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
12.05.2024

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

на тему

Автоматизация продаж женских платьев

БГУИР КР 1-40 05 01-10 № 170 ПЗ

Студент (подпись студента)

Курсовая работа
представлена на проверку
12.05.2024
_____
(подпись студента)

2024

Реферат

БГУИР КР 1-40 05 01-10 № 170 ПЗ, гр. 784371

, Автоматизация продаж женских платьев, Минск: БГУИР - 2024.

Пояснительная записка 138140 с., 28 рис., 19 табл.

Ключевые слова: Платья, платья женские, сарафаны, вечерние платья.

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

Автоматизация продаж женских платьев – это реальный способ увеличения объёма продаж, как в отдельной торговой точке, так и в случае сетевой организации бизнеса. Особый эффект достигается при большом ассортименте товаров и повышенном потоке покупателей.

Automation of sales of women's dresses is a real way to increase sales, both in a separate point of sale, and in the case of a network business organization. A special effect is achieved with a large range of products and an increased flow of customers.

Содержание

Введение

Курсовой проект выполнен на тему «Интернет-магазин женских платьев». Интернет-магазин – сайт, торгующий товарами посредством сети Интернет. Позволяет пользователям онлайн, в своём браузере или через мобильное приложение, сформировать заказ на покупку, выбрать способ оплаты и доставки заказа, оплатить заказ. Создание веб-сайтов является одной из важнейших технологий разработки ресурсов Интернет. Веб-сайт – это информационный ресурс, состоящий из связанных между собой гипертекстовых документов (Веб-страниц), размещённый на веб-сервере и имеющий индивидуальный адрес. Интернет-магазин содержит информацию о продаваемом товаре, контактные данные и другую полезную информацию, что способствует привлечению новых покупателей и как следствие получение наибольшей выгоды. Получение наибольшей выгоды является основной задачей любого предприятия и предпринимателя, таким образом, внедрение интернет-магазин способствует развитию бизнеса. В курсовой работе поставлена цель: разработка интернет-магазина. Для достижения

поставленной цели необходимо решить следующие задачи: описание проекта; выбор технологий и инструментария; создание дизайн-макета интернета-магазина; выбор архитектурного шаблона проектирования; программирование сайта; наполнение сайта информацией; размещение на открытый репозиторий.

1 ЛИСТ ЗАДАНИЯ

Министерство образования Республики Беларусь

Учреждение образования

Белорусский государственный университет информатики и радиоэлектроники

Факультет непрерывного и инновационного обучения

Кафедра проектирования информационно-компьютерных систем

«УТВЕРЖДАЮ»
Заведующий кафедрой
_____ В.В. Хорошко
«___» _____ 2020

З А Д А Н И Е

к курсовой работе по дисциплине «Современные технологии проектирования информационных систем»

Фамилия, имя, отчество Матюшёнок Екатерина Аркадьевна

группа 784371

1.Тема проекта: Автоматизация продаж женских платьев

2.Сроки сдачи студентом законченного проекта: 3 мая 2020 г.

3.Исходные данные к проекту:

3.1.Описание к выполнению Автоматизация продаж женских платьев с использованием архитектурного шаблона проектирования MVC.

3.2.Язык и среда программирования – на выбор студента. Однако разработанное

программное обеспечение должно быть реализовано на объектно-ориентированном языке.

3.3.В реализации программного обеспечения учесть возможность использования сервера.

3.4.Пояснительную записку и графический материал выполнять по СТП БГУИР 01-2013.

3.5.Другие требования уточняются студентом в процессе работы.

4. Содержание расчётно-пояснительной записки (перечень подлежащих разработке вопросов):

Титульный лист. Заполненный бланк задания с приложением. Содержание (1-2 *стр.*)

Введение (*1 – 3 стр. Актуальность темы курсовой работы; цель и перечень задач, которые планируется решить; детальная постановка задачи*).

4.1.Описание проекта (*10 – 15 стр. Описание серверной и клиентской части разрабатываемого проекта*).

4.2.Обоснование выбора технологий (*7-15 стр. Технологии программирования, используемые для решения поставленных задач. Реализация объектно-ориентированных технологий программирования в современных программно-математических средах*).

4.3.Инструментарий (*5-7 стр. Обоснование используемых инструментов. Использование системы контроля версий GIT. Обязательна ссылка на репозиторий с проектом, например github.com.*).

4.4.Архитектурный шаблон проектирования MVC (*5-7 стр. Разработка схемы алгоритма, диаграммы последовательности и диаграммы состояний (схемы в Приложении) с детальными пояснениями каждого компонента шаблона проектирования MVC или его модификаций*).

4.5.Шаблон проектирования практических решений (*7-10 стр. Использование шаблонов проектирования практических решений для решения практических задач*).

Заключение (*1 стр. Выводы по курсовой работе*).

Список литературных источников (*1, 2 стр. Перечень литературы и интернет-источников, которые были реально использованы при выполнении курсовой работы).*

Приложения (*3 и более стр. Ведомость документов, листинг программного кода и др.).*

5.Перечень графического материала (с указанием обязательных чертежей и графиков):

5.1.Структура графического пользовательского интерфейса (формат А3 или несколько А4)

5.2.Схема алгоритма (формат А3 или несколько А4)

5.3.Диаграмма последовательности (формат А3 или несколько А4)

5.4.Диаграмма состояний (формат А3 или несколько А4)

6.Консультант по работе: Михалькевич Александр Викторович

7.Дата выдачи задания:

8.Календарный график работы над проектом на весь период проектирования:

№ п/п	Наименование этапов курсового проекта	Срок выполнения этапов проекта	Примечание

1.	1-я опроцентовка (пп. 4.1, 4.2, 5.1)	04.03.2020	40%
2.	2-я опроцентовка (пп. 4.3, 4.4, 5.2, 5.3)	01.04.2020	70%...80%
3.	3-я опроцентовка (пп. 4.5, 4.6, 5.4, приложения)	29.04.2020	95%
4.	Сдача на проверку и защита курсового проекта	03.05.2020	100%
5.	Защита курсового проекта	10-11.05.2020	Согласно графику

Руководитель _____

А.В.Михалькевич

Задание принял к исполнению

2 ОПИСАНИЕ ПРОЕКТА

Темой проекта является: интернет-магазин с каталогом товаров для женщин, а именно платьев. Данный веб-сайт создаётся для того, чтобы познакомить пользователя с перечнем продаваемых товаров, информацией о данных товарах, а также с контактной информацией магазина.

1. **Дизайн-макет веб-сайта**

В качестве цветового решения выбран светлый фон для того, чтобы пользователь концентрировал внимание на продаваемые товары.

Далее представим структуру страниц будущего сайта.

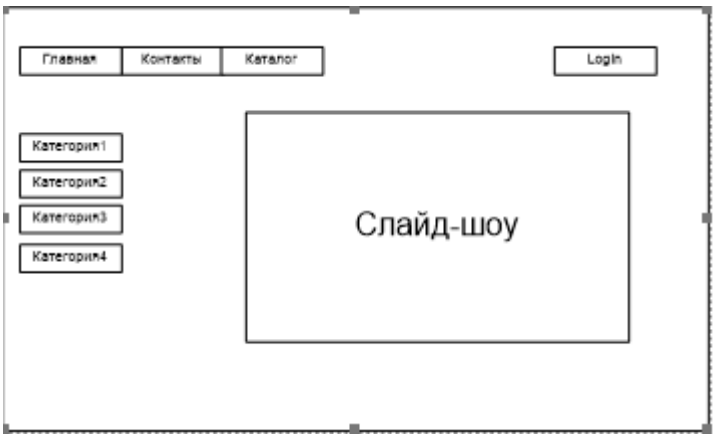

Рисунок 1 – Структура главной страницы сайта

Главная страница содержит область со слайд-шоу, в которой представлены некоторые товары, ссылки на контакты, каталог товаров и ссылка на регистрацию для добавления товаров, а также ссылки на категории товаров для быстрого поиска.

Перейдя по ссылке контакты, пользователь может найти всю необходимую информацию о

магазине (адрес, время работы, телефон, e-mail и т.д.).

На рисунке 2 показана структура страницы каталог, на котором представлены товары магазина.

Кликая на именование товара (название товара 1, название товара 2,…) или на картинку, пользователь перейдет на страницу с увеличенной картинкой и кратким описанием этого товара (размер изделия и т.д.).

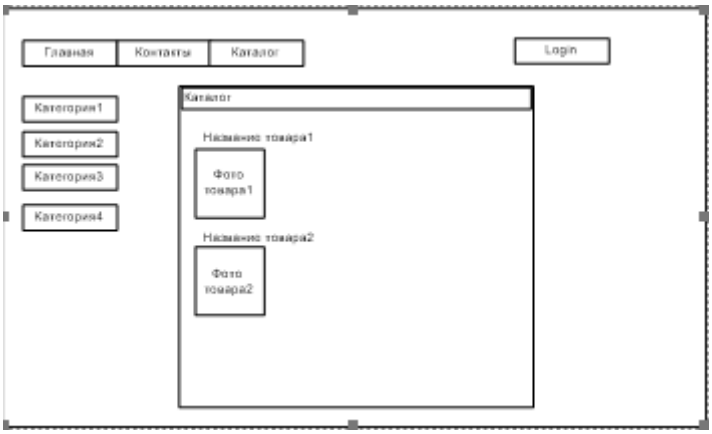

Рисунок 2 – Структура страницы каталог

1.2.Технологии и инструментарий проекта

Технологии проекта: *PHP*, *MySQL*, *Apache*.

Обоснование выбора технологии: выбор программ обусловлен тем, что они являются объектно-ориентированными программами, а также веб-оптимизированными программами.

Инструментарий: Программной средой проекта был выбран *Open Server*. Ссылка скачивания: http://open-server.ru/

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

При запуске *Open Server* становятся доступны следующие возможности.

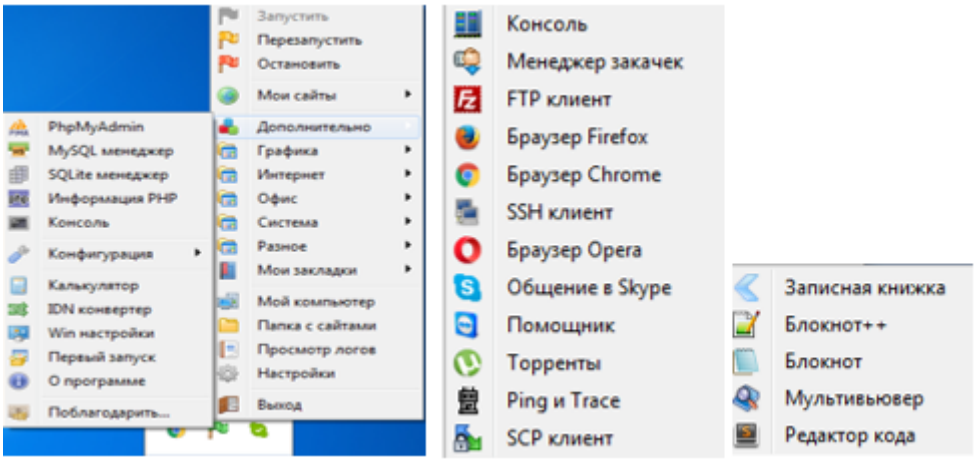

Рисунок 3- *Open Server*

В разделе 1 описан проект: показан дизайн-макета веб-сайта, и описан интерфейс пользователя, обоснованы технологии и инструментарий проекта. Основными технологиями проекта являются *PHP, MySQL*. Программная среда проекта- *Open Server*.

3 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.

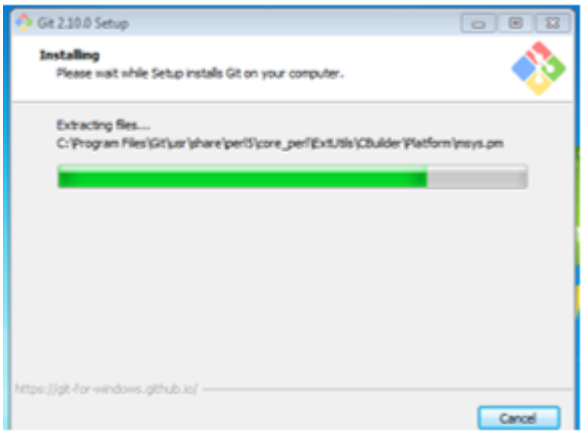

Рисунок 4 – Установка *Git*

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.

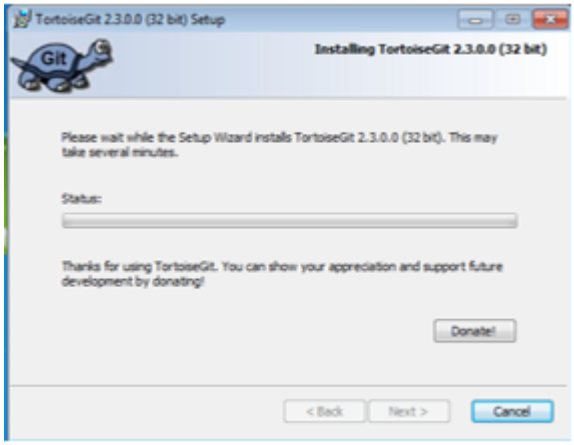

Рисунок 5 – Установка TortoiseGit

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* – добавление удалённого репозитория;

5. *Git push project master* – вносим изменения в удаленный репозиторий.

Ссылка на удаленный репозиторий:

https://github.com/chupris-ek/website_shop

В разделе 2 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 LARAVEL

Laravel – фреймворк для построения веб-приложений с выразительным и элегантным синтаксисом. Процесс разработки только тогда наиболее продуктивен, когда работа с фреймворком приносит радость и удовольствие.

Laravel – попытка сгладить все острые и неприятные моменты в работе *php*-разработчика. Он берёт на себя аутентификацию, роутинг, работу с сессиями, кеширование, внедрение зависимостей и многое другое, что встречается в большинстве приложений, оставив вам только фокус на вашей задаче.

Laravel стремится сделать процесс разработки приятным для разработчика без ущерба для функциональности приложений. Для этого мы попытались объединить всё самое лучшее из того, что мы видели в других фреймворках, – *RubyOnRails*, *ASP.NET* и Синатра, *Kohana*, *Yii*. Превосходный *IoCcontainer*, встроенные миграции и интегрированная поддержка юнит-тестов дают вам мощные инструменты для того, чтобы сделать именно тот функционал, который вам нужен.

Laravel использует менеджер зависимостей *Composer*. В нашем проекте используется, *OpenServer*, *composer* поставлен совместно с пакетом программы *OpenServer-a*. Отдельно его скачивать и устанавливать не нужно. Прежде чем устанавливать *Laravel*, давайте убедимся, что *Composer* работает. Для этого запустим встроенную консоль *OpenServer*.

В консоли наберём команду «*Composer*»:

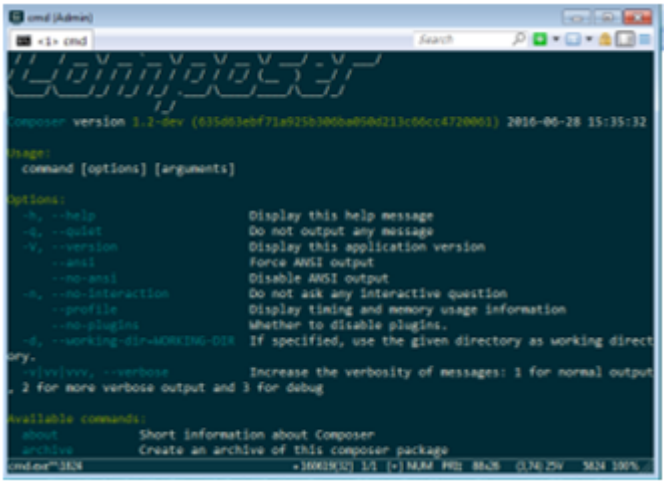

Рисунок 6 – Консоль

Такой ответ консоли свидетельствует о том, что *composer* установлен.

Устанавливаем *laravel* через консоль, для этого в консоли прописываем команду: *Comoser create-project laravel/laravel -prefer-dist*.

Composer создаст папку *laravel*, куда установится проект *laravel*, со следующей структурой:

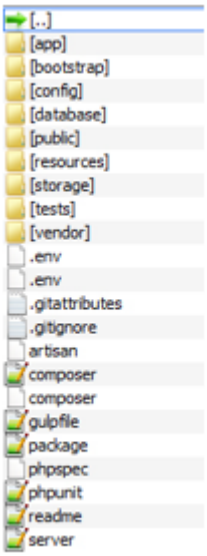

Рисунок 7 – *LARAVEL*

3.1 Теория 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.

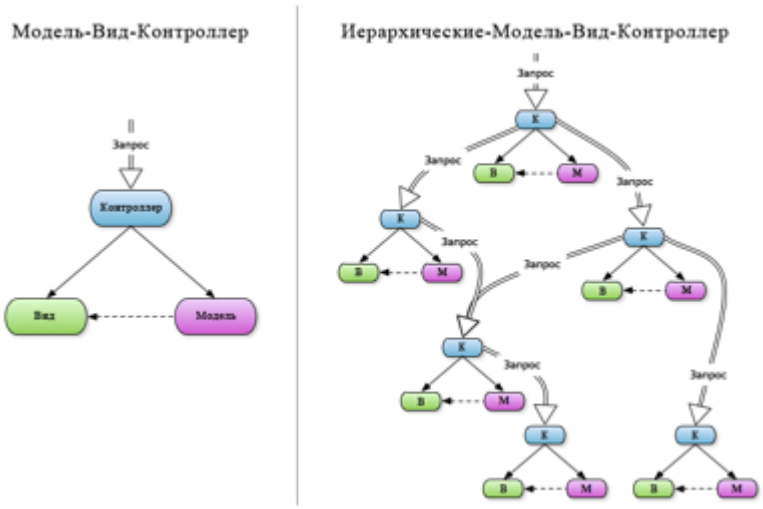

Рисунок 8 – Модели

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 собирать 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.

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*). Всё 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.

Машрутизация

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

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

Например, регистрация маршрута, отвечающего на *GET*-запрос:

```
Route::get('/', function () {
  return "Привет, мир!";
});
```

Первый параметр – адрес маршрута, который вы регистрируете в маршрутизаторе (*Router*). Второй параметр – функция, содержащая логику для этого маршрута. Маршруты регистрируются без задания ведущего слеша (/) – единственное исключение – корневой маршрут, который состоит из одного слеша ('/').

Применение роутов в данном курсовом проекте:

```
Route::get('/catalog','CatalogController@getAll');
Route::get('/', 'BaseController@getIndex'Auth::routes();
Route::get('/home', 'HomeController@index');
//defoult route-всегда последний
Route::get('/{id}','StaticController@getIndex');
```

Контроллеры

Контроллеры хранятся в папке *app/http/controllers/*. Этот путь, в свою очередь определён в файле *composer.json* в настройке *classmap*.

Все контроллеры должны наследовать класс *BaseController*. Этот класс также может хранится в папке *app/controllers*, и в него можно поместить общую логику для других контроллеров. *BaseController* расширяет базовый класс *Controller*.

Для создания контроллера в консоли прописали данную команду:

php artisan make:controller NewsController.

Есть несколько способов определения маршрута для контроллера.

В файле *app/routes.php.*

Определение маршрута для контроллера с помощью метода *get*:

```
Route::get('static', 'StaticController@index');
```

С помощью метода cotroller:

```
Route::controller(
        'cabinet' => 'CabinetController',
);
```

С помощью метода *controllers*:

```
Route::controllers([
        'cabinet' => 'CabinetController',
        'user' => 'UserController',
        'works' => 'WorksController',
        'portfolio' => 'PortfolioController',
        'auth' => 'Auth\AuthController',
        'password' => 'Auth\PasswordController',
]);
```

А вот так будет выглядеть сам контроллер:

```
namespace App\Http\Controllers;
class CabinetController extends BaseController {
        public function getIndex()
        {
          echo 'Ok';
        }
}
```

Применение контроллеров в данном курсовом проекте. Контроллер для создания каталога:

CatalogController.php

```
Namespace App\Http\Controllers;
use App\Catalog;
use Illuminate\Http\Request;
use App\Http\Requests;
class CatalogController extends Controller
{
  public function getIndex(){
     $text=Catalog::where('url','index')->first();
     return view('catalog')->with('text',$text);//вывод результата на экран, метод экшин

  }
  public function getAll(){
      $all = Catalog::get();
      return view('catalogall')->with('all', $all);  }}
```

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 будет 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.

Создание модели *User*

Проверка подлинности пользователя требуется почти в каждом современном веб-приложении. Вместо того чтобы заставлять вас писать свою собственную модель пользователя, в *Laravel* уже есть модель пользователя прямо из коробки.

В каталоге *app* находится файл *User.php*. Все модели должны помещаться в этом каталоге и именоваться таким же образом. Например, в данном курсовом проекте создали модель для каталога, файл модели называется *app /Catalog.php*.

Модель *Catalog.php.*

```
php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Catalog extends Model
{
   public $table = "catalog";
}
```

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

```
'mysql' => array(
                    'driver'    => 'mysql',
                    'host'      => 'localhost',
                    'database'  => 'project',
                    'username'  => 'root',
                    'password'  => '',
                    'charset'   => 'utf8',
                    'collation' => 'utf8_unicode_ci',
```

```
            'prefix'    => '',
        ),
```

Миграции

Миграции базы данных являются весьма полезными для любого проекта, особенно для проектов с несколькими разработчиками, позволяя иметь последнюю версию базы данных у всех разработчиков. В *Laravel* для этого достаточно выполнить одну команду в командной строке.

Для создания новой миграции понадобился интерфейс командной строки *Laravel* – «*Artisan*».

Открыли консоль командной строки из папки, где расположен файл *artisan*. В консоли ввели следующую команду:

php artisan make:migration create_categories

В папке *database/migration* создался новый файл 2016_10_25_111814_*Catalog.php*.

Класс миграции содержит два метода *up*() для внесения изменений в таблицу базы данных и *down*() для отмены действий метода *up*(). Например, если мы создаем таблицу в *up*(), то в *down*() её нужно удалить.

Допишем действия *up*() и *down*()

2016_10_25_111814_*Catalog.php.*

```
class Catalog extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        //
      Schema::create('catalog', function(Blueprint $t){
      $t -> increments('id');
      $t ->string('name');
      $t ->text('body');
      $t ->string('url');
      $t ->timestamps();
         });
   }
    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        //
      shema::drop('catolog');
    }
}
```

Внутри функции мы можем использовать следующие методы для определения структуры таблицы:

increments() — добавить автоинкрементируемое поле – его будет иметь боольшая часть ваших таблиц

string() – создать поле *VARCHAR* – правда, «строка» куда более описательное имя, чем в стандарте *SQL*

integer() – добавить целочисленное поле

float() – поле с дробным числом (число с плавающей точкой)

boolean() – логическое («булево») поле – истина (*true*) или ложь (*false*)

date() – поле даты

timestamp() – поле «отпечатка времени», так называемый «*Unix timestamp*»

text() – текстовое поле без ограничения по длине

blob() – большой двоичный объект (*BLOB*)

Перед тем как на основе существующих миграций создали таблицы, создали таблицу *migrations*, в которой *laravel* хранит данные о самих миграциях:

php artisan migrate:install

После того как создали таблицу, выполнили саму миграцию:

php artisan migrate.

Шаблоны

По умолчанию, *laravel* работает с шаблонизатором *blade*. Шаблоны создаются в папке *app/views* и имеют расширение *blade.php*. Шаблоны подключаются в экшне через хелпер *view*(), входящим параметром в который передаётся имя шаблона без расширения *blade.php*.

Сперва создали в папке *view* папку *layouts* для хранения базовых шаблонов.

catalogall.blade.php

```
@extends('layouts.app')

@section('content')
   <div class="container">
      <div class="row">
         <div class="col-md-8 col-md-offset-2">
            <div class="panel panel-default">
               <div class="panel-heading">Каталог</div>
               <div class="panel-body">
                  @foreach($all as $one):
                     <div class="catalogs">
                        <h2>{{$one->name}}</h2>
                     </div>
                  @endforeach
               </div>
            </div>
         </div>
      </div>
   </div>
@endsection
```

3.2 Авторизация

Модуль авторизации поставляется совместно с фрэймворком *Laravel*. Файл конфигурации авторизации находится в файле *config/auth.php*.

По умолчанию, для сохранения пользовательских данных, *Laravel* использует модель *App\User.*

Также модуль поставляется с двумя контроллерами аутентификации из коробки, которые находятся в *App\HTTP\Controllers\Auth*. *AuthController* содержит методы регистрации нового пользователя и аутентификации, в то время как *PasswordController* содержит логику помощи существующим пользователям сбросить свои забытые пароли. Каждый из этих контроллеров использует трэйты (*traits*), чтобы включить их необходимые методы. Для многих приложений вам не нужно будет изменять эти контроллеры вообще.

Для создания шаблонов и ротов авторизации, выполнили следующую команду:

php artisan make:auth

Эта команда создала необходимые папки с шаблонами: *resources/views/auth* и *resources/views/layouts*, обновит файл *routes.php* и создаст ещё один контроллер, *HomeController*, куда будет перенаправляться пользователь после успешной авторизации.

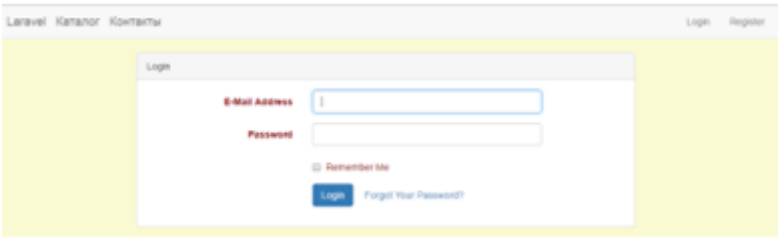

Рисунок 9 – Авторизация

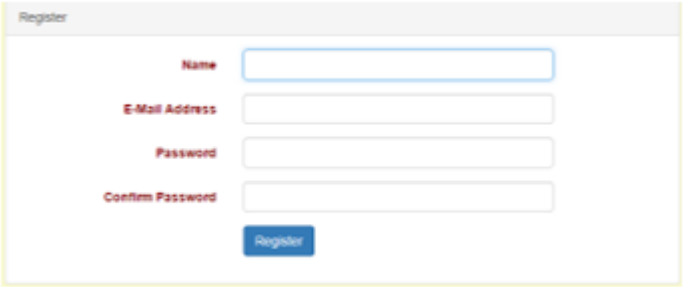

Рисунок 10 – Регистрация

После успешной авторизации пользователь перенаправляется в */home*. Чтобы перенаправить пользователя на другую страницу, в контроллере *AuthController* добавили свойство *$redirectTo*:

```
protected $redirectTo = '/home';
```

Объект авторизированного пользователя мы можем получить с помощью класса *Auth*:

```
$user = Auth::user();
```

Проверка, прошел ли пользователь авторизацию:

```
if (Auth::check()) {
    // The user is logged in...
```

В маршрутах для авторизированных пользователей, мы можем использовать *middlewareauth*:

```
Route::get('profile', ['middleware' => 'auth', function() {
    // Only authenticated users may enter...
}]);
// Using A Controller...
```

```
Route::get('profile', [
'middleware' => 'auth',
    'uses' => 'ProfileController@show'
]);
```

Тот же *middleware* мы можем использовать в конструкторах контроллера:

```
public function __construct()
{
    $this->middleware('auth');
}
```

В разделе 3 рассмотрен фреймворк *Laravel* , на основе которого создан данный проект. Описана теория *MVC* и *HMVC*, а также рассмотрена на примере при создании каталога товаров.

Организован и описан модуль авторизации с помощью данного фреймворка.

5 ПРИМЕНЕНИЕ ШАБЛОНА ПРОЕКТИРОВАНИЯ ПРАКТИЧЕСКИХ РЕШЕНИЙ

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.

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.

Структурные паттерны: адаптер, декоратор, заместитель, компоновщик, мост, приспособленец, фасад. Они описывают создание более сложных объектов, либо упрощают работу с другими объектами системы.

Adapter – на основании некоторого класса создаём необходимый клиенту интерфейс.

Facade – описываем унифицированный интерфейс для облегчения работы с набором подсистем.

Composite – работаем с базовыми и составными 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.

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 чёткое 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.

4.1 *CRUD*

Create, Read, Update and Delete (создание, чтение, обновление и удаление)

Рассмотрим разработку *CRUD* по пунктам.

1 Маршрутизация *(Get*-данные)

Прописан маршрут в роутах (*web.php*):

Route::get('/home', 'HomeController@index');

2 Создание форм

Создана форма в *home.blade.php* (в папке *views*).

Код:

home.blade.php

```
<form method='POST' action='home' enctype='multipart/form-data'>
   {{csrf_field()}}

<div class="form_inner">
   <label for="name">
      <input name="name" id="name">
</div>

<div class="form_inner">
   <label fpr="email">
      <input type="email" name="email" id="email">
</div>

<input type="submit" name="send" value="send">

</form>
```

Сформированы необходимые массивы переменных в *HomeController.php:*

$_POST[‘name’];

$_POST[‘email’];

$_POST[‘send’];

3 Маршрутизация (*POST*-данные)

Для перехвата *POST*-данных использован метод Route::post

Прописан маршрут в роутах (*web.php*):

Route::post ('/home', 'HomeController@index');

4 Создание экшена *POSTINDEX* в контроллере *HOMECONTROLLER*

Код экшена *PostIndex* в *HomeController*.

PostIndex

```
Public function postIndex(){
  dd($_POST); // вывод на экран
}
```

5 Создание миграции, модели, *REQUEST*

Миграция, модель и request созданы в разделе 4.

Миграция: 2016_11_15_071254_Products;

Модель: *Product.php;*

Request: RequestProduct.php.

6 Работа с REQUEST в hoMECONTROLLER

Подключение request в контроллере HomeController:

Подключение *request*

```
use App\Http\Requests;
use Auth;
use App\Product;
```

Для перехвата данных из *request*-запроса, прописана функция *postIndex* в *HomeController.php*. Данный контроллер будет помещать *request*-запросы в таблицу *works* через модель, с помощью множественной вставки (метод *create*()).

Функция *postIndex* в *HomeController.php*

```
public function postIndex(Requests\RequestProduct $r)
{

   $r['user_id'] = Auth::user()->id;
   $r['cat_id'] = 0;
   $r['price'] = '-';
   $r['picture'] = '-';
   $r['picture_small'] = '-';
   Product::create($r->all());
   return redirect('home');
}
```

Добавление в таблицу нового поля user_id:

$r['user_id'] = Auth::user()->id;

8	user_id	int(11)			Нет	Нет

Рисунок 11 – Поле *user id* в базе данных

Функция Index в HomeController.*php*

```
public function index()
{

   $all = Product::where('user_id', Auth::user()->id)->orderBy('id', 'DESC')->paginate(10);

   return view('home')->with('all', $all);
}
```

7 Обработка кнопок «Удалить», «Редактировать»

7.1 Создание формы

Код *HomeBlade.php*

```
@if(isset($all))
   @foreach($all as $one)
      <h1 class="zag">{{$one->name}}</h1>
      <p class="times">{{$one->created_at}}</p>

      <div class="menu">
         <a href="{{asset('home/edit/'.$one->id)}}"
           class='btn btn-block btn-default'> // стиль кнопки
            Редактировать
         </a>
         <a href="{{asset('home/delete/'.$one->id)}}"
           class='btn btn-block btn-default'>
            Удалить
         </a>
      </div>
```

Asset() - создает *URL* для актива с использованием текущей схемы запроса (*HTTP* или *HTTPS*).

7.2 Маршрутизация

Прописаны следующие роуты в файле web.*php*:

Route::get('/home/delete/{id}', 'HomeController@getDelete'), Route::get('/home/edit/{id}', 'HomeController@getEdit').

7.3 Создание экшинов

Прописаны следующие функции для кнопок в HomeController.*php*.

Функция удаления товара по *id*

```
public function getDelete($id=null){
   Product::where('user_id',Auth::user()->id)->where('id',$id)->delete();
   return redirect('home');
}
```

Функция редактирования

```
public function getEdit($id){
   $one = Product::where('user_id',Auth::user()->id)->first();
}
```

7.4 Создание шаблона edit.blade.*php*

edit.blade.php

```
@if ($errors)
   @foreach($errors as $one)
      <div class="errors">{{$one}}</div>
      @endforeach
   @endif
<form action="home/edit/{{$one->id}}" method="post" enctype="multipart/fprm-data">
   {{scrf-field()}}
   <div class="form_inner">
      <label for="name"></label>
      <input name="name" id="name" value="{{$one->name}}">
   </div>
   <div class="form_inner">
      <label for="email"></label>
      <input type="email" name="email" id="email" value="{{$one->email}}">
   </div>
   <input type="submit" name="send" value="send">
</form>
```

7.5 Маршрутизация *action postEdit*

Прописан следующий *route:*

Route::post('/home/edit/{id}', 'HomeController@postEdit');

7.6 Создание *action postEdit*

HomeController

```
public function postEdit(ProductRequest $r, $id){
   $r['user_id']=Auth::user()->id;
   $r['picture']='-';
   Product::updateOrCreate($r->all());
   return redirect ('home');
}
```

8 Стиль кнопок

Код в style.css

```
.menu {
   width:200px;
   float:right;
}
.product{
   border-bottom:solid 1px lightgray;//ограничительная линия
   margin-bottom: 10px;
   padding: 10px;
}
```

6 QUICKADMIN БЫСТРОЕ АДМИНИСТРИРОВАНИЕ

Описание

Модель *User* с ролью администратора;

Seeds для загрузки таблицы с двумя пользователями. Один с ролью admin, другой – авторизированный пользователь;

Все необходимые действия с авторизированными пользователями: вход в систему, выход, сброс и запоминание пароля;

CRUD-действия с таблицей *users*;
Обработка статусов страниц 403, 404, 500.
Пошаговая установка:
composer create-project laravel/laravel ProjectName –prefer-dist;
composer require laraveldaily/quickadmin.
Прописан следующий класс в *config/app.php*
Laraveldaily\Quickadmin\QuickadminServiceProvider::class
php artisan quickadmin:install
Прописан следующий класс в *config/app.php*
Laraveldaily\Quickadmin\QuickadminServiceProvider::class
Прописан следующий класс *в App\Http\Kernel.php*
Laraveldaily\Quickadmin\QuickadminServiceProvider::class

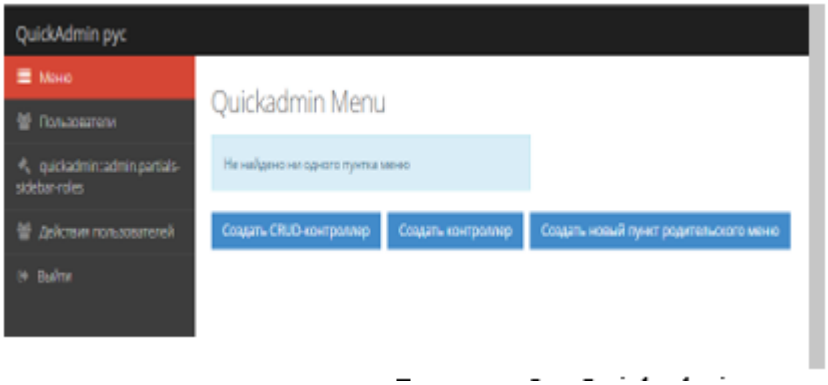

Рисунок 12 – Quickadmin

В разделе 5 описана установка и возможности *Quickadmin* – быстрого администрирования.

7 РЕЗУЛЬТАТ РАБОТЫ

В данном разделе представлены скриншоты готового интернет-магазина.

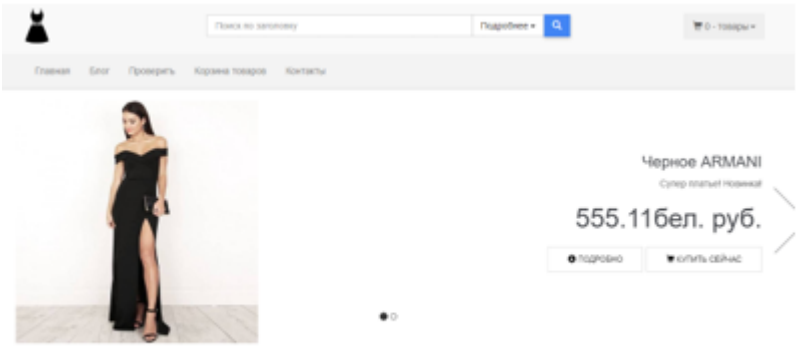

Рисунок 13 – Главная страница

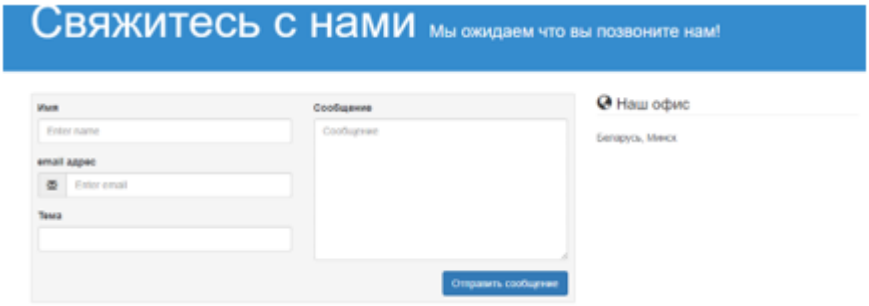

Рисунок 14 – Контакты

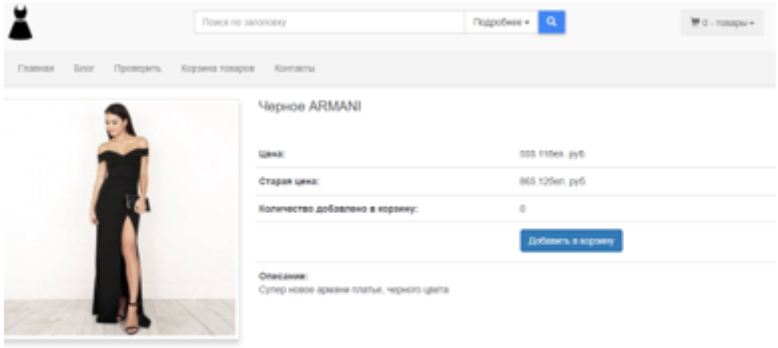

Рисунок 15 – подробный просмотр товара

Рисунок 16 – Добавление товара в корзину

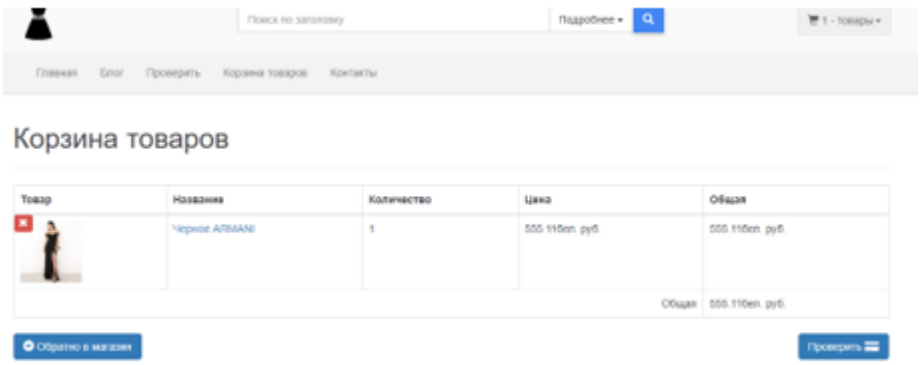

Рисунок 17 – Корзина товаров

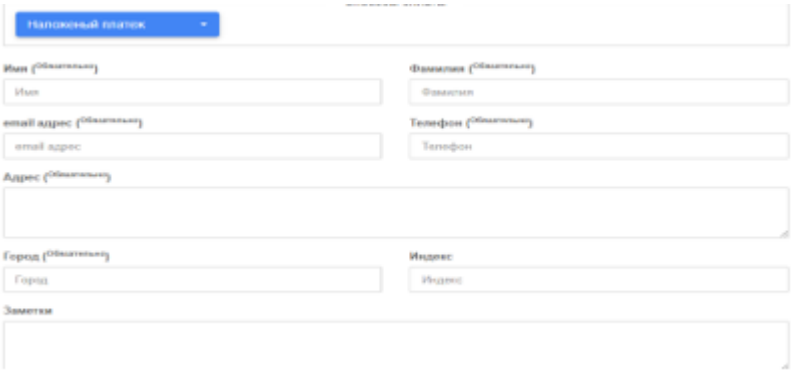

Рисунок 18 – заполнить форму для заказа товара

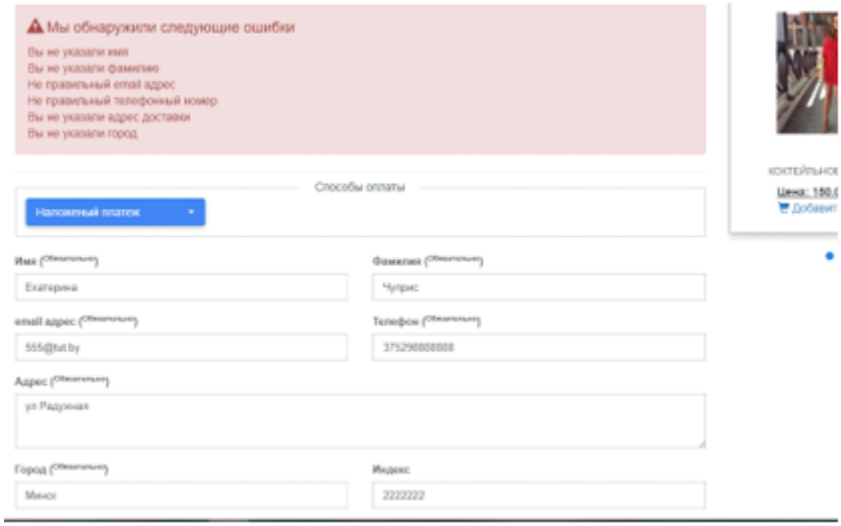

Рисунок 19 – обязательное заполнение полей формы

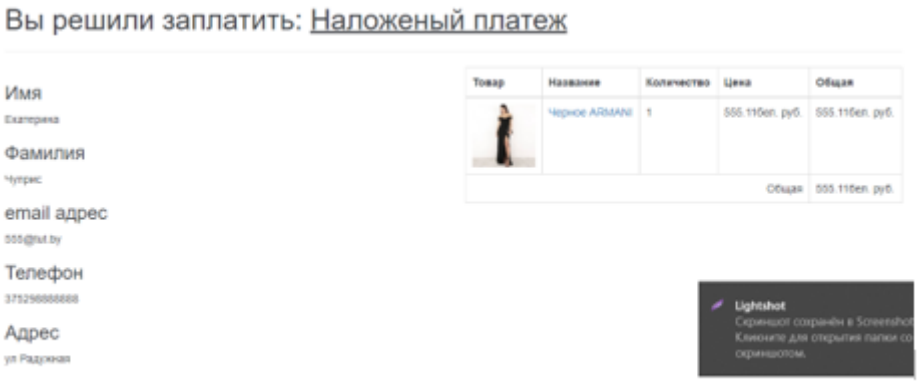

Рисунок 20 – обработанная форма заказа

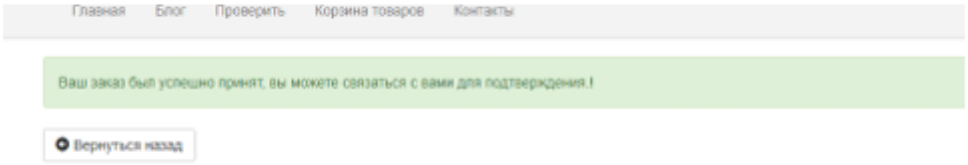

Рисунок 21 – заказ успешно принят

8 ПРИЛОЖЕНИЕ А Листинг программы

```
/**
 * CodeIgniter
 *
 * An open source application development framework for PHP
 *
 * This content is released under the MIT License (MIT)
 *
 * Copyright (c) 2014 - 2016, British Columbia Institute of Technology
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to deal
 * in the Software without restriction, including without limitation the rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
 * THE SOFTWARE.
 *
 * @package         CodeIgniter
```

```
 * @author		EllisLab Dev Team
 * @copyright	Copyright (c) 2008 - 2014, EllisLab, Inc. (https://ellislab.com/)
 * @copyright	Copyright (c) 2014 - 2016, British Columbia Institute of Technology (http://bcit.ca/)
 * @license	http://opensource.org/licenses/MIT	MIT License
 * @link	https://codeigniter.com
 * @since	Version 1.0.0
 * @filesource
 */

/*
 *---------------------------------------------------------------
 * APPLICATION ENVIRONMENT
 *---------------------------------------------------------------
 *
 * You can load different configurations depending on your
 * current environment. Setting the environment also influences
 * things like logging and error reporting.
 *
 * This can be set to anything, but default usage is:
 *
 *     development
 *     testing
 *     production
 *
 * NOTE: If you change these, also change the error_reporting() code below
 */
	define('ENVIRONMENT', 'development');

/*
 *---------------------------------------------------------------
 * ERROR REPORTING
 *---------------------------------------------------------------
 *
 * Different environments will require different levels of error reporting.
 * By default development will show errors but testing and live will hide them.
 */
switch (ENVIRONMENT)
{
	case 'development':
		error_reporting(-1);
```

```
                ini_set('display_errors', 1);
        break;

        case 'testing':
        case 'production':
                ini_set('display_errors', 0);
                if (version_compare(PHP_VERSION, '5.3', '>='))
                {
                        error_reporting(E_ALL & ~E_NOTICE & ~E_DEPRECATED &
~E_STRICT & ~E_USER_NOTICE & ~E_USER_DEPRECATED);
                }
                else
                {
                        error_reporting(E_ALL & ~E_NOTICE & ~E_STRICT &
~E_USER_NOTICE);
                }
        break;

        default:
                header('HTTP/1.1 503 Service Unavailable.', TRUE, 503);
                echo 'The application environment is not set correctly.';
                exit(1); // EXIT_ERROR
    }

    /*
     *---------------------------------------------------------------
     * SYSTEM FOLDER NAME
     *---------------------------------------------------------------
     *
     * This variable must contain the name of your "system" folder.
     * Include the path if the folder is not in the same directory
     * as this file.
     */
        $system_path = 'system';

    /*
     *---------------------------------------------------------------
     * APPLICATION FOLDER NAME
     *---------------------------------------------------------------
     *
     * If you want this front controller to use a different "application"
```

```
 * folder than the default one you can set its name here. The folder
 * can also be renamed or relocated anywhere on your server. If
 * you do, use a full server path. For more info please see the user guide:
 * https://codeigniter.com/user_guide/general/managing_apps.html
 *
 * NO TRAILING SLASH!
 */
	$application_folder = 'application';

/*
 *---------------------------------------------------------------
 * VIEW FOLDER NAME
 *---------------------------------------------------------------
 *
 * If you want to move the view folder out of the application
 * folder set the path to the folder here. The folder can be renamed
 * and relocated anywhere on your server. If blank, it will default
 * to the standard location inside your application folder. If you
 * do move this, use the full server path to this folder.
 *
 * NO TRAILING SLASH!
 */
	$view_folder = '';


/*
 * --------------------------------------------------------------------
 * DEFAULT CONTROLLER
 * --------------------------------------------------------------------
 *
 * Normally you will set your default controller in the routes.php file.
 * You can, however, force a custom routing by hard-coding a
 * specific controller class/function here. For most applications, you
 * WILL NOT set your routing here, but it's an option for those
 * special instances where you might want to override the standard
 * routing in a specific front controller that shares a common CI installation.
 *
 * IMPORTANT: If you set the routing here, NO OTHER controller will be
 * callable. In essence, this preference limits your application to ONE
 * specific controller. Leave the function name blank if you need
 * to call functions dynamically via the URI.
```

```
 *
 * Un-comment the $routing array below to use this feature
 */
	// The directory name, relative to the "controllers" folder.  Leave blank
	// if your controller is not in a sub-folder within the "controllers" folder
	// $routing['directory'] = '';

	// The controller class file name.  Example:  mycontroller
	// $routing['controller'] = '';

	// The controller function you wish to be called.
	// $routing['function']		= '';


/*
 * -------------------------------------------------------------------
 *  CUSTOM CONFIG VALUES
 * -------------------------------------------------------------------
 *
 * The $assign_to_config array below will be passed dynamically to the
 * config class when initialized. This allows you to set custom config
 * items or override any default config values found in the config.php file.
 * This can be handy as it permits you to share one application between
 * multiple front controller files, with each file containing different
 * config values.
 *
 * Un-comment the $assign_to_config array below to use this feature
 */
	// $assign_to_config['name_of_config_item'] = 'value of config item';



// --------------------------------------------------------------------
// END OF USER CONFIGURABLE SETTINGS.  DO NOT EDIT BELOW THIS LINE
// --------------------------------------------------------------------

/*
 * --------------------------------------------------------------
 *  Resolve the system path for increased reliability
 * --------------------------------------------------------------
 */
```

```
	// Set the current directory correctly for CLI requests
	if (defined('STDIN'))
	{
		chdir(dirname(__FILE__));
	}

	if (($_temp = realpath($system_path)) !== FALSE)
	{
		$system_path = $_temp.'/';
	}
	else
	{
		// Ensure there's a trailing slash
		$system_path = rtrim($system_path, '/').'/';
	}

	// Is the system path correct?
	if ( ! is_dir($system_path))
	{
		header('HTTP/1.1 503 Service Unavailable.', TRUE, 503);
		echo 'Your system folder path does not appear to be set correctly. Please open the following file and correct this: '.pathinfo(__FILE__, PATHINFO_BASENAME);
		exit(3); // EXIT_CONFIG
	}

/*
 * --------------------------------------------------------------------
 *  Now that we know the path, set the main path constants
 * --------------------------------------------------------------------
 */
	// The name of THIS file
	define('SELF', pathinfo(__FILE__, PATHINFO_BASENAME));

	// Path to the system folder
	define('BASEPATH', str_replace('\\', '/', $system_path));

	// Path to the front controller (this file)
	define('FCPATH', dirname(__FILE__).'/');

	// Name of the "system folder"
```

```
define('SYSDIR', trim(strrchr(trim(BASEPATH, '/'), '/'), '/'));

// The path to the "application" folder
if (is_dir($application_folder))
{
	if (($_temp = realpath($application_folder)) !== FALSE)
	{
		$application_folder = $_temp;
	}

	define('APPPATH', $application_folder.DIRECTORY_SEPARATOR);
}
else
{
	if ( ! is_dir(BASEPATH.$application_folder.DIRECTORY_SEPARATOR))
	{
		header('HTTP/1.1 503 Service Unavailable.', TRUE, 503);
		    echo 'Your application folder path does not appear to be set correctly. Please open the following file and correct this: '.SELF;
		exit(3); // EXIT_CONFIG
	}

	define('APPPATH', BASEPATH.$application_folder.DIRECTORY_SEPARATOR);
}

// The path to the "views" folder
if ( ! is_dir($view_folder))
{
	if ( ! empty($view_folder) && is_dir(APPPATH.$view_folder.DIRECTORY_SEPARATOR))
	{
		$view_folder = APPPATH.$view_folder;
	}
	elseif ( ! is_dir(APPPATH.'views'.DIRECTORY_SEPARATOR))
	{
		header('HTTP/1.1 503 Service Unavailable.', TRUE, 503);
		  echo 'Your view folder path does not appear to be set correctly. Please open the following file and correct this: '.SELF;
		exit(3); // EXIT_CONFIG
	}
	else
```

```
		{
			$view_folder = APPPATH.'views';
		}
	}

	if (($_temp = realpath($view_folder)) !== FALSE)
	{
		$view_folder = $_temp.DIRECTORY_SEPARATOR;
	}
	else
	{
		$view_folder = rtrim($view_folder, '/\\').DIRECTORY_SEPARATOR;
	}

	define('VIEWPATH', $view_folder);

/*
 * --------------------------------------------------------------------
 * LOAD THE BOOTSTRAP FILE
 * --------------------------------------------------------------------
 *
 * And away we go...
 */
require_once BASEPATH.'core/CodeIgniter.php';
```

9 ПРИЛОЖЕНИЕ Б Графический материал

Рисунок 22 – Диаграмма вариантов использования

Рисунок 23 – Диаграмма последовательности

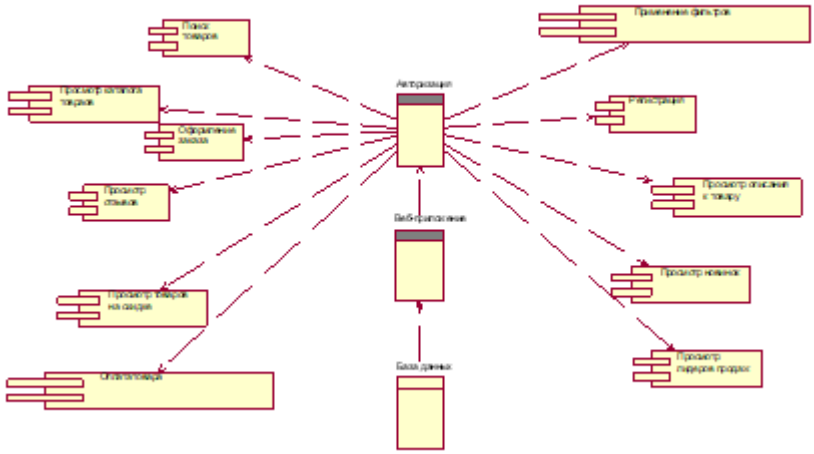

Рисунок 24 – Диаграмма компонентов

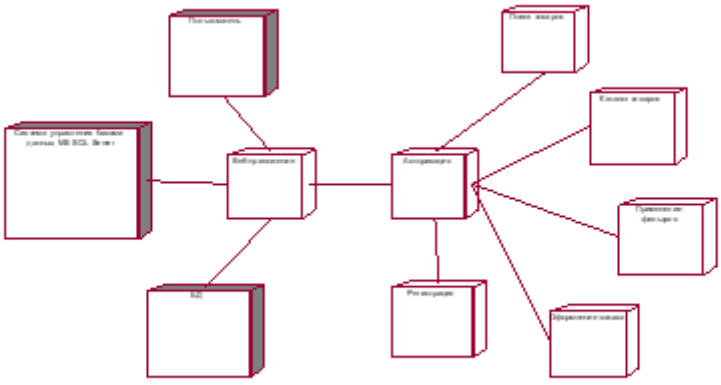

Рисунок 25 – Диаграмма развертывания

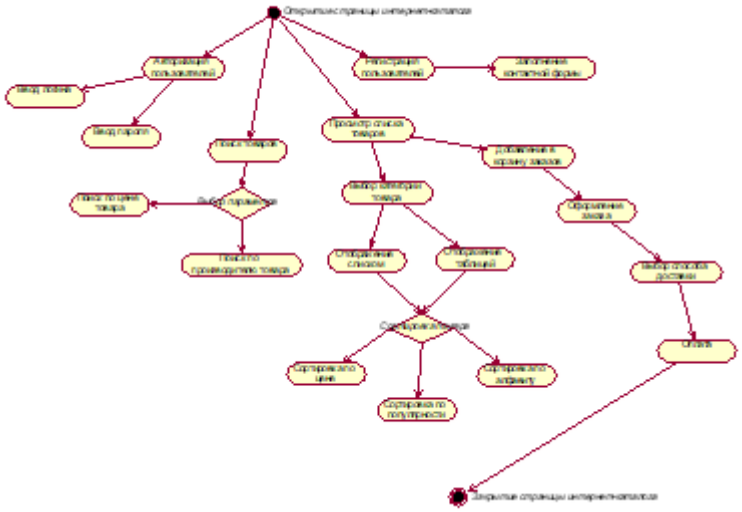

Рисунок 26 – Диаграмма состояний

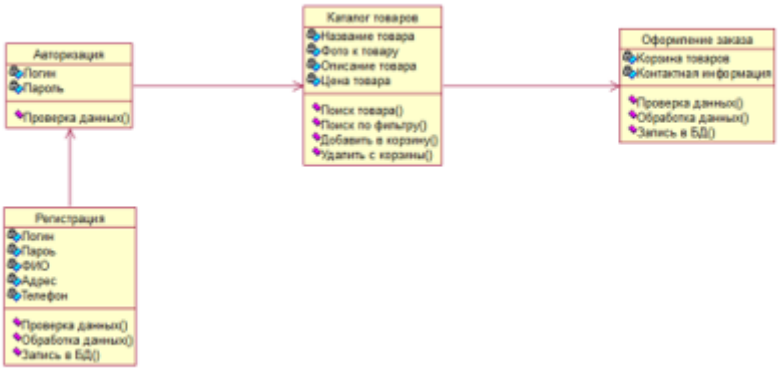

Рисунок 27 – Диаграмма классов

10 СПИСОК ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ

1. Браун, М. HTML в подлиннике / М. Браун, Д. Ханикатт. – Спб. : Издательство «БХВ-Петербург», 2002. – 1048с.
2. Белунцов, В. Новейший самоучитель по разработке Web-страниц / В. Белунцов. – Москва. : Издательство «Десс-ком», 2000. – 448с.
3. Богомолов, В. И. Основы Web-мастерства. Учебный куср / В. И. Богомолов. – Спб. : Питер, 2001. – 352с.
4. Божко, А. Dreamweaver 4. Базовый курс / А. Божко. – Москва. : Издательство «Десс-ком», 2001. – 448с.
5. Гончаров, А. Самоучитель по HTML / А. Гончаров. – Спб. : Питер, 2002. – 240с.
6. Тиге, Д. К. DHTML и Internet / Д. К. Тиге. – М. : Издательство «НТ Пресс», 2005. – 520с.
7. Дубаков, М. А. Создание Web-страниц: искусство верстки / М. А. Дубаков. – Мн. : Новое знание, 2004. – 287с.
8. Дубаков, М. А. Веб-мастеринг средствами CSS / М. А. Дубаков. – СПб. : Издательство «БХВ-Петербург», 2002. – 544 с.
9. Дунаев, В. HTML, скрипты и стили / В. Дунаев. – Спб. : Издательство «БХВ- Петербург», 2008.

– 1024с.
10. Дунаев, В. HTML, шаблоны и каскадные стили / В. Дунаев. – Спб. : Издательство «БХВ-Петербург», 2010. – 504с.
11. Кастро, Э. Создание Web-страниц с помощью HTML / Э. Кастро. – М. : Издательство «НТ Пресс», 2005. – 144с.
12. Коржинский, С. Н. Настольная книга Web-мастера: эффективное применение HTML, CSS и Javascript / С. Н. Коржинский. – М. : Издательский торговый дом «Кнорус», 2000. – 320с.
13. Мальчуков, Н. Н. HTML и CSS. Самоучитель / Н. Н. Мальчуков. – М. : Издательский дом «Вильямс», 2008. – 396 с.

Заключение

В данном курсовом проекте был разработан интернет-магазин женских платьев. 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 перечнем продаваемых товаров, информацией о данных товарах, а также с контактной информацией магазина. Создан дизайн-макет веб-сайта, который совпадает с готовым проектом. Установлен Git и TortoiseGit. Git – мощная и сложная распределённая 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. Проект создан на основе фреймворка Laravel. Он берёт на себя аутентификацию, роутинг, работу с сессиями, кеширование, внедрение зависимостей и многое другое. Также организован модуль авторизации с помощью данного фреймворка. Для решения распространённых проблем, возникающих при разработке программного обеспечения используется – шаблон проектирования. В данной пояснительной записки показано на примерах создания CRUD. Установлен Quickadmin для быстрого администрирования. В целом предлагаемый веб-сайт позволит значительно повысить эффективность деятельности магазина, а также девушкам и женщинам приобрести понравившиеся платье, что способствует привлечению новых покупателей и как следствие получение наибольшей выгоды.

Список использованных источников

Приложения

1. [электронный документ] 5ed049bbd11da_1.png