Урок 6

Интерактивность: методы передачи данных GET и POST, работа с формами и пользовательскими данными

Изучение методов обмена информацией посредством протокола HTTP и его встроенных методов. Обработка пользовательских форм, сохранение данных на сервере

До этого занятия мы всё время использовали PHP исключительно как средство генерации страниц на основании статических данных, которые довольно редко и неудобно меняются с течением времени. С одной стороны, это даёт понимание основ языка, с другой – это скучно и несовременно. Сайт должен взаимодействовать с пользователем, реагировать на его действия. При этом нужно понимать, что каждый пользователь ведёт себя по-своему. На сегодняшнем занятии мы научимся делать сайт как интерактивным, так и безопасным.

Основные методы передачи данных на сервер

Для начала вспомним принципы передачи информации в сети Internet.

Одним из основополагающих протоколов является протокол HTTP (HyperText Transfer Protocol). При создании он задумывался как протокол передачи гипертекстовых документов (то есть простых HTML-файлов).
Протокол HTTP предполагает использование клиент-серверной структуры передачи данных. Клиентское приложение (в общем случае – браузер) отвечает за формирование запроса и отправку его серверу. Сервер, в свою очередь, имеет возможность обработки запроса при помощи прикладного ПО (Apache, NGINX). Сформированный ответ возвращается клиенту, и клиентское приложение может продолжать работу.
Стандартная задача для HTTP – это обеспечение обмена данными между браузером и веб-сервером. Именно HTTP обеспечивает работу современного Internet.
Однако на этом функционал HTTP не заканчивается, предоставляя возможность передавать информацию в формате других протоколов прикладного уровня: SOAP, XML-RPC и WebDAV. Тогда о протоколе HTTP говорят, что он используется как «транспорт». API многих продуктов также подразумевает использование HTTP для передачи данных – в таком случае они могут иметь любой формат, например, XML или JSON.
В большинстве случаев передача данных по протоколу HTTP выполняется при помощи TCP/IP-соединения. Это накладывает на серверное ПО требование открыть TCP-порт под номером 80. Вместе с тем допустимо использование и других портов.
Для формирования HTTP-запроса требуется описать стартовую строку, а также задать как минимум один заголовок – это заголовок Host, который обязателен и должен присутствовать в каждом запросе. Мы помним, что преобразование доменного имени в IP-адрес осуществляется на стороне клиента. Таким образом, при открытии TCP-соединения удалённый сервер совершенно ничего не знает о том, какой именно адрес использовался для соединения. На одном сервере могут находиться различные ресурсы. Все они будут слушать порт 80. Именно заголовок Host будет говорить серверу о том, к какому именно ресурсу происходит обращение.

Стартовая строка запроса для HTTP 1.1 составляется по следующей схеме:

- метод URI HTTP/Версия;
- такой запрос вызовет главную страницу сайта;
- GET/HTTP/1.1.

Разберём этот вызов подробнее.

Метод – это последовательность из любых символов, кроме управляющих и разделителей, определяющая операцию, которую нужно осуществить с указанным ресурсом. Спецификация HTTP

1.1 не ограничивает количество возможных используемых методов, но для достижения соответствия общепринятым стандартам и соблюдения совместимости с широким перечнем ПО применяется лишь небольшой перечень стандартных методов. Нас будут интересовать четыре из них:

1. GET.
2. POST.
3. PUT.
4. DELETE.

GET используется для запроса содержимого указанного ресурса. При помощи метода GET можно также начать какой-либо процесс. В этом случае в тело ответного сообщения следует включить информацию о ходе выполнения процесса.

Клиент может передавать параметры выполнения запроса в URI целевого ресурса после символа «?»:
GET /path/resource?param1=value1&param2=value2 HTTP/1.1

В данном запросе скрипту передано 2 параметра. Зачем они вообще могут быть нужны скрипту?
Динамическая страница, в отличие от статической, может формировать для клиента различные ответы, зависящие от определённых условий. Параметры же такими условиями и являются.
Согласно спецификации HTML GET является идемпотентным – это значит, что сервер должен возвращать одни и те же ответы на идентичные запросы (при условии, что ресурс не изменился между ними по иным причинам). Такая особенность позволяет кэшировать ответы, снижая нагрузку на сеть.
POST применяется при передаче пользовательских данных выбранному ресурсу. К примеру, в блогах посетители могут вводить свои комментарии к записям в HTML-форму, после чего они передаются серверу методом POST, а сам сервер, в свою очередь, помещает их на страницу. Передаваемые данные включаются в тело запроса.
Многократное повторение одних и тех же запросов POST может возвращать разные результаты (например, после каждой отправки комментария будет появляться очередная копия этого комментария). Как следствие, сообщение ответа сервера на выполнение метода POST не кэшируется.
При результате выполнения 200 (Ok) в тело ответа следует включить сообщение об итоге выполнения запроса. Если был создан ресурс, то серверу следует вернуть ответ 201 (Created) с указанием URI нового ресурса в заголовке Location.
PUT применяется для загрузки содержимого запроса на указанный в запросе URI. Если по заданному URI ресурса не существует, сервер создаёт его и возвращает статус 201 (Created). Если же ресурс был изменён, сервер возвращает 200 (Ok) или 204 (No Content). Сервер не должен игнорировать некорректные заголовки Content-*, передаваемые клиентом вместе с сообщением. Если какой-то из этих заголовков не может быть распознан или недопустим при текущих условиях, необходимо вернуть код ошибки 501 (Not Implemented).
Фундаментальное различие методов POST и PUT заключается в понимании предназначений URI ресурсов. Метод POST предполагает, что по указанному URI будет производиться обработка передаваемого клиентом содержимого. Используя PUT, клиент предполагает, что загружаемое содержимое соответствует находящемуся по данному URI ресурсу.
Сообщения ответов сервера на метод PUT не кэшируются.
DELETE Удаляет указанный ресурс. В современных приложениях используются в основном GET (для получения информации) и POST (для её передачи на обработку на стороне сервера).

Применение в PHP

На прошлом уроке мы уже столкнулись с работой с GET-параметрами, но применяли их, не задумываясь о принципах работы. Мы даже реализовали фотогалерею и новостную ленту. Теперь же мы разберём всё подробно.

1. Создадим модуль отправки HTML-формы для сбора отзыва о сайте.
2. Напишем обработчики отправки данных.
3. Дадим возможность сохранения отзывов в БД.
4. Позаботимся о безопасности.

Итак, для работы с переданными от пользователя данными нам понадобится один из суперглобальных массивов. Поскольку мы выбрали метод передачи данных POST, массив будет соответствующий.
Массив $_POST ассоциативный. Ключами в нём являются значения атрибутов name у элементов в форме.
Важный момент, который немного усложняет логику скрипта, заключается в том, что на данную страничку пользователь может попасть как методом POST, так и методом GET. В случае, если он нажал на кнопку отправить, мы имеем метод POST, так как указали это на форме в атрибуте method. Во всех остальных вариантах (набрал адрес в строке браузера, перешёл по ссылке) пользователь попадает на страницу методом GET. Поэтому в нашем скрипте мы проверяем, каким именно методом пользователь попал на нашу страничку, корректируя содержимое в зависимости от этого. Делается это с помощью специальной функции isset(). Она возвращает true в том случае, если элемент в массиве существует, и false – в противном.
Если мы определили, что запрос осуществлялся методом POST, то обрабатываем переданные данные и сохраняем их в БД, возвращая пользователю результат операции. Здесь выгода PHP очевидна, так как без него мы просто не смогли бы обработать отправку формы.

Обратите внимание на особенность обработки строк:

```
$feedback_body = mysqli_real_escape_string($db_link,
(string)htmlspecialchars(strip_tags($_POST['review'])));
```

Эта громоздкая конструкция позволяет нам избежать вредоносных SQL-инъекций.
SQL-инъекция – это атака на базу данных, которая позволит выполнить некоторое действие, не входящее в планы создателя скрипта. Как такое возможно? Начнём с псевдокода.
Отец написал маме записку, чтобы она дала Васе 100 рублей, и положил её на стол. Переработав это в шуточный SQL-язык, мы получим:

```
ДОСТАНЬ ИЗ кошелька 100 РУБЛЕЙ И ДАЙ ИХ Васе
```

Отец плохо написал записку (корявый почерк) и оставил её на столе. Записку увидел брат Васи – Петя. Он проявил смекалку, дописав в конце «ИЛИ Пете», и запрос получился таким:

```
ДОСТАНЬ ИЗ кошелька 100 РУБЛЕЙ И ДАЙ ИХ Васе ИЛИ Пете
```

После прочтения записки мама решила, что отдаст 100 рублей Пете, поскольку Васе она уже давала деньги вчера. Не фильтруя данные (мама еле разобрала почерк), финансы ушли хакеру, а не целевому пользователю.
Инъекция появляется из входящих данных, которые не фильтруются. Для их фильтрации мы используем несколько слоёв защиты:

1. strip_tags – удаляет HTML и PHP теги. Честно говоря, это мало относится к SQL-инъекциям, но атаку через форму без этого тега провести можно.
2. htmlspecialchars – производятся следующие преобразования:
    a. '&' (амперсанд) преобразуется в '&'
    b. '"' (двойная кавычка) преобразуется в '"' в режиме ENT_NOQUOTES is not set.
    c. ''' (одиночная кавычка) преобразуется в ''' (или ') только в режиме ENT_QUOTES.
    d. '<' (знак "меньше чем") преобразуется в '<'
    e. '>' (знак "больше чем") преобразуется в '>'
3. (string) – строго приводим тип к строке.
4. mysqli_real_escape_string – самый мощный инструмент, экранирует специальные символы в строке для использования в SQL выражении, используя текущий набор символов соединения.

Таким образом, наш код становится безопасным и не даёт злоумышленнику вставить в форму что попало. Разумеется, сама по себе форма далека от идеала безопасности, однако мы рассмотрели один из наиболее важных аспектов серверной безопасности.

Реализация CRUD-пакета действий

Теперь реализуем CRUD-пакет работы с отзывами на практике. Что такое CRUD?

CRUD (сокр. от англ. create, read, update, delete – «создать, прочесть, обновить, удалить») – сокращённое именование четырёх базовых функций, используемых при работе с персистентными хранилищами данных. Создание и чтение мы уже реализовали. Теперь нужно добавить изменение и удаление отзыва.

Загрузка файлов на сервер

Ещё одним немаловажным аспектом форм является возможность загрузки файлов на сервер, позволяя делать их общедоступными, не имея прямого подключения к файловой системе.
Данная возможность позволяет загружать как текстовые, так и бинарные файлы. С помощью PHP-функций вы получаете полный контроль над тем, как загружается файл, и что должно быть сделано после его загрузки.
Страница для загрузки файлов может быть реализована при помощи специальной формы, которая выглядит примерно так:

```
<!-- Тип кодирования данных, enctype, ДОЛЖЕН БЫТЬ указан ИМЕННО так -->
<form enctype="multipart/form-data" action="__URL__" method="POST">
    <!-- Поле MAX_FILE_SIZE должно быть указано до поля загрузки файла -->
    <input type="hidden" name="MAX_FILE_SIZE" value="30000" />
    <!-- Название элемента input определяет имя в массиве $_FILES -->
    Отправить этот файл: <input name="userfile" type="file" />
    <input type="submit" value="Send File" />
```

```
</form>
```

В приведённом выше примере __URL__ необходимо заменить ссылкой на PHP-скрипт.
Скрытое поле MAX_FILE_SIZE (значение необходимо указывать в байтах) должно предшествовать полю для выбора файла, и его значение является максимально допустимым размером принимаемого файла в PHP. Рекомендуется всегда использовать эту переменную, так как она предотвращает тревожное ожидание пользователей при передаче огромных файлов только для того, чтобы узнать, что файл слишком большой и передача фактически не состоялась. Помните, обойти это ограничение на стороне браузера можно достаточно просто, следовательно, вы не должны полагаться на то, что все файлы большего размера будут блокированы при помощи этой возможности. По большому счёту, это удобная возможность для пользователей клиентской части вашего приложения. Тем не менее настройки PHP (на сервере) касательно максимального размера обойти невозможно.
Глобальный массив $_FILES содержит всю информацию о загруженных файлах. Его содержимое для нашего примера приводится ниже. Обратите внимание, что здесь предполагается использование имени userfile для поля выбора файла, как и в приведенном выше примере. На самом деле имя поля может быть любым.

$_FILES['userfile']['name'] – оригинальное имя файла на компьютере клиента.
$_FILES['userfile']['type'] – Mime-тип файла, в случае, если браузер предоставил такую информацию. Пример: "image/gif". Этот mime-тип не проверяется в PHP, так что не полагайтесь на его значение без проверки.
$_FILES['userfile']['size'] – размер в байтах принятого файла.
$_FILES['userfile']['tmp_name'] – временное имя, с которым принятый файл был сохранен на сервере.
$_FILES['userfile']['error'] – код ошибки, которая может возникнуть при загрузке файла.

На сервере скрипт обработки загрузки будет выглядеть следующим образом:

```php
<?php
$uploaddir = WWW_ROOT . '/img/uploads/';
$uploadfile = $uploaddir . basename($_FILES['userfile']['name']);
echo '<pre>';
if (move_uploaded_file($_FILES['userfile']['tmp_name'], $uploadfile)) {
    echo "Файл корректен и был успешно загружен.\n";
} else {
    echo "Возможная атака с помощью файловой загрузки!\n";
}
?>
```

PHP-скрипт, принимающий загруженный файл, должен реализовывать логику, необходимую для определения дальнейших действий над принятым файлом. Например, вы можете проверить переменную $_FILES['userfile']['size'], чтобы отсечь слишком большие или слишком маленькие файлы.
Также можно использовать переменную $_FILES['userfile']['type'] для исключения файлов, которые не соответствуют критерию касательно типа файла. Принимайте во внимание, что это поле полностью контролируется клиентом, поэтому используйте его только в качестве первой из серии проверок. Можно использовать $_FILES['userfile']['error'] и коды ошибок при реализации вашей логики, чтобы оповещать пользователя о возникающих в процессе работы ошибках.
Вне зависимости от того, какую модель поведения Вы выбрали, нужно удалить файл из временной папки или переместить его в другую директорию.

В случае, если при отправке формы файл выбран не был, PHP установит переменную $_FILES['userfile']['size'] значением 0, а переменную $_FILES['userfile']['tmp_name'] – пустой строкой none.
В случае, если принятый файл не был переименован или перемещён, по окончанию работы скрипта он будет автоматически удалён из временной папки.

Итоги

На сегодняшнем занятии мы научили наши страницы реагировать на действия пользователей, создавая скрипты, которые обрабатывают различные входящие HTTP-запросы.
Более того, PHP даёт возможность не только реагировать на действие пользователей, но и сохранять переданные данные и файлы, что открывает перед нами новые горизонты возможностей.
И самое главное – всегда помните о безопасности вашего кода!

Практическое задание

1. Создать форму-калькулятор операциями: сложение, вычитание, умножение, деление. Не забыть обработать деление на ноль! Выбор операции можно осуществлять с помощью тега <select>.
2. Создать калькулятор, который будет определять тип выбранной пользователем операции, ориентируясь на нажатую кнопку.
3. Добавить функционал отзывов в имеющийся у вас проект.
4. Создать страницу каталога товаров.
    a. Товары хранятся в БД (структура прилагается).
    b. Страница формируется автоматически.
    c. По клику на товар открывается карточка товара с подробным описанием.
    d. Подумать, как лучше всего хранить изображения товаров.
5. *Написать CRUD-блок для управления выбранным модулем через единую функцию (doFeedbackAction()).

Дополнительные материалы

1. HTTP - https://ru.wikipedia.org/wiki/HTTP#.D0.9C.D0.B5.D1.82.D0.BE.D0.B4.D1.8B
2. Идемпотентность - https://ru.wikipedia.org/wiki/%D0%98%D0%B4%D0%B5%D0%BC%D0%BF%D0%BE%D1%82%D0%B5%D0%BD%D1%82%D0%BD%D0%BE%D1%81%D1%82%D1%8C

Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

1. Изучаем PHP и MySQL – Линн Бейли, Майкл Моррисон.
2. “PHP 5 в подлиннике” – Д.Котеров.