML
3	iFrame, JSON	Script, XML	XMLHttpRequest, HTML
4	Script, XML	XMLHttpRequest, HTML	Fetch, JSON
5	XMLHttpRequest, HTML	Fetch, JSON	iFrame, XML
6	Fetch, JSON	iFrame, XML	Script, Plaintext
7	iFrame, XML	Script, Plaintext	XMLHttpRequest, JSON
8	Script, Plaintext	XMLHttpRequest, JSON	Fetch, XML
9	XMLHttpRequest, JSON	Fetch, XML	iFrame, Plaintext
10	Fetch, XML	iFrame, Plaintext	Script, HTML
11	iFrame, Plaintext	Script, HTML	XMLHttpRequest, Plaintext
12	Script, HTML	XMLHttpRequest, Plaintext	Fetch, HTML

4.4 Порядок выполнения работы

1. На странице «Регистрация пользователя» реализовать проверку занятости логина без перезагрузки страницы (использовать событие onBlur или кнопку «Проверить занятость»). Использовать для отправки данных на сервер метод и формат данных, указанный в колонке «Задание №1» таблицы 4.3 в соответствии с вариантом задания.

2. Реализовать возможность комментирования записей блога для авторизированных пользователей сайта. В случае, если пользователь авторизирован отображать после каждой записи блога кнопку или ссылку «Добавить комментарий». При нажатии кнопки/ссылки отображать модальное окно (реализовать с использванием, например, с использованием абсолютно спозиционированного блока div), содержащее поле ввода текста комментария и кнопку «Отправить». Реализовать отправку комментария на сервер и его отображение (дата и время, автор, текст комментария) под записью блога (или под уже существующими комментариями данной записи) без полной перезагрузки страницы. Комментарии хранить в специально разработанной таблице базы данных. Использовать для отправки данных на сервер метод и формат данных, указанный в колонке «Задание №2» таблицы 4.3 в соответствии с вариантом задания.

3. В зоне администратора реализовать возможность редактирования записей блога. Отображать возле каждой записи блога кнопку или ссылку «Изменить». При нажатии кнопки/ссылки отображать окно (div), содержащее поле ввода с темой записи блога и поле ввода с текстом записи блога для редактирования, а также кнопку/ссылку «Сохранить изменения». Реализовать отправку данных и их отображение без полной перезагрузки страницы. Использовать для отправки данных на сервер метод и формат данных, указанный в колонке «Задание №3» таблицы 4.3 в соответствии с вариантом задания.

4.5 Содержание отчета

Цель работы, порядок выполнения работы, тексты переработанных и разработанных Javascript и PHP скриптов, внешний вид переработанных Web-страниц и результаты работы PHP-скриптов, выводы по результатам работы.

4.6 Контрольные вопросы

4.6.1. Что представляет собой технология AJAX?

4.6.2. Каковы преимущества использования AJAX?

4.6.3. Какие технологии используются в AJAX?

4.6.4. Как в AJAX может организовываться обмен данными с сервером?

4.6.5. Какие форматы данных можно передавать между клиентом и сервером при использовании технологии AJAX?

4.6.6. Приведите пример использования объекта XMLHttpRequest?

4.6.7. Приведите пример использования Fetch?

4.6.8. Приведите пример использования iFrame для организации ассинхронного обмена с сервером?

4.6.9. Как можно реализовать ассинхронный запрос с испрлтзованием динамически создаваемого тега script?

5 ЛАБОРАТОРНАЯ РАБОТА №12

«Исследование возможностей фреймворка Laravel для разработки веб-приложений»

5.1 Цель работы

Ознакомиться с возможностями фреймворка Laravel и приобрести практические навыки реализации веб-приложений с его помощью.

5.2 Краткие теоретические сведения

5.2.1 Фреймворк Laravel

Laravel – MVC (Model View Controller – модель-представление-контроллер) веб-фреймворк с открытым исходным кодом [11, 15]. Некоторые особенности, лежащие в основе архитектуры Laravel и обеспечившие фреймворку высокую популярность, перечислены ниже:

- Использование *пакетов* (packages) позволяет создавать и подключать к приложению на Laravel модули в формате Composer [10].
- Использование REST-full контроллеров обеспечивает отделение логики обработки HTTP запросов.
- Использование ORM(Object-Relational Mapping - объектно-реляционное отображение) – система объектно-реляционного отображения *Eloquent,* используемая в Laravel, реализует шаблон проектирования ActiveRecord [17].
- Использование *миграций* – системы управления версиями для базы данных. Позволяет связывать изменения в коде приложения с изменениями, которые требуется внести в структуру БД, что упрощает развёртывание и обновление приложения.
- Использование шаблонизатора *Blade* обеспечивает гибкое построение представлений с использованием таких управляющих структур, как условные операторы, циклы и тп.
- Встроенный страничный вывод (pagination) упрощает генерацию страниц, заменяя различные способы решения этой задачи единым механизмом, встроенным в Laravel.

  Встроенное модульное тестирование(юнит-тесты) обеспечивает.

5.2.2 Установка Laravel

Перед установкой Laravel 5 необходимо убедиться, что установлено следующее ПО:

- PHP >= 5.5.9
- PDO расширение для PHP (для версии Laravel 5.1+)
- MCrypt расширение для PHP (для версии Laravel 5.0)
- OpenSSL (расширение для PHP)

- Mbstring (расширение для PHP)
- Tokenizer (расширение для PHP)

Также для установки и использования Laravel потребуется PHP менеджер зависимостей *Composer* [10], поэтому следует сначала установить его. Используя Composer, возможно скачать установщик Laravel коммандой:

composer global require "laravel/installer"

После этого, добавив каталог ~/.composer/vendor/bin в переменную PATH(чтобы исполняемый файл laravel мог быть найден системой), команда *laravel new* произведёт установку Laravel в указанный каталог. Например,

laravel new blog

создаст каталог с именем blog, содержащий Laravel со всеми установленными зависимостями.

Также возможна установка всего необходимого ПО с использованием VirtualBox [20] и Vagrant [21]. Для этого разработчики Laravel подготовили Laravel Homestead [11] – образ (box) для Vagrant. Homestead запускается на ОС Windows, Mac и Linux, и включает в себя веб-сервер Nginx, PHP 7.0, MySQL, Postgres, Redis, Memcached, Node и другое полезное ПО, которое может понадобиться для разработки Laravel-приложений.

5.2.3 Создание web приложения на Laravel

Для того, чтобы получить представление о процессе создания веб приложений с использованием Laravel, рассмотрим создание приложения реализующего список задач.

5.2.3.1 Подготовка базы данных. Миграции БД

Для определения таблицы базы данных для хранения задач необходимо создать *миграцию БД*. Миграции БД в Laravel позволяют простым способом определить структуру таблицы базы данных и выполнять модификации с использованием простого и выразительного PHP кода. Вместо того, чтобы вручную добавлять столбцы в локальные копии БД, можно просто запустить миграции.

Для создания заготовок различных классов(миграций, моделей и тп) в Laravel может быть использован интерфейс Artisan. Lля создания миграции базы данных для таблицы tasks может быть использована команда:

php artisan make:migration create_tasks_table --create=tasks

Миграция будет помещена в каталог database/migrations проекта. Необходимо отредактировать файл миграции и добавить дополнительный столбец string для имён задач:

```
<?php
 use Illuminate\Database\Schema\Blueprint;
```

```
use Illuminate\Database\Migrations\Migration;


class CreateTasksTable extends Migration
{
  /**
  * Запуск миграций
  *
  * @return void
  */
  public function up()
  {
    Schema::create('tasks', function (Blueprint $table) {
      $table->increments('id');
      $table->string('name');
      $table->timestamps();
    });
  }

  /**
  * Откатить миграции
  *
  * @return void
  */
  public function down()
  {
    Schema::drop('tasks');
  }
}
```

Чтобы запустить миграцию, необходимо использовать команду Artisan migrate:

php artisan migrate

Эта команда создаст таблицу БД tasks, которая содержит столбцы, определённые в миграции. Теперь необходщимо определить модель Eloquent ORM для наших задач.

5.2.3.2 Модели Eloquent

Определим модель Task, которая будет соответствовать только что созданной таблице tasks. Мы снова можем использовать команду Artisan, чтобы сгенерировать эту модель. В этом случае, мы будем использовать команду make:model:

php artisan make:model Task

Модель будет помещена в каталог app приложения. По умолчанию класс модели пуст. Нам не надо явно указывать, какой таблице соответствует Eloquent модель, потому что подразумевается, что имя таблицы – это имя модели во множественном числе (s на конце). В этом случае модель Task, как предполагается, соответствует таблице базы данных tasks. Пустая модель выглядит следующим образом:

```
<?php
 namespace App;
 use Illuminate\Database\Eloquent\Model;
 class Task extends Model
 {
  //
 }
```

5.2.3.3 Маршрутизация. Заглушки маршрутов

Теперь необходимо добавить в приложение несколько маршрутов. Маршруты используются для связи URL с контроллерами или анонимными функциями, которые должны быть выполнены, когда пользователь переходит на данную страницу. По умолчанию все маршруты Laravel определены в файле app/Http/routes.php, который автоматически добавляется в каждый новый проект.

Для приложения список задач, будут нужны по крайней мере три маршрута: маршрут для вывода на экран списка всех наших задач, маршрут для добавления новых задач и маршрут для удаления существующих задач. Заглушки для всех этих маршрутов в файле app/Http/routes.php выглядят следующим образом:

```
use App\Task;
use Illuminate\Http\Request;
/**
 * Вывести список всех задач
 */
Route::get('/', function () {
  //
});
/**
 * Добавить новую задачу
 */
Route::post('/task', function (Request $request) {
  //
});
/**
 * Удалить существующую задачу
 */
Route::delete('/task/{id}', function ($id) {
  //
});
```

Заполним маршрут /. По этому маршруту отобразим HTML-шаблон, который содержит форму добавления новой задачи, а также список всех текущих задач.

В Laravel все HTML-шаблоны хранятся в каталоге resources/views, и возможно использовать вспомогательную функцию view(), чтобы возвратить один из этих шаблонов по маршруту:

```
Route::get('/', function () {
```

```
  return view('tasks');
});
```

Теперь необходимо создать это представление

5.2.3.4 Создание макетов и представлений

Приложение содержит одно представление с формой добавления новых задач, а также список всех текущих задач.

Почти все веб-приложения используют один макет на всех своих страницах. Например, у приложения есть верхняя панель навигации, которая присутствовала бы на каждой странице (если бы у нас их было больше одной). Laravel упрощает использование этих общих функций на всех страницах, используя макеты Blade.

Определим представление нового макета в resources/views/layouts/app.blade.php. Расширение .blade.php даёт фреймворку команду использовать механизм шаблонной обработки Blade, чтобы отобразить это представление. В Laravel также могут быть использованы и простые PHP-шаблоны. Однако Blade позволяет ускорить написание шаблона.

Представление app.blade.php должно выглядеть примерно так:

```
// resources/views/layouts/app.blade.php
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Laravel Quickstart - Basic</title>
    <!-- CSS и JavaScript -->
  </head>
  <body>
    <div class="container">
      <nav class="navbar navbar-default">
        <!-- Содержимое Navbar -->
      </nav>
    </div>
    @yield('content')
  </body>
</html>
```

Строка @yield('content') в макете – это специальная Blade-директива для указания всем дочерним страницам, наследующим этот шаблон, где они внедряют своё содержимое.

Определим дочернее представление, которое будет использовать этот макет и выводить его основной контент. Определим это представление в resources/views/tasks.blade.php:

```
// resources/views/tasks.blade.php
@extends('layouts.app')
@section('content')
  <!-- Bootstrap шаблон... -->
  <div class="panel-body">
    <!-- Отображение ошибок проверки ввода -->
    @include('common.errors')
```

```
    <!-- Форма новой задачи -->
    <form action="/task" method="POST" class="form-horizontal">
      {{ csrf_field() }}
      <!-- Имя задачи -->
      <div class="form-group">
        <label for="task" class="col-sm-3 control-label">Задача</label>
        <div class="col-sm-6">
          <input type="text" name="name" id="task-name" class="form-control">
        </div>
      </div>

      <!-- Кнопка добавления задачи -->
      <div class="form-group">
        <div class="col-sm-offset-3 col-sm-6">
          <button type="submit" class="btn btn-default">
            <i class="fa fa-plus"></i> Добавить задачу
          </button>
        </div>
      </div>
    </form>
  </div>
  <!-- TODO: Текущие задачи -->
@endsection
```

Директива @extends сообщает Blade, что используется макет, который определен в resources/views/layouts/app.blade.php. Все содержимое между @section('content') и @endsection будет добавлено вместо строки директивы @yield('content') в макете app.blade.php.

Директива @include('common.errors') загрузит шаблон, расположенный в resources/views/common/errors.blade.php, который будет определен позже.

Теперь можно добавить код в маршрут POST /task, чтобы обработать входящие данные из формы и добавить новую задачу в БД.

5.2.3.5 Добавление задач. Проверка ввода

Теперь, когда на представлении есть форма, необходимо добавить код к маршруту POST /task, чтобы проверить входящие данные из формы и создать новую задачу.

Для этой формы создадим обязательное поле name и проверим, что оно должно содержать не более 255 символов. Если проверка не пройдёт, то перенаправим пользователя назад к URL /, а также возвратим в сессию введённые данные с указанием на ошибки:

```
Route::post('/task', function (Request $request) {
  $validator = Validator::make($request->all(), [
    'name' => 'required|max:255',
  ]);
  if ($validator->fails()) {
    return redirect('/')
```

```
        ->withInput()
        ->withErrors($validator);
    }
    // Создание задачи...
});
```

Вызов ->withErrors($validator) поместит в сессии ошибки ввода данного экземпляра, и к ним можно будет обратиться через переменную $errors в представлении.

Директива @include('common.errors') использованная в представлении, позволяет показывать ошибки ввода в одинаковом формате на всех страницах. Определим содержимое этого представления:

```
// resources/views/common/errors.blade.php
@if (count($errors) > 0)
  <!-- Список ошибок формы -->
  <div class="alert alert-danger">
    <strong>Ошибка!</strong>
    <br><br>
    <ul>
      @foreach ($errors->all() as $error)
        <li>{{ $error }}</li>
      @endforeach
    </ul>
  </div>
@endif
```

Переменная $errors доступна в любом представлении Laravel. Если не будет ошибок ввода, она просто будет пустым экземпляром ViewErrorBag.

5.2.3.6 Создание задачи

Теперь, когда обрабатывается ввод данных, можно создавать новые задачи. Как только новая задача была создана, мы перенаправим пользователя назад к URL /. Чтобы создать задачу, мы можем использовать метод save() после создания и установки свойств для новой модели Eloquent:

```
Route::post('/task', function (Request $request) {
  $validator = Validator::make($request->all(), [
    'name' => 'required|max:255',
  ]);
  if ($validator->fails()) {
    return redirect('/')
      ->withInput()
      ->withErrors($validator);
  }
  $task = new Task;
  $task->name = $request->name;
  $task->save();
  return redirect('/');
```

```
});
```

5.2.3.7 Отображение существующих задач

Во-первых, необходимо отредактировать маршрут /, чтобы передать все существующие задачи в представление. Функция view() принимает массив данных вторым параметром, который будет доступным для представления. Каждый ключ массива станет переменной в представлении:

```
Route::get('/', function () {
  $tasks = Task::orderBy('created_at', 'asc')->get();
  return view('tasks', [
    'tasks' => $tasks
  ]);
});
```

Когда данные переданы, можно обращаться к задачам в представлении tasks.blade.php и выводить их на экран в виде таблицы. Blade-конструкция @foreach позволяет кратко создавать циклы:

```
@extends('layouts.app')
@section('content')
  <!-- Форма создания задачи... -->
  <!-- Текущие задачи -->
  @if (count($tasks) > 0)
    <div class="panel panel-default">
      <div class="panel-heading">
        Текущая задача
      </div>

      <div class="panel-body">
        <table class="table table-striped task-table">
          <!-- Заголовок таблицы -->
          <thead>
            <th>Task</th>
            <th> </th>
          </thead>
          <!-- Тело таблицы -->
          <tbody>
            @foreach ($tasks as $task)
              <tr>
                <!-- Имя задачи -->
                <td class="table-text">
                  <div>{{ $task->name }}</div>
                </td>
                <td>
                  <!-- TODO: Кнопка Удалить -->
                </td>
              </tr>
            @endforeach
          </tbody>
        </table>
      </div>
    </div>
```

```
    @endif
@endsectio
```

5.2.3.8 Удаление задач

Добавим кнопку удаления к каждой строке нашего списка задач в представлении tasks.blade.php вместо комментария *<!-- TODO: Кнопка Удалить -->*. Создадим однокнопочную форму для каждой задачи в списке. После нажатия кнопки приложению будет отправляться запрос DELETE /task:

```
<tr>
  <!-- Имя задачи -->
   <td class="table-text">
    <div>{{ $task->name }}</div>
  </td>

  <!-- Кнопка Удалить -->
  <td>
    <form action="/task/{{ $task->id }}" method="POST">
      {{ csrf_field() }}
      {{ method_field('DELETE') }}
      <button>Удалить задачу</button>
    </form>
  </td>
</tr>
```

В форме кнопки удаления method объявлен как POST, несмотря на то, что мы отвечаем на запрос, используя маршрут Route::delete. HTML-формы позволяют использовать только GET и POST методы HTTP. Необходимо сымитировать запрос DELETE от формы. Для этого используется функция method_field('DELETE'). Эта функция генерирует скрытый ввод формы, который распознается Laravel и используется, чтобы переопределить вызываемый метод HTTP. Сгенерированное поле будет похоже на:

```
<input type="hidden" name="_method" value="DELETE">
```

Наконец, добавим к нашему маршруту логику удаления текущей задачи. Мы можем использовать Eloquent-метод findOrFail(), чтобы получить модель по ID или выдать ошибку 404, если модель не существует. Как только получим модель, будем использовать метод delete(), чтобы удалить запись. Как только запись удалена, перенаправим пользователя назад к URL /:

```
Route::delete('/task/{id}', function ($id) {
  Task::findOrFail($id)->delete();
  return redirect('/');
});
```

5.3 Варианты заданий

Работа выполняется на основе проекта, полученого в ходе выполнения предыдущих лабораторных работ. Небходимо доработать проект в соответствии с указаниями, данными в п.5.4 данной лабораторной работы.

5.4 Порядок выполнения работы

В ходе выполнения лабораторной работы необходимо с использованием фреймворка Laravel реализовать страницы «Редактор Блога» и «Загрузка сообщений блога» в разделе администратора персонального сайта и «Мой Блог» в пользовательском разделе.

Предусмотреть возможность редактирования и удаления записей блога на странице «Редактор Блога», реализовав валидацию средствами Laravel.

Для этого разработать: миграцию, Eloquent модель, маршруты роутинга, контроллеры, валидации, Blade-макеты пользовательской и административной части сайта, а также необходимые представления на основе имеющихся HAML/SCSS файлов (дизайн всего сайта и разработанного «Блога» должен остаться прежним).

5.5 Содержание отчета

Цель работы, порядок выполнения работы, исходные тексты разработанных миграций, моделей, контроллеров, макетов и представлений, изображения разработанных Web-страниц, выводы по результатам работы.

5.6 Контрольные вопросы

5.6.1. Для чего используется фреймворк Laravel?
5.6.2. Как осуществляется маршрутизация в Laravel?
5.6.3. Приведите пример простейшего класса контроллера в Laravel.
5.6.4. Как создать новую модель в Laravel? Приведите пример.
5.6.5. Для чего в Laravel используются миграции?
5.6.6. Расскажите о возможностях шаблонизатора Blade?

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Sublime Text [Электронный ресурс] // https://www.sublimetext.com: официальный сайт Sublime HQ Pty Ltd, 2007-2020. Дата обновления 2020. URL: https://www.sublimetext.com/3 (дата обращения: 01.06.2020).
2. Visual Studio Code [Электронный ресурс] // https://code.visualstudio.com: официальный сайт Visual Studio Code, Microsoft, 2018-2020. Дата обновления 2020. URL: https://code.visualstudio.com/docs (дата обращения: 01.06.2020).
3. Сервер Apache [Электронный ресурс] // http://httpd.apache.org: официальный сайт ПО Apache, The Apache Software Foundation, 1997-2020. Дата обновления 2020. URL: http://httpd.apache.org/download.cgi (дата обращения: 01.06.2020).
4. СУБД MySQL [Электронный ресурс] // https://www.mysql.com: официальный сайт ПО MySQL, Oracle Corporation, 2020. Дата обновления 2020. URL: https://www.mysql.com/downloads (дата обращения: 01.06.2020).
5. PHP [Электронный ресурс] // https://www.php.net: официальный сайт ПО PHP, The PHP Group, 2001-2020. Дата обновления 2020. URL: https://www.php.net/downloads (дата обращения: 01.06.2020).
6. Платформа Open Server [Электронный ресурс] // https://ospanel.io: официальный сайт ПО Open Server Panel, ospanel.io, 2010-2020. Дата обновления 2020. URL: https://ospanel.io/download (дата обращения: 01.06.2020).
7. Платформа WampServer [Электронный ресурс] // https://www.wampserver.com/ru: официальный сайт ПО WampServer, 2020. Дата обновления 2020. URL: https://www.wampserver.com/ru (дата обращения: 01.06.2020).
8. Дистрибутив XAMPP [Электронный ресурс] // https://www.apachefriends.org/index.html: официальный сайт ПО XAMPP, Apache Friends, 2020. Дата обновления 2020. URL: https://www.apachefriends.org/download.html (дата обращения: 01.06.2020).
9. Дистрибутив MAMP [Электронный ресурс] // https://www.mamp.info/ru: официальный сайт ПО MAMP, MAMP GmbH, 2020. Дата обновления 2020. URL: https://www.mamp.info/ru/downloads (дата обращения: 01.06.2020).
10. Composer – менеджер для PHP [Электронный ресурс] // https://getcomposer.org: официальный сайт ПО Composer, Nils Adermann, Jordi Boggiano, 2020. Дата обновления 2020. URL: https://getcomposer.org/download (дата обращения: 01.06.2020).

11. Laravel, фреймворк для PHP [Электронный ресурс] // https://laravel.com: официальный сайт ПО Laravel, Laravel LLC, 2011-2020. Дата обновления 2020. URL: https://laravel.com/docs/7.x/homestead (дата обращения: 01.06.2020).
12. Расширение PHP-GTK [Электронный ресурс] // gtk.php.net: официальный сайт, PHP-GTK Team, 2001-2016. Дата обновления 2016. URL: https://www.php.net/downloads (дата обращения: 01.06.2020).
13. Стандарты PSR [Электронный ресурс] // https://www.php-fig.org: сообщество разработчиков PHP, PHP FIG, 2020. Дата обновления 2020. URL: https://www.php-fig.org/psr (дата обращения: 01.06.2020).
14. Yii-фреймворк [Электронный ресурс] // https://www.yiiframework.com: официальный сайт, Yii, 2008-2020. Дата обновления 2020. URL: https://www.yiiframework.com/doc/guide/2.0/ru (дата обращения: 01.06.2020).
15. Laravel по русски [Электронный ресурс] // https://laravel.ru: русскоязычный сайт Laravel. 2006-2019. Дата обновления 2019. URL: http://laravel.ru (дата обращения: 01.06.2020).
16. Zend-фреймворк для PHP [Электронный ресурс] // https://framework.zend.com: официальный сайт Zend, Zend by Perforce, 2006-2020. Дата обновления 2020. URL: https://framework.zend.com/learn (дата обращения: 01.06.2020).
17. Фаулер М. Шаблоны корпоративных приложений / (Signature Series) = Patterns of Enterprise Application Architecture (Addison-Wesley Signature Series). – М.: «Вильямс», 2012. – 544 с. – ISBN 978-5-8459-1611-2.
18. Описание Fetch API [Электронный ресурс] // https://whatwg.org: сообщество WHATWG, WHATWG, 2018-2020. Дата обновления 2020. URL: https://fetch.spec.whatwg.org (дата обращения: 01.06.2020).
19. Объект Promise [Электронный ресурс] // https://developer.mozilla.org/ru: материалы для веб-разработки, Mozilla, 2005-2020. Дата обновления 2020. URL: https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Promise (дата обращения: 01.06.2020).
20. Виртуализатор VirtualBox [Электронный ресурс] // https://www.virtualbox.org: сайт ПО VirtualBox, Oracle, 2020. Дата обновления 2020. URL: https://www.virtualbox.org/wiki/Downloads (дата обращения: 01.06.2020).
21. Менеджер ВМ Vagrant [Электронный ресурс] // https://www.vagrantup.com: официальный сайт ПО Vagrant, HashiCorp, 2020. Дата обновления 2020. URL: https://www.vagrantup.com/downloads (дата обращения: 01.06.2020).

ПРИЛОЖЕНИЕ А
(справочное)
ОСНОВНЫЕ ОПЕРАТОРЫ ЯЗЫКА PHP

Таблица А.1 – Арифметические операторы

Обозначение	Название	Пример
+	Сложение	$a + $b
-	Вычитание	$a - $b
*	Умножение	$a * $b
/	Деление	$a / $b
%	Остаток от деления	$a % $b

Таблица А.2 – Строковые операторы

Обозначение	Название	Пример
.	Конкатенация (сложение строк)	$c = $a . $b (это строка, состоящая из $a и $b)

Таблица А.3 – Операторы присваивания

Обозначение	Название	Описание	Пример
=	Присваивание	Переменной слева от оператора будет присвоено значение, полученное в результате выполнения каких-либо операций или переменной/константы с правой стороны	$a = ($b = 4) +5; ($a будет равна 9, $b 4-м)
+=		Сокращение. Прибавляет к переменной число и затем присваивает ей полученное значение	$a += 5; (эквивалентно $a = $a + 5;)
.=		Сокращенно обозначает комбинацию операций конкатенации и присваивания (сначала добавляется строка, потом полученная строка записывается в переменную)	$b = “Привет”; $b .= “всем”; (эквивалентно $b = $b . “всем”;) В результате: $b=“Привет всем”

Таблица А.4 – Логические операторы

Обозначение	Название	Описание	Пример
and &&	И	$a и $b истинны (True)	$a and $b $a && $b
or ||	Или	Хотя бы одна из переменных $a или $b истинна (возможно, что и обе)	$a or $b $a || $b
xor	Исключающее или	Одна из переменных истинна. Случай, когда они обе истинны, исключается	$a xor $b
!	Инверсия (NOT)	Если $a=True, то !$a=False и наоборот	! $a

Таблица А.5 – Операторы сравнения

Обозначение	**Название**	**Пример**	**Описание**
==	Равенство	Значения переменных равны	$a == $b
===	Эквивалентность	Равны значения и типы переменных	$a === $b
!=	Неравенство	Значения переменных не равны	$a != $b
<>	Неравенство		$a <> $b
!==	Неэквивалентность	Переменные не эквивалентны	$a !== $b
<	Меньше		$a < $b
>	Больше		$a > $b
<=	Меньше или равно		$a <= $b
>=	Больше или равно		$a >= $b

Таблица А.6 – Операторы инкремента и декремента

Обозначение	**Название**	**Описание**	**Пример**
++	Пре-инкремент	Увеличивает $a на единицу и возвращает $a	++$a
	Пост-инкремент	Возвращает $a, затем увеличивает $a на единицу	$a++
--	Пре-декремент	Уменьшает $a на единицу и возвращает $a	--$a
	Пост-декремент	Возвращает $a, затем уменьшает $a на единицу	$a--

Заказ № _____ от «_____» _____2020г. Тираж _____экз.

Изд-во СевГУ