**ТИТУЛЬНЫЙ ЛИСТ**

**Cодержание**

**Введение**

Мой условный путь в программирование начался задолго до обучения в GeekBrains. Будучи студентом машиностроительного техникума, я изучал веб-разработку с использованием довольно простых инструментов по типу Adobe Muse или Tilda, что по своей сути является конструктором сайтов. Если заглянуть непосредственно в код проектов, сделанных таким образом, можно ужаснуться. Немного позже я углубился в изучение HTML и CSS по огромной книге «Head First HTML and CSS» (Робсон Элизабет и Фримен Эрик) после этого JavaScript по книге «Изучаем программирование на JavaScript» за авторством тех же писателей. Но после недолгих практических шагов в JavaScript, жизненные обстоятельства заставили на какое-то время отстраниться от этой ниши.

Спустя шесть лет мне придется вернуться к одному из своих проектов, чтобы продемонстрировать результат своего обучения.

Размышляя над темой дипломного проекта, мне очень пригодился свой предыдущий опыт. Чтобы проект не выглядел выдуманным из головы и не высосанным из пальца, я решил модернизировать существующую франшизу. Раньше приходилось брать много задач и делать их в одиночку, не считая настройки рекламных кампаний. А это написание текста для сайта, создание оффера, создание дизайн макета сайта, логотипа, и, конечно же все это необходимо было сверстать и заставить работать.

В плане на эту дипломную работу – взять созданный мной ранее проект, который реализован на HTML/CSS и JavaScript, без использования каких-либо фреймворков и сделать из него полноценное веб-приложение. Для этого я воспользуюсь фреймворком Laravel.

Мы рассмотрим его преимущества в дальнейшем, но если говорить вкратце, то я остановил свой выбор на конкретном фреймворком по простым причинам:

* открытый исходный код
* безопасность
* возможность масштабирования
* простота работы с базой данных

И в целом, мне как новичку Laravel показался очень удобным и интересным фреймворком, который я хочу в дальнейшем изучить глубже.

В задуманном мной проекте, конечно, можно обойтись без таких функциональных фреймворков, но в качестве презентации своих навыков и умений, как fullstack разработчика – отлично подойдет, плюсом послужит и то, что приложение можно будет масштабировать по своему желанию.

Сам заказчик, на момент обращения ко мне, занимался изготовлением loft мебели в Санкт-Петербурге под заказ. Насколько я понимаю на текущий момент продолжает заниматься, но сам сайт уже изменил. Во избежание недопониманий и просто на всякий случай я заменю всю контактную информацию и некоторые фотографии заглушками. Но все созданное в проекте является моей интеллектуальной собственностью (помимо фотографий), и я считаю допустимым непубличное использование веб-сайта в качестве основы моего дипломного проекта.

План проекта:

1. Разработать структуру и дизайн веб-страницы.
2. Сверстать страницу, используя HTML.
3. Использовать CSS, чтобы оформить страницу.
4. Использовать JavaScript, чтобы написать логику взаимодействия со стороны пользователя.
5. Подключить базу данных для работы с пользователями и товарами.
6. Разработать систему нотификации.
7. Создать административную панель для управления контентом, не прибегая к помощи специалистов.

Инструменты, которые будут использованы в проекте:

- Adobe Photoshop ()

- Visual Studio Code (для работы с языком программирования PHP лучше использовать PHP Storm, но для нашего проекта можно обойтись и VSCode) \*можно описать какие внутри инструменты

* Git
* Laravel 11.0
* Postman
* PhpMyAdmin
* Composer

Состав предполагаемой команды: маркетолог, копирайтер, дизайнер, frontend разработчик,

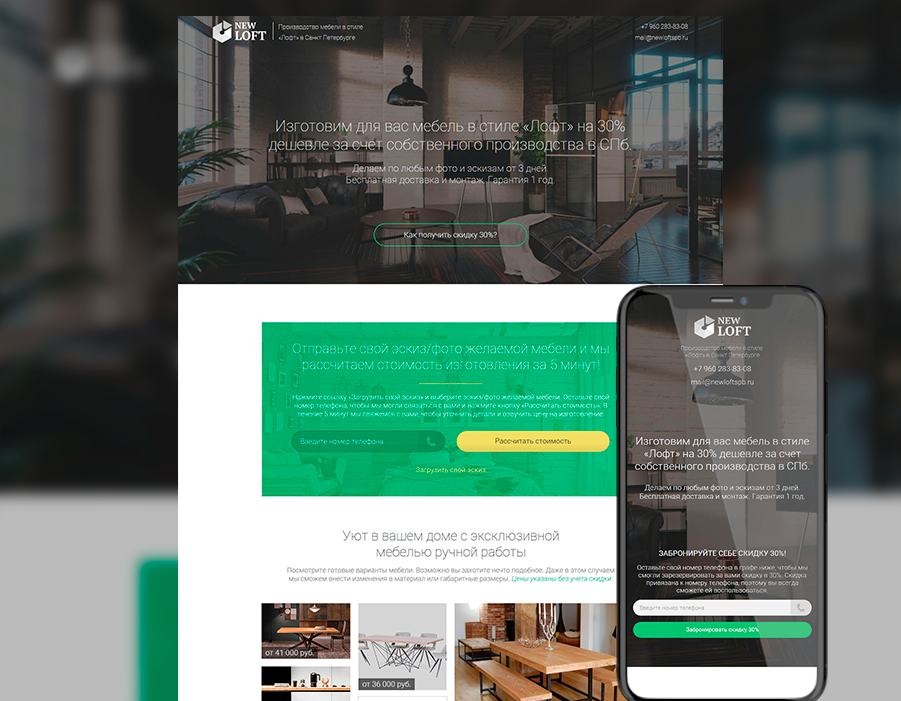
backend разработчик (или fullstack вместо frontend и backend). В качестве вышеперечисленных работников буду выступать непосредственно я сам.

**Верстание страницы по дизайн-макету.**

На текущий момент разработчики в большинстве своем используют Figma или аналогичные онлайн-сервисы для работы с прототипами интерфейсов в реальном времени. В нашем случае дизайн-проект разрабатывался в Adobe Photoshop. Ничего плохого в этом нет, но сам Photoshop довольно ресурсоемкий для операционной системы и не очень подходит для таких задач, как посмотреть расстояние между элементами. Естественно, что лучше использовать что-то проще.

Итак, допустим, что дизайнер разработал будущий вид сайта и нам, как fullstack разработчику, необходимо все это собрать в уже работающее приложение.

Так как «скелет» проекта у меня уже есть, я попросту его доработаю, используя новые технологии и знания. При этом не будем углубляться во внешний вид. Оставим все так, как и было.



Начнем с верстки, используя HTML.

**HTML5**

HTML (HyperText Markup Language или язык гипертекстовой разметки) – это язык разметки, который используют для создания веб-страниц. Если говорить простым языком, весь контент, который вам необходимо разместить помещается в определенные теги. Для каждого типа контента предназначен свой собственный тег.

На текущий момент используют последнюю пятую версию HTML. Она позволяет располагать на странице мультимедиа, не прибегая к помощи других программ, как это было ранее, также улучшилась сама семантика: появились новые теги, которые помимо всего прочего помогают браузеру лучше ранжировать страницы, а это плюс к SEO продвижению, ну и конечно, с ростом технологий HTML5 лучше поддерживается смартфонами.

**CSS3**

Если говорить о верстке страницы, нам не обойтись без визуальной составляющей. В этом нам поможет CSS (Cascading Style Sheets / каскадная таблица стилей). Это своего рода правила визуализации HTML элементов. С помощью CSS мы «сообщаем» браузеру как в конечном итоге должна выглядеть страница.

В наше время CSS перетянул очень много функций с JavaScript. Поэтому очень многое можно сделать, не прибегая к прямому программированию. А как мы знаем, что HTML, что CSS не являются языками программирования.

**JavaScript**

Хотя я и сказал, что очень многое можно сделать, не прибегая к JS, это вовсе не значит, что можно обойтись без него. Вся логика внесена именно сюда. А такие фреймворки как ReactJS упрощают конечное производство за счет огромных библиотек готовых решений. Вам уже не придется писать какие-либо банальные функции, тратя свое драгоценное время.

**Подытожим…**

Итак, три кита построения веб страницы: HTML, CSS, JavaScript.

Сделаю некоторое отступление от темы и объясню некоторые моменты на будущее.

В сложном проекте скорее всего придется работать с фреймворками по типу Vue или библиотекой ReactJS, которые упрощают жизнь фронтенд-разработчикам. Они используют декларативный подход, который состоит в описании конечного результата, вместо императивного, где описываются конкретные шаги для получения результата.

В данном проекте не имеет смысла использовать ничего такого. Потому что, по сути, у нас одна страница. В конечном итоге ее можно разбить на компоненты, при масштабировании проекта или при его переосмыслении. В текущем проекте мы используем простой шаблон blade. Но это я забегаю вперед.

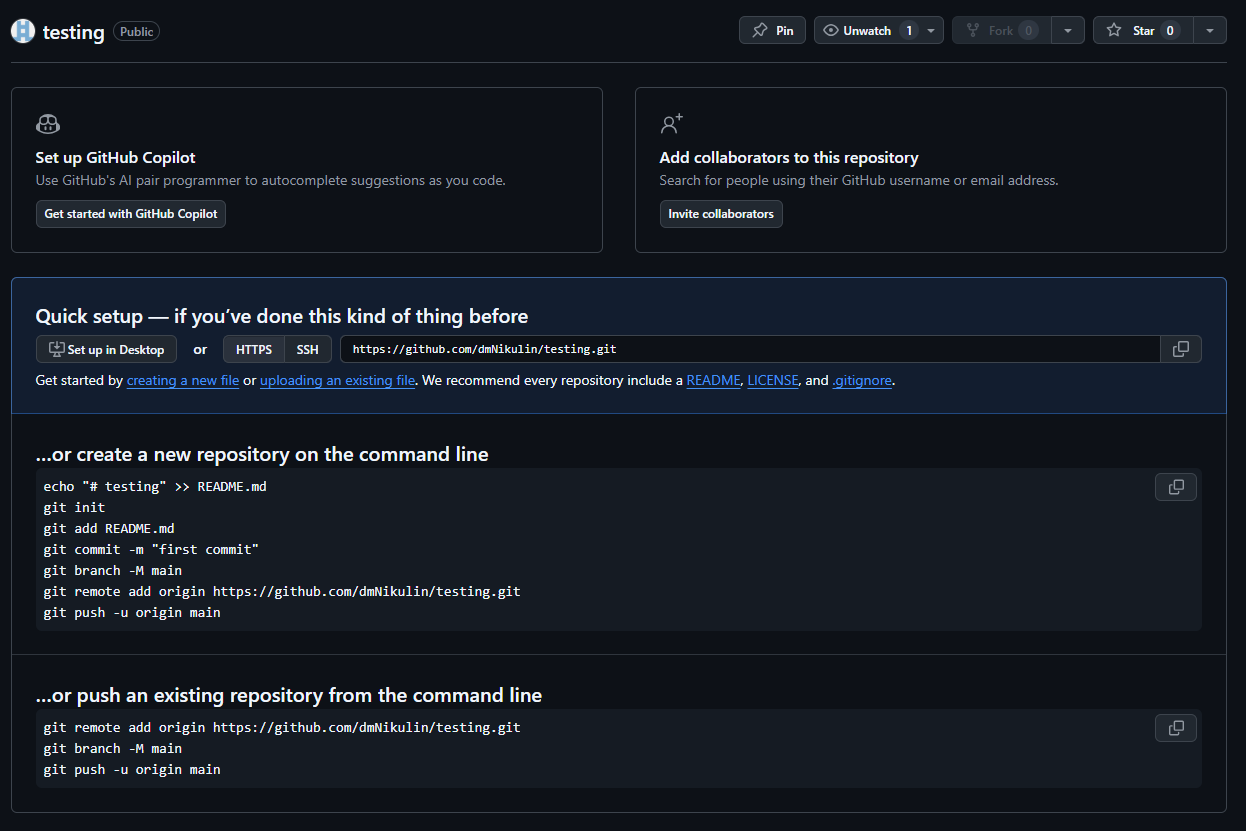
**Перед началом работы.**

Чтобы как-то контролировать процесс разработки: иметь возможность быстро откатывать все назад при необходимости, мы будем пользоваться системой контроля версий GitHub.

В целом вся суть таких систем заключается в том, чтобы синхронизировать работу нескольких разработчиков. То есть где-то на сервере хранится исходный код веб-приложения (репозиторий) и каждый из команды имеет к нему доступ (а может и не только из команды), имеет возможность загрузить этот код себе, работать с кодом и отправить изменения в репозиторий. Однако только владелец может принять изменения и сделать слияние с исходным кодом.

Инициализировать такую систему лучше при старте разработки и не забывать вовремя коммитить свою работу (то есть зафиксировать определенный результат на текущее время).

Чтобы подключить git к своему проекту не понадобится много времени. Для начала регистрируемся на gith устанавливаем на компьютер, после этого на сайте github создаем новый репозиторий, настраиваем под свои задачи, а после создания сам сервис подскажет что дальше делать.



Есть и альтернативный способ это сделать напрямую через терминал VsCode, но тут кому как удобнее.

**Начинаем верстку**

Заходим в наш редактор кода и начинаем с базовой разметки HTML. Для облегчения и скорости работы нам понадобятся некоторые расширение по типу HTML5 Snippets, JS-CSS-HTML Formatter, Emmet и Sass.

**<head>** - записываем сюда все метаданные, описание страницы, ключи для ранжирования в поиске, заголовок страницы, подключаемые шрифты, стили и прочее. Иногда сюда, при определенных случаях, можно вставить и ссылку на какой-либо скрипт. В моем случае была подключена метрика.

**<body> -** основное тело нашей страницы. Размещаем сюда весь контент, а это такие теги, как заголовки разного уровня (<h1>, <h2> и т.д.), параграфы (<p>), ссылки (<a>), картинки (<img>), видео (<video>), аудио <audio>, списки (<ul>, <li>) и прочие.

У большинства тегов есть свои атрибуты – это дополнительный значения, которые настраивают элемент соответствующим образом. Есть обязательные атрибуты, такие как href у ссылки, а есть и дополнительные, такие как id или class. Один из основных атрибутов, которые нам нужно использовать практически для каждого элемента это class:

**<a href=”#” class=”some-class”/>**

Атрибут class присваивает идентификатор элементу, по которому мы сможем обращаться в CSS. Причем атрибут class с одним и тем же именем может быть присвоен группе элементов в отличие от атрибута id, который, во избежание ошибок, должен быть уникальным и единственным.

Разрабатывая разметку нам сразу предстоит думать и о названиях классов для будущей работы со стилями. Мы будем использовать препроцессор Sass, который ускоряет написание стилей. Sass позволяет создавать переменные, делать вложенные стили, миксины, использовать фрагментирование и множество других функций, упрощая разработку и делая ее очень удобной.

Вот наш начальный блок. Обратите внимание на название классов.

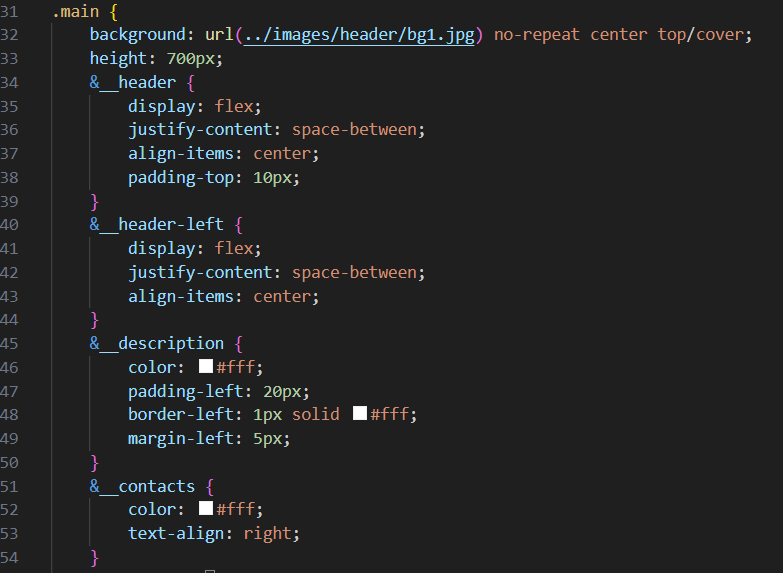
Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Получается некая древесная структура, где main – корень, а приписки вида \_\_header – ветви. Отчетливо это будет видно в файле css.

В CSS мы начинаем прописывать некие правила по схеме «свойство: значение». Описывать их все не имеет смысла, потому что их очень много, не говоря уже о системе позиционирования элементов. Но можно сказать, что если вам что-то придет в голову в плане визуализации элементов, то вероятнее всего это можно сделать с помощью стилей.

Обратите внимание как препроцессоры облегчают читаемость кода. У нас есть основной класс main и все свойства, что описаны внутри этого блока, касаются конкретного блока в html.



Таким образом описываем каждый блок с дизайн макета и получаем практически готовую веб-страницу. Что касается форм, то мы пропишем их логику и взаимодействие непосредственно в Laravel. Сейчас нам остается прописать некоторый функционал по всплывающему окну на JavaScript.

**JavaScript**

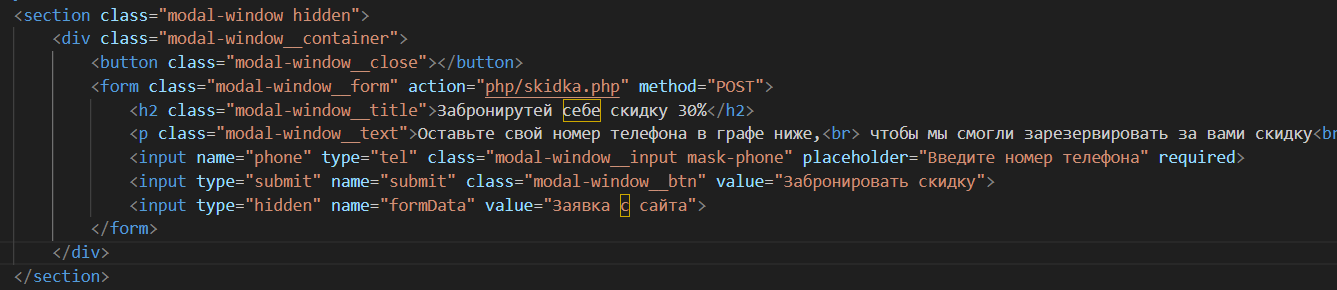
Во время обучения мне показалось, что я нашел тот язык программирования, который больше всего нравится и это JS. Хотя до этого каждый язык мне нравился, но чем-то именно JavaScript меня зацепил. Возможно, потому что я изучал его до этого и это связывало меня с прошлым. Еще даже до создания сайтов, будучи подростком я пытался ковырять игровой движок Unity, где одним из вариантов прописывания скриптов был JS.

Делая дипломный проект, я немного разочаровался, потому что пришлось буквально чуть-чуть коснуться этого языка.

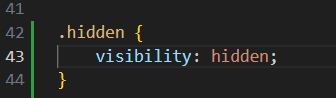
Я принял решение не перетягивать без изменений ранее созданную страницу и немного ее модернизировал. До сих пор сильны воспоминания, как я пытался создать модальные окна и в итоге перетягивал скрипты с jQuery. Не могу даже предположить в чем была проблема.

Сейчас же это просто пара строчек и все работает исправно.

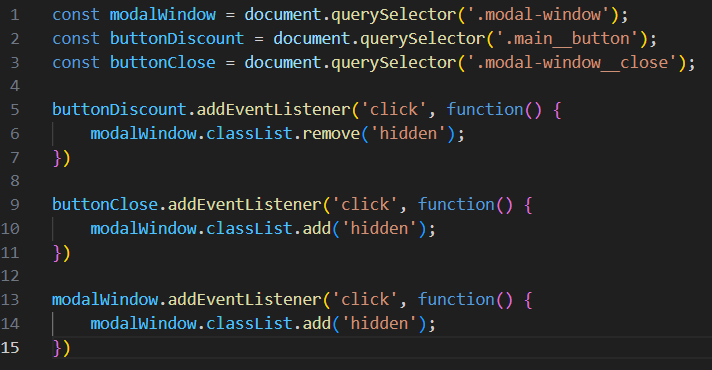
Размечаем окно в HTML.



Не буду прикладывать скриншот стилей для него, думаю понятно, что делаем все по макету. Единственное, что нам действительно понадобится это:



Соответственно накладываем этот класс на наше окно и все, что осталось сделать:



1. Найти все кнопки, которые нам нужны.
2. Добавить им событие клика.
3. Добавить или убрать класс hidden.

Проще не придумаешь. Даже жаль, что это все, что нам понадобится в этом проекте. Можно было бы поработать с масками на поля ввода, но думаю, что это уже лучше делать в Laravel на PHP, потому что все равно придется делать валидацию перед внесением в базу данных.

**Установка необходимых приложений**

Следующим шагом мы создадим проект Laravel. Во-первых нам понадобиться Composer – менеджер пакетов и зависимостей для языка PHP. Сам PHP тоже необходимо установить.Во-вторых нам понадобится Open Server Panel – локальный веб сервер, а также установить PHP MyAdmin для работы с базами данных.

**Composer**

Для чего нужен Composer? Ответ простой: для облегчения работы с необходимыми библиотеками, которые нужны для функционирования приложения. Самым нужным файлом является composer.json, где описаны все необходимые пакеты и их зависимости.

Это упрощает разработку, так как даже в git вам не надо перетягивать все библиотеки, что уменьшает вес исходного проекта. Вместо этого, все пакеты, которые необходимо будет скачать, находятся в одном файле и одной командой вы можете это все установить.

1. На всякий случай обновим сам composer:

$ composer self-update

1. Создадим проект Laravel командой:

$ composer create-project laravel/laravel . , где точка это текущая директория.

Если есть какие-либо ошибки с зависимостями или версиями пакетов, установите модуль fileinfo, откройте php.ini и раскомментируйте строку:

;extension=fileinfo

После очередной установки или обновления все ошибки должный уйти, если нет:

$ composer install --ignore-platform-reqs

$ composer update --ignore-platform-reqs

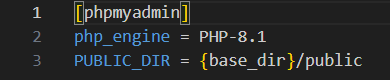
Эти две команды устанавливают/обновляют пакеты, игнорируя проблемы с зависимостями.

**Open Server Panel**

Устанавливаем с официального сайта и для полноценной работы PHP MyAdmin нам понадобится настроить локальный домен. Для новичка это не самый простой процесс, и столкнувшись впервые самолично, я потратил много времени, чтобы все заработало. В списке литературы я прилагаю статью, в которой лучше всего описано как сделать все правильно.

Но если вкратце:

1. Перейдите в директорию OSPanel/home и создайте там папку phpmyadmin, и внутри еще две .osp и public.
2. В директории .osp создайте файл project.ini и добавьте в него следующее:



И перезагрузим программу.

1. Затем нужно скачать архив с последней версией PhpMyAdmin и распаковать их в ранее созданную папку: OSPanel\home\phpmyadmin\public\
2. Переименуйте файл config.sample.inc.php в config.inc.php.
3. Введите случайную символьную строку в качестве секрета шифрования и создайте сервер:



1. Запустите OSPanel **от имени администратора** и включите необходимые модули. Можно через терминал:

osp on php-8.1

osp on mysql-8.0

После этого по адресу phpmyadmin/ у вас должна появиться админ панель, все параметры для входа описаны в config.inc.php.

*Все прописанные в конфигах версии могут отличаться.*

Теперь мы можем работать с базами данных. Альтернативный вариант – это MySQL Workbrench или можно воспользоваться SQLite. Выбирайте на свой вкус.

**Предварительная настройка Laravel**

Так изначально выглядит наш созданный проект:

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Все команды выполняются через интерфейс командной строки Artisan.

Сгенерируем ключ приложения командой:

$ php artisan key:generate

Затем надо подключить базу данных, иначе при запуске сервера вы получите ошибку. Большинство настроек приложения вы найдете в файле .env. Если у вас пока нет системы управления базы данных можно поменять SESSION\_DRIVER на file, вместо database. Или, как вариант, можно использовать SQLite, которая позволит вам работать с базой данныз прямо из редактора кода. Но для VSCode придется установить определенные расширения для корректной работы по типу SQLite 3 Editor.

Так выглядит настройка базы данных в моем случае:

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

Мы практически готовы к тестовому запуску. Не забываем включить OSPanel, чтобы не получить ошибку соединения с базой данных.

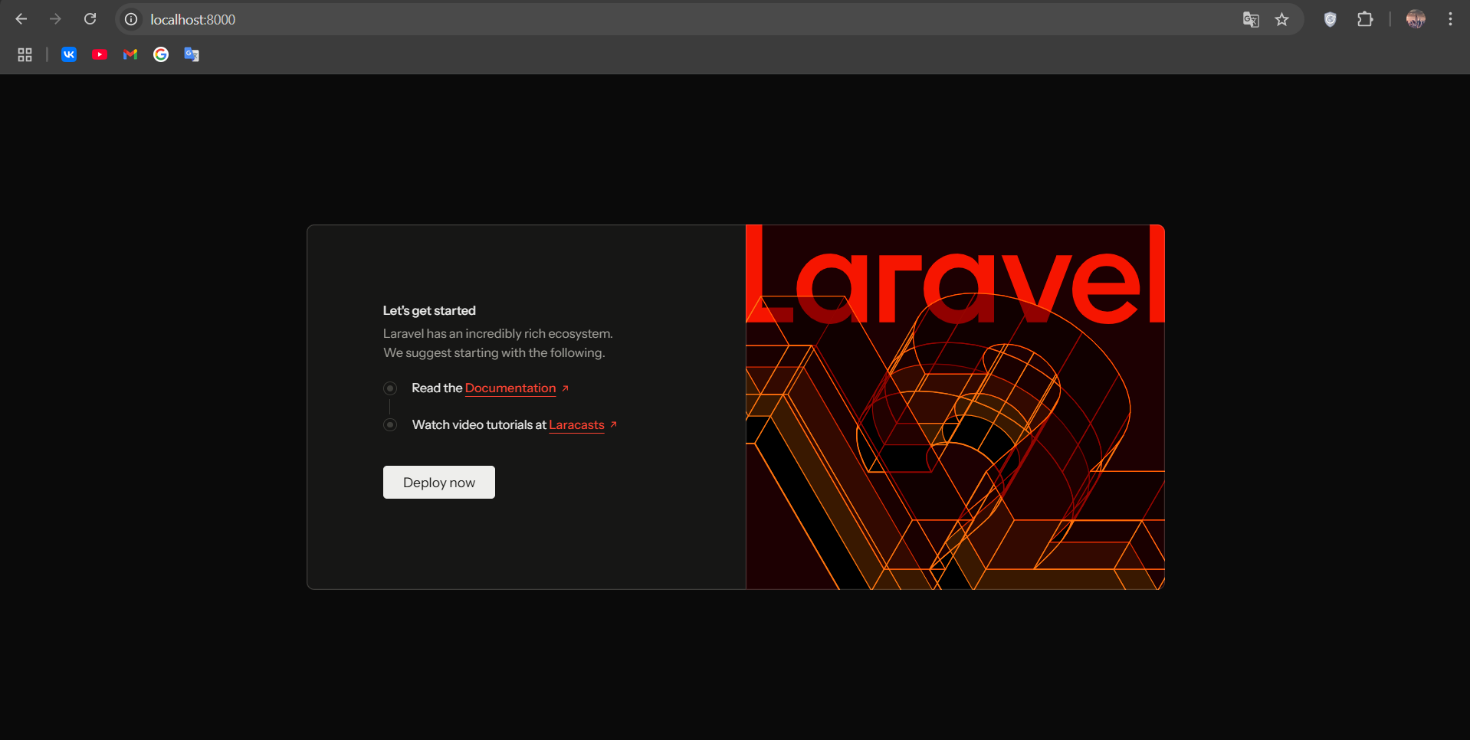
Выполняем команду, которая обновит наш файл конфигурации .env, сделаем миграции (синхронизация проекта с базой данных) и запускаем встроенный сервер:

$ php artisan config:clear

$ php artisan migrate

$ php artisan serve

И добро пожаловать на приветственную страницу Laravel. Раз все загрузилось успешно и без ошибок, продолжим настраивать наше приложение, но сначала немного о самом Laravel.



**Laravel**

Многофункциональный фреймворк с открытым исходным кодом, который использует за основу язык программирования PHP. Во фреймворке построение архитектуры разрабатывается по принципу Model View Controller (MVC). Такую архитектуру или паттерн, если угодно, использует, например, Angular JS. Возможно, если бы в курсе обучения были курсы по Angular, то под этот проект он бы подошел лучше.

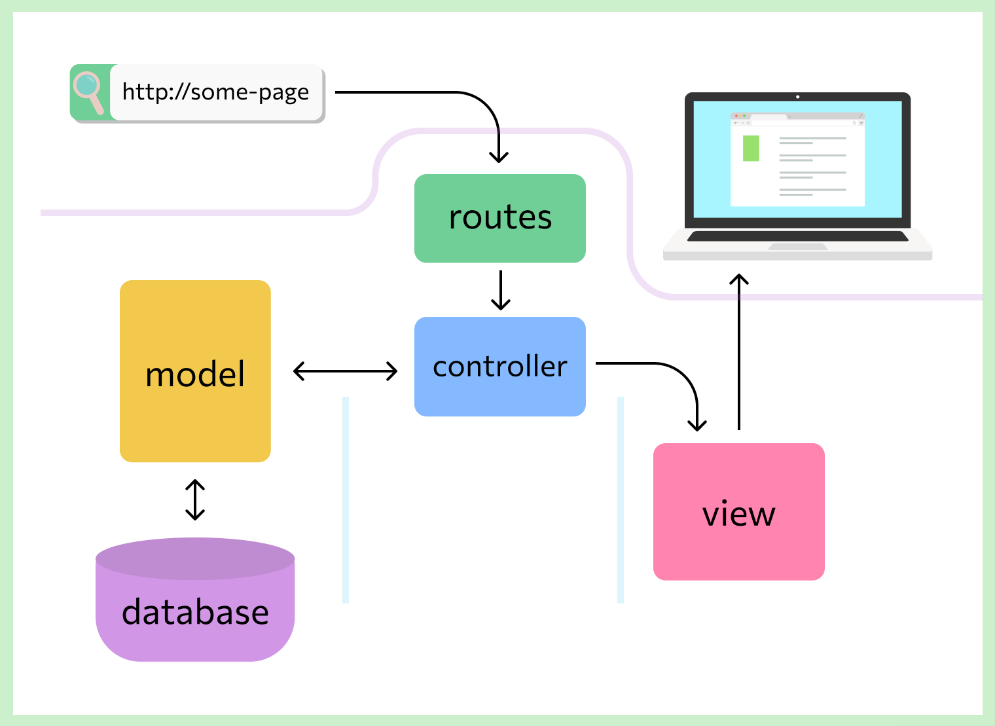
Как же работает архитектура MVC? По сути это три сущности, каждая из которых отвечает за определенный аспект работы и при этом они тесно связаны между собой.

- Model – данные и методы работы с этими данными (проверки и запросы в базу данных)

- View/представление – внешний вид или представление. То с чем взаимодействует конечный пользователь.

- Controller – устанавливает взаимосвязь между представлением и моделью. Направляет данные от пользователя к системе.

Пользователь взаимодействует с представлением или интерфейсом и у него есть некие варианты управления: отправка форм, нажатие кнопок, переходы по ссылкам и прочее. Это взаимодействие как раз и обрабатывается контроллером. Контроллер в свою очередь взаимодействует с моделью, сообщая ему о поступивших изменениях или запросах. А результаты этих изменений снова попадают в представление, сообщая пользователю итог.



Преимущества Laravel

* Открытый исходный код. Это значит, что вы можете досконально узнать и разобрать как и что работает.
* Безопасность. Собственная технология программирования связей базы данных и языка Eloquent, в которую встроена защита от SQL инъекций. Также возможность экранировать теги позволяет организовать дополнительную защиту от XSS-атак. Есть встроенная функция генерации CSRF токена для каждой сессии и многое другое.
* Мультиязычность. Возможность использовать несколько языков для вашего проекта.
* Масштабирование. Огромное количество пакетов, которые пригодятся для решения различных задач, которые могут возникнуть в будущем.

Недостатки:

* Обновления и совместимость. При обновлении Laravel могут возникнуть различные проблемы с пакетами и зависимостями, на решение которых может уйти приличное количество времени.
* Зависимость от сторонних пакетов. Как и в предыдущем пункте, если какой-то из пакетов перестанет поддерживаться или будут проблемы с зависимостями, вам нужно будет потратить какое-то время на решение этих проблем.
* Объем кода. Имея такой огромный потенциал к масштабированию, можно предположить какой объем кода может содержаться в этом проекте, что в конечном итоге может повлиять на производительность

Но недостатки есть в каждом фреймворке, поэтому несмотря ни на что Laravel остается одним из самых популярных.

Начало работы.

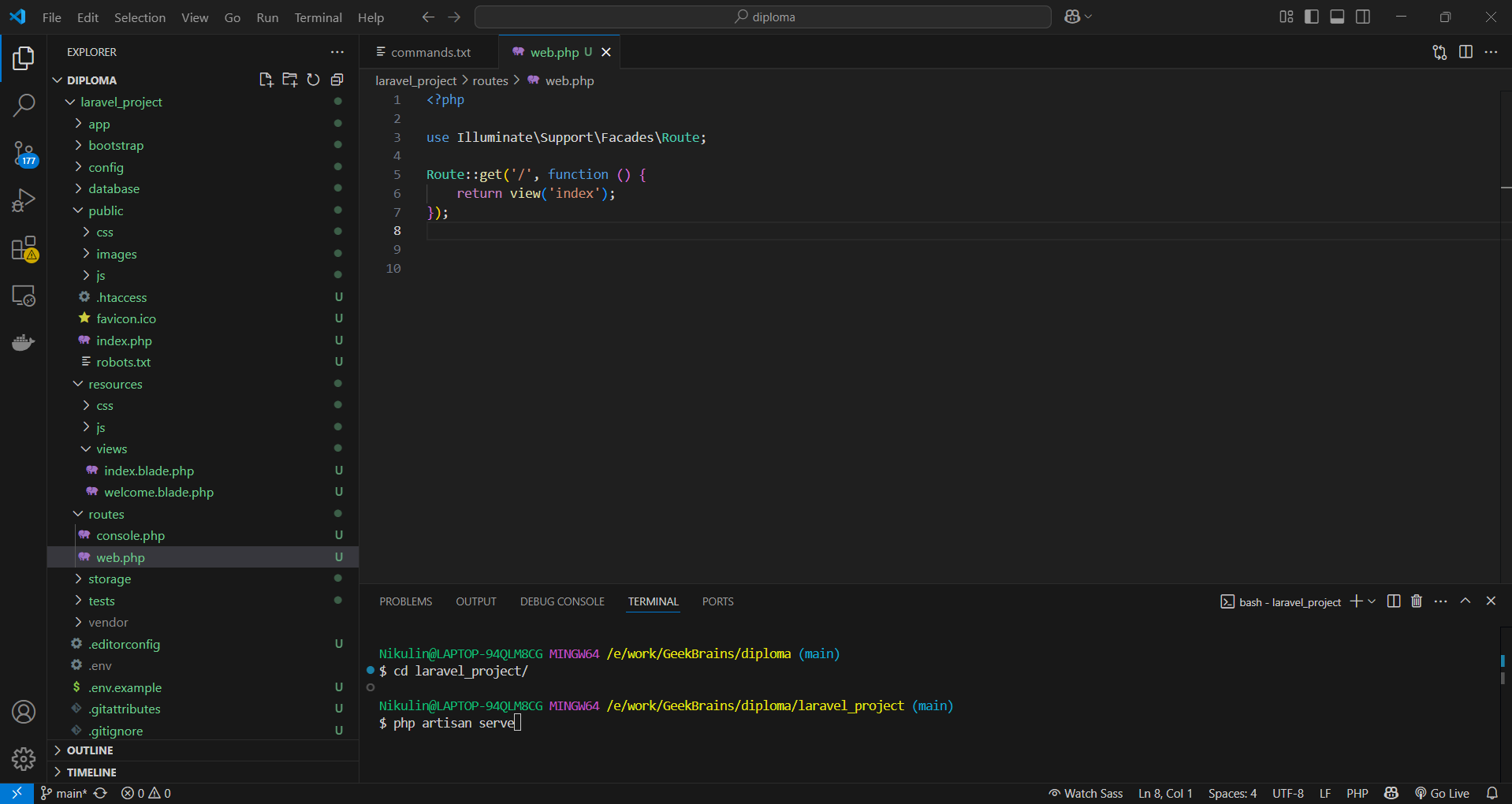
Так как изначально сайт создавался вне системы Laravel, нам надо заняться переносом всех ресурсов.

Переносим файл html в папку resources/views и переименовываем его под стандарт шаблонизатора blade (index.blade.php). Blade – это движок для шаблонов, в котором можно использовать код PHP. Он изначально входит в Laravel.

Также необходимо перенести остальные ресурсы, которые использует сайт: картинки, видео, скрипты, стили и прочее. Для этого нам понадобится директория public/. Создаем соответствующие папки: images, js, css.

Можно проверить корректно ли все отображается или нет. Прописываем маршрут в файле web.php. Это простейший маршрутизатор, в котором вы можете указать конечный маршрут и прописать какую-либо логику в функции. На скриншоте можете обратить внимание на структуру файлов слева.

Пишем команду $ php artisan serve и проверяем. Если наша страница отображается корректно, переходим к следующим шагам.

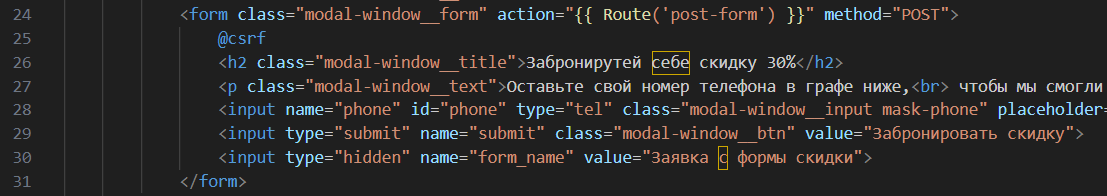


Необходимо добавить csrf-токены к каждой нашей форме на сайте. Они защитят пользователя и наше приложение от csrf атак (cross-site request forgery – межсетевая подделка запроса). Laravel генерирует токены автоматически для каждой пользовательской сессии.

Для этого внутри тега head вставляем:

    <meta name="csrf-token" content="{{ csrf\_token() }}">

И в каждую форму необходимо добавить скрытое поле @csrf.



Если вы этого не сделаете, то каждый раз при отправке формы будете получать ошибку. На этапе разработки можно исключить проверку на csrf-токен, для этого нужно вписать исключение в файле: vendor\laravel\framework\src\Illuminate\Foundation\Http\Middleware\VerifyCsrfToken.php

Но так делать не рекомендуется, это подвергает риску ваш проект, поэтому лучше изначально делать все правильно.

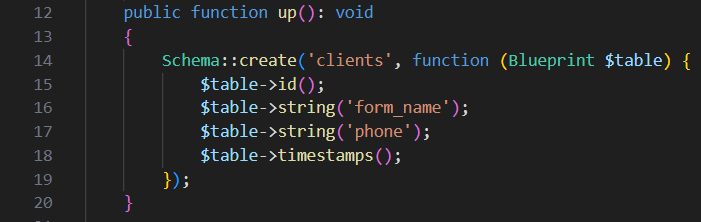
В нашем случае на странице размещены простые формы с единственным полем для номера телефона и одна форма, где помимо телефона, пользователю предлагается прикрепить эскиз или фотографию.

То есть в базе данных для наших клиентов будет два поля: название формы, с которой пришла заявка и номер телефона, также 3 стандартных поля id, crated\_at и updated\_at.

Создаем сразу несколько сущностей для дальнейшей работы командой:

$ php artisan make:model Client -mfsc

Artisan создаст модель, контроллер, фабрику, миграцию и наполнитель базы данных. О фабрике и наполнителях поговорим позже. Сейчас нам надо заполнить файл миграции.



Здесь добавляем необходимые поля, которые будут созданы в новой таблице. Все изменения можно зафиксировать командой:  
$ php artisan migrate

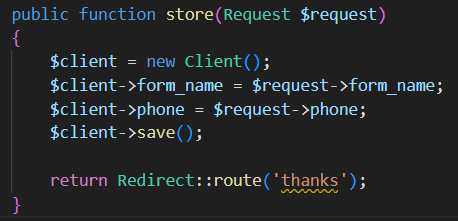
Создастся таблица с указанными полями. Далее нужно проработать модель клиента. Прописываем те поля, которые будут приходить после отправки формы из Request.

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

Тут все довольно просто, так как нам нужны только два поля из формы. Поля id, created\_at, updated\_at содаются автоматически.

Далее контроллер. Создаем функцию index(), которая отображает наш шаблон blade. Также нам понадобится функция, записывающая приходящие данные. Назовем её store(). Нам понадобится класс Request – способ взаимодействия с HTTP запросами.



1. Создаем экземпляр класса Client
2. Прописываем в поля необходимую информацию с помощью Request
3. Сохраняем изменения.
4. Отправляем пользователя на страницу «спасибо за заявку»

Стоит упомянуть, что, если вы используете VSCode и у вас нет специальных расширений, чтобы используемые классы автоматически подключались к вашему файлу, придется прописывать все вручную.

Изображение выглядит как текст, Шрифт, снимок экрана, Графика

Контент, сгенерированный ИИ, может содержать ошибки.

Можно было бы использовать PHP Storm для работы в Laravel, но он платный, поэтому обходимся тем, что есть. Прописываем используемые классы. Иногда их придется поискать и потратить время.

Следующим шагом нам стоит прописать пути к шаблонам. Заходим в уже знакомый файл web.php и добавляем роутинги.

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

1. Подключаем контроллер клиента и создаем get событие с ранее созданным классом index, который отображает наш шаблон.
2. Создаем событие POST. Оно будет срабатывать при отправке любой из форм.
3. Создаем роутинг для страницы спасибо.
4. Задавать имена для каждого роутинга не обязательно, но так работать гораздо удобнее.
5. Переходим в наш index.blade.php и прописываем в формы action. То есть то, что будет происходить после отправки формы.

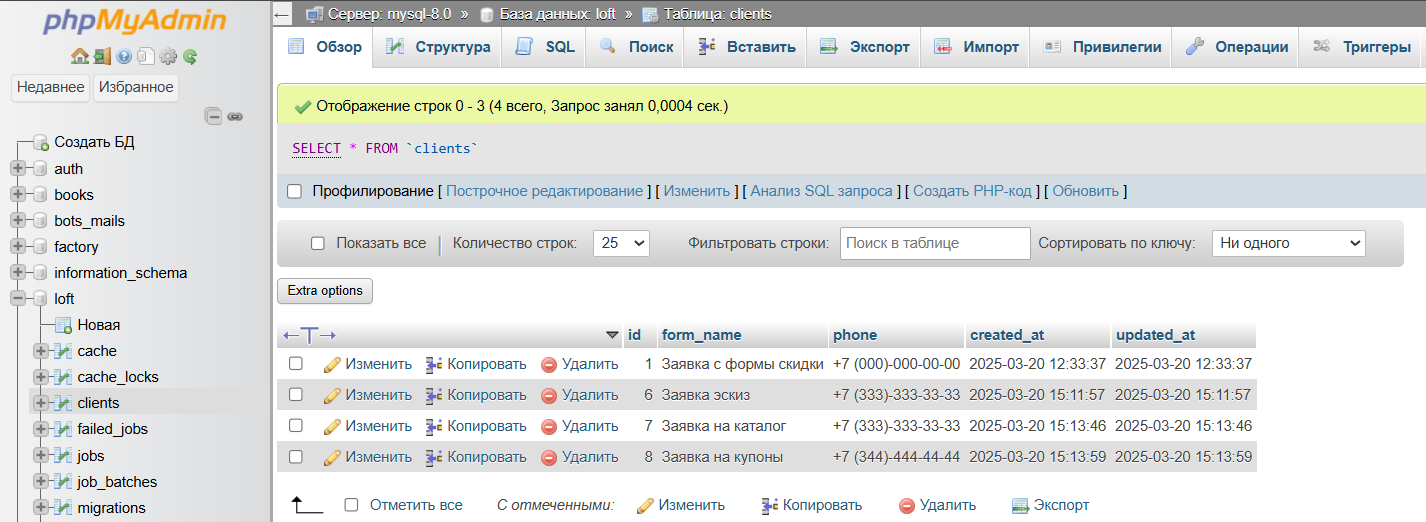
action="{{ Route('post-form') }}"

В данном случае мы указываем на имя события, которое до этого задали в web.php.

1. Теперь необходимо очистить кэш роутинга, иначе будем получать ошибку раз за разом. Пишем в командную строку:

$ php artisan route:cache

После этого можно запустить сервер и проверить работу наших форм, а также как это все записывается в базу данных.

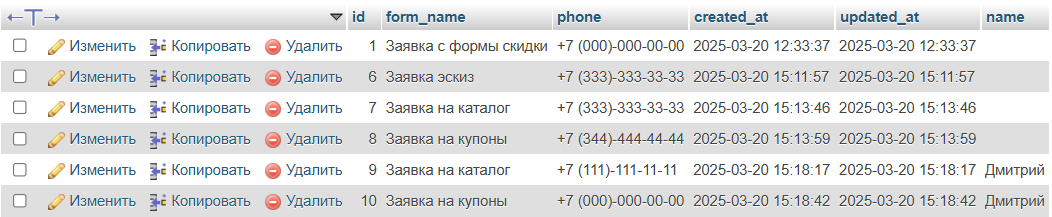


Данные для связи с клиентом сохраняются у нас на сервере и в дальнейшем мы сможем создать новую таблицу, где будем дополнять информацию о клиенте и таким образом можем делать в дальнейшем интересные предложения, например.

Думаю стоит упомянуть, что я забыл добавить в таблицу колонку с именами, потому что есть формы, где требуется ввести имя. Чтобы добавить новую конку необходимо создать миграцию с привязкой к нужной таблице:

$ php artisan make:migration put\_field\_name --table=clients

Далее добавить поле в модель клиента, а также в функцию store(), которая обрабатывает HTTP запросы. При этом есть формы, где нет поля имени, поэтому такой вариант надо обработать, иначе будет вылетать ощибка. И все готово.

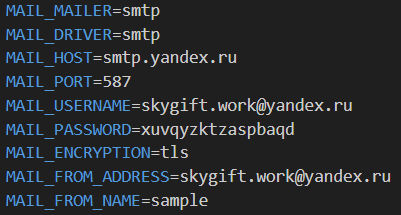


Далее сделаем удобную нотификацию, чтобы менеджер вовремя получал уведомление о полученной заявке.

**Настройка почтовых писем для уведомления.**

Для начала сделаем уведомления по почте через сервис Яндекс. В Laravel есть встроенный инструмент для отправки сообщений SwiftMailer.

1. Создаем почтовый аккаунт в яндекс
2. Прописываем конфигурационные настройки в файле .env



1. Переходим в настройки почтового клиента <https://mail.yandex.ru/setup/>

И разрешаем доступ к почтовому ящику с помощью почтовых клиентов.

Переходим в раздел безопасности и находим пароли приложений

<https://id.yandex.ru/security/app-passwords>

Затем выбираем почта, придумываем имя паролю и нам высвечивается автоматически сгенерированный пароль, его лучше куда-нибудь записать.

Он как раз записан в MAIL\_PASSWORD файла .env.

1. Переходим в наш редактор кода и создаем класс при помощи artisan командой

$ php artisan make:mail BidFromSite

После этого в папке App создастся соответствующий класс.

1. Нам нужен шаблон blade для письма. Своего рода это тело или же контент, который мы отправляем конечному пользователю (в данном случае себе). Создаем letteronmail.blade.php в папке resources/views

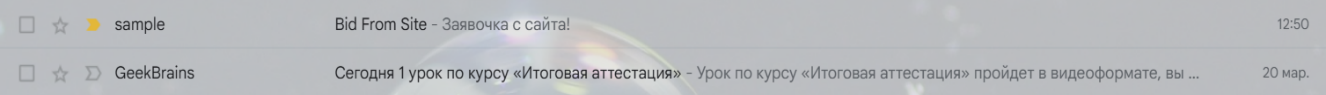
Пока что пишем простой заголовок, что пришла заявка.

1. Переходим в наш Mail класс и в функции envelope() указываем тему письма. А в функции контент ссылаемся на ранее созданный шаблон.
2. Остается прописать действие в роутинге на странице thanks.

 $email = 'skygift.work@gmail.com';

 Mail::to($email)->send(new BidFromSite());

Используем свою почту и с помощью фасада Mail отправляем письмо. Также не забываем очистить кэш роутинга.



И мы получаем заветное письмо, что по сути является для нас одним из способов уведомления.

Ghtbv

Список источников:

Документация по Laravel Git и прочим инструментам.

«Head First HTML and CSS» Робсон Элизабет и Фримен Эрик

Изучаем программирование на JavaScript Робсон Элизабет и Фримен Эрик

<https://ru.hexlet.io/blog/posts/pochemu-laravel-odin-iz-luchshih-php-freymvorkov-dlya-startapov-i-enterprayza>

<https://www.qmedia.by/blog/laravel.html>

<https://blog.skillfactory.ru/glossary/laravel/>

<https://laravel.su/docs/11.x/blade>

https://laravel.su/docs/11.x/routing

<https://education.yandex.ru/journal/chto-takoe-react-i-kak-ego-osvoit>

<https://sass-scss.ru/guide/>

<https://gb.ru/blog/github-nastrojka/>

https://habr.com/ru/articles/439200/

<https://delphinpro.ru/a/6> - phpmyadmin

<https://code.tutsplus.com/ru/build-a-react-app-with-laravel-restful-backend-part-1-laravel-5-api--cms-29442t>

ЧТО ЕЩЕ МОЖНО НАПИСАТЬ

- Маска для ввода телефона

-Что то сделать с картинками

- Переделать кнопки

- Валидация форм

- Мобильные версии

- Рассказать про отладочную панель

- Больше информации про React

- больше информации по Laravel

- Рассказать больше про csrf

- разобрать ключи в файле енв