Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет Повышение квалификации и переподготовка

Кафедра проектирования информационных компьютерных систем

Дисциплина "Маршрутизация, контроллеры, модели и элементы представления необходимые для создания web-приложения"

К защите допустить:
Руководитель курсовой работы
_____
12.07.2025

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

на тему

Разработка web-приложения на Laravel

БГУИР КР 1-39 03 02 № 43 ПЗ

Студент (подпись студента) И.И.Иванов

Курсовая работа
представлена на проверку
12.07.2025
_____
(подпись студента)

Минск 2025

Реферат

БГУИР КР 1-39 03 02 № 43 ПЗ, гр. 90421

И.И.Иванов, Разработка web-приложения на Laravel, Минск: БГУИР - 2025.

Пояснительная записка 186403 с., 4 рис., 1 табл.

Ключевые слова:

Предмет Маршрутизация, контроллеры, модели и элементы представления необходимые для создания web-приложения,

Содержание

Введение
1 Описание проекта
2 Технологии и инструментарий
3 Система контроля версий
4 Маршрутизация
5 Контроллер
6 Модель
7 Миграции
8 Удаление public из url-запросов адресной строки в Laravel
9 Авторизация
Заключение
Список использованных источников
Приложения

Введение

Курсовая работа выполнена на тему «Сайта-визитка с каталогом товаров для мебельного магазина».

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), размещенный на веб-сервере и имеющем индивидуальный адрес.

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. Получение наибольшей выгоды является основной задачей любого предприятия и предпринимателя, таким образом, внедрение сайта-визитки способствует развитию бизнеса.

В курсовой работе поставлена цель: разработка сайта визитки.

Для достижения поставленной цели необходимо решить следующие задачи:

Описание проекта;

Выбор технологий и инструментария;

Создание дизайн-макета веб-сайта;

Выбор архитектурного шаблона проектирования;

Программирование сайта;

Наполнение сайта информацией;

Размещение на открытый репозиторий.

1 Описание проекта

Темой проекта является: сайт-визитка с каталогом товаров для мебельного магазина. Данный веб-сайт создается для того чтобы познакомить пользователя с перечнем продаваемых товаров, информацией о данных товарах, а также с контактной информацией магазина.

Дизайн-макет веб-сайта

В качестве цветового решения выбраны светлые тана для того чтобы пользователь концентрировал внимание на продаваемые товары.

Далее представим структуру страниц будущего сайта.

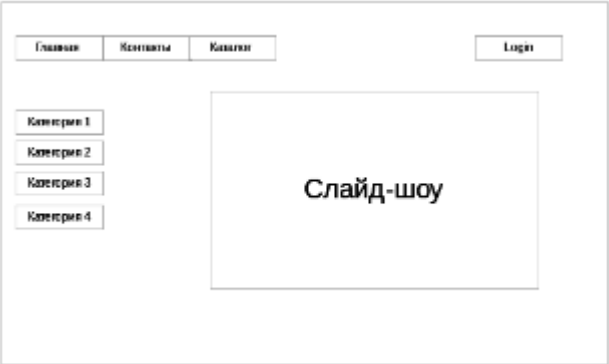

Рисунок 1 – Структура главной страницы сайта

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

Кликая на кнопку *Login* пользователь попадает в раздел добавления товаров.

2 Технологии и инструментарий

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.

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.

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/viktoriya-gr/website_of_shop/commit -добавление

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

Ссылка на удаленный репозиторий:
https://github.com/viktoriya-gr/website_of_shop

4 Маршрутизация

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 ре-гистрируете в маршрутизаторе (Router). Второй параметр — функ-ция, содержащая логику для это-го маршрута. Маршруты регистрируются без задания ведущего слеша (/) — единственное исключение — корневой маршрут, который со-стоит из одного слеша ('/').

Используемые маршруты в данной курсовой работе:

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');

5 Контроллер

Контроллеры хранятся в папке app/http/controllers/. Этот путь, в свою очередь определен 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.BaseControllerрасширяет 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*.

1.Определение маршрута для контроллера с помощью метода get:

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

2.С помощью метода cotroller:

```
Route::controller(

'cabinet' => 'CabinetController',

);
```

3. 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';
 }
}
```

Рассмотрим для создания каталога:

```
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);
 }
```

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

6 Модель

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' => '',

),
```

В дальнейшем, все запросы к базе данных ведутся через модель. Модель является связующим звеном между приложением и конкретной таблицей базы данных. Запросы в таблицу ведутся через модель с использованием конструктора запросов.

Все модели должны помещаться в этом каталоге и именоваться таким же образом. Например, в данной лабораторной работе создали модель для каталога, файл модели называться app /Catalog.*php*.

Модель Catalog

```
namespace App;
use Illuminate\Database\Eloquent\Model;
class Catalog extends Model
{
 public $table = "catalog";
}
```

7 Миграции

Миграции базы данных являются весьма полезны для любого проекта, особенно для проектов с несколькими разработчиками, позволяя иметь последнюю версию базы данных у всех разработчиков. В *Laravel* для этого достатчно 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() ее нужно удалить.

Допишем действия up() и down()

```
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.
```

8 Удаление public из url-запросов адресной строки в Laravel

Одной из первых задач, с которыми сталкивается разработчик на Laravel - это избавление от ключевого слова public в запросах адресной строки.

Для этого в корне проекта создадим файл .htaccess со следующим содержимым.

Перенаправление в папку public, файл .htaccess.

```
RewriteEngine On
RewriteRule ^(.*)$ public/$1 [L]
```

9 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 изменить 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 еще один контроллер, HomeController, куда будет перенаправляться пользователь после успешной авторизации.

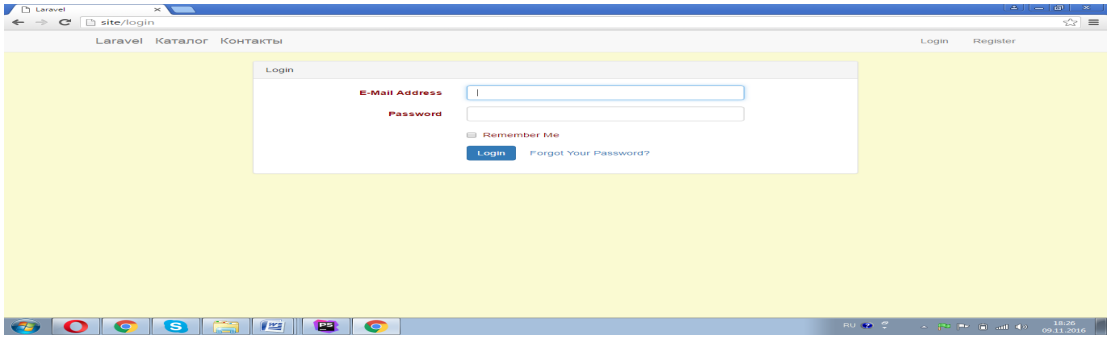

Рисунок 3 – Авторизация

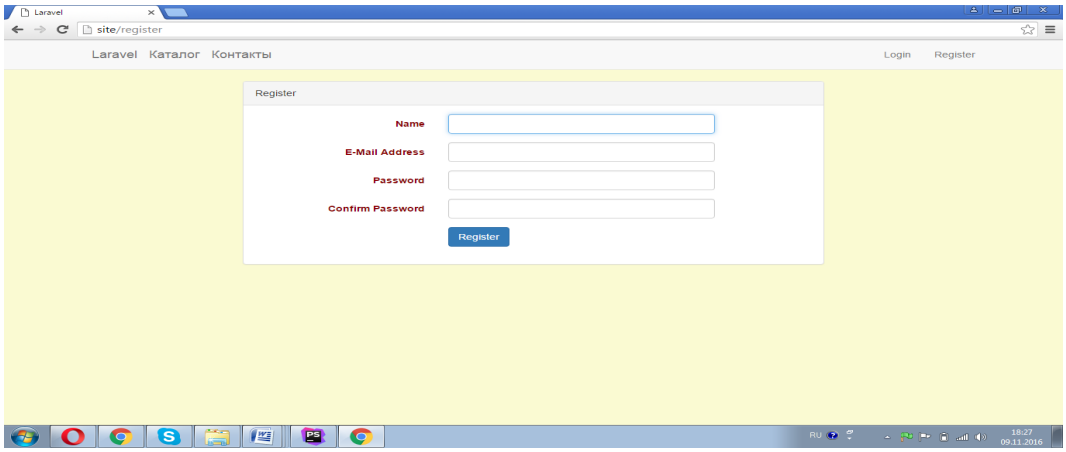

Рисунок 3 – Регистрация

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()) {
```

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');
}
```

Заключение

Список использованных источников

Приложения

1. [more] 5c18ada96e544_Собеседование.pdf