Geekbrains

**Погружение в деятельность fullstack-разработчика.**

**Создание сайта по заданному макету и техническому заданию при помощи фреймворка Laravel.**

Программа: Программист

Специализация: Fullstack-разработчик

Никулин Дмитрий Сергеевич

Санкт-Петербург

2025 год

**Содержание**

Введение – 2 стр

**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 прочтения мне захотелось углубиться в программирование на языке JavaScript. И следующей книгой стала «Изучаем программирование на JavaScript» за авторством тех же писателей. Но после недолгих практических шагов в JS, жизненные обстоятельства заставили на какое-то время отстраниться от этого.

До этого я успел поработать со многими людьми и разработать несколько простых сайтов, которые назывались посадочными страницами или landing page. По сути, одностраничный сайт-визитка без особого функционала.

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

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. Сайт, который я делал в 2018 году для одного из своих клиентов.

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 можно обойтись без таких функциональных фреймворков, но в качестве презентации своих навыков и умений, как fullstack-разработчика – отлично подойдет, тем более придется много всего затронуть. 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. На текущий момент он продолжает этим, но сам сайт уже изменил. Во избежание недопониманий и просто на всякий случай я заменю всю контактную информацию и некоторые фотографии заглушками. Все созданное в проекте является моей интеллектуальной собственностью (помимо фотографий), и я считаю допустимым непубличное использование веб-сайта в качестве основы моего дипломного проекта.

**Цель проекта**

Создать веб-сайт, через который посетитель сможет ознакомиться с продукцией и оставить заявку. Занести информацию о клиенте в базу данных для дальнейшей передачи менеджеру. Разработать систему уведомлений для оповещения менеджеров.

**План проекта**

1. Разработать структуру и дизайн веб-страницы.
2. Сверстать страницу на HTML, в дальнейшем это будет blade-шаблон
3. Оформить страницу с помощью CSS, использовать Sass для удобства.
4. С помощью JavaScript создать логику взаимодействия со стороны посетителя.
5. Подключить базу данных для работы с пользователями, а в будущем с товарами.
6. Разработать систему нотификации.

**Инструменты**

* Adobe Photoshop – для работы с макетом
* Visual Studio Code – удобный редактор кода для всех языков
* Git – система управления версиями
* Laravel 11.0 – бесплатный веб-фреймворк
* PhpMyAdmin – интерфейс для работы с базой данных
* Composer – менеджер пакетов

**Состав команды**

* Маркетолог
* Копирайтер
* Дизайнер
* Frontend-разработчик
* Backend-разработчик
* Fullstack-разработчик (вместо двух предыдущих)

В качестве вышеперечисленных работников буду выступать непосредственно я сам.

**Глава 1. Задачи fullstack-разработчика.**

**1.1 Кто такой fullstack-разработчик и какие у него задачи**

Fullstack-разработчик – это специалист, который воплощает в себе функции двух других специалистов frontend-разработчика и backend-разработчика.

Frontend занимается разработкой клиентской части приложения. Грубо говоря, это то, что вы видите через свои мониторы и тот функционал, с которым непосредственно можете взаимодействовать: картинки, видео, кнопки, формы и т. д. Фактически frontend-разработчик в среде веб-сайтов занимается версткой страницы на HTML, задает этой странице внешний вид (желательно по дизайн-макету, который создает уже другой специалист – веб-дизайнер). За внешний вид отвечает каскадная таблица стилей CSS. Следующий задачей разработчика является разработка функционала с помощью JavaScript. В нем можно реализовать любую логику, а это переключение картинок, всплывающие окна, анимации, подключение сторонних плагинов. Что не менее важно, с помощью JS клиентская часть приложения генерирует условный контент на основе приходящей информации из сервера.

Backend-разработчик занимается тем, что конечный пользователь не увидит. Это серверная часть веб-приложения. Обычно бэкенд-разработчики заняты написанием серверного кода, разработкой логики работы приложения, и поддержанием его инфраструктуры. В их обязанности может входить множество вещей, в зависимости от конкретного приложения: оптимизация кода, работа с базой данных, настройка API, отладка и написание тестов.

Fullstack-разработчик совмещает в себе все эти элементы и отвечает за обе части веб приложения: клиентскую и серверную. Обычно на них лежит огромный объем работы, но при этом в молодых компаниях они пользуются спросом, потому что можно завербовать специалиста 2 в 1.

**1.2. Необходимые инструменты для разработчика.**

Чтобы эффективно работать и не тратить свое время на выполнение простых и рутинных вещей, разработчику требуется установить некоторый стек инструментов.

Для начала нужен редактор кода, в котором будет происходить большая часть работы. Вряд ли кто-то в здравом уме будет писать код в текстовом редакторе. Какой выбрать зависит только от предпочтений разработчика и языка программирования, на котором будет происходить работа. Самый универсальный – это Visual Studio Code. Он не тяжеловесный, подходит для большинства языков программирования и имеет достаточно хорошую библиотеку расширений, которые облегчат работу программиста.

Для управления версиями приложения понадобится такая система, как Git. О ней подробнее в следующей главе.

В моем случае понадобится менеджер пакетов Composer. Он выполняет функцию загрузки необходимых пакетов для работы приложения. Представим, что для работы приложения необходимо двадцать файлов с кодом и все они находятся в разных местах интернета. Конечно, можно потратить время и качать каждый файл по отдельности. А можно использовать composer-файл, в котором прописаны все зависимости что и откуда брать и как установить. Поэтому одной командой можно освободить себе кучу времени.

Также необходимо будет установить графический интерфейс для работы с базами данных PhpMyAdmin. Можно управлять базой данных и через консоль, но так гораздо удобнее. Есть и альтернативные приложения по типу MySQL Workbrench, но тут уже на вкус и цвет.

Еще понадобится Open Server — это портативный локальный WAMP/WNMP сервер, имеющий многофункциональную управляющую программу и большой выбор подключаемых компонентов. Так как при разработке внешнего сервера скорее всего нет, необходимо его чем-то заменить, в конкретном случае локальным сервером.

**1.3 Перед началом любой работы. GIT.**

Прежде, чем говорить о какой-либо разработке стоит задуматься о какой-нибудь системе бэкапов и задаться правильными вопросами.

Как контролировать процесс разработки? Иметь возможность быстро откатить изменения при необходимости? А если над проектом трудится не один человек, как сравнивать и анализировать или сливать изменения между разработчиками? Решением этих задач является система контроля версий Git.

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, 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).

Один коммит — это пакет изменений, хранящий информацию с добавленными, отредактированными или удалёнными файлами кода. В один коммит принято добавлять не более десяти изменений — так получается длинная история версий, которая позволяет в случае ошибки откатиться с минимальной потерей работоспособного кода (см рис. 1 в приложении).

Таким образом при любой непредвиденной ситуации можно избавить себя от лишний мороки, начав все с последней работающей версии.

**1.4 Что такое HTML и зачем он нужен?**

Вся работа так или иначе начинается с идеи, которая перерастает в прототип, которая перерастает в дизайн-макет. Так как фронтенд-разработчику приходится совмещать все виды работ, он должен уметь верстать страницы. Для этого существует HTML.

HTML (Hyper Text 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 в интернете 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.

HTML сам по себе не является языком программирования, как может быть указано на различных ресурсах, однако это важнейший инструмент разметки с большим разнообразием тегов (см. рис. 2 в приложении). Все страницы, что можно увидеть в интернете – это HTML-страницы с подключенными к нему стилями CSS.

**1.5 CSS и Sass. Как раскрасить веб-сайт.**

В базовом виде HTML разметка не представляет из себя веб-сайты, которые мы привыкли видеть, потому что это не задача HTML. Чтобы заставить страницу выглядеть приемлемо нам нужен другой инструмент.

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 итоге должна выглядеть страница по принципу свойство: значение. Этих самых свойств или параметров в CSS огромное количество и можно управлять практически каждой визуальной составляющей: цвет, форма, тени, расположение, размер и даже несложная анимация. Можно сказать, что CSS3 перетянул на себя некоторый функционал JavaScript в плане анимации. Теперь, даже не зная языка программирования можно делать очень красивые проекты. Но прогресс не стоит на месте и появились такие вещи как препроцессоры, такие как Sass.

Sass — это метаязык (язык для описания другого языка), который упрощает и ускоряет написание CSS-кода. Его называют препроцессором. То есть у Sass свой удобный синтаксис, который генерируется в обычный CSS. Благодаря препроцессорам код стилей стал выглядеть еще более удобно, потому что даже в самом простом файле CSS может быть уйма правил.

Sass позволяет создавать переменные и использовать их в любом участке кода, делать вложенные стили, миксины (примеси - по своей сути тоже является переменной, но внутри нее целый участок кода CSS), использовать фрагментирование и множество других функций, упрощая разработку и делая ее очень удобной, а главное ёмкой и легкой для чтения.

**1.6 Javascript. Оживи сайт.**

JavaScript - это легкий, интерпретируемый, объектно-ориентированный язык программирования данных, основанный на прототипах и первоклассных функциях.

Изначально JavaScript был создан, чтобы описывать функционал для веб-страниц, тем самым делая его более «живым». Программы на этом языке называются скриптами. Они могут встраиваться в HTML и выполняться автоматически при загрузке веб-страницы. Скрипты распространяются и выполняются, как простой текст. Им не нужна специальная подготовка или компиляция для запуска, они обрабатываются непосредственно браузером внутренним «движком». С помощью сценариев JavaScript можно включать в статическую страницу динамические элементы, такие как карты, формы, числовые операции, анимацию, интерактивную инфографику и другое.

Преимущества JavaScript:

* скорость чтения и исполнения;
* понятный синтаксис;
* его не нужно компилировать - браузеры его интерпретируют;
* совместимость с широким спектром браузеров и платформ;

Сейчас JS не только браузерный язык, который изначально так задумывался. С появлением различных фреймворков, API, улучшений и созданием сотен функций, можно использовать JavaScript в мобильных приложениях, десктопном софте и даже в бэкенде.

Также с языком JS связаны популярные фреймворки по типу Vue или библиотека 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.

Еще существует библиотека jQuery, ныне устаревшая из-за более популярных фреймворков, но в свое время использовалась довольно часто.

Все это говорит о том, что большинство решений можно найти, даже хорошо не зная JavaScript, а изучать его по лекалам других разработчиков.

Вот мы и рассмотрели трех китов создания веб-сайта. Однако при написании кода нужны и другие инструменты, такие как отладчик в браузере.

**1.7 Отладка написанного кода в браузере**

Отладка – это процесс поиска ошибок и, говоря о наших «трех китах» веб-сайта, я имею в виду поиск ошибок непосредственно в HTML, CSS и JavaScript. Все браузеры имеют графический интерфейс отладчика (см рис. 3 в приложении), с помощью которого можно смотреть что и каким образом обрабатывается браузером, что сильно упрощает процесс разработки. Чем же отладочная панель так полезна?

* Можно посмотреть, как элементы выглядят в HTML файле;
* Какие стили работают для конкретного файла или всего сайта;
* Какие ресурсы загружает браузер;
* Есть встроенная консоль, помогающая с работой в JS и не только;
* Можно пошагово отследить работу исполняемого скрипта;

При разработке к ней обращаются очень часто. В некоторых ситуациях можно написать HTML или CSS прямо в отладочной панели, чтобы отследить как задумка будет работать и будет ли работать вообще.

**1.8 Фреймворк 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 (см. рис. 4 в приложении)

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 (модель) – данные и методы работы с этими данными (проверки и запросы в базу данных)
* View (представление) – внешний вид. То, с чем взаимодействует конечный пользователь.
* Controller (контроллер) – 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 можно 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 есть возможность экранировать теги, что 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 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, 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.

**1.9 Структура каталогов Laravel.**

При установке Laravel приложения можно столкнуться с довольно внушительным каталогом файлов (см. рис 5 в приложении). Всю структуру можно настроить как нравится конкретному разработчику, так как Laravel не накладывает на это ограничений, однако Composer должен иметь возможность автоматически загружать классы.

Каталог app содержит основной код все классы архитектуры MVC будут находиться в нем.

Каталог bootstrap содержит файл app.php, который загружает фреймворк. В этом каталоге также находится каталог cache, содержащий файлы, сгенерированные фреймворком для оптимизации производительности, например файлы кеша маршрутов и служб.

Каталог config содержит все файлы конфигурации приложения.

Каталог database содержит миграции баз данных, фабрики моделей и наполнители.

Каталог public содержит файл index.php, который является точкой входа, также сюда размещаются стили, картинки и скрипты.

Каталог resources содержит шаблоны, о них в следующей главе.

Каталог routes содержит все определения маршрутов. Простыми словами ссылки сайта и их обработка (тело функции).

В файле console.php можно определить анонимные консольные команды и задачи.

Каталог storage содержит журналы (логи), скомпилированные шаблоны Blade, файлы сессий, кеша и другие файлы, созданные фреймворком.

Каталог tests содержит автоматизированные тесты. Примеры модульного Pest или PHPUnit и функционального тестов предоставляются из коробки.

Каталог vendor содержит ваши Composer-зависимости.

**1.10 Blade шаблоны.**

Blade – это движок шаблонов, входящий в состав Laravel. Можно сказать, что это html структура с вкраплением PHP кода, который исполняется при работе приложения. На самом деле, все шаблоны Blade компилируются в обычный PHP-код и кешируются до тех пор, пока не будут изменены, что означает, что Blade добавляет фактически нулевую нагрузку приложению. Файлы шаблонов Blade используют расширение файла «.blade.php» и хранятся в каталоге resources/views.

**Глава 2. Создание проекта с нуля. HTML/CSS/JavaScript**

**1.1 Создание репозитория в GitHub.**

Перед началом любой работы, помимо того, что нужно установить необходимые инструменты, необходимо инициализировать систему контроля версий. В нашем случае – это GitHub.

Чтобы подключить Git к своему проекту не понадобится много времени. Для начала регистрируемся на сервисе GitHub и устанавливаем на компьютер, после этого на сайте github создаем новый репозиторий, настраиваем под свои задачи, а после создания сам сервис подскажет что дальше делать (см. рис. 6 в приложении). Есть альтернативный способ это сделать напрямую через терминал VSCode, кому как удобнее. Далее главное не забывать «коммитить» изменения через определенный промежуток времени.

**1.2 Верстка сайта на HTML.**

Чтобы начать верстать сайт необходим дизайн-макет (см. рис 7 в приложении). То файл, где элементы уже расположены в нужном порядке и уже с заданными стилями. Обычно созданием макетов занимается дизайнер и он никак не причастен к написанию кода. В последнее время для создания таких макетов используют такие веб-приложения как Figma. В моем проекте я использовал Adobe Photoshop, что не очень удобно, потому что он довольно ресурсоемкий.

Заходим в редактор кода и начинаем с базовой разметки HTML. Для облегчения и скорости работы понадобятся некоторые расширения:

* HTML5 Snippets – подсказки при написании кода.
* JS-CSS-HTML Formatter – форматирует код при сохранении.
* Emmet – заполнение «рыбным текстом» (Lorem ipsum).
* Sass – препроцессор для CSS (см. главу 1.5)

Рассмотрим основные теги, которые используются в проекте. Все начинается с тега <html lang=”ru”>, который и сообщает браузеру формат документа и язык. Все дальнейшая разметка должна быть исключительно в этом теге.

<head> - здесь располагаются метаданные с тегами <meta> (описание страницы, ключи для ранжирования в поиске, кодировка), заголовок страницы <title>, через тег <link> подключаются шрифты и стили. Иногда сюда, при определенных случаях, можно вставить и ссылку на какой-либо скрипт <script>. В моем случае была подключена Яндекс Метрика. Но скрипты принято размещать перед закрывающим тегом </body>, чтобы сайт у пользователя загружался быстрее, поэтому здесь скорее исключение из правил.

<body> - основное тело нашей страницы. Здесь размещается весь контент, а это такие теги, как заголовки разного уровня (<h1>, <h2>, <h3> и т.д.), параграфы (<p>), ссылки (<a>), картинки (<img>), видео (<video>), аудио <audio>, списки (<ul>, <li>) и прочие.

Отдельно стоит сказать, что с версией HTML5 появились новые теги, которые лучше раскрывают структуру страницы. Например, тег <main>, в теле которого лучше всего располагать уникальное содержимое страницы. Это содержимое должно относиться к главной теме страницы и быть уникальным.

Также в проекте используются такие теги как <header> и <footer>, которые предназначены для шапки сайта и подвала. Иначе говоря, верхний и нижний колонтитул. Стоит сказать и про тег <section>. Он представляет собой общий автономный раздел документа, который не имеет более конкретного семантического элемента для его представления. Хотя никто не запрещает использовать <div> (пустой контейнер), однако все новые атрибуты помогают браузерам лучше ранжировать сайты в поиске, а также грамотно структурировать документ.

Еще один важный тег <form> и вложенные в него теги <input>. С их помощью задаются формы на сайте, которые передают информацию на сервер методом HTTP запроса POST.

У большинства тегов есть свои атрибуты – это дополнительные 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 нужно использовать практически для каждого элемента это class: <a href=”#” class=”some-class”/>

Атрибут class присваивает идентификатор элементу, по которому можно обращаться в CSS и задавать необходимые стили. 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, сразу стоит продумать названия классов для будущей работы со стилями. Мы будем использовать препроцессор Sass, и чтобы делать вложенную структуру, названия классов должны иметь определенный вид: classname\_\_elementname (см рис. 8 в приложении). Получается некая древесная структура, где «main» – корень, а приписки вида «\_\_header» – ветви. Отчетливо это будет видно в файле Sass. Вообще вся верстка html идет параллельно с написанием стилей, поэтому переходим к следующему пункту. Полный html файл можно посмотреть в папке loft\_new.

**1.3 Задание стилей для элементов. CSS\Sass.**

В CSS мы начинаем прописывать правила для элементов по схеме «свойство: значение». Описывать их все не имеет смысла, потому что их очень много. Но можно сказать, что если вам что-то придет в голову в плане визуализации элементов, то вероятнее всего это можно сделать с помощью стилей. Рассмотрим самые часто встречающиеся свойства.

Для шрифтов существуют точно такие же свойства как в любом текстовом редакторе. Это самое правильное описание, потому что для любого текста можно задать семейство шрифтов, размер, толщину. Все эти параметры задаются через параметр font, а именно font-family, font-size, font-weight. Значения этих свойств задается в пикселях или в других относительных единицах (об этом чуть далее). Параметры выравнивания - text-align (значения такие как center, right, left), цвета – color (значения либо по названию white, либо кодом #ffffff).

Но одними из самых основных свойств являются те, которые отвечают за расположение контента на странице. Это отдельная и объемная тема. Во-первых, у каждого элемента на сайте есть габариты, то есть длина и высота (width, height). Во-вторых, каждый элемент имеет свои отступы относительно другого элемента. Существуют внутренние отступы padding и внешние отступы margin. Значения у таких параметров задаются какой-то единицей измерения. Существует два вида:

* Абсолютная единица (пиксели px, сантиметры cm, миллиметры mm)
* Относительная единица (em – размер относительно родителя, проценты %)

Из абсолютных единиц в веб разработке используют пиксели, остальные едины скорее для печати. А вот относительных единиц много и их можно использовать, но для этого нужно грамотно продумать структура своей страницы. Все относительные единицы, как можно догадаться, заданы относительно родителя. Например, если родитель сто пикселей, то дочерний элемент с параметром пятьдесят процентов будет иметь размер пятьдесят пикселей. Если всерьез продумать структуру, то можно создать страницу, которая будет адаптировать все размеры относительно ширины или высоты экрана устройства.

На самом деле вышеперечисленные параметры составляют объемную часть всего кода. Но стоит упомянуть и параметр position. Он задает элементу позиционирование. Может быть статичное, то есть так как и расположено в html документе, relative – позиция относительно родительского элемента, absolute – «вырывает» из потока html элемент, и он находится как бы вне структуры. Регулировать положение элементами можно свойствами right, left, top, bottom.

Думаю, что следует упомянуть про свойство float, которое указывает, что элемент должен быть исключен из потока и располагается в левой, правой части родительского элемента (right, left). Если посмотреть проект, который я писал в 2018 году (он находится в папке loft\_old\_2018), можно увидеть, что все элементы расположены в документе именно таким образом. Откровенно говоря, это не очень удобно, потому что есть более продвинутое свойство display: flex. Но видимо в книге, которую я прочитал на тот момент, его не было, иначе не вижу причин использовать float. Поэтому, модернизируя проект, я использовал flex.

CSS Flexbox — это технология для создания сложных гибких макетов за счёт правильного размещения элементов на странице. И она будет использована в большинстве случаев при позиционировании элементов. Задается она параметром display:flex родительскому элементу. После этого открывается множество настроек, которые действуют на дочерние. Такие как justify-content с параметрами flex-start, flex-end, center, space-between, space-around, которые зададут выравнивание по главной. Flex-direction задает направление по перпендикулярной оси, gap задает одинаковые отступы между элементами.

Flexbox самый универсальный способ выравнивания не сложных систем. Но также упомяну display: grid. Он не используется в проекте, но вкратце, это свойство пришло на замену табличным выравниваниям (table), но в отличие от таблиц grid предоставляет возможность работать с каждой ячейкой по отдельности.

Вышеперечисленные параметры покроют 80% кода в css, но работа происходит в Sass, поэтому можно увидеть, как вложенная структура делает код визуально более приятным (см. рис. 9 в приложении). У нас есть основной класс main и все свойства, что описаны внутри этого блока, касаются конкретного блока в html.

Также с помощью препроцессора задаем переменные для основных цветов сайта. Это дает некоторую экономию времени, если в будущем придется изменить цвета. Не придется искать необходимое свойство в семиста строках кода, а заменить только значение переменной.

Таким образом описываем каждый блок с дизайн макета и получаем практически готовую веб-страницу.

**1.4 Прописываем логику взаимодействия. JavaScript.**

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

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

Необходимо создать всплывающее окно, которое появится при нажатии кнопки.

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 модальное окно в HTML в любой части страницы как показано на рисунке 10 (см. в приложении). Из важных стилей для этого окна нам понадобится свойство visibility с параметром hidden. Оно скроет из виду этот блок. Далее необходимо подключить скрипт к проекту и сделать три вещи (см. рис 11 в приложении):

1. Найти все кнопки, которые нам нужны с помощью метода querySelector (возвращает первый элемент, соответствующий указанному параметру) и задать переменные.

2. Добавить кнопкам событие клика с помощью функции addEventListener (добавляет элементу действие, которое будет выполнено после срабатывания события. Например, на клик мышки или нажатие клавиши).

3. Добавить или убрать класс hidden с помощью метода classList, то есть сделать переключатель.

Элементарная логика. Однако, в проекте больше нет ничего сложного, что бы требовало обработки скриптами. Поэтому было принято решение немного поразмыслить на перспективу.

В более сложных проектах JavaScript строит страницу на основе базы данных. Допустим, что в базе данных у нас существует таблица с товарами, где есть цена, указан путь к картинке и написано описание товара. Эти данные будут периодически обновляться и, если говорить честно, вряд ли захочется при каждом изменении лезть в html код и добавлять это все вручную.

Поэтому я создал отдельную страницу example.html в качестве примера реализации, чтобы не рушить целостность структуры основной страницы. В самом коде html практически пусто. Существует некий заголовок с подзаголовком и пустой контейнер для размещения товаров. Разумеется, при проектировании я заранее сверстал то, как будут выглядеть карточки товаров, чтобы задать стили и опираться на них при написании скрипта, но в конечной версии тут пусто.

Допустим, что товары из базы данных нам приходят в самом популярном формате – JSON (см. рис 12 в приложении). Я разместил этот файл непосредственно в папке scripts. Чтобы разместить все товары надлежащим образом необходимо превратить json-файл в объект JavaScript. Это делается методом JSON.parse(). JSON.stringify совершает обратное действие. После преобразования с этим объектом можно работать. Также нужно найти стартовую точку – ту, куда мы будем класть карточки товаров. Делаем известным методом document.querySelector().

Далее каждый элемент объекта надо добавить на страницу. Для этого используется цикл forEach. Он перебирает каждый элемент списка. И для каждого элемента списка необходимо задать правила:

1. Создаем контейнер <div> с помощью метода document.createElement(‘div’)
2. Добавляем контейнеру необходимый класс со стилями с помощью известного метода classList.add (поэтому заранее создается шаблон карточки, чтобы в файле CSS уже были прописаны правила и оставалось только подключить класс с этими стилями, а не писать каждый стиль в javascript).
3. Создаем элемент картинки, задаем класс и в атрибут src нужно задать путь. Так как мы перебираем каждый элемент из JSON объекта, можно обратиться к ключу picture, значение которого и является путь к необходимому файлу.
4. Таким образом добавляем каждый нужный элемент и последним шагом требуется все эти файлы добавить на страницу с помощью метода appendChild (см. рис. 13 в приложении).

Таким образом у нас получается автоматически работающая система. Сколько бы товаров не добавил менеджер по товарам они будут оперативно обрабатываться. Все файлы можно посмотреть в папке loft\_new или в папке Laravel\_project.

Помимо реализации добавления товаров автоматически, я задумался о создании простой сортировки по убыванию и возрастанию цены. Если рассматривать проект на перспективу, то из него должен вырасти интернет-магазин. Это в идеале. А в каком магазине нет сортировки товаров?

Я решил вынести код в отдельный скрипт sortof.js.

* + - 1. Для начала добавляем 2 кнопки в html файл, которые будут отвечать за сортировку. Здесь это представлено в качестве примера, поэтому стили для них не заданы. В любом случае главное – это функционал, а подключить его можно будет к любому «красивому» элементу на странице.
      2. Находим кнопки в скрипте с помощью querySelector. Для начала пропишем логику для одной. Добавляем событие при клике на кнопку addEventListener(‘click’, function(){}).
      3. Если задумываться об алгоритмах, то есть множество примеров реализации сортировки. Можно много чего почерпнуть из книги «Грокаем алгоритмы». Создадим функцию sortAscending(). Мне пришла в голову следующая идея: создаем новый массив, перебираем каждый элемент базы данных и добавляем цену в отдельный список (так как у нас объект, сразу добраться до цены будет проблематично).
      4. Используем функцию sort, где в качестве функции задаем ей простое сравнение, после чего текущий массив будет отсортирован по возрастанию.
      5. Создаем еще один список, где будем хранить отсортированные объекты. После этого циклом for пройдемся по значениям в списке цен и по объектам, сравнивая цены. В конечном итоге получаем список с отсортированными объектами. C помощью return возвращаем список из функции. Он понадобится далее.
      6. Чтобы придерживаться целостной структуры, немного доработаем gridRealize.js. Всю логику действий по построению контента поместим в функцию, параметром которой она будет принимать список. Перед стартом цикла forEach удалим весть контент в блоке. Это нужно для того, чтобы при сортировке контент не накладывался друг на друга, а заменялся.
      7. Осталось только поместить gridRealize() в функцию sortAscending() (помним, что она возвращает список из объектов. При нажатии кнопки весь контент на странице перестроится.
      8. Чтобы добавить сортировку по убыванию, в событие для второй кнопки добавляем то же самое. Ведь список у нас уже отсортирован, нужно только его перевернуть. В этом поможет метод reverse().

Таким образом получена заготовка для работы с будущим каталогом. Итоговый скрипт можно посмотреть на рисунке 11 в приложении, а также в папке new\_loft/scripts или Laravel\_project/public/scripts.

**Глава 3. Установка и работа в Laravel.**

**3.1 Подготовка**

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

1. На всякий случай обновим сам composer командой: $ composer self-update

2. Создадим проект Laravel командой: $ composer create-project laravel/laravel . , где точка — это текущая директория.

Если возникают какие-либо ошибки с зависимостями или версиями пакетов, можно решить проблему установкой модуля fileinfo, который не установлен по умолчанию. Для этого необходимо открыть php.ini в папке, куда был установлен php и снять комментарий со строки: ;extension=fileinfo. После очередной установки или обновления все ошибки должный уйти, если нет существуют команды:

$ composer install --ignore-platform-reqs

$ composer update --ignore-platform-reqs

Эти две команды устанавливают/обновляют пакеты, игнорируя проблемы с зависимостями, однако это не панацея и проблемы могут возникнуть далее по ходу работы с проектом.

Следующим шагом нужно установить PhpMyAdmin с официального сайта. Нам понадобится настроить систему управления базой данных. 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. Можно выбрать и другую СУБД, например MySQL Workbrench или ограничиться SQLite. Но в проекте использовалась PhpMyAdmin, поэтому рассмотрим ее настройки.

После этого необходимо установить Open Server Panel – локальный веб-сервер и далее нужно заставить все исправно работать.

1. Переходим в директорию OSPanel/home и создаем папку phpmyadmin, и внутри еще две .osp и public.

2. В директории .osp создаем файл project.ini и добавляем в него код (см рис. 14 в приложении). Перезагружаем OSPanel, чтобы настройки вступили в силу.

3. Затем нужно скачать архив с последней версией PhpMyAdmin и распаковать их в ранее созданную папку: OSPanel\home\phpmyadmin\public\

4. Переимновываем файл config.sample.inc.php в config.inc.php и открываем.

5. Вводим случайную символьную строку в качестве секрета шифрования и создаем сервер (см рис. 15 в приложении).

6. Запускам OSPanel от имени администратора и включаем необходимые модули. Можно воспользоваться командами через терминал: osp on php-8.1, osp on mysql-8.0.

После этого в браузере по адресу phpmyadmin/ должна появиться административная панель, все параметры для входа описаны в config.inc.php. Теперь мы можем работать с базами данных. Возвращаемся к терминалу редактора кода.

Для удобства работы с Laravel предусмотрена система работы через командную строку под названием artisan. Artisan – это интерфейс командной строки, входящий в состав Laravel. Он предлагает ряд полезных команд, которые помогут при создании приложения. Для просмотра списка всех доступных команд Artisan можно использовать команду list.

Сгенерируем ключ приложения командой:

$ php artisan key:generate

Затем надо подключить базу данных, иначе при запуске локального сервера получим ошибку. Все настройки приложения можно найти в файле .env. Можно работать и без базы данных, при разработке проекта может возникнуть такая ситуация. Пока нет системы управления базой данных можно поменять SESSION\_DRIVER на file, вместо database. Или, как вариант, можно использовать SQLite, которая позволит работать с базой данных прямо из редактора кода. Но для VSCode придется установить определенные расширения для корректной работы: SQLite 3 Editor. Настройки базы данных представлены на рисунке 16 в приложении.

После всех настроек мы почти готовы к тестовому запуску проекта. 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

Добро пожаловать на приветственную страницу Laravel (см. рис 17 в приложении). Раз все загрузилось успешно и без ошибок, продолжим настраивать приложение.

**3.2 Перенос файлов и настройка форм.**

Так как изначально сайт создавался вне системы Laravel, 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 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 можно указать конечный маршрут и прописать какую-либо логику в функции (см. рис 18 в приложении).

Пишем команду php artisan serve и проверяем. Если страница отображается корректно, переходим к следующим шагам, если нет, то Laravel выведет ошибку, по которой можно разобраться в проблеме.

Для дальнейшей работы нужно настроить формы для корректной работы с базой данных. Необходимо добавить csrf-токены к каждой форме на сайте. Они защитят пользователя и приложение от csrf атак (cross-site request forgery – межсетевая подделка запроса).

CSRF или XSRF (Cross-Site Request Forgery) — атака на авторизованного пользователя сайта или веб-приложения, при которой злоумышленник обманом заставляет его выполнить опасное или некорректное действие.

Атака происходит в 4 действия:

1. Пользователь авторизуется на сайте. После успешного входа в систему сервер генерирует сессионный идентификатор, который обычно хранится в виде cookie в браузере пользователя. Эти cookie позволяют ему оставаться авторизованным без повторного ввода учетных данных.
2. Злоумышленник создает ссылку, которая «обращается» к сайту от имени пользователя. Мошенник отправляет ссылку через электронную почту, социальные сети или размещает ее на других веб-сайтах.
3. Пользователь кликает на ссылку. Когда пользователь переходит по мошеннической ссылке, браузер автоматически «обращается» к серверу доверенного сайта и включает в него сессионные cookie, так как они связаны с доменом доверенного сайта. С точки зрения сервера все в порядке, потому что «обращение» невозможно отличить от намеренного действия пользователя.
4. Система выполняет запрос злоумышленника. Поскольку система считает, что все корректно, сервер выполняет содержащееся в запросе действие от имени авторизованного пользователя, которому выдана сессионная cookie. Так злоумышленник может достичь своей цели без прямого доступа к учетным данным пользователя.

Laravel генерирует токены автоматически для каждой пользовательской сессии в целях предотвращения csrf-атак. Для этого внутри тега head вставляем мета тег (см рис. 19 в приложении), также в каждую форму необходимо добавить скрытое поле @csrf (см рис. 20 в приложении).

Если этого не сделать, то каждый раз при отправке формы получим 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 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.

**3.2 Настройка модели и контроллера**

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. Класс-наполнитель Seeders позволяет заполнить базу данных определенными сущностями. Обычно их используют для заполнения таблиц тестовыми значениями. Но не сами seeders создают эти значения. Значения генерируют фабрики (factory). Это встроенный класс, который создает экземпляры необходимых моделей.

Сейчас необходимо заполнить файл миграции (см рис. 20 в приложении). 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 (см рис. 21 в приложении).

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 (см рис. 22 в приложении).

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»

5. Также можно задать небольшую валидацию на поля формы, чтобы в базу данных не попало ничего лишнего (см рис 23 в приложении).

Стоит упомянуть, что, если мы используем VSCode, у вас нет специальных расширений, чтобы используемые классы автоматически подключались к файлу, придется прописывать все вручную (см рис. 24 в приложении). 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 (см рис. 25 в приложении).

1. Подключаем контроллер клиента и создаем get событие с ранее созданным классом index, который отображает 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 index.blade.php и прописываем в формы action (действие). То есть то, что будет происходить после отправки формы (см рис. 26 в приложении). В данном случае мы указываем на имя события, которое до этого задали в web.php.

6. 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 форм, а также как это все записывается в базу данных (см рис. 27 в приложении).

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.

Думаю, стоит упомянуть, что я забыл добавить в таблицу колонку с именами, потому что есть формы, где требуется ввести имя. Чтобы добавить новую колонку, 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 ошибка. И все готово (см рис. 28 в приложении).

Сделаем наполнитель и фабрику базы данных для наполнения тестовыми значениями. Сами классы были созданы выше, каждый выполняющий свою функцию. Классы seeders находятся в папке database/seeders. Переходим в ClientSeeder, подключаем модель клиента и добавляем в класс run() единственную строчку, которая сгенерирует 10 сущностей клиента в базу данных:

Client::factory(10)->create();

Основной класс, который отвечает за наполнение – это DatabaseSeeder, поэтому в его функции run запускаем наш seeder:

$this->call(ClientSeeder::class);

Как видим в seeder используется фабрика, поэтому следующим действием переходим в database/factories. Здесь уже создана фабрика клиента в ней функция definition(), которая возвращает данные, для заполнения таблицы в базе. Используем библиотеку faker() PHP, которая генерирует случайные значения в зависимости от заданных настроек, и прописываем заполняемые поля (см. рис 29 в приложении).

С помощью команды php artisan db:seed запускаем наш наполнитель, после чего переходим в базу данных и смотрим результат. База данных заполнилась десятью случайными значениями (см. рис 30 в приложении).

Далее делаем удобную нотификацию, чтобы менеджер вовремя получал уведомления о полученной заявке.

**3.3 Создание нотификации через почту.**

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 (см рис. 29 в приложении)

3. 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 почту, придумываем имя паролю и высвечивается автоматически сгенерированный пароль, его лучше куда-нибудь записать. Он записывется в MAIL\_PASSWORD файла .env.

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

$ php artisan make:mail BidFromSite

После этого в папке App создастся соответствующий класс.

1. Также 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. Пишем простой заголовок, что пришла заявка и в параграфе создаем переменные для названия формы, имени клиента и номера телефона (см рис. 30 в приложении).
2. Переходим в наш класс-контроллер и дописываем пару строчек для создания письма. Нам понадобится фасад Mail. Передаем экземпляру класса почты название формы, с которой пришла заявка, имя клиента и его номер телефона (см рис. 31 в приложении).
3. В самом классе почты мы записываем эти данные в конструктор со свойством Public. После чего blade-шаблон успешно может до них «дотянуться». Есть и альтернативный способ передавать переменные с помощью ключа with в функции content(). Важно чтобы имена переменных в blade и в классе были одинаковыми.
4. Запускаем сервер и пробуем отправить письмо (см рис. 32 в приложении). И мы получаем его, что по сути является одним из способов уведомления. В перспективе с письмами можно придумать много различных действий, по типу отправки купонов клиентам из базы данных, но для этого стоит переработать формы на сайте или создать процесс регистрации, чтобы получать контакты почты.

**3.4 Создание нотификации в Telegram.**

В современных реалиях удобнее использовать мессенджеры для нотификации. Помимо уведомлений о заявках, можно и вовсе отслеживать всю работу сайта. Поэтому следующим шагом мы создадим телеграм бота. Задаем новую зависимость в composer и устанавливаем telegram-bot-sdk.

$ composer require irazasyed/telegram-bot-sdk ^3.0

Затем требуется опубликовать конфигурацию телеграм командой:

$ php artisan vendor:publish

После чего появится телеграм конфиг-файл telegram.php. Следующим шагом создаем сам бот в телеграм с помощью бота BotFather (см рис. 33 в приложении). Тут все довольно просто: создаем бота, даем имя, затем пользовательское имя и в итоге бот пришлет токен, который понадобится дальше. Бот создан. Можем проверить HTTP запросом статус бота. Для этого пишем в адресной строке:

[https://api.telegram.org/bot<token>/getUpdates](https://api.telegram.org/bot%3ctoken%3e/getUpdates) ,

где <token> - это токен, который мы получили при создании бота. В итоге мы должны получить результат в JSON формате, где ok:true.

Далее создаем группу в телеграм, куда добавляем нашего бота и даем ему права администратора. Теперь нам нужно получить id только что созданного чата. Напишем что-нибудь в чат и там, где мы смотрели результат HTTP запроса, при обновлении должны появиться изменения. Тут мы можем найти Id чата (см рис. 34 в приложении). После этого добавим переменные в .env файл конфигурации (см рис 35 в приложении).

Остается только написать логику отправки сообщений. Сообщения отправляются через специальный фасад телеграма sendMessage (см рис. 36 в приложении), но на этапе разработки могут возникнуть определенные трудности с SSL-сертификатом, так как проект не находится в интернете, а расположен на компьютере локально. Поэтому, после перехода по нужной ссылке мы получаем ошибку SSL-сертификата (см. рис 37 в приложении).

Так как, у нас его нет, можно было бы использовать самоподписанный сертификат и использовать утилиту ngrok, которая вывела бы наш сайт в интернет. Однако на данный момент ngrok не работает в России и даже, скачав утилиту через VPN, она не работает с ip адресом, такая же ситуация будет при установке утилиты через composer.

LocalTunnel также не позволяет отправить сообщение по какой-то причине, поэтому, потратив на это чрезмерно много времени, используем более простой метод через функцию file\_get\_contents. Впоследствии, после того как приобретем домен и SSL сертификат, можно будет переписать эту функцию в пару строчек. Добавляем код обработки в контроллер,не забывая про переменные, которые передадим менеджеру. (см. рис. 38 в приложении).

Можно было бы создать отдельный класс для отправки нужных сообщений, чтобы не загружать контроллер. Так бы было правильно, но на данный момент это весь функционал, который нам необходим и в перспективе, когда потребуются более обширные действия, этот класс нужно будет разгрузить. Теперь, отправляя заявку на сайте, мы получаем уведомление и в телеграм чат (см рис. 39 в приложении).

**3.5 Планировщик задач. Логирование.**

Laravel поставляется со встроенным API, который поможет выполнять регулярно те или иные задачи: очистка логов, отправка сообщений, резервное копирование различных элементов сайта, генерация отчетов и т.д. Функция планирования вызывается как обычная функция Laravel, поэтому для реализации понадобится cron. Cron – это утилита, позволяющая выполнять скрипты на сервере в назначенное время с заранее определенной периодичностью.

В Laravel существует удобный инструмент для работы с задачами schedule. Чтобы посмотреть все существующие задачи нужно воспользоваться командой php artisan schedule :list.

Создадим консольную команду php artisan make:command AppLogFileSending. В функции handler() необходимо прописать алгоритм действия при запуске задачи. Отправим в уже созданный чат телеграм сообщение о статусе нашего запроса к сайту, чтобы иметь возможность вовремя среагировать на какую-либо ошибку.

Поместим запрос в конструкцию try{} catch{}, чтобы при возникновении ошибки соединения задача все равно сработала (см. рис. 41 в приложении). А если ошибка все-таки возникнет, запишем ее текст в log файл, который находится по пути storage/laravel.log. Логирование – это не маленькая часть от разработки приложения. Оно поможет досконально описать работу приложения, все его ошибки, баги. Помогает в исправлении этих ошибок и в будущем предотвращать появление. Кроме указанного, информация из логов позволяет маркетологам получать данные про объем трафика, время посещения страниц, страницы входа/выхода, географию посетителей и другую важную для повышения конверсии информацию. Таким образом, логирование является важным этапом контроля и мониторинга работы системы, позволяющим увеличить коэффициент ее производительности.

Нам же необходимо прописать консольную команду в файле routes/console.php:

Schedule::command('app:log-file-sender')->hourly();

Используя cron, эта команда позволит выполнять консольную задачу каждый час, и мы сможем отслеживать работу приложения в том же чате, где принимаем заявки от пользователей. Поэтому настройка чат-бота – это не только нотификация, а еще собственный журнал, который можно настроить, как угодно.

**Глава 4. Заключение**

**4.1 Подведение итогов**

В данном проекте я постарался затронуть большую часть деятельности fullstack-разработчика насколько это было возможно: начиная от верстки сайта и заканчивая работой с базой данных и подключением API.

В процессе обучения в Geekbrains, я больше проникся frontend-разработкой, так как такая деятельность была мне ближе еще задолго до обучения. Во время обучения, в какой-то момент, я подумал, что backend вовсе мне не подходит, до блока с разработкой на Laravel. Сейчас же я вижу столько возможностей с backend-разработкой, что она автоматически захватила мое внимание.

Размышляя над тем, что можно было бы разработать еще в рамках дипломного проекта, я пришел к выводу, что все ограничивается только фантазией. По ходу разработки проекта приходили мысли о том, как можно все переделать и, возможно, то, что я взял за основу старый проект, в какой-то степени, ограничивало меня в разрабатываемом функционале. Также я засомневался в том, что выбрал laravel, как основной фреймворк. Во время разработки я узнал немного про Angular, что меня заинтересовало, но во время обучения его не было, поэтому выбор остался на laravel. В конечном итоге для таких простых сайтов нет необходимости использовать такие фреймворки, потому что функционал сайта довольно примитивный, однако, этот проект полностью дееспособен и может выполнять свою основную задачу.

**4.2 Что можно сделать в перспективе.**

Если говорить об улучшении проекта, то конечно стоит переделать сам дизайн и расширить сайт до интернет-магазина. В проекте уже описана функция реализации каталога по данным, которые будут приходить с сервера, а также несложный метод сортировки.

Можно сделать мобильные и планшетные версии для сайта. Они описаны у меня в старом проекте с помощью media-запросов, но что бы сэкономить немного времени, не реализованы в самом проекте laravel.

Также стоит рассмотреть систему регистрации пользователей, что делается не очень сложно в laravel с помощью библиотеки Laravel Breeze. Также можно подключить OAuth авторизацию и собирать данные для дальнейшей работы и упростить авторизацию пользователей.

Для администратора или менеджера создать административную панель, через которую можно будет управлять контентом на сайте: добавлять товары, редактировать и обновлять (CRUD – функции).

Реализовать подробное логирование системы, для дальнейшей обработки всевозможных ошибок и сбора информации о сессиях пользователя.

Можно использовать websocket - протокол компьютерной связи , обеспечивающий одновременный двусторонний канал связи по одному соединению Transmission Control Protocol (TCP). Это поможет обеспечить связь с технической поддержкой. В прошлом для таких функций я использовал сторонние сервисы, сейчас же есть понимание как это можно воссоздать самостоятельно.

И это только первое, что приходит в голову. В работающем проекте постоянно сталкиваешься с какими-либо проблемами и их решение расширяет функционал сайта настолько, что в текущей точке разработки сложно себе представить.

**Список источников:**

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

«Изучаем программирование на JavaScript» Робсон Элизабет и Фримен Эрик

<https://geekbrains/> - образовательный портал

<https://blog.skillfactory.ru/kto-takoj-frontend/> – Кто такой фулстек-разработчик.

<https://blog.skillfactory.ru/backend-razrabotchik/> – Кто такой бэкенд-разработчик.

<https://skillbox.ru/media/code/fullstack-razrabotchik-kto-takoy-chto-delaet-i-skolko-zarabatyvaet/> – Кто такой бэкенд-разработчик.

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

<https://wp-kama.ru/note/interesnye-novye-html5-tegi> – новые теги в HTML5.

<https://msiter.ru/tutorials/html5/new_elements> – новое в HTML5

<https://tproger.ru/translations/how-css-flexbox-works> – flexbox в CSS.

<https://sass-scss.ru/guide/> – гайд по работе с препроцессами Sass.

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

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

<https://bobday.ru/javascriptvwebrazrabotke> – JavaScript в веб-разработке.

<https://learn.javascript.ru/debugging-chrome> – отладочная панель в Chrome

https://delphinpro.ru/a/6 – настройка phpmyadmin и начало работы.

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

<https://www.qmedia.by/blog/laravel.html> - зачем используют Laravel

<https://blog.skillfactory.ru/glossary/laravel/> - больше о Laravel

<https://laravel.su/docs/11.x/structure> – каталог файлов Laravel

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

<https://startx.team/blog/statyi/csrf-ataka-metody-zashhity-i-primery-uyazvimostej/> – статья о CSRF атаках

<https://laravel.su/docs/11.x/routing> – роутинги в Laravel

<https://laravel.su/docs/11.x/validation> – валидация в Laravel

<https://fakerphp.org/formatters/numbers-and-strings/> - использование faker() для генерации случайных значений

<https://laravel.su/docs/11.x/mail> – настройки почтовых функций в Laravel

https://developer.mozilla.org/ru/docs/Learn\_web\_development/Core/Styling\_basics/Values\_and\_units

<https://habr.com/ru/articles/439200/> - composer для самых маленьких

<https://www.youtube.com/watch?v=1J8l3pCc8p8> – настройка webhook для телеграм бота

<https://timeweb.com/ru/community/articles/chto-takoe-cron> - что такое cron

<https://laravel.su/docs/11.x/scheduling> - планировщик задач

<https://habr.com/ru/articles/795445/> - для чего нужно логирование

**Приложения**.

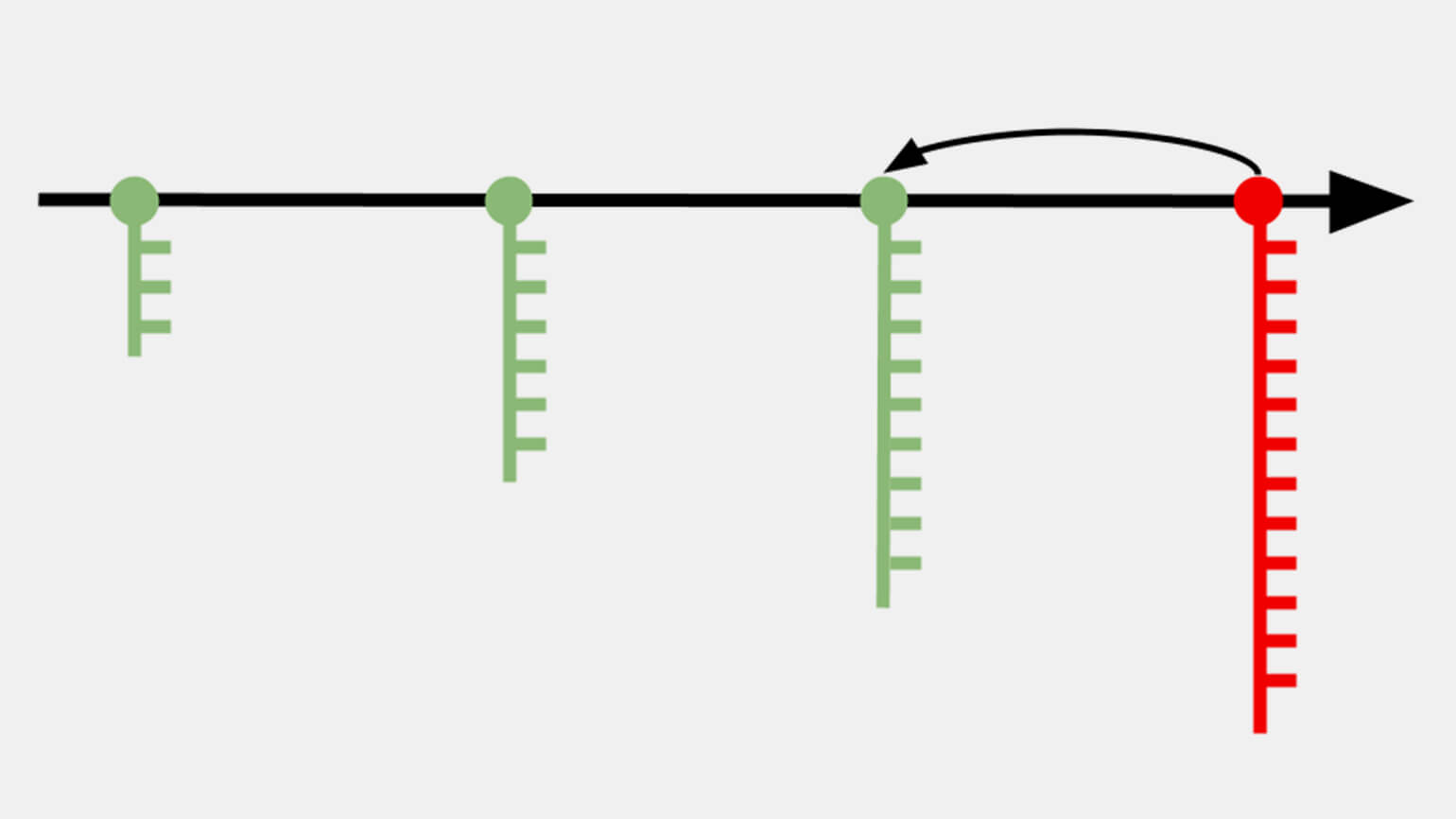


Рисунок 1 – Схема контроля версий

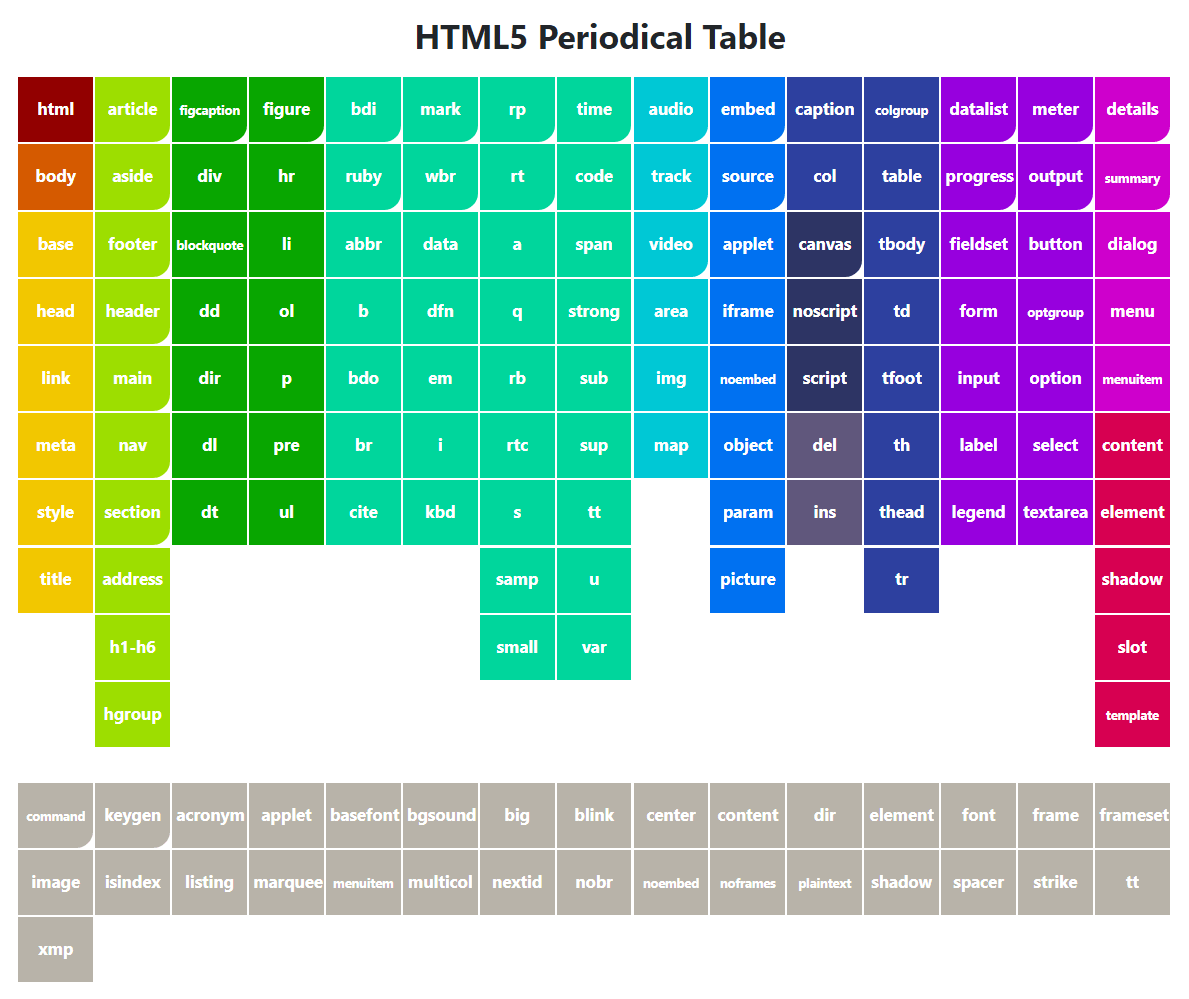


Рисунок 2 - Разнообразие HTML тегов

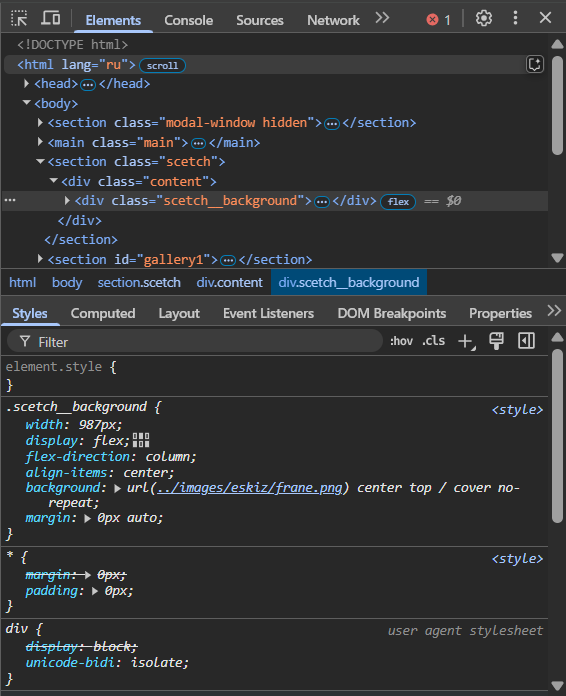


Рисунок 3 – Отладочная панель в браузере Chrome

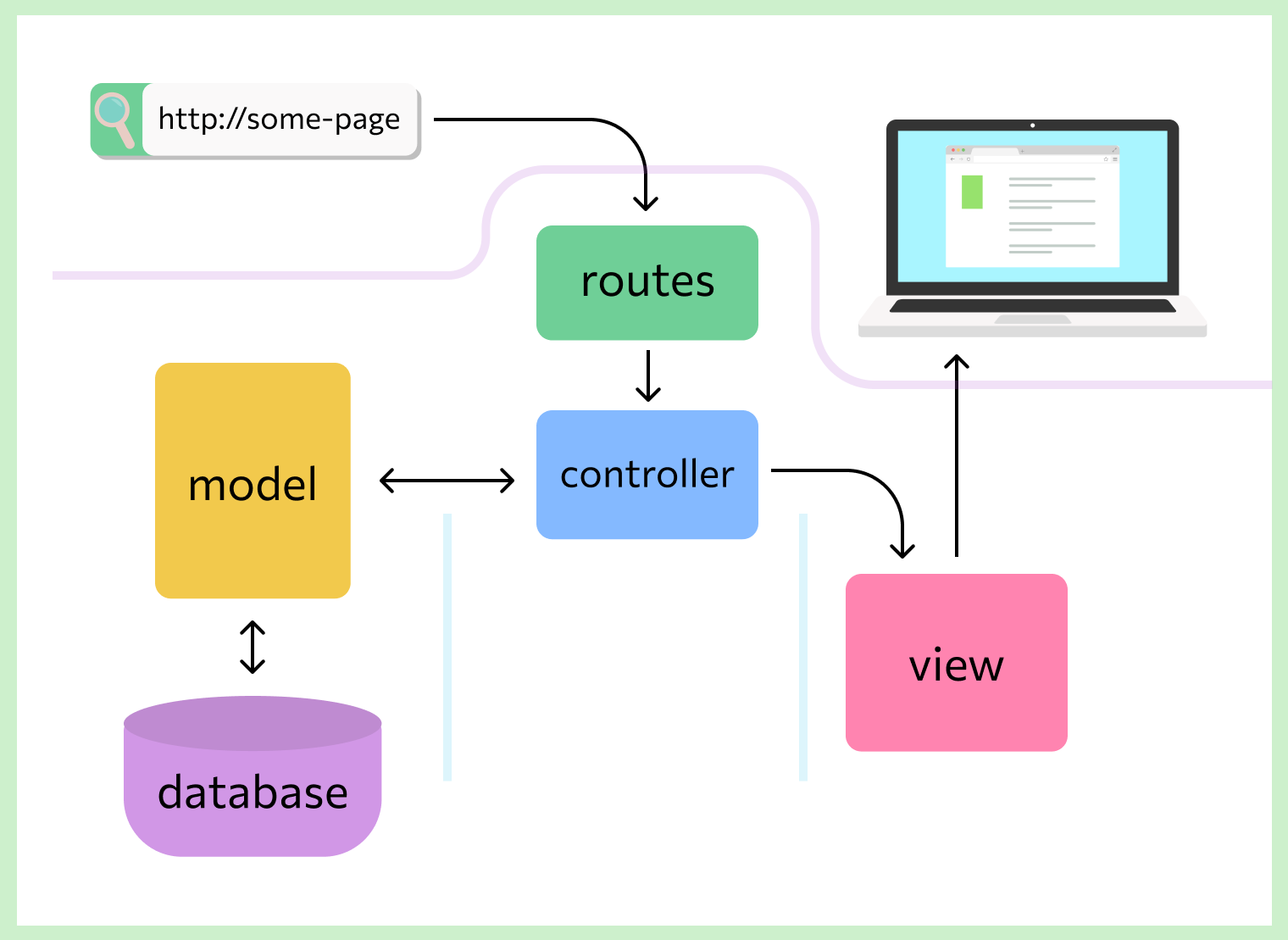


Рисунок 4 – Архитектура MVC

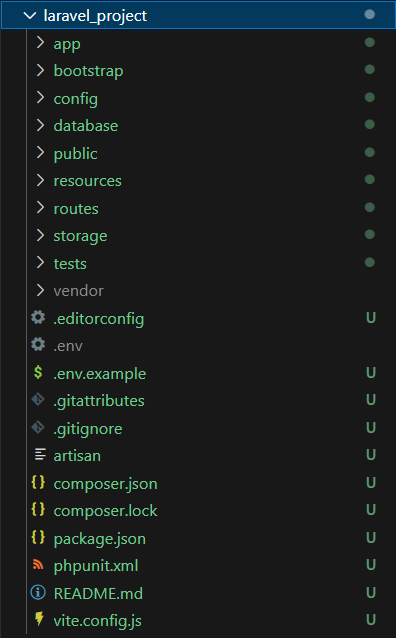


Рисунок 5 – Стандартный каталог файлов Laravel

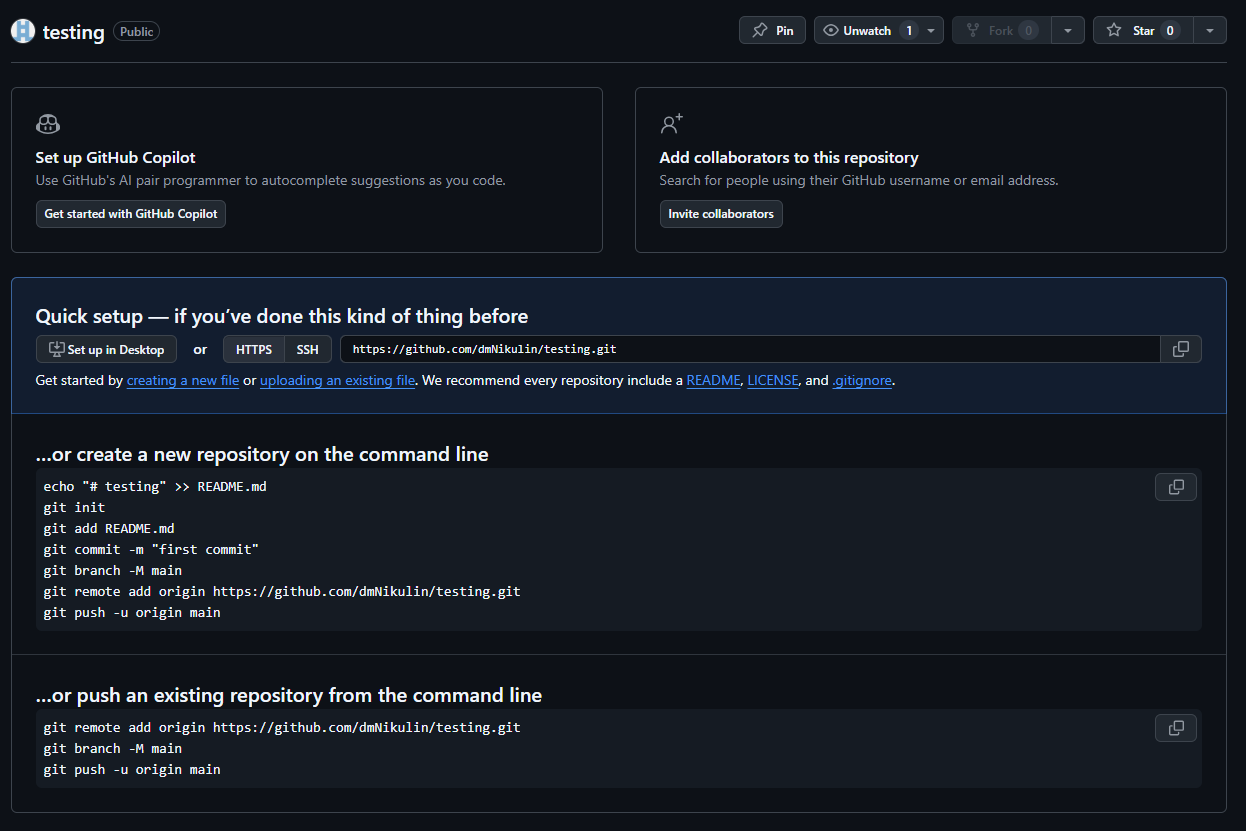


Рисунок 6 – Создания git репозитория

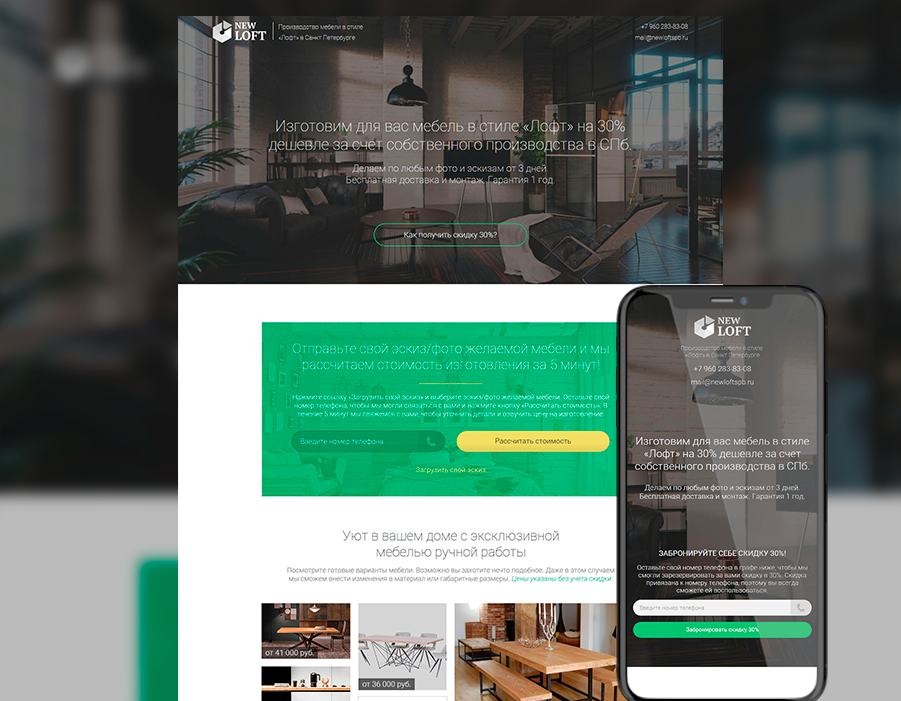


Рисунок 7 – Пример дизайн-макета

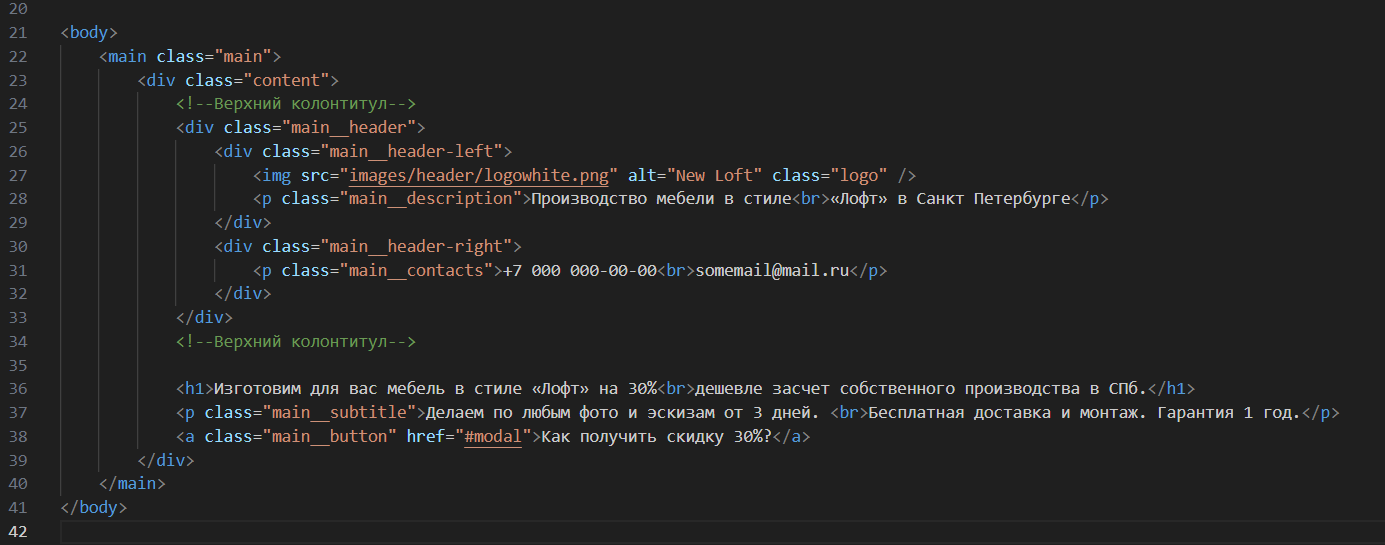


Рисунок 8 – Названия классов в html

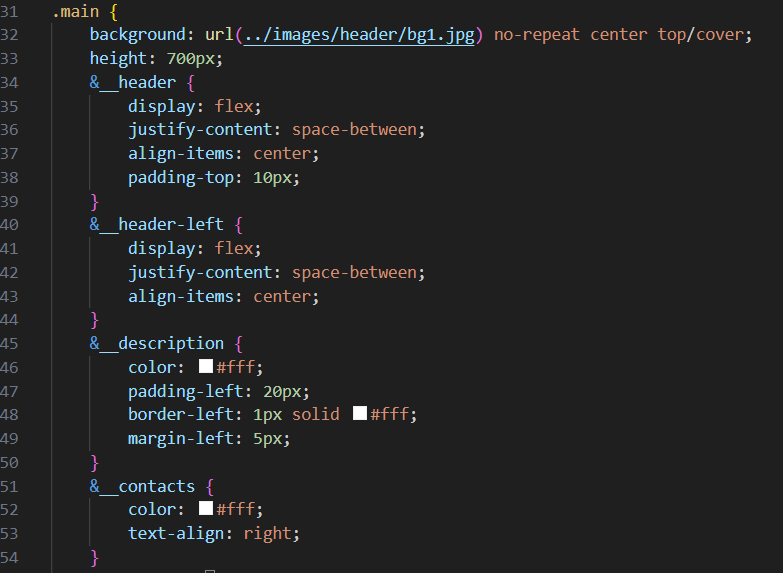


Рисунок 9 – Вложенность свойств в Sass

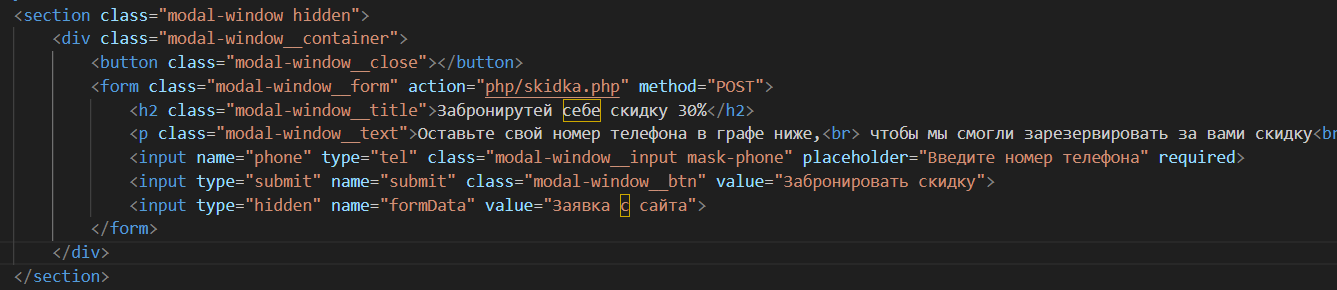


Рисунок 10 – Модальное окно в HTML

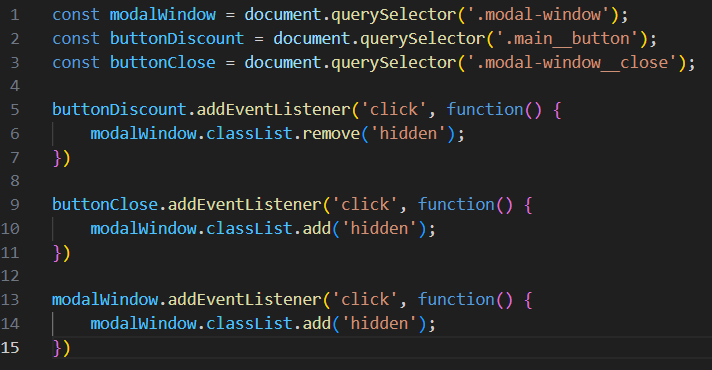


Рисунок 11 – Скрипт работы модального окна

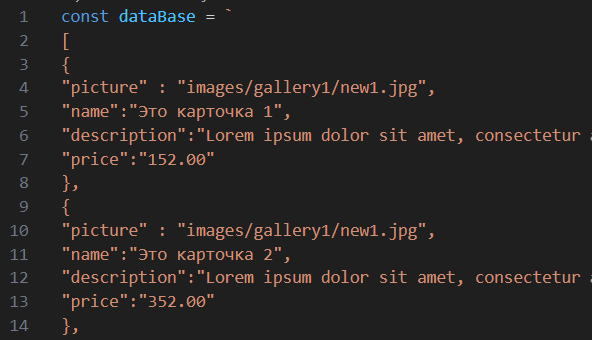


Рисунок 12 – JSON файл с товарами.

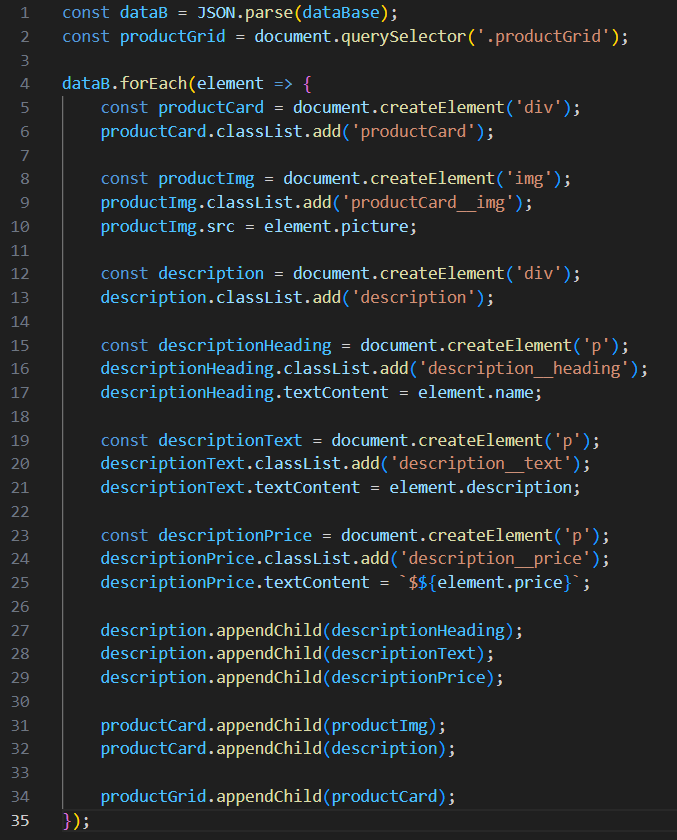


Рисунок 13 – Скрипт реализации товаров из JSON

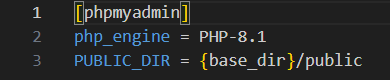


Рисунок 14 – Код файла project.ini



Рисунок 15 – Настройки config.php для OSP.

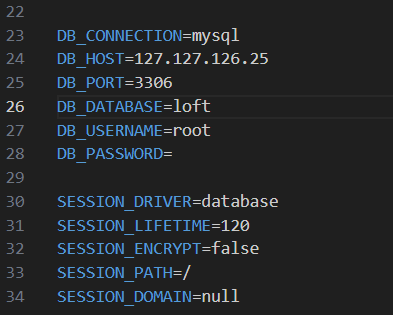


Рисунок 16 – Настройки базы данных в файле .env

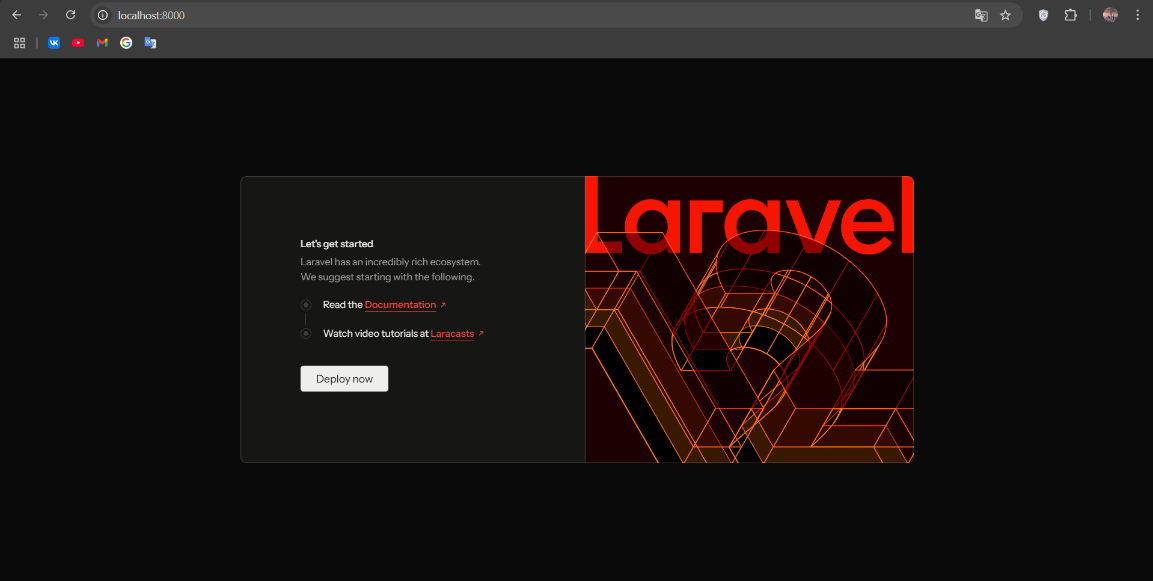


Рисунок 17 – Приветственная страница Laravel

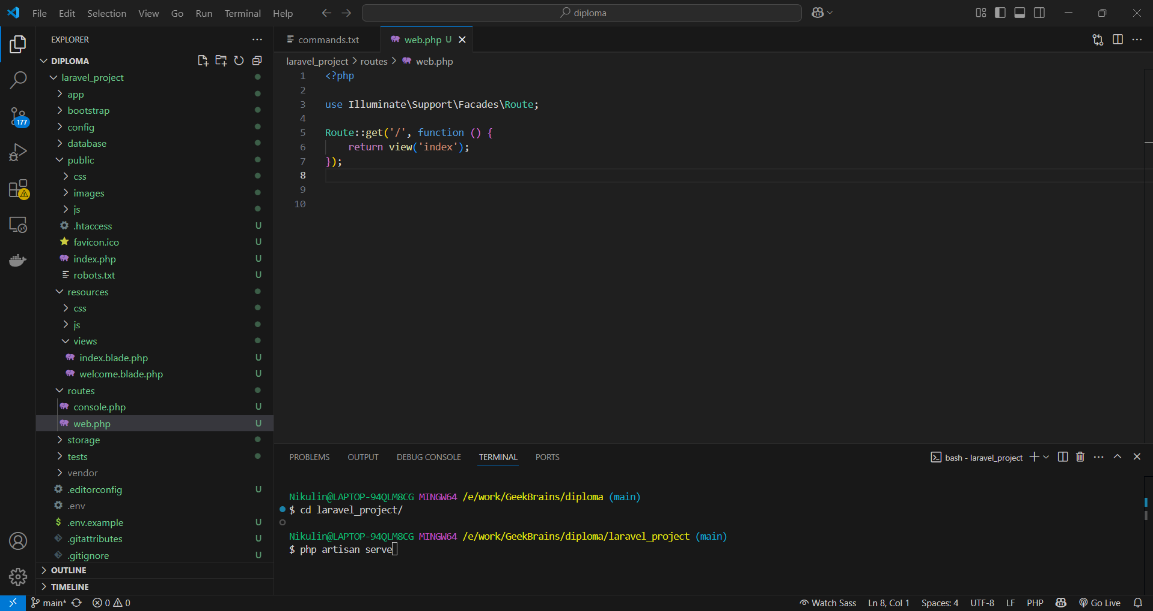


Рисунок 18 – Роутинг для главной страницы



Рисунок 19 – Мета-тег для работы csrf токена

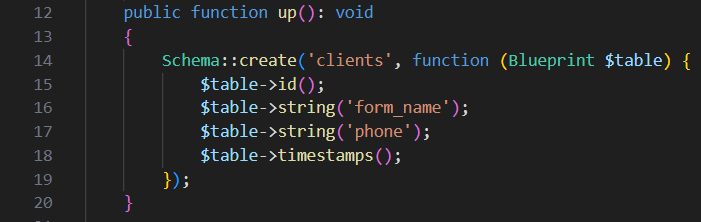


Рисунок 20 – Файл миграции базы данных.

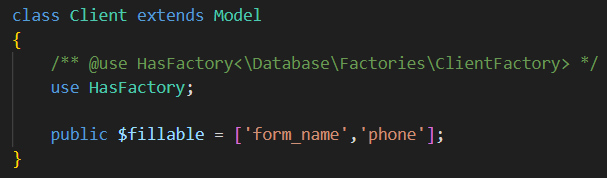


Рисунок 21 – Модель клиента

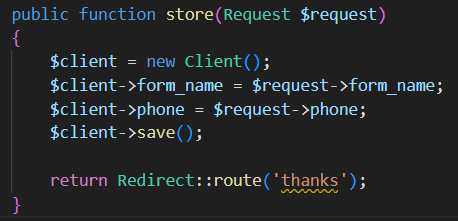


Рисунок 22 – заполнение класса store()

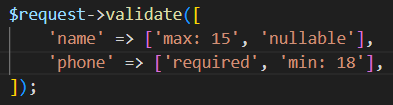


Рисунок 23 – Валидация полей формы

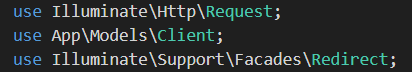


Рисунок 24 – Подключение классов

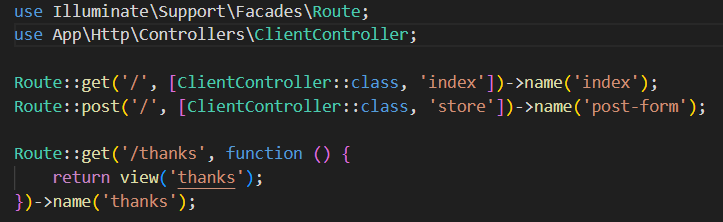


Рисунок 25 – Роутинг на страницу после отправки заявки.



Рисунок 26 – Действие после отправки формы

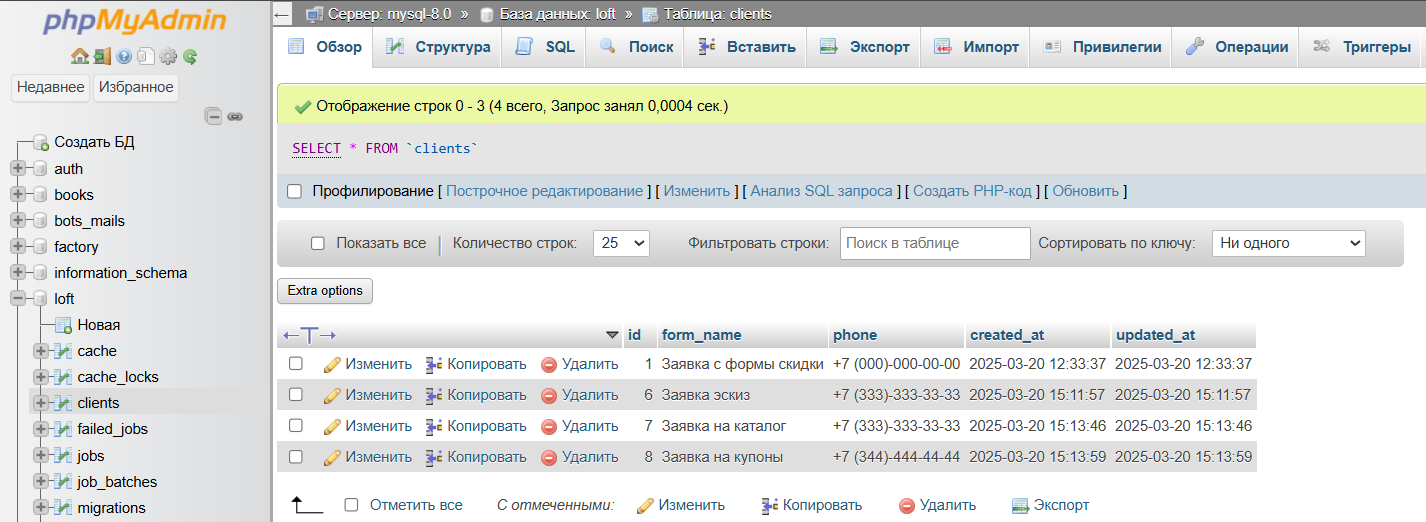


Рисунок 27 – База данных

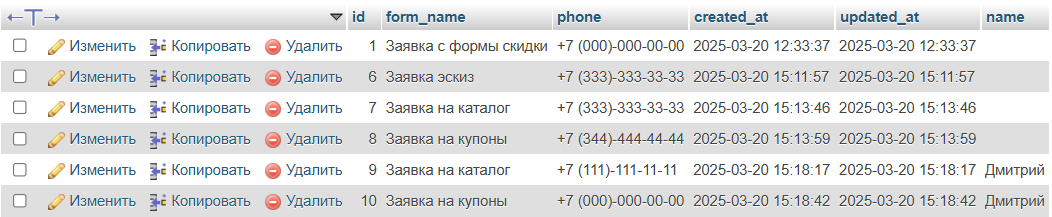


Рисунок 28 – Добавление нового поля в БД

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 29 – Создание фабрики для наполнителя.

Изображение выглядит как текст, снимок экрана, Шрифт, число

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 30 – Результат заполнения тестовыми значениями.

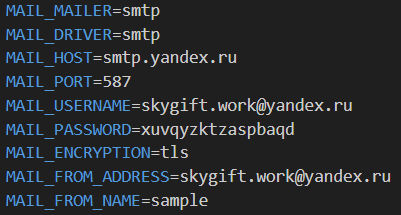


Рисунок 29 – Конфигурационные настройки для почты

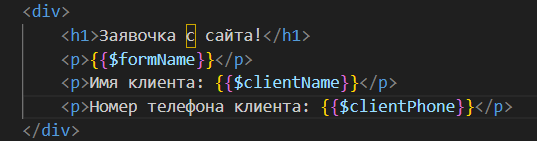


Рисунок 30 – Blade-шаблон для уведомления.



Рисунок 31 – Отправка сообщения. Логика в контроллере.

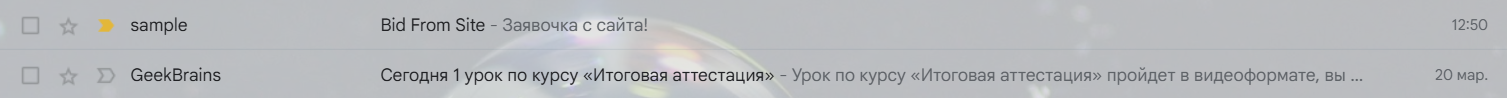


Рисунок 32 – Уведомление о заявке

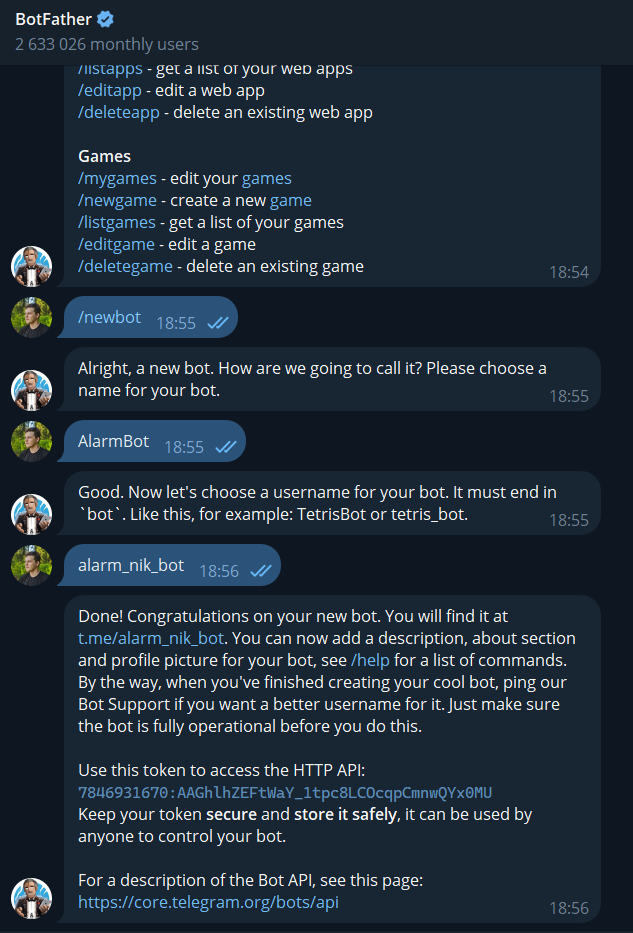


Рисунок 33 – Создание телеграм бота.

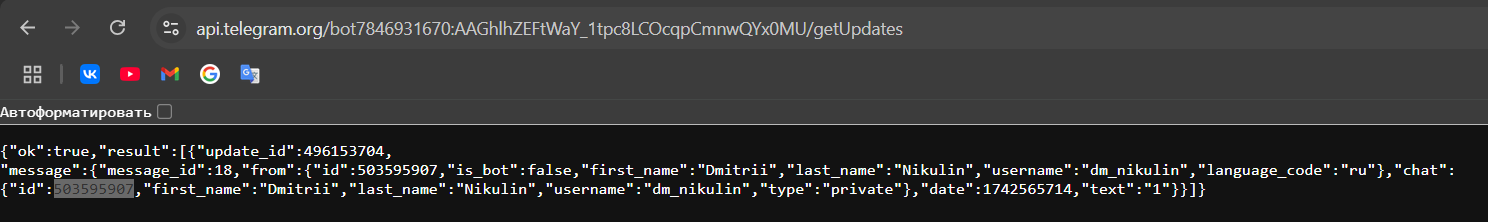


Рисунок 34 – HTTP запрос к боту.



Рисунок 35 – Переменные для работы чат-бота

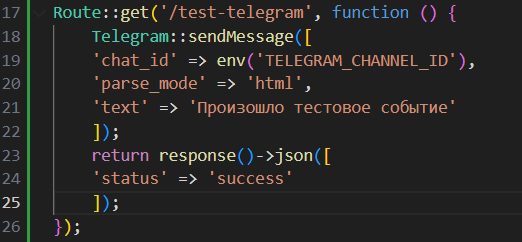


Рисунок 36 – Роутинг с отправкой сообщения через бота.

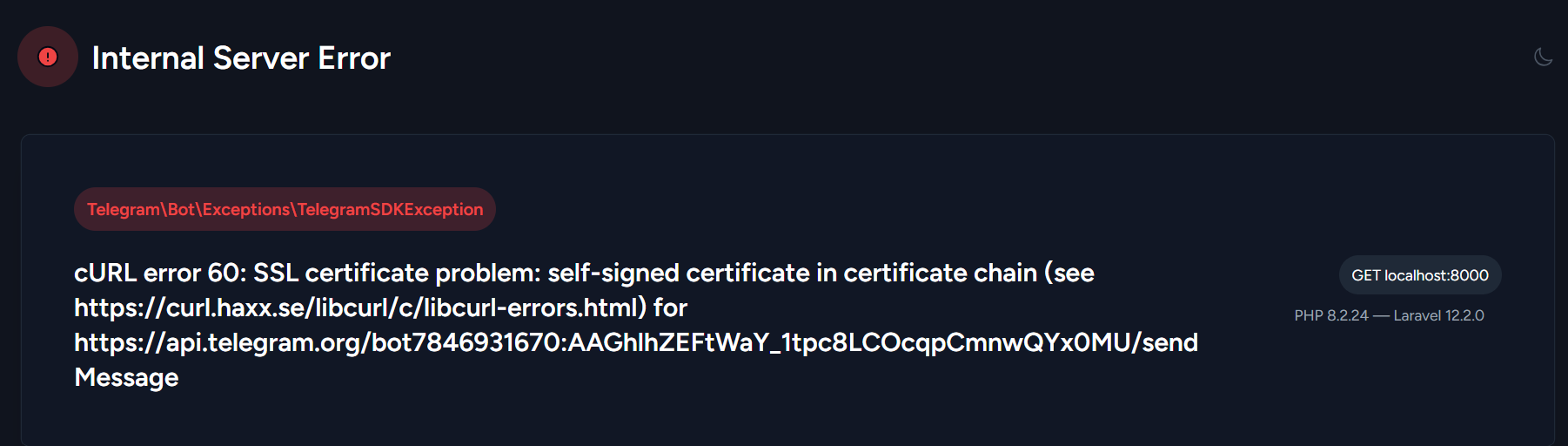


Рисунок 37 – Ошибка самоподписанного сертификата

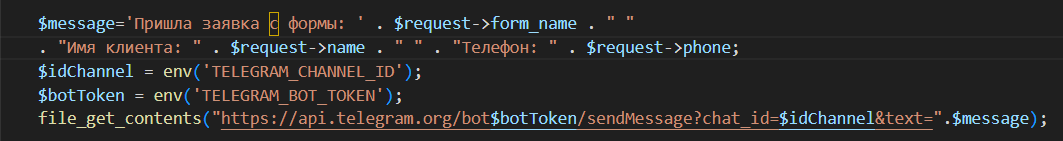


Рисунок 38 – Код для отправки сообщения в телеграм

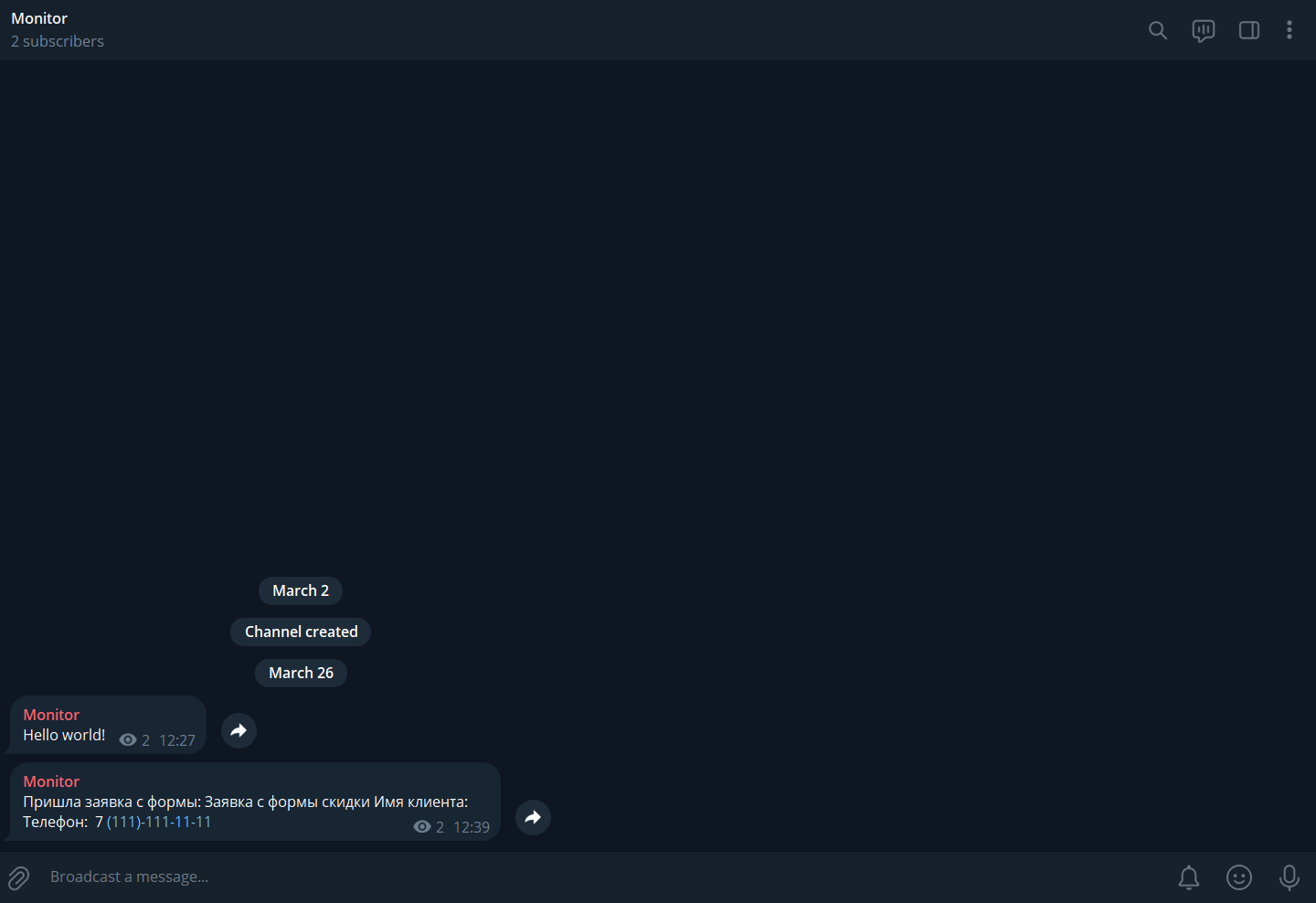


Рисунок 39 – Уведомление о получении заявки.

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

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 41 – Планировщик задач.

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 42 – Уведомления о статусе сайта.