**Министерство образования и науки Российской Федерации**

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«Владимирский государственный университет**

**имени Александра Григорьевича и Николая Григорьевича Столетовых»**

(ВлГУ)

Кафедра информационных систем и программной инженерии

Технологии разработки веб-приложений

Методические указания к лабораторным работам

**«Знакомство с фреймворком Laravel»**

Владимир 2020

**Цель работы**

Познакомиться со структурой приложений Laravel и разработать приложение ”Корзина покупателя”.

**Теоретические сведения**

Laravel — бесплатный веб-фреймворк с открытым кодом, предназначенный для разработки с использованием архитектурной модели MVC. Laravel выпущен под лицензией MIT.

Исходный код проекта размещается на GitHub. В результате опроса sitepoint.com в декабре 2013 года о самых популярных PHP-фреймворках Laravel занял место самого многообещающего проекта на 2014 год.

**Ключевые особенности, лежащие в основе архитектуры Laravel:**

* Пакеты (англ. packages) — позволяют создавать и подключать модули в формате Composer к приложению на Laravel. Многие дополнительные возможности уже доступны в виде таких модулей.
* Eloquent ORM — реализация шаблона проектирования ActiveRecord на PHP. Позволяет строго определить отношения между объектами базы данных. Стандартный для Laravel построитель запросов Fluent поддерживается ядром Eloquent.
* Логика приложения — часть разрабатываемого приложения, объявленная либо при помощи контроллеров, либо маршрутов (функций-замыканий). Синтаксис объявлений похож на синтаксис, используемый в каркасе Sinatra.
* Обратная маршрутизация связывает между собой генерируемые приложением ссылки и маршруты, позволяя изменять последние с автоматическим обновлением связанных ссылок. При создании ссылок с помощью именованных маршрутов Laravel автоматически генерирует конечные URL.
* REST-контроллеры — дополнительный слой для разделения логики обработки GET- и POST-запросов HTTP.
* Автозагрузка классов — механизм автоматической загрузки классов PHP без необходимости подключать файлы их определений в include. Загрузка по требованию предотвращает загрузку ненужных компонентов; загружаются только те из них, которые действительно используются.
* Составители представлений (англ. view composers) — блоки кода, которые выполняются при генерации представления (шаблона).
* Инверсия управления (англ. Inversion of Control) — позволяет получать экземпляры объектов по принципу обратного управления. Также может использоваться для создания и получения объектов-одиночек (англ. singleton).
* Миграции — система управления версиями для баз данных. Позволяет связывать изменения в коде приложения с изменениями, которые требуется внести в структуру БД, что упрощает развёртывание и обновление приложения.
* Модульное тестирование (юнит-тесты) — играет очень большую роль в Laravel, который сам по себе содержит большое число тестов для предотвращения регрессий (ошибок вследствие обновления кода или исправления других ошибок).
* Страничный вывод (англ. pagination) — упрощает генерацию страниц, заменяя различные способы решения этой задачи единым механизмом, встроенным в Laravel.

**Выполнение работы**

В рамках данной работы будет разработано приложение для работы с продуктами в online-магазине.

Для начала необходимо установить необходимые средства разработки и сгенерировать проект.

Для удобства работы с зависимостями в проекте будем использовать Composer. Composer ([getcomposer.org](http://getcomposer.org/)) — это популярный менеджер зависимостей для PHP.

Документация по установке Composer доступна на данной странице <https://getcomposer.org/doc/00-intro.md>

После установки Composer можно сгенерировать Laravel приложение

composer create-project --prefer-dist laravel/laravel lab-laravel

Для того, чтобы проверить правильность генерирования проекта необходимо выполнить следующие команды

cd lab-laravel

php artisan -V

Если проект сгенерирован успешно в консоль выведется *Laravel Framework 7.0.8*

На следующем шаге, используя знания полученные в предыдущих лабораторных работах необходимо самостоятельно создать БД (любая СУБД) для использования в проекте.

Далее необходимо подключить эту БД к проекту, для этого необходимо отредактировать переменные окружения в файле **.env** в корне проекта. Необходимо добавить новые строки со своими значениями

DB\_CONNECTION=mysql

DB\_HOST=127.0.0.1

DB\_PORT=3306

DB\_DATABASE=lab\_laravel

DB\_USERNAME=<your db username>

DB\_PASSWORD=<your db password>

Следующий шаг - описание сущностей в проекте. Для того, чтобы сгенерировать код и миграцию для БД необходимо выполнить следующую команду

php artisan make:model Product -m

Эта команда создаст модель Product в проекте и сгенерирует SQL скрипт для создания соответствующей таблицы в БД.

Чтобы добавить поля для модели Product надо открыть сгенерированный файл миграции из папки database/migrations и отредактировать.

После строки $table->id(); надо добавить описание остальных полей полей

$table->string("name", 255)->nullable();

$table->text("description")->nullable();

$table->string("photo", 255)->nullable();

$table->decimal("price", 6, 2);

Для запуска миграции выполняем команду

php artisan migrate

Для того, чтобы заполнить БД начальными значениями надо создать скрипт

php artisan make:seed ProductsSeeder

Отредактируем сгенерированный файл

Внутрь функции run() надо добавить команды для создания записей в БД. Пример одной записи.

DB::table('products')->insert([

'name' => 'LG',

'description' => 'Protection Corning Gorilla Glass 4. MISC Colors Space Black, Luxe White, Modern Beige, Ocean Blue, Opal Blue.',

'photo' => 'https://i.ebayimg.com/00/s/NjQxWDQyNA==/z/VDoAAOSwgk1XF2oo/$\_35.JPG?set\_id=89040003C1',

'price' => 39999.00

]);

**Самостоятельно** добавить еще **5** продуктов в *ProductsSeeder.*

Для того, чтобы создать записи в БД надо запустить процесс сидирования. Для этого надо выполнить 2 команды.

composer dump-autoload

и

php artisan db:seed --class=ProductsSeeder

После того, что БД наполнена данными можно переходить к реализации логики.

Создаем контроллер

php artisan make:controller ProductsController

Для вывода продуктов на экран добавим функцию внутрь описания класса *ProductsController*

use App\Product;

class ProductsController extends Controller

{

public function index()

{

$products = Product::all();

return view('products', compact('products'));

}

}

Данная функция описывает выбор всех записей из таблицы Products в БД и присваивание в переменную для использования в шаблоне.

Для определения шаблонов сначала создадим *layout.blade.php* в папке resources/views

<!DOCTYPE html>

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />

<title>@yield('title')</title>

<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css">

</head>

<body>

<div class="container page">

@yield('content')

</div>

@yield('scripts')

</body>

</html>

Данный файл является базовым для всех остальных шаблонов. В нем подключается css фреймворк bootstrap для удобства работы с разметкой.

Для непосредственного вывода продуктов создадим файл *products.blade.php* с содержимым

@extends('layout')

@section('title', 'Products')

@section('content')

<div class="container products">

<div class="row">

@foreach($products as $product)

<div class="col-xs-18 col-sm-6 col-md-3">

<div class="thumbnail">

<img src="{{ $product->photo }}" width="500" height="300">

<div class="caption">

<h4>{{ $product->name }}</h4>

<p>{{ str\_limit(strtolower($product->description), 50) }}</p>

<p><strong>Price: </strong> {{ $product->price }}$</p>

<p class="btn-holder"><a href="{{ url('add-to-cart/'.$product->id) }}" class="btn btn-warning btn-block text-center" role="button">Add to cart</a> </p>

</div>

</div>

</div>

@endforeach

</div>

</div>

@endsection

В этом шаблоне в цикле выводятся все продукты из массива *$products,* который был определен в *ProductsController.*

Для того, чтобы страница с продуктами открывалась по умолчанию надо поменять файл routes/web.ph и заменить там путь к `/` на

Route::get('/', 'ProductsController@index');

После этого надо зайти запустить встроенный сервер php или настроить apache/denwer на каталог /public в проекте

Для запуска встроенного сервера

php artisan serve

Результат можно посмотреть по пути [http://localhost:8000](http://localhost:8000/)

**Задания**

1. Воспроизвести шаги из методички и показать работоспособность.
2. Реализовать 2 CRUD операции с продуктами на выбор.
3. Реализовать логику работы корзины покупателя с использованием объекта session (при перезагрузке страницы данные не теряются).