МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федерально автономное образовательное учреждение высшего образования
«Севастопольский государственный университет»
кафедра Информационных систем

Куркчи Ариф Эрнестович

Институт информационных технологий и управления в технических системах
курс 3 группа ИС/б-31-о
09.03.02 Информационные системы и технологии (уровень бакалавриата)

ОТЧЕТ

по лабораторной работе №6

по дисциплине «Веб-технологии»

на тему «Исследование возможностей хранения данных на стороне сервера. Работа с файлами. Работа с реляционными СУБД.»

Отметка о зачете _____ _____
(дата)

Руководитель практикума

ст. преподаватель	_____	А. Л. Овчинников
(должность)	(подпись)	(инициалы, фамилия)

Севастополь 2017

1. ЦЕЛЬ РАБОТЫ

Изучить возможности хранения данных на стороне сервера: работу с файлами и СУБД MySQL из PHP, приобрести практические навыки организации хранения данных на стороне сервера в файлах, в базах данных MySQL, а также овладение навыками постраничного вывода данных.

2. ПОСТАНОВКА ЗАДАЧИ

2.1. Разработать базовый класс BaseActiveRecord для работы с базой данных, который реализует паттерн ActiveRecord (данный класс разместить в папке /my_site/app/core). Данный класс должен реализовать следующие функции:

Таблица 2.1 – Составные компоненты главного AR класса

Метод	Возвращаемое значение	Описание
save()	Текущий AR (т.е. this)	Создает или обновляет соответствующую запись в БД. Создание происходит для новых записей. Если у записи уже есть $id то происходит выполнение SQL запроса ‘UPDATE’.
delete()	true	Удаляет запись из БД к которой привязан данный ActiveRecord объект.
find($id)	AR объект	Ищет в таблице, привязанной к текущему AR строку с id = $id и возвращает объект текущего класса, у которого все поля проставлены из колонок соответствующей таблицы.
findAll()	Массив AR объектов	Возвращает список всех строк таблицы. Каждая строка – это AR объект.

2.2. Для всех таблиц, которые будут использоваться при выполнении данной лабораторной работы, создать дочерние AR классы. Для каждого из классов определить все поля и названия таблиц (данные классы необходимо разместить в папке /my_site/app/models). Например, пусть у нас будут две дочерние таблицы guest_book и blog.

2.3. Создать новую страницу "Гостевая книга". Страница должна содержать форму ввода (Фамилия, Имя, Отчество, E –mail, Текст отзыва), а также таблицу сообщений, оставленных пользователями. Сообщения в таблице должны располагаться в порядке убывания даты добавления сообщения. Для хранения сообщений пользователей использовать текстовый файл messages.inc, содержащий разделенные символом «;»

данные: Дату сообщения, ФИО, E–mail и Текст отзыва. (Для получения текущей даты сервера возможно использовать PHP функцию *date('d.m.y'))*.

2.4. Реализовать страницу "Загрузка сообщений гостевой книги", содержащую форму загрузки подготовленного заранее файла messages.inc на сервер.

2.5. Реализовать на странице "Тест по дисциплине" сохранение ответов пользователей и правильности ответов в разработанную таблицу базы данных MySQL, с возможностью просмотра сохраненных данных (дата, ФИО, ответы, верно/неверно).

2.6. Разработать страницу «Редактор Блога», позволяющую добавлять записи Блога. Страница должна содержать форму добавления записи Блога и список выдаваемых постранично записей, отсортированных в порядке убывания даты. Форма добавления должна содержать поля ввода:

- Тема сообщения – поле ввода однострочного текста (заполнение обязательно);
- Изображение – поле ввода файла (заполнение не обязательно);
- Текст сообщения (поле ввода многострочного текста);

Данные хранить в разработанной таблице базы данных MySQL. Валидацию данных осуществлять с использованием класса FormValidation, разработанного при выполнении ЛР №5.

2.7. Разработать страницу «Мой Блог», содержащую упорядоченные в порядке убывания даты добавления, выдаваемые постранично данные:

- Дата и время сообщения;
- Тема сообщения;
- Изображение;
- Текст сообщения;

Данные извлекать из таблицы базы данных MySQL.

Формат постраничного вывода определяется из Таблицы 2.2 в соответствии с вариантом задания.

Таблица 2.2

№	Вариант строки ссылок на страницы	Количество записей на странице
4	Всего страниц: 31 Предыдущая 21 22 23 24 25 Следующая	Постоянное, определяется константой

2.8. Реализовать возможность добавления записей на страницу «Мой Блог» из файла формата CSV, содержащего следующие поля: title, message, author, created_at. Например:

"тема 1","сообщение 1","Vasiliy","2014-01-01 14:00"

Для этого необходимо разработать страницу «Загрузка сообщений блога», содержащую форму загрузки файла формата CSV.

Добавление записей из файла в БД осуществлять с использованием подготавливаемых запросов (см. п. 2.4). Валидацию данных осуществлять с использованием класса FormValidation, разработанного при выполнении ЛР №5.

3. ИСХОДНЫЙ КОД

app/Core/Database

```php
<?php

namespace App\Core;

/**
* Class Database
 * @package App\Core
 * @inheritdoc \PDO
 */
class Database {

    /**
     * @var \App\Core\Database
     */
    protected static $_instance;
    protected $_pdo;

    private function __construct( $host, $db, $username, $password = '' ) {
        $dsn        = sprintf( 'mysql:host=%s;dbname=%s', $host, $db );
        $this->_pdo = new \PDO( $dsn, $username, $password );
    }

    public static function pdo(): \PDO {
        return static::instance()->_pdo;
    }

    public static function instance(): Database {
        if ( is_null( static::$_instance ) ) {
            static::$_instance = new static( 'localhost', 'test', 'test', '12345678' ); // TODO move to config
            if ( ! function_exists( 'pdo' ) ) {
                function pdo(): \PDO {
                    return Database::pdo();
                }
            }
        }

        return static::$_instance;
    }

    public static function init() {
        static::instance();
    }

    public static function __callStatic( $name, $arguments ) {
        return call_user_func_array( [ static::instance(), $name ], $arguments );
    }

    public function __call( $name, $arguments ) {
        if ( method_exists( $this->_pdo, $name ) ) {
            return call_user_func_array( [ $this->_pdo, $name ], $arguments );
        }

        return false;
    }

}
```

app/Core/ActiveRecord

```php
<?php

namespace App\Core;

class ActiveRecord {
    protected static $table = '';
    protected static $id_field = 'id';
    protected static $fields = [
        'id',
    ];

    protected $_attributes = [];
    protected $_original = [];
    protected $_inserted = false;
    protected $_lastError = false;

    public function __construct( array $attributes = [], bool $inserted = false ) {
        $this->_attributes = $attributes;
        if ( $inserted ) {
            $this->_inserted = $inserted;
            $this->_original = $this->_attributes;
        }
    }

    public static function create( array $attributes ) {
        $instance = new static( $attributes );
        $instance->save();

        return $instance;
    }

    public function save() {
        $table    = static::$table;
        $prepares = [];
        $values   = [];
        $fields   = [];

        if ( $this->_inserted ) {
            $id_field              = static::$id_field;
            $id_prepare            = ":__$id_field";
            $values[ $id_prepare ] = $this->_original[ static::$id_field ];
            foreach ( $this->_attributes as $key => $value ) {
                if ( $this->_original[ $key ] !== $value ) {
                    $prepares[]           = "`$key` = :$key";
                    $values[ ':' . $key ] = $value;
                }
            }
            if ( ! empty( $prepares ) ) {
                $keys    = implode( ', ', $prepares );
                $prepare = pdo()->prepare( "UPDATE `$table` SET $keys WHERE $id_field = $id_prepare" );
                if ( $prepare->execute( $values ) ) {
                    $this->_original = array_merge( $this->_original, $this->_attributes );

                    return true;
                } else {
                    $this->_lastError = [
                        'code' => $prepare->errorCode(),
                        'info' => $prepare->errorInfo(),
                    ];

                    return false;
                }
            }

            return true;
        } else {
            foreach ( $this->_attributes as $key => $value ) {
                $fields[]             = "`$key`";
                $prepares[]           = ":$key";
                $values[ ':' . $key ] = $value;
            }
            if ( ! empty( $prepares ) ) {
                $fields  = implode( ', ', $fields );
                $keys    = implode( ', ', $prepares );
                $prepare = pdo()->prepare( "INSERT INTO `$table`($fields) VALUES($keys)" );
                if ( $prepare->execute( $values ) ) {
                    if ( empty( $this->_attributes[ static::$id_field ] ) ) {
                        $this->_attributes[ static::$id_field ] = pdo()->lastInsertId();
                    }
                    $this->_original = array_merge( $this->_original, $this->_attributes );
                    $this->_inserted = true;

                    return true;
                } else {
                    $this->_lastError = [
                        'code' => $prepare->errorCode(),
                        'info' => $prepare->errorInfo(),
                    ];

                    return false;
                }
            }

            return false;
        }
```

```
    }

    public static function find( $id = false ) {
        $table = static::$table;
        if ( $id !== false ) {
            $id_field   = static::$id_field;
            $id_prepare = ":__$id_field";
            $prepare    = pdo()->prepare( "SELECT * FROM `$table` WHERE `$id_field` = $id_prepare" );
            $prepare->bindValue( $id_prepare, $id );
        } else {
            $prepare = pdo()->prepare( "SELECT * FROM `$table`" );
        }
        if ( $prepare->execute() ) {
            if ( $id !== false ) {
                return ( $row = $prepare->fetch( \PDO::FETCH_ASSOC ) ) !== false ? new static( $row, true ) : false;
            } else {
                $result = [];
                foreach ( $prepare->fetchAll( \PDO::FETCH_ASSOC ) as $data ) {
                    $result[] = new static( $data, true );
                }

                return $result;
            }
        } else {
            return false;
        }
    }

    public static function paginate( $offset = 0, $limit = 0 ) {
        $table   = static::$table;
        $prepare = pdo()->prepare( "SELECT * FROM `$table` ORDER BY `created_at` DESC LIMIT :offset, :limit" );
        $prepare->bindParam( ':offset', $offset, \PDO::PARAM_INT );
        $prepare->bindParam( ':limit', $limit, \PDO::PARAM_INT );
        if ( $prepare->execute() ) {
            $result = [];
            foreach ( $prepare->fetchAll( \PDO::FETCH_ASSOC ) as $data ) {
                $result[] = new static( $data, true );
            }

            return $result;
        } else {
            return false;
        }
    }

    public static function count() {
        $table    = static::$table;
        $id_field = static::$id_field;
        $prepare  = pdo()->prepare( "SELECT COUNT(`$id_field`) FROM `$table`" );
        if ( $prepare->execute() ) {
            return $prepare->fetchColumn();
        } else {
            return 0;
        }
    }

    public static function deleteAll() {
        $table = static::$table;

        return pdo()->prepare( "DELETE FROM `$table`" )->execute();

    }

    public function delete() {
        if ( $this->_inserted ) {
            $table      = static::$table;
            $id_field   = static::$id_field;
            $id_prepare = ":__$id_field";
            $prepare    = pdo()->prepare( "DELETE FROM `$table` WHERE `$id_field` = $id_prepare" );
            $prepare->bindValue( $id_prepare, $this->_original[ $id_field ] );

            if ( $prepare->execute() ) {
                $this->_inserted = false;

                return true;
            } else {
                $this->_lastError = [
                    'code' => $prepare->errorCode(),
                    'info' => $prepare->errorInfo(),
                ];

                return $this;
            }
        }

        return $this;
    }

    public function __get( $name ) {
        if ( array_key_exists( $name, $this->_attributes ) ) {
            return $this->_attributes[ $name ];
        }

        return null;
    }

    public function __set( $name, $value ) {
```

```
        if ( in_array( $name, static::$fields ) ) {
            $this->_attributes[ $name ] = $value;
        }
    }

}
```

app/Controller/BlogController

```php
<?php

namespace App\Controller;

use App\Core\Controller;
use App\Core\Form;
use App\Core\Validation;
use App\Model\Post;
use function App\Core\view;

class BlogController extends Controller {

    protected $per_page = 10;
    protected $_rules = [
        'title'   => 'required',
        'message' => 'required',
    ];
    protected $_messages = [
        'title'   => [
            'required' => 'Поле обязательное',
            'words'    => 'Введите заголовок',
        ],
        'message' => [
            'required' => 'Поле обязательное',
            'email'    => 'Введите содержимое поста',
        ],
    ];

    function index() {
        $this->page( 1 );
    }

    function page( $n ) {
        $total = (int) ceil( Post::count() * 1.0 / $this->per_page );
        if ( $n <= 0 || $n > $total ) {
            $n = 1;
        }
        $posts    = Post::paginate( $this->per_page * ( $n - 1 ), $this->per_page ) ?: [];
        $count    = count( $posts );
        $paginate = [
            'per_page'   => $this->per_page,
            'on_page'    => $count,
            'is_empty'   => count( $posts ) === 0,
            'current'    => $n,
            'total'      => $total,
            'is_last'    => $n >= $total,
            'is_first'   => $n == 1,
            'left_tail'  => max( 1, $n - 2 ),
            'right_tail' => min( $total, $n + 2 ),
            'prev'       => max( 1, $n - 1 ),
            'next'       => min( $total, $n + 1 ),
        ];

        view( 'blog.index', compact( 'posts', 'paginate' ) );
    }

    function create() {
        $form = new Form( $_REQUEST, Validation::run( $this->_rules, $this->_messages ) );
        if ( $form->success() ) {
            $image = '';
            if ( ! empty( $_FILES['image'] ) ) {
                $file  = $_FILES['image'];
                $image = sprintf(
                    '/uploads/%s.%s',
                    hash( 'sha256', $file['name'] . time() ),
                    pathinfo( $file['name'], PATHINFO_EXTENSION )
                );
                $img   = new \Imagick( realpath( $file['tmp_name'] ) );
                $img->scaleImage( 300, 300, true );
                $img->writeImage( __APP_ROOT__ . 'public' . $image );
                $img->destroy();
            }
            Post::create( [
                'title'   => $form->val( 'title' ),
                'message' => $form->val( 'message' ),
                'image'   => $image,
            ] );
            header( 'Location: /blog' );
            exit;
        }

        view( 'blog.create', compact( 'form' ) );
    }

    function import() {
        $result = false;
        if ( count( $_REQUEST ) && ! empty( $_FILES['file'] ) ) {
```

```
            $rewrite = ! empty( $_REQUEST['rewrite'] );
            $file    = $_FILES['file'];
            if ( substr_compare( $file['name'], '.csv', - 4, 4, true ) !== 0 ) {
                $result = [
                    'type'    => 'danger',
                    'message' => 'Расширение файла должно быть .csv',
                ];
            } else {
                $data   = file_get_contents( $file['tmp_name'] );
                $count  = $this->import_posts( $data, $rewrite );
                $result = [
                    'type'    => 'success',
                    'message' => sprintf( 'Успешно загружено %d записей', $count ),
                ];
            }
        }

        view( 'blog.import', compact( 'result' ) );
    }

    protected function import_posts( $data, $rewrite ) {
        $imported = 0;
        if ( $rewrite ) {
            Post::deleteAll();
        }
        foreach ( explode( PHP_EOL, $data ) as $tmp ) {
            if ( empty( trim( $tmp ) ) ) {
                continue;
            }
            $tmp = explode( '","', substr( trim( $tmp ), 1, - 1 ) );
            if ( count( $tmp ) !== 5 ) {
                continue;
            }
            $created_at = date_create_from_format( 'h:i:s d.m.Y', $tmp[3] );
            $updated_at = date_create_from_format( 'h:i:s d.m.Y', $tmp[4] );
            if ( $created_at === false || $updated_at === false ) {
                continue;
            }
            Post::create( [
                'title'      => trim( $tmp[0] ),
                'message'    => trim( $tmp[1] ),
                'image'      => trim( $tmp[2] ),
                'created_at' => $created_at,
                'updated_au' => $updated_at,
            ] );
            $imported ++;
        }

        return $imported;
    }

}
```

app/Controller/GuestController

```
<?php

namespace App\Controller;

use App\Core\Controller;
use App\Core\Form;
use App\Core\Validation;
use function App\Core\view;

class GuestController extends Controller
{
    public $_ajaxOnly = ['validate'];
    protected $_rules = [
        'name'  => 'required,words:2-3',
        'email' => 'required,email',
        'text'  => 'required',
    ];
    protected $_messages = [
        'name'  => [
            'required' => 'Поле обязательное',
            'words'    => 'Введите в формате: Фамилия Имя Отчество',
        ],
        'email' => [
            'required' => 'Поле обязательное',
            'email'    => 'Введите действительную почту',
        ],
        'text'  => 'Поле обязательное',
    ];

    protected $storage = 'storage/messages.inc';

    function index()
    {
        $form = new Form($_REQUEST, Validation::run($this->_rules, $this->_messages));
        if ($form->success()) {
            $matches = mb_split('\s+', $form->val('name'));
            $message = [
                'first_name'  => $matches[1],
                'last_name'   => $matches[0],
                'middle_name' => empty($matches[2]) ? '' : $matches[2],
```

```
41                  'email'       => $form->val('email'),
42                  'text'        => $form->val('text'),
43              ];
44              $this->add_message($message);
45              $form->clear();
46          }
47          $messages = $this->get_messages();
48
49          view('guest.index', compact('messages', 'form'));
50      }
51
52      private function add_message($message)
53      {
54          $message['date'] = date('h:i:s d.m.Y', time());
55          $file = fopen($this->storage, 'a');
56          fputs($file, implode(';', $message) . PHP_EOL);
57          fclose($file);
58      }
59
60      private function get_messages()
61      {
62          $messages = [];
63          if (file_exists($this->storage)) {
64              $file = fopen($this->storage, 'r');
65
66              while ($tmp = fgets($file)) {
67                  $tmp = explode(';', trim($tmp));
68                  $messages[] = [
69                      'first_name'  => $tmp[0],
70                      'last_name'   => $tmp[1],
71                      'middle_name' => $tmp[2],
72                      'email'       => $tmp[3],
73                      'text'        => $tmp[4],
74                      'date'        => date_create_from_format('h:i:s d.m.Y', $tmp[5]),
75                  ];
76              }
77              fclose($file);
78          }
79
80          return $messages;
81      }
82
83      function import()
84      {
85          $result = false;
86          if (count($_REQUEST) && !empty($_FILES['file'])) {
87              $rewrite = !empty($_REQUEST['rewrite']);
88              $file = $_FILES['file'];
89              if (substr_compare($file['name'], '.inc', -4, 4, true) !== 0) {
90                  $result = [
91                      'type'    => 'danger',
92                      'message' => 'Расширение файла должно быть .inc',
93                  ];
94              } else {
95                  $data = file_get_contents($file['tmp_name']);
96                  $count = $this->import_messages($data, $rewrite);
97                  $result = [
98                      'type'    => 'success',
99                      'message' => sprintf('Успешно загружено %d записей', $count),
100                 ];
101             }
102         }
103
104         view('guest.import', compact('result'));
105     }
106
107     private function import_messages($data, $rewrite = false)
108     {
109         $messages = $rewrite ? [] : $this->get_messages();
110         $imported = 0;
111         foreach (explode(PHP_EOL, $data) as $tmp) {
112             if (empty(trim($tmp))) {
113                 continue;
114             }
115             $tmp = explode(';', trim($tmp));
116             if (count($tmp) !== 6) {
117                 continue;
118             }
119             $date = date_create_from_format('h:i:s d.m.Y', $tmp[5]);
120             if($date === false) {
121                 continue;
122             }
123             $messages[] = [
124                 'first_name'  => $tmp[0],
125                 'last_name'   => $tmp[1],
126                 'middle_name' => $tmp[2],
127                 'email'       => $tmp[3],
128                 'text'        => $tmp[4],
129                 'date'        => date_create_from_format('h:i:s d.m.Y', $tmp[5]),
130             ];
131             $imported++;
132         }
133         usort($messages, function ($a, $b) {
134             return $a['date']->getTimestamp() -
135                 $b['date']->getTimestamp();
136         });
137         foreach($messages as &$message) {
```

```
138             $message['date'] = $message['date']->format('h:i:s d.m.Y');
139         }
140 
141         $file = fopen($this->storage, 'w');
142         foreach ($messages as $message) {
143             fputs($file, implode(';', $message) . PHP_EOL);
144         }
145         fclose($file);
146 
147         return $imported;
148     }
149 
150     function validate()
151     {
152         echo json_encode(Validation::run($this->_rules, $this->_messages));
153     }
154 
155 }
```

Вывод

В ходе выполнения лабораторной работы были изучены возможности хранения данных на стороне сервера, работа с файлами и СУБД MySQL из PHP, приобретены практические навыки организации хранения данных на стороне сервера в файла, в базах данных MySQL, а также был освоен навык постраничного вывода данных.