Валидация данных

Введение

Валидация формы — это проверка данных, которые ввёл пользователь. Если на вашем сайте есть форма без валидации, пользователи будут заполнять её как захотят. Кто-то пропустит важное поле, кто-то неправильно введёт телефон или номер банковской карты. В результате обрабатывать такие данные станет сложнее, да и небезопасно.

Валидация на стороне клиента

Валидация на стороне клиента — это проверка данных до отправки формы. Она помогает пользователю **быстро узнать** об ошибках при заполнении полей и исправить их. Вы можете проверить поля для ввода логина, пароля, почты, номера телефона и других типов данных, которые должны соответствовать определённым критериям.

Валидация на стороне клиента условно подразумевает следующее:

- валидацию с помощью **атрибутов**;
- валидацию на **JavaScript**.

Валидация с помощью атрибутов. Самый простой способ проверки — добавить HTML-атрибуты полям ввода. С помощью атрибутов вы можете указать тип данных, который ожидаете в каждом поле, отметить обязательные поля, задать ограничение по длине строки или установить максимальное и минимальное значение для чисел.

Если пользователь вводит некорректные данные или не заполняет обязательное поле, ему показывается сообщение об ошибке

Не пренебрегайте атрибутами, даже если используете для валидации JavaScript. Порой скрипты не загружаются на страницу, например, при слабом интернет-соединении. В таком случае именно атрибуты не позволяют пользователю ввести данные в некорректном формате.

Валидация на JavaScript. Дополнительно стоит настроить валидацию с помощью JavaScript. На JS пишутся функции, которые проверяют данные и выводят сообщение, если пользователь неправильно заполнил поле. Такие подсказки можно показывать как в момент набора текста, так и при попытке отправить форму.

Какой-то единой функции для проверки валидации не существует — вам нужно написать свои под вашу конкретную задачу. Например, простая валидация формы входа в личный кабинет может выглядеть так:

В реальных проектах валидация формы бывает сложнее, поэтому для неё могут использоваться библиотеки,

Валидация на стороне сервера

Проверку на стороне клиента можно обойти, поэтому важно валидировать данные и на стороне сервера — то есть после отправки формы. Такая проверка надёжнее, так как не зависит от клиента и помогает точнее обнаружить ошибки в данных.

Обычно валидация выполняется на PHP, Python, Java и других языках программирования. Этой задачей занимаются бэкендеры. При проверке они используют регулярные выражения, условные операторы и другие средства языка программирования.

В каждом языке свои особенности валидации. Если обобщить, то проверка может выглядеть примерно так:

- Получаем данные от клиента через HTTP-запрос и сохраняем в переменные на сервере.
- Определяем правила валидации. Проверяем, что данные соответствуют ожидаемому формату, например, что они не пустые.
- Пишем код для валидации.
- Обрабатываем результаты валидации. Если данные не проходят проверку, возвращаем ошибку клиенту и даём инструкции по исправлению. Если данные корректны, продолжаем выполнение запроса, обрабатываем введённые данные.
- Если валидация прошла успешно, можно сохранить их в базе данных или использовать для выполнения других действий.

При проведении валидации на стороне сервера также важно помнить о безопасности. Например, можно экранировать все входные данные, чтобы предотвратить XSS-атаки.

Валидация — это важная задача, за которую отвечают и фронтендеры, и бэкендеры. Первые должны настроить валидацию на стороне клиента. Вторые — позаботиться о проверке на стороне сервера.

Если же вы работаете над проектом в одиночку, валидация — только ваша задача. Не пренебрегайте ею, чтобы ваше веб-приложение работало без ошибок и оставалось безопасным.

> ***Примечание***. Для уверенного освоения материала необходимо повторить следующие уроки курса:
>
> Тема: **Отправка данных на сервер**:
>
> - Урок 3. **Отправка форм**
> - Урок 6. **Регулярные выражения, часть 1**
> - Урок 7. **Регулярные выражения, часть 2**

В уроке предложены коды демонстрационных примеров, реализующих валидацию пользовательских данных на стороне сервера. В

самостоятельных работах закрепляем навыки дополнительными заданиями.

> ***Важно***. **Валидация данных на стороне клиента** (несмотря на заголовок следующего раздела) действие не менее важное, но не входящее в перечень рассматриваемых в курсе тем.

Клиенты — это зло

Очень часто в сети Интернет, да и вообще, можно услышать фразу, что проверку на **стороне клиента** легко обойти, именно поэтому так важно валидировать данные на **стороне сервера**. А все ли понимают, что это значит и как обойти данные на стороне клиента? Разберемся.

В демонстрационном примере **example_1** вам предложена форма, которая имеет поля с некоторыми правилами заполнения.

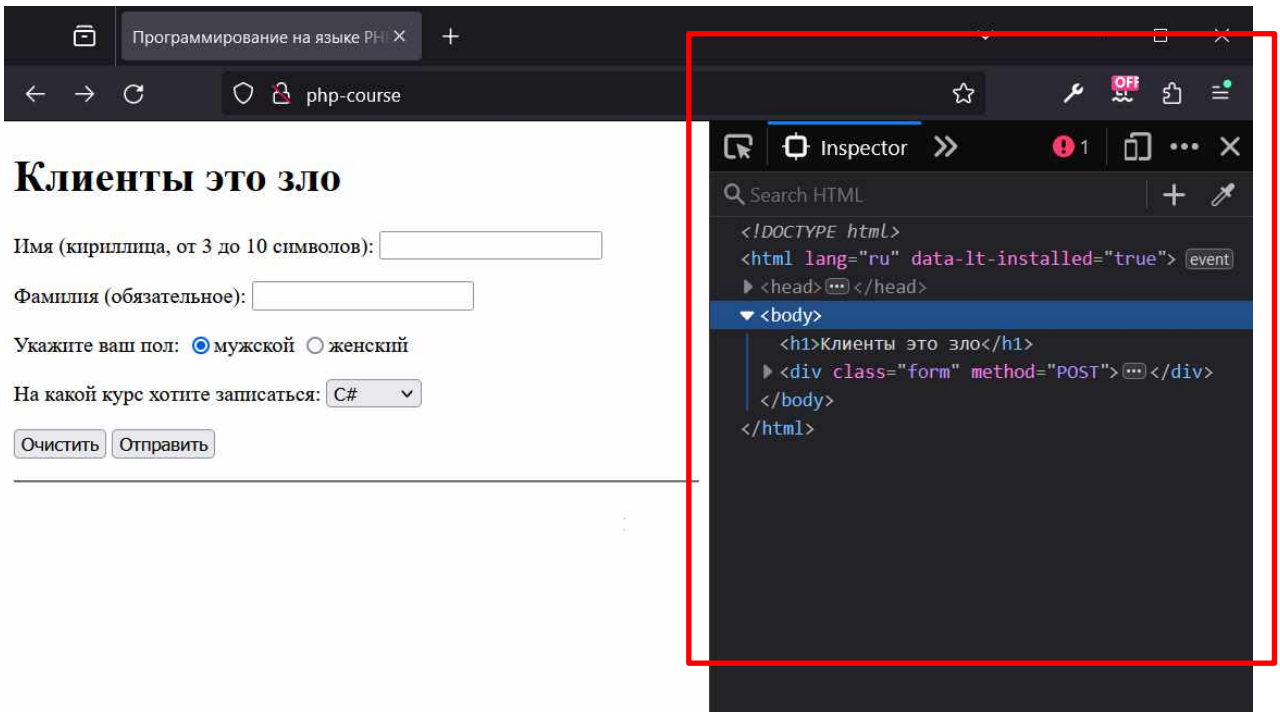
Поле формы	Правила заполнения
Имя	Кириллица, от 3 до 10 символов
Фамилия	Обязательное
Укажите ваш пол	• Мужской (m) • Женский (f)
На какой курс хотите записаться	• C# • Python • PHP • Java

example_1. index.html

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

    <p>Имя (кириллица, от 3 до 10 символов):
```

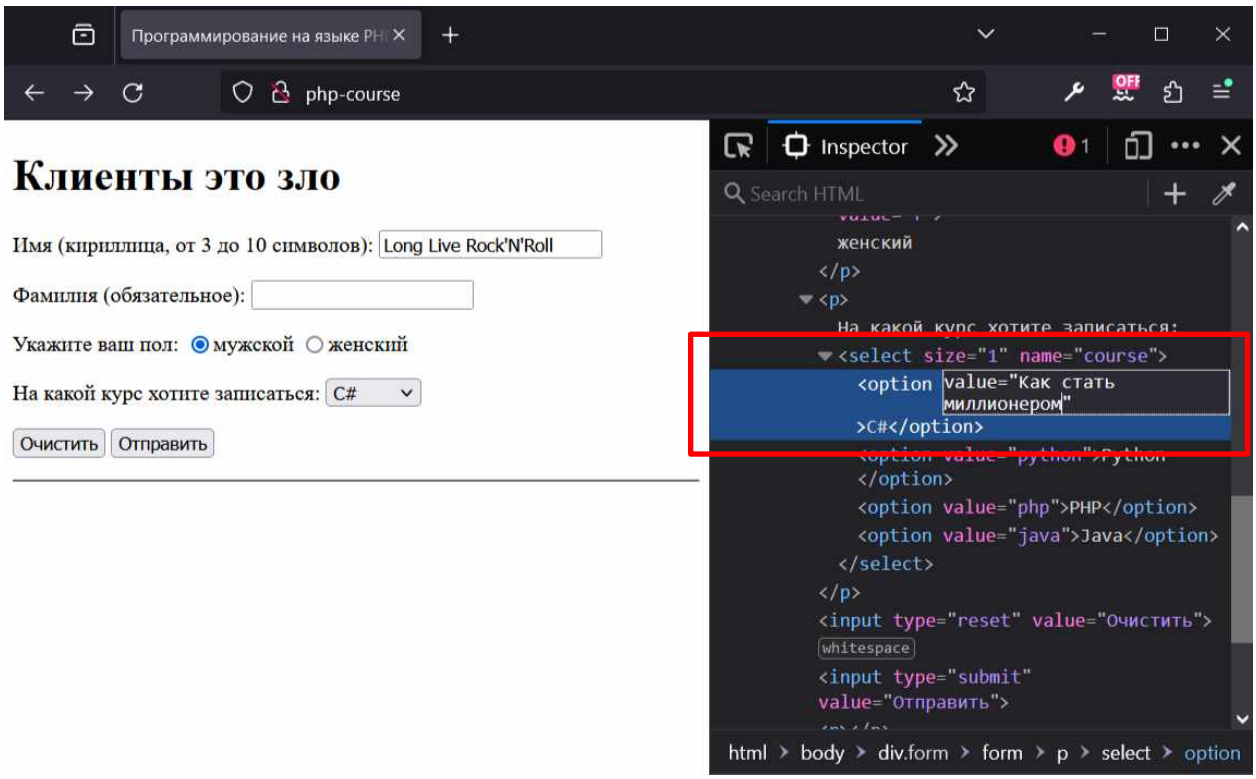
```
        <input pattern="[а-яА-ЯёЁ]{3,10}" type="text"
        name="name">
    </p>
    <p>Фамилия (обязательное):
        <input required type="text" name="surname">
    </p>
    <p>Укажите ваш пол:
        <input type="radio" name="sex" value="m" checked>мужской
        <input type="radio" name="sex" value="f">женский
    </p>
    <p>На какой курс хотите записаться:
        <select name="course">
            <option value="c#">C#</option>
            <option value="python">Python</option>
            <option value="php">PHP</option>
            <option value="java">Java</option>
        </select>
    </p>
        <input type="reset" value="Очистить">
        <input type="submit" value="Отправить"></p>
    <hr />
</form>
```

Открываем форму, заходим в **режим разработчика** (F12) и спокойно правим код формы.



В предложенном примере я внес следующие изменения:

- Убрал атрибут **pattern** поля **Имя** (name).
- Убрал атрибут **required** поля **Фамилия** (surname).
- Сделал шаг в пользу европейской толерантности в поле: **Укажите ваш пол** (sex) :(
- Записался на курс: **Как стать миллионером**. И это не беда, что такого курса нет (ну я же хочу...).

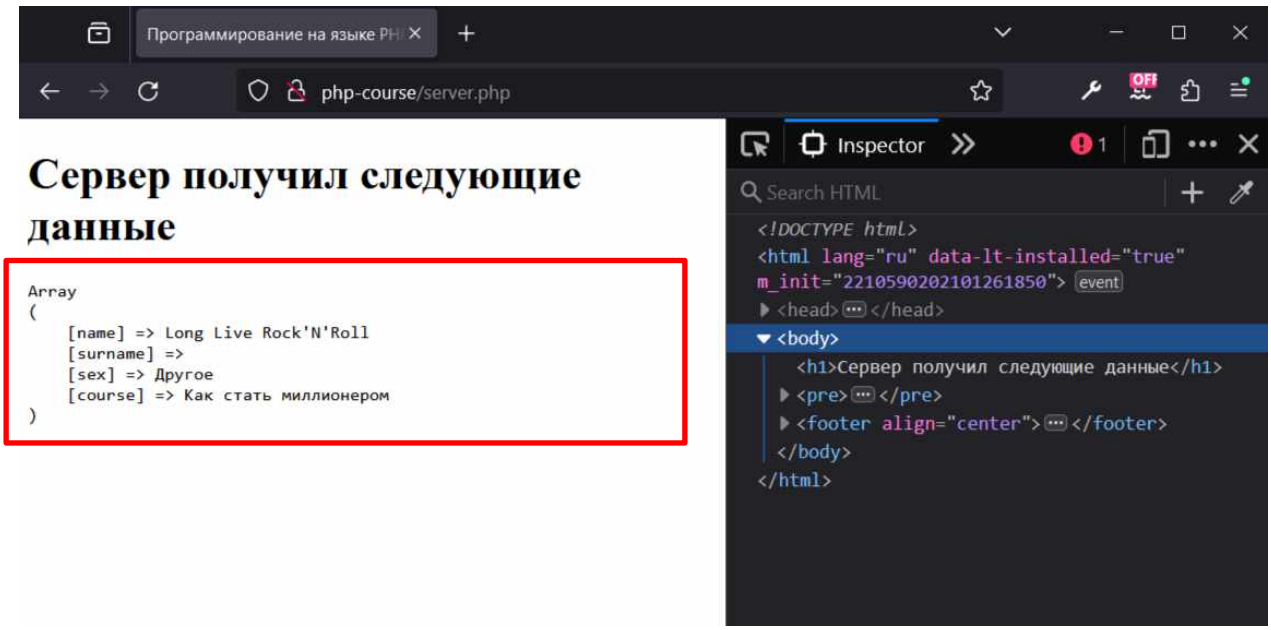


После внесения всех изменений, отправил форму на сервер.

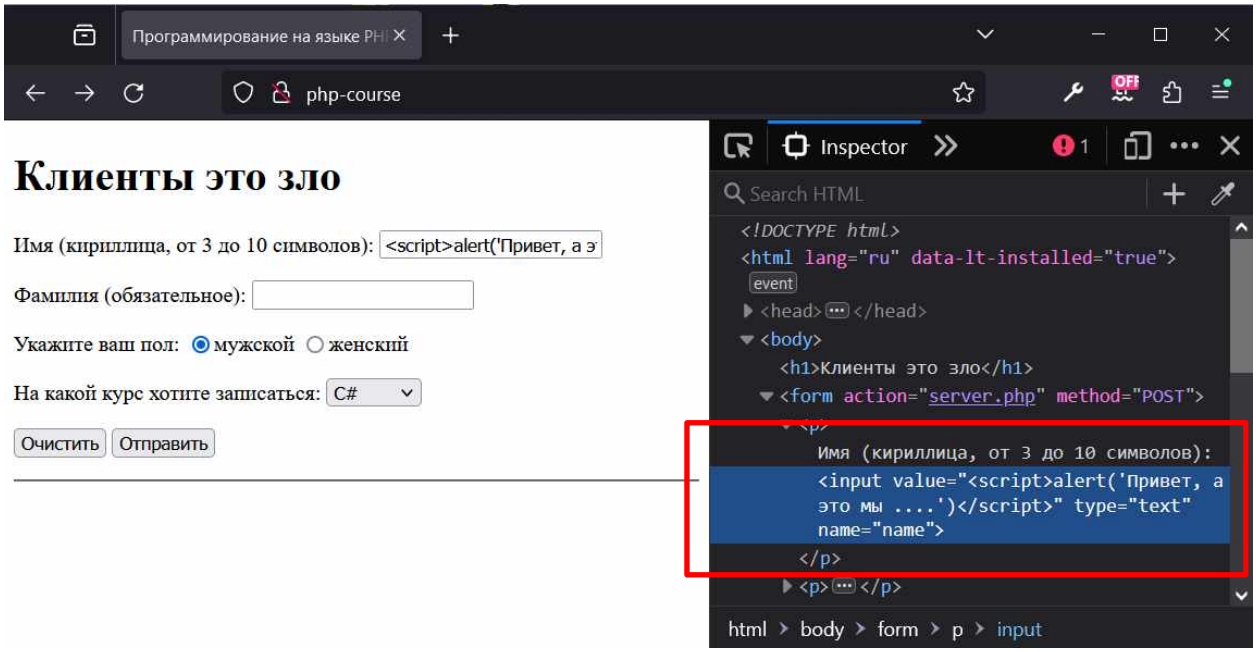
example_1. server.php

```
<h1>Сервер получил следующие данные</h1>

<?php

    echo "<pre>";

    print_r($_POST);

    echo "</pre>";

?>
```

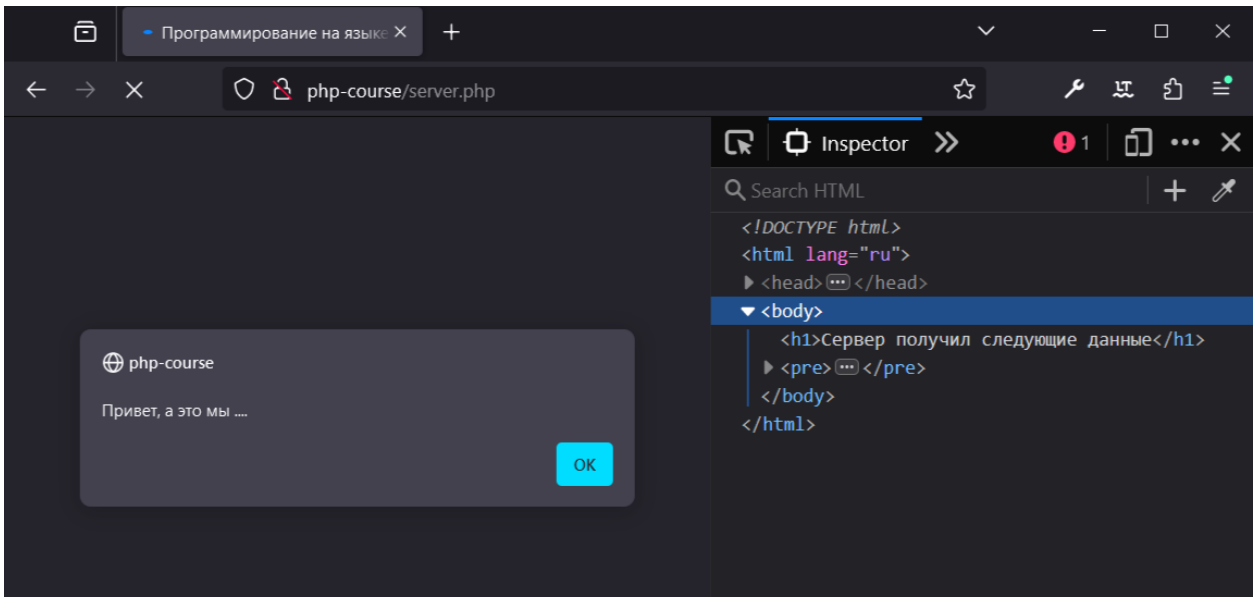
На выходе получил данные, представленные на следующем скриншоте.



Нарушены все правила, которые только можно было нарушить. Но это еще половина беды. С таким же успехом можно внедрить и клиентский скрипт.



И это еще хорошо, что **вредоносный скрипт**, отправленный на сервер, просто здоровается. Может быть и хуже, но это другая история



Надеюсь, после всех экспериментов необходимость применения валидации на стороне сервера стала очевидной.

Простая валидация

Валидация на стороне сервера тоже может быть разной. Для разогрева выполним несколько простых примеров.

Демонстрационный пример example_2

В демонстрационном примере валидируем поля раздела **Персональные данные**:

- **Фамилия**
- **Имя**
- **Отчество**

Правила валидации для полей определены в следующих таблицах.

Фамилия			
разрешенные символы	**количество символов**	**обязательное**	**regExp**
кириллица (строчные,	3 - 25	да	`/^[а-яёА-ЯЁ\s-]{3,25}$/u`

прописные), пробел, тире			
пример	Петров, Сумароков-Эльстон, де Сент-Экзюпери		

Имя			
разрешенные символы	**количество символов**	**обязательное**	**regExp**
кириллица (строчные, прописные), пробел	3 - 15	да	`/^[а-яёА-ЯЁ\s]{3,15}$/u`
пример	Татьяна, Евгений		

Отчество			
разрешенные символы	**количество символов**	**обязательное**	**regExp**
кириллица (строчные, прописные)	3 - 20	нет	`/^[а-яёА-ЯЁ]{3,20}$/u`
пример	Сергеевич, Александровна		

Обратите внимание, посетитель сайта с именем Ян под правила валидного имени не попадает.

example_2. index.php

```
<?php

    // если есть ошибка заполнения обязательных полей

    if (isset($_GET["err-required"]))

        echo "<div id='msg'>{$_GET['err-required']}</div>";

?>

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

    <p>Фамилия (кириллица, тире, пробел; 3 - 25 символов):</p>

    <!-- шаблон: кириллица (строчные, прописные), тире, пробел;
```

```
    от 3 до 25 символов -->
    <input type="text" name="surname" value="Панкратов" required>
    <p>Имя (кириллица, пробел; 3 - 15 символов):</p>
    <!-- шаблон: кириллица (строчные, прописные), пробел; от 3 до
    15 символов -->
    <input type="text" name="name" value="Инокентий" required>
    <p>Отчество (кириллица; 3 - 20 символов):</p>
    <!-- шаблон: кириллица (строчные, прописные); от 3 до 20
    символов -->
    <input type="text" name="patronymic" value="Караулович" >
    <p>
    <input type="reset" value="Очистить">
    <input type="submit" value="Отправить">
</form>
<p>P.S. * - сбор информации используется в корыстных целях</p>
```

example_2. server.php

```
<?php
    // сохраняем данные формы в переменные
    $surname = trim($_POST['surname']);
    $name = trim($_POST['name']);
    $patronymic = trim($_POST['patronymic']);
    // если обязательные поля пустые
    if (empty($surname) || empty($name)) {
        header('location:/?err-required=Заполните обязательные
        поля');
    }
```

```
?>

<!-- /// -->

<?php

    // echo "<pre>";

    // print_r ($_POST);

    // echo "</pre>";

    // массив сбора возможных ошибок

    $_ERROR_VALID = array();

    // проверяем на валидность данные формы

    if (!preg_match('/^[a-яёА-ЯЁ\s-]{3,25}$/u', $surname))
    $_ERROR_VALID['surname'] = $surname;

    if (!preg_match('/^[a-яёА-ЯЁ\s]{3,15}$/u', $name))
    $_ERROR_VALID['name'] = $name;

    if (!preg_match('/^[a-яёА-ЯЁ]{3,20}$/u', $patronymic))
    $_ERROR_VALID['patronymic'] = $patronymic;

    // echo "<pre>";

    // print_r ($_ERROR_VALID);

    // echo "</pre>";

    // если массив ошибок не пуст

    if(count($_ERROR_VALID)){

        foreach ($_ERROR_VALID as $key => $val) {

            printf ('<p>Ошибка данных в поле \'%s\': %s. </p>',
            $key, $val);

        }

    } else

        echo "<h3>Данные валидны</h3>";

?>
```

Демонстрационный пример example_3

В демонстрационном примере валидируем поля раздела **Данные портала Госуслуг**:

- **Логин**
- **Пароль**

Правила валидации для полей определены в следующих таблицах.

Логин			
разрешенные символы	**количество символов**	**обязательное**	**regExp**
кириллица, латиница (строчные, прописные), цифры от 0 до 9	5 - 15	да	`/^[a-zA-Z0-9]{5,15}$/u`
пример	Moderator12, superAdmin		

С полем пароль все немного (чуть-чуть) сложнее. Есть соблазн написать следующее правило:

`/^[0-9a-zA-Z!@#$%^&*]{6,20}$/`

но тогда валидными будут значения:

- 123456
- аааааааа

Но наша задача, как раз, предостеречь потенциального пользователя от использования таких паролей. Поэтому мы должны указать, что в пароле обязательно **должны присутствовать**:

- **цифра**;
- символ в **верхнем** регистре;
- **специальный** символ;

- и при этом пароль должен соответствовать правилу: `/^[0-9a-zA-Z!@#$%^&*]{6,20}$/`

Используем для этого правило **позитивного просмотра вперед (?= …)** (опережающий поиск /lookahead):

- `(?=.*[0-9])`
- `(?=.*[A-Z])`
- `(?=.*[!@#$%^&*])`

Что это означает:

- `(?=.*[0-9])` – заглянем вперед, есть ли в строке любой символ любое количество раз (.*), а затем число в диапазоне – [0-9].
- `(?=.*[A-Z])` – заглянем вперед, есть ли в строке любой символ любое количество раз (.*), а затем символ верхнего регистра в диапазоне – [A-Z].
- `(?=.*[!@#$%^&*])` – заглянем вперед, есть ли в строке любой символ любое количество раз (.*), а затем любой символ из диапазона – [!@#$%^&*].

И только если все три заглядывания вперед вернут **истину**, проверим основное правило регулярного выражения: `/^[0-9a-zA-Z!@#$%^&*]{6,20}$/`

Таким образом, правило проверки валидности пароля приобрело вид:

`/^(?=.*[0-9])(?=.*[A-Z])(?=.*[!@#$%^&*])[0-9a-zA-Z!@#$%^&*]{6,20}$/`

Пароль			
разрешенные символы	**количество символов**	**обязательное**	**regExp**
латиница (строчные, прописные), цифры от 0 до 9 и символы:	6 - 20	да	`/^(?=.*[0-9])(?=.*[A-Z])(?=.*[!@#$%^&*])[0-9a-zA-Z!@#$%^&*]{6,20}$/`

!@#$%^&*			
пример	Password@123*, creaTive@#!		

example_3. index.php

```
<?php
    // если есть ошибка заполнения обязательных полей
    if (isset($_GET["err-required"]))
        echo "<div id='msg'>{$_GET['err-required']}</div>";
?>
<form action="server.php" method="post">
    <p>Логин (кириллица, латиница, цифры от 0 до 9; 5 - 15 символов):</p>
    <!--
    шаблон: кириллица, латиница (строчные, прописные), цифры от 0 до 9;
    от 5 до 15 символов
    -->
    <input type="text" name="login" value="moderator" required>
    <p>Пароль (латиница, символы !@#$%^&*, цифры от 0 до 9; 6 - 20 символов):</p>
    <!--
    шаблон: латиница (строчные, прописные), символы: !@#$%^&*, цифры от 0 до 9;
    от 6 до 20 символов
    -->
    <input type="password" name="pwd" value="Password@123*" required>
    <p>
```

```
    <input type="reset" value="Очистить">
    <input type="submit" value="Отправить">
</form>
<p>P.S. * - сбор информации используется в корыстных целях</p>
```

Кроме позитивного просмотра вперед регулярные выражения поддерживают **позитивный просмотр назад – (?<=)** (ретроспективный поиск / lookbehind).

> ***Примечание***. В следующих примерах используйте правило **позитивного просмотра вперед**, там, где посчитаете нужным.

example_3. server.php

```
<?php
    // сохраняем данные формы в переменные
    $login = trim($_POST['login']);
    $pwd = trim($_POST['pwd']);
    // если обязательные поля пустые
    if (empty($login) || empty($pwd)) {
        header('location:/?err-required=Заполните обязательные
        поля');
    }
?>
<!-- /// -->
<?php
    // echo "<pre>";
    // print_r ($_POST);
    // echo "</pre>";
```

```php
    // массив сбора возможных ошибок
    $_ERROR_VALID = array();
    // проверяем на валидность данные формы
    if (!preg_match('/^[a-zA-Za-яёА-ЯЁ0-9]{5,15}$/u', $login))
    $_ERROR_VALID['login'] = $login;
    // простая проверка поля pwd
    // '/^[0-9a-zA-Z!@#$%^&*]{6,20}$/'
    // правильная проверка поля pwd
    if (!preg_match('/^(?=.*[0-9])(?=.*[A-Z])(?=.*[!@#$%^&*])[0-
    9a-zA-Z!@#$%^&*]{6,20}$/', $pwd)) $_ERROR_VALID['pwd'] =
    $pwd;
    // echo "<pre>";
    // print_r ($_ERROR_VALID);
    // echo "</pre>";
    // если массив ошибок не пуст
    if(count($_ERROR_VALID)){
    // выводим сообщения об ошибках пользователю
        foreach ($_ERROR_VALID as $key => $val) {
            printf ('<p>Ошибка данных в поле \'%s\': %s. </p>',
            $key, $val);
        }
    } else
        echo "<h3>Данные валидны</h3>";
?>
```

Примечание. Правила **опережающего** и **ретроспективного** поиска содержат возможность **захвата** части подстроки. Возможности регулярных выражений чрезвычайно велики. По регулярным

> выражениям можно создать отдельную тему.

Демонстрационный пример example_4

В демонстрационном примере валидируем поля раздела **Реквизиты банковской карты**:

- **Владелец**
- **Номер**
- **Действительна**
- **Код CVC**

Правила валидации для полей определены в следующих таблицах.

Владелец			
разрешенные символы	**количество символов**	**обязательное**	**regExp**
латиница (прописные), пробел	5 - 30	да	`/^[A-Z\s]{5,30}$/`
пример	PANKRATOV INOKENTIY		

Номер			
разрешенные символы	**количество символов**	**обязательное**	**regExp**
цифры от 0 до 9, пробел	группа из 4 цифр разделенные пробелом – 4 раза	да	`/ ^[0-9]{4}\s[0-9]{4}\s[0-9]{4}\s[0-9]{4}$/`
пример	1254 4568 4687 2135		

Действительна			
разрешенные символы	**количество символов**	**обязательное**	**regExp**

символ: /, цифры от 0 до 9	5	да	/^[0-9]{2}\/[0-9]{2}$/
пример	06/23		

Код CVC			
разрешенные символы	**количество символов**	**обязательное**	**regExp**
цифры от 0 до 9	3	да	/^ [0-9]{3}$/
пример	573		

example_4. index.php

```
<?php
    // если есть ошибка заполнения обязательных полей
    if (isset($_GET["err-required"]))
        echo "<div id='msg'>{$_GET['err-required']}</div>";
?>
<form action="server.php" method="post">
    <p>Владелец (латиница, пробел; 5 - 30 символов):</p>
    <!-- шаблон: латиница (прописные), пробел; от 5 до 30
    символов -->
    <input type="text" name="card_owner" value="PANKRATOV
    INOKENTIY" required>
    <p>Номер (4 цифры разделенные пробелом - 4 раза):</p>
    <!-- шаблон: группа из 4 цифр разделенные пробелом - 4 раза -
    ->
    <input type="text" name="card_number" value="1254 4568 4687
    2135" required>
    <p>Действительна до (2 цифры знак "/" и еще 2 цифры):</p>
```

```
    <!-- шаблон: 2 цифры знак "/" и еще 2 цифры -->
    <input type="text" name="card_date" value="12/25" required>
    <p>Код CVC (3 цифры):</p>
    <!-- шаблон: 3 цифры от 0 до 9 -->
    <input type="text" name="card_cvc" value="125" required>
    <p>
    <input type="reset" value="Очистить">
    <input type="submit" value="Отправить">
</form>
<p>P.S. * - сбор информации используется в корыстных целях</p>
```

example_4. server.php

```
<?php
    // сохраняем данные формы в переменные
    $card_owner = trim($_POST['card_owner']);
    $card_number = trim($_POST['card_number']);
    $card_date = trim($_POST['card_date']);
    $card_cvc = trim($_POST['card_cvc']);
    // если обязательные поля пустые
    if (
        empty($card_owner) ||
        empty($card_number) ||
        empty($card_date) ||
        empty($card_cvc)
    ) {
        header('location:/?err-required=Заполните обязательные
        поля');
```

```
    }
?>
<!-- /// -->
<?php
    // echo "<pre>";
    // print_r ($_POST);
    // echo "</pre>";
    // массив сбора возможных ошибок
    $_ERROR_VALID = array();
    // проверяем на валидность данные формы
    if (!preg_match('/^[A-Z\s]{5,30}$/', $card_owner))
    $_ERROR_VALID['card_owner'] = $card_owner;
    if (!preg_match('/^[0-9]{4}\s[0-9]{4}\s[0-9]{4}\s[0-9]{4}$/',
    $card_number)) $_ERROR_VALID['card_number'] = $card_number;
    if (!preg_match('/^[0-9]{2}\/[0-9]{2}$/', $card_date))
    $_ERROR_VALID['card_date'] = $card_date;
    if (!preg_match('/^[0-9]{3}$/', $card_cvc))
    $_ERROR_VALID['card_cvc'] = $card_cvc;
    // echo "<pre>";
    // print_r ($_ERROR_VALID);
    // echo "</pre>";
    // если массив ошибок не пуст
    if (count($_ERROR_VALID)){
        // выводим сообщения об ошибках пользователю
        foreach ($_ERROR_VALID as $key => $val) {
            printf ('<p>Ошибка данных в поле \'%s\': %s. </p>',
            $key, $val);
        }
```

```
    } else

        echo "<h3>Данные валидны</h3>";

?>
```

Демонстрационный пример example_5

В демонстрационном примере валидируем поля раздела **Разное**:

- **Вид документа**
- **E-mail**
- **Полных лет**
- **Дата заполнения** (проверяем встроенной функцией)

Правила валидации для полей определены в следующих таблицах.

Вид документа			
разрешенные символы	**количество символов**	**обязательное**	**regExp**
кириллица (строчные, прописные), пробел	7 – 25	нет	`/^[а-яА-Я\s]{7,25}$/u`
пример			

E-mail			
разрешенные символы	**количество символов**	**обязательное**	**regExp**
любой, должен присутствовать символ: @	нет ограничений	нет	`/@/`
пример			

Полных лет			
разрешенные	**количество**	**обязательное**	**regExp**

символы	символов		
цифры от 0 до 9	2	нет	/^[0-9]{2}$/
пример			

example_5. index.html

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

    <p>Вид документа:</p>

    <!-- шаблон: кириллица (строчные, прописные); от 7 до 25
    символов -->

    <select size="1" name="document">

        <option value="Паспорт">Паспорт</option>

        <option value="Удостоверение личности">Удостоверение
        личности</option>

        <option value="Водительские права">Водительские
        права</option>

        <option value="Купюра 100$">Купюра 100$</option>

    </select>

    <p>E-mail (стандартный):</p>

    <!-- в поле email достаточно использовать проверку на наличие
    символа @ -->

    <input type="email" name="email" value="1@1">

    <p>Полных лет на момент согласия (диапазон: 18-65):</p>

    <!-- стандартные ограничения на поле с type=number -->

    <!-- диапазон возможных значений от 18 до 65 -->

    <input type="number" min="18" max="65" name="age" value="20">

    <p>Дата заполнения согласия (гггг.мм.дд):</p>

    <!-- стандартные ограничение на type=date -->
```

```
    <input type="date" name="date" value="2020-07-15"></p>
    <p>Согласие на действия с персональными данными:
    <input type="checkbox" name="approval" value="no">
    <p>
    <input type="reset" value="Очистить">
    <input type="submit" value="Отправить">
</form>
<p>P.S. * - сбор информации используется в корыстных целях</p>
```

example_5. server.php

```
<?php
    // echo "<pre>";
    // print_r ($_POST);
    // echo "</pre>";
    // проверяем, есть ли согласие на обработку данных
    if (isset($_POST['approval'])) {
        // массив сбора возможных ошибок
        $_ERROR_VALID = array();
        // сохраняем данные формы в переменные
        $document = trim($_POST['document']);
        $email = trim($_POST['email']);
        $age = trim($_POST['age']);
        $date = trim($_POST['date']);
        // проверяем на валидность данные формы
        // пункт - Купюра 100$ - не пройдет :((
        if (!preg_match('/^[а-яА-Я\s]{7,25}$/u', $document))
        $_ERROR_VALID['document'] = $document;
```

```php
        if (!preg_match('/@/', $email)) $_ERROR_VALID['email'] =
        $email;
        if (!preg_match('/^[0-9]{2}$/', $age))
        $_ERROR_VALID['age'] = $age;
        if (!strtotime($date)) $_ERROR_VALID['date'] = $date;
        // echo "<pre>";
        // print_r ($_ERROR_VALID);
        // echo "</pre>";
        // если массив ошибок не пуст
        if(count($_ERROR_VALID)){
            // выводим сообщения об ошибках пользователю
            foreach ($_ERROR_VALID as $key => $val) {
                printf ('<p>Ошибка данных в поле \'%s\': %s.
                </p>', $key, $val);
            }
        } else
        echo "<h3>Данные валидны</h3>";
    } else {
        echo '<h3>Для работы с данными необходимо согласие на их
        обработку</h3>';
    echo '<a href="/">Назад</a>';
    }
?>
```

Продвинутая валидация

Пришло время навернуть немного сложности в наш код, сделать его более профессиональным (запутанным или).

Демонстрационный пример example_6

Вернемся к форме **Персональных данных** и ревьюируем код валидации.

example_6. index.php

```
<?php
    // если есть ошибка заполнения обязательных полей
    if (isset($_GET["err-required"]))
        echo "<div id='msg'>{$_GET['err-required']}</div>";
?>
<form action="server.php" method="post">
    <p>Фамилия (кириллица, тире, пробел; 3 - 25 символов):</p>
    <!-- шаблон: кириллица (строчные, прописные), тире, пробел;
    от 3 до 25 символов -->
    <input type="text" name="surname" value="Панкратов" required>
    <p>Имя (кириллица, пробел; 3 - 15 символов):</p>
    <!-- шаблон: кириллица (строчные, прописные), пробел; от 3 до
    15 символов -->
    <input type="text" name="name" value="Инокентий" required>
    <p>Отчество (кириллица; 3 - 20 символов):</p>
    <!-- шаблон: кириллица (строчные, прописные); от 3 до 20
    символов -->
    <input type="text" name="patronymic" value="Караулович" >
    <p>
    <input type="reset" value="Очистить">
    <input type="submit" value="Отправить">
</form>
<p>P.S. * - сбор информации используется в корыстных целях</p>
```

Индексный файл мы уже разбирали. А вот в сценарии **server.php** есть изменения:

- вынесли правила валидации в **отдельный массив**,
- в массиве сбора возможных ошибок используем **названия ключей на кириллице** (упс...).

example_6. server.php

```
<?php

    // массив сбора возможных ошибок

    $_ERROR_VALID = array();

    // массив правил валидации полей формы

    $_VALIDATION_RULE = [

        'surname' => '/^[a-яёА-ЯЁ\s-]{3,25}$/u',

        'name' => '/^[a-яёА-ЯЁ\s]{3,15}$/u',

        'patronymic' => '/^[a-яёА-ЯЁ]{3,20}$/u'

    ];

    // проверяем на валидность данные формы

    if (!preg_match($_VALIDATION_RULE['surname'], $surname))
    $_ERROR_VALID['Фамилия'] = $surname;

    if (!preg_match($_VALIDATION_RULE['name'], $name))
    $_ERROR_VALID['Имя'] = $name;

    if (!preg_match($_VALIDATION_RULE['patronymic'],
    $patronymic)) $_ERROR_VALID['Отчество'] = $patronymic;

    // echo "<pre>";

    // print_r ($_ERROR_VALID);

    // echo "</pre>";

    // если массив ошибок не пуст

    if(count($_ERROR_VALID)){
```

```
        // выводим сообщения об ошибках пользователю
        foreach ($_ERROR_VALID as $key => $val) {
            printf ('<p>Ошибка данных в поле \'%s\': %s. </p>',
            $key, $val);
        }
    } else
    echo "<h3>Данные валидны</h3>";
?>
```

Называя ключи массива кириллицей, вы рискуете развернуть бурную дискуссию окружающих вас программистов. В этой дискуссии кроме новых фактов о своей персоне ;), можно услышать примерно равное количество аргументов как "за", так и "против".

> ***Тем не менее***. Никаких ограничений на использование кириллицы в ключах массива нет. Поддерживается и работает без проблем.
>
> Аргументов "против" нет, кроме довольно условной **эстетики** программирования и **стандартов** именования переменных.

И все-таки, в реальном приложении я бы предпочел обойтись без использования кириллических символов.

Демонстрационный пример example_7

Переходим к заключительным демонстрационным примерам. Объединяем разделы файла формы в один большой файл. Тем интереснее будет оценить ревьюированный обработчик такой формы – сценарий файла **server.php**.

example_7. index.php

```
<?php
```

```
    // если есть ошибка заполнения обязательных полей
    if (isset($_GET["err-required"]))
        echo "<div id='msg'>{$_GET['err-required']}</div>";
?>
<form action="server.php" method="post">
    <h2>Персональные данные</h2>
    <p>Фамилия (кириллица, тире, пробел; 3 - 25 символов):</p>
    <!-- шаблон: кириллица (строчные, прописные), тире, пробел;
    от 3 до 25 символов -->
    <input type="text" name="surname" value="Панкратов" required>
    <p>Имя (кириллица, пробел; 3 - 15 символов):</p>
    <!-- шаблон: кириллица (строчные, прописные), пробел; от 3 до
    15 символов -->
    <input type="text" name="name" value="Инокентий" required>
    <p>Отчество (кириллица; 3 - 20 символов):</p>
    <!-- шаблон: кириллица (строчные, прописные); от 3 до 20
    символов -->
    <input type="text" name="patronymic" value="Караулович" >
    <p>Укажите ваш пол</p>
    <p>
    <input type="radio" name="sex" value="m" checked>мужской
    <input type="radio" name="sex" value="f">женский
    <input type="radio" name="sex" value="h">другой
    </p>
    <p><hr /></p>
    <h2>Данные портала Госуслуг</h2>
    <p>Логин (кириллица, латиница, цифры от 0 до 9; 5 - 15
    символов):</p>
```

```
<!-- шаблон: кириллица, латиница (строчные, прописные), цифры от 0 до 9; от 5 до 15 символов -->
<input type="text" name="login" value="moderator" required>
<p>Пароль (латиница, символы !@#$%^&*, цифры от 0 до 9; 6 - 20 символов):</p>
<!-- шаблон: латиница (строчные, прописные), символы: !@#$%^&*, цифры от 0 до 9; от 6 до 20 символов -->
<input type="password" name="pwd" value="Password@123*" required>
<p><hr /></p>
<h2>Реквизиты банковской карты</h2>
<p>Владелец (латиница, пробел; 5 - 30 символов):</p>
<!-- шаблон: латиница (прописные), пробел; от 5 до 30 символов -->
<input type="text" name="card_owner" value="PANKRATOV INOKENTIY" required>
<p>Номер (4 цифры разделенные пробелом - 4 раза):</p>
<!-- шаблон: группа из 4 цифр разделенные пробелом - 4 раза - ->
<input type="text" name="card_number" value="1254 4568 4687 2135" required>
<p>Действительна до (2 цифры знак "/" и еще 2 цифры):</p>
<!-- шаблон: 2 цифры знак "/" и еще 2 цифры -->
<input type="text" name="card_date" value="12/25" required>
<p>Код CVC (3 цифры):</p>
<!-- шаблон: 3 цифры от 0 до 9 -->
<input type="text" name="card_cvc" value="125" required>
<p><hr /></p>
<h2>Разное</h2>
```

```
    <p>Вид документа:</p>
    <!-- шаблон: кириллица (строчные, прописные); от 7 до 25
    символов -->
    <select size="1" name="document">
        <option value="Паспорт">Паспорт</option>
        <option value="Удостоверение личности">Удостоверение
        личности</option>
        <option value="Водительские права">Водительские
        права</option>
        <option value="Купюра 100$">Купюра 100$</option>
    </select>
    <p>E-mail (стандартный):</p>
    <!-- в поле email достаточно использовать проверку на наличие
    символа @ -->
    <input type="email" name="email" value="1@1">
    <p>Полных лет на момент согласия (диапазон: 18-65):</p>
    <!-- стандартные ограничения на поле с type=number -->
    <!-- диапазон возможных значений от 18 до 65 -->
    <input type="number" min="18" max="65" name="age" value="20">
    <p>Дата заполнения согласия (гггг.мм.дд):</p>
    <!-- стандартные ограничение на type=date -->
    <input type="date" name="date" value="2020-07-15"></p>
    <p>Согласие на действия с персональными данными:
    <input type="checkbox" name="approval" value="yes">
    <p>
    <input type="reset" value="Очистить">
    <input type="submit" value="Отправить">
</form>
```

```
<h2>Спасибо за понимание! ;)</h2>
```

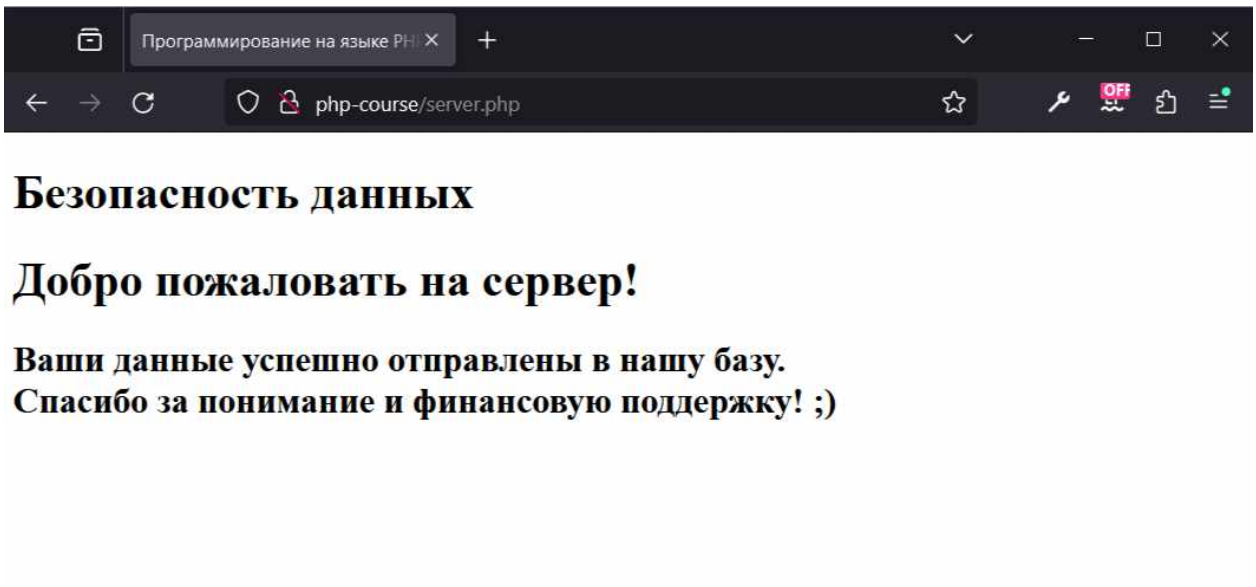
В сценарии файла **server.php** основной объем кода занимает разросшийся массив правил валидации полей формы и отладочная информация в комментариях просмотра содержимого массивов $_**POST** и $_ **ERROR_VALID**.

example_7. server.php

```
<?php
    // массив сбора возможных ошибок
    $_ERROR_VALID = array();
    // массив правил валидации полей формы
    $_VALIDATION_RULE = [
        'surname' => '/^[a-яёА-ЯЁ\s-]{3,25}$/u',
        'name' => '/^[a-яёА-ЯЁ\s]{3,15}$/u',
        'patronymic' => '/^[a-яёА-ЯЁ]{3,20}$/u',
        'login' => '/^[a-zA-Za-яёА-ЯЁ0-9]{5,15}$/u',
        'pwd' => '/^(?=.*[0-9])(?=.*[A-Z])(?=.*[!@#$%^&*])[0-9a-
        zA-Z!@#$%^&*]{6,20}$/',
        'card_owner' => '/^[A-Z\s]{5,30}$/',
        'card_number' => '/^[0-9]{4}\s[0-9]{4}\s[0-9]{4}\s[0-
        9]{4}$/',
        'card_date' => '/^[0-9]{2}\/[0-9]{2}$/',
        'card_cvc' => '/^[0-9]{3}$/',
        'document' => '/^[a-яА-Я\s]{7,25}$/u',
        'email' => '/@/',
        'age' => '/^[0-9]{2}$/',
        'date' => '/^[0-9]{4}\-[0-9]{2}\-[0-9]{2}$/',
        'sex' => '/^[mfh]{1}$/',
```

```
        'approval' => '/^yes$/'
    ];
    // цикл проверки полей правилам регулярных выражений
    foreach ($_POST as $key => $val){
        // для каждого элемента массива POST обрезаем пробелы
        $_POST[$key] = trim($_POST[$key]);
        // проверяем на валидность текущий элемент массива POST
        if (!preg_match($_VALIDATION_RULE[$key], $_POST[$key]))
        $_ERROR_VALID[$key] = $val;
    };
    // echo "<pre>";
    // var_dump ($_POST);
    // echo "</pre>";
    // echo "<pre>";
    // print_r ($_ERROR_VALID);
    // echo "</pre>";
?>
```

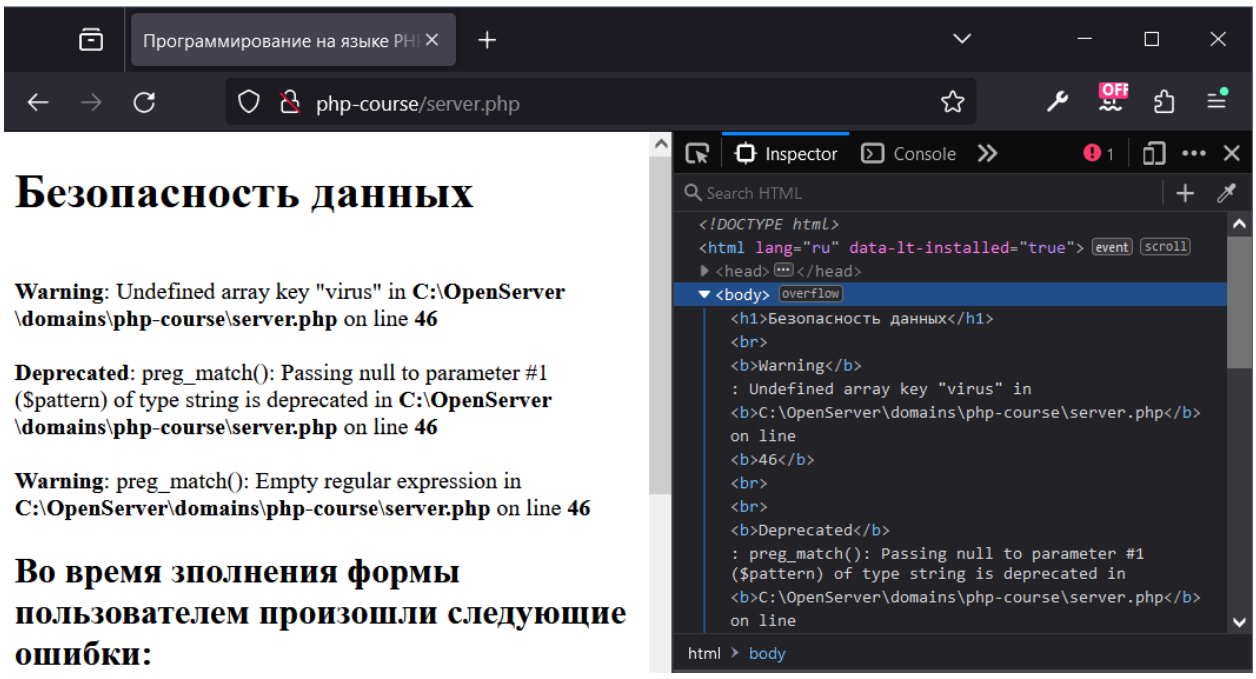
На следующем скриншоте показан результат **валидной проверки данных** пользовательской формы.



Все хорошо до того момента, пока мы снова не включимся в игру хороший-плохой пользователь. Зайдем в режим разработчика стартовой страницы и **добавим в форму следующие** элементы:

```
<p>
<input type="text" name="virus" value="Hello, I'm virus!">
```

Еще раз отправим форму на сервер. Результат предсказуем, в массиве $_**VALIDATION_RULE** нет ключа с именем **virus**.



В следующем примере рассмотрим другой подход к проверке.

Демонстрационный пример example_8

Если в демонстрационном примере **example_7** мы итерировали массив $_**POST** (поэтому получили ошибку от злонамеренного клиента), то теперь будем итерировать массив $_**VALIDATION_RULE**.

> ***Важно***. Именно в этом принципиальное отличие между демонстрационными примерами **example_7** и **example_8**. Будьте внимательны при наборе и анализе кода примеров.

Есть еще небольшие отличия данного примера:

- вынесли массив в отдельный файл (**config.php**);
- расширили полезную информацию массива с правилами валидации.

Ввиду большого объема, код файла **config.php** публикую с небольшим сокращением.

example_8. config.php

```
<?php

    // сообщение - приветсвие об успешном заполнении полей формы

    $welcome = "<h1>Добро пожаловать на сервер!</h1>";

    $welcome .= "<h2>Ваши данные успешно отправлены в нашу
    базу.<br /> Спасибо за понимание и финансовую поддержку!
    ;)</h2>";

    // сообщение об ошибке валидации формы

    $error_valid_form = "<h2>Во время заполнения формы
    пользователем произошли следующие ошибки:</h2>";

    // сообщение об ошибке заполнения поля формы

    $error_valid_field = 'Ошибка заполнения поля: <b> %s </b>

    <ul>
```

```
    <li>Описание ошибки: %s</li>
    <li>Правило: %s</li>
    <li>Значение: %s</li>
    </ul>';
    // массив правил валидации плюс дополнительная информация
    $_VALIDATION_RULE = [
    'surname' => [
    'Фамилия',
    '/^[a-яёA-ЯЁ\s-]{3,25}$/u',
    'Поле не может содержать символы кроме указанных в правиле'
    ],
    'name' => [
    'Имя',
    '/^[a-яёA-ЯЁ\s]{3,15}$/u',
    'Поле не может содержать символы кроме указанных в правиле'
    ],
    /* .... */
    'sex' => [
    'Пол',
    '/^[mfh]{1}$/',
    'Простите, но у нас есть только мужчины и женщины'
    ],
    'approval' => [
    'Согласие на обработку',
    '/^yes$/',
    'Согласие на обработку данных придется дать ;)'
    ]
```

```
    ];
?>
```

Файл формы идентичен примеру из **example_7**.

Окончательный код обработчика формы представлен в файле **server.php**.

example_8. server.php

```
<?php
    // подключим файл конфигурации
    include "config.php";
    // массив для сбора информации о возможных ошибках проверки
    $_ERROR_VALID = array();
    // проверяем в цикле соответсвие пользовательских данных
    // правилам регулярных выражений
    foreach ($_VALIDATION_RULE as $key => $val){
        // если клиент вмешается в код формы
        $_POST[$key] = isset($_POST[$key])? trim($_POST[$key]) :
        '';
        // перебираем ключи массива $_VALIDATION_RULE (ключи
        массива - это названия полей формы)
        if (!preg_match($_VALIDATION_RULE[$key][1],
        $_POST[$key])) $_ERROR_VALID[$key] =
        $_VALIDATION_RULE[$key];
    };
    // echo "<pre>";
    // var_dump ($_POST);
    // echo "</pre>";
    // echo "<pre>";
```

```
    // print_r($_ERROR_VALID);
    // echo "</pre>";
    // если массив ошибок не пуст, соберем ошибки в строку
    if(count($_ERROR_VALID)){
        $error_message = $error_valid_form;
        foreach ($_ERROR_VALID as $key => $item) {
            $error_message .= sprintf (
            $error_valid_field,
            $item[0],
            $item[2],
            $item[1],
            $_POST[$key]
            );
        }
    }
?>
<!-- /// -->
<?php
    // если переменная не определена - Добро пожаловать
    if (empty($error_message)){
        // выводим что-то приветственное пользователю,
        // который смог правильно заполнить форму ))
        echo $welcome;
    } else {
        // или выводим блок ошибок
        echo $error_message;
    };
```

```
?>
```

Заключение

С развитием интернет-технологий безопасность приложений стала одной из наиболее актуальных и важных тем в области информационной безопасности.

Веб-приложения используются повсеместно для обработки чувствительных данных, таких как личная информация пользователей, банковские и корпоративные данные. Несоблюдение стандартов безопасности может привести к утечкам данных, нарушению конфиденциальности и потере репутации ресурса. Обеспечение безопасности веб-приложений является критически важным аспектом.

Лучшие практики безопасности веб-приложений

Валидация входных данных. Одной из основных атак, на которые подвергаются веб-приложения, является атака на входные данные. Разработчики должны всегда проверять и валидировать данные, получаемые от пользователей, чтобы предотвратить SQL-инъекции, кросс-сайтовый скриптинг (XSS) и другие атаки.

Защита от аутентификации и управление сеансами. Управление сеансами и аутентификация пользователей — это ключевые аспекты безопасности веб-приложений. Надежная аутентификация с использованием сильных паролей и двухфакторной аутентификации обязательна. Кроме того, сессии пользователей должны быть защищены от перехвата и подделки.

Обновление и патчи. Регулярное обновление и установка патчей для веб-приложений и используемых библиотек и фреймворков — это обязательное условие для обеспечения безопасности. Уязвимости,

обнаруженные в сторонних компонентах, могут стать легкой добычей для злоумышленников.

Ограничение прав доступа. Применение принципа наименьших привилегий важно для уменьшения рисков. Пользователи и компоненты приложения должны иметь только те права доступа, которые необходимы для выполнения своих задач.

Мониторинг и журналирование. Ведение журналов действий пользователей и системных событий позволяет обнаруживать аномалии и атаки в реальном времени. Мониторинг позволяет оперативно реагировать на угрозы и предотвращать утечки данных.

Основные виды уязвимостей веб-приложений

SQL-инъекции. SQL-инъекции возникают, когда злоумышленник внедряет вредоносный SQL-код в запросы к базе данных. Это может привести к незаконному доступу к данным или их изменению. Правильная валидация и использование параметризованных запросов помогают предотвратить SQL-инъекции.

Кросс-сайтовый скриптинг (XSS). XSS — это атака, при которой злоумышленник внедряет вредоносный JavaScript-код в веб-страницу, который выполняется в браузере пользователя. Защита от XSS включает в себя экранирование данных и использование Content Security Policy (CSP).

Кросс-сайтовая подделка запроса (CSRF). CSRF — это атака, при которой злоумышленник заставляет пользователя выполнять нежелательные действия без его согласия. Защита от CSRF включает в себя использование токенов запросов (CSRF-токены) и проверку Referer-заголовка.

Недостатки аутентификации и управления сеансами. Слабая аутентификация и управление сеансами могут привести к

компрометации аккаунтов пользователей. Для защиты следует использовать сильные пароли, двухфакторную аутентификацию и надежное управление сеансами.

> ***Примечание***. Текущая тема целиком посвящена именно вопросам безопасной обработке данных.

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 информационными угрозами.