Министерство образования Республики Беларусь
Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет Компьютерных технологий

Кафедра проектирования информационных компьютерных систем

Дисциплина "Современные технологии проектирования информационных систем"

К защите допустить:
Руководитель курсовой работы
старший преподаватель
кафедры
_____ А.В.Михалькевич
10.07.2025

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

на тему

Разработка сайта для фотографа "Фотомастер"

БГУИР КР 1-40 05 01-10 № 166 ПЗ

Студент (подпись студента)

Курсовая работа
представлена на проверку
10.07.2025
_____
(подпись студента)

2025

Реферат

БГУИР КР 1-40 05 01-10 № 166 ПЗ, гр. 784371

, Разработка сайта для фотографа "Фотомастер", Минск: БГУИР - 2025.

Пояснительная записка 79471 с., 21 рис., 4 табл.

Ключевые слова:

Предмет Современные технологии проектирования информационных систем, А.В.Михалькевич

-

-

Содержание

Введение

Курсовой проект посвящен изучению теории и практики разработки и проектирования распределённых баз данных. В данном курсовом проекте объектом исследования является среда WEB, как платформа приложений баз данных в информационных системах. Предметом исследования является система организации и ведения распределённых баз данных в Интернет. Целью курсового проекта является разработка интернет-каталога натяжных потолков. Курсовой проект предполагает выполнение следующих задач: – спроектировать базу данных, описать предметную область, создать диаграмму классов и диаграмму состояний; – разработать с использованием WEB-интерфейса, алгоритма работы сайта-портфолио фотографа. В качестве практической части в рамках курсового проекта создаётся сайт-портфолио фотографа с использованием технологий языка программирования PHP и СУБД MySQL.

1 ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ

Архитектура сайта – систематизация информации и навигации по ней с целью помочь посетителям более успешно находить нужные им данные. Хорошо продуманная грамотная архитектура сайта гарантирует, что пользователи потратят меньше времени на поиск нужной информации.

Архитектура сайта ведётся с учётом наиболее важной информации с точки зрения продвижения товаров и услуг на интернет-рынке. Грамотное распределение приоритетов между разделами и страницами сайта, делает их основными точками входа на сайт, что позволяет потенциальному потребителю быстро найти необходимую ему информацию об искомых товарах/услугах и повышает успешность бизнеса в интернете.

Архитектура сайта проста и интуитивно удобна, состоит из клиентской части, программной части и администрирования.

Программная часть архитектуры сайта рассматривается как взаимосвязь операционной части и серверной части.

В операционной части рассматривается среда разработки сайта.

Серверная часть содержит в себе размещение сайта на сайте провайдера, поддерживающие технологии, используемые при создании сайта.

Сайт разрабатывается в среде *PHP*, либо – *Perl, ASP.NET, ColdFusion* и *Java*.

В клиентской части архитектуры разрабатывается максимально удобная и доступная работа потенциального клиента на страницах сайта. Разработка интерфейса, доступные и понятные диалоговые окна. Немаловажным фактором является обратная связь, позволяющая высказать клиенту свое мнение о работе фотографа, о качестве обслуживания и сайта в целом.

Проанализировав работу уже имеющихся сайтов, делается вывод о том, что обязательно будет реализовано в курсовом проекте.

Структура сайта оформляется так, чтобы пользователь имел возможность получить о сайте исчерпывающую информацию (описание в виде текста плюс несколько фотографий).

Для создания сайта отдаётся предпочтение язык программирования *PHP*. Это мощная среда для разработки, совместимая со всеми операционными системами и браузерами, не требующая высоких аппаратных средств компьютера, довольно проста в освоении и продолжает развиваться и совершенствоваться. Также он поддерживается подавляющим большинством платных хостингов, что является несомненным плюсом.

Выбор платного хостинга заключается в том, что есть хоть какие-то гарантии, сайт получает имя на доменном уровне, поддерживаются все современные технологии, не будет назойливых рекламных баннеров, не относящихся к тематике сайта, скорость закачки будет заметно выше, обслуживание таких сайтов удобнее, есть возможности для развития, введения новых услуг для привлечения клиентов. Также можно заключить долгосрочный договор, что будет гарантировать бесперебойную работу сайта, его защиту от взлома и вирусов, позволит избежать неприятных сюрпризов вроде прекращения существования данного хостинга.

Проведём проектирование базы данных сайта-портфолио фотографа.

Проектирование базы данных состоит главным образом в определении элементов данных, которые нужно включить в базу данных, отношения между ними и ограничений на значения данных. Внешнее представление содержит только те сущности, атрибуты и связи предметной области, которые интересны пользователю.

Помимо этого, различные представления могут по-разному отображать одни и те же данные. Разработаем базу данных для автоматизации работы сайта.

Для рационального управления сайта необходимо контролировать различную поступающую

информацию, которую необходимо структурировать и хранить в различных базах данных.

Имеющиеся базы данных должны быть взаимосвязаны между собой. Для правильного создания баз данных с такой информацией необходимо определить сущности сайта.

В соответствии с заданием в курсовом проекте необходимо спроектировать базу данных сайта по хранению информации об услугах.

2 ОБОСНОВАНИЕ ВЫБОРА ТЕХНОЛОГИЙ

Наиболее распространенным языком разработки сайта является Язык разметки гипертекстовых страниц (*HTML – Hypertext Markup Language*) представляет собой язык, разработанный специально для создания *Web*-документов. Он определяет синтаксис и размещение специальных инструкций (тегов), которые не выводятся на экран, но указывают браузеру, как отображать содержимое документа. Он также используется для создания ссылок на другие документы, локальные или сетевые, например, находящиеся в сети Интернет.

Стандарт *HTML* и другие стандарты для *Web* разработаны под руководством консорциума *W3C* (*World Wide Web Consortium).* Стандарты, спецификации и проекты новых предложений можно найти на сайте *http://www.3w.org/.* В настоящее время действует спецификация *HTML5.0,* поддержка которой со стороны основных браузеров постоянно растет.

На практике на стандарт *HTML* большое влияние оказывает наличие тегов, предложенных и поддерживаемых наиболее известными браузерами, такими как *Microsoft Internet Explorer* и *Netscape Navigator.* Эти теги в данный момент могут, как входить, так и не входить в состав действующей спецификации *HTML*.

PHP – язык программирования, используемый на стороне *Web*-сервера для динамической генерации *HTML*-страниц. Об этом говорит и расшифровка его названия: *PHP – Personal HyperText Processor*.

PHP – один из немногих языков программирования, созданных специально для разработки *Web*-приложений. Поэтому он включает в себя все функции, необходимые именно для работы на *Web*-сервере, и при этом лишен избыточности, свойственной многим его конкурентам.

Очень приятная особенность *PHP* – то, что его команды включаются в обычные *HTML*-страницы с помощью специальных тегов, которые и заставляют *PHP*-машину выполнять на сервере нужные действия. Программам на *PHP* не нужны специальные *CGI*-директории с особыми правами доступа. Более того, на одной страничке можно произвольно чередовать «простой» *HTML* и *PHP*-код.

PHP не зависит от платформы. *PHP* прекрасно интегрируется во все популярные *Web*-серверы: *Apacce* и *IIS*, *Zens* и *Netscape Enterprise Server*, работает под *Windows* и *OS/2*, *MacOS* и практически всеми *UNIX*-подобными системами. Как следствие – *PHP* работает практически у всех хостеров, разрешающих собственные выполняемые скрипты.

Замечательная особенность *PHP* – его интегрированность практически со всеми современными интернет-технологиями. *PHP* поддерживает большинство современных *Web*-протоколов: *IMAP, FTP, POP, XML, SNMP* и другие. *PHP* прекрасно работает с базами данных. Трудно найти СУБД, поддержка которой не была бы реализована в *PHP*. *MySQL* и *MS SQL*

Server, *PostgreSQL* и *Oracle*, *Sybase* и *Interbase*. Один только перечень баз данных, поддерживаемых *PHP*, займет, наверное, целый экран.

PHP включает в себя огромное количество встроенных функций: обработки строк и массивов, работы с файловой системой и с *HTTP*, электронной почтой, датой и временем, кириллицей и другими национальными алфавитами, и большим количеством встроенных функций. Благодаря им многие алгоритмы, требующие в большинстве языков написания программного кода размером в несколько экранов, реализуются на *PHP* одной командой (точнее, вызовом одной функции).

Современные тенденции развития языков программирования не обошли стороной и *PHP*. Средства объектно-ориентированного программирования появились еще в *PHP3*. А в объектной модели *PHP4* в полном объеме реализованы классические понятия объектно-ориентированного программирования: наследование, инкапсуляция и полиморфизм.

Основное отличие от *CGI*-скриптов, написанных на других языках, типа *Perl* или *C* – это то, что в *CGI*-программах вы сами пишете выводимый *HTML*-код, а, используя *PHP* – вы встраиваете свою программу в готовую *HTML*-страницу, используя открывающий и закрывающий теги.

Отличие *PHP* от *JavaScript*, состоит в том, что *PHP*-скрипт выполняется на сервере, а клиенту передается результат работы, тогда как в *JavaScript*-код полностью передается на клиентскую машину и только там выполняется.

3 ИНСТРУМЕНТАРИЙ

Для реализации проекта нами были выбраны следующие инструменты работы:

Adobe Photoshop — многофункциональный графический редактор, разработанный и распространяемый фирмой *Adobe Systems*. В основном работает с растровыми изображениями, однако имеет некоторые векторные инструменты. Продукт является лидером рынка в области коммерческих средств редактирования растровых изображений и наиболее известным продуктом фирмы *Adobe*.

Visual Studio Code - многофункциональный текстовый редактор с широким набором удобных инструментов для выделения, маркировки и обработки текстовых фрагментов кода.

Интерфейс этого редактора очень лаконичен. Зато скорость работы и отклика на все Ваши действия на достаточно высоком уровне. Поддерживает огромное количество языков (*C++, Dylan, Erlang, HTML, Haskell, Java, JavaScript, Lua, Markdown, MATLAB, Perl, PHP, Python, Ruby, SQL, XML и др.)* и предлагает на выбор около 20 цветовых схем. Весьма удобно, что реализован полноэкранный режим - очень полезно, если не хотите, чтобы Вас что-то отвлекало от полноценной работы.

Основными технологиями проекта являются *PHP, MySQL*. Программная среда проекта- *Open Server*.

Open Server включает в себя следующий набор инструментов: *HeidiSQL, Adminer, PHPMyAdmin, PHPPgAdmin, PgAdmin, Perl, FTP* сервер, *Sendmail, Memcached* сервер.

Система контроля версий

Git- мощная и сложная распределенная система контроля версий.

Система контроля версий (СКВ) — это система, регистрирующая изменения в одном или нескольких файлах с тем, чтобы в дальнейшем была возможность вернуться к определённым старым версиям этих файлов.

СКВ даёт возможность возвращать отдельные файлы к прежнему виду, возвращать к прежнему состоянию весь проект, просматривать происходящие со временем изменения, определять, кто последним вносил изменения во внезапно переставший работать модуль, кто и когда внёс в код какую-то ошибку, и многое другое. Вообще, если, пользуясь СКВ, вы всё испортите или потеряете файлы, всё можно будет легко восстановить.

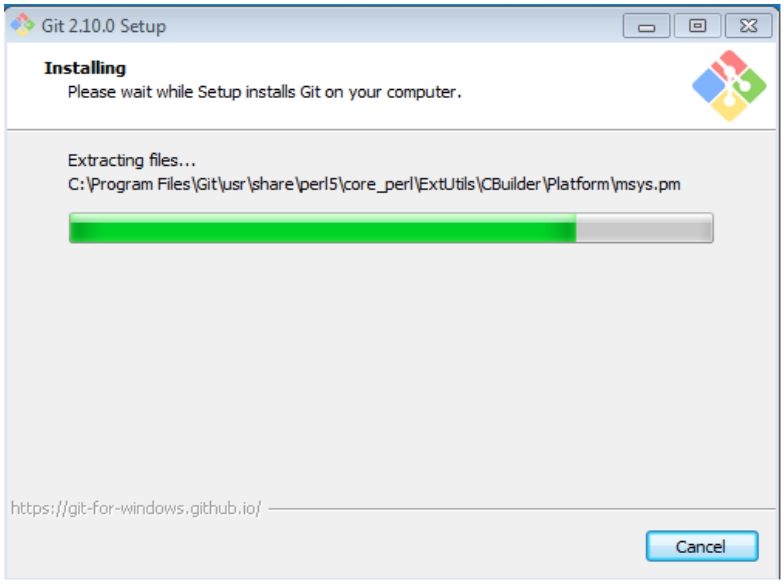

Рисунок 3.1 – Установка *Git*

TortoiseGit — визуальный клиент системы управления исходными кодами программ *git* для ОС *Microsoft Windows*. Распространяется по лицензии *GNU GPL*.

Реализован как расширение проводника *Windows* (*shell extension*). Подрисовывает иконки к файлам, находящимся под управлением *Git*, для отображения их статуса в *Git*.

Взаимодействие с системой контроля версий основано на *mSysGit*, *TortoiseGit* использует его внутри себя, и требует установки последнего на машину.

Предназначен для удобства работы: по существу, все операции с репозиторием *Git* можно выполнять из графического интерфейса *TortoiseGit*, без использования консольных команд.

Зарегистрировались на GitHub.com.

Прописали следующие команды в консоли:

1. Git init – создание репозитория;

2. Git add * - добавляет содержание рабочей директории в индекс (staging area) для последующего изменения;

3. Git commit –m “сообщение”- изменение последнего изменения;

4.Git remote add project https://github.com/darkTeiman/Fotograf-master

добавление удаленного репозитория;

5. Git push project master - вносим изменения в удаленный репозиторий.

Ссылка на удаленный репозиторий:

https://github.com/darkTeiman/Fotograf-master

В разделе 3 описаны возможности и установка Git и TortoiseGit.

Git- мощная и сложная распределенная система контроля версий.

Система контроля версий (СКВ) — это система, регистрирующая изменения в одном или нескольких файлах с тем, чтобы в дальнейшем была возможность вернуться к определённым старым версиям этих файлов.

Также приведена ссылка на удаленный репозиторий, по которой можно найти данный проект.

4 АРХИТЕКТУРНЫЙ ШАБЛОН ПРОЕКТИРОВАНИЯ HMVC

Основной паттерн для создания фреймворков и других web-приложений. Фреймворки в *PHP* зачастую используют для больших проектов. Основное преимущество - это, конечно же, предоставление возможности строить проект при помощи паттерна *MVC* (*Model-View-Controller*).

Плюсы:

• вложенность шаблонов

• независимость представления от контроллера

• целостность шаблона

• возможность кэширования

• видимость переменных

• лаконичность кода

Расшифруем само понятие *MVC*:

Model - модели данных, которые многие и без того используют без фреймфорков. Фактически обычные классы для работы с разными данными.

View - представления. Это шаблонизатор, например, *SMARTY* либо собственный. Представления - это вид, в котором отображаются данные.

Controller – основной вызываемый класс, содержащий базовую логику приложения.

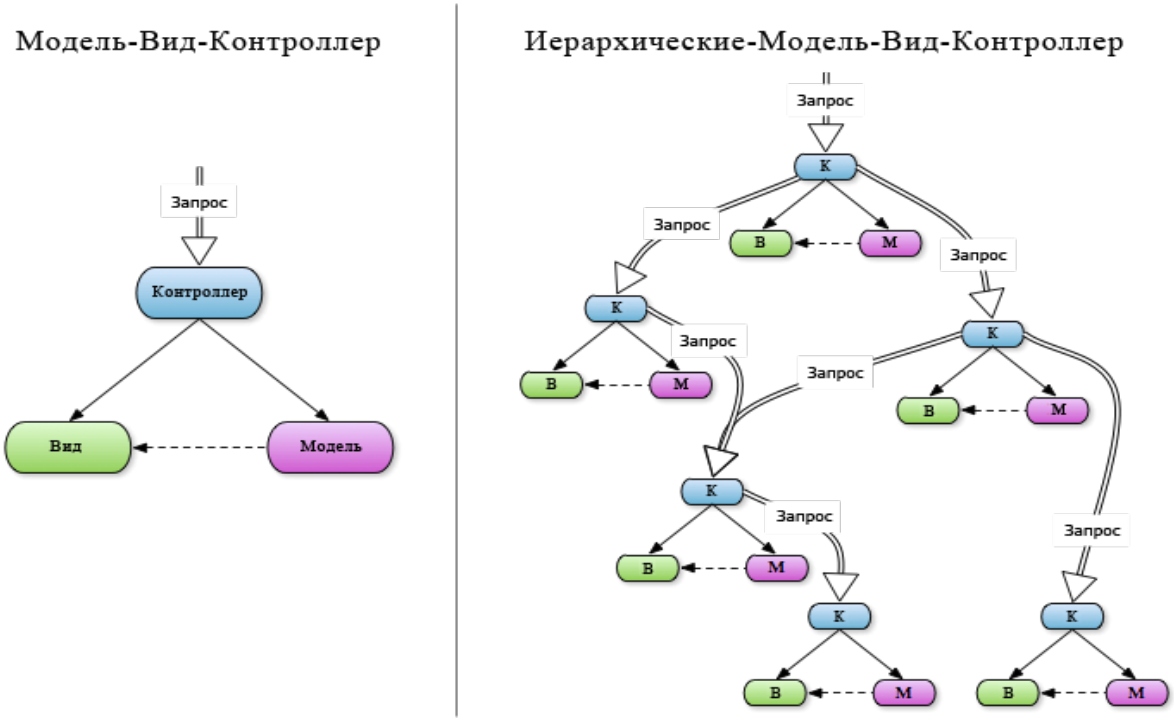

Рисунок 4.1 – Модели

Для понимания модели *HMVC* необходимо также иметь представление о роутинге (или маршрутизации) запросов. Главное отличие от *MVC*-паттерна - возможность передачи запроса по контроллерам.

По такому принципу построены почти все современные web-фреймворки (за исключением клиентских, таких как Angular, Backbone и др.)

В *HMVC* фактически процесс не меняется, т.к. последовательность действий в случае использования фреймворка остается той же, что и без него (принимаем данные - обрабатываем их в модели - выводим результат через представление).

HMVC позволяет легко соби рать воедино и легко управлять большими частями кода. А фреймворк вносит существенную долю автоматизации и простоты управления. Фреймворк - это склад различных классов и библиотек, которые позволяют отказаться от изобретения велосипедов и начать использовать готовые решения, тем самым увеличив скорость разработки. Любой разработчик, если он занимается профессиональной разработкой, со временем приходит к созданию собственной библиотеке классов, основанной, как правило, на уже готовых классах.

Современные фрэймворки не только предлагают для использования готовые классы, но и свою структуру папок.

У каждого из фрэймворков есть свои преимущества и свои недостатки.

3.1 HMVC

По концепции *MVC*, когда мы делаем запрос, мы, сперва, попадаем в контроллер (Controller). Затем в контроллере может происходить вызов модели (Model) (т.е. получение данных из модели),а затем передача этих данных в шаблон представления (View). Все очень просто, но это не всегда бывает удобно, хотя бы потому, что часто приходится вносить

изменения в контроллеры либо дублировать контроллеры из-за того, что в них вносятся незначительные
изменения.

В связи с этим придумали концепцию *HMVC*, т.е. иерархическая *MVC*. По данной концепции мы также сперва делаем запрос к контроллеру, который в свою очередь может передать запрос к другому контроллеру. Взаимосвязь самого контроллера с моделью и шаблоном представления осталась той же. Концепцию *HMVC* помогают понять следующие технологии:

Наследование классов,
Использование переменных-шаблонов.

Маршрутизация

Запрос из адресной строки попадает в так называемый обработчик маршрутов, или маршрутизатор, или роутер (*routes*). Маршрутизатор определяет, какой контроллер необходимо вызывать.

Маршруты описываем в routes.*php*, указывая какой контролер и какой экшин будет отвечать за формирование той или иной страницы.

Route.*php*

```
namespace application\core;
use application\core\View;
class Router {
    protected $routes = [];
    protected $params = [];
       public function __construct() {
        $arr = require 'application/config/routes.php';
        foreach ($arr as $key => $val) {
            $this->add($key, $val);
        }
    }
    public function add($route, $params) {
        $route = preg_replace('/{([a-z]+):([^\}]+)}/', '(?P<\1>\2)', $route);
        $route = '#^'.$route.'$#';
        $this->routes[$route] = $params;
    }
    public function match() {
        $url = trim($_SERVER['REQUEST_URI'], '/');
        foreach ($this->routes as $route => $params) {
            if (preg_match($route, $url, $matches)) {
                foreach ($matches as $key => $match) {
                    if (is_string($key)) {
                        if (is_numeric($match)) {
                            $match = (int) $match;
                        }
                        $params[$key] = $match;
                    }
                }
                $this->params = $params;
                return true;
            }
        }
        return false;
    }
public function run(){
        if ($this->match()) {
            $path='application\controllers\\'.ucfirst($this-params['controller']).'Controller';
            if (class_exists($path)) {
                $action = $this->params['action'].'Action';
                if (method_exists($path, $action)) {
                    $controller = new $path($this->params);
                    $controller->$action();
                } else {
                   View::errorCode(404);
                }
            } else {
                View::errorCode(404);
            }
        } else {
            View::errorCode(404);
        }
    }

}
```

Контроллеры

Контроллеры хранятся в папке core/controller/. Все контроллеры должны наследовать класс Controller. Этот класс также может хранится в папке core/controller, и в него можно поместить общую логику для других контроллеров. MainController расширяет базовый класс Controller.

Есть несколько способов определения маршрута для контроллера.

В файле config/routes.*php*.

А вот так будет выглядеть сам контроллер:

```
namespace application\controllers;
use application\core\Controller;
class MainController extends Controller {
echo ‘OK’;
        }
```

Применение контроллеров в данной курсовой работе, контроллер для главной страницы

MainController.*php*

```
namespace application\controllers;
use application\core\Controller;
use application\models\Admin;
class MainController extends Controller {
	public function indexAction() {

		$this->view->render('Главная страница');
	}
	public function aboutAction() {
		$this->view->render('Обо мне');
	}
public function lovestoryAction() {
		$this->view->renderfoto('Любовная история ');
	}
	public function peddingAction() {
		$this->view->renderfoto('В ожидании');
	}
	public function familyAction() {
		$this->view->renderfoto('Семнйное');
	}
	public function eventsAction() {
		$this->view->renderfoto('Мероприятие');
	}
	public function artdressAction() {
		$this->view->renderfoto('АРТ');
	}
	public function carAction() {
		$this->view->renderfoto('Автомероприятие');
	}
	public function weddingAction() {
		$this->view->renderfoto('Свадьба');
	}
	public function portretAction() {
		$this->view->renderfoto('Портрет');
	}
	public function contactAction() {
		if (!empty($_POST)) {
			if (!$this->model->contactValidate($_POST)) {
				$this->view->message('error', $this->model->error);
			}
			mail('titef@p33.org', 'Сообщение из блога',
$_POST['name'].'|'.$_POST['phone'].'|'.$_POST['text']);
			$this->view->message('success', 'Сообщение отправлено Администратору');
		}
		$this->view->render('Контакты');
	}
```

Модели

Модель должна содержать всю бизнес-логику вашего приложения. Или, другими словами, то, как приложение взаимодействует с базой данных.

Бизнес-логика — это:

Объекты реального мира, которые используются в вашем приложении;

То, как эти объекты взаимодействуют друг с другом;

Набор правил для доступа к этим объектам и для их обновления.

Поэтому, например, *Users* станет важным объектом в *Cribbb*, потому что это социальное приложение.

Мы должны хранить информацию о наших пользователях, и поэтому нам нужна модель *User* и таблица *User* в базе данных.

Пользователям надо будут вводить имя, адрес электронной почты и пароль, а также другие детали профиля. Чтобы удостовериться, что они вводят правильно отформатированные данные, мы должны проверять их ввод.

Пользователи смогут создавать сообщения. Пользователь может иметь много сообщений, и каждое сообщение должно принадлежать пользователю.

Это основные особенности работы моделей в приложениях *MVC*. По существу для каждой важной вещи в приложении, вероятно, потребуется модель. Вам, возможно, понадобится проверять данные, используемые в вашей модели, а так же здесь должна быть вся логика, отвечающая за взаимодействие моделей друг с другом.

Main.php

```
namespace application\models;
use application\core\Model;
class Main extends Model {
	public $error;
	public function contactValidate($post) {
		$nameLen = iconv_strlen($post['name']);
		$textLen = iconv_strlen($post['text']);
		if ($nameLen < 3 or $nameLen > 20) {
			$this->error = 'Имя должно содержать от 3 до 20 символов';
			return false;
		} elseif (!filter_var($post['email'], FILTER_VALIDATE_EMAIL)) {
			$this->error = 'E-mail указан неверно';
			return false;
		} elseif ($textLen < 10 or $textLen > 500) {
			$this->error = 'Сообщение должно содержать от 10 до 500 символов';
			return false;
		}
		return true;
	}
```

Подключение и настройка базы данных осуществляется в файле application/config/db.*php*. Для подключения нужной базы данных прописали настройки сервера баз данных в массив :

```
return [
	'host' => 'localhost',
	'name' => 'blog',
	'user' => 'root',
	'password' => '',
];
```

View

View.*php*

```
namespace application\core;
class View {
	public $path;
	public $route;
	public $layout = 'default';
		public function __construct($route) {
		$this->route = $route;
		$this->path = $route['controller'].'/'.$route['action'];
	}
	public function render($title, $vars = []) {
		extract($vars);
		$path = 'application/views/'.$this->path.'.php';
		if (file_exists($path)) {
			ob_start();
			require $path;
			$content = ob_get_clean();
			require 'application/views/layouts/'.$this->layout.'.php';
		}
	}
	public function renderfoto($title, $vars = []) {
		extract($vars);
		$path = 'application/views/'.$this->path.'.php';
		if (file_exists($path)) {
			ob_start();
			require $path;
			$content = ob_get_clean();
			require 'application/views/layouts/partfolios.php';
		}
		}
	public function redirect($url) {
		header('location: /'.$url);
		exit;
	}
	public static function errorCode($code) {
		http_response_code($code);
		$path = 'application/views/errors/'.$code.'.php';
		if (file_exists($path)) {
			require $path;
		}
		exit;
	}
	public function message($status, $message) {
		exit(json_encode(['status' => $status, 'message' => $message]));
	}

	public function location($url) {
		exit(json_encode(['url' => $url]));
	}
}
```

5 ПРИМЕНЕНИЕ ШАБЛОНА ПРОЕКТИРОВАНИЕ ПРАКТИЧЕСКОГО РЕШЕНИЯ

Шаблон проектирования (*design patterns*) – это многократно используемые решения распространённых проблем, возникающих при разработке программного обеспечения.

Главная польза каждого отдельного шаблона состоит в том, что он описывает решение целого класса абстрактных проблем. Таким образом, за счёт шаблонов производится унификация терминологии, названий модулей и элементов проекта. Однако есть мнение, что слепое применение шаблонов из справочника, без осмысления причин и предпосылок выделения шаблона, замедляет профессиональный рост программиста, так как подменяет творческую работу механической подстановкой.

Что бы программист ни разрабатывал, на каком бы языке он ни писал, если он стремится к хорошему коду, он будет использовать шаблоны проектирования. Он стремится повторно воспользоваться решениями, которые оказались удачливыми ранее.

Паттерны проектирования представляют наилучшие решения часто встречаемых задач и упрощают повторное использование удачных решений. Шаблоны проектирования не должны ограничивать.

В зависимости от назначения выделяют:

1. Структурные шаблоны (*structural patterns*) – показывают, как объекты и классы объединяются для образования сложных структур.

2. Порождающие шаблоны (*creational patterns*) – контролируют процесс создания и жизненный цикл объектов.

3. Шаблоны поведения (*behavioral patterns*) – используются для организации, управления и объединения различных вариантов поведения объектов.

Каждый шаблон проектирования описывает задачи, с которыми программисту часто приходится сталкиваться. И затем описывает основу решения этой задачи таким образом, что вы можете воплотить это решение при разработке других программ, ни разу не повторившись.

Практические решения, в свою очередь, подразделяются на порождающие паттерны, структурные паттерны и паттерны поведения.

1) Порождающие паттерны или паттерны создания объектов: абстрактная фабрика, одиночка, прототип, строитель, фабричный метод.

Первая группа - это *creational* паттерны. Они в той или иной степени работают с механизмами создания объектов.

Singleton - обеспечиваем существование в системе ровно одного экземпляра некоторого класса;

Factory Method - делегируем процесс создания объектов классам-наследникам;

Prototype - клонируем объекты на основании некоторого базового объекта;

Builder - отделяем процесс создания комплексного объекта от его представления;

Abstract Factory - описываем сущность для создания целых семейств взаимосвязанных объектов.

2) Структурные паттерны: адаптер, декоратор, заместитель, компоновщик, мост,

приспособленец, фасад. Они описывают создание более сложных объектов, либо упрощают работу с другими объектами сиситмы.

Adapter - на основании некоторого класса создаем необходимый клиенту интерфейс;

Facade - описываем унифицированный интерфейс для облегчения работы с набором подсистем;

Composite - работаем с базовыми исоставными объектами единым образом;

Decorator - динамически добавляем новую функциональность некоторому объекту, сохраняя его интерфейс;

Proxy - создаем объект, который перехватывает вызовы к другому объекту;

Bridge - разделяем абстракцию от интерфейса, позволяя им меняться независимо;

Flyweight - эффективно работаем с огромным количеством схожих объектов.

3) Паттерны поведения: интерпретатор, итератор, команда, наблюдатель, посетитель, посредник, состояние, стратегия, хранитель, цепочка обязанностей, шаблонный метод.

Они определяют эффективные способы взаимодействия различных объектов в системе.

Strategy - описываем набор взаимозаменяемых алгоритмов с единым интерфейсом;

Iterator - обеспечиваем доступ к коллекциям объектов без раскрытия внутреннего устройства этих коллекций;

Observer - создаем объект для отслеживания изменений в подсистеме и нотификации других подсистем;

Memento - сохраняем внутреннее состояние объекта для последующего использования без нарушения инкапсуляции;

Command - описываем объект, представляющий собой некоторое действие, которое можно выполнить в необходимый момент;

Interpreter - определяем способ вычисления выражений некоторого языка;

Mediator - создаем объект, которые регулирует взаимодействие между набором подсистем;

State - позволяем объекту менять свое поведение при изменении его внутреннего состояния;

Template method - описываем алгоритм, возлагая реализацию некоторых частей алгоритма на подклассы;

Visitor - отделяем алгоритм от структуры, с которыми алгоритм работает;

Chain of responsibility - пропускаем некоторый запрос через набор обработчиков событий, до тех пор пока запрос не будет обработан.

В *PHP*, *Java* и других объектно-ориентированных языках программирования программистами всего мира уже реализованы и описаны эти практические решения.

Сразу же стоит указать на ограничения по их применению.

Во-первых, шаблоны группы практических решений необходимо применять только тогда, когда имеется четкое понимание необходимости их использования и дополнительная гибкость действительно необходима. Если за гибкость приходится платить усложнением дизайна либо ухудшением производительности, либо подгоном решения под выбранный паттерн, тогда применение паттернов практических решений необоснованно. Необоснованное применение сложных паттернов при решении простых задач усложняет задачу. Поэтому дальнейшее проектирование системы зависит от ответа на этот важный вопрос: использовать или не

использовать при решении конкретной задачи готовое практическое решение.

Шаблоны проектирования нужны для того, чтобы помочь реализовать какую-то идею, а не для того, чтобы уместить идею в рамки некоторого паттерна.

Во-вторых, использовать шаблоны практических решений рационально только после изучение конкретного языка программирования.

6 РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ РАЗРАБОТАННОЙ СИСТЕМЫ, И ОЦЕНКА ВЫПОЛНЕНИЯ ЗАДАЧ

После запуска исполняемого файла перед пользователем открывается главная страница сайта, главная страница сайта при входе пользователя приведена на рисунке 6.1.

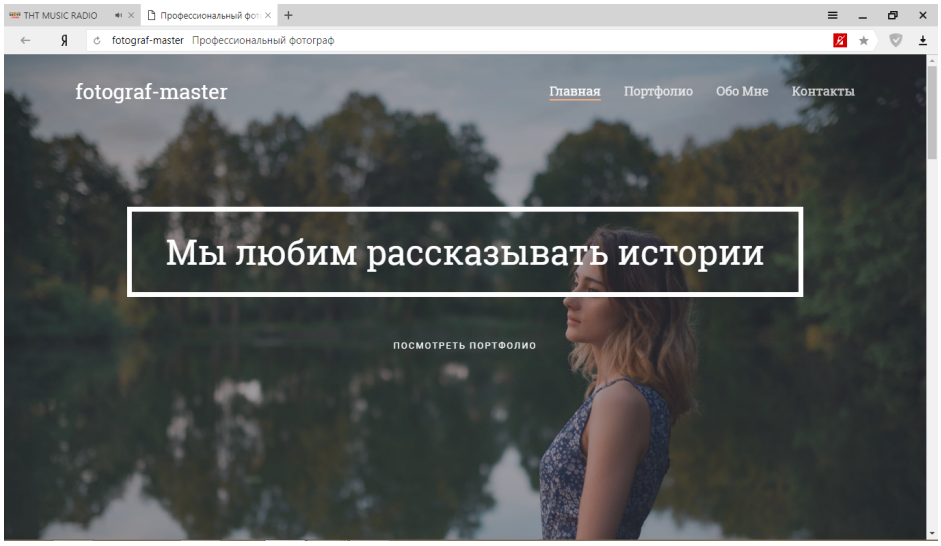

Рисунок 6.1 – Главная страница сайта-портфолио фотографа

С главной страницы можно просмотреть портфолио, информацию о фотографе и его контакты. Результаты данных процедур представлены на рисунке 6.2 – 6.4.

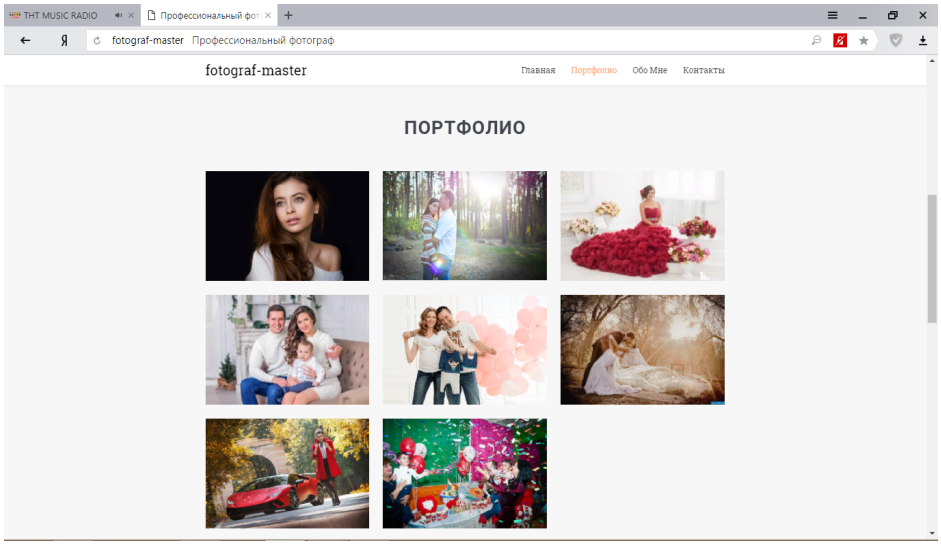

Рисунок 6.2 – Портфолио

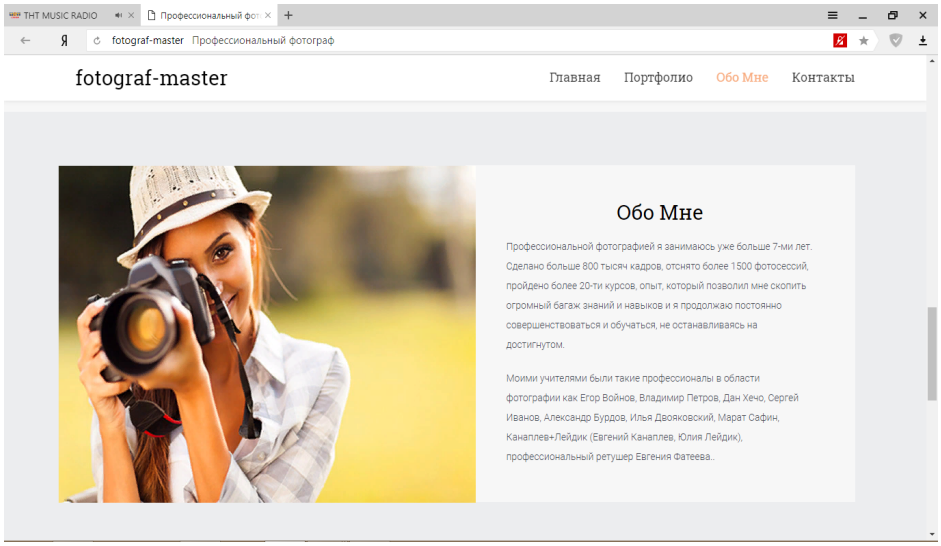

Рисунок 6.3 – Обо мне

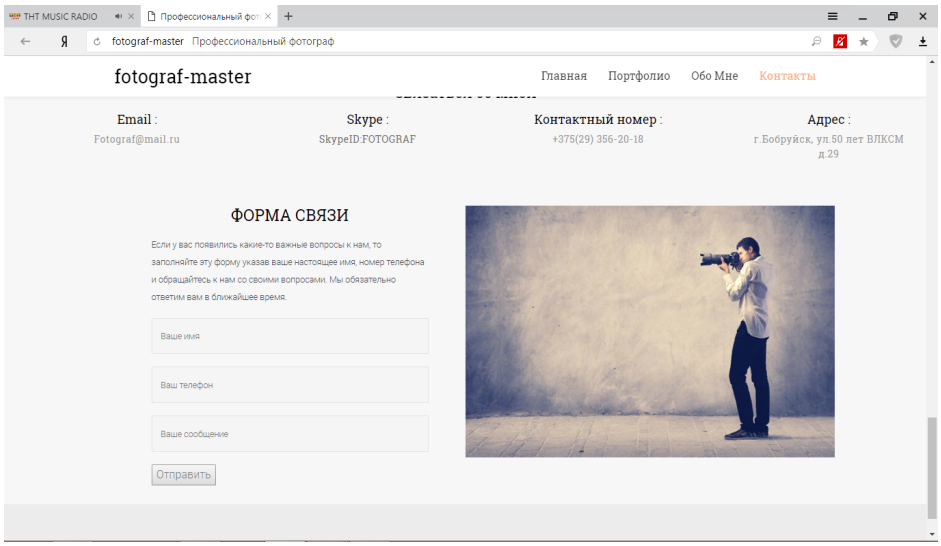

Рисунок 6.4 – Контакты

Также на сайте доступна возможность просмотра дополнительной информации к портфолио, а именно: фотоссесии. Результаты данных процедур представлены на рисунках 6.5 – 6.12.

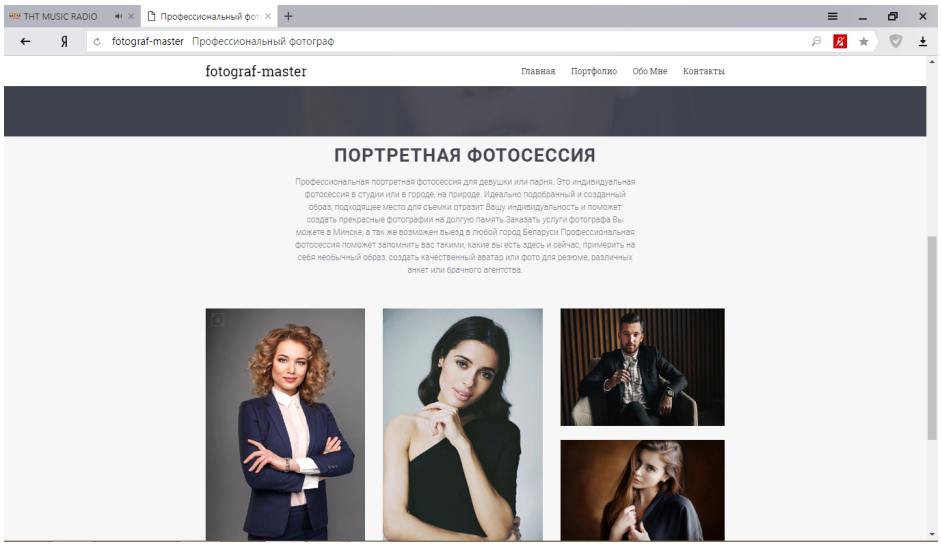

Рисунок 6.5 – Фотоссесия Портрет

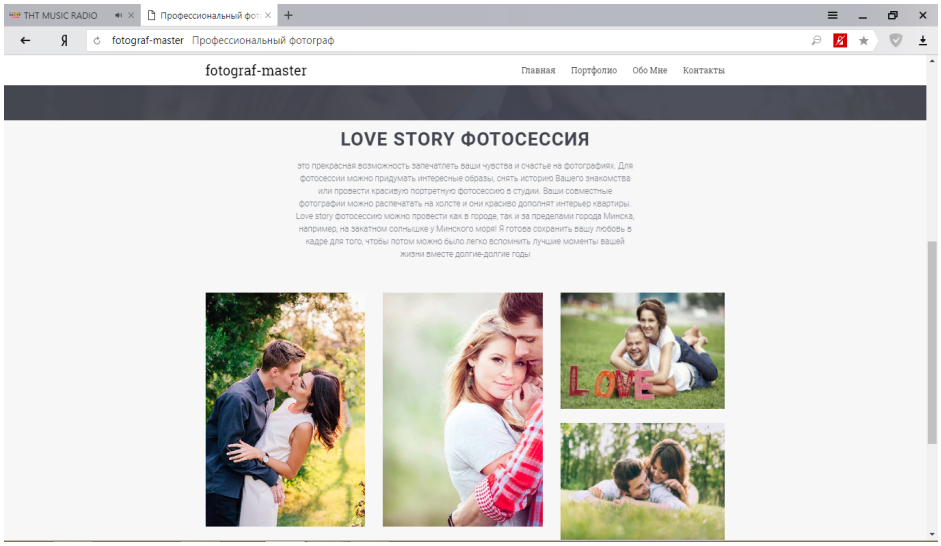

Рисунок 6.6 – Фотоссесия Love Story

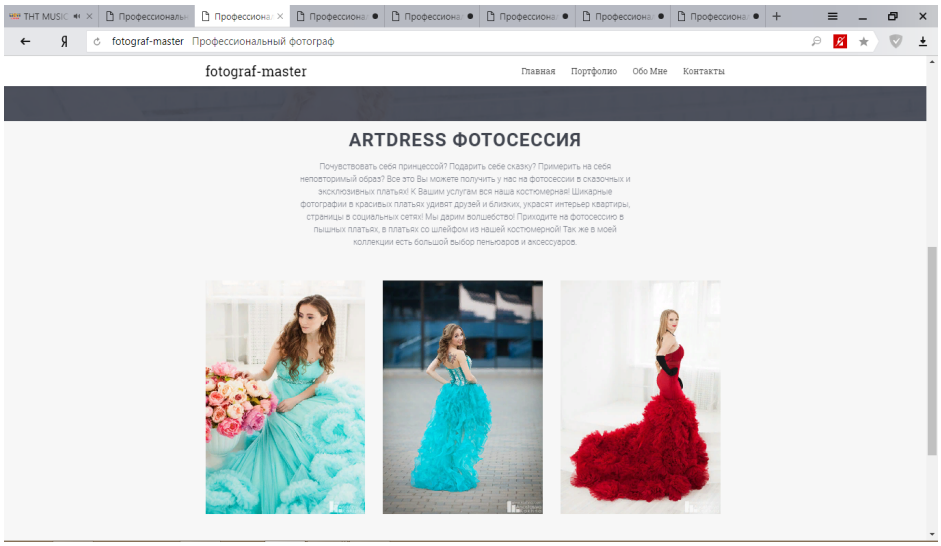

Рисунок 6.7 – Фотоссесия Artdress

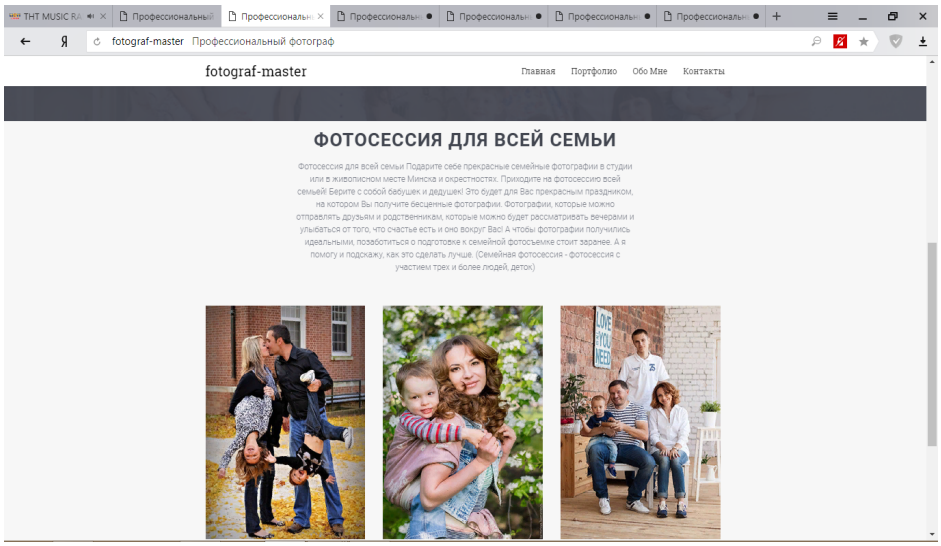

Рисунок 6.8 – Фотоссесия для всей семьи

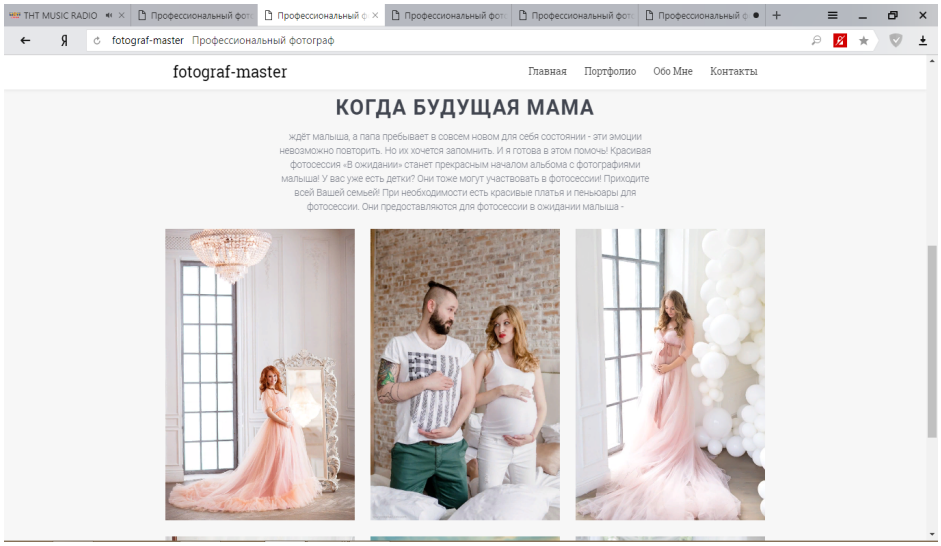

Рисунок 6.9 – Фотоссесия для будущих мам

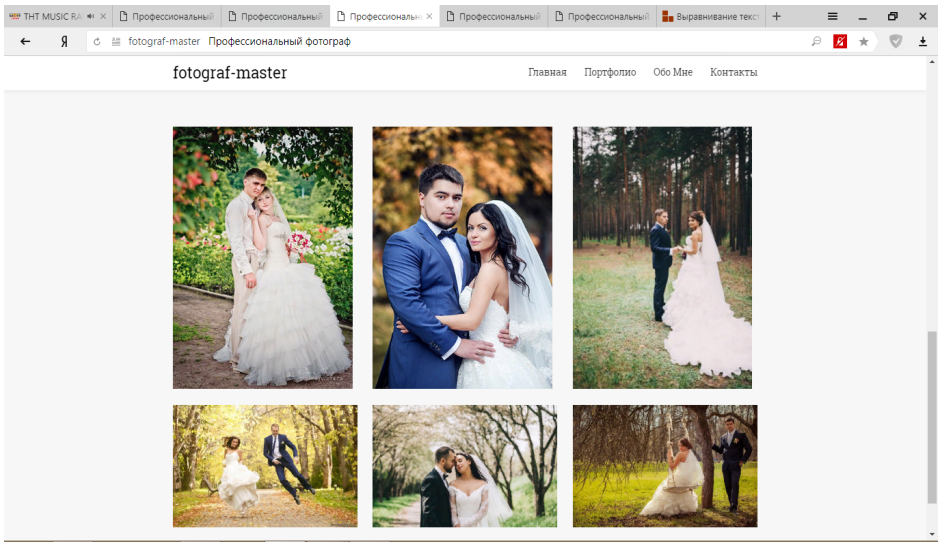

Рисунок 6.10 – Свадебная фотоссесия

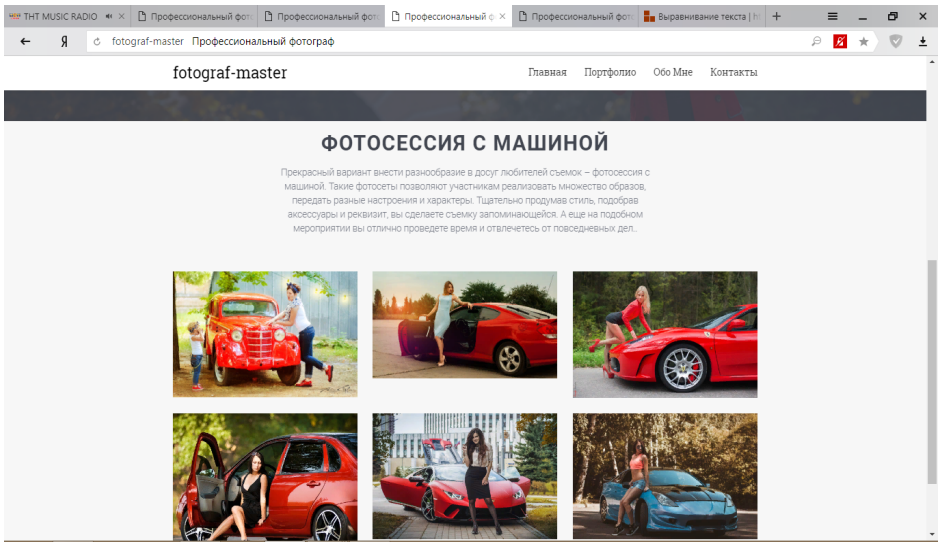

Рисунок 6.11 – Фотоссесия с машиной

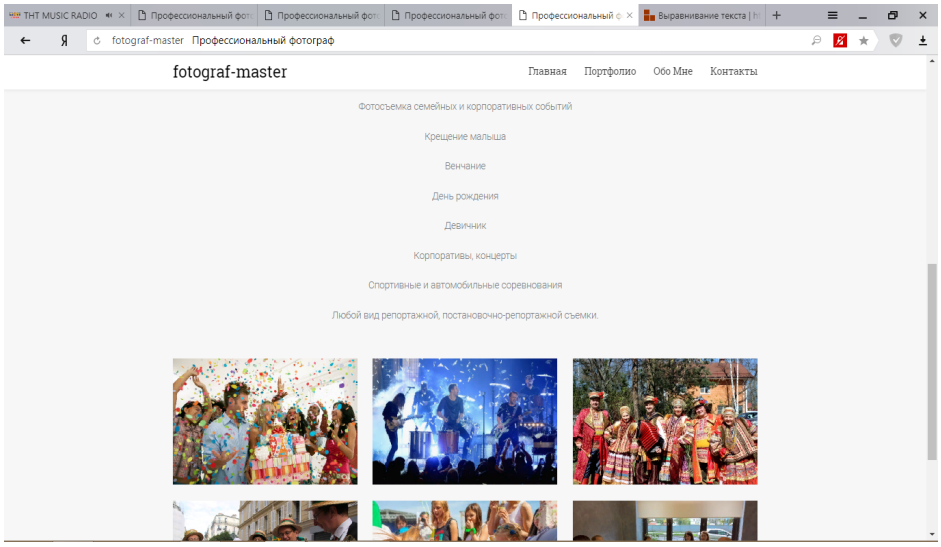

Рисунок 6.12 – Фотоссесия на мероприятиях

Так же на сайте доступна панель администратора для редактирования базы данных.

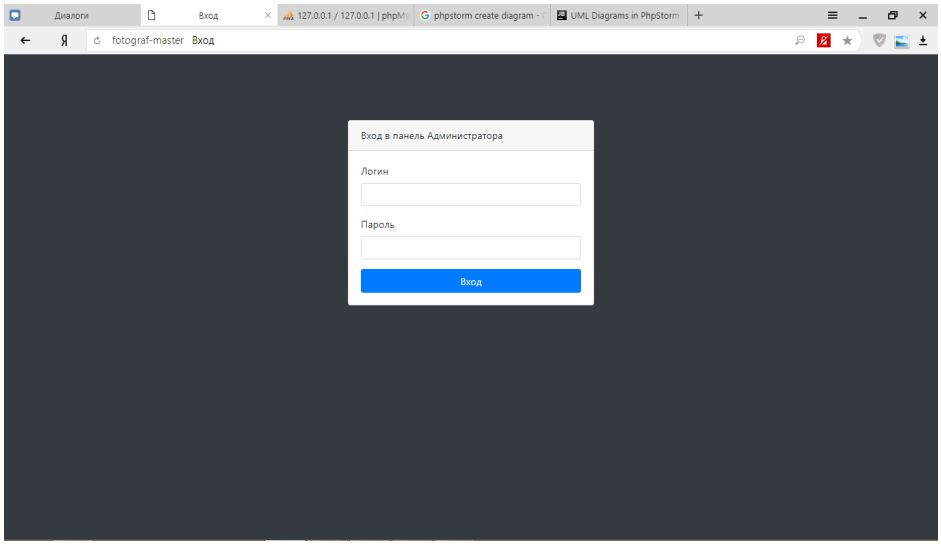

Рисунок 6.13 – Вход в панель администратора

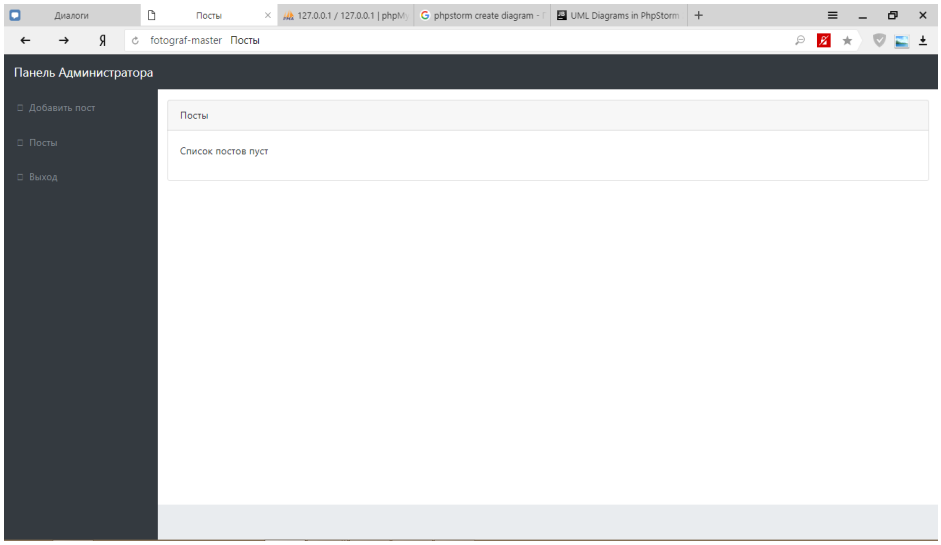

Рисунок 6.14 – Список постов

Таким образом, был разработан сайт для фотографа "Фотомастер", позволяющий просматривать всю необходимую информацию. Разработанным веб-приложением предусмотрен подробное описание к портфолио и так далее.

7 ПРИЛОЖЕНИЕ А (Листинг)

https://github.com/darkTeiman/Fotograf-master

8 ПРИЛОЖЕНИЕ Б

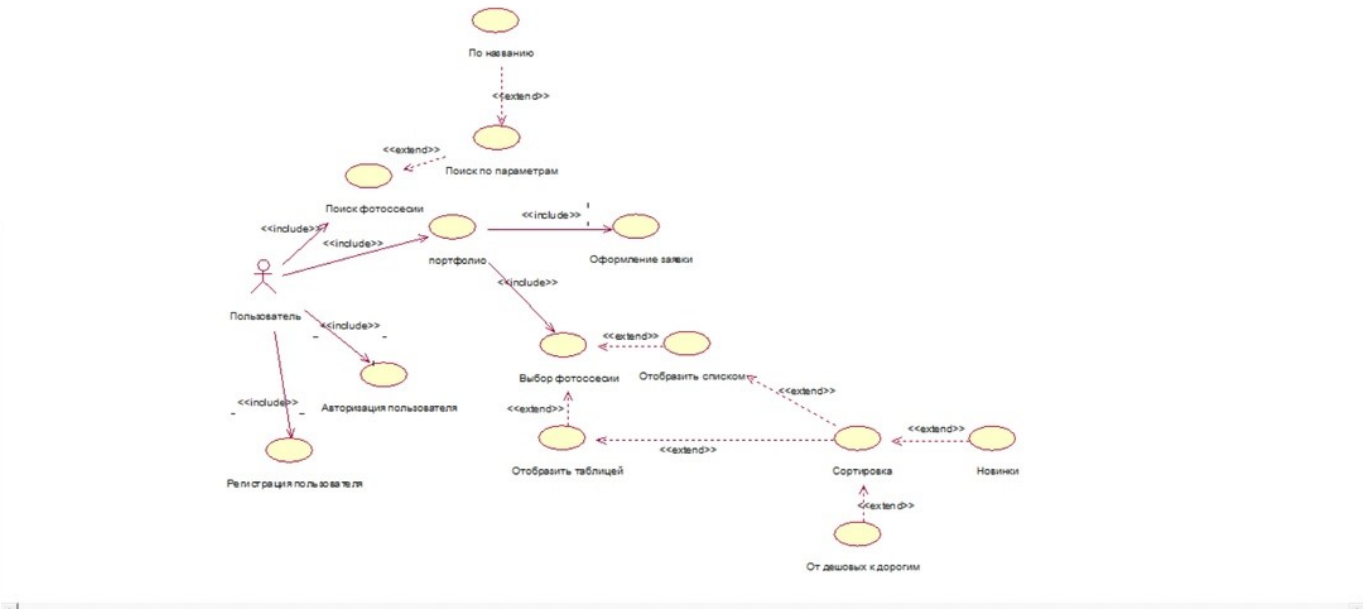

Рисунок 1 - Диаграмма вариантов использования

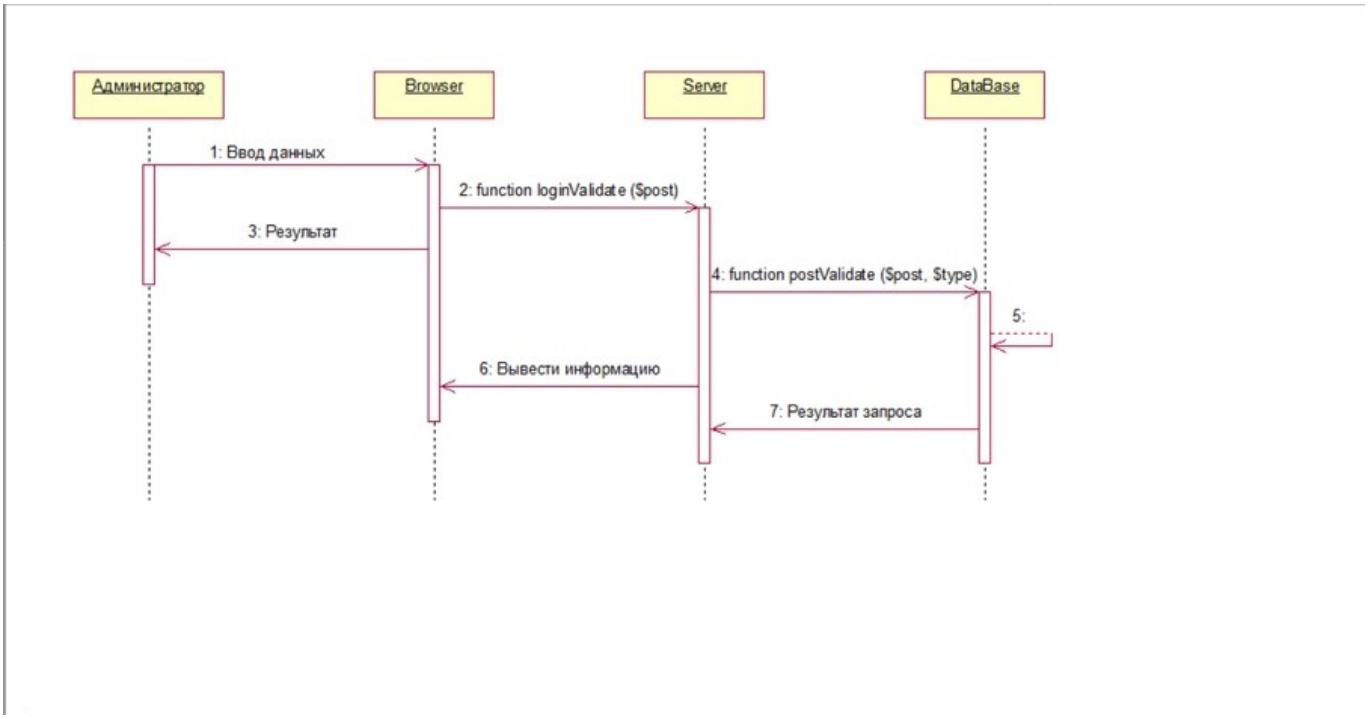

Рисунок 2 – Диаграмма последовательности

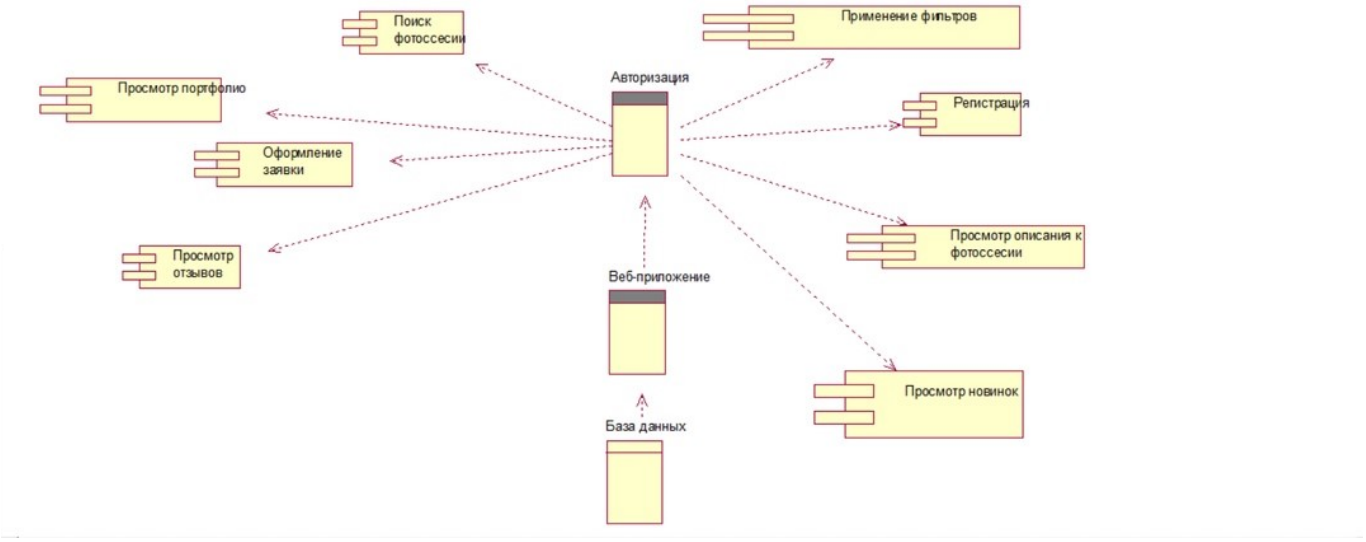

Рисунок 3 – Диаграмма компонентов

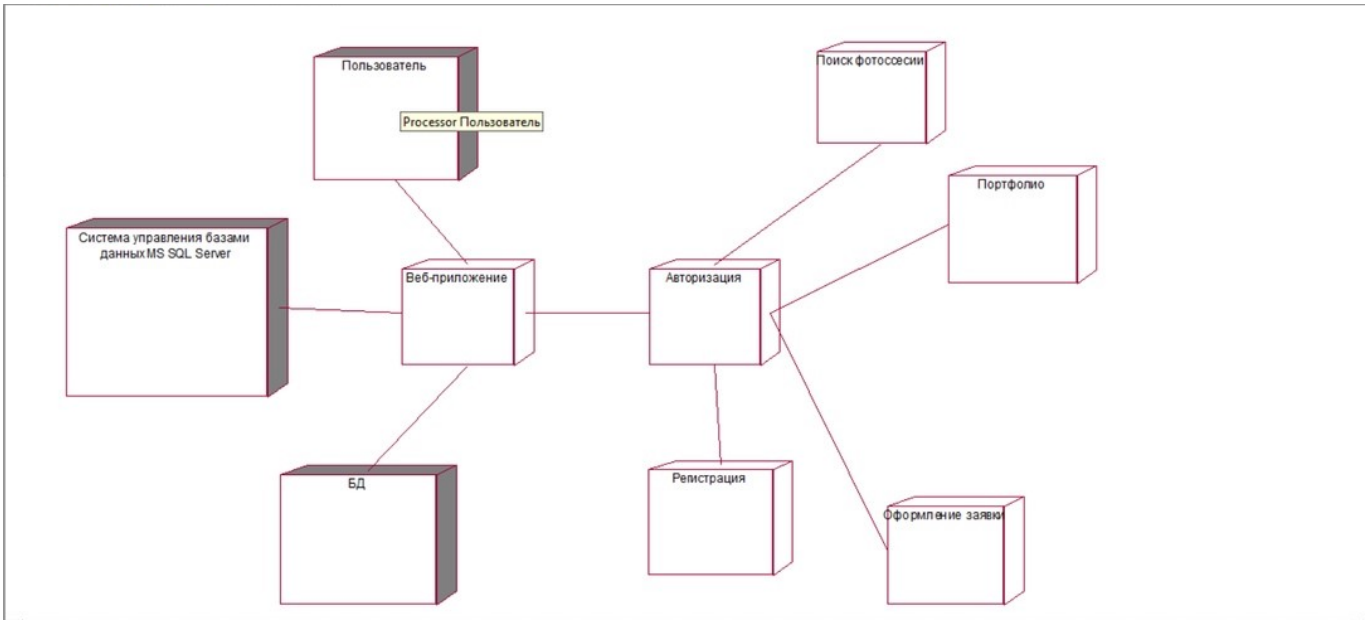

Рисунок 4 - Диаграмма развертывания

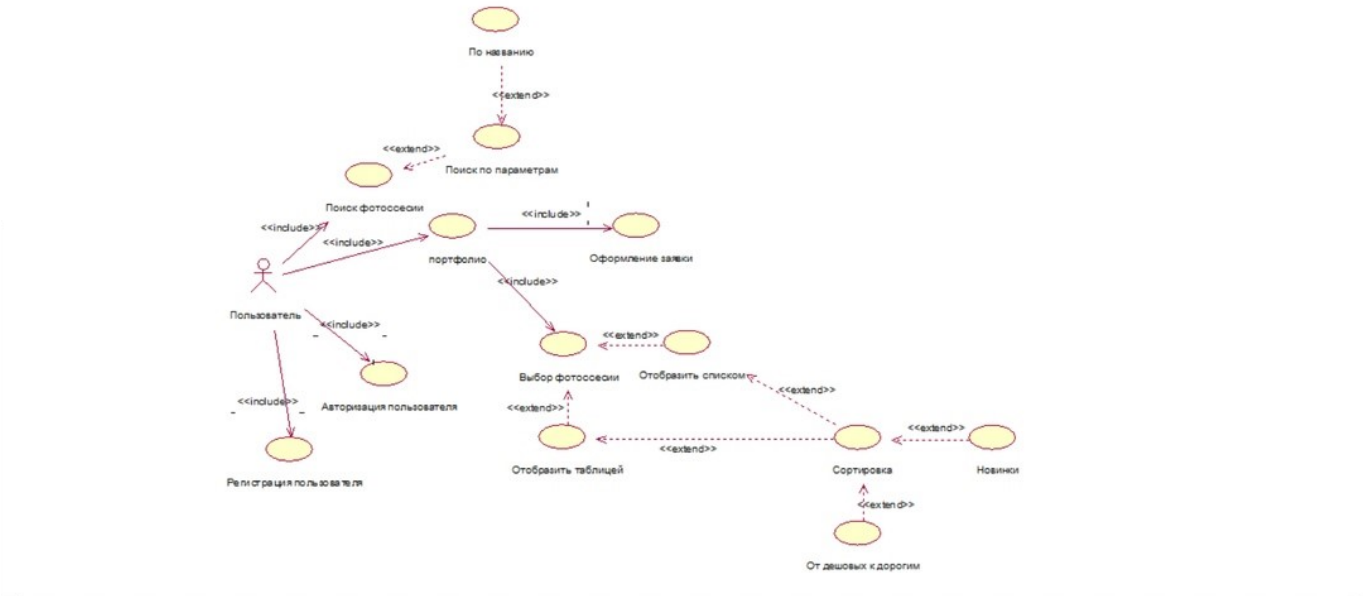

Рисунок 5 – Диаграмма состояний

Заключение

В данной курсовой работе был разработан сайта для фотографа «Фотомастер». Основными технологиями проекта являются PHP, MySQL. Программная среда проекта- Open Server. В результате работы, поставленные задачи были выполнены. А именно, описан проект: веб-сайт создан для того чтобы познакомить пользователя с работой фотографа, информацией о фотоссесиях, а также с контактной информацией фотографа. Создан дизайн-макет веб-сайта, который совпадает с готовым проектом. Git- мощная и сложная распределенная система контроля версий. Система контроля версий (СКВ) — это система, регистрирующая изменения в одном или нескольких файлах с тем, чтобы в дальнейшем была возможность вернуться к определённым старым версиям этих файлов. Также проект размещен на удаленном репозитории. Ссылка на репозиторий: https://github.com/darkTeiman/Fotograf-master. Для решения распространённых проблем, возникающих при разработке программного обеспечения используется - шаблон проектирования. В целом предлагаемый веб-сайт позволит значительно повысить эффективность деятельности фотографа, так как сайт-визитка содержит основную информацию об организации, частном лице, компании, товарах или услугах, прайс-листы, контактные данные и другую полезную информацию, что способствует привлечению новых покупателей и как следствие получение наибольшей выгоды.

Список использованных источников

1. [url] **github** https://github.com/darkTeiman/Fotograf-master
2. [печатное издание] **[1] Березнева, Е. А. Автоматизация работы с документами: от простого к сложному / Е. А. Березнева – Москва : Книжный мир, 2006. – 175 с.** [1] Березнева, Е. А. Автоматизация работы с документами: от простого к сложному / Е. А. Березнева – Москва : Книжный мир, 2006. – 175 с.
3. [печатное издание] **[2] Wikipedia [Электронный ресурс]. – Электронные данные. – Режим доступа : http://ru.wikipedia.org/wiki/Веб-приложение/.**
4. [url] **[3] Blojek [Электронный ресурс]. – Электронные данные. – Режим доступа : http://blojek.info/preimushhestva-i-nedostatki-veb-prilozhenij/.**
5. [url] **4] Web-приложения – преимущества и недостатки [Электронный ресурс]. – Электронные данные. – Режим доступа : http://mydiv.net/arts/view-web-prilozhenija_preimuxhestva_i_nedostatki.html.**
6. [печатное издание] **[5] Ожегов, С. И. Толковый словарь русского языка / С. И. Ожегов, Н. Ю. Шведова. – М. : ООО «ИТИ Технологии», 2006. – 944 стр.**
7. [url] **[6] Wikipedia [Электронный ресурс]. – Электронные данные. – Режим доступа : http://ru.wikipedia.org/wiki/Электронный_документ/.**
8. [url] **[7] Philosoft [Электронный ресурс]. – Электронные данные. – Режим доступа : http://www.philosoft.ru/gost34asconcept.zhtml.**
9. [печатное издание] **[8] Методология функционального моделирования IDEF0. Руководящий документ / Научно-исследовательский Центр CALS – технологий «Прикладная Логистика». – М. : ИПК «Издательство стандартов», 2000. – 75 стр.**
10. [печатное издание] **[9] Буч, Г. Язык UML. Руководство пользователя. 2-е изд. / Г. Буч, Д. Рамбо, И. Якобсон. – М. : ДМК Пресс, 2006. – 496с.**
11. [печатное издание] **[10] Гамма, Э. П75 Приемы объектно-ориентированного проектирования. Паттерны проектирования / Э. Гамма [и др.]. – СПб : Питер, 2001. – 368 с.: ил.**
12. [печатное издание] **[11] Похилько, А. Ф. CASE-технология моделирования процессов с использованием средств BPWin и ERWin учебное пособие / А. Ф. Похилько, И. В. Горбачев. – Ульяновск : УлГТУ, 2008. – 120 с.**

13. [печатное издание] **12] Маклаков, С. В. BPwin и ERwin. CASE-средства разработки информационных систем / С. В. Маклаков. – М. : ДИАЛОГ-МИФИ, 1999. – 256 с.**

Приложения

1. [электронный документ] 5ed00023c7f19_Лист задания.docx