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

Факультет компьютерного проектирования

Кафедра проектирования информационно-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 агентства интернет-маркетинга "EWORLD"

№ 131 ПЗ

Студент (подпись студента) В.К. Лагун

Курсовая работа
представлена на проверку
10.07.2025
_____
(подпись студента)

2025

Реферат

№ 131 ПЗ, гр.

В.К. Лагун, Разработка сайта агентства интернет-маркетинга "EWORLD", Минск: - 2025.

Пояснительная записка 141953 с., 24 рис., 4 табл.

Ключевые слова: агентство, маркетинг,

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

* Интернет-маркетинг – актуальный и современный подход к развитию бизнеса, это часть классического маркетинга. Он включает систему мероприятий по успешному продвижению компании, её товаров и услуг в сети Интернет, ориентированную на потребности конкретного сегмента рынка. Исследование потенциальных возможностей рынка позволит идентифицировать целевую аудиторию в интернете, изучить конкурентов (их специфику маркетингово-технических сторон сайтов и средств рекламы) и составить целостную картину конъюнктуры интерент-рынка в нише. Когда-то была дилемма – как сделать сайт, потом думали – как его продвинуть, последние пару лет в моде аналитика. Сейчас актуальна практика использования элементов классического маркетинга в интернете, так как интернет-маркетинг, это такой же маркетинг, только в digital-среде. *

* Internet marketing is an up-to-date and modern approach to business development, it is part of classic marketing. It includes a system of measures for the successful promotion of the company, its products and services on the Internet, focused on the needs of a specific market segment. Research of potential market opportunities will allow you to identify the target audience on the Internet, study competitors (their specific marketing and technical aspects of sites and advertising tools) and create a complete picture of the situation of the Internet market in the niche. Once there was a dilemma – how to make a site, then we thought-how to promote it, the last couple of years in the fashion of Analytics. Now the practice of using elements of classical marketing on the Internet is relevant, since Internet marketing is the same marketing, only in the digital environment. *

Содержание

Введение

Курсовая работа выполнена на тему « Разработка сайта агенства интернет-маркетинга "EWORLD"». 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, услугах, прайс-листы, контактные данные и другую полезную информацию, что способствует привлечению новых клиентов 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 агенства интернет-маркетинга "EWORLD". 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 хостинг.

1 ОПИСАНИЕ ПРОЕКТА

Темой проекта является: Разработка сайта агенства интернет-маркетинга "EWORLD". Данный веб-сайт создается с целью информирования клиентов, об услугах, нвых акций, проведенных работ и т.д.

1.1 Дизайн-макет веб-сервера

В качестве цветового решения выбраны теплые тона и выполнено в стиле минимализма для того чтобы пользователь концентрировал внимание на нужной информации.

Далее представим структуру страниц будущего сайта.

Рисунок 1 – Структура главной страницы сайта

Главная страница содержит область с главным меню, в которой представлены новости, акции, портфолио, контакты.

На рисунке 2 показана структура страницы услуги, на котором представлены все услуги.

Кликая на нужную услугу (услуга 1, услуга 2, и т.д.), пользователь отобразит все описание услуги оказываемой компанией EWORLD.

Рисунок 2 – Структура страницы услуги

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.

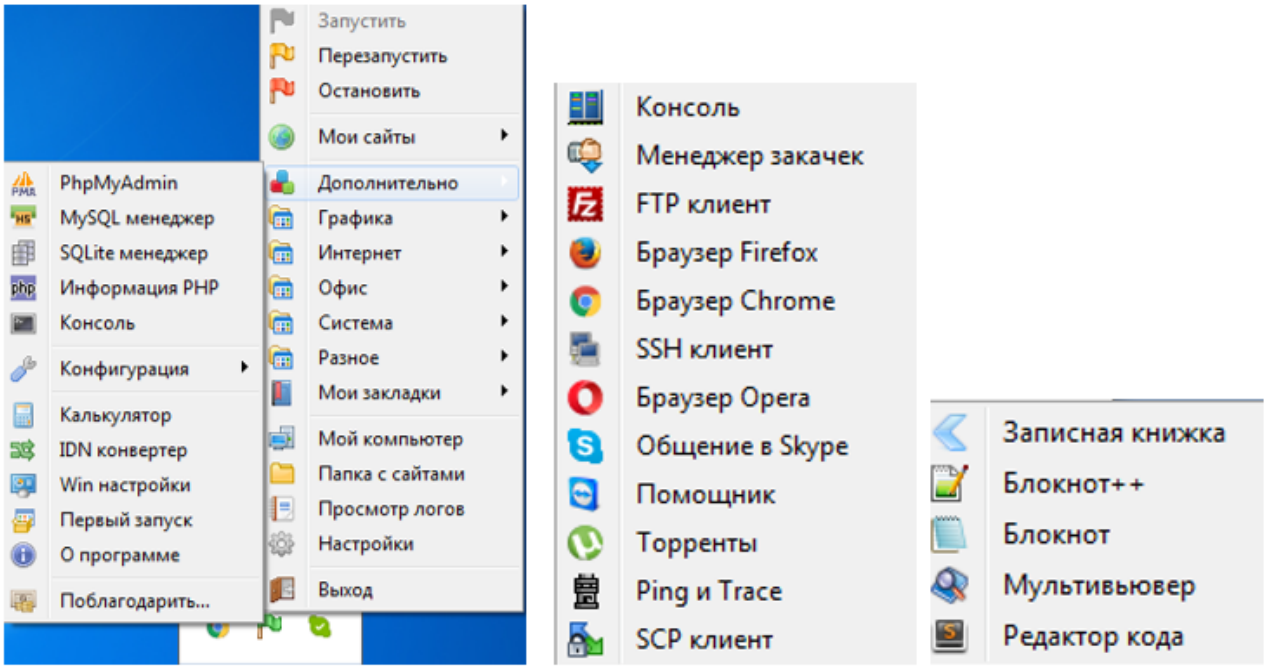


Рисунок 3- *Open Server*

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

1.3 Модели представления системы

Диаграмма вариантов использования *UML*.

UML – это язык для визуализации, специфицирования, конструирования и документирования артефактов программных систем.

Язык представляет словарь и правила комбинирования, входящих в него слов в целях коммуникации. Язык моделирования – это язык, словарь и правила которого сосредоточены на концептуальном и физическом представлении системы. *UML* – стандартное средство представления «чертежей» программного обеспечения.

Определив цели моделирования программной системы, приведем диаграмму вариантов использования для наглядной демонстрации поведения будущей системы на рисунке 4.

В диаграмме представлен контекст системы автоматизации добавления контента администратором в данную систему. Вы видите пользователей, которые подразделяются на две группы:

– администратор;

– пользователь.

В данном контексте они представлены в роли администратора и пользователей, один из которых взаимодействует с системой с ограничениями, а второй имеет преимущества.

Система имеет следующие варианты использования:

– просмотр контента;

– добавление контента;

– изменение контента;

– авторизация в администраторской панели;

– изменение личных данных администратора;

– добавление новых ролей администраторов (журналист, редактор, главный редактор и т.д.);

– добавление новых администраторов;

– изменение личных данных администраторов;

– изменение роли администраторов;

Для наших двух ролей мы определили взаимодействие с системой. Итак, пользователь имеет возможность воспользоваться только просмотром контента на сайте. Администратор взаимодействует с системой через поведение «Авторизация» которая доступна по специальной ссылке http://eworld.by/*admin*. И только после применения этого варианта использования он получает доступ к остальным моделям поведения:

Мы определили структуру вариантов использования. Теперь опишем поведение некоторых из них.

Добавления нового администратора. Главный администратор системы создает нового администратора и указывает роль на информационном портале путешественника. После создания нового администратора, он может войти в свой кабинет путем авторизации. В своем кабинете он может создавать, редактировать и удалять контент согласно с выданными правами, изменить личные данные, логин и пароль, а также изменять, удалять фотографию профиля. Авторизация администратора. Для входа в кабинет администратора необходимо ввести логин и пароль.

Изменение личных данных администратора. Предоставляется возможность менять свои личные данные, а также предусматривается изменение логина и пароля, и фотографию профиля.

Итак, смоделировав диаграмму вариантов использования, мы получили наглядное представление о нашей системе. Были определены действующие лица модели (администратор и пользователь), набор вариантов использования и их связи. Описав все модели поведения системы и пути, мы определили ряд требований, предъявляемых к нашему порталу.

Рисунок 4 – Диаграмма вариантов использования

Диаграмма последовательности

Диаграмма последовательности – это диаграмма взаимодействия, которая подчеркивает временной порядок сообщений. Изображается как таблица, в которой представлены объекты, расположенные вдоль оси X, и сообщения, упорядоченные по ходу времени – вдоль оси Y.

Приведем описание диаграммы последовательности, представленной на рисунке 5. Данная диаграмма описывает процесс добавления новости пользователем на сайт. Итак, объектом, инициирующим взаимодействие, является пользователь. Зависящими объектами будут:

– браузер;

– сервер;

– база данных.

Все эти объекты расположены горизонтально. Вертикальные пунктирные линии, которые есть у каждого объекта, символизируют существование объекта в течение некоторого периода времени. Теперь перейдем к основному содержимому диаграммы последовательности – сообщениям. Они изображаются стрелками, направленными от одной линии жизни к другой. Стрелка указывает на приемник сообщения.

Инициатором взаимодействия, как и говорилось ранее, является администратор. На странице представлена форма заполнения новости, где администратор пишет новостную статью, которую собирается разместить на портале. Имеется заголовок, краткое описание, подробное описание и выбор фотографии. Таким образом, от администратора браузер получает данные. Далее браузер передает эти данные на сервер, сервер обрабатывает полученные данные и сообщает об этом базе данных. База данных получает данные и заполняет в соответствии с введенными данными и добавляет новую запись. Далее база данных возвращает добавленную запись на сервер. Сервер передает эту запись в браузер, и администратор видит размещенную новость на сервере и в списке всех новостей.

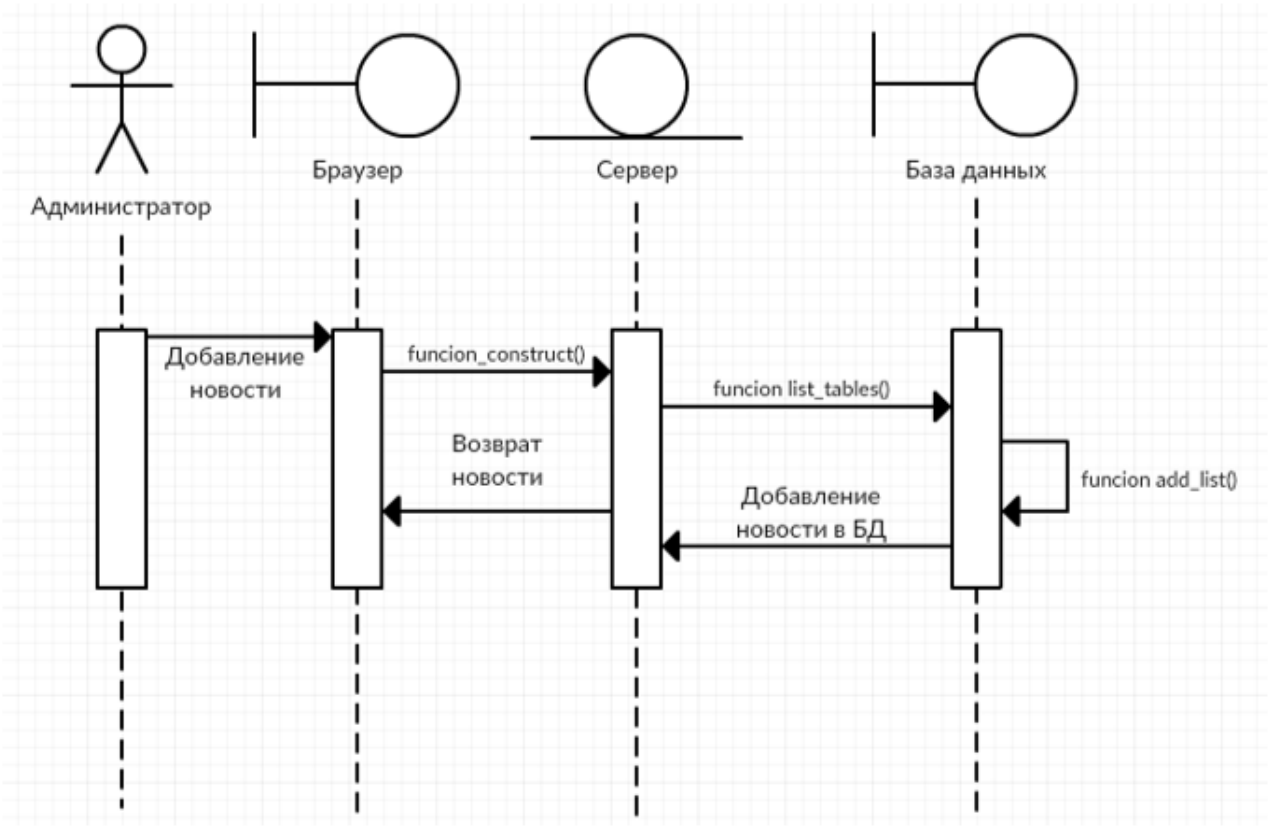


Рисунок 5 – Диаграмма последовательности

Диаграмма состояний

Диаграммы состояний – это один из пяти видов диаграмм UML, предназначенных для моделирования динамических аспектов поведения систем. Диаграмма состояний показывает конечный автомат. И диаграммы деятельности, и диаграммы состояний подходят для моделирования жизненного цикла объекта. Однако в то время, как диаграмма деятельности демонстрирует поток управления от одной деятельности к другой через множество объектов, диаграмма состояний отображает поток управления от состояния к состоянию внутри отдельного объекта.

Диаграмма состояний приведена на рисунке 6. Сначала перечислим все состояния новости на сервере:

– несформированный новость;

– новость в режиме добавления;

– шаблон новости;

– заполненная форма для размещения новости;

– шаблон и данные в режиме обработки;

– обработанные данные;

– сформированная новость в разделе «Новости».

Начальным состоянием является состояние «несформированная новость». Для перехода в состояние «новость в режиме добавления» необходимо совершить переход «выбор текста для новости». После выбора нужного текста, новость переходит в состояние «шаблон». Если

пользователь введет необходимые для формирования новости данные, новость перейдет в состояние «заполненная форма для размещения новости». После подтверждения ввода состоянием новости будет «шаблон и данные в режиме обработки». Если обнаружены ошибки, то осуществляется переход в состояние «неправильно введенные данные», где происходит перенаправление документа в состояние «заполненная форма для образца». Далее новость получает состояние «обработанные данные». Если же формирование новости прошло успешно, то последним переходом будет переход в состояние «сформированная новость в разделе «Новости».

В этой диаграмме были определены возможные состояния размещения новости на сервер. Моделирование этой диаграммы позволяет нам получить больше информации о системе.

Рисунок 6 – Диаграмма состояний

2 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.

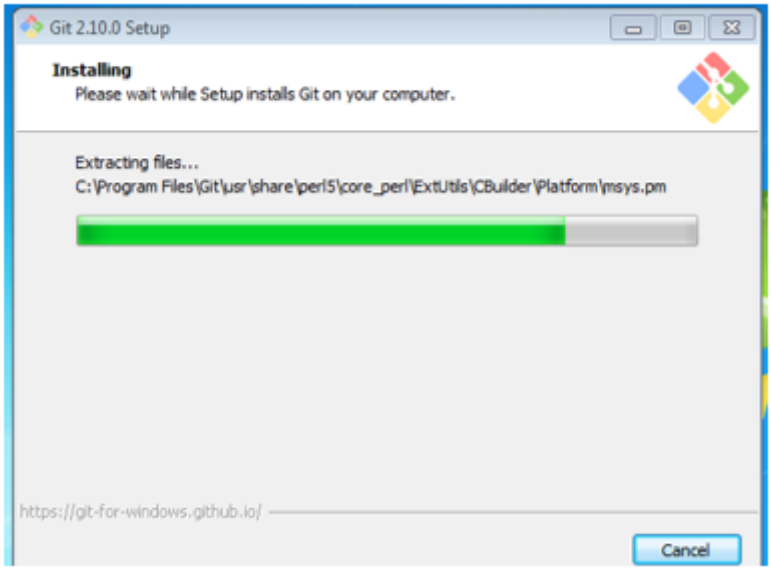


Рисунок 7 – Установка *Git*

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.

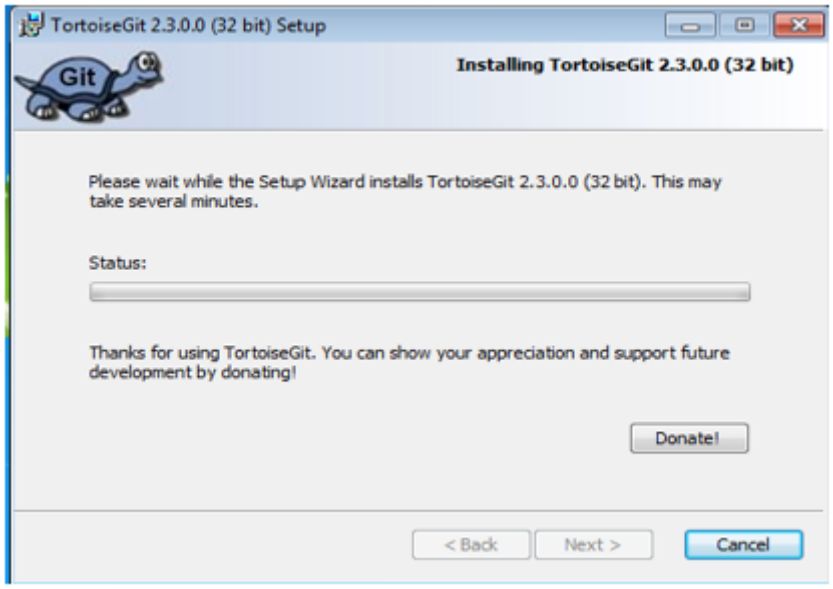


Рисунок 8 – Установка *TortoiseGit*

Зарегистрировались на *GitHub.com*.

Прописали следующие команды в консоли:

Git init – создание репозитория;
Git add . - добавляет содержание рабочей директории в индекс (*staging area*) для последующего изменения;
Git commit –m “сообщение что изменилось” - коммит;
Git remote add project https://github.com/vitaliyremi/soligorsk.loc.git -добавление удаленного репозитория;
Git push project master - вносим изменения в удаленный репозиторий.

Ссылка на удаленный репозиторий:

https://github.com/vitaliyremi/eworld

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.

3 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 берет 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 все 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*. Отдельно его скачивать и устанавливать не нужно. Прежде чем устанавливать *Laravel*, давайте убедимся, что *Composer* работает. Для этого запустим встроенную консоль *OpenServer*.

В консоли наберем команду “*Composer*”:

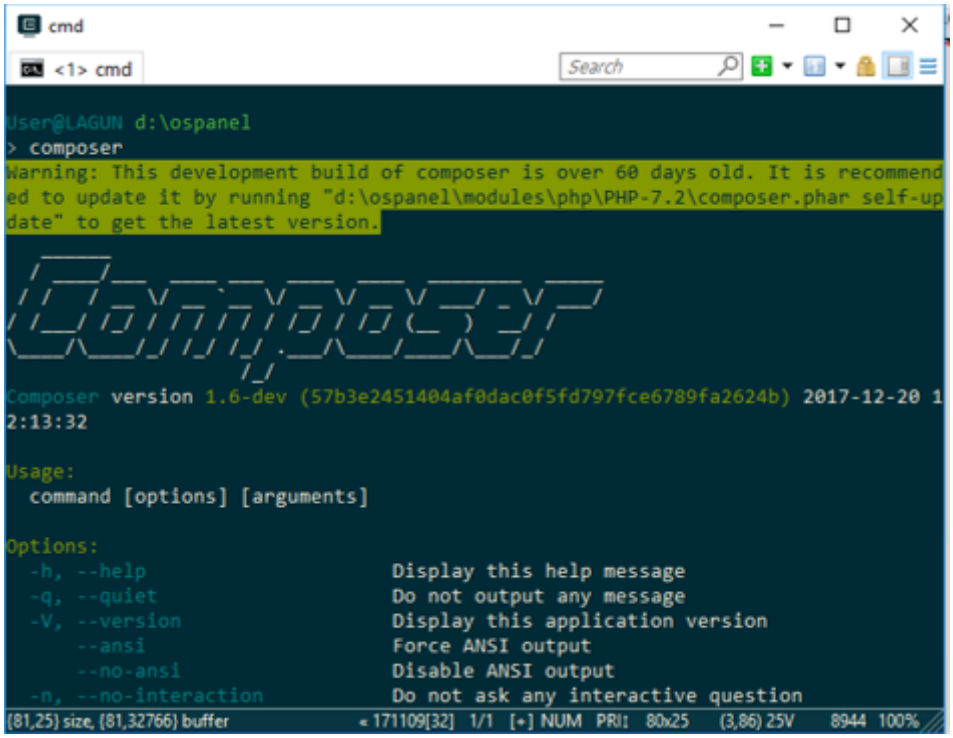


Рисунок 9 – Консоль

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

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

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

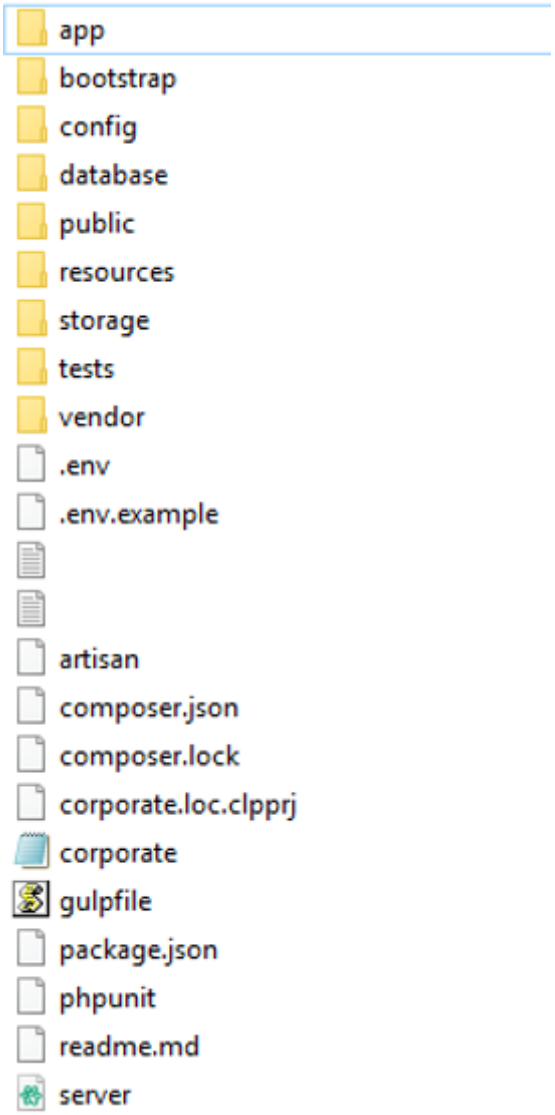


Рисунок 10 – *Laravel*

3.2 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.

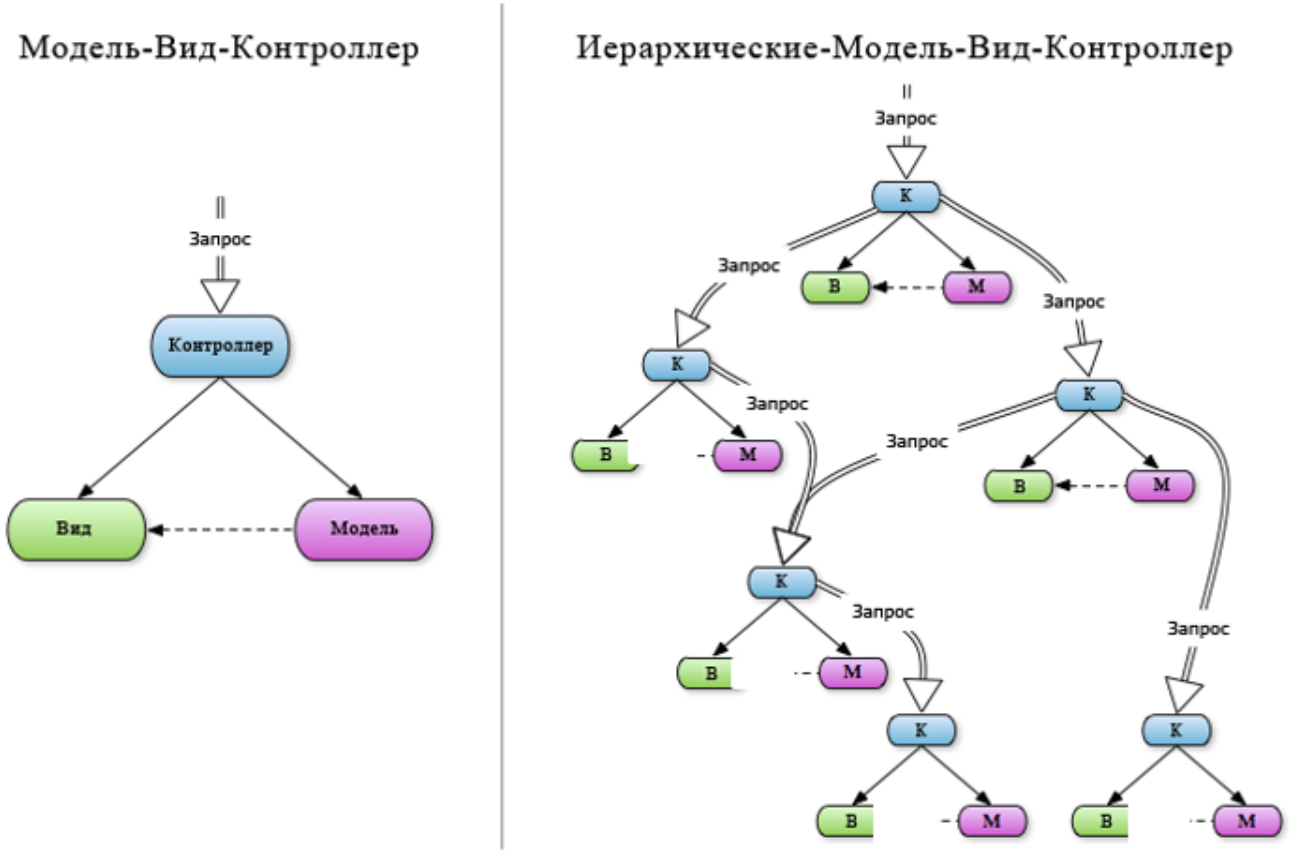


Рисунок 11 – Модели

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.

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.

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 данной курсовой работе:

```
Rout Route::get('/lodging', function () {
    return view('lodging'); // страница жилье
});
```

```
Route::get('/sight', function () {
    return view('sight'); //достопримечательности
});

Route::get('/leisure', function () {
    return view('leisure'); //досуг
});
```

Контроллеры

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. *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*.

1) Определение маршрута для контроллера с помощью метода *get*:

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

2) С помощью метода cotroller:

```
Route::controller(
     'profile' => 'ProfileController',
);
```

3) С помощью метода controllers:

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

А вот так будет выглядеть сам контроллер:

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

Применение контроллеров в данной курсовой работе, контроллер для создания новости *NewsController.php*

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


  }
  public function getAll(){
      $all = News::get();
      return view('newsall')->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 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 также 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 данной курсовой работе создали модель для каталога, файл модели называться app /News.*php*.

```
namespace App;

use Illuminate\Database\Eloquent\Model;

class News extends Model
{
   public $table = "news";
     }
```

Подключение и настройка базы данных осуществляется в файле /.env. Для подключения нужной базы данных прописали:

```
DB_CONNECTION=mysql
DB_HOST=localhost
DB_PORT=3306
DB_DATABASE=bdsoltur
DB_USERNAME=root
DB_PASSWORD=
```

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 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_News --create=news

В папке database/migration создался новый файл 2018_06_04_165249_create_news.*php*.

```
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class CreateNewsTable extends Migration
{
    public function up()
    {
        Schema::create('news', function (Blueprint $table) {
            $table->increments('id');
            $table->string('title',255);
            $table->text('text');
            $table->text('desc');
            $table->string('alias',150)->unique;
            $table->string('img');
            $table->timestamps();
        });
    }




    public function down()
    {
        Schema::drop('news');
    }}
```

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

increments() — добавить автоинкрементируемое поле — его будет иметь большая 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.

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 *resources\views* и имеют расширение *blade.php*. Шаблоны подключаются в экшне через хелпер *view*(), входящим параметром в который передается имя шаблона без расширения *blade.php*.

В папке *view* хранятся базовый шаблон *welcome.blade.php*

```
@extends('layouts.app')
@section('content')
<div class="container">
    <div class="row">
        <div class="col-md-10 col-md-offset-1">
            <div class="panel panel-default">
                <div class="panel-heading">Welcome</div>
                <div class="panel-body">
                    Your Application's Landing Page.
                </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 изменить эти контроллеры вообще.

Для создания шаблонов и ротов авторизации, выполнили следующую команду:

```
php laravel 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.

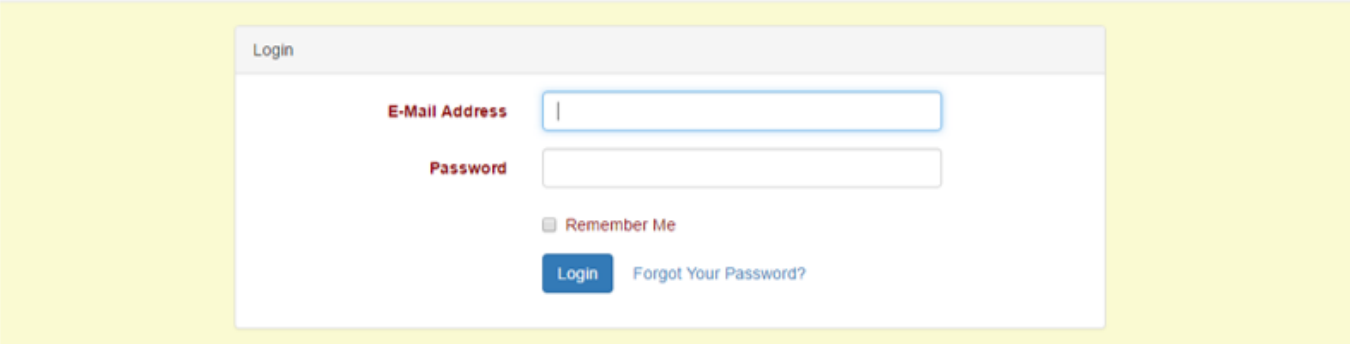


Рисунок 12 – Авторизация

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.

4 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.

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 системы.

Adapter – на основании некоторого класса создаем 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.

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.

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('/News', 'NewsController@index');

2 Создание форм

Создана форма в *news.create.php* (в папке views).

```
<div class="container">
    <div class="row">
        <div class="col-xs-12 col-sm-12 col-md-4 well well-sm">
            <legend>
                <?php echo validation_errors(); ?>
            </legend>
            <?php echo form_open('news/create') ?>
            <input class="form-control" name="title" placeholder="..." type="text" /><br />
             <textarea name="shorttext" id="shorttext" class="form-control" rows="4" cols="15"
```

```
required="required"
                        placeholder="..."></textarea><br/>
                <textarea name="text" id="text" class="form-control" rows="7" cols="25"
required="required"
                        placeholder="..."></textarea>
            <button class="btn btn-lg btn-primary btn-block" type="submit">
                ...
            </button>
            </form>
        </div>
    </div>
```

Сформированы необходимые массивы переменных в *create.php*:

```
$_POST['name'];
$_POST['email'];
$_POST['send'];
```

3 Маршрутизация (POST-данные)

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

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

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

4 Создание экшена *POSTINDEX* в контроллере *view*

Код экшена *PostIndex* в *view*.

```
<?php echo $news_item['title']; ?>
<?php echo $news_item['text']; ?>
```

5 Создание миграции, модели, auth

Миграция, модель и *auth* созданы в разделе 4.

Подключение *request* в контроллере *HomeController*:

```
use App\Http\Requests;
use Auth;
use Application\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*()).

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

    $r['id'] = Auth::post()->id;
    $r['autor_id'] = '-';
    $r['title'] = '-';
```

```
$r['excerpt'] = '-';
$r['body'] = '-';
$r['image'] = '-';
$r['slug'] = '-';
$r['meta_description'] = '-';
$r['meta_keywords'] = '-';
$r['status'] = '-';
$r['created_at'] = '-';
$r['updated_at'] = '-';
Product::create($r->all());
return redirect('news');
```

Добавление в таблицу нового поля:

```
$r['post_id'] = Auth::post()->id;
```

id	author_id	title	excerpt	body	image	slug	meta_description	meta_keywords	status	create
1	0	Hello World	Hang the jib grog grog blossom grapple dance the	<p>Hello World. Scallywag grog swab Cat o'nine tai...	pages/page1.jpg	hello-world	Yar Meta Description	Keyword1, Keyword2	ACTIVE	2018-0

Рисунок 13 – Поле id в базе данных

Демонстрация работы

Добавление новости:

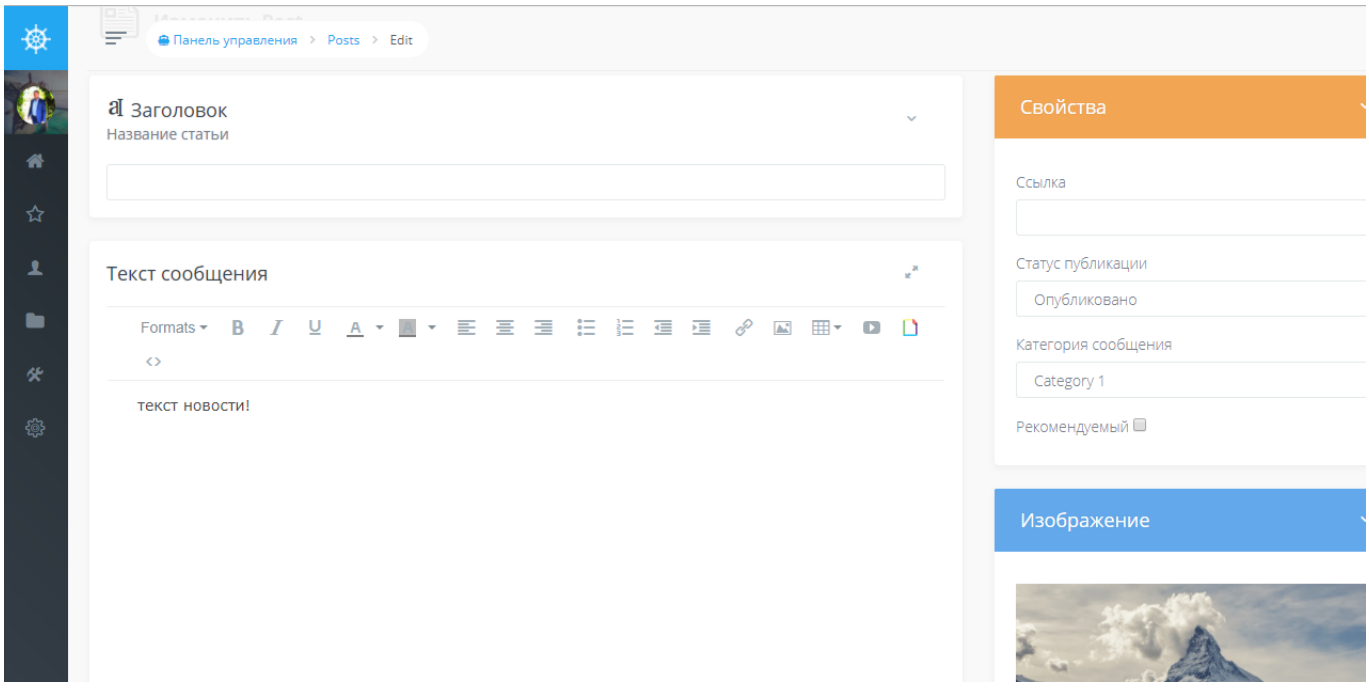
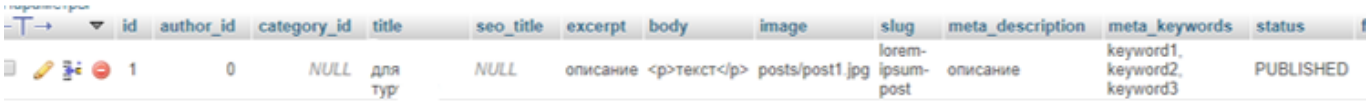


Рисунок 14 – Добавление новости

В базе добавился данный товар с id=1.



id	author_id	category_id	title	seo_title	excerpt	body	image	slug	meta_description	meta_keywords	status
1	0	NULL	для тур	NULL	описание	текст	posts/post1.jpg	lorem-ipsum-post	описание	keyword1, keyword2, keyword3	PUBLISHED

Рисунок 15 – База данных

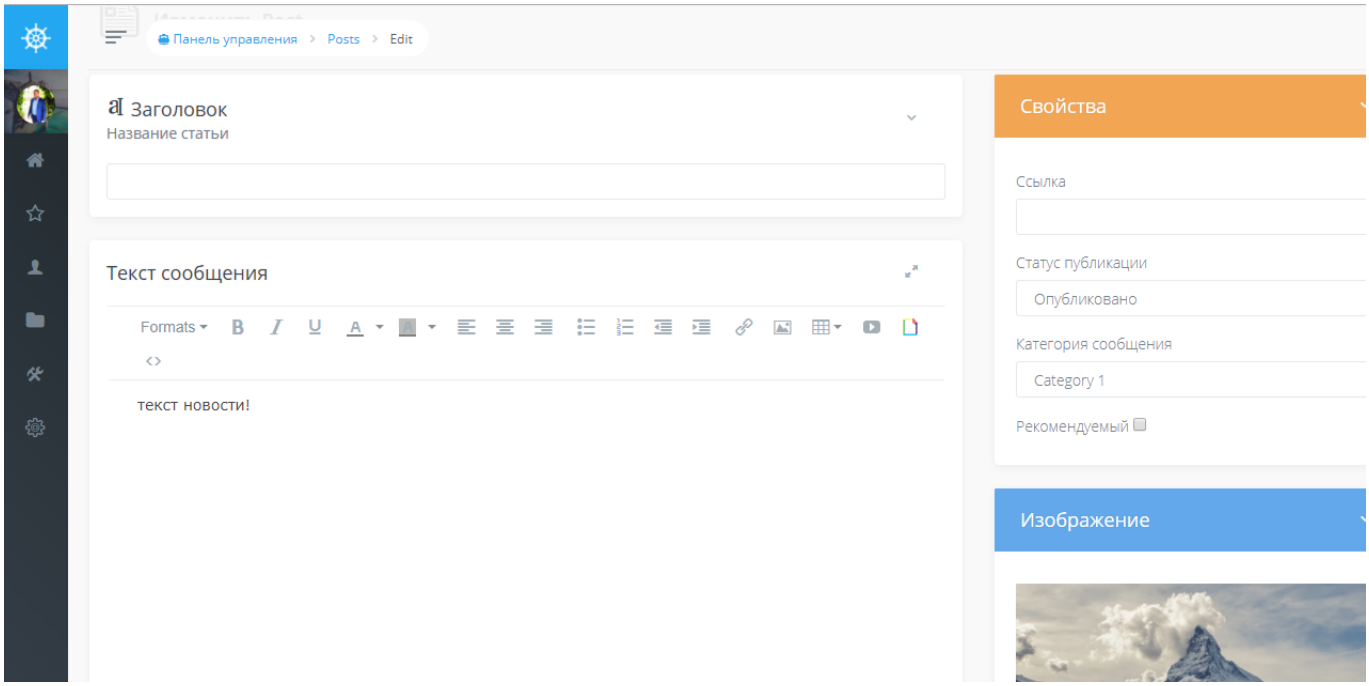


Рисунок 16 – Добавленная новость

В 4 разделе рассмотрено применение шаблона проектирования практических решений, показано на примерах создания *CRUD*.

5 VOYAGER БЫСТРОЕ АДМИНИСТРИРОВАНИЕ

Описание

Модель *User* с ролью администратора;

Seeds для загрузки таблицы с двумя пользователями. Один с ролью админ, 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 require tcg/voyager

php artisan voyager:install --with-dummy

Прописываем следующий класс в config/app.*php:*

TCG\Voyager\VoyagerServiceProvider::class,

В разделе 5 описана установка и возможности *Voyager*.

6 РЕЗУЛЬТАТ РАБОТЫ

В данном разделе представлены скриншоты готового веб-сайта.

Рисунок 17 - Главная страница

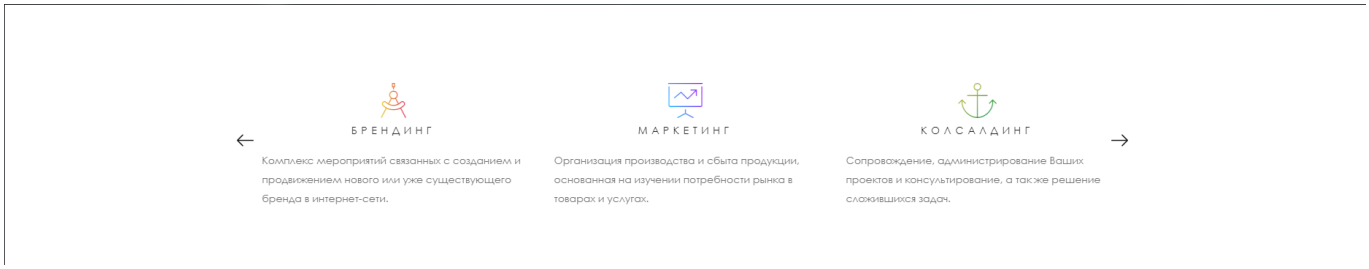


Рисунок 18 - Слайдер услуг

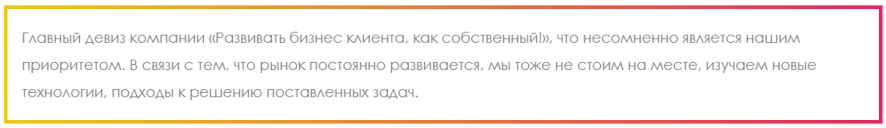


Рисунок 19 - Слоган компании

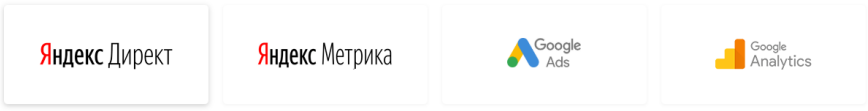


Рисунок 20 - Сертификаты подтверждающие профессионализм

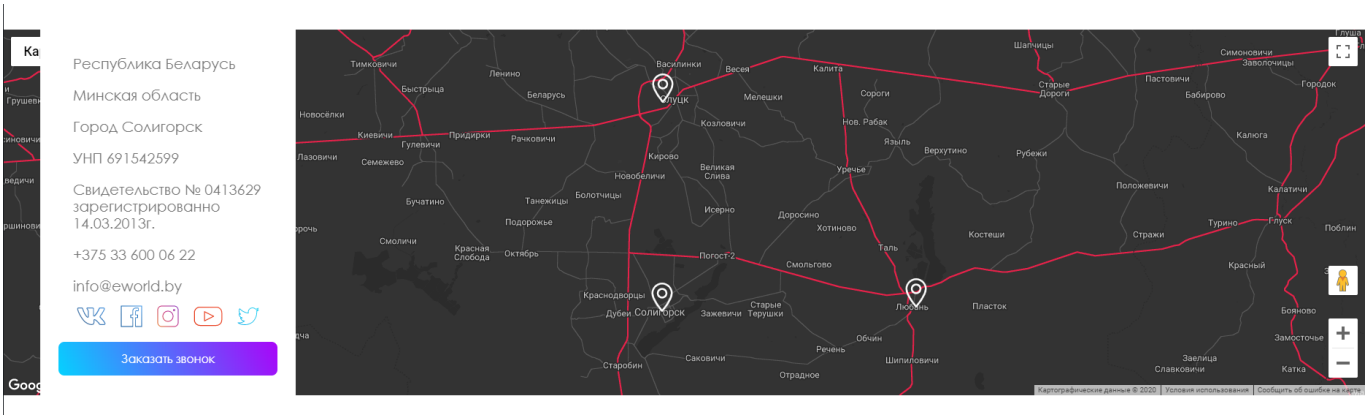


Рисунок 20 - Контакты компании

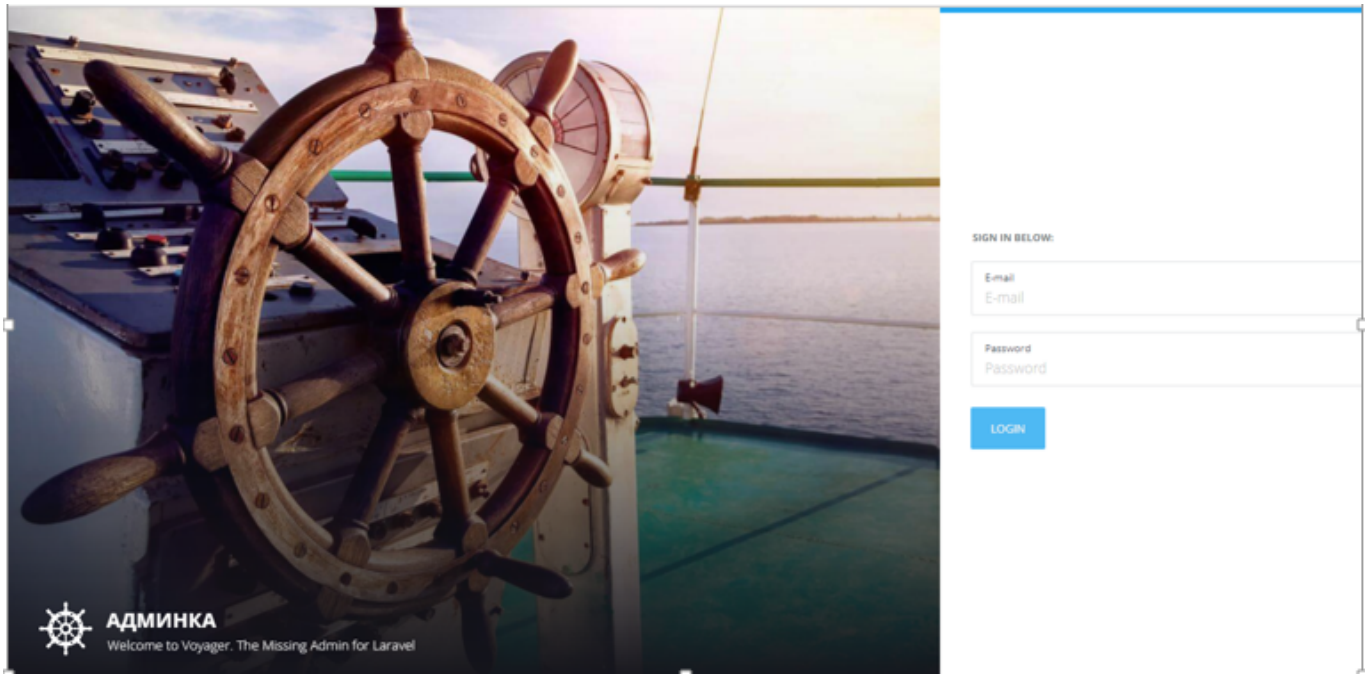


Рисунок 21 - Авторизация в админку

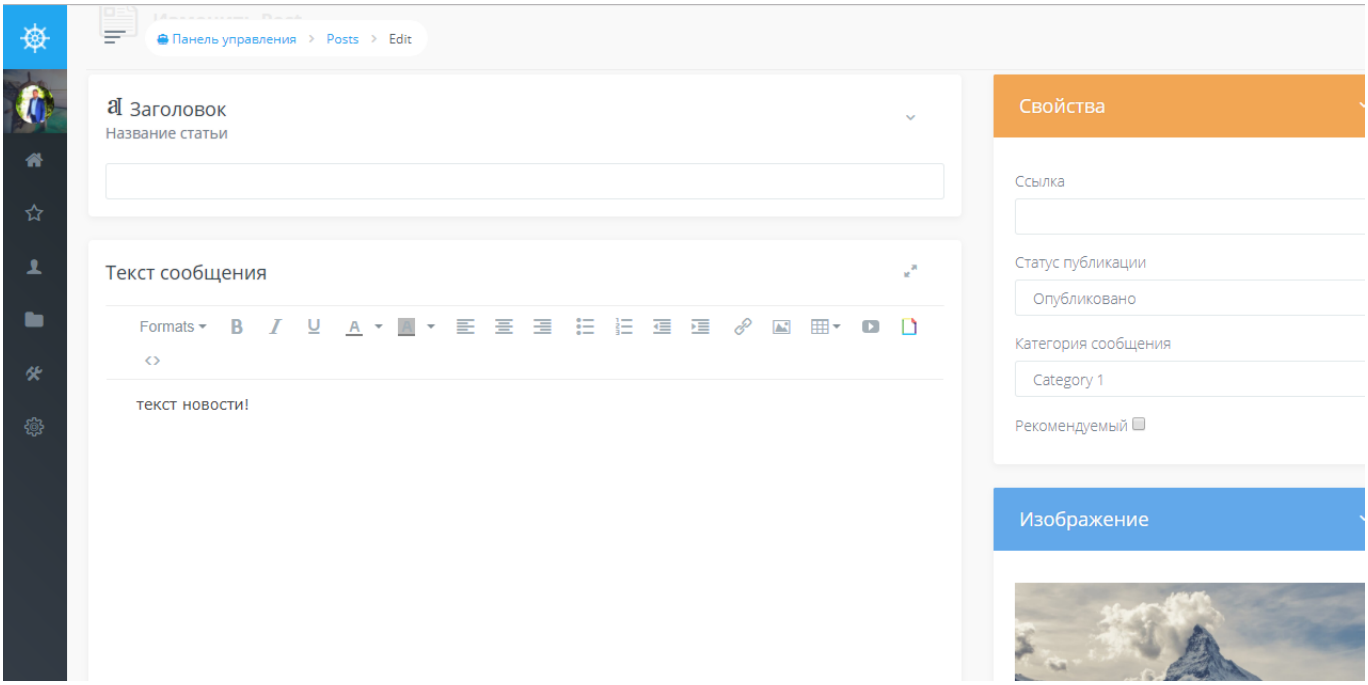


Рисунок 22 - Форма добавления текста

7 ПРИЛОЖЕНИЕ А (обязательное) Листинг кода (к разделу 2)

```
<!DOCTYPE html>
<html lang="ru">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>EWORLD - АГЕНСТВО ИНТЕРНЕТ-МАРКЕТИНГА</title>
<meta name="description" content="Агентсво комплексного интенет-маркетинга в Солигорске: привлечение клиентов в бизнес, лидогенерация, поисковое продвижение сайта (SEO), контекстная реклама и др.">
<link rel="shortcut icon" href="favicon.ico" type="image/x-icon">

<meta property="og:title" content="EWORLD - АГЕНСТВО ИНТЕРНЕТ-МАРКЕТИНГА" />
<meta property="od:description" content="Агентсво комплексного интенет-маркетинга в Солигорске: привлечение клиентов в бизнес, лидогенерация, поисковое продвижение сайта (SEO), контекстная реклама и др.">
<meta property="og:url" content="https://dev.eworld.by" />
<meta property="og:image" content="img/OpenGraph/soclogo.png" />

<meta property="vk:title" content="EWORLD - АГЕНСТВО ИНТЕРНЕТ-МАРКЕТИНГА" />
<meta property="vk:description" content="Агентсво комплексного интенет-маркетинга в Солигорске: привлечение клиентов в бизнес, лидогенерация, поисковое продвижение сайта (SEO), контекстная реклама и др." />
<meta property="vk:url" content="https://dev.eworld.by" />
<meta property="vk:image" content="img/OpenGraph/soclogo.png" />
<meta property="vk:description" content="Агентсво комплексного интенет-маркетинга в Солигорске: привлечение клиентов в бизнес, лидогенерация, поисковое продвижение сайта (SEO), контекстная реклама и др." />

<meta property="twitter:card" content="summary_large_image" />
<meta property="twitter:site" content="@eworldby" />
<meta property="twitter:title" content="EWORLD - АГЕНСТВО ИНТЕРНЕТ-МАРКЕТИНГА" />
<meta property="twitter:description" content="Агентсво комплексного интенет-маркетинга в Солигорске: привлечение клиентов в бизнес, лидогенерация, поисковое продвижение сайта (SEO), контекстная реклама и др." />
<meta property="twitter:image" content="img/OpenGraph/soclogo.png" />
<meta property="twitter:url" content="https://dev.eworld.by" />

<link rel="stylesheet" href="css/reset.css">
<link rel="stylesheet" href="css/et-line.css">
<link rel="stylesheet" href="css/swiper.min.css">
<link rel="stylesheet" href="css/style.css">
</head>
<body>
<!-- HEADER -->
<header data-vide-bg="video/MainPageBg">
<div class="name-company">
<div id="hello" class="hello">
<img class="main-logo" src="img/logoBIGWhite.png" alt="АГЕНСТВО ИНТЕРНЕТ-МАРКЕТИНГА">
<h1>АГЕНСТВО ИНТЕРНЕТ-МАРКЕТИНГА</h1>
</div>
</div>
<div id="overlay" class="overlay"></div>
</header>
<!-- MAIN -->
<div class="container">
<div class="slider">
```

```
<!-- Slider main container -->
<div class="swiper-container">
<!-- Additional required wrapper -->
<div class="swiper-wrapper">
<!-- Slides -->
<div class="swiper-slide">
<div class="slide-1">
<div class="alt-features-icon">
<span class="icon-circle-compass"></span>
</div>
<h2 class="services-title">Брендинг</h2>
<h3 class="services-description">Комплекс мероприятий связанных с созданием и продвижением нового или уже существующего бренда в интернет-сети.</h3>
</div>
</div>
<div class="swiper-slide">
<div class="slide-2">
<div class="alt-features-icon">
<span class="icon-presentation"></span>
</div>
<h2 class="services-title">Маркетинг</h2>
<h3 class="services-description">Организация производства и сбыта продукции, основанная на изучении потребности рынка в товарах и услугах.</h3>
</div>
</div>
<div class="swiper-slide">
<div class="slide-3">
<div class="alt-features-icon">
<span class="icon-anchor"></span>
</div>
<h2 class="services-title">Колсалдинг</h2>
<h3 class="services-description">Сопровождение, администрирование Ваших проектов и консультирование, а так же решение сложившихся задач.</h3>
</div>
</div>
</div>
</div>
<!-- If we need navigation buttons -->
<button class="arrow-left"></button>
<button class="arrow-right"></button>
</div>
</div>
<div class="container">
<div class="about">
<h2>Главный девиз компании «Развивать бизнес клиента, как собственный!», что несомненно является нашим приоритетом. В связи с тем, что рынок постоянно развивается, мы тоже не стоим на месте, изучаем новые технологии, подходы к решению поставленных задач. </h2>
</div>
</div>
<div class="container">
<div class="certificate">
<a href="#" class="link-block-cert">
<img src="img/certificate/yandex-direct.png" alt="Сертификат EWORLD Яндекс Директ">
</a>
<a href="#" class="link-block-cert">
<img src="img/certificate/yandex-metrika-iconx.png" alt="Сертификат EWORLD Яндекс Метрика">
```

```
</a>
<a href="#" class="link-block-cert">
<img height="40" src="img/certificate/GoogleAds.png" alt="Сертификат EWORLD Google Ads" >
</a>
<a href="#" class="link-block-cert">
<img src="img/certificate/google-analytics-logo.png" alt="Сертификат EWORLD Google Analytics">
</a>
</div>
</div>

<footer>
<div id="map"></div>
<div class="container">
<div class="contact">
<h2 class="contact-description">Республика Беларусь</h2>
<h2 class="contact-description">Минская область</h2>
<h2 class="contact-description">Город Солигорск</h2>
<h2 class="contact-description">УНП 691542599</h2>
<h2 class="contact-description">Свидетельство № 0413629 зарегистрированно 14.03.2013г.</h2>
<h2 class="contact-description">+375 33 600 06 22</h2>
<h2 class="contact-description">info@eworld.by</h2>
<div class="social">
<li><a href="https://vk.com/eworldby" class="social-link-vk" target="_blank"></a></li>
<li><a href="https://fb.com/eworldby" class="social-link-facebook" target="_blank"></a></li>
<li><a href="https://instagram.com/eworld_by" class="social-link-instagram" target="_blank"></a></li>
<li><a href="https://www.youtube.com/channel/UC8XbO3PH4wTH2p4wYQxQvzA" class="social-link-youtube"
target="_blank"></a></li>
<li><a href="https://twitter.com/eworldby" class="social-link-twitter" target="_blank"></a></li>
<!-- <li><a href="#" class="social-link-ok" target="_blank"></a></li>-->
</div>
<div class="buttom-phone">
<a href="#">Заказать звонок</a>
</div>
</div>
</div>
</footer>

<script src="https://cdn.jsdelivr.net/npm/jquery@3.4.1/dist/jquery.min.js"></script>
<script src="js/jquery.vide.min.js"></script>
<script src="js/blur.js"></script>
<script src="js/swiper.min.js"></script>
<script src="js/main.js"></script>
<script src="js/map.js"></script>

<script async defer
src="https://maps.googleapis.com/maps/api/js?key=AIzaSyBv7K9XwSvKVGgszi6pdG8CZQlEmbR73LM&callback=initMap"
type="text/javascript"></script>
<!-- Yandex.Metrika counter -->
<script type="text/javascript" > (function(m,e,t,r,i,k,a){m[i]=m[i]||function(){(m[i].a=m[i].a||[]).push(arguments)};
m[i].l=1*new
Date();k=e.createElement(t),a=e.getElementsByTagName(t)[0],k.async=1,k.src=r,a.parentNode.insertBefore(k,a)})
(window, document, "script", "https://mc.yandex.ru/metrika/tag.js", "ym"); ym(57634066, "init", { clickmap:true,
trackLinks:true, accurateTrackBounce:true, webvisor:true }); </script> <noscript><div><img
src="https://mc.yandex.ru/watch/57634066" style="position:absolute; left:-9999px;" alt="" /></div></noscript>
<!-- /Yandex.Metrika counter -->
```

</body>
</html>

8 ЛИСТ ЗАДНИЯ

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 Лагун Виталий Константинович

группа 784371

1.Тема проекта: Разработка сайта агенства интернет-маркетинга "EWORLD"

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

3.Исходные данные к проекту:

3.1.Описание к выполнению Разработка сайта агенства интернет-маркетинга "EWORLD" 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

Заключение

В данной курсовой работе был разработан веб-сайт агенства интернет-маркетинга "EWORLD". Основными технологиями проекта являются PHP, MySQL. Программная среда проекта Open Server. Инструментами разработки являются интегрированная среда разработки PhpStorm, Adobe Photoshop CC, Atom, FilleZilla, NotePad ++, Movie Studio Platinum и сервис Creately.com. В результате работы, поставленные задачи были выполнены. А именно, информационный проект создан для того чтобы размещать актуальную информацию для клиентов. Создан дизайн-макет информационного ресурса, который совпадает с готовым проектом. Установлен 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. Проект размещен на удаленном репозитории. Ссылка на репозиторий: https://github.com/vitaliyremi/eworld.git. А также размещен по адресу https://eworld.by Проект создан на основе фреймворка Laravel. Он берет 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 Voyager для быстрого администрирования. В целом предлагаемый сайт позволит значительно повысить имидж компании EWORLD, так как сайт содержит всю необходимую информацию об услугах, контактах, что позволяет клиенту легко ориентироваться, найти необходимую услугу, решать быстро сложившиеся вопросы для ростабизнеса.

Список использованных источников

1. [url] eworld.by
2. [url] **Laravel** https://laravel.com
3. [url] **bootstrap** https://getbootstrap.com
4. [url] **composer** https://getcomposer.org
5. [url] **Open Server Panel** https://ospanel.io

Приложения