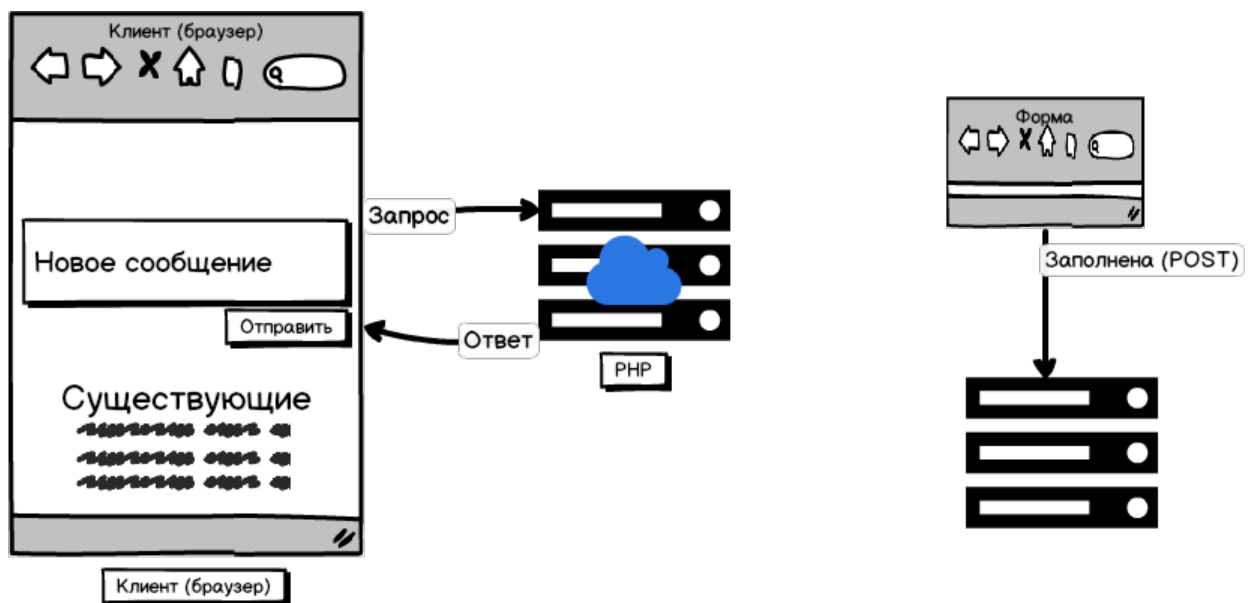


## Лабораторная работа: добавление и чтение записей файла в PHP

Создадим приложение, которое демонстрирует простые возможности PHP по работе с файлами. Оно представляет собой простейший логгер или журнал сообщений (об ошибках или о чём угодно). Логирование - это оставление в файле типа log.txt типовых сообщений длиной в одну строку, т.е. сообщения отделены друг от друга символом \n (символом с десятичным кодом 10).

### Составные части нашего приложения:

1. Клиентская часть - форма с полями, куда посетитель сайта вводит три части своего отзыва, и список существующих отзывов.
2. Серверная часть - сценарий на PHP, который принимает сообщение, записывая его в файл и выдаёт из файла существующие сообщения



## Оптимальный цикл работы над проектом

На сайте github.com создаём репозиторий php_logger  Далее будем отправлять код в репозиторий с локального компьютера а на удалённом сервере его получать и запускать.	
На компьютере разработчика	На сервере
<pre>mkdir logger &amp;&amp; cd \$_ git init git remote add origin https://github.com/myRepo/php_logger.git  git add -A git status git commit -m 'first' git push</pre>	<pre>apt-get install git mkdir logger &amp;&amp; cd \$_  git clone https://github.com/myRepo/php_logger.git .  далее git pull</pre>

---

У нас на одной странице будет располагаться форма (состоящая из одного поля и одной кнопки) и список уже существующих сообщений, извлечённых из файла log.txt

С точки зрения организации программы мы будем иметь два PHP-файла: основной (index.php) и вспомогательный list.php, который формирует список сообщений, прочитанных из файла.

Список мы будем формировать как готовую разметку, которая будет отправляться в браузер как есть.

*Замечание о безопасности и пустых данных.*

При отправке формы типичная проблема - то, что пользователь забыл или намеренно не заполнил поле, а также - что заполнил чем-то ошибочным или даже вредоносным, например, кодом на языке JavaScript или SQL.

Защиту от пустых значений можно реализовать на двух уровнях:

- самой формы (pattern, requires);
- PHP-сценария (оператор coalesce, функция empty или isset);

Что касается защиты от вредоносных значений, то здесь на уровне PHP-сценария нужна дополнительная проверка (валидация, санация). Предположим, посетитель вводит в форму нечто подобное этому:

**Имя**

  
**Email**

ilia.gossoudarev@gmail.com

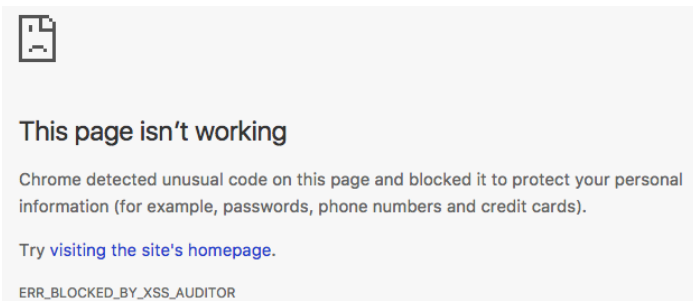
**Отзыв**

<script>alert(1)</script>

☒ Согласен с условиями

Отправить отзыв

т.е. пытается осуществить XSS-атаку, внедряя в то, что по идее должно быть просто текстом, то, что является сценарием на JavaScript. Если не предпринимать специальных мер, это будет прямо так записано в базу данных, а при выводе... мы увидели бы в данном случае всплывающее окно с единицей... или такое предупреждение:



Суперглобальные переменные часто используются злоумышленниками, пытающимися отыскать средства для атаки и вмешательства в работу сайта. Они загружают в \$\_POST, \$\_GET или в другие суперглобальные переменные вредоносный код, в том числе команды UNIX или MySQL, которые, если по незнанию к ним обратиться, могут разрушить или отобразить незащищенные данные.

Именно поэтому перед применением суперглобальных переменных их всегда следует подвергать предварительной обработке. Для этого можно воспользоваться PHP-функцией htmlentities. Она занимается преобразованием всех символов в элементы HTML.

Например, символы «меньше» и «больше» (< и >) превращаются в строки &lt; и &gt;;, то же самое делается для перевода в безопасное состояние всех кавычек, обратных слешей и т. д. В результате вышеприведённая «атака» запишется в файле так:

&lt;script&gt;alert(1)&lt;/script&gt;

```
$r = htmlentities($_POST['message']) ?? 'Пустое сообщение';
```

Т.е. мы берём значение, отправленное методом POST и прогоняем его через функцию htmlentities. А если окажется NULL, то подставим текст по умолчанию.

На момент выполнения данной лабораторной работы пример работающего приложения *может быть* доступен по такому адресу как:

<http://92.53.66.41/logger/>

Открытие файла для чтения или записи осуществляется с помощью функции fopen, которая принимает аргумент, содержащий имя файла, и аргумент, обозначающий режим (чтение, запись и т.д.)

чтение (list.php)	запись (index.php)
<code>\$f = fopen('log.txt', 'r');</code>	<code>\$f = fopen('log.txt', 'a');</code>

При первом обращении (это чтение) файл может быть автоматически создан (как это скорее всего произойдёт в c9.io) или может возникнуть проблема с правами доступа (на vscale.io мы увидим в логе ошибок /var/log/apache2/error.log сообщение **fopen(log.txt): failed to open stream: No such file or directory**).

<p><b>*Дополнительный материал</b></p> <p>Если возникают проблемы с правами, то вариантом решения является следующий.</p> <p>Нам нужно создать файл log.txt вручную (touch log.txt и посмотреть права на него с помощью ls -la log.txt).</p> <pre>-rw-r--r-- 1 root root ...</pre> <p>это означает, что все права на файл принадлежат пользователю root.</p> <p>Выясним, от чьего имени запускается php.</p> <pre>ps auxwww   grep apache</pre> <p>это www-data</p> <p>значит нужно дать пользователю www-data права на запись</p> <p><a href="https://askubuntu.com/questions/244406/how-do-i-give-www-data-user-to-a-folder-in-my-home-folder">https://askubuntu.com/questions/244406/how-do-i-give-www-data-user-to-a-folder-in-my-home-folder</a></p> <pre>chgrp www-data log.txt</pre> <p>и снова посмотреть права:</p> <pre>-rw-r--r-- 1 root www-data ...</pre> <p>Теперь добавляем для www-data право на запись:</p> <pre>chmod g+w log.txt</pre> <p>и снова смотрим права:</p> <pre>-rw-rw-r-- 1 root www-data ..</pre> <p>После этого сценарий, скорее всего, сможет работать с файлом.</p>
---

Так выглядит результат отправки сообщения в лог:

← → ↻ ⓘ 92.53.66.41/logger/index.php

## Работа с логом...

Данные успешно добавлены, спасибо!

Добавить ещё один отзыв...

Сообщение

### Список отзывов...

- Привет!

Теперь посмотрим, как должны выглядеть наши два файла:

Первый файл выводит на страницу форму и подключает второй файл для вывода списка уже существующих сообщений. Если запрос приходит методом POST, то сценарий ещё и добавляет пришедшее сообщение в файл log.txt

## index.php

```
index.php x
1 <meta charset="utf-8">
2 <h1>Работа с логом... </h1>
3 <?php
4     $f = '<h2>Форма для заполнения... </h2>';
5     if ($_SERVER['REQUEST_METHOD']=='POST') {
6         $r = htmlentities($_POST['message']) ?? 'Пустое сообщение';
7
8         $f = fopen('log.txt', 'a');
9         fwrite($f, $r);
10        fwrite($f, "\n");
11        fclose($f);
12
13        $f = '<style>.right {width: 60%; margin-left: 35%; zoom: 80%;}</style>';
14        $f .= '<div class="right"><h2>Добавить ещё один отзыв...</h2></div>';
15        echo '<h3 style="color:green">Данные успешно добавлены, спасибо!</h3>';
16    }
17    echo $f;
18    echo "<div class=\"right\">";
19    echo "<form method=\"post\" action=\"\"/{$_SERVER['SERVER_NAME']}{$_SERVER['SCRIPT_NAME']}\>";
20    echo <<<_END
21        <div>
22            <label>Сообщение</label>
23            <input id="message" name="message" required="required" type="text">
24            <button type="submit">Отправить сообщение</button>
25        </div>
26    _END;
27    echo "</form></div>";
28    echo "<h2>Список отзывов... </h2>\n";
29    echo require_once ('list.php');
```

Второй файл формирует список сообщений и возвращает его - он не может работать самостоятельно в том смысле что не содержит команд вывода и играет роль функции, которую вызывает наш главный файл index.php

### list.php

```
list.php x
1  <?php
2      $f = fopen('log.txt', 'r');
3      $list = '';
4      $list .= '<ul>';
5      function wrap($x){
6          return '<li>'.$x.'</li>';
7      }
8      while (($line = fgets($f)) !== false) {
9          $list .= wrap($line);
10     }
11     fclose($f);
12     $list .= '</ul>';
13     return $list;
```

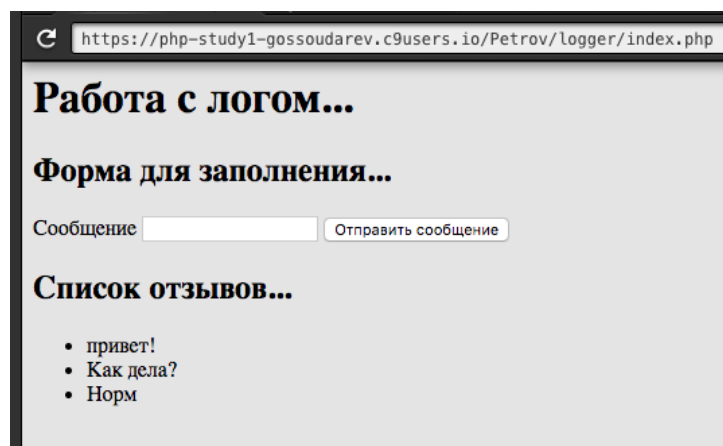
Итак, вам нужно как минимум создать у себя в c9.io подкаталог logger с двумя файлами, приведёнными выше.

Далее перейдя по адресу, такому как

<https://php-study1-gossoudarev.c9users.io/Petrov/logger/index.php>

( можно не добавлять в конце /index.php )

вы увидите форму и, если файл log.txt не пустой, список строк из него:



The screenshot shows a web browser window with the address bar displaying <https://php-study1-gossoudarev.c9users.io/Petrov/logger/index.php>. The page content includes:

- A heading "Работа с логом..." (Working with log...).
- A sub-heading "Форма для заполнения..." (Form for filling...).
- A form with a text input field labeled "Сообщение" (Message) and a button labeled "Отправить сообщение" (Send message).
- A sub-heading "Список отзывов..." (List of reviews...).
- A bulleted list of messages: "привет!" (hello!), "Как дела?" (How are you?), and "Норм" (Normal).





## Дополнительные сведения о функциях вывода в PHP

В PHP можно воспользоваться многострочной последовательностью, используя оператор `<<<`, который обычно называют *here-document* («здесь документ») или *heredoc*. Он представляет собой способ указания строкового литерала, сохраняющего в тексте обрывы строк и другие пустые пространства (включая отступы).

```
<pre><?php

    $author = 'Brian W. Kernighan';
    echo <<<_END
    Debugging is twice as hard as writing the code in the first place.
    Therefore, if you write the code as cleverly as possible, you are,
    by definition, not smart enough to debug it.
    - $author.
_END;
?></pre>
```

Этот код предписывает PHP вывести все, что находится между двумя тегами `_END`, как будто все это является строкой, заключенной в двойные кавычки (за исключением того, что изменять предназначение кавычек в heredoc не нужно). Это означает, что разработчику можно, например, написать целый раздел HTML-кода прямо в коде PHP, а затем заменить конкретные динамические части переменными PHP. Тег `_END` — лишь пример, так как его использование где-нибудь еще в коде PHP маловероятно, он годится на роль ограничителя блока текста. Вы можете использовать по собственному усмотрению любой тег, например `_SECTION1` или `_OUTPUT` и т. д. И еще, чтобы отличать подобные теги от переменных или функций, обычно в начале их имени ставят знак подчеркивания.

Важно запомнить, что закрывающий тег `_END`; *должен* появляться строго в начале новой строки и быть *единственным* содержимым этой строки — к ней не разрешается добавлять даже комментарии (нельзя ставить даже одиночный пробел). Как только многострочный блок закрыт, можно снова воспользоваться тем же самым именем тега.

Используя heredoc-конструкцию `<<<_END..._END;`, вы не должны добавлять символы `\n`, чтобы отправить команду на перевод строки, достаточно просто нажать клавишу Enter и приступить к набору новой строки. В отличие от других строк, заключенных в одинарные или двойные кавычки, внутри конструкции heredoc можно по своему усмотрению совершенно свободно пользоваться всеми одинарными или двойными кавычками, не изменяя их первоначального предназначения с помощью обратного следа (`\`).

Команде echo есть альтернатива, которой также можно воспользоваться: команда print. Эти две команды очень похожи друг на друга, но print — конструкция, похожая на функцию, воспринимающую единственный параметр и имеющую возвращаемое значение (которое всегда равно 1), а echo не может быть использована как часть выражения.

```
$b = 1;
```

```
$b ? print 'TRUE ' : print 'FALSE ';
```

```
TRUE
```

```
php > $b ? echo 'TRUE ' : echo 'FALSE';
```

```
PHP Parse error: syntax error, unexpected 'echo' (T_ECHO)
```

```
php > echo $b ? 'TRUE ' : 'FALSE';
```

```
TRUE
```

Если что-то вычисляется как ложное значение (типа  $2 > 2 + 2$ ), то оно никак не выводится командами echo или print, т.к. константа FALSE определена как NULL (ничто).

```
php > echo 2 > 2 + 2;
```

```
php > echo 2 > 2 - 2;
```

```
1
```

Т.е. не выводится НИЧЕГО, это не пробел и не пустая строка.