Безопасная обработка данных

Введение

Защита пользовательских данных является главным приоритетом современной разработки приложений. В условиях роста утечек данных и кибератак компании и разработчики должны предпринять все необходимые шаги для обеспечения безопасности конфиденциальной информации пользователей.

Неспособность защитить пользовательские данные может иметь катастрофические последствия – от серьезных финансовых потерь и ущерба репутации бренда до потенциальных юридических последствий из-за несоблюдения отраслевых правил и стандартов.

Обеспечение безопасности веб-приложений — это постоянный процесс, требующий внимания и усилий со стороны разработчиков, администраторов и тестировщиков. Лучшие практики и знание уязвимостей позволяют создавать надежные и защищенные веб-приложения, способные устоять перед современными угрозами внешнего вторжения.

Безопасность приложений – это безопасность обрабатываемых им данных. Безопасность данных в свою очередь, это контроль **происхождения** этих данных, их **санитизация** (очищение) и **валидация** (проверка данных на соответствие некоторым требованиям).

В текущем уроке разберем, какие данные и каким образом нужно обрабатывать и как быть с двоичными файлами данных, с которыми взаимодействуют серверные сценарии.

Пример данных не требующих санитизации. Только **валидация**:

- **Пароль.** Пример: *QwertY@2001#*.
- **Логин**. Пример: *moderator*.
- **Владелец банковской карты.** Пример: *PANKRATOV INOKENTIY*.
- **Номер банковской карты**. Пример: *1254 4568 4687 2135*.
- и др.

> ***Примечание.*** Никаких импровизаций со стороны пользователя здесь не допускается. **Только** набор **разрешенных** символов.

Пример данных, требующих как **сантизации**, так и **валидации**:

- **E-mail**. Пример: *master@ya.com.ru*.
- **Заголовок статьи**. Пример: *Как работать в FinalCut Pro^{x}: инструкция*.
- **Курсы повышения квалификации:** Пример: *Big Data и Data Science: начни погружение с нуля**.
- **Почтовый адрес**. Пример: *Печорский пр-т, дом 39, корп. 1^{б}, кв 123*.

- и др.

> ***Примечание.*** Допускаются некоторые теги и специальные символы, при этом используются только символы русского / латинского алфавита (зависит от политики ресурса).

Пример данных, требующих только **сантизации**:

- **Тизер к статье**.
- **Контент статьи**.
- **Описание товара**.
- **Примечание**.
- и др.

> ***Примечание.*** Невозможно предсказать используемые символы в тизере или контенте статьи. Это могут быть символы русского, английского, арабского алфавитов, специальные символы, валюта и прочее.

Безопасная обработка текстовых и числовых данных

Теперь рассмотрим коды демонстрационных примеров приведенных ситуаций.

Данные, требующие только валидации

example_1. index.html

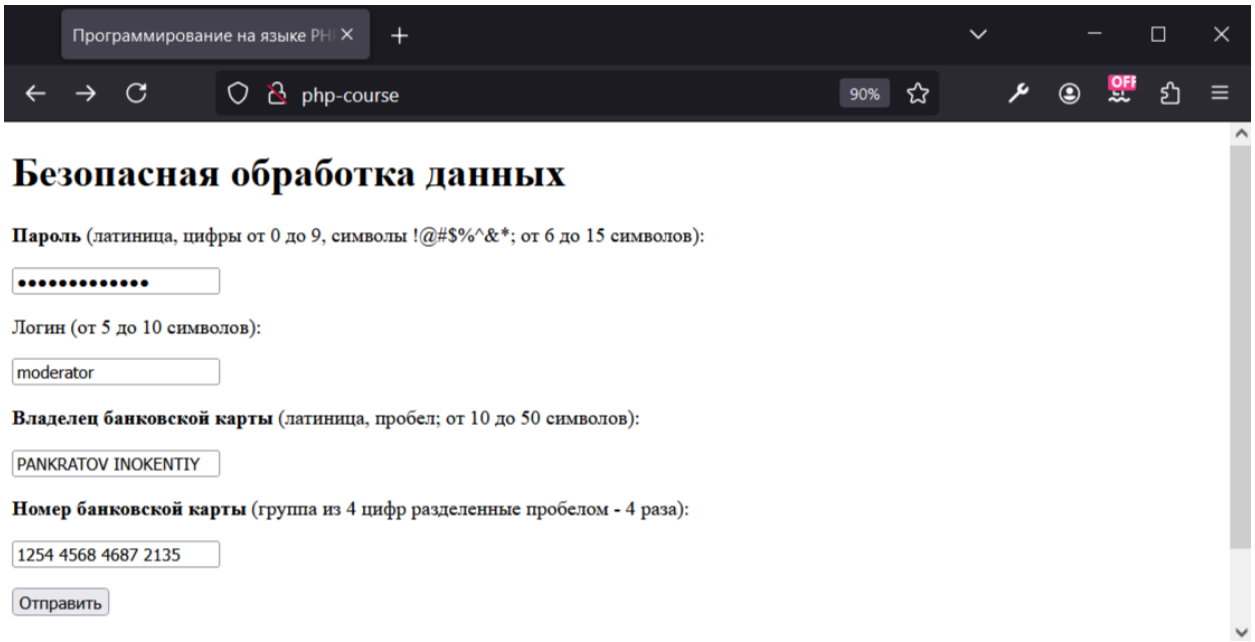
```
<form action="server.php" method="post">

    <p><b>Пароль</b> (латиница, цифры от 0 до 9, символы
    !@#$%^&*; от 6 до 15 символов):<p>

    <input type="password" name="pwd" value="Password@123*">

    <p>Логин (от 5 до 10 символов):</p>

    <input type="text" name="login" value="moderator"><p>

    <p><b>Владелец банковской карты</b> (латиница, пробел; от 10
    до 50 символов):</p>

    <input type="text" name="card_owner" value="PANKRATOV
    INOKENTIY">

    <p><b>Номер банковской карты</b> (группа из 4 цифр
    разделенные пробелом - 4 раза):</p>

    <input type="text" name="card_number" value="1254 4568 4687
    2135">

    <p><input name="submit" type="submit" value="Отправить"></p>

</form>
```

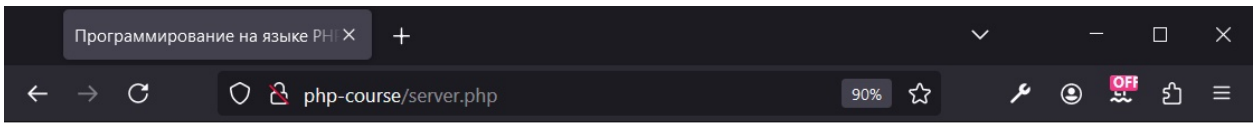
example_1. server.php

```
<?php

    // если выполнена отправка формы

    if(isset($_POST["submit"])){

        // массив для сбора ошибок

        $_ERROR_VALID = array();

        // валидация поля Пароль

        if (!preg_match('/^[0-9a-zA-Z!@#$%^&*]{6,15}$/',
```

```
        $_POST["pwd"]))
            $_ERROR_VALID[] = 'Пароль';
        // валидация поля Логин
        if (!preg_match('/^(?=.{5,10}$)[a-z\d]+[_-]?[a-z\d]+$/',
        $_POST["login"]))
            $_ERROR_VALID[] = 'Логин';
        // валидация поля Владелец банковской карты
        if (!preg_match('/^[A-Z\s]{10,50}$/',
        $_POST['card_owner']))
            $_ERROR_VALID[] = 'Владелец банковской карты';
        // валидация поля Номер банковской карты
        /*
        if (!preg_match('/^[0-9\s]{19}$/',
        $_POST['card_number']))
            $_ERROR_VALID[] = 'Номер банковской карты';
        */
        if (!preg_match('/^[0-9]{4}\s[0-9]{4}\s[0-9]{4}\s[0-
        9]{4}$/', $_POST['card_number']))
            $_ERROR_VALID[] = 'Номер банковской карты';
        // если массив ошибок не пуст
        if (count($_ERROR_VALID)){
            echo "<h3>Неверно заполненные поля формы:</h3>";
            // выводим список полей не прошедших валидацию
            foreach ($_ERROR_VALID as $val) {
                printf ('Ошибка заполнения поля: <b>%s.</b><br
                />', $val);
            }
        }
```

```php
        // если ошибок не возникло
        else {
            echo "<h2>Данные готовы для безопасной обработки:</h2>";
            echo 'Пароль: ' . $_POST["pwd"] . '<br>';
            echo 'Логин: ' . $_POST["login"] . '<br>';
            echo 'Владелец банковской карты: ' . $_POST["card_owner"] . '<br>';
            echo 'Номер банковской карты: ' . $_POST["card_number"] . '<br>';
        }
    }
?>
```





Безопасная обработка данных

Данные готовы для безопасной обработки:

Пароль: Password@123*
Логин: moderator
Владелец банковской карты: PANKRATOV INOKENTIY
Номер банковской карты: 1254 4568 4687 2135

Данные, требующие как сантизации, так и валидации

При отправке данных **поля email** хорошо бы использовать элемент **input** с типом **email**, но тогда труднее будет допустить ошибку заполнения, а она нам понадобится для тестирования.

example_2. index.html

```
<form action="server.php" method="post">

    <p><b>E-mail</b> (стандартный):</p>

    <input type="text" name="email" value="master@ya.com.ru">

    <!--<input type="email" name="email"
    value="master@ya.com.ru">-->

    <p><b>Заголовок</b> статьи:</p>

    <input type="text" name="title" value="Как работать в
    FinalCut Pro<sup>X</sup>: инструкция">

    <p><b>Курсы</b> повышения квалификации:</p>

    <input type="text" name="course" value="<b>Big Data</b> и
    <b>Data Science</b>: начни погружение с нуля*">

    <p><b>Почтовый</b> адрес:</p>

    <textarea name="addr">Печорский пр-т, дом 39, корп
```

```
    1<sup>б</sup>, кв 123</textarea><p>
    <input name="submit" type="submit" value="Отправить">
</form>
```

example_2. server.php

```
<?php
    // если выполнена отправка формы
    if (isset($_POST["submit"])){
        // массив для сбора ошибок
        $_ERROR_VALID = array();
        // очищаем данные
        $email = filter_var($_POST['email'],
        FILTER_SANITIZE_EMAIL);
        $title = strip_tags($_POST['title'], ["sup", "sub", "u",
        "i"]);
        $course = htmlentities($_POST['course']);
        // проверяем данные на валидность
        if (!filter_var($email, FILTER_VALIDATE_EMAIL))
            $_ERROR_VALID[] = "Адрес email";
        if (!preg_match('/^[a-zA-Za-яёА-ЯЁ\s\/<>:-]{1,100}$/u',
        $title))
            $_ERROR_VALID[] = 'Заголовок статьи';
        if (!preg_match('/^[a-zA-Za-яёА-ЯЁ\s\*\/&;:-]{1,100}$/u',
        $course))
            $_ERROR_VALID[] = 'Курсы повышения квалификации';
        // если массив ошибок не пуст
        if (count($_ERROR_VALID)){
            echo "<h3>Неверно заполненные поля формы:</h3>";
```

```
            // выводим список полей не прошедших валидацию
            foreach ($_ERROR_VALID as $val) {
                printf ('Ошибка заполнения поля: <b>%s.</b><br
                />', $val);
            }
        }
        // если ошибок не возникло
        else {
            echo "<h2>Данные готовы для безопасной
            обработки:</h2>";
            echo  'E-mail: ' . $email . '<br>';
            echo  'Заголовок: ' . $title . '<br>';
            echo  'Курс: ' . $course . '<br>';
        }
    }
?>
```

Данные, требующие только сантизации

example_3. index.html

```
<form action="server.php" method="post">
    <b>Заголовок</b><br>
    <input type="text" name="title" value="Что такое
    концептуальный брендинг и как дизайнеру над ним работать"><p>
    <b>Введите тизер статьи:</b><br>
    <textarea name="teazer"><b>Часто дизайнер</b> отвечает не
    только за то, как выглядит бренд, но и за эмоции, которые ...
    </textarea><p>
```

```
    <b>Введите текст статьи:</b><br>
    <textarea name="content"><h2>Какой брендинг можно считать
    концептуальным</h2>Концептуальный брендинг делает упор на ...
    </textarea><p>
    <input name="submit" type="submit" value="Отправить">
</form>
```

example_3. server.php

```
<?php
    // если выполнена отправка формы
    if(isset($_POST["submit"])){
        // очищаем данные
        $title = trim(strip_tags($_POST["title"]));
        $teazer = trim(strip_tags($_POST["teazer"], "<b>"));
        $content = trim(strip_tags($_POST["content"], "<h2>,
        <h3>, <b>,<p>"));
        echo "<h2>Данные готовы для безопасной обработки:</h2>";
        echo '<h3>' . $title . '</h3>';
        echo '<p>Тизер: ' . $teazer . '</p>';
        echo '<p>Контент: ' . $content . '</p>';
    }
?>
```

Числовые данные

С элементами формы, позволяющими вводить числовые фиксированные значения, должно быть полегче, например:

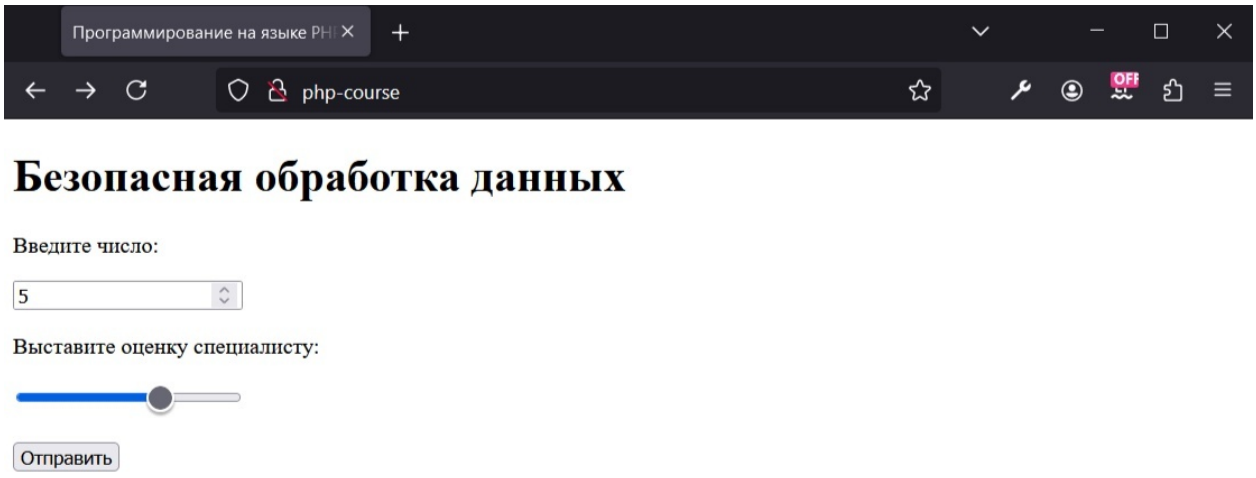
```
<p>Введите число:</p>
```

```
<input type="number" min=1 max=10 name="num" value="5">
<p>Выставите оценку специалисту:</p>
<input type="range" name="range" min="1" max="10" step="1"
value="5"><p>
<p>Укажите код CVC:</p>
<input type="text" name="cvc" pattern="^[\d]{3}$" value=""><p>
```

Такие элементы управления позволяют вводить только **установленные разработчиком** значения.

example_4. index.html

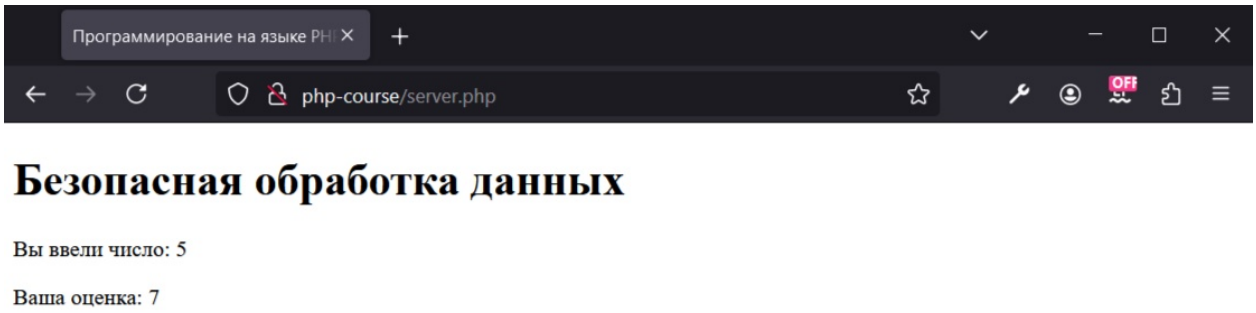
```
<form action="server.php" method="post">
    <p>Введите число:</p>
    <input type="number" min=1 max=10 name="num" value="5">
    <p>Выставите оценку специалисту:</p>
    <input type="range" name="range" min="1" max="10" step="1"
    value="5"><p>
    <input name="submit" type="submit" value="Отправить">
</form>
```



И в принимающем файле все выглядит красиво.

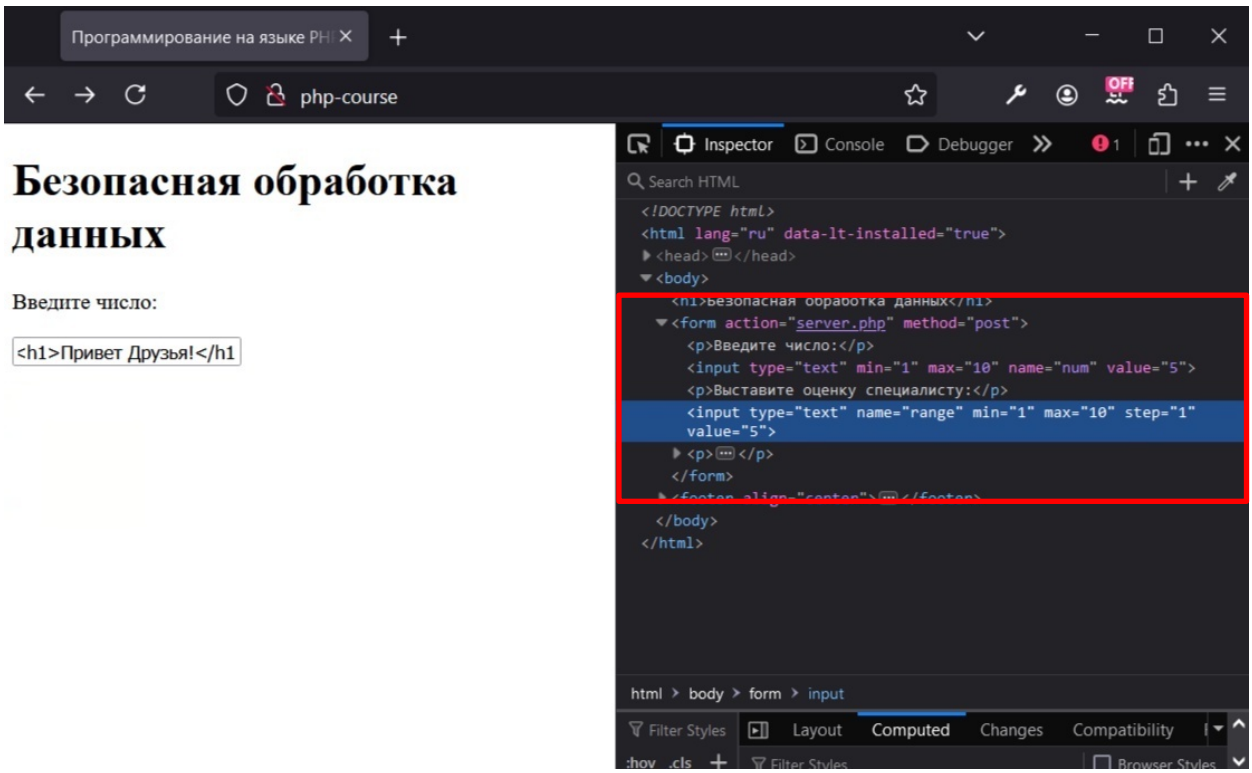
example_4. server.php

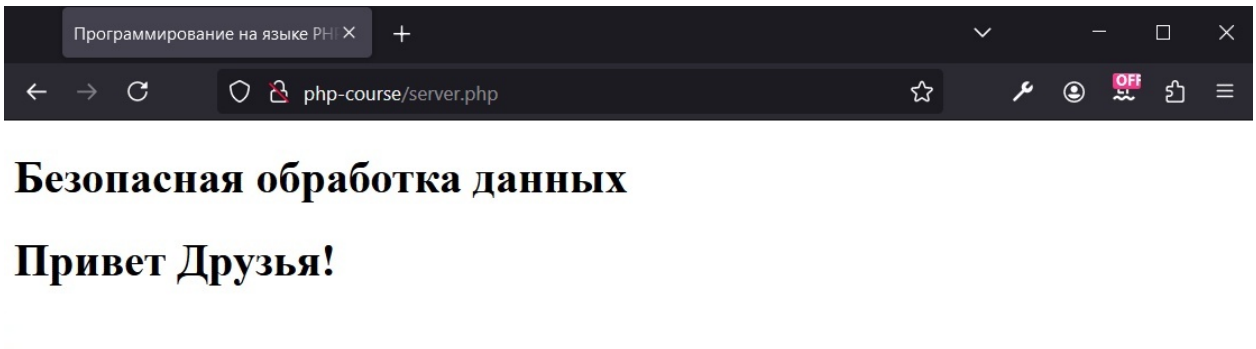
```php
<?php
    // если выполнена отправка формы
    if (isset($_POST["submit"])){
        // данные элементов input
        echo 'Вы ввели число: ' . $_POST["num"] . '<p>';
        echo 'Ваша оценка: ' . $_POST["range"];
    }
?>
```



Все хорошо, но вы помните, что пользователь — это зло?

Вспомним, почему он зло. Зайдем в режим разработчика, подправим типы текстовых полей и заменим скучные ограничения, **установленные разработчиком**.





Если это можем сделать мы, где гарантия, что это не сделает пользователь ресурса.

> ***Впрочем.*** Можно ли называть пользователем, посетителя ресурса, проделывающего этакие трюки?

Вывод простой – **контролируем любые данные,** полученные извне.

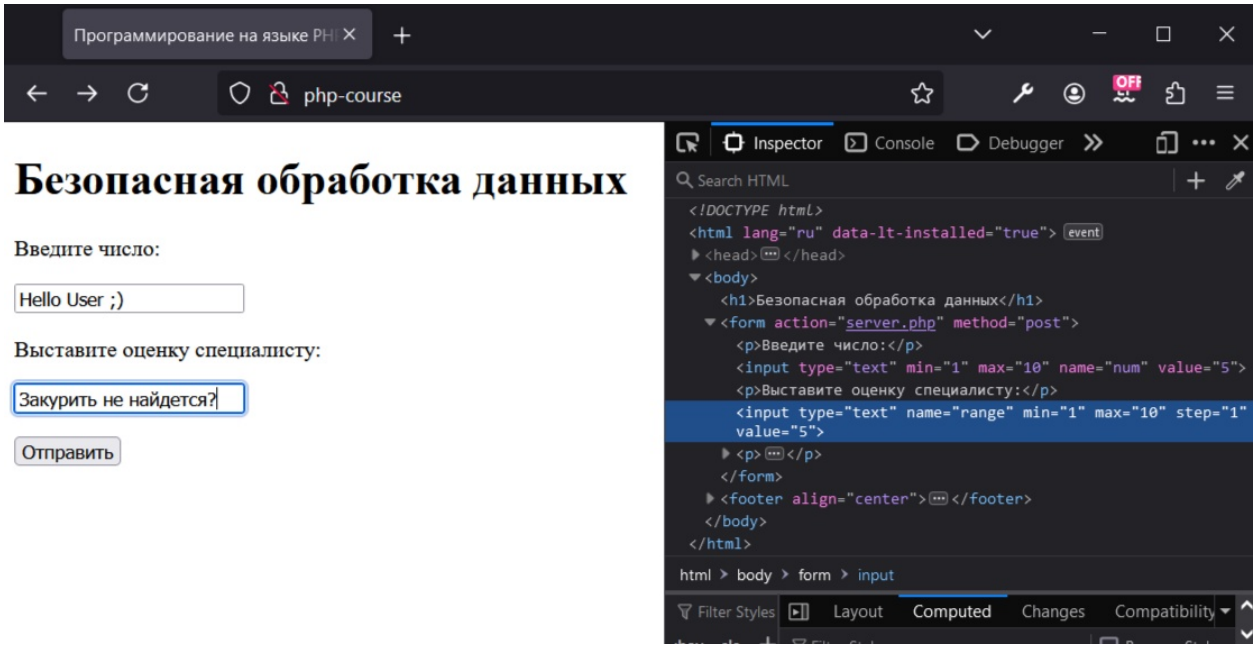
example_5. index.html

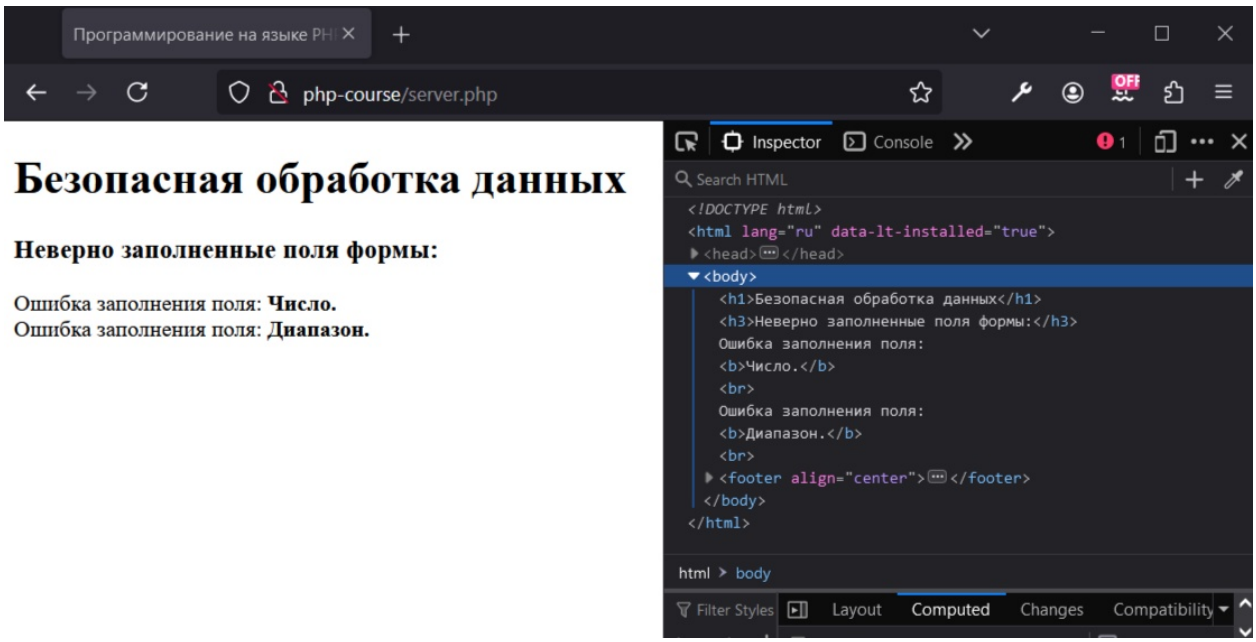
```
<form action="server.php" method="post">
    <p>Введите число:</p>
    <input type="number" min=1 max=10 name="num" value="5">
    <p>Выставите оценку специалисту:</p>
    <input type="range" name="range" min="1" max="10" step="1"
    value="5"><p>
    <input name="submit" type="submit" value="Отправить">
</form>
```

example_5. server.php

```
<?php
    // если выполнена отправка формы
    if (isset($_POST["submit"])){
        $_ERROR_VALID = array();
        // валидация поля Введите число
        if (!preg_match('/^[0-9]{1,2}$/', $_POST["num"]))
            $_ERROR_VALID[] = 'Число';
        // валидация поля Выставите оценку специалисту
        if (!preg_match('/^[0-9]{1,2}$/', $_POST["range"]))
            $_ERROR_VALID[] = 'Диапазон';
        // если массив ошибок не пуст
        if (count($_ERROR_VALID)){
            echo "<h3>Неверно заполненные поля формы:</h3>";
            // выводим список полей не прошедших валидацию
            foreach ($_ERROR_VALID as $val) {
                printf ('Ошибка заполнения поля: <b>%s.</b><br
```

```
                />', $val);

            }

        }

        // если ошибок не возникло

        else {

            echo 'Вы ввели число: ' . $_POST["num"] . '<p>';

            echo 'Ваша оценка: ' . $_POST["range"];

        }

    }

?>
```





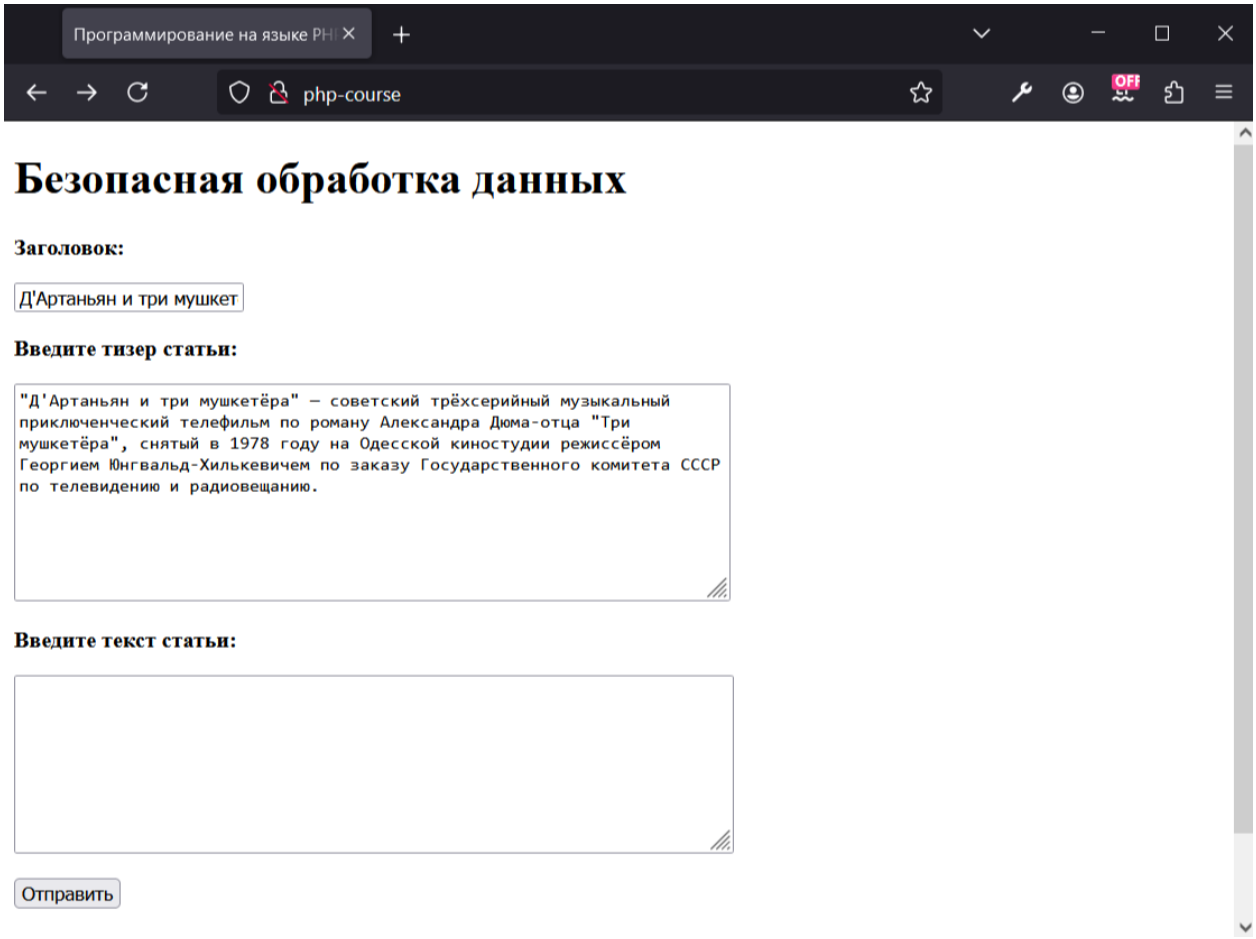
Функции addslashes(), stripslashes()

Функции **addslashes**() и **stripslashes**(). Рассмотрим эти функции применительно к практике сохранения данных в базе данных.

example_6. index.html

```
<form action="server.php" method="post">

    <p><b>Заголовок:</b></p>

    <input type="text" name="title" value="Д'Артаньян и три
    мушкетера"><p>

    <p><b>Введите тизер статьи:</b></p>

    <textarea name="teaser">"Д'Артаньян и три мушкетёра" —
    советский трёхсерийный музыкальный приключенческий телефильм
    ...

    </textarea><p>

    <p><b>Введите текст статьи:</b></p>

    <textarea name="content"></textarea><p>
```

```
    <input name="submit" type="submit" value="Отправить">
</form>
```



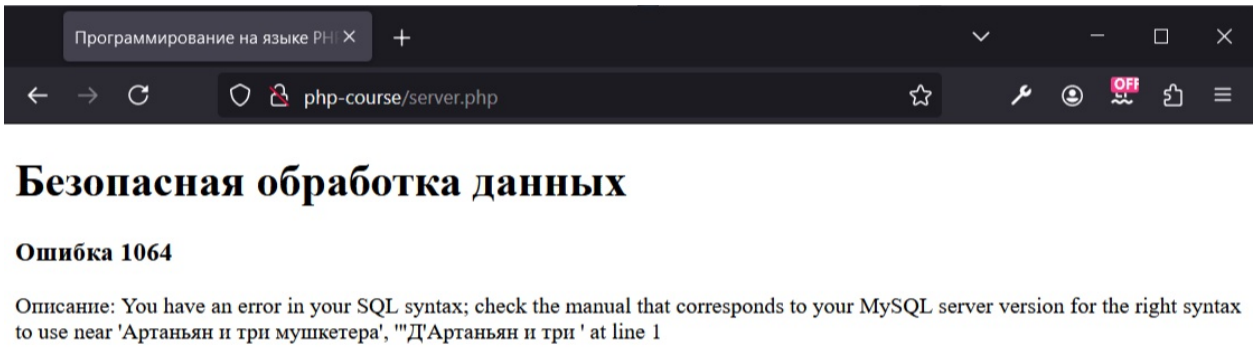
example_6. server.php

```
<?php
    // если выполнена отправка формы
    if (isset($_POST["submit"])){
        // очищаем данные
        $title = trim(strip_tags($_POST["title"]));
        $teaser = trim(strip_tags($_POST["teaser"], "<b>"));
        // подключение к серверу СУБД
        $mysqli = new mysqli('localhost', 'root', '',
```

```
        'db_publication');

        // формирование строки запроса
        $insert = sprintf("
            INSERT INTO `posts`(`id`,`title`,`teaser`)
            VALUES(NULL, '%s', '%s')",
            $title,
            $teaser
        );
        // управление режимом протоколирования ошибок
        mysqli_report(MYSQLI_REPORT_OFF);
        // выполнение запроса на вставку
        $result = $mysqli->query($insert);
        // проверяем результат выполнения запроса
        if(!$result){
            // если возникла ошибка выполнения
            // тестовый вывод состояния
            // echo "<pre>";
            // print_r($mysqli);
            // echo "</pre>";
            echo '<h3>Ошибка ' . $mysqli->errno . '</h3>';
            echo 'Описание: ' . $mysqli->error;
        } else {
            // если запрос выполнен успешно
            // тестовый вывод состояния
            echo "<pre>";
            print_r($mysqli);
            echo "</pre>";
```

```
        }
    }
?>
```

Получили ошибку. Происхождение ошибки – **кавычки**, используемые **внутри строк,** накладываются на кавычки, используемые внутри **SQL-выражений**.



Используем рассматриваемые функции. Файл формы **example_7/index.html** без изменений.

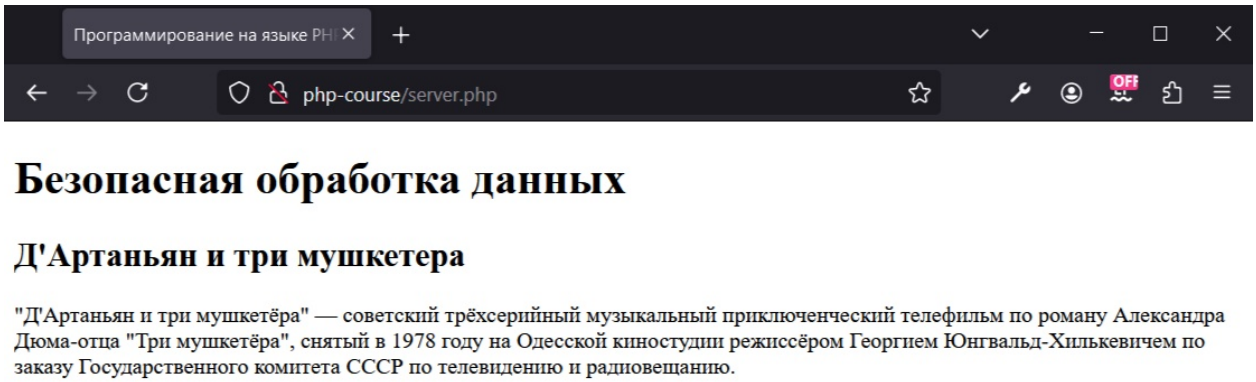
example_7. server.php

```
<?php
    // если выполнена отправка формы
    if (isset($_POST["submit"])){
        // очищаем данные
        $title = trim(strip_tags($_POST["title"]));
        $teaser = trim(strip_tags($_POST["teaser"], "<b>"));
        // подключение к серверу СУБД
        $mysqli = new mysqli('localhost', 'root', '',
        'db_publication');
        // формирование строки запроса
```

```php
$insert = sprintf("
    INSERT INTO `posts`(`id`,`title`,`teaser`)
    VALUES(NULL, '%s', '%s')",
    addslashes($title),
    addslashes($teaser)
);
// управление режимом протоколирования ошибок
mysqli_report(MYSQLI_REPORT_OFF);
// выполнение запроса на вставку
$result = $mysqli->query($insert);
// проверяем результат выполнения запроса
if (!$result){
    // если возникла ошибка выполнения
    // тестовый вывод состояния
    // echo "<pre>";
    // print_r($mysqli);
    // echo "</pre>";
    echo '<h3>Ошибка ' . $mysqli->errno . '</h3>';
    echo 'Описание: ' . $mysqli->error;
} else {
    // если запрос выполнен успешно
    // извлекаем вствленные данные
    $select = 'SELECT * FROM `posts` WHERE `id`=' .
    $mysqli->insert_id;
    $row = $mysqli->query($select)->fetch_assoc();
    echo '<h2>' . stripslashes($row['title']) . '</h2>';
    echo '<div>' . stripslashes($row['teaser']) .
    '</div>';
```

```
        }

    }

?>
```

Теперь все хорошо.



Строго говоря, официальные источники рекомендуют использовать не функции (в данном случае), а синтаксис **подготовленных запросов**.

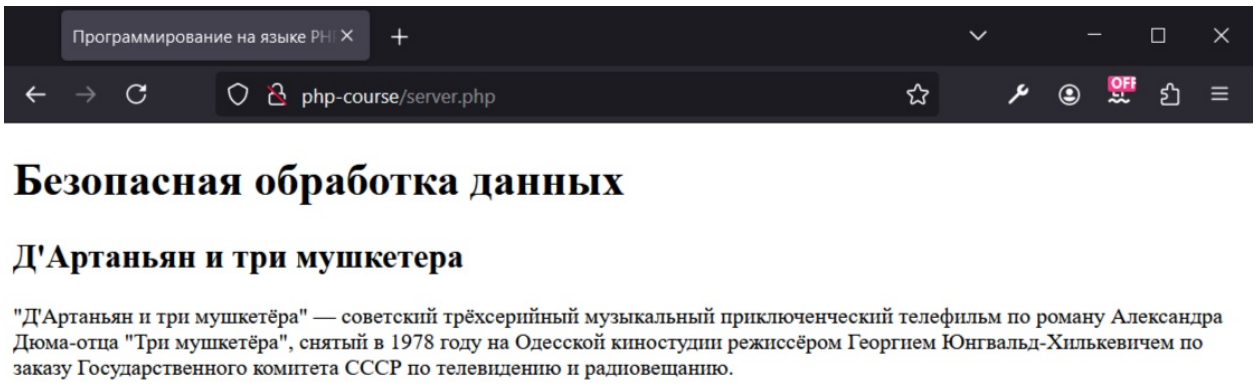
Файл формы **example_8/index.html** без изменений.

example_8. server.php

```
<?php

    // если выполнена отправка формы

    if (isset($_POST["submit"])){

        // очищаем данные

        $title = trim(strip_tags($_POST["title"]));

        $teaser = trim(strip_tags($_POST["teaser"], "<b>"));

        // подключение к серверу СУБД

        $mysqli = new mysqli('localhost', 'root', '',
        'db_publication');

        // формирование строки запроса
```

```
        $stmt = $mysqli->prepare("INSERT INTO
        `posts`(`id`,`title`,`teaser`) VALUES(NULL, ?, ?)");
        $stmt->bind_param('ss', $title, $teaser);
        $stmt->execute();
        // тестовый вывод состояния
        // echo "<pre>";
        // print_r ($stmt);
        // echo "</pre>";
        // проверяем результат выполнения запроса
        if ($stmt->errno) {
            // если возникла ошибка выполнения
            echo '<h3>Ошибка ' . $mysqli->errno . '</h3>';
            echo 'Описание: ' . $mysqli->error;
        } else {
            // если запрос выполнен успешно
            $select = 'SELECT * FROM `posts` WHERE `id`=' .
            $mysqli->insert_id;
            $row = $mysqli->query($select)->fetch_assoc();
            echo '<h2>' . $row['title'] . '</h2>';
            echo '<div>'  . $row['teaser'] . '</div>';
        }
    }
?>
```

Синтаксис **подготовленных запросов** дает результат, равный использованию функций и ряд других преимуществ, обеспечивающих выполнение безопасных запросов к базе данных.



Безопасная обработка бинарных данных

Обрабатывать необходимо не только текстовые или числовые данные, полученные от пользователя и содержащиеся в массиве $_**REQUEST** (т.е. $_POST или $_GET), но и прочие файлы, загружаемые пользователем и попадающие в массив $_**FILES**.

Проверка бинарных файлов состоит из несколько шагов, обязательных к реализации в любом приложении.

Для примера реализуем загрузку и сохранение в базе данных файла изображения.

example_9. index.html

```
<form action="server.php" enctype="multipart/form-data"
method="post">
    <!-- поле MAX_FILE_SIZE определяет максимальный размер
    загружаемого файла -->
    <input type="hidden" name="MAX_FILE_SIZE" value="300000" />
    Загрузите ваш аватар: <input type="file" name="avatar"><p>
    <input type="submit" name="load" value="Загрузить файл">
</form>
```

Файл формы выбора файла для всех примеров раздела будет неизменным.

Прежде чем получить значение переменной необходимо выполнить несколько стандартных проверок значений массива $_**FILES**.

Были ли ошибки при отправке файла

Проверим массив $_**FILES** на наличие ошибок. Если поле ошибки $_FILES["avatar"]["error"] имеет значение 0, значит все прошло успешно и нужно продолжить выполнение сценария.

Если значение отлично от нуля, что-то дало сбой, в таком случае сохраним информацию об ошибке. Другая логика приложения может переадресовывать пользователя на страницу вывода ошибок для указания возникшей проблемы.

example_9. server.php

```
<?php
    // были ли ошибки при отправке файла
    // массив для сбора ошибок
    $_ERROR_LOADING = [];
    // проверим загрузку на наличие ошибок
    if ($_FILES['avatar']["error"] !== UPLOAD_ERR_OK) {
        // если при загрузке произошла ошибка, запомним
        информацию о ней
        switch ($_FILES['avatar']['error']) {
            case UPLOAD_ERR_INI_SIZE:
                $_ERROR_LOADING[] = "Размер принятого файла
                превысил максимально допустимый размер, который
                задан директивой upload_max_filesize
                конфигурационного файла php.ini (код ошибки: 1)";
```

```
                break;
            case UPLOAD_ERR_FORM_SIZE:
                $_ERROR_LOADING[] = "Размер загружаемого файла
                превысил значение MAX_FILE_SIZE, указанное в
                HTML-форме (код ошибки: 2)";
                break;
            case UPLOAD_ERR_PARTIAL:
                $_ERROR_LOADING[] = "Загружаемый файл был получен
                только частично (код ошибки: 3)";
            break;
            case UPLOAD_ERR_NO_FILE:
                $_ERROR_LOADING[] = "Файл не был загружен (код
                ошибки: 4)";
        }
    } else {
        echo "<p>Файл загружен во временную директоию
        сервера.</p>";
    }
    // выведем список ошибок если таковые имеются
    if (count($_ERROR_LOADING)){
        echo "<h3>Список возникших ошибок:</h3>";
        echo "<pre>";
        print_r($_ERROR_LOADING);
        echo "</pre>";
    }
?>
```

Отправляется ли файл по протоколу HTTP

PHP функция **is_uploaded_file**() гарантирует, что предоставленное имя файла имеет отношение к файлу, который отправлен на сервер по протоколу HTTP (используемому веб-браузерами и HTML-формами). Таким образом, если предоставленное имя будет указывать на файл из файловой системы сервера, функция вернет **false**, это признак того, что попытку загрузки необходимо прекратить, с сохранением информации об ошибке.

Для получения имени файла используем свойство $_FILES["avatar"]["tmp_name"]. Благодаря этому свойству мы получаем ссылку на временное имя файла в директории сервера.

example_10. server.php

```php
<?php
    // отправляется ли файл по протоколу HTTP
    // массив для сбора ошибок
    $_ERROR_LOADING = [];
    // проверим загрузку на наличие ошибок
    if ($_FILES['avatar']["error"] !== UPLOAD_ERR_OK) {
        // ...
    } else {
        // отправляется ли файл по протоколу HTTP
        if (!is_uploaded_file($_FILES["avatar"]["tmp_name"])) {
            $_ERROR_LOADING[] = "Запрос на отправку локального
            файла: " . $_FILES["avatar"]["tmp_name"];
        }
    }
    // выведем список ошибок если таковые имеются
    if (count($_ERROR_LOADING)){
        echo "<h3>Список возникших ошибок:</h3>";
```

```
        echo "<pre>";
        print_r($_ERROR_LOADING);
        echo "</pre>";
    } else
        echo "<p>Ошибок загрузки не обнаружено</p>";
?>
```

Является ли отправленный файл изображением

Итак, мы имеем ссылку на загружаемый файл и знаем, что он не из разряда подделок с именем, указывающим на какой-нибудь защищенный файл в файловой системе сервера.

Следующий шаг – убедиться в том, что это файл изображения (или любого другого требуемого формата). Ничто не мешает пользователю случайно (или специально) отправить документ Word, а также ничто не препятствует отправке какого-нибудь файла с кодом JavaScript или исполняемого файла.

PHP позволяет довольно просто проверять принадлежность файлов к изображениям. Для этого нужно воспользоваться функцией **exif_imagetype**().

example_11. server.php

```
<?php
    // является ли отправленный файл изображением
    // массив для сбора ошибок
    $_ERROR_LOADING = [];
    // проверим загрузку на наличие ошибок
    if ($_FILES['avatar']["error"] !== UPLOAD_ERR_OK) {
        // ...
```

```
    } else {
        // является ли загружаемый файл изображением
        if (!exif_imagetype($_FILES["avatar"]["tmp_name"])) {
            $_ERROR_LOADING[] = "Загружаемый файл не является
            файлом изображения";
        }
    }
    // выведем список ошибок если таковые имеются
    if (count($_ERROR_LOADING)){
        echo "<h3>Список возникших ошибок:</h3>";
        echo "<pre>";
        print_r($_ERROR_LOADING);
        echo "</pre>";
    } else
        echo "<p>Ошибок загрузки не обнаружено</p>";
?>
```

Произошла правильная отправка файла по протоколу HTTP и этот файл является изображением. Осталось только переместить это изображение из временного места хранения, которое используется веб-сервером для загружаемых файлов, в директорию постоянного хранения.

Перемещен ли файл в директорию постоянного хранения

Когда сервер получает файл, он отправляет его в директорию, заранее определенную **конфигурацией**. Он также, скорее всего, воспользуется именем, **не совпадающим с исходным именем** пользовательского файла.

Поэтому следующие шаги:

- присвоить файлу имя,
- переместить его в директорию постоянного хранения.

Существует множество различных подходов к присваиванию имени. Вы можете придумать что-нибудь связанное с пользователем, отправившим файл на сервер, например использовать в имени файла **логин пользователя**.

Но зачастую проще всего дать файлу уникальное **числовое имя**. Самым простым способом для этого служит получение **текущего времени** и создание имени файла на его основе — это практически надежный путь получения уникального имени файла.

В демонстрационном примере **example_12** переместим файл из его временного местоположения в постоянное место хранения. Предварительно в директории сервера должна быть создана директория постоянного хранения файлов. Пусть это будет директория **upload**.

example_12. server.php

```php
<?php
    // перемещен ли файл в директорию постоянного хранения
    // массив для сбора ошибок
    $_ERROR_LOADING = [];
    // проверим загрузку на наличие ошибок
    if ($_FILES['avatar']["error"] !== UPLOAD_ERR_OK) {
        // ...
    } else {
        // место постоянного хранения файла
        $dir = __DIR__ . '/upload/';
        $filename = time() . "_" . $_FILES["avatar"]["name"];
        $path = $dir . $filename;
```

```
        // перемещение загруженного файла
        if (!move_uploaded_file($_FILES["avatar"]["tmp_name"],
        $path)) {
            $_ERROR_LOADING[] = "Не удалось переместить файл в
            директорию хранения";
        }
    }
    // выведем список ошибок если таковые имеются
    if (count($_ERROR_LOADING)){
        echo "<h3>Список возникших ошибок:</h3>";
        echo "<pre>";
        print_r($_ERROR_LOADING);
        echo "</pre>";
    } else {
        echo '<h3>Файл ' . $_FILES['avatar']['name'] . ' успешно
        загружен!</h3>';
        echo 'Проверьте директорию <b>' . $dir . '</b>';
    }
?>
```

Запись пути к файлу в базу данных

Соберём все вместе. Добавим код сохранения изображения в базу (а точнее путь к изображению), выводим результат.

example_13. server.php

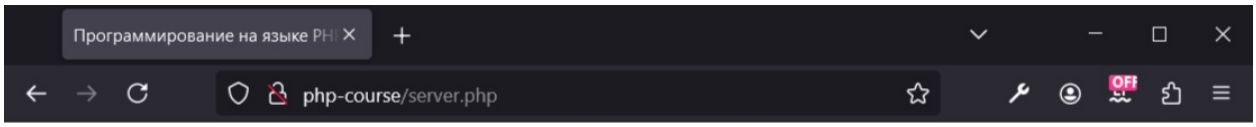
```
<?php
    // массив для сбора ошибок
    $_ERROR_LOADING = [];
```

```
// проверим загрузку на наличие ошибок
if ($_FILES['avatar']["error"] !== UPLOAD_ERR_OK) {
    // если при загрузке произошла ошибка, запомним информацию о ней
    switch ($_FILES['avatar']['error']) {
        case UPLOAD_ERR_INI_SIZE:
            $_ERROR_LOADING[] = "Размер принятого файла превысил максимально допустимый размер, который задан директивой upload_max_filesize конфигурационного файла php.ini (код ошибки: 1)";
            break;
        case UPLOAD_ERR_FORM_SIZE:
            $_ERROR_LOADING[] = "Размер загружаемого файла превысил значение MAX_FILE_SIZE, указанное в HTML-форме (код ошибки: 2)";
            break;
        case UPLOAD_ERR_PARTIAL:
            $_ERROR_LOADING[] = "Загружаемый файл был получен только частично (код ошибки: 3)";
            break;
        case UPLOAD_ERR_NO_FILE:
            $_ERROR_LOADING[] = "Файл не был загружен (код ошибки: 4)";
        }
} else {
    // отправляется ли файл по протоколу HTTP
    if (!is_uploaded_file($_FILES["avatar"]["tmp_name"])) {
        $_ERROR_LOADING[] = "Попытка отправки локального файла: " .
        $_FILES["avatar"]["tmp_name"];
    }
    // является ли загружаемый файл изображением
    if (!exif_imagetype($_FILES["avatar"]["tmp_name"])) {
        $_ERROR_LOADING[] = "Загружаемый файл не является файлом изображения";
    }
```

```
        // место постоянного хранения файла
        $dir = __DIR__ . '/upload/';
        $filename = time() . "_" . $_FILES["avatar"]["name"];
        $path = $dir . $filename;
        // проверим существовании директории хранения файлов
        if (file_exists($dir)) {
            // перемещение загруженного файла
            if (move_uploaded_file($_FILES["avatar"]["tmp_name"], $path)) {
                // сохраним файл в базе данных
                // подключение к базе данных
                $mysqli = new mysqli ('localhost', 'root', '',
                'db_secure_data');
                // формирование строки запроса на вставку в базу данных
                $sql = sprintf("INSERT INTO `posts` (`id`, `file`) VALUES
                (NULL, '%s')", $filename);
                // выполнение запроса на сохранение в базу
                if ($mysqli->query($sql)) {
                    $id = $mysqli->insert_id;
                } else {
                    $_ERROR_LOADING[] = "Не удалось сохранить файл в базу
                    данных";
                }
            } else {
                $_ERROR_LOADING[] = "Неизвестная ошибка перемещения файла";
            };
        } else {
            $_ERROR_LOADING[] = "Не удалось переместить файл в директорию
            хранения";
        }
    }
    // выведем список ошибок если таковые имеются
    if (count($_ERROR_LOADING)){
```

```
            echo "<h3>Список возникших ошибок:</h3>";
            echo "<pre>";
            print_r($_ERROR_LOADING);
            echo "</pre>";
        } else {
            // если сохранение файла выполнено успешно
            $row = $mysqli->query('SELECT `file` FROM `posts` WHERE `id`=' .
            $id)->fetch_assoc();
            echo '<h3>Файл ' . $_FILES['avatar']['name'] . ' успешно
            загружен!</h3>';
            echo 'Проверьте директорию <b>' . $dir . '</b><br>';
            echo '<img src="upload/' . $row['file'] . '" width="200px">';
        }
?>
```

Результат загрузки и последующего вывода в браузер представлен на рисунке.



Безопасная обработка данных

Файл no-photo.jpg успешно загружен!

Проверьте директорию **C:\OpenServer\domains\php-course/upload/**

Код получился не маленьким, но, если вы хотите исключить всякого рода неожиданности и быть уверенными, что работаете с **безопасными данными** – альтернативы нет.