Министерство образования и науки Российской Федерации

Севастопольский государственный университет

Кафедра ИС

Отчет

По дисциплине: “Web-технологии”

Лабораторная работа №1

“ Исследование архитектуры MVC приложения и возможностей обработки данных HTML-форм на стороне сервера с использованием языка PHP”

Выполнил:

ст.гр. ИС/б-17-2

Долженко И.А.

Проверил:

Овчинников А.Л.

Севастополь

2020

1 ЦЕЛЬ РАБОТЫ

Изучить основы синтаксиса PHP и принципы функционирования MVC приложения на стороне сервера. Приобрести практические навыки использования языка PHP для генерации HTML-кода и обработки HTML-форм в MVC приложении.

2 ПОСТАНОВКА ЗАДАЧИ

1. Для страницы «Фотоальбом» реализовать хранение данных фото (подписей и имен файлов) в соответствующих константах модели Photo. Вывод таблицы, содержащей фото, реализовать в представлении с использованием операторов циклов.

2. Для страницы «Мои интересы» реализовать хранение данных (название категорий, интересов, описание интересов) в соответствующих константах модели Interest. Вывод меню ссылок и интересов реализовать в представлении с использованием операторов циклов.

3. Реализовать класс FormValidation, выполняющий валидацию данных форм, передаваемых на сторону сервера. Рекомендуемая структура класса:

- Rules – поле(массив), содержащее набор правил для проверки валидности данных;

- Errors – поле(массив), содержащее тексты ошибок, возникших при проверке валидности данных;

- isNotEmpty(data) – метод проверки является ли значение data не пустым – возвращает сообщение об ошибке, если таковая имеется;

- isInteger(data) – метод проверки является ли значение data строковым представлением целого числа – возвращает сообщение об ошибке, если таковая имеется;

- isLess(data, value) – метод проверки является ли значение data строковым представлением целого числа и не меньшим, чем value – возвращает сообщение об ошибке, если таковая имеется;

- isGreater(data, value) – метод проверки является ли значение data строковым представлением целого числа и не большим, чем value – возвращает сообщение об ошибке, если таковая имеется;

- isEmail(data) – метод проверки является ли значение data строковым представлением email – возвращает сообщение об ошибке, если таковая имеется;

- SetRule(field\_name, validator\_name) – метод, добавляющий в массив Rules проверку для поля field\_name типа validator\_name;

- Validate(post\_array) – метод выполняющий проверку элементов в массиве post\_array, в соответствии с правилами Rules и сохраняющий сообщения об ощибках в поле Errors;

- ShowErrors() – метод, выводящий все сообщения об ошибках из поля Errors в формате HTML.

4. С использованием разработанного класса реализовать валидацию форм «Контакт» и «Тест по дисциплине».

5. Реализовать дочерний класс CustomFormValidation от класса FormValidation, дополнив его возможностью выполнения специализированной проверки формы «Тест по дисциплине» на стороне сервера.

6. Реализовать дочерний класс ResultsVerification от класса CustomFormValidation, дополнив его возможностью проверки правильности ответов, введенных пользователем на странице "Тест по дисциплине" (реализовать проверку правильности для вопросов с элементами ввода типа RadioButton, ComboBox или однострочный текст) и вывода результатов проверки пользователю.

3 ХОД РАБОТЫ

1. Для страницы «Фотоальбом» реализуем хранение данных фото в соответствующих константах модели Photo. Вывод таблицы, содержащей фото, реализуем в представлении с использованием операторов циклов.

class PhotoModel extends Model {

    public function get\_data() {

*return* [

            [ 'title' => 'Птица', 'img' => 'animal-4454043\_1280.jpg' ],

            [ 'title' => 'Город', 'img' => 'norway-4453338\_1280.jpg' ],

            [ 'title' => 'Деревья', 'img' => 'plantation-4452538\_1280.jpg' ],

            [ 'title' => 'Рыба', 'img' => 'stingray-4392776\_1280.jpg' ],

            [ 'title' => 'Улитка', 'img' => 'snail-4428838\_1280.jpg' ],

            [ 'title' => 'Попугай', 'img' => 'ara-3601194\_640.jpg' ],

            [ 'title' => 'Сова', 'img' => 'owl-4461236\_640.jpg' ],

            [ 'title' => 'Пустыня', 'img' => 'desert-1916882\_1280.jpg' ],

            [ 'title' => 'Собака', 'img' => 'pets-4415649\_1280.jpg' ],

            [ 'title' => 'Птица', 'img' => 'port-4401223\_1280.jpg' ],

            [ 'title' => 'Деревья', 'img' => 'tree-4450514\_1280.jpg' ],

            [ 'title' => 'Горы', 'img' => 'twilight-4453762\_1280.jpg' ],

            [ 'title' => 'Воздушные шары', 'img' => 'hot-air-balloons-4381674\_1280.jpg' ],

            [ 'title' => 'Шишки', 'img' => 'pine-cones-4454654\_1280.jpg' ],

            [ 'title' => 'Птица', 'img' => 'african-grey-parrot-4424746\_1280.jpg' ],

            [ 'title' => 'Озеро', 'img' => 'landscape-4449408\_1280.jpg' ],

            [ 'title' => 'Белка', 'img' => 'squirrel-4425612\_1280.jpg' ],

            [ 'title' => 'Черепаха', 'img' => 'sea-2361247\_640.jpg' ]

        ];

    }

}

Вывод в представлении:

<section *class*="content container">

    <div *class*="row">

        <?php

            $size = ceil(count($data) / 3);

            $array\_result = array\_chunk($data, $size);

*foreach*($array\_result as $chunk) {

                echo '<div class="col-md-4">';

*foreach*($chunk as $item) {

                    echo '

                        <div class="img-elem">

                            <div class="img-wrapper rounded">

                                <img class="img-fluid rounded inner-img" src="/public/img/photo/'.$item['img'].'" title='.$item['title'].' />

                            </div>

                            <h5 class="photo\_\_subtitle">'.$item['title'].'</h5>

                        </div>

                    ';

                }

                echo '</div>';

            }

        ?>

    </div>

</section>

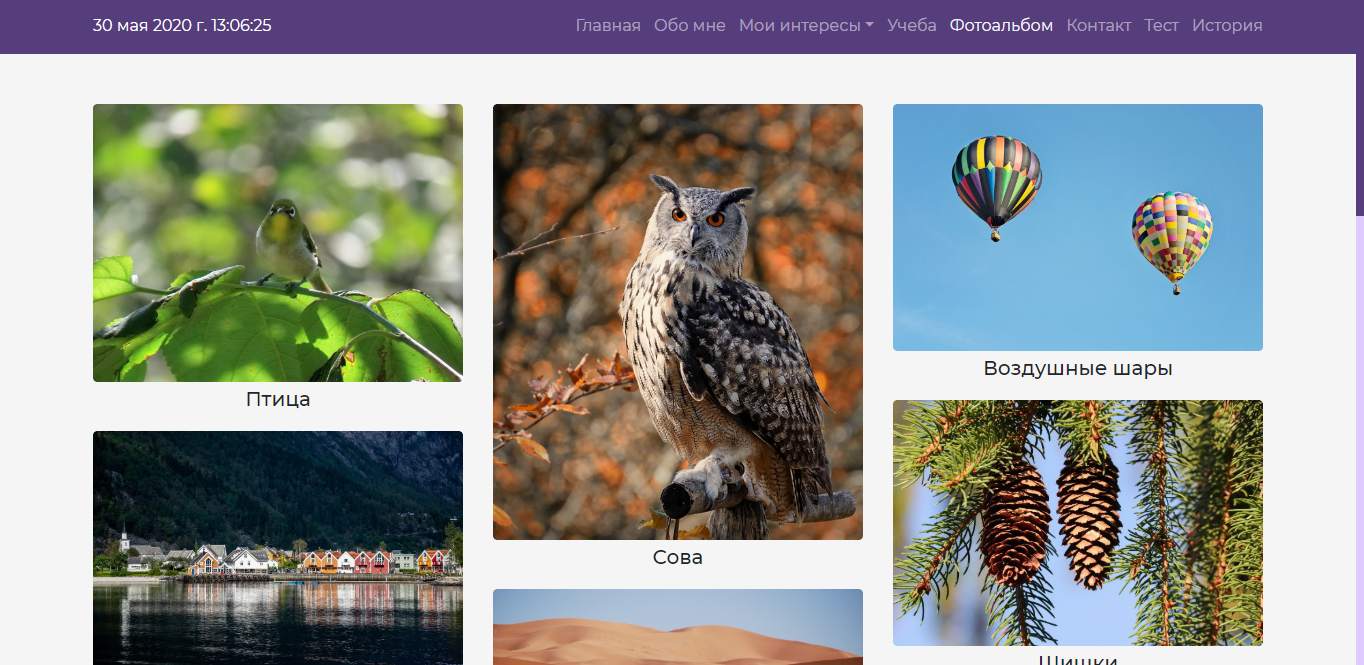


Рисунок 1 – Результат вывода фотоальбома

2. Для страницы «Мои интересы» реализуем хранение данных в соответствующих константах модели Interest. Вывод меню ссылок и интересов реализуем в представлении с использованием операторов циклов.

class HobbyModel extends Model {

    public function get\_data() {

*return* [

            [

                'id' => 'films',

                'title' => 'Фильмы',

                'picture' => 'films.jpg',

                'desc' => 'Текст...'

            ],

            [

                'id' => 'music',

                'title' => 'Музыка',

                'picture' => 'music.png',

                'desc' => 'Текст...'

            ],

            [

                'id' => 'books',

                'title' => 'Книги',

                'picture' => 'books.jpg',

                'desc' => 'Текст...'

            ],

            [

                'id' => 'sport',

                'title' => 'Спорт',

                'picture' => 'sport.jpg',

                'desc' => 'Текст...'

            ],

            [

                'id' => 'games',

                'title' => 'Игры',

                'picture' => 'games.png',

                'desc' => 'Текст...'

            ]

        ];

    }

}

Вывод в представлении:

<section *class*="container">

    <?php

        $i = 0;

*foreach*($data as $row) {

            $reverse = ++$i % 2 == 0 ? 'flex-row-reverse' : '';

            echo '

                <div class="'.$row['id'].' pt-6" id='.$row['id'].'>

                    <div class="row align-items-center '. $reverse .'">

                        <div class="col-md-4">

                            <img src="/public/img/'.$row['picture'].'" class="rounded img-fluid" alt="">

                        </div>

                        <div class="col-md-8">

                            <h1 class="about\_\_title mb-4">

                                '.$row['title'].'

                            </h1>

                            <p class="about\_\_desc">

                                '.$row['desc'].'

                            </p>

                        </div>

                    </div>

                </div>

            ';

        }

    ?>

</section>

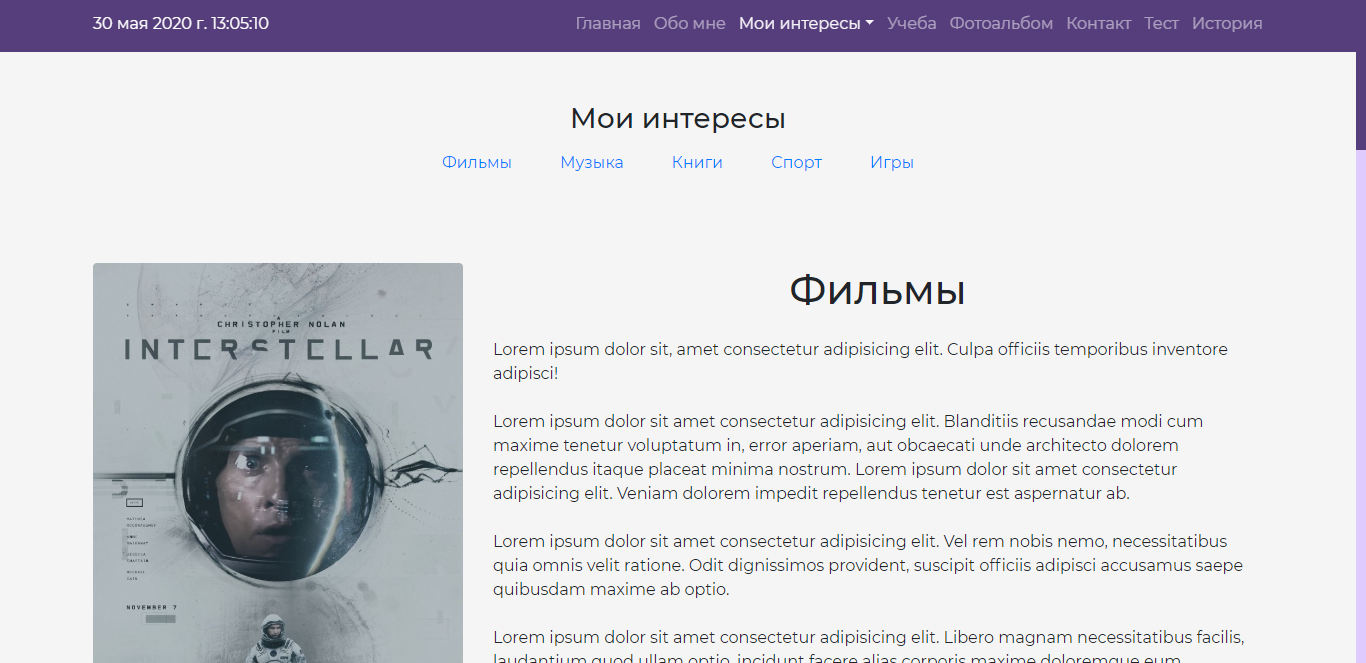


Рисунок 2 – Результат вывода

3. Реализуем класс FormValidation, выполняющий валидацию данных форм, передаваемых на сторону сервера.

class FormValidation {

    public $errors = [];

    public $rules = [

        'fullname' => [

            'isNotEmpty'

        ],

        'group' => [

            'isNotEmpty'

        ],

        'phone' => [

            'isNotEmpty',

            'isPhone'

        ],

        'date' => [

            'isNotEmpty'

        ],

        'Email' => [

            'isEmail'

        ]

    ];

    public function setRule($field\_name, $validator\_name) {

*if* (!*$this*->rules[$field\_name]) {

*$this*->rules[$field\_name] = [];

        }

        array\_push(*$this*->rules[$field\_name], $validator\_name);

    }

    public function getErrors() {

*return* *$this*->errors;

    }

    public function validate($post\_array) {

*foreach* ($post\_array as $key => $item) {

*if* (*$this*->rules[$key]) {

*foreach* (*$this*->rules[$key] as $rule) {

*$this*->$rule($item, $key);

                }

            }

        }

    }

    public function isNotEmptySelect($data, $key) {

*if* ($data == 'Выберите ответ') {

            array\_push(*$this*->errors, "Поле $key не должно быть пустым");

        }

*return* true;

    }

    public function isNotEmpty($data, $key) {

*if* (empty($data)) {

            array\_push(*$this*->errors, "Поле $key не должно быть пустым");

        }

*return* true;

    }

    public function isInteger($data, $key = null, $value = null) {

*if* (ctype\_digit($data)) {

            array\_push(*$this*->errors, "Поле $key содержит числа");

        }

*return* false;

    }

    public function isLess($data, $key, $value = null){

*if* (*$this*->isInteger($data) && ((int)*$this*->isInteger($data) >= $value)) {

            array\_push(*$this*->errors, "Поле $key слишком длинное");

        }

*return* true;

    }

    public function isGreater($data, $key, $value = null){

*if* (*$this*->isInteger($data) && ((int)*$this*->isInteger($data) <= $value)) {

            array\_push(*$this*->errors, "Поле $key слишком короткое");

        }

*return* true;

    }

    public function isEmail($data, $key, $value = null) {

*if* (preg\_match('/^((([0-9A-Za-z]{1}[-0-9A-z\.]{1,}[0-9A-Za-z]{1})|([0-9А-Яа-я]{1}[-0-9А-я\.]{1,}[0-9А-Яа-я]{1}))@([-A-Za-z]{1,}\.){1,2}[-A-Za-z]{2,})$/u', $data)) {

*return* true;

        }

        array\_push(*$this*->errors, "В поле $key неверно введена почта");

    }

    public function isPhone($data, $key, $value = null) {

*if* (preg\_match('/^(\+7|\+3)([0-9]{8,10})$/', $data)) {

*return* true;

        }

        array\_push(*$this*->errors, "В поле $key неверно введен номер телефона");

    }

}

4. С использованием разработанного класса реализуем валидацию форм «Контакт» и «Тест по дисциплине».

ContactController:

class ContactController extends Controller {

    function indexAction() {

*$this*->view->render('ContactView.php', 'Контакт');

    }

    function checkAction() {

*if* (!empty($\_POST)) {

*$this*->model->validator->validate($\_POST);

            $errors = *$this*->model->validator->getErrors();

            $vars = [ 'errors' => $errors ];

*$this*->view->render('ContactView.php', 'Контакт', $vars);

        } *else* {

*$this*->view->render('ContactView.php', 'Контакт');

        }

    }

}

TestController:

class TestController extends Controller {

    function \_\_construct() {

*$this*->model = new TestModel();

*$this*->view = new View();

    }

    function indexAction() {

*$this*->view->render('TestView.php', 'Тест');

    }

    function checkAction() {

*if* (!empty($\_POST)) {

*$this*->model->validator->validate($\_POST);

            $errors = *$this*->model->validator->getErrors();

*if* (empty($errors)) {

*$this*->model->validator->checkAns($\_POST);

                $result = *$this*->model->validator->getResult();

                $vars = [ 'result' => $result ];

            }

*else* {

                $vars = [ 'errors' => $errors ];

            }

*$this*->view->render('TestView.php', 'Тест', $vars);

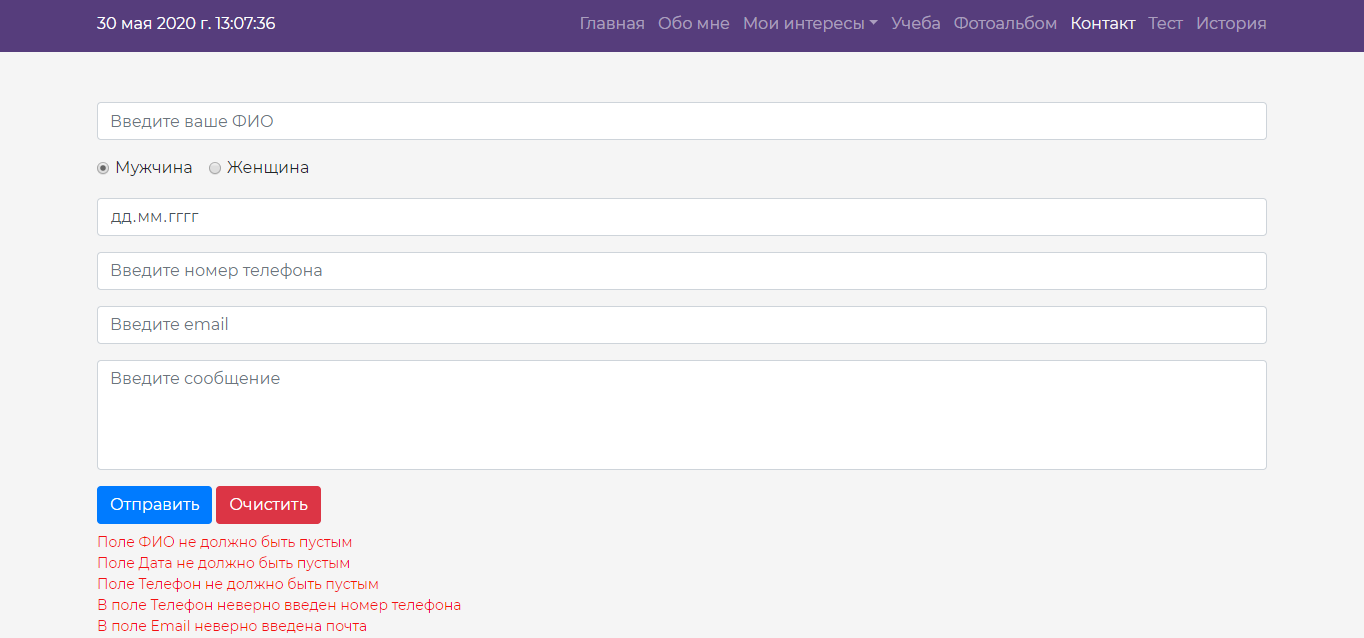
        } *else* {

*$this*->view->render('TestView.php', 'Тест');

        }

    }

}

Рисунок 3 – Результат работы валидации

5. Реализуем дочерний класс CustomFormValidation от класса FormValidation, дополнив его возможностью выполнения специализированной проверки формы «Тест по дисциплине» на стороне сервера.

class CustomFormValidation extends FormValidation {

    public function \_\_construct() {

*$this*->setRule('q2', 'isNotEmptySelect');

*$this*->setRule('q3', 'isNotEmpty');

*$this*->setRule('q3', 'isInteger');

    }}

6. Реализуем дочерний класс ResultsVerification от класса CustomFormValidation, дополнив его возможностью проверки правильности ответов, введенных пользователем на странице "Тест по дисциплине" и вывода результатов проверки пользователю.

class ResultsVerification extends CustomFormValidation {

    private $result = 0;

    private $answers = [];

    public function \_\_construct() {

        parent::\_\_construct();

*$this*->setAnswer('q2', '20');

*$this*->setAnswer('q3', 'Тест' );

    }

    public function checkAns($post\_array) {

*foreach* (*$this*->answers as $key => $value) {

*if* ($post\_array[$key] == $value) {

*$this*->result++;

            }

        }

*if* (isset($post\_array['checkbox1']) && !isset($post\_array['checkbox2']) && isset($post\_array['checkbox3'])) {

*$this*->result++;

         }

    }

    public function getResult() {

*return* *$this*->result;

    }

    public function setAnswer($field\_name, $ans) {

*$this*->answers[$field\_name] = $ans;

    }

}

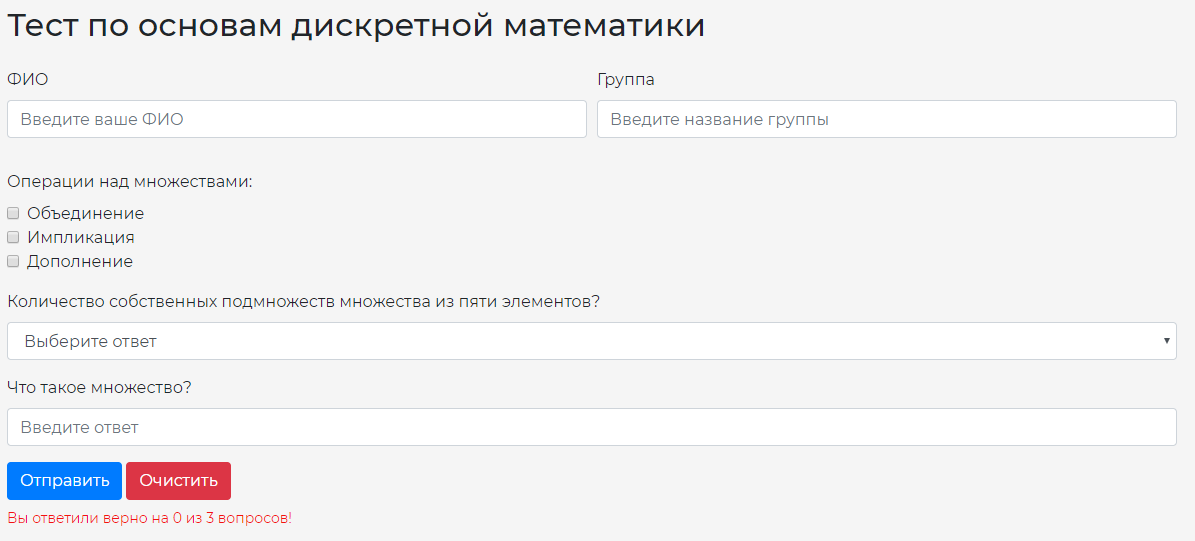


Рисунок 4 – Результат проверки на правильность ответов

ВЫВОД

В ходе выполнения лабораторной работы были изучены основы синтаксиса PHP и принципы функционирования MVC приложения на стороне сервера. Приобретены практические навыки использования языка PHP для генерации HTML-кода и обработки HTML-форм в MVC приложении.