Муниципальное бюджетное учреждение
средняя общеобразовательная школа № 30 г. Пензы
(МБОУ СОШ № 30 г. Пензы)

Зайцев В. А.

Учебно-методическое пособие

Технологии создания информационных систем с WEB-интерфейсом

Пенза 2020

Содержание

Термины, сокращения и определения основных понятий

WEB – Интернет-пространство, «Всемирная паутина», распределённая система, предоставляющая доступ к связанным между собой документам, расположенным на различных компьютерах, подключённых к сети Интернет.

WEB-страница, HTML-документ – текстовый документ, написанный на специальном языке разметки, который позволяет браузеру выводить мультимедийную информацию.

WEB-ресурс, сайт – одна или несколько логически связанных между собой WEB-страниц, воспринимаемых пользователями как единое целое.

WEB-браузер – прикладное программное обеспечение для просмотра WEB-страниц.

WEB-сервер – комплект программного обеспечения, установленного на специальном компьютере в сети Интернет (сервере), принимающий запросы по протоколу HTTP от клиентов (WEB-браузеров) и выдающий им HTTP-ответы.

Хостинг, WEB-хостинг – услуга по предоставлению ресурсов для размещения информации на сервере, постоянно имеющем доступ к сети Интернет.

Информационная система (ИС) – система, предназначенная для хранения, поиска и обработки информации.

База данных (БД) – совокупность информационных материалов (данных), систематизированных таким образом, чтобы они могли быть найдены и обработаны с помощью компьютера.

Реляционная БД – это БД, представленная в виде двумерных таблиц.

Система управления базами данных (СУБД) – совокупность программных средств, обеспечивающих управление созданием и использованием баз данных.

SQL – язык запросов с помощью которых можно управлять данными в реляционной БД с помощью СУБД.

MySQL – одна из самых распространённых свободно распространяемая СУБД, используемая на большинстве WEB-ресурсов.

Интерфейс системы – способ взаимодействия пользователя системы с данными в БД.

WEB-интерфейс – графический интерфейс системы, созданный в виде WEB-ресурса.

HTML – стандартизированный язык разметки WEB-страниц.

CSS – формальный язык описания внешнего вида WEB-страницы, написанного с использованием языка разметки.

PHP – один из самых распространённых языков программирования, с помощью которого создаются динамические WEB-ресурсы.

phpMyAdmin – веб-интерфейс для администрирования СУБД MySQL.

FTP – протокол передачи файлов по сети Интернет.

FTP-сервер – комплект программного обеспечения, установленного на специальном компьютере в сети Интернет (сервере), позволяющий передавать информацию по протоколу FTP.

Введение

В наше время сложно представить себе мир без Интернета. Все сферы деятельности человека обращены к глобальной сети и тесно с ней взаимосвязаны. Учебные учреждения используют электронные журналы и дневники, магазины организуют свои системы доставки и Интернет-магазины, банки оперируют электронными деньгами. Существуют онлайн-музеи, галереи, библиотеки и т. д. Почти все люди имеют страницы в социальных сетях, а некоторые даже личные блоги и сайты.

Данное пособие научит вас создавать информационные системы в виде Web-сайтов. Важно понимать, что WEB-сайт – это лишь форма выражения, а главное в системе – это её идея, функционал, назначение. Хорошая идея для сайта – залог его успеха у пользователей.

Для примера создания системы решено было выбрать систему тестирования. Перед нами не стоит цель создать идеальную, многофункциональную, защищённую систему – мы всего лишь учимся. Если ли вы захотите создавать реальные системы и это станет вашей работой, то, безусловно, научитесь создавать многофункциональный интерфейс, защищать передаваемые данные, делать удобный и красивый дизайн.

Предполагается также, что вы:

- имеете базовые знания в HTML и CSS для создания оформления WEB-страниц;
- знаете, как работать с базами данных в СУБД MySQL (создание таблиц и связей между ними средствами, например, phpMyAdmin);
- имеете знания о основных SQL-запросах, для работы с данными: SELECT, INSERT, UPDATE и DELETE.

А если у вас есть определённые знания языка PHP, то Вам намного легче будет осваивать предложенный материал.

Итак, приступим!

1. Описание создаваемой системы

Далее будет рассмотрено создание информационной системы «Тесты» с графическим интерфейсом в виде WEB-приложения. В качестве системы хранения информации будет использоваться реляционная база данных.

Что представляет из себя система «Тесты»?

На главной странице системы будут присутствовать форма авторизации пользователей, список категорий тестов (то есть тесты будут сгруппированы по какому-либо признаку, например, по учебному предмету).

Функции, создаваемой системы:

- регистрация и авторизация пользователей с возможностью разграничения их прав по работе с информацией в системе;
- вывод необходимой информации (списка категорий, списка тестов в категории, страницы теста с вопросами и вариантами ответов, страницы результатов);
- добавление, редактирование и удаление элементов системы (тесты, вопросы, ответы);
- подсчёт результата прохождения теста.

2. Инструментарий

Перед началом разработки необходимо выбрать язык программирования, СУБД, вспомогательные программы и необходимые для работы библиотеки.

Мы создаём систему с WEB-интерфейсом, и поэтому нам понадобиться WEB-сервер, СУБД и FTP-сервер. Всё это можно получить двумя способами:

- арендовать готовый WEB-хостинг;
- развернуть собственный WEB-сервер со всем необходимым.

Будем исходить из того, что у нас уже есть: пустая база данных, FTP-доступ к папке нашего сайта на сервере и адрес в сети Интернет (или локальной сети), по которому мы можем его посмотреть.

Нам понадобится:

1. Программа для редактирования текстовых документов. Можно пользоваться обычным блокнотом, но это неудобно, поэтому рекомендуется установить специализированный редактор с подсветкой синтаксиса и нумерацией строк (например, Notepad++).

2. Программа FTP-клиент для доступа к нашей папке на сервере (например, FileZilla).

3. Для упрощения работы с базой данных, используем графическую оболочку (на большинстве хостингов имеется инструмент phpMyAdmin).

4. Интерпретатор языка PHP (входит в состав почти любого WEB-сервера).
5. PHP-класс для безопасной и удобной работы с базой данных «SafeMySQL».

3. Разработка информационной модели системы «Тесты»

База данных – это основа любой системы. Она должна быть продумана изначально и для этого нужно представлять себе, как должна работать система.

В нашей системе, как и почти в любой ИС, должны быть пользователи, одни из них создают тесты и управляют системой, другие непосредственно являются потребителями, то есть проходят тестирование. Соответственно в системе должны быть две связанные таблицы: «Группы пользователей», назовём её **`groups`**, и «Пользователи», назовём её **`users`**. Групп пользователей достаточно двух – «Администратор» и «Пользователь». В таблице пользователей должны быть: имя пользователя, логин и пароль для авторизации в системе, а также ссылка на идентификатор группы, которая будет определять к какой группе он относиться.

`groups`

Имя	Тип	Сравнение	Атрибуты	Null	По умолчанию	Дополнительно
id	int(11)			Нет	*Нет*	AUTO_INCREMENT
name	varchar(100)	utf8_general_ci		Нет	*Нет*	

`users`

Имя	Тип	Сравнение	Атрибуты	Null	По умолчанию	Дополнительно
id	int(11)			Нет	*Нет*	AUTO_INCREMENT
name	varchar(100)	utf8_general_ci		Нет	*Нет*	
login	varchar(100)	utf8_general_ci		Нет	*Нет*	
pass	varchar(64)	utf8_general_ci		Нет	*Нет*	
_group	int(11)			Нет	*Нет*	

Далее обратимся к системе хранения самих тестов. Каждый знает, что не хорошо, когда вещи разбросаны, где попало, в огромной куче никогда нельзя быстро найти то, что нужно. Поэтому чтобы было удобнее распределим все тесты по категориям. Это, например, может быть разбиение на учебные предметы (или любой другой принцип группировки тестов). За категории будет отвечать таблица **`categories`**, в которой нам достаточно названия и краткого описания.

Имя	Тип	Сравнение	Атрибуты	Null	По умолчанию	Дополнительно
id	int(11)			Нет	*Нет*	AUTO_INCREMENT
name	varchar(100)	utf8_general_ci		Нет	*Нет*	
info	text	utf8_general_ci		Нет	*Нет*	
_parent	int(11)			Да	*NULL*	

> *Также в таблице категорий будет присутствовать ссылка на саму себя, это предусмотрено для тех, кто захочет попробовать самостоятельно реализовать вложенные категории (по принципу папок на компьютере в операционной системе Windows).*

Таблица тестов будет называться **`tests`** и включать в себя следующие данные: заголовок теста, краткая информация о нём, дата и время создания, ссылка на категорию, в который он будет размещён, а также ссылка на идентификатор пользователя, который его создал.

Имя	Тип	Сравнение	Атрибуты	Null	По умолчанию	Дополнительно
id	int(11)			Нет	*Нет*	AUTO_INCREMENT
title	varchar(150)	utf8_general_ci		Нет	*Нет*	
info	text	utf8_general_ci		Нет	*Нет*	
date_add	datetime			Нет	*Нет*	
_user	int(11)			Нет	*Нет*	
_category	int(11)			Нет	*Нет*	

Таблицу вопросов назовём **`questions`**, в ней будет поле для текста вопроса и ссылка на тест, к которому относится вопрос.

Имя	Тип	Сравнение	Атрибуты	Null	По умолчанию	Дополнительно
id	int(11)			Нет	*Нет*	AUTO_INCREMENT
_test	int(11)			Нет	*Нет*	
info	text	utf8_general_ci		Нет	*Нет*	
picture	varchar(300)	utf8_general_ci		Нет	*Нет*	

> *В таблице вопросов вы заметите ещё одно поле – для картинки. Специально для тех, кто решит самостоятельно реализовать вопросы с картинками. В таком случае сами картинки разумно загружать в папку на сервере, а в базе данных хранить только ссылку.*

Таблица вариантов ответов на вопрос пусть называется **answers** и содержит следующие поля: текст ответа, количество баллов за его выбор (неправильные варианты 0, а правильные 1 или более) и ссылку на вопрос, к которому этот вариант ответа относится.

Имя	Тип	Сравнение	Атрибуты	Null	По умолчанию	Дополнительно
id	int(11)			Нет	*Нет*	AUTO_INCREMENT
info	text	utf8_general_ci		Нет	*Нет*	
score	int(11)			Нет	*Нет*	
picture	varchar(300)	utf8_general_ci		Нет	*Нет*	
_question	int(11)			Нет	*Нет*	

Поле для картинки также для тех, кто решит самостоятельно реализовать ответы с картинками по аналогии с вопросами.

Ещё нам нужна таблица, в которой будут храниться результаты прохождения тестов пользователями. Назовём её **results**. Там будут ссылки на пользователя и тест, который он проходил, а также количество набранных баллов.

Имя	Тип	Сравнение	Атрибуты	Null	По умолчанию	Дополнительно
_user	int(11)			Нет	*Нет*	
_test	int(11)			Нет	*Нет*	
summa	int(11)			Нет	*Нет*	

Формируем связи между таблицами. Для этого поле, которое начинается с _ (например, **_user**), делаем индексируемым (то есть создаём для него индекс) и используем для связи с ключевым полем **id** родительской таблицы. Поля для связи специально названы похожим образом с теми таблицами, с которыми они связываются. Итак, создадим следующие связи:

```
users -> groups
categories -> categories
tests -> users
tests -> categories
questions -> tests
answers -> questions
results -> users
```

results -> tests

После создания связей БД готова к использованию, но для наглядности дальнейшей разработки и проверки работоспособности системы по ходу разработки её необходимо наполнить первичными «тестовыми» данными.

Минимум нужно создать 2 группы: «Администратор» и «Пользователь». Хотя бы двух пользователей, по одному в каждую группу, чтобы иметь возможность просматривать систему от имени администратора и от имени пользователя. Несколько категорий тестов, чтобы видеть, как выглядит список категорий. В одной из категорий создать несколько тестов, чтобы видеть список тестов в категории. Для одного из тестов создать несколько вопросов, и для каждого из этих вопросов создать по несколько вариантов ответа (среди которых должен быть один правильный) – это необходимо для проверки работы самого прохождения теста и подсчёта результата.

4. Работа с БД в PHP

Для безопасной и удобной работы с базой данных мы будем использовать методы PHP-класса SafeMySQL.

Необходимо поместить файл класса **safemysql.php** в корень сайта.

Подключение к БД будет выглядеть следующим образом:

```
<?php
    require_once 'safemysql.php';
    $db = new SafeMySQL(array('user'=>'mylogin', 'pass'=>'mypassword',
        'db'=>'mydbname', 'charset'=>'utf8'));
?>
```

mylogin – ваш пользователь базы данных,
mypassword – ваш пароль от базы данных,
mydbname – название базы данных.

Для удобства подключения к БД рекомендуется создать отдельный файл **setting.php**, поместить в него код подключения к БД, разместить его в корне сайта и в начале каждого файла подключать его:

```
<?php
    include_once 'setting.php';
?>
```

Важно помнить!
Все файлы должны создаваться в кодировке UTF-8 без BOM

Работа с БД осуществляется с помощью следующих методов:

`$db->query();` – используется для выполнения запросов к БД (таких как INSERT, UPDATE, DELETE и др.), возвращает mysqli resource – аналог стандартной функции mysqli_query().

`$db->getOne();` – получение скаляра, содержимого одного поля одной строки таблицы БД.

`$db->getRow();` – получение одномерного массива, содержимого одной строки таблицы БД.

`$db->getCol();` – получение одномерного массива, содержимого одного поля нескольких строк таблицы.

`$db->getAll();` – получение двумерного массива, содержимого нескольких строк таблицы БД.

`$db->getInd();` – получение двумерного массива, содержимого нескольких строк таблицы БД, индексированного значениями поля, указанного первым параметром.

`$db->getIndCol();` – получение одномерного массива, содержимого одного поля нескольких строк таблицы, индексированного полем, указанным в первом параметре. Используется для составления словарей вида key => value.

Указанные выше методы, кроме **`getInd()`** и **`getIndCol()`** используют первым параметром запрос, содержащий плейсхолдеры – специальные метки для подстановки данных, которые перечисляются по порядку в остальных параметрах метода. В методах **`getInd()`** и **`getIndCol()`** добавлено первым параметром поле для индексации данных.

Важно, чтобы типы данных в параметрах совпадали с типом указанного плейсхолдера, а количество плейсхолдеров совпадало с количеством данных в параметрах метода!

Библиотека SafeMySQL поддерживает плейсхолдеры 6 типов:

- **`?s`** – строки (а также типы, приравненные к ним: DATE, FLOAT и др);
- **`?i`** – целые числа;
- **`?n`** – имена полей и таблиц БД;
- **`?p`** – для вставки уже обработанных частей запроса, т.е. вставка без обработки!
- **`?a`** – набор значений для оператора IN (строка вида 'a','b','c');

– **?u** – набор значений для оператора SET (строка вида `field`='value',`field`='value').

Полученные с помощью методов массивы данных удобно перебирать, используя цикл **foreach**. Примеры использования:

Вывод элемента одномерного массива (слева пример без сохранения ключа, справа пример, когда ключ сохраняется, и его можно использовать внутри цикла):

```php
<?php
foreach($massive as $val) {
    echo $val;
}
?>
```

```php
<?php
foreach($massive as $key => $val) {
    echo $val;
}
?>
```

$massive – массив, **$val** – элемент массива, **$key** – ключ (индекс) элемента

Работа с элементами двумерного массива. Цикл выделяет строку, для доступа к элементам которой нужно использовать их ключи (если данные получены из БД, то это названия полей)

```php
<?php
foreach($matrix as $row) {
    echo $row['key_name1'];
    echo $row['key_name2'];
}
?>
```

$matrix – двумерный массив,
$row – строка массива, представляющая из себя одномерный массив,
key_name1 и **key_name2** – индексы элементов в строке (имена полей БД)

5. Реализация подсистемы «Пользователи»

Начать реализацию любой системы рекомендуем с разграничения прав доступа к различным её частям. Для этого в системе существуют пользователи – они позволяют однозначно определить личность вошедшего на страницы системы. Это нужно не только для разграничения прав пользования той или иной частью системы, но и в случае нашей системы «Тесты» позволяет записать результат прохождения теста на конкретного человека. У каждого пользователя системы будут свои авторизационные данные – логин и пароль, которые в реальных системах обеспечивают защиту информации.

Мы не будем останавливаться на вопросах безопасности, нам важно понять сам принцип регистрации и авторизации пользователей в системе. Будем использовать

хранение пароля в реальном виде (так как его ввёл пользователь). При этом хочется сделать оговорку, что так нельзя делать в реальных системах, т.к. существует огромная вероятность кражи данных пользователя и использования их для несанкционированных действий. В настоящих системах пароли кодируются или хэшируются и не хранятся в реальном виде ни в БД, ни на компьютере клиента.

5.1. Регистрация

Регистрацию сделаем очень простой при помощи двух файлов. Один – это форма регистрации **reg_form.php**, другой выполняет сам процесс записи данных пользователя в БД **reg_action.php**.

reg_form.php

```
<form action="reg_action.php" method="post">
    Введите имя нового пользователя: <input type="text" name="newname"><br>
    Введите логин: <input type="text" name="newlogin"><br>
    Введите пароль: <input type="password" name="newpass"><br>
    Группа пользователя: <select name="newgroup">
                            <option value="2">Пользователь</option>
                            <option value="1">Администратор</option>
                         </select><br>
    <input type="submit" name="reg" value="Добавить пользователя"/>
</form>
```

reg_action.php

```
<?php
include_once 'setting.php';
$db->query('INSERT INTO `users` SET `name`=?s, `login`=?s, `pass`=?s, `_group`=?i',
           $_POST['newname'], $_POST['newlogin'], $_POST['newpass'], $_POST['newgroup']);
?>
```

5.2. Авторизация при помощи cookie (вход)

Авторизация – это процесс идентификации пользователя в системе путём ввода авторизационных данных (логина и пароля), проверки их существования в БД и сохранения данных о пользователе в файлах cookie на определённое время, для того чтобы не приходилось вводить авторизационные данные при любом переходе между страницами.

Для осуществления этого процесса нам понадобится три файла:

- форма авторизации **auth_form.php**,
- файл проверки данных пользователя в БД и записи его данных в cookie **auth_action.php**,
- поддержка авторизации пользователя **auth.php**, который будет подключаться в начале каждой страницы (возможно есть смысл интегрировать его в файл настроек **setting.php**).

Разберём их подробнее. Форму авторизации делаем самую простую: логин, пароль и кнопка «Войти».

```
<form method="post" action="auth_action.php">
    <input type="text" name="login"><br>
    <input type="password" name="pass"><br>
    <input type="submit" value="Войти">
</form>
```

Процесс авторизации сделаем следующим: по введённому пользователем логину и паролю идёт запрос в БД. Если пользователь с такими данными нашёлся, то его логин и пароль записываем в файлы cookie на определённое время (например, 3600 мс), после чего возвращаемся на главную страницу системы.

```
$user=$db->getRow ('SELECT * FROM `users` WHERE
            `login`=?s AND `pass`=?s LIMIT 1',
            $_POST ['login'], $_POST['pass']);

if ($user) {
    setcookie('login', $user ['login'], time()+3600);
    setcookie('pass', $user ['pass'], time()+3600);
    header('location: '.$_SERVER['HTTP_REFERER']);
    exit();
}
?>
```

Идея поддержки авторизации заключается в том, что мы по данным из файлов cookie делаем запрос в БД и записываем все данные пользователя в массив **$user**, который далее можно использовать для идентификации пользователя и для определения его прав по номеру группы. Поддержку авторизации можно создать отдельным файлом и подключать его после настроек на каждой странице или интегрировать в конец файла настроек.

```
$user=$db->getRow('SELECT * FROM `users` WHERE
        `login`=?s AND `pass`=?s LIMIT 1',
        $_COOKIE ['login'], $_COOKIE ['pass']);
```

5.3. Прекращение авторизации (выход)

Сделаем кнопку выхода, которая разрушает авторизацию пользователя, очень простой и поместим в файл **logout_form.php**:

```
<form action="logout_action.php" method="post">
    <input type="submit" name="exit" value="Выйти"/>
</form>
```

Сам выход обработаем просто уничтожением данных в файлах cookie.

logout_action.php

```
<?php
setcookie('login', '', time()-60*60*24);
setcookie('pass', '', time()-60*60*24);
header("Location: ".$_SERVER['HTTP_REFERER']);
exit();
?>
```

5.4. Личный кабинет пользователя

Личный кабинет пользователя – это страничка доступная только зарегистрированным пользователям, все данные на которой зависят от личности пользователя.

Поместим туда пока просто приветствие и форму регистрации новых пользователей для администраторов. В дальнейшем туда можно будет выводить результаты тестов, а администратору – формы добавления и редактирования различных данных и другие служебные функции.

cabinet.php

```
<?php echo 'Привет, '.$user['name'];
if ($user['_group']==1) include_once 'reg_form.php'; ?>
```

Ссылку на кабинет выведем на главной странице **index.php** только для авторизованных пользователей, а неавторизованным вместо неё выведем ссылку на авторизацию.

```php
<?php
if ($user) {
    echo '<a href="cabinet.php">Личный кабинет</a>';
    include 'logout_form.php';
}else{
    echo '<a href="auth_form.php">Авторизация</a>';
}
?>
```

> *В дальнейшем ссылки можно заменить самими формами, встроив их в дизайн системы. Всё зависит от вашей задумки! Для встраивания форм используйте* ***include****.*

6. Реализация основного функционала системы

6.1. Список категорий, страница категории.

На главной странице **index.php** подключаем страницу категорий **categories.php**:

```php
<?php
include 'categories.php';
?>
```

Страница списка категорий **categories.php** формируется из запроса в БД на получение всех категорий и вывода списка категорий в виде ссылок на страницу категории:

```php
<?php
    include_once 'setting.php';
    $cats=$db->getAll("SELECT * FROM `categories`");
?>
```

```
<ul>
<?php
    foreach ($cats as $row) {
        echo '<li><a href="category.php?cat='.$row['id'].'">'
            .$row['name'].'</a></li>';
    }
?>
</ul>
```

Страница категории **`category.php`** получает данные категории и все тесты, входящие в неё по GET-параметру **`'cat'`** и выводит информацию о категории и список тестов в виде ссылок на страницу тестов. Дополнительно заготавливаем ссылки на редактирование и удаление тестов, а также ссылку на создание нового теста в этой категории:

```
<?php
    include_once 'setting.php';
    $tests=$db->getAll("SELECT * FROM `tests` WHERE
        `_category`=?i",$_GET['cat']);
    $cat=$db->getRow("SELECT * FROM `categories` WHERE
        `id`=?i",$_GET['cat']);
?>
<h1>Тесты в категории: <?php echo $cat['name'];?></h1>
<ul>
<?php
    foreach ($tests as $row) {
        echo '<li><a href="test.php?test='.$row['id'].'">'.
            $row['title'].'</a> (<a href="edit_test.php?id='.
            $row['id'].'">Изменить</a>)</li>';
    }
?>→
</ul>
<a href="new_test.php?cat=<?php echo $cat['id'];?>">Новый
тест</a>
```

6.1. Страница теста

Страница вывода теста **test.php** содержит вывод информации о тесте (заголовок, описание и т.д.), а также все вопросы теста с вариантами ответа:

```
<?php
    include_once 'setting.php';
    $test=$db->getRow("SELECT * FROM `tests` WHERE `id`=?i", $_GET['test']);
    $q=$db->getAll("SELECT * FROM `questions` WHERE `_test`=?i", $_GET['test']);
?>
<H1><?php echo $test['title'];?></H1>
<p><?php echo $test['info'];?></p>
<form method="POST" action="result_action.php">
<?php foreach ($q as $row) {
    $vars=$db->getAll("SELECT * FROM `answers` WHERE `_question`=?i", $row['id']);
    ?>
    <p><?php echo $row['info'];?></p>
    <select name="<?php echo $row['id'];?>">
    <?php foreach ($vars as $v) {?>
        <option value="<?php echo $v['id'];?>"> <?php echo $v['title'];?></option>
    <?php } ?>
    </select>
<?php } ?>
<br><br><br>
<input type="submit" value="Завершить">
</form>
```

Все вопросы собраны в одну форму для отправки данных в исполняемый файл **result_action.php**.

6.3. Сбор, подсчёт и сохранение результатов

Исполняемые файлы – это файлы, реализующие функции системы, их задача – обрабатывать полученные в виде POST-запросов данные от форм и отправлять данные в БД.

Исполняемый файл для теста **result-action.php** должен обрабатывать выбранные пользователем варианты ответов, получая их стоимость в баллах,

находить общую сумму баллов, набранных пользователем в тесте и записывать результат в таблицу результатов:

```php
<?php include_once 'setting.php';
$data = $db->getCol("SELECT `value` FROM `variants` WHERE `id` IN (?a)",
$_POST);

$s=0;
foreach ($data as $value) {
    $s=$s+$value;
}
echo 'Вы набрали: '.$s.' баллов. Молодец!';
?>
<a href="index.php">На главную</a>
```

В представленном выше исполняемом файле нет записи результата в БД, т.к. для этого должна быть реализована авторизация пользователей в системе и в форме на странице теста **test.php** должен передаваться идентификатор теста. Это можно реализовать в виде скрытого поля формы:

```
<input type="hidden" name="test" value="<?php echo
$test['id']; ?>">
```

6.4. Страница с результатами тестов в личном кабинете

В личный кабинет пользователя добавляем вывод результатов пройденных им тестов. Для этого делаем запрос в таблицу БД, где хранятся результаты тестов по конкретному пользователю. Перебираем массив результатов с помощью цикла foreach и внутри цикла делая запросы названий тестов, выводим их результаты в виде простого списка.

```php
$results=$db->getAll('SELECT * FROM `results`
                WHERE `_user`=?i',$user['id']);
echo '<h2>Результаты тестов:</h2><ul>';
foreach ($results as $res) {
    $title=$db->getOne('SELECT `title` FROM
    `tests` WHERE `id`=?i LIMIT 1',$res['_test']);
    echo '<li>'.$title.' - '.$res['summa'].'
    баллов</li>';
}
echo '</ul>';
```

> *По вашему желанию результаты можно оформить в виде таблицы или другим более интересным способом. Здесь мы выбрали самый простой способ – обычный список.*

7. Служебные функции системы

7.1. Добавление данных

Добавление нового теста в систему можно реализовать следующим образом – форма для добавления **new_test.php**:

```
<?php
    include_once 'setting.php';
    $cats=$db->getAll("SELECT * FROM `categories`");
?>
<h2>Новый тест</h2>
<form method="POST" action="new_test_action.php">
Название: <input type="text" name="title"><br>
Описание: <textarea name="info"> </textarea><br>
Категория: <select name="category">
<?php
foreach ($cats as $cat) {
?>
    <option value="<?php echo $cat['id'];?>" <?php if ($cat['id'] ==
    $_GET['cat']) echo 'selected';?>><?php echo $cat['name'];?></option>
<?php
}
?>
</select><br>
<input type="submit" value="Создать">
</form>
```

Исполняемый файл для формы **new_test_action.php**:

```
<?php
    include_once 'setting.php';

    $db->query("INSERT INTO `tests` SET `title`=?s, `info`=?s,
    `date_add`=NOW(), `_user`=?i, `_category`=?i",$_POST['title'],$_POST[
    'info'],$user['id'],$_POST['category']);
?>
<a href="index.php">На главную</a>
```

В данном случае кроме полученных из формы данных в запросе добавляется дата создания при помощи SQL-функции **NOW()** и идентификатор пользователя из

массива данных авторизованного пользователя, если авторизация на данном этапе не работает, то вместо **$user['id']** можно временно подставить идентификатор любого пользователя из таблицы БД **users**.

Подобным же образом реализуются страницы добавления вопросов к тесту и вариантов ответа к вопросу:

- ✓ форма добавление вопроса **new_question.php**

```php
<?php
include_once 'setting.php';
$test=$db->getRow('SELECT * FROM `tests` WHERE `id`=?i',$_GET['test']);
?>
<h2>Новый вопрос</h2>
<form method="POST" action="new_question_action.php">
Вопрос: <textarea name="info"> </textarea><br>
Тест: <input type="hidden" name="test" value="<?php echo $test['id'];?>">
 <?php echo $test['title']; ?><br>
<input type="submit" value="Добавить">
</form>
```

- ✓ добавление вопроса **new_question_action.php**

```php
<?php
include_once 'setting.php';

$db->query("INSERT INTO `questions` SET `info`=?s, `_test`=?i",
$_POST['info'],$_POST['test']);
?>
<a href="index.php">На главную</a>
```

- ✓ форма добавления ответа **new_answer.php**

```php
<?php
include_once 'setting.php';
$q=$db->getRow('SELECT * FROM `questions` WHERE `id`=?i',$_GET['q']);
?>
<h2>Добавление варианта ответа</h2>
<form method="POST" action="new_answer_action.php">
Текст ответа: <input type="text" name="title"><br>
Кол-во баллов: <input type="text" name="value"><br>
Вопрос: <input type="hidden" name="q" value="<?php echo $q['id'];?>">
 <?php echo $q['info']; ?><br>
<input type="submit" value="Добавить">
</form>
```

✓ добавление ответа **new_answer_action.php**

```php
<?php
    include_once 'setting.php';

    $db->query("INSERT INTO `answers` SET `title`=?s, `value`=?i,
    `_question`=?i",$_POST['title'],$_POST['value'],$_POST['q']);
?>
<a href="index.php">На главную</a>
```

Ссылки на добавление вопросов и вариантов ответа можно добавить, например, на страницу теста **test.php** в виде:

```
<a href="new_question.php?test=<?php echo $test['id'];
                ?>">Добавить вопрос</a>
<a href="new_answer.php?q=<?php echo $row['id'];
                ?>">Добавить ответ к вопросу</a>
```

В последней ссылке значение $row['id'] содержит идентификатор конкретного вопроса и, естественно, она должна быть расположена в цикле вывода вопросов около каждого вопроса.

7.2. Редактирование данных

Редактирование данных очень похоже на добавление. Формы аналогичные, только в них подставляются уже существующие данные и скрытое поле, которое содержит идентификатор изменяемых данных. В исполняемых файлах вместо запроса «INSERT» делаем запрос «UPDATE». Пример:

✓ форма редактирования теста **edit_test.php**

```php
<?php
    include_once 'setting.php';

    $cats=$db->getAll("SELECT * FROM `categories`");
    $test=$db->getRow("SELECT * FROM `tests` WHERE `id`=?i",
    $_GET['id']);
?>
<h2>Редактировать тест</h2>
<form method="POST" action="edit_test_action.php">
<input type="hidden" name="id" value="<?php echo $test['id']; ?>
">
```

```
Название: <input type="text" name="title" value="<?php echo
$test['title']; ?>"><br>
Описание: <textarea name="info"><?php echo $test['info']; ?>
</textarea><br>
Категория: <select name="category">
<?php foreach ($cats as $cat) { ?>
    <option <?php if ($cat['id']==$test['_category']) echo
    'selected';?> value="<?php echo $cat['id'];?>" <?php if (
    $cat['id']==$_GET['cat']) echo 'selected';?>><?php echo $cat
    ['name'];?></option>
<?php }?>
</select><br>
<input type="submit" value="Изменить">
</form>
```

- ✓ редактирование теста **edit_test_action.php**

```
<?php
    include_once 'setting.php';

    $db->query("UPDATE `tests` SET `title`=?s, `info`=?s, `_category`=?i
    WHERE `id`=?i",$_POST['title'],$_POST['info'],$_POST['category'],
    $_POST['id']);
?>
<a href="index.php">На главную</a>
```

- ✓ форма редактирование вопроса **edit_question.php**

```
<?php
    include_once 'setting.php';
    $q=$db->getRow('SELECT * FROM `questions` WHERE `id`=?i',$_GET['q']);
?>
<h2>Изменить вопрос</h2>
<form method="POST" action="edit_question_action.php">
Вопрос: <textarea name="info"><?php echo $q['info'];?></textarea><br>
<input type="hidden" name="q" value="<?php echo $q['id'];?>"><br>
<input type="submit" value="Применить">
</form>
```

- ✓ редактирование вопроса **edit_question_action.php**

```php
<?php
include_once 'setting.php';

$db->query("UPDATE `questions` SET `info`=?s WHERE `id`=?i",
$_POST['info'],$_POST['q']);
?>
<a href="index.php">На главную</a>
```

Подобным же образом можно реализовать редактирование ответов.
edit_answer.php
edit_answer_action.php

Ссылку на редактирование вопроса размещаем в test.php, например, рядом с вопросом. В ней обязательно должен быть идентификатор вопроса:

```
<a href="edit_question.php?q=<?php echo $row['id'];
?>">Изменить вопрос</a>
```

7.3. Удаление данных

Удаление реализуется проще всего. Для этого достаточно поместить в нужном месте ссылку на исполняемый файл, в которой GET-параметром отправить идентификатор удаляемого объекта (тест, вопрос или ответ). Например:

```
<a href="delete_question.php?q=<?php echo $row['id'];
?>">Удалить вопрос</a>
```

А в исполняемом файле выполнить запрос «DELETE»:

```
$db->query("DELETE FROM `questions` WHERE `id`=?i",
$_GET['q']);
```

То же самое проделать для удаления ответов к вопросам и самих тестов.

8. Тестирование и доработка системы

После завершения создания системы необходимо полностью проверить её функционал:

- регистрацию новых пользователей;
- авторизацию и выход из системы;
- какие данные доступны администратору и обычному пользователю;
- прохождение теста (правильно ли считается результат, записываются ли результаты в БД, выводятся ли они в личном кабинете пользователя);
- создание нового теста, вопросов, вариантов ответов (корректно ли они записываются в БД);
- редактирование данных (теста, вопросов, вариантов ответов);
- удаление данных (теста, вопросов, вариантов ответов);
- другие функции, которые мы создавали.

Системе нужен графический дизайн, создайте его самостоятельно, для этого Вам понадобится знание HTML, CSS, блочной верстки макетов сайтов.

Список используемых ресурсов

1. Википедия – свободная энциклопедия: [Электронный ресурс]. URL: https://ru.wikipedia.org. (Дата обращения: 16.06.2020).
2. Класс для безопасной и удобной работы с MySQL: [Электронный ресурс]. URL: http://phpfaq.ru/safemysql. (Дата обращения: 16.06.2020).

Ссылки на используемые инструменты и полезные ресурсы

1. PHP-класс SafeMySQL: https://github.com/colshrapnel/safemysql
2. Удобный текстовый редактор: https://notepad-plus-plus.org/downloads
3. FTP-клиент: https://filezilla-project.org
4. Комплект серверного ПО для установки в локальной сети: https://ospanel.io
5. Справочник по HTML и CSS: http://htmlbook.ru
6. Справочник по языку PHP: https://php.ru/manual

Для заметок:

Для заметок:

Зайцев Владимир Анатольевич

Технологии создания информационных систем с WEB-интерфейсом

Учебно-методическое пособие

Создано в рамках проекта:
«Технологии виртуальной и дополненной реальности в преподавании предметной области «Информатика» мероприятия «Развитие и распространение лучшего опыта в сфере формирования цифровых навыков образовательных организаций, осуществляющих образовательную деятельность по общеобразовательным программам, имеющих лучшие результаты в преподавании предметных областей «Математика», «Информатика» и «Технология» в рамках федерального проекта «Кадры для цифровой экономики» национальной программы «Цифровая экономика»