

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ  
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
«САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ  
ПЕТРА ВЕЛИКОГО»  
Институт компьютерных наук и технологий

**Отчет о прохождении производственной технологической  
(проектно-технологической) практики  
на тему: «Адаптация верстки к October CMS»**

Докукина Эмилия Арменовича, 3 курс, гр. 3530903/90302

**Направление практической подготовки:** 09.03.03 Прикладная информатика.

**Место прохождения практической подготовки:** ФГАОУ ВО «СПбПУ», ИКНТ,  
ВШИСиСТ г. Санкт-Петербург, ул. Обручевых, д.1, лит. В.

**Сроки практической подготовки:** с 11.06.2022 по 09.07.2022.

**Руководитель практической подготовки от ФГАОУ ВО «СПбПУ»:** Щербаков  
Н., доцент ВШИСиСТ, Кандидат технических наук.

**Консультант практической подготовки от ФГАОУ ВО «СПбПУ»:** Пархоменко  
Владимир Андреевич, ассистент ВШИСиСТ.

**Оценка:** \_\_\_\_\_

Руководитель практической подготовки  
от ФГАОУ ВО «СПбПУ»

Н. Щербаков

Консультант практической подготовки  
от ФГАОУ ВО «СПбПУ»

В.А. Пархоменко

Обучающийся

Э.А. Докукин

Дата: 09.07.2022

## СОДЕРЖАНИЕ

Введение .....	4
Глава 1. Системы управления сайтом: виды, принципы работы, область применения .....	6
1.1. Основные функции CMS .....	6
1.2. Виды CMS .....	7
1.2.1. Traditional CMS .....	7
1.2.2. Decoupled CMS .....	8
1.2.3. Headless CMS .....	9
1.3. Выводы .....	10
Глава 2. October CMS: возможности системы, особенности, структура .....	11
2.1. Общие сведения .....	11
2.2. Область применения October CMS .....	11
2.3. Архитектура October CMS .....	12
2.4. Возможности October CMS .....	13
2.5. Выводы .....	17
Глава 3. Адаптация верстки к October CMS .....	18
3.1. Технология адаптации верстки .....	18
3.2. Описание проекта .....	19
3.3. Создание темы: шаблоны, страницы, ассеты .....	21
3.4. Создание плагина .....	23
3.4.1. Создание моделей и контроллеров .....	23
3.4.2. Создание миграций и контроллеров .....	27
3.4.3. Создание Backend меню .....	28
3.4.4. Работа с представлениями .....	29
3.4.5. Настройка плагина для отправки почты .....	30
3.5. Выводы .....	31
Глава 4. Проверка правильности работы сайта: создание записей, проверка их отображения, отправка почты .....	32
4.1. Создание записей с услугами .....	32
4.2. Создание записей с проектами .....	33
4.3. Проверка корректности отображения логотипов с партнерами .....	36
4.4. Проверка работоспособности формы обратной связи .....	36
4.5. Выводы .....	38
Заключение .....	39
Список использованных источников .....	41

Приложение 1. Код плагина .....	42
Приложение 2. Код страниц и макета .....	47

## ВВЕДЕНИЕ

В наше время интернет занимают огромную часть нашей жизни. С каждым днём всё больше сфер переносятся в виртуальную среду. Мы можем заказать себе еду, записаться на прием к врачу, созвониться с другом на другом конце земли и даже открыть счет в банке, не вставая с дивана. Сложно представить, что всё это было еще недоступно менее чем 50 лет назад.

Неудивительно, что сфера разработки веб-сайтов или веб-приложений значительно актуализировалась и стремительно продолжает развиваться по сей день. Особенно это важно для бизнеса, ведь сложно представить себе даже маленькую компанию, которая не хотела бы привлекать к себе клиентов не только с помощью радио, телевидения и наружной рекламы, но и посредством сети интернет с большим охватом.

С каждым днем появляются новые технологии и инструменты разработки веб-сайтов. Одним из этих инструментов является CMS (content management system). Управление и регулярное обновление веб-сайтов компании имеет решающее значение в цифровом мире, в котором мы работаем и живем. Предприятиям без собственных веб-разработчиков нужна интуитивно понятная система, позволяющая любому легко обновлять информацию, сохраняя при этом идентичность своего бренда и поддерживая обмен сообщениями с потенциальными клиентами.

Система управления контентом, чаще всего называемая CMS, позволяет частным лицам и компаниям редактировать, управлять и поддерживать существующие страницы веб-сайта в едином интерфейсе, не требуя специальных технических знаний.

Это экономит значительное количество времени и денег. С точки зрения обновления и обслуживания, CMS позволяет сосредоточиться на более важных аспектах проекта, которые необходимо улучшить, например, на создании и добавлении новых сообщений в блог компании или на разработке контента, который поможет привлечь больше потенциальных клиентов.

Здесь будет представлена набирающая популярность на данный момент October CMS. Основной **целью данной работы** является изучение процесса адаптации уже имеющейся верстки к October CMS. Создадим свой плагин для уже существующего проекта, создадим необходимые зависимости в базе данных посредством миграций от фреймворка Laravel, внедрим возможность получения заявок от пользователей с помощью уже имеющегося плагина Magic Forms, обеспечим пересылку информации о заявках на электронную почту, адаптируем верстку к October CMS и протестируем работоспособность системы.

#### **Задачи работы:**

- Изучение системы управления содержимым сайта October CMS. Установка необходимых средств, обеспечивающих работу данной CMS: Open Server, Composer.
- Создание плагина в данной CMS, обеспечивающего интерфейс взаимодействия администратора сайта с CMS. Создание необходимых зависимостей с БД.
- Адаптация верстки к CMS, реализованная созданным ранее плагином.
- Внедрение возможности просмотра заявок пользователей в панели управления.
- Проверка работоспособности системы управления. Создание нескольких тестовых страниц разного содержания. Тестирование обрабатываемости заявок.

**Результатом работы** будет являться, готовый к наполнению и деплою на хостинг веб-сайт, имеющий удобную панель администратора для создания и изменения страниц и/или их содержимого.

## **ГЛАВА 1. СИСТЕМЫ УПРАВЛЕНИЯ САЙТОМ: ВИДЫ, ПРИНЦИПЫ РАБОТЫ, ОБЛАСТЬ ПРИМЕНЕНИЯ**

В данной главе рассматривается общее устройство и принцип работы CMS, их предназначение, виды и различия. Также обозначается область применения данных систем: когда их стоит использовать, а когда выгодней будет от них отказаться.

### **1.1. Основные функции CMS**

Главная задача систем управления - обеспечить разработчика возможностью достаточно быстро и эффективно создать необходимый интерфейс, для того, чтобы можно было манипулировать контентом сайта. Проще говоря, если стоит задача - создать блог, то должна быть возможность привязать определенный интерфейс CMS к созданию постов, категорий, их характеристик и прочего, что характерно для статьи.

В общем случае основная функция CMS - возможность создания необходимого интерфейса и зависимостей для данного проекта, чтобы в дальнейшем посредством cms заказчик мог полноценно менять контент сайта, не задумываясь о том, что находится под капотом.

В каждой CMS это реализовано по-разному. Какие-то системы управления контентом больше заточены для того, чтобы создавать удобный интерфейс взаимодействия с интернет-магазином (часто используют Open Cart), какие-то лучше подходят для блогов (WordPress), а есть и такие, с помощью которых очень удобно создавать и дешевые, ненагруженные функционалом, сайты (Tilda, Wix).

## 1.2. Виды CMS

### 1.2.1. Traditional CMS

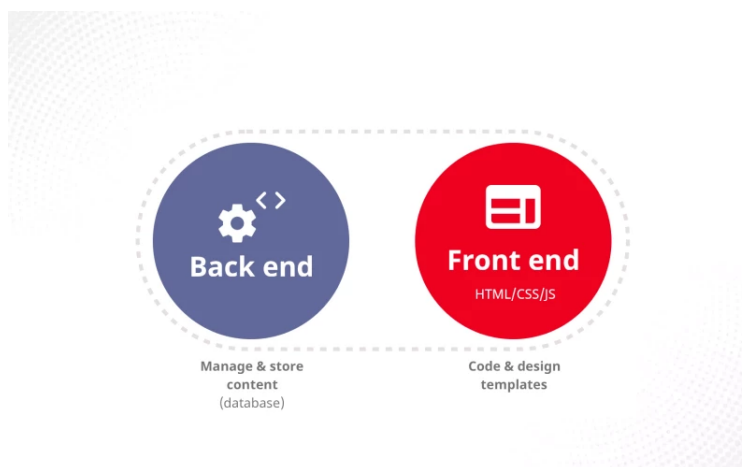


Рис.1.1. Устройство Traditional CMS

Если мы говорим о традиционной архитектуре CMS, то проще всего называть ее связанной. В Traditional CMS весь контент, в том числе все assets, создается, управляется и хранится на серверной части сайта. Данные достаются из базы данных во внешний интерфейс. Внешний интерфейс имеет встроенные шаблоны тем и CSS, которые отображают контент на веб-сайте.

В такой архитектуре frontend и backend достаточно тесно связаны и не имеют иногда не достающей разделенности. База данных, код, HTML-шаблоны, файлы CSS и JavaScript, из которых состоит тема сайта (внешняя и внутренняя), предварительно настроены. Внесение изменений в такую систему может потребовать сравнительно больше времени для разработчика, если бы он использовал для этих задач другой тип CMS.

Платформы для ведения блогов, такие как WordPress, Squarespace и Wix, являются примерами традиционной архитектуры CMS.

### 1.2.2. Decoupled CMS

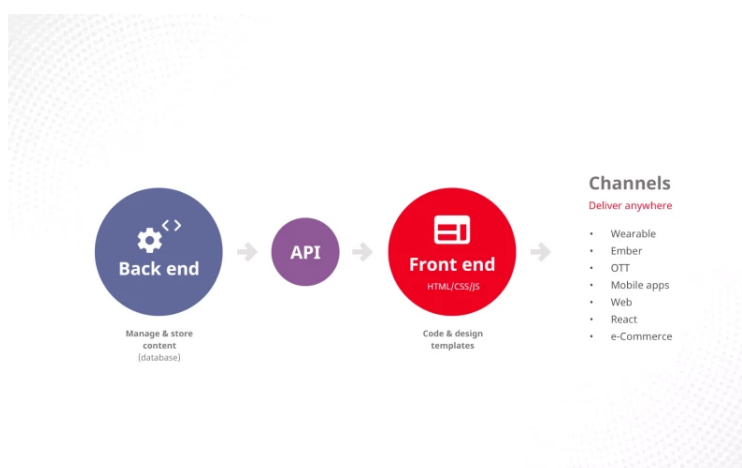


Рис.1.2. Устройство Decoupled CMS

Как следует из названия, в Decoupled CMS интерфейсная и серверная архитектуры отделены друг от друга.

Интерфейс и серверная часть достаточно похожи на традиционную CMS. Однако обычная CMS не справляется со своей задачей, когда нам нужна интерфейсная система, требующая большей настройки для создания оптимального взаимодействия с пользователем.

Вот почему в Decoupled CMS две или более систем могут взаимодействовать, не будучи связанными. Вы можете изменить уровень представления (внешний интерфейс) или кодовую базу (внутренний интерфейс), при этом изменения внешнего интерфейса не повлияют на серверную часть. Таким образом, появляется возможность контролировать внешний вид веб-сайта без обязательного изменения содержимого.

Несмотря на то, что серверная часть и клиентское приложение функционируют независимо друг от друга, архитектура клиентской части CMS предопределена с определенной средой (например, React или React Native). Таким образом, две системы тесно связаны и могут функционировать как одна.



### 1.2.3. Headless CMS

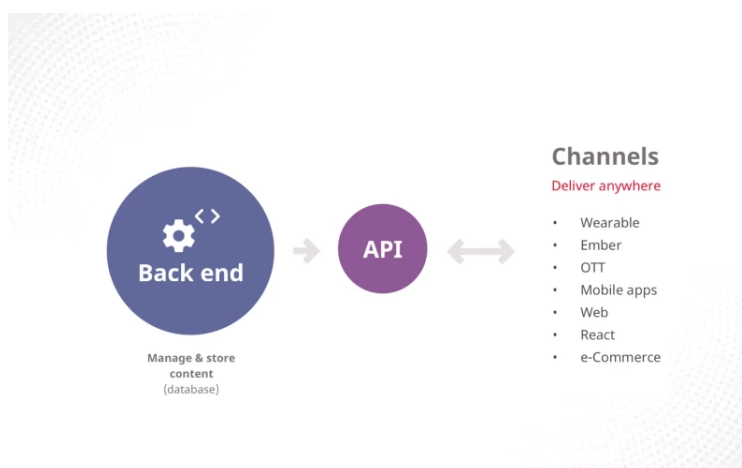


Рис.1.3. Устройство Headless CMS

Как было сказано ранее, CMS позволяет клиенту менять контент сайта, без специальных технических знаний. Достаточно зайти в панель администратора, добавить определенную запись в посте, и на той стороне пользователь увидит обновленную информацию.

Обычно это происходит так, что для каждой платформы разрабатывается собственная архитектура, подготавливается контент, настраивается интерфейс. Создание продуктов с таким подходом требует достаточно больших временных и денежных ресурсов. Это ограничивает возможности компаний в плане других каналов взаимодействия с данными.

Помимо традиционных видов CMS, существуют и так называемые Headless CMS. Они решают проблему управления контентом с использованием различных платформ. Здесь содержимое создается, хранится и редактируется без привязки к какой-то конкретной технологии, используемой для его представления на клиентском оборудовании (браузере, телефоне или другом устройстве).

Headless CMS предоставляет некоторое API, с помощью которого можно обращаться к измененным данным на со стороны front-end'а, мобильного приложения или любого другого приложения, которое может использовать наши данные для преобразования их в своей среде.

Проще говоря, такие CMS не подразумевают под собой необходимость привязки данных, измененных администратором сайта, к определенному представлению их в виде html страницы. Такие системы управления предоставляют API, благодаря которому front-end или мобильное приложение может обратиться к нему и получить необходимые данные.

Содержимое легко переносится в новые интерфейсы. Например, при реализации того же продукта на IOS, при наличии web- и Android-версий, не возникнет необходимости создавать новый бэкенд. Вместо этого к существующему набору просто добавляется еще одно клиентское приложение.

Среди Headless CMS можно выделить Strapi, Sanity, Agility и Ghost.

### 1.3. Выводы

Помимо классической дилеммы выбора CMS, можно заметить, что преимущества одного типа CMS часто оборачиваются ограничениями другого типа CMS. Выбор правильного типа зависит от потребностей бизнеса и требований компании.

Headless CMS - хороший вариант, если проекту потребуется гибкая интерфейсная архитектура, которую не предоставляет традиционная CMS. Headless CMS, ориентированная на API, позволит эффективно продвигать свой контент по различным каналам, что делает ее отличным инструментом для созданием кросс-платформенного продукта.

Однако, если задачей стоит создание автономного веб-сайта для бизнеса, лучше всего подойдет decoupled CMS - если не традиционная.

## **ГЛАВА 2. OCTOBER CMS: ВОЗМОЖНОСТИ СИСТЕМЫ, ОСОБЕННОСТИ, СТРУКТУРА**

В этой главе мы познакомимся с October CMS, к которой в дальнейшем и будет адаптирована верстка. Рассмотрим ее возможности, функционал, особенности и структуру. Будет проанализировано, для каких проектов она больше всего подходит и почему в данной работе выбрана именно эта CMS.

### **2.1. Общие сведения**

October - это платформа системы управления контентом (CMS), первоначально выпущенная в 2014 году и предназначенная для помощи разработчикам в создании и редактировании веб-сайтов.

Это платная платформа, с помощью которой пользователи могут выполнять такие задачи, как управление текстовым, графическим или видеоконтентом, проектирование внешнего вида своих веб-сайтов, отслеживание сеансов пользователей, размещение форумов и сбор комментариев посетителей.

October CMS предпочтительнее из-за своей простоты и повышенной безопасности. Разработчики утверждают, что не клиенты создают веб-сайты, а разработчики. В этом есть смысл, ведь чего-то гораздо большего, чем личный блог, конечный пользователь CMS редко является разработчиком, и чаще всего клиенты нанимают кого-то для создания своего веб-сайта. October CMS построена на основе этой концепции и ориентирована на то, чтобы быть платформой для разработчиков с предположением, что разработчики имеют базовые знания в области HTML, CSS и PHP.

### **2.2. Область применения October CMS**

October CMS представляет собой нечто немного большее, чем традиционная CMS. Ядро легкое и создает прочную основу, поскольку основой является фреймворк `laravel`. По умолчанию доступны только основные функции, что означает отсутствие ненужных функций, которые вы никогда не сможете использовать. Платформа очень гибкая, и дополнительные функции добавляются путем установки плагинов с октябрьского рынка.

Поскольку из коробки есть только минимальная функция, и каждая функция добавляется с помощью плагинов, то есть устанавливаются только те функции, которые нужны для работы конкретного веб-сайта. Благодаря такому модульному подходу October CMS можно использовать для создания чего угодно, от простого веб-сайта, такого как статический веб-сайт или личный блог, до сложной системы корпоративного уровня.

Ключевые особенности:

- Простой интерфейс
- Широкие возможности настройки
- Открытый исходный код
- Удобный backend-интерфейс
- Расширяемость платформы
- Шаблонизатор TWIG
- Быстрое развитие
- Простая AJAX-инфраструктура
- Файловые шаблоны CMS

### 2.3. Архитектура October CMS

Данная система управления сайта построена на PHP фреймворке Laravel. Фреймворк достаточно прочно зарекомендовал себя на рынке как удобный и практичный продукт. Философия как Laravel, так и October CMS достаточно схожи, и неудивительно, что October использует этот тип фреймворка. Это сочетание обеспечивает простоту, скорость и элегантности в программировании.

Laravel обеспечивает простую и интуитивно понятную маршрутизацию, используя простые имена для идентификации маршрутов, а не длинные имена путей. Использование идентификаторов маршрутов также упрощает поддержку приложений, поскольку имя маршрута можно изменить в одном месте, а не менять его повсюду.

Laravel включает в себя ряд функций безопасности, включая аутентификацию пользователей, авторизацию ролей пользователей, проверку электронной почты, службы шифрования, хеширование паролей и функции сброса пароля.

Фреймворк также обеспечивает контроль версий для баз данных приложений с помощью миграций. Миграции отслеживают, как база данных изменялась с течением времени, упрощая работу с ней.

## 2.4. Возможности October CMS

Директория Theme содержит в себе набор тем, которые могут использоваться системой. Этот каталог содержит все подкаталоги и файлы, необходимые для работы. По умолчанию October CMS запускает демонстрационную тему. Создать свой собственный так же просто, как создать новый каталог, файл конфигурации темы (theme.yaml) в корне этого каталога, и каталог страниц с файлом подкачки в нем. А пока давайте изучим демонстрационную тему, чтобы понять внутреннюю работу.

Каждая тема состоит из подкаталогов для страниц, фрагментов, макетов, файлов содержимого и ресурсов. Каждый из этих каталогов может содержать подкаталоги, что позволяет упростить крупномасштабные проекты. Как выглядит структура файла с темой можно увидеть на рис.2.1.

```
themes/  
  demo/  
    assets/  
      css/  
      images/  
      javascript/  
      ...  
    content/  
    layouts/  
    pages/  
    partials/  
    theme.yaml
```

Рис.2.1. Структура файла с темой в October CMS

Файлы шаблонов имеют простую структуру и используют разметку Twig. Twig - это PHP-движок шаблонов, улучшающий и ускоряющий создание шаблонов. Любой файл страницы, макета и частичного шаблона состоит из трех частей:

- Конфигурация
- PHP-код
- Twig разметка

Раздел конфигурации структурирован как файл `php.ini` и отмечен как завершенный двумя символами `=`, уступая место разделу PHP. Раздел PHP является необязательным для любого файла шаблона, а также помечается как завершенный путем ввода еще двух символов `=`. Наконец, разметка Twig содержит фактическое содержимое файла шаблона. На рис.2.2 показано, как может выглядеть файл шаблона, взятый с их сайта:

```
url = "/blog"
layout = "default"
==
function onStart()
{
    $this['posts'] = ...;
}
==
<h3>Blog archive</h3>
{% for post in posts %}
    <h4>{{ post.title }}</h4>
    {{ post.content }}
{% endfor %}
```

Рис.2.2. Файл шаблона в October CMS

Файлы папки `pages` описывают страницы разрабатываемого сайта. Они принимают три обязательных параметра в разделе конфигурации:

1. `url` - адрес страницы (обязательно)
2. `title` - название страницы (обязательно)
3. `layout` – макет страницы, который может указывать на файл макета (необязательно)
4. `description` – описание страницы в панели администратора (необязательно)

URL-адрес страницы может принимать параметры в зависимости от рассматриваемой страницы. Также можно указать страницы 404 и ERROR. На рис.2.3 можно увидеть простой пример макета страницы:

```
url = "/"
title = "Welcome"
description = "The home page."
==
<h1>Welcome Home</h1>
```

Рис.2.3. Пример структуры страницы в October CMS

В October CMS существуют так называемые Partial файлы. Из названия можно понять, что они содержат частичные фрагменты кода. Partial файлы обладают мощными возможностями, поскольку их можно вызывать и повторно использовать на страницах, макетах или других частичных файлах. Для этого можно использовать тег `% partial %` для вызова Partial из другого файла. Partial поддерживают только один параметр конфигурации:

`description` — описание Partial в панели администратора (необязательно). Базовый пример partial можно увидеть на рис.2.4.

```
description = "Right sidebar."
==
<div class="sidebar">
  <aside>
    {% partial "recent-posts" %}
  </aside>
</div>
```

Рис.2.4. Пример вызова partial в файле October CMS

Также важно отметить возможность создания макетов или Layouts в October CMS. Они позволяют, например выделить определенную повторяющуюся структуру множества страниц и в дальнейшем менять только их контент без ненужного повторения кода. Так, можно выделить в Layout header и footer сайта, если на всех страницах они одинаковы.

На рис.2.5 можно увидеть пример макета layout.

```
name = "Default"
description = "The default layout for our template"
==
<!doctype html>
<html>
  <body>
    {% page %}
  </body>
</html>
```

Рис.2.5. Пример Layout в October CMS

Предположим, мы назвали этот файл макета как default.htm и поместили его в каталог макетов. Пример вызова Layout на странице можно посмотреть на рис.2.6 .

```
title = "Welcome"
url = "/"
layout = "default"
description = "Welcome home!"
==
<h2>Welcome Home!</h2>
<p>...</p>
```

Рис.2.6. Пример вызова Layout на странице в October CMS

Плагины - это основа расширения October CMS за пределы стандартного функционала. Они могут определять компоненты, добавлять внутренние страницы, взаимодействовать и редактировать функциональность других плагинов. Они легко описываются, настраиваются и находятся в каталоге /plugins.

Плагины находятся в подкаталоге plugins каталога приложения. Пример структуры каталога можно увидеть на рис.2.7 :

```
|— plugins
|   |— acme      <== Author Name
|       |— blog <== Plugin Name
|           |— classes
|           |— components
|           |— controllers
|           |— models
|           |— updates
|           |— plugin.php
```

Рис.2.7. Пример структуры каталога плагина в October CMS

Серверная часть October CMS реализует Model-View-Controller (MVC) паттерн. Контроллеры управляют внутренними страницами и реализуют различные функции, такие как формы и списки.

Каждый контроллер представлен скриптом PHP, который находится в подкаталоге /controllers каталога плагинов. Представления контроллера - это файлы



.htm, которые находятся в каталоге представлений контроллера. Имя каталога представления контроллера соответствует имени класса контроллера, написанному строчными буквами. Каталог представления также может содержать файлы конфигурации контроллера.

Существует два способа настройки плагинов — с помощью формы настроек на backend и с помощью конфигурационных файлов. Использование параметров базы данных с внутренними страницами обеспечивает лучший пользовательский интерфейс, но требует дополнительных затрат на первоначальную разработку.

В October CMS можно создавать модели для хранения параметров в базе данных, реализуя поведение `SettingsModel` в классе модели. Эта модель может быть использована непосредственно для создания формы внутренних настроек.

Классы моделей параметров должны расширять класс `Model` и реализовывать поведение `SettingsModel`. Модели настроек, как и любые другие модели, должны быть определены в подкаталоге `models` каталога плагинов.

## 2.5. Выводы

Таким образом, можно сказать, что October CMS - это легкая, возвращающаяся к основам система управления контентом, построенная на Laravel и призванная упростить рабочий процесс веб-разработки. Система может похвастаться своей простотой и быстротой. Он масштабируется и расширяется с помощью системы плагинов, легко обслуживается благодаря своей файловой системе и позволяет легко создавать административные внутренние интерфейсы.

## ГЛАВА 3. АДАПТАЦИЯ ВЕРСТКИ К OCTOBER CMS

В этой главе будет рассмотрен процесс адаптации верстки к October CMS. Здесь же будет представлен процесс создания плагина, зависимостей, создание моделей, контроллеров и миграций в базу данных. Также будет показана работа с установленным плагином Magic Forms, который позволяет обрабатывать заявки на сайте.

### 3.1. Технология адаптации верстки

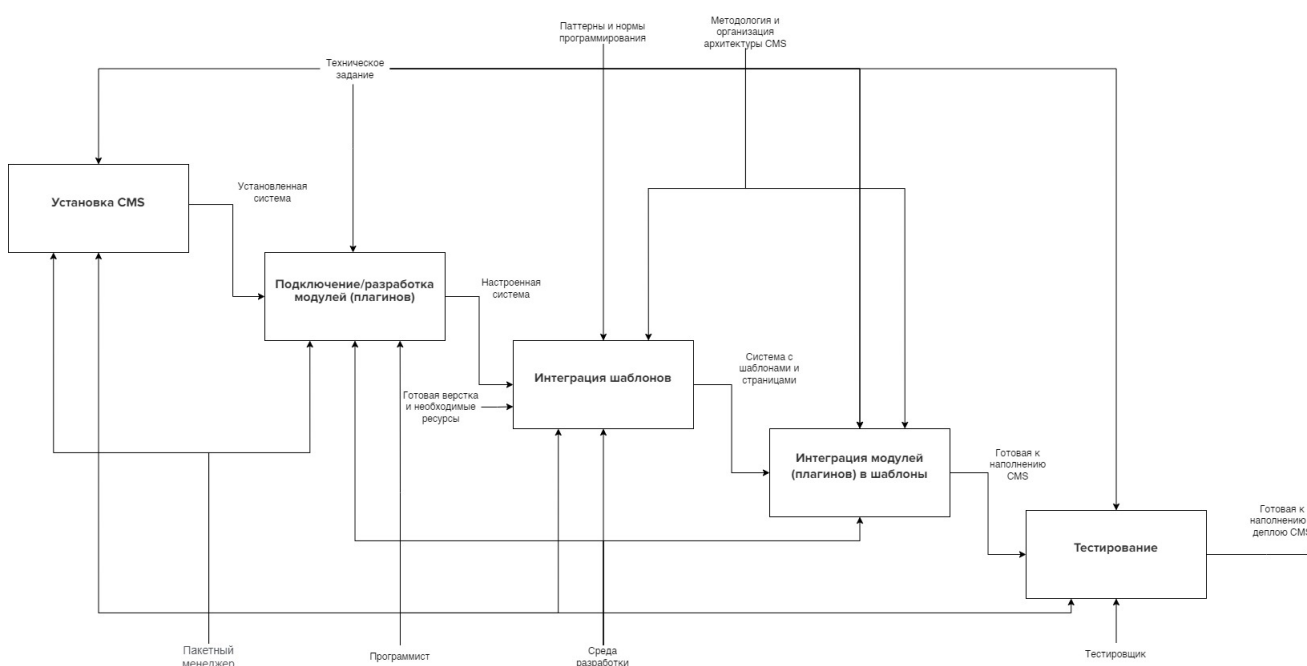


Рис.3.1. IDEF0 диаграмма, описывающая процесс адаптации верстки к CMS

Прежде чем приступить к самой адаптации верстки необходимо было изучить этот процесс. На основании изученных материалов, были составлены этапы процесса, у каждой функции были проанализированы входы, выходы, управление и механизмы. Сформированные этапы процесса позволяют более точно представить себе данный процесс:

- А. Установка CMS
- В. Подключение/разработка модулей (плагинов)
- С. Интеграция шаблонов
- Д. Интеграция модулей (плагинов) в шаблоны
- Е. Тестирование

Первым этапом является установка CMS, здесь важной особенностью является то, какую систему выбрать, это зависит от конкретного проекта и контекста разработки, то есть от технического задания. Этот этап несмотря на свою простоту, является важным, поскольку недальновидный выбор системы управления может сказаться на расширяемости проекта, на ограничении его возможностей или наоборот - сделать разработку более простой, удобной и быстрой. Результатом является готовая к разработке система управления.

Далее подключение и/или разработка модулей. Здесь важно проанализировать уже имеющийся функционал системы, понять, стоит ли использовать уже имеющиеся плагины или имеет смысл разработать свой. Если смотреть с точки зрения устройства Headless CMS, то на этом этапе работа с самой CMS как таковой уже бы закончилось, но не закончилась бы адаптация. То есть результатом здесь будет готовая к выводу на front end часть сайта.

Третьим этапом является интеграция шаблонов. Здесь важно проанализировать имеющуюся верстку, её ресурсы, выделить повторяющиеся фрагменты кода, выделить их в шаблоны. Здесь же создаются страницы в соответствии с тем, как это организовано в конкретной системе управления. Результатом будет готовая back end система, которая будет иметь пока статическое наполнение страниц. На страницы можно будет зайти, но с базы данных подгружаться данные не будут.

Последним перед тестированием этапом, является интеграция ранее созданных модулей. Все открытые и административные части CMS подвергаются динамическому программированию, исходя из поставленных задач в техническом задании. Результатом данного этапа является готовый к наполнению как на стороне Backend, так и на стороне Frontend сайт.

Заключительным этапом является тестирование и апробация сайта, здесь проверяется корректность отображения информации со стороны пользователя и сравнение соответствует ли она введенной информации в панели администратора. На рис.3.1 можно посмотреть диаграмму IDEF0, которая отображает структуру процесса адаптации верстки независимо от конкретной системы управления контентом.

### **3.2. Описание проекта**

Прежде чем разбирать адаптацию верстки к CMS, для начала необходимо разобраться, что представляет из себя проект, какие страницы там присутствуют,

а также их взаимосвязь. Необходимо также выделить конкретные сущности, их характеристики, чтобы проще было описывать модели данных. На рис.3.2 можно увидеть структуру сайта.

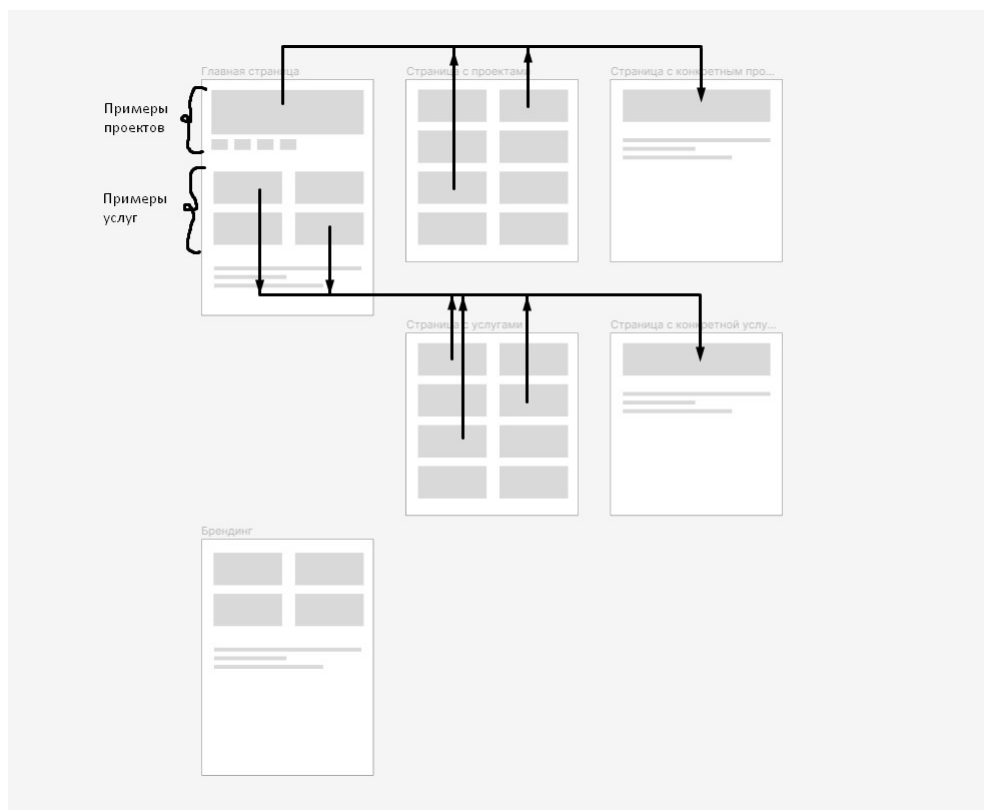


Рис.3.2. Структура сайта

Сайт разрабатывается для компании Parabrand, компания занимается тем, что помогает другим компаниям в продвижении, рекламе и популяризации. На сайте имеется 6 страниц:

1. Главная страница
2. Страница с услугами
3. Страница с конкретной услугой
4. Страница с имеющимися проектами
5. Страница с конкретным имеющимся проектом
6. Страница с описанием брендинга

На каждой странице есть меню, в котором можно попасть: на главную, на страницу с перечнем услуг, на страницу с уже выполненными проектами, и на страницу с описанием брендинга. То есть практически с любой страницы можно попасть на любую страницу. Также на каждой странице есть список партнеров и футер, в котором есть форма обратной связи, контакты и логотип.

На главной странице в хедере имеется слайдер с фотографией и описанием уже имеющихся проектов, к каждому проекту можно перейти, нажав кнопку "Подробнее". Далее располагается секция с услугами, где перечислены первые 8 имеющихся на сайте услуг, кликнув на каждую из них можно также перейти на конкретную услугу. Ниже можно увидеть секцию с тремя иконками, рядом с которыми есть заголовок и описание. Далее располагается секция с логотипами партнеров компании.

На странице с услугами перечислены все услуги, которые есть у компании с некоторым текстом. То же самое и у страницы с проектами, где перечислены все проекты, но здесь их можно сортировать по услугам.

На странице "Брендинг" имеется главная фотография, и некоторое количество блоков с текстом и фотографиями.

### 3.3. Создание темы: шаблоны, страницы, ассеты

Первоначально была создана тема, в October CMS это можно сделать посредством интерфейса или просто создав папку в директории themes со своим именем, там же создать папку с макетами (layouts), страницами (pages), фрагментами (partials) и необходимыми файлами (assets). Помимо всего этого обязательно должен присутствовать файл theme.yaml, в котором прописывается название темы, автор, описание и другие настройки.

В папке layouts был создан файл layout.htm, с помощью которого можно было выделить повторяющиеся элементы на каждой странице. Как выше было сказано - это меню, footer и блок с партнерами. Для этого была размещена соответствующая html разметка, а в месте, где должен находиться сам контент страницы была вставлена переменная с соответствующим twig синтаксисом: `{% page %}`.

Помимо всего прочего, пути до изображений, js скриптов, иконок, css стилей необходимо было переопределить, поскольку они находились в другой директории. Для этого использовалась конструкция `{{ 'path' | theme }}`. Как подключались ассеты, можно посмотреть на рис.3.3. Полный код можно увидеть в приложении 1.

Далее были созданы страницы в папке pages, к каждой странице был подключен созданный ранее layout в разделе конфигурации, их можно увидеть на рис.3.4. Все страницы можно посмотреть в приложении 2.

```

<!DOCTYPE html>
<html lang="ru">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>{{ this.page.title }}</title>

  <link rel="stylesheet" href="{{ '/assets/swiper.css' |theme }}" />
  <link rel="stylesheet" href="{{ '/assets/style.css' |theme }}" />
  <link rel="stylesheet" href="{{ '/assets/media.css' |theme }}" />

  {% styles %}

  <link rel="icon" type="image/x-icon" href="{{ '/assets/images/favicon.svg' |theme }}" />
</head>

```

Рис.3.3. Пример подключения необходимых ассетов в October CMS

```

url = "/our-projects/:slug"
layout = "layout"
title = "certain-project"

[builderDetails projectsList]
modelClass = "Emil\Papabrand\Models\Projects"
identifierValue = "{{ :slug }}"
modelKeyColumn = "slug"
displayColumn = "slug"
notFoundMessage = "Record not found"
==
{% set record = projectsList.record %}
{% set displayColumn = projectsList.displayColumn %}
{% set notFoundMessage = projectsList.notFoundMessage %}

<section class="bread">
  <div class="container">
    <div class="bread-container">
      <a href="/">Главная</a>
      <a href="/our-projects">Наши проекты</a>
      <a href="{{ record.slug }}">{{ record.title }}</a>
    </div>
  </div>
</section>

<section class="certain-service-slider">
  <div class="container">
    <div class="certain-service-slider-container">
      <div class="swiper-buttons-container">
        <div class="swiper-button-prev">
          
        </div>

```

Рис.3.4. Пример создания страницы в October CMS

На этом создание темы, шаблона и страниц закончилось, но в дальнейшем все равно необходимо было обращаться к созданным страницам для подключения компонентов и внесения данных из backend во frontend.

Структуру темы можно посмотреть на рис.3.5:

Код шаблона, страниц и фрагментов можно посмотреть в приложении 2.

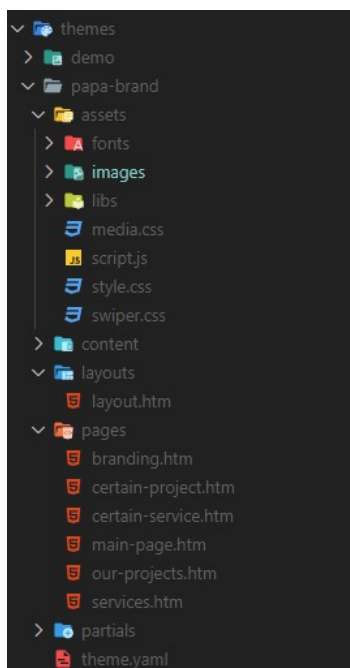


Рис.3.5. Пример создания темы в October CMS

### 3.4. Создание плагина

Для создания плагина необходимо было соблюсти структуру папок в папке `plugins`. Сначала создается папка с именем разработчика плагина, затем в ней создается папка с именем самого плагина, в ней же и будут находиться все файлы плагина. Также необходимо создать регистрационный файл плагина `Plugin.php`. Скрипт регистрации должен содержать класс с именем `Plugin`, который расширяет `/System/Classes/PluginBase` класс.

Также были созданы папки `controllers`, `models`, `lang` и `updates`. В папке `updates` будут располагаться `php` файлы с необходимыми миграциями, а также файл `version.yaml`, в котором будут указаны версии плагина и соответствующие им миграции.

#### 3.4.1. Создание моделей и контроллеров

Для нашего плагина нам необходимо было создать как минимум три модели: услуга, проект и партнер. В October CMS это можно сделать через помощник `artisan` командой `"php artisan create:model Author.PluginName ModelName"`. После чего создадутся соответствующая папка и файл данной модели в папке `models`. UML диаграмму с классами, описывающими модели можно посмотреть на рис.??.

В папках располагаются файлы `columns.yaml` и `fields.yaml`. Эти два файла необходимы для работы наших моделей в панели администратора OctoberCMS.



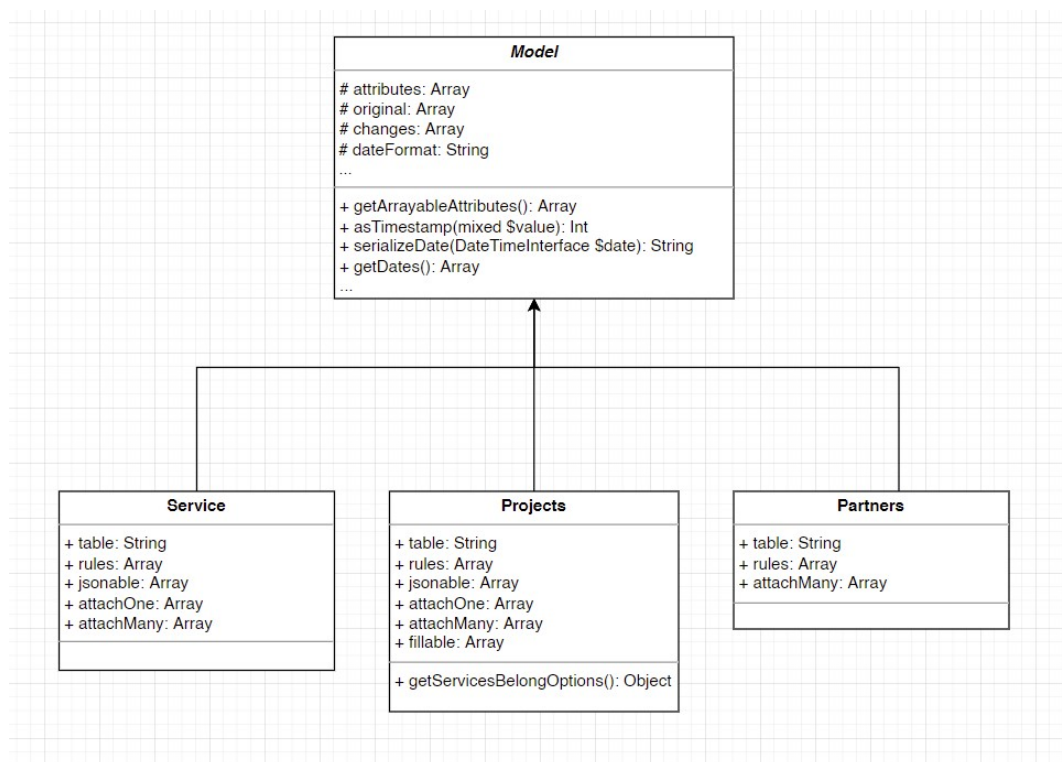


Рис.3.6. UML диаграмма классов, описывающих модели

columns.yaml с одержит параметры и информацию столбцов, которые нужно выводить при просмотре списка элементов модели в панели администратора. fields.yaml с одержит параметры и информацию полей формы, которые нужно выводить при редактировании или создании элемента в панели администратора.

Так, например, выглядит файл Projects.php на рис.3.7:

Здесь содержится класс Projects, который наследуется от класса Model. В нем содержатся переменная table, в которой находится название таблицы, к которой будет привязана модель, rules, в которой можно прописать правила валидации, у нас она везде пустая.

Переменные attachMany и attachOne необходимы для того, чтобы модель понимала, откуда будут браться файлы (в нашем случае изображения). Путь до изображений хранится непосредственно в October CMS, не в базе данных. Также была введена функция getServicesBelongOptions(), которая возвращает список услуг, которые в дальнейшем можно было выставить к каким услугам данный проект будет относиться.

То, к каким услугам будет принадлежать конкретный проект должно было храниться в базе данных в конкретной таблице в конкретной записи в столбце services\_to\_show в качестве строки. Для этого необходимо было пометить эту структуру как jsonable, в этой связи и добавилась переменная jsonable. Так, October



```

<?php namespace Emil\Papabrand\Models;

use Model;

/**
 * Model
 */
class Projects extends Model
{
    use \October\Rain\Database\Traits\Validation;

    /**
     * @var string The database table used by the model.
     */
    public $table = 'emil_papabrand_projects';

    /**
     * @var array Validation rules
     */
    public $rules = [
    ];

    public $attachMany = [
        'project_images' => 'System\Models\File'
    ];

    public $attachOne = [
        'descr_image' => 'System\Models\File'
    ];

    public function getServicesBelongOptions(){
        $model = Services::lists('title', 'slug');
        return $model;
    }

    protected $jsonable = [
        'descr', 'services_belong'
    ];
}

```

Рис.3.7. Пример файла, описывающего модель в October CMS

CMS сможет распарсить информацию и достать необходимую информацию о том, к каким услугам принадлежит данный проект.

Помимо всего прочего, в папке, которая соответствует данной модели необходимо было настроить файлы `columns.yaml` и `fields.yaml`. Так, например в файле `columns.yaml` модели `Projects` были введены следующие колонки для описания записей `Projects`: `id`, `created_at`, `updated_at`, `title` и `services_belong`. Также были выставлены такие характеристики полей, как `label` (название), `type` (тип данных), `invisible` (видимость) и `sortable` (будет ли данный столбец участвовать в сортировке записей). Пример заполнения файла `columns.yaml` можно увидеть на рис.3.8.

В файле `fields.yaml` описываются поля, которые будут доступны при создании/редактировании конкретной записи о проекте. Здесь были введены такие поля, как название, текстовое описание, `url` страницы, который автоматически преобразовывается из названия проекта, однако его можно поменять. В блоке "Описание" выставлен тип "Repeater" поскольку здесь можно будет добавлять блоки с описанием и их можно будет добавлять опционально или несколько. В самом репиторе имеются поля для подзаголовка, изображения, текста до и после фото.

```
columns:
  id:
    label: id
    type: number
    invisible: true
  created_at:
    label: created_at
    type: datetime
    invisible: true
  updated_at:
    label: updated_at
    type: datetime
    invisible: true
  title:
    label: 'Название проекта'
    type: text
    sortable: true
  services_belong:
    label: 'К каким услугам принадлежит'
    type: text
    sortable: true
```

Рис.3.8. Пример заполнения файла columns.yaml в модели в October CMS

В дальнейшем на каждой странице с проектом можно будет добавлять множество блоков с описанием проекта: текст до фото или текст после фото можно будет не вставлять, как и фотографию, то есть можно юудет комбинировать эти блоки любым образом. Пример заполнения файла fields.yaml можно посмотреть на рис.3.9.

```
fields:
  title:
    label: 'Название проекта'
    span: auto
    type: text
  project_images:
    label: 'Изображения проекта'
    mode: image
    useCaption: true
    thumbOptions:
      mode: crop
      extension: auto
    span: auto
    type: fileupload
  slugg:
    label: 'URL страницы'
    span: auto
    preset:
      field: title
      type: slug
    type: text
  services_belong:
    label: 'К каким услугам'
    span: auto
    type: checkboxlist
  descr:
    label: 'Описание проекта'
    prompt: 'Add new item'
    style: default
    span: full
    type: repeater
    form:
      fields:
        descr_subtitle:
          label: Подзаголовок
          span: auto
          type: text
```

Рис.3.9. Пример заполнения файла fields.yaml в модели в October CMS

Аналогичные действия были выполнены и для создания других моделей.

### 3.4.2. Создание миграций и контроллеров

Поскольку в файлах с моделями мы указали таблицы, хранящие записи, которые описывает модель, нам необходимо собственно создать данные таблицы. В папке updates artisan уже создал необходимые файлы миграций, осталось только настроить их. Создается класс, наследующийся от класса Migration. В нем обязательно должны (уже присутствуют) функции up() и down(), с помощью которых в дальнейшем можно будет применять или удалять выполненные изменения данной миграции при манипуляции с версиями плагина. Рассмотрим файл с миграцией для таблицы с проектами, представленный на рис.3.10. Все файлы миграций можно посмотреть в приложении 1.

```
<?php namespace Emil\Papabrand\Updates;

use Schema;
use October\Rain\Database\Updates\Migration;

class BuilderTableCreateEmilPapabrandProjects extends Migration
{
    public function up()
    {
        Schema::create('emil_papabrand_projects', function($table)
        {
            $table->engine = 'InnoDB';
            $table->increments('id')->unsigned();
            $table->timestamp('created_at')->nullable();
            $table->timestamp('updated_at')->nullable();
            $table->string('title')->nullable();
            $table->text('descr')->nullable();
            $table->text('services_belong')->nullable();
        });
    }

    public function down()
    {
        Schema::dropIfExists('emil_papabrand_projects');
    }
}
```

Рис.3.10. Пример заполнения файла миграции в October CMS

Здесь были указаны все, соответствующие ранее созданным моделям, поля, их типы данных и nullable значения. В функции down просто прописывается, что при понижении версии, если данная таблица существует, то она будет удалена.

После этого мы должны были указать OctoberCMS, что у нас в миграционных файлах есть изменения, и при выполнении обновления нужно применить их.

В файле version.yaml в папке /updates и были добавлены новые версии с перечислением файлов, которые нужно выполнить при обновлении. Пример регистрации версий на рис.3.11.

```
1.0.1:
- 'Initialize plugin.'
1.0.2:
- 'Created table emil_papabrand_partners'
- builder_table_create_emil_papabrand_projects_page.php
1.0.3:
- 'Created table emil_papabrand_services'
- builder_table_create_emil_papabrand_services.php
```

Рис.3.11. Регистрация версий в плагине в October CMS

У нас имелись модели, с заполненными миграционными файлами. И все что нам нужно было сделать - это вызвать обновление, которое выполнит эти скрипты. Для этого в корневой папке была выполнена команда: `php artisan october:up`

Для создания интерфейса и управления содержимым и категориями в админке, нам необходимо было создать контроллеры для каждой модели. Делается это как и с моделями - через `artisan`. Однако название контроллеров должно совпадать с названием моделей, только во множественном числе. Так, например, контроллер для модель проект был создан с помощью команды: `php artisan create:controller Emil.PapaBrand Projects`. В наших контроллерах необходимости что-то натсраивать не было, поскольку `artisan` автоматически всё создал и настроил.

### 3.4.3. Создание Backend меню

Для того, чтобы у нас был доступ к соответствующим записям, нам необходимо было создать backend меню, в котором были бы айтемы, соответствующие нашим моделям. Для этого в файле, лежащем рядом с `plugin.php` `plugin.yaml` необходимо было указать структуру: `plugin` и `navigation`, где в `plugin` необходимо было разместить информацию о плагине, а в блоке `navigation` необходимо разместить соответствующую структуру айтемов: главная, услуги, наши услуги. Для каждого айтема в меню указывалось название, `url` и иконка из встроенных системных иконок October CMS. Пример заполнения файла `plugin.yaml` можно увидеть на рис.3.12.

```
plugin:
  name: 'emil.papabrand::lang.plugin.name'
  description: 'emil.papabrand::lang.plugin.description'
  author: Emil
  icon: oc-icon-coffee
  homepage: ''
navigation:
  main-menu-item:
    label: 'Papa Brand'
    url: emil/papabrand/papabrand
    icon: icon-male
    sideMenu:
      side-menu-item2:
        label: Главная
        url: emil/papabrand/papabrand/update/1
        icon: icon-sitemap
      side-menu-item:
        label: Услуги
        url: /emil/papabrand/services
        icon: icon-tags
      side-menu-item6:
        label: 'Страница "Наши услуги"'
        url: emil/papabrand/servicespage/update/1
        icon: icon-th-large
      side-menu-item4:
        label: 'Наши проекты'
        url: emil/papabrand/projects
        icon: icon-trello
      side-menu-item5:
        label: 'Страница "Наши проекты"'
        url: emil/papabrand/projects_page/update/1
```

Рис.3.12. Пример заполнения файла `plugin.yaml` в October CMS

Перейдя в панель управления, можно было увидеть соответствующее меню и проверить работоспособность плагина - рис.3.13.

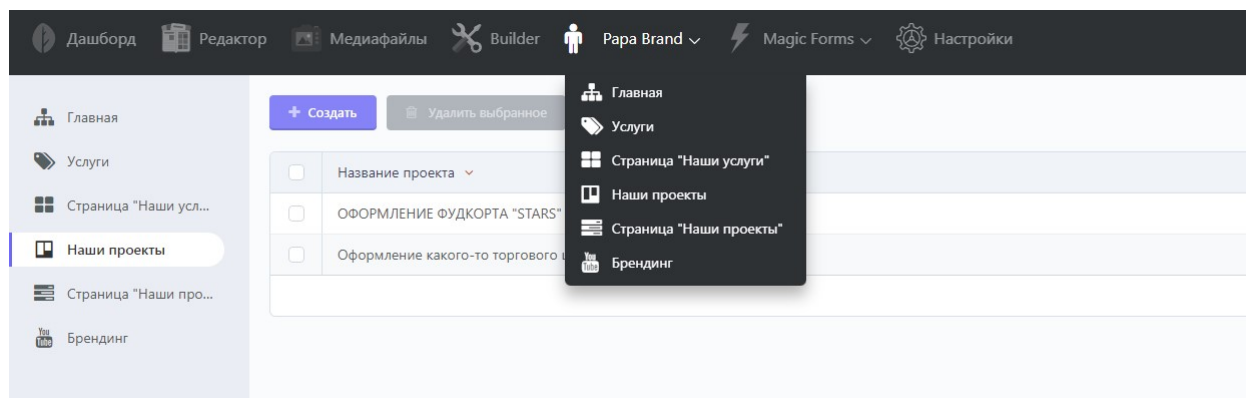


Рис.3.13. Backend меню для плагина в October CMS

#### 3.4.4. Работа с представлениями

Когда данные были готовы к тому, чтобы появиться на сайте, необходимо было вернуться к страницам и вставить в них необходимую twig разметку.

На главной странице, как было видно из описания, необходимо разместить все проекты в слайдере. Что касается секции с услугами - дизайн и верстка рассчитаны только на 8 айтемов, поэтому необходимо вывести только 8 первых услуг из всех услуг. Делалось это достаточно просто: для начала необходимо было в панели на страницу добавить соответствующих компонент, настроить какую модель класса он будет использовать, а далее просто обращаться к этому компоненту через свойство records.

Так, например, можно было получить список всех проектов. Дальше этот список был занесен в определенную переменную и дальше, был вставлен цикл for, благодаря которому можно было пройти по всем проектам и достать из каждого из них необходимую нам информацию, такую как изображение, название и ссылку на данную страницу.

Для того, чтобы можно было связать эти страницы, на странице со списком проектов, необходимо было указать страницу, с конкретным проектом, то есть Details Page и Details Key Column, в нашем случае это slug, то есть это та часть, которая будет появляться в url в качестве страницы. На странице с конкретным проектом необходимо было также добавить компонент и указать предыдущую страницу и также указать Details Page и Details Key Column. В качестве url на странице с конкретным проектом необходимо было дописать certain-page/:slug.

То же самое проделывалось и для секции с услугами. Пример вывода списков проектов во Frontend представлен на рис.3.14.

```
{% set records = listPartners.records %}
{% set displayColumn = listPartners.displayColumn %}
{% set noRecordsMessage = listPartners.noRecordsMessage %}

{% set projects = projectsList.records %}
{% set detailsKeyColumn = projectsList.detailsKeyColumn %}
{% set detailsPage = projectsList.detailsPage %}
{% set detailsUrlParameter = projectsList.detailsUrlParameter %}

{% set servs = services_list.records %}
{% set detailsPageServ = services_list.detailsPage %}
{% set detailsKeyColumnServ = services_list.detailsKeyColumn %}
{% set detailsUrlParameterServ = services_list.detailsUrlParameter %}

<header>
  <div class="swiper-header-container">
    <div class="swiper-header">
      <div class="swiper-wrapper">
        {% for proj in projects %}
          <div class="swiper-slide" style="background-image: url('{{proj.project_images[0].path}}');">
            <div class="descr-outer-container">
              <div class="descr-container">
                <h2>{{proj.title}}</h2>
                <a href="{{ detailsPage|page({ (detailsUrlParameter): attribute(proj, detailsKeyColumn) ) }}">Подробнее</a>
              </div>
            </div>
          </div>
        {% endfor %}
      </div>
    </div>
  </div>
```

Рис.3.14. Пример вывода списков проектов во Frontend в October CMS

### 3.4.5. Настройка плагина для отправки почты

Было принято решение использовать уже имеющийся плагин для отправки почты под названием MagicForms от Martin. Для того, чтобы установить сторонний плагин в October CMS имеется возможность скачать их непосредственно из панели администрирования. В настройках во вкладке "Обновления" просто находится данный плагин, скачивается и устанавливается без всяких проблем. Также любой плагин можно установить через уже упомянутый ранее artisan командой: `php artisan plugin:install AuthorName.pluginName`.

После установки необходимо было добавить в layout.htm (поскольку именно там содержится форма обратной связи, то есть на любой странице) компонент стандартной ajax формы. В компоненте необходимо было указать email получателя, шаблон письма (имеется из коробки в плагине) и написать скрипт, который будет вызываться при успешной отправке письма, в нашем случае просто вызывалась javascript функция, которая вызывала модальное окно с информацией об успешной отправке почты. Помимо этого в самой форме необходимо было просто указать атрибут data-request со значением `genericForm ::onFormSubmit` и `form_token` в самой форме.

Помимо всего прочего в настройках самой CMS необходимо было настроить саму почту. В нашем случае использовалась почта от google, поэтому мы использовали smtp протокол, указав соответствующий порт, протокол шифрования, логин



и пароль. Также в настройках плагина был отредактирован шаблон для почты - переведен на русский язык, указаны соответствующие тема и описание письма.

В итоге мы получили рабочую форму обратной связи, заявки с которой можно просмотреть как на почтовом ящике, указаном в админке, так и непосредственно в панели администрирования - рис.3.15.

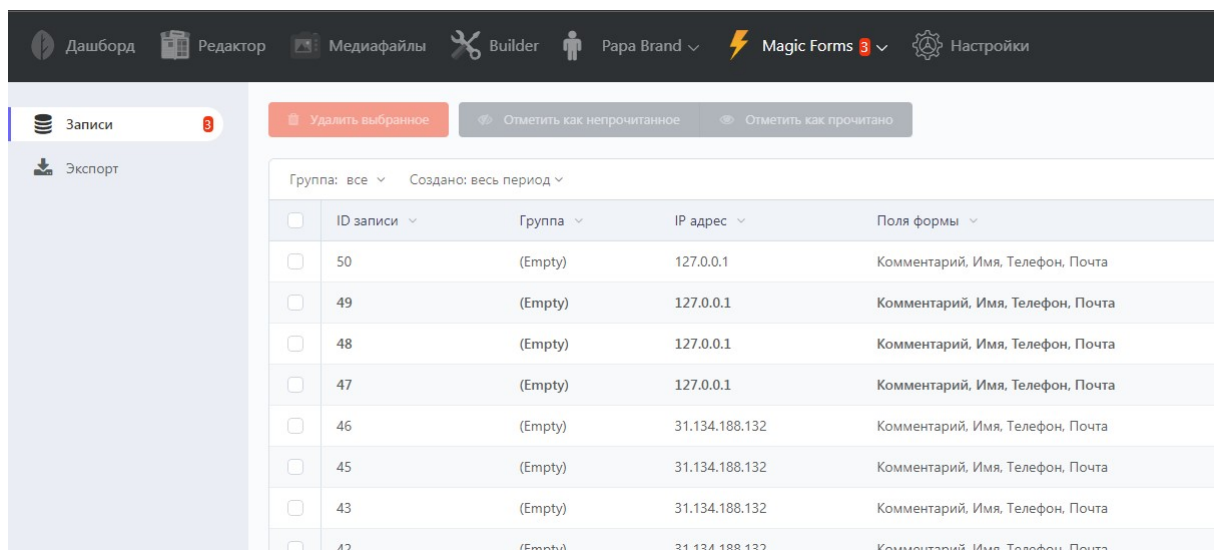


Рис.3.15. Просмотр заявок в панели администратора в October CMS

### 3.5. Выводы

Таким образом, в этой главе был рассмотрен процесс создания темы, разбиения ее на необходимые составляющие для дальнейшего использования. Также был создан плагин со всеми зависимостями и соответствующим интерфейсом в панели администрирования, в котором можно добавлять, редактировать, удалять и изменять записи соответствующих моделей. Также был рассмотрен процесс внедрения этих данных в верстку. Была настроена генерация страниц со всей необходимой информацией в них.

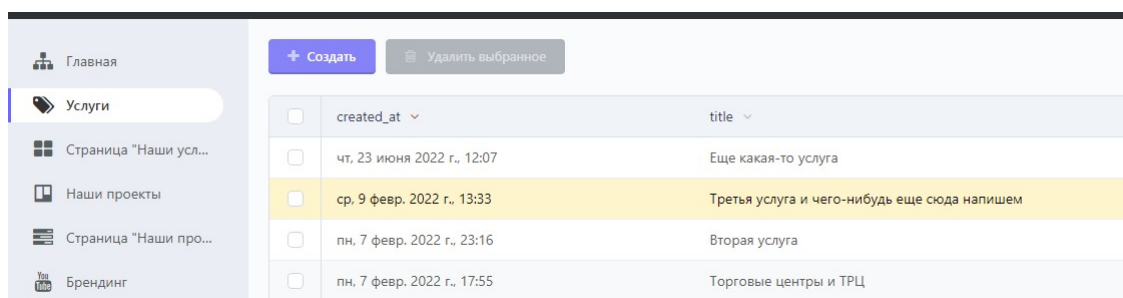
В конце была настроена отправка почты посредством стороннего плагина MagicForms, с помощью которого можно просматривать заявки не только на почтовом ящике, но и в панели администратора, что в дальнейшем заказчику поможет не потеряться в них. Готовый код плагина можно посмотреть в приложении 1.

## ГЛАВА 4. ПРОВЕРКА ПРАВИЛЬНОСТИ РАБОТЫ САЙТА: СОЗДАНИЕ ЗАПИСЕЙ, ПРОВЕРКА ИХ ОТОБРАЖЕНИЯ, ОТПРАВКА ПОЧТЫ

В данной главе проводится проверка правильности работы сайта. Здесь будет рассмотрено то, каким образом происходит редактирование информации в панели администрирования и как это в дальнейшем скажется на отображении сайта. Также будет выполнена проверка - должным ли образом всё отображается, создаются ли станции, можно ли на них перейти и так далее. Будет показано каким образом работает форма обратной связи - каким образом отображаются заявки в панели администратора и на почтовом ящике.

### 4.1. Создание записей с услугами

Для того, чтобы проверить, каким образом отображается секция на главной странице, странице со всеми услугами и странице с конкретной услугой было создано четыре записи. Их можно увидеть на рис.4.1.



<input type="checkbox"/>	created_at ▾	title ▾
<input type="checkbox"/>	чт, 23 июня 2022 г., 12:07	Еще какая-то услуга
<input type="checkbox"/>	ср, 9 февр. 2022 г., 13:33	Третья услуга и чего-нибудь еще сюда напишем
<input type="checkbox"/>	пн, 7 февр. 2022 г., 23:16	Вторая услуга
<input type="checkbox"/>	пн, 7 февр. 2022 г., 17:55	Торговые центры и ТРЦ

Рис.4.1. Создание записей об услугах в панели администрирования

Также в каждой из них были добавлены изображения, название и url. Помимо в качестве описания было добавлено два блока Repeater, в котором имелся подзаголовок, изображение, текст до и после. Это можно посмотреть на рис.4.2.

На главной странице всё отобразилось корректно - рис.4.3.

На странице "Наши услуги" также всё отлично отображалась. Это можно проверить на рис.4.4.

На странице конкретного проекта также отображалось всё корректно, была проверена работоспособность сортировки проектов по услугам, результат можно посмотреть на рис.4.5.



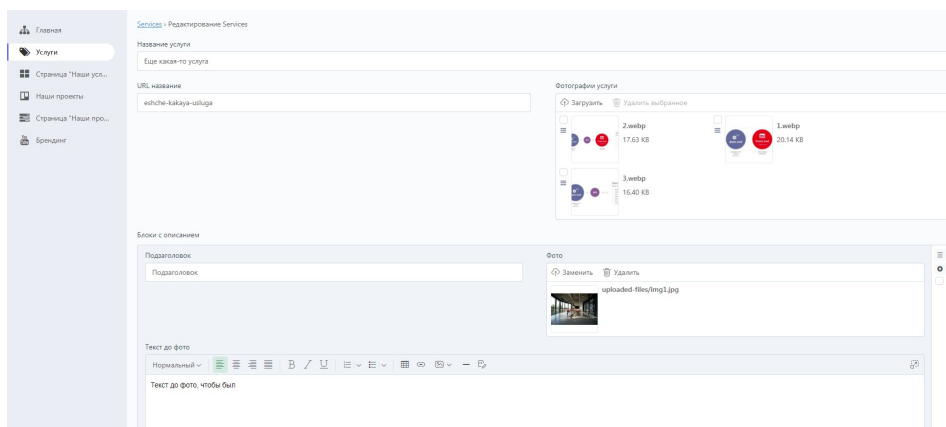


Рис.4.2. Пример внесения информации об услуге в панели администрирования

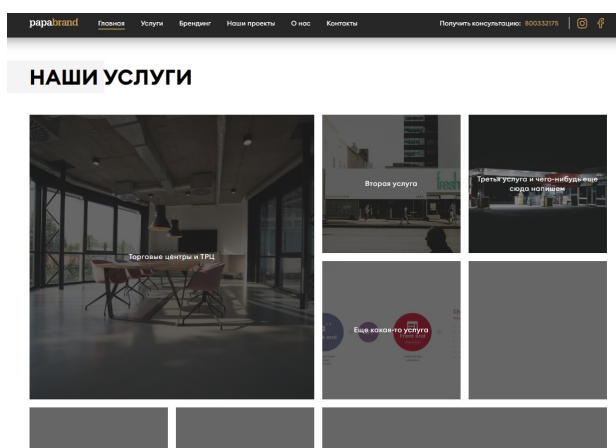


Рис.4.3. Проверка корректности работы главной страницы

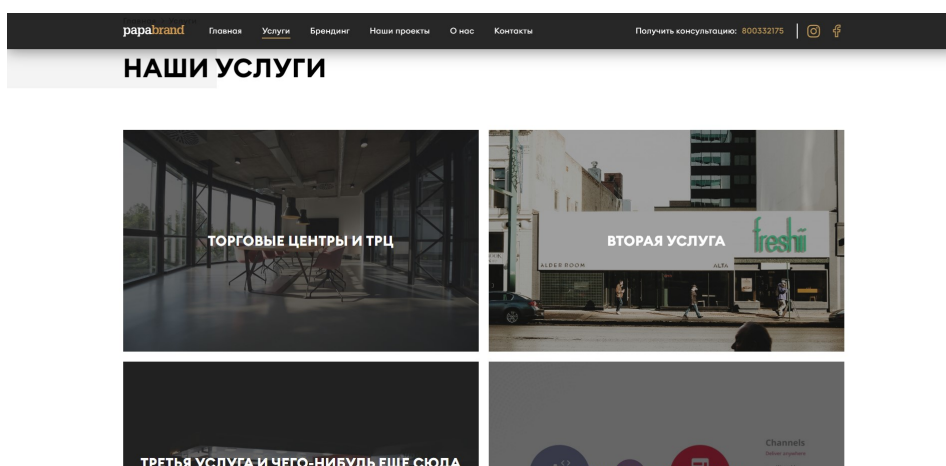


Рис.4.4. Проверка корректности работы страницы с услугами

## 4.2. Создание записей с проектами

Для того, чтобы проверить, каким образом отображается слайдер на главной странице, странице со всеми проектами и странице с конкретным проектом было создано две записи, их можно увидеть на рис.4.6.

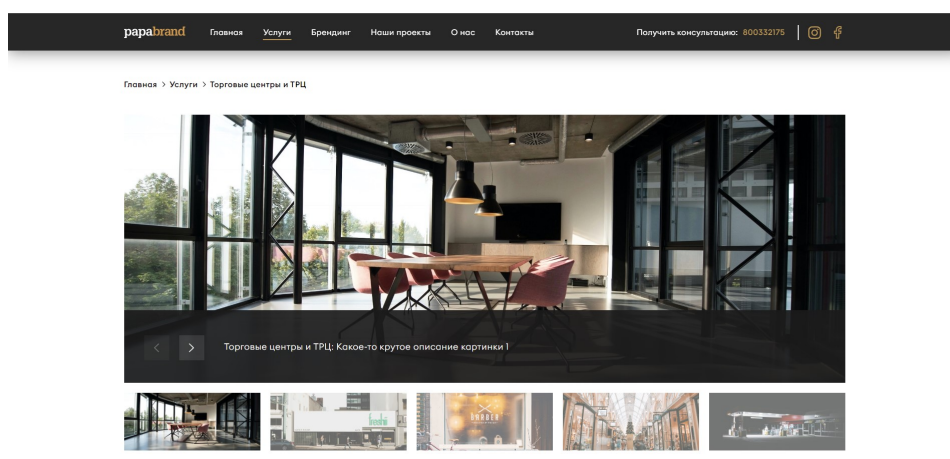


Рис.4.5. Проверка корректности работы страницы с конкретной услугой

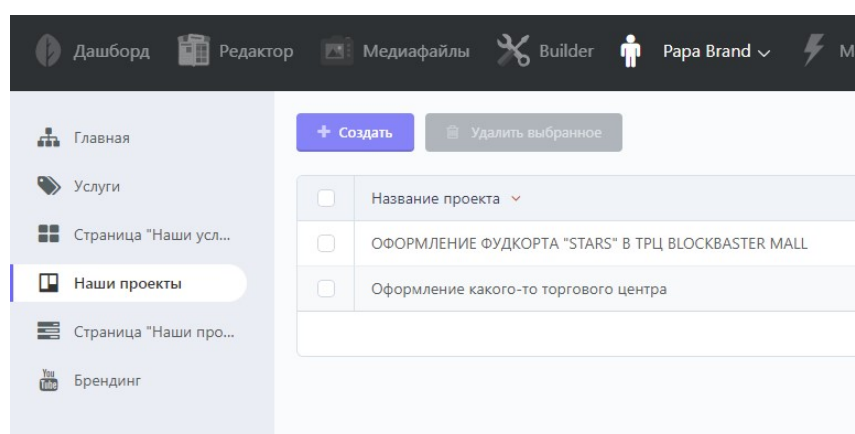


Рис.4.6. Создание записей о проектах в панели администрирования

Также в каждой из них были добавлены изображения, название и url. Помимо этого в качестве описания было добавлено два блока Repeater, в котором имелся подзаголовок, изображение, текст до и после - рис.4.7.

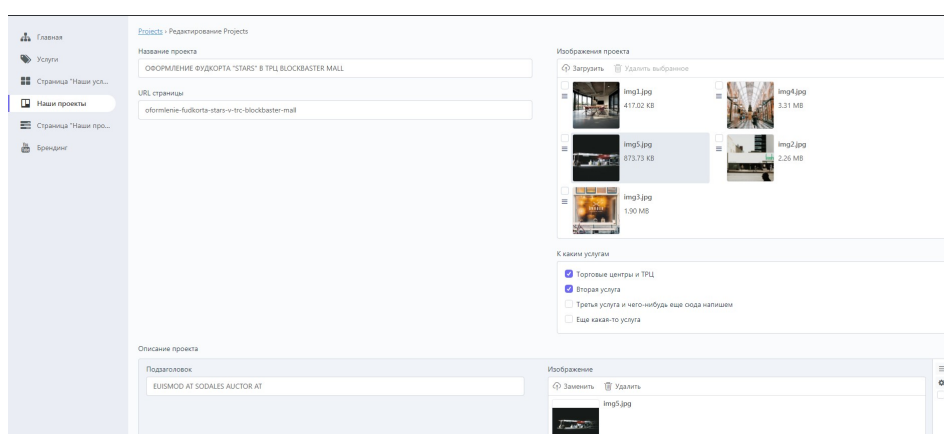


Рис.4.7. Пример создания записи о проекте в панели администрирования

На главной странице всё отобразилось корректно - рис.4.8.

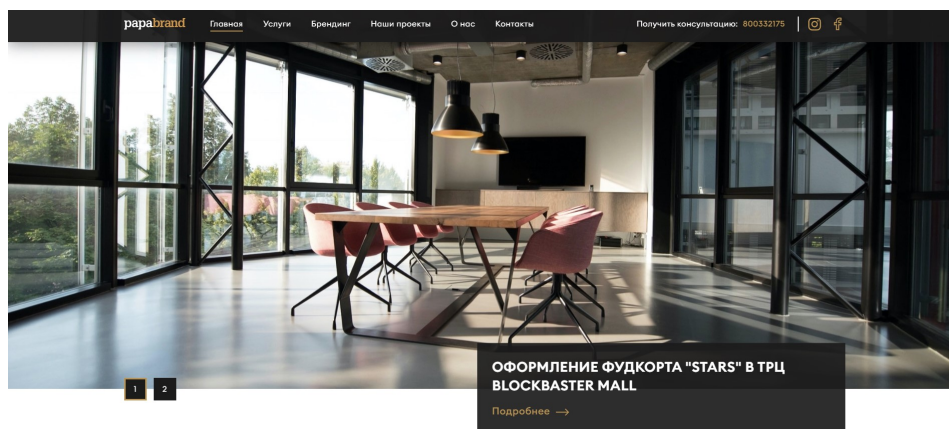


Рис.4.8. Проверка корректности работы главной страницы

На странице "Наши проекты" также всё отлично отображалась, была проверена работоспособность сортировки проектов по услугам. Это можно увидеть на рис.4.9.

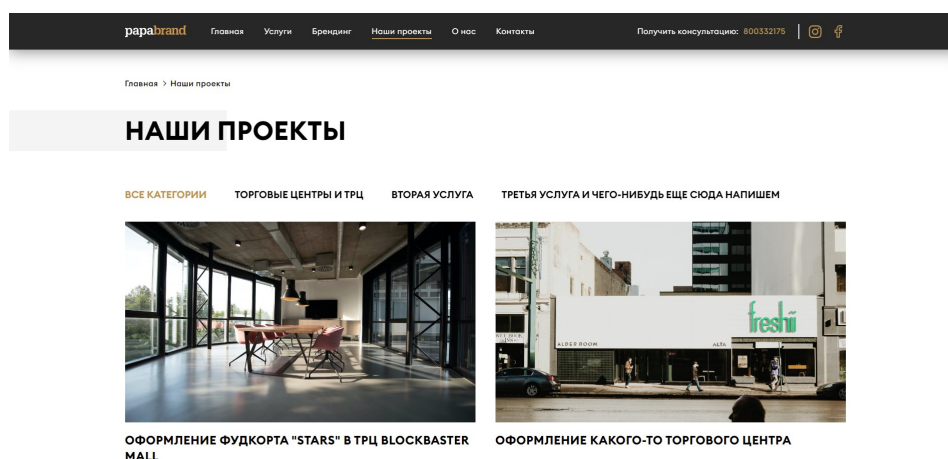


Рис.4.9. Проверка корректности работы страницы с проектами

На странице конкретного проекта также отображалось всё корректно, была проверена работоспособность сортировки проектов по услугам - рис.4.10.

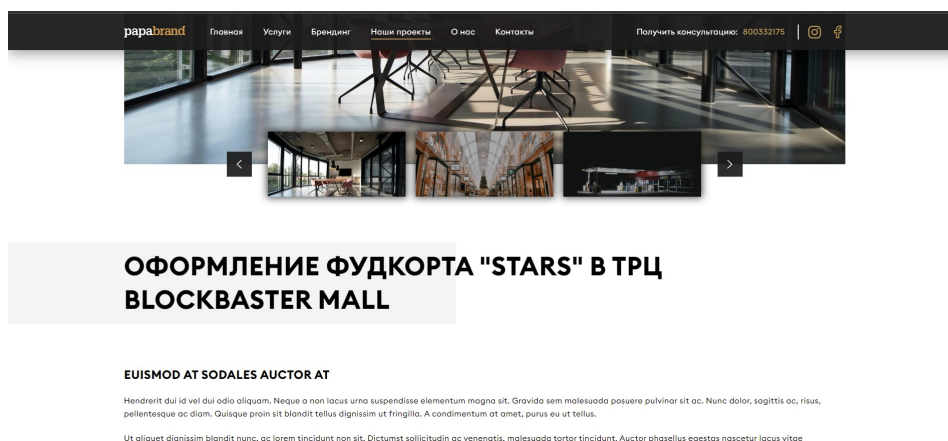


Рис.4.10. Проверка корректности работы страницы с конкретным проектом

### 4.3. Проверка корректности отображения логотипов с партнерами

Для того, чтобы проверить, каким образом отображается слайдер с логотипами партнеров на всех страницах, были загружены логотипы партнеров. Пример загрузки изображений в панели администрирования можно увидеть на рис.4.11.

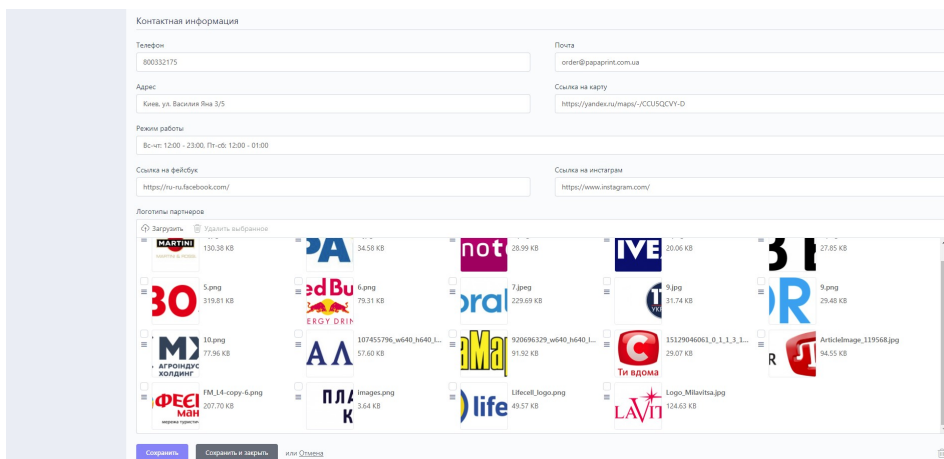


Рис.4.11. Пример загрузки изображений в панели администрирования

Проверка всех страниц показала, что все логотипы отображаются корректно, также имеется возможность поменять их расположение. Результат можно посмотреть на рис.4.12.

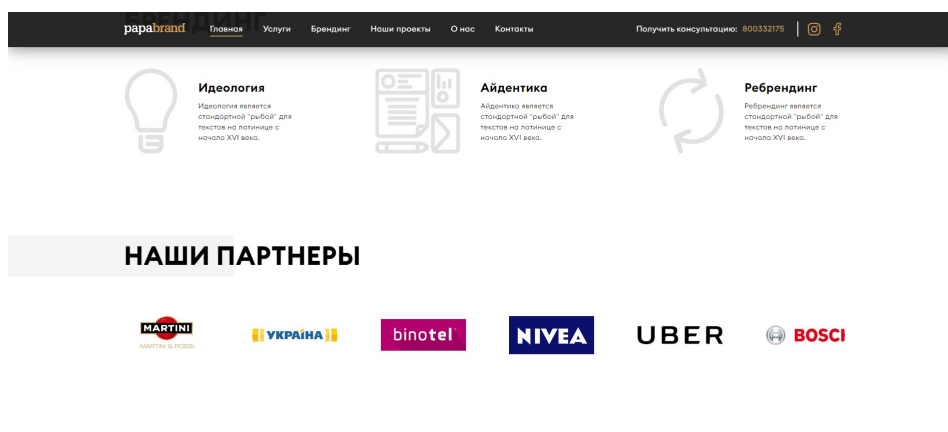


Рис.4.12. Проверка корректности отображения логотипов

### 4.4. Проверка работоспособности формы обратной связи

Для проверки работоспособности формы обратной связи была произведена отправка заявки с сайта на тестовый email - рис.4.13.

Необходимый javascript код сработал, следовательно плагин рассчитал нашу заявку успешно отправленной. Результат можно посмотреть на рис.4.14.

Рис.4.13. Пример заполнения формы обратной связи

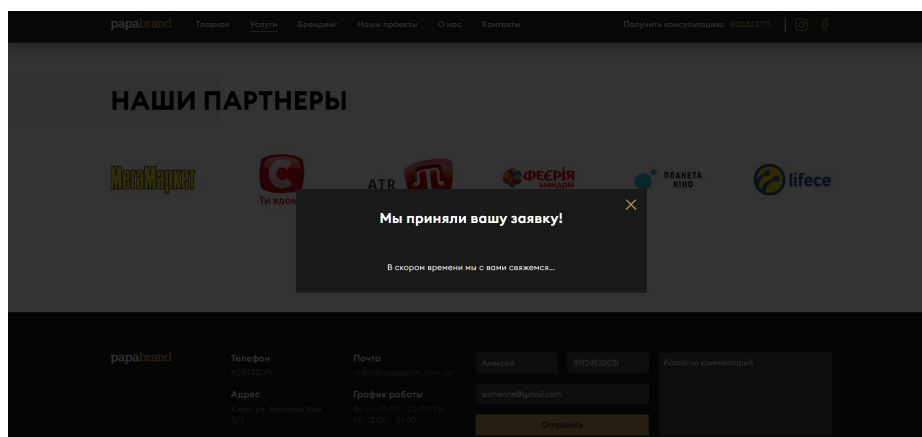


Рис.4.14. Проверка отображения модального окна при успешной отправке заявки

В панели администрирования появилась соответствующая запись - рис.4.15.

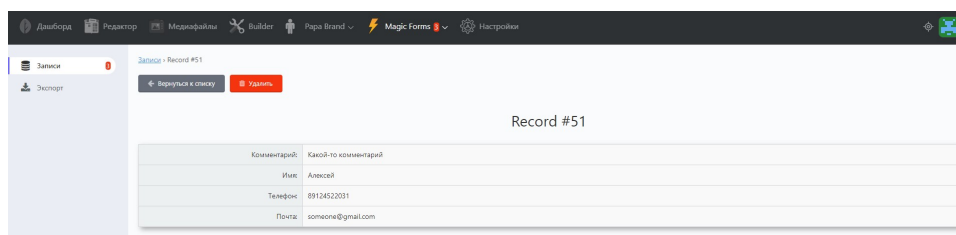


Рис.4.15. Проверка отображения заявки в панели администрирования

На почту заявка также пришла, поэтому считаем, что проверка корректности работы формы обратной связи была успешной. Результат также приведен на рис.4.16

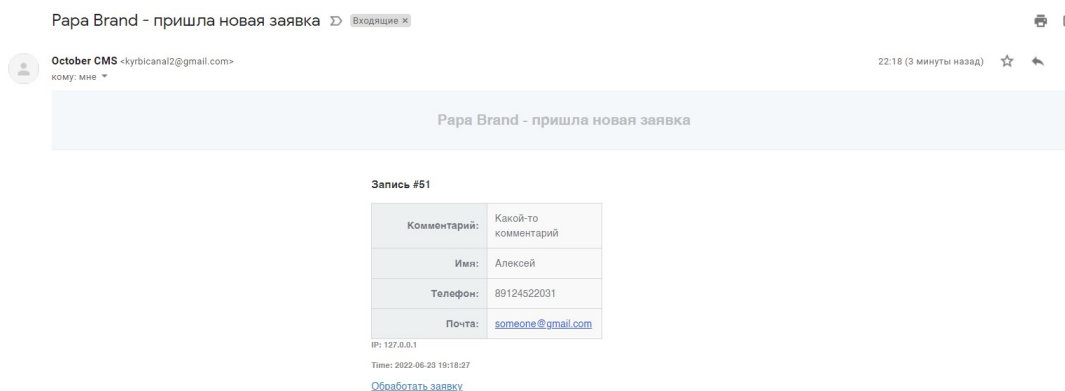


Рис.4.16. Проверка отображения заявки в почтовом ящике

#### **4.5. Выводы**

Таким образом, можно сказать, что все страницы отображаются корректно. На главной странице можно посмотреть на проекты, услуги и перейти на них. Также корректно отображаются логотипы партнеров на каждой странице. Правильно отображается контент на страницах с проектами и услугами. Что касается формы обратной связи - она также была протестирована: заявки хорошо просматриваются как в панели администрирования, так и на почтовом ящике.

## ЗАКЛЮЧЕНИЕ

Таким образом, в данной работе была изучена технология адаптации верстки к системе управления сайтом, была проанализирована актуальность систем управления контентом (CMS), выяснилось, что они являются достаточно востребованными на рынке, при осознанном использовании они помогут как разработчикам, так и заказчикам сэкономить время и деньги на разработку. Более того, если выбрать качественную CMS систему - помогут упростить и сделать разработку удобной и приятной.

Помимо всего прочего, был разобран принцип работы CMS. Обозначили главную цель систем управления - обеспечить разработчика возможностью достаточно быстро и эффективно создать необходимый интерфейс, для того, чтобы можно было манипулировать контентом сайта. Также были проанализированы типы CMS - их устройство, преимущества и недостатки. Разобрались с тем, когда стоит использовать Headless CMS, когда Decoupled CMS, а когда достаточно будет и традиционной CMS. Было отмечено, что преимущества одного типа CMS часто оборачиваются ограничениями другого типа CMS.

Выбор правильного типа зависит от потребностей бизнеса и требований компании. Headless CMS является оптимальным вариантом, если проекту потребуется гибкая интерфейсная архитектура, которую не предоставляет традиционная CMS. Headless CMS, ориентированная на API, позволяет эффективно продвигать свой контент по различным каналам, что будет важно при создании кросс-платформенного продукта. Однако, если задачей стоит создание автономного веб-сайта для бизнеса, лучше всего подойдет decoupled CMS - если не традиционная.

Помимо этого, в работе была представлена October CMS, которая может работать как и в обычном режиме, так и в режиме Headless. Также достаточно подробно была изучена архитектура October CMS, ее движок Laravel, разобрались, каким образом создаются плагины, макеты и фрагменты.

Далее был разобран пример работы с October CMS на примере реального проекта. После описания проекта, мы обозначили предметную область, описали каждую страницу. На основе этого далее был рассмотрен процесс создания темы, разбиения ее на необходимые составляющие для дальнейшего использования. Далее был разработан плагин со всеми зависимостями и соответствующим интерфейсом в панели администрирования, в котором можно добавлять, редактировать, удалять и изменять записи соответствующих моделей. Также был рассмотрен



процесс внедрения этих данных в верстку. Была настроена генерация страниц со всей необходимой информацией в них. Финальным этапом являлась настройка отправки почты с помощью плагина MagicForms, с помощью которого можно просматривать заявки не только на почтовом ящике, но и в панели администратора, что в дальнейшем заказчику поможет не потеряться в них.

В последней главе была проверена работоспособность всего сайта. Было проверено, корректно ли отображаются проекты и сервисы на главной странице. Правильно ли расположены логотипы партнеров на каждой странице. Корректно ли отображается контент на страницах с проектами и услугами. Заявки также хорошо просматривались как в панели администрирования, так и на почтовом ящике.

В заключении хотелось бы еще раз подчеркнуть важность осознанности выбора инструмента разработки, как для веб-сайта, так и для любого продукта. От правильности выбора необходимой среды зависит времязатратность и самое главное - качество продукта.



## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Staff B.* Статья компании Brightspot. — URL: <https://www.brightspot.com/learn/articles/decoupled-cms-and-headless-cms-platforms> (дата обращения: 23.06.2022).
2. *Wallis J.* Статья компании Webo Digital. — URL: <https://webo.digital/blog/headless-vs-decoupled-cms-platforms-whats-the-actual-difference/#topic-2> (дата обращения: 23.06.2022).
3. Документация October CMS. — URL: <https://docs.octobercms.com/2.x/setup/installation.html> (дата обращения: 23.06.2022).
4. Ресурс для ИТ-специалистов "Habr". — URL: <https://habr.com/ru/post/518160/> (дата обращения: 23.06.2022).
5. Ресурс для ИТ-специалистов "ProgrammersPub". — URL: <https://programmerspub.com/blog/octobercms/detailed-review-octobercms> (дата обращения: 23.06.2022).

## Код плагина

## Plugin.php:

```

1 <?php namespace Emil\Papabrand;
2
3 use System\Classes\PluginBase;
4
5 class Plugin extends PluginBase
6 {
7     public function registerComponents()
8     {
9     }
10
11     public function registerSettings()
12     {
13     }
14 }
15

```

## PapaBrand.php:

```

1 <?php namespace Emil\Papabrand\Controllers;
2
3 use Backend\Classes\Controller;
4
5 use BackendMenu;
6
7 class PapaBrand extends Controller
8 {
9     public $implement = [
10         'Backend\Behaviors\
11         ListController',      'Backend\Behaviors\
12         FormController',      'Backend\Behaviors\
13         ReorderController'    ];
14
15     public $listConfig = 'config_list.yaml';
16     public $formConfig = 'config_form.yaml';
17     public $reorderConfig = 'config_reorder.yaml';
18
19     public function __construct()
20     {
21         parent::__construct();
22         BackendMenu::setContext('Emil.Papabrand', 'main-menu-
23         item');
24     }
25 }
26

```

### Services.php:

```

<?php namespace Emil\Papabrand\Controllers;

use Backend\Classes\Controller;
5 use BackendMenu;

class Services extends Controller
{
    public $implement = [          'Backend\Behaviors\
        ListController',          'Backend\Behaviors\
        FormController'      ];

    public $listConfig = 'config_list.yaml';
    public $formConfig = 'config_form.yaml';

    public function __construct()
15 {
        parent::__construct();
        BackendMenu::setContext('Emil.Papabrand', 'main-menu-
            item', 'side-menu-item');
    }
}

```

### Projects.php:

```

<?php namespace Emil\Papabrand\Controllers;

use Backend\Classes\Controller;
5 use BackendMenu;

class Projects extends Controller
{
    public $implement = [          'Backend\Behaviors\
        ListController',          'Backend\Behaviors\
        FormController'      ];

    public $listConfig = 'config_list.yaml';
    public $formConfig = 'config_form.yaml';

    public function __construct()
15 {
        parent::__construct();
        BackendMenu::setContext('Emil.Papabrand', 'main-menu-
            item', 'side-menu-item4');
    }
}

```

## builder\_table\_create\_emil\_papabrand\_projects.php:

```

<?php namespace Emil\Papabrand\Updates;

use Schema;
5 use October\Rain\Database\Updates\Migration;

class BuilderTableCreateEmilPapabrandProjects extends
    Migration
{
    public function up()
    10 {
        Schema::create('emil_papabrand_projects', function(
            $table)
        {
            $table->engine = 'InnoDB';
            $table->increments('id')->unsigned();
            15 $table->timestamp('created_at')->nullable();
            $table->timestamp('updated_at')->nullable();
            $table->string('title')->nullable();
            $table->text('descr')->nullable();
            $table->text('services_belong')->nullable();
            20 });
        }

    public function down()
    {
        25 Schema::dropIfExists('emil_papabrand_projects');
    }
}

```

## builder\_table\_create\_emil\_papabrand\_services.php:

```

<?php namespace Emil\Papabrand\Updates;

use Schema;
5 use October\Rain\Database\Updates\Migration;

class BuilderTableCreateEmilPapabrandServices extends
    Migration
{
    public function up()
    10 {
        Schema::create('emil_papabrand_services', function(
            $table)
        {
            $table->engine = 'InnoDB';

```

```

15         $table->increments('id')->unsigned();
        $table->timestamp('created_at')->nullable();
        $table->timestamp('updated_at')->nullable();
        $table->string('title')->nullable();
        $table->text('short_descr')->nullable();
        $table->text('descr')->nullable();
20     });
    }

    public function down()
    {
25         Schema::dropIfExists('emil_papabrand_services');
    }
}

```

#### builder\_table\_update\_emil\_papabrand\_partners.php:

```

<?php namespace Emil\Papabrand\Updates;

use Schema;
5 use October\Rain\Database\Updates\Migration;

class BuilderTableUpdateEmilPapabrandPartners extends
    Migration
{
    public function up()
10    {
        Schema::table('emil_papabrand_partners', function(
            $table)
        {
            $table->string('subtitle_1')->nullable();
            $table->string('subtitle_2')->nullable();
            $table->string('subtitle_3')->nullable();
            $table->text('subtitle_1_descr')->nullable();
            $table->text('subtitle_2_descr')->nullable();
            $table->text('subtitle_3_descr')->nullable();
15        });
    }

    public function down()
    {
25        Schema::table('emil_papabrand_partners', function(
            $table)
        {
            $table->dropColumn('subtitle_1');
            $table->dropColumn('subtitle_2');

```

```
30 |         $table->dropColumn('subtitle_3');  
        $table->dropColumn('subtitle_1_descr');  
        $table->dropColumn('subtitle_2_descr');  
        $table->dropColumn('subtitle_3_descr');  
    });  
    }  
}
```

## Код страниц и макета

main-page.htm:

```

url = "/"
layout = "layout"
title = "Papa Brand"
5
[builderList ListPartners]
modelClass = "Emil\Papabrand\Models\Partners"
scope = "-"
scopeValue = "{{ :scope }}"
10 displayColumn = "id"
noRecordsMessage = "No records found"
detailsPage = "-"
detailsUrlParameter = "id"
pageNumber = "{{ :page }}"
15
[builderList projectsList]
modelClass = "Emil\Papabrand\Models\Projects"
scope = "-"
scopeValue = "{{ :scope }}"
20 displayColumn = "title"
noRecordsMessage = "No records found"
detailsPage = "certain-project"
detailsKeyColumn = "slug"
detailsUrlParameter = "slug"
25 pageNumber = "{{ :page }}"

[builderList services_list]
modelClass = "Emil\Papabrand\Models\Services"
scope = "-"
30 scopeValue = "{{ :scope }}"
displayColumn = "title"
noRecordsMessage = "No records found"
detailsPage = "certain-service"
detailsKeyColumn = "slug"
35 detailsUrlParameter = "slug"
pageNumber = "{{ :page }}"
==
{% set records = ListPartners.records %}
{% set displayColumn = ListPartners.displayColumn %}
40 {% set noRecordsMessage = ListPartners.noRecordsMessage %}

```

```

{% set projects = projectsList.records %}
{% set detailsKeyColumn = projectsList.detailsKeyColumn %}
{% set detailsPage = projectsList.detailsPage %}
45 {% set detailsUrlParameter = projectsList.detailsUrlParameter
    %}

{% set servs = services_list.records %}
{% set detailsPageServ = services_list.detailsPage %}
{% set detailsKeyColumnServ = services_list.detailsKeyColumn
    %}
50 {% set detailsUrlParameterServ = services_list.
    detailsUrlParameter %}

    <header>
        <div class="swiper-header-container">

55         <div class="swiper-header">
            <div class="swiper-wrapper">

                {% for proj in projects %}
                    <div class="swiper-slide" style="
                        background-image: url({{proj.
                            project_images[0].path}});">
60                     <div class="descr-outer-cotainer">
                        <div class="descr-container">
                            <h2>{{proj.title}}</h2>
                            <a href="{{ detailsPage|
                                page({ (
                                    detailsUrlParameter):
                                        attribute(proj,
                                            detailsKeyColumn) }}) }}">Подробнее</a>
65                             </div>
                        </div>
                    </div>
                {% endfor %}

            </div>

70         <div class="pagination-container">
            <div class="swiper-pagination"></div>
        </div>

75     </div>

```



```

        </div>
</header>
<main>

80
    <section class="services">
        <div class="container">
            <h1>Наши услуги</h1>
            <div class="swiper-services">
85
                <div class="swiper-wrapper services-item-
                    container swiper-services-const">

                    {% for i in 0..7 %}
                        <a href="{{ detailsPageServ|page({ (
                            detailsUrlParameterServ): attribute
                            (servs[i], detailsKeyColumnServ) })
                            }}" class="swiper-slide services-
                            item" style="background-image: url
                            ({{servs[i].service_images[0].path
                            }});" ><span>{{servs[i].title}}</
                            span></a>
                    {% endfor %}

                </div>
            </div>
        </section>

90
    <section class="branding" id="brand">
        <div class="container">
            <h1>Брендинг</h1>
            <div class="branding-item-container">
100
                <div class="branding-item">
                    
                    <div class="branding-item-text">
                        <h3>{{ records[0].subtitle_1 }}</
                        h3>
                        <span>{{ records[0].
                            subtitle_1_descr }}</span>
                    </div>
                </div>
            </div>
            <div class="branding-item">
105

```

```

110         
            <div class="branding-item-text">
                <h3>{{ records[0].subtitle_2 }}</
                    h3>
                <span>{{ records[0].
                    subtitle_2_descr }}</span>
            </div>
        </div>
        <div class="branding-item">
115         
            <div class="branding-item-text">
                <h3>{{ records[0].subtitle_3 }}</
                    h3>
                <span>{{ records[0].
                    subtitle_3_descr }}</span>
            </div>
        </div>
120     </div>
    </div>
</section>

```

certain-project.htm:

```

url = "/our-projects/:slug"
layout = "layout"
title = "certain-project"
5
[builderDetails projectsList]
modelClass = "Emil\Papabrand\Models\Projects"
identifierValue = "{{ :slug }}"
modelKeyColumn = "slug"
10 displayColumn = "slug"
notFoundMessage = "Record not found"
==
{% set record = projectsList.record %}
{% set displayColumn = projectsList.displayColumn %}
15 {% set notFoundMessage = projectsList.notFoundMessage %}

<section class="bread">
    <div class="container">
        <div class="breads-container">
20         <a href="/">Главная</a>

```

```

        <a href="/our-projects">Наши проекты</a>
        <a href="{{ record.slugg }}">{{ record.title }}</a
    >
    </div>
</div>
25 </section>

<section class="certain-service-slider">
    <div class="container">
        <div class="certain-service-slider-container">
30
            <div class="swiper-buttons-container">
                <div class="swiper-button-prev">
                    
                </div>
35
                <div class="swiper-button-next">
                    
                </div>
            </div>
        </div>
40
        <div class="swiper-wrapper">

            {% if record %}

                {% for proj_img in record.project_images
                    %}
45
                    <div class="swiper-slide">
                        
                        </div>
                    {% endfor %}

            {% else %}
50
                {{ notFoundMessage }}
            {% endif %}

        </div>
55
    </div>

    <div class="certain-service-slider-container-mini">
        <div class="swiper-wrapper">
60

```

```

        {% if record %}

            {% for proj_img in record.project_images
            %}

                <div class="swiper-slide">
                    
                </div>
            {% endfor %}

            {% else %}
                {{ notFoundMessage }}
70            {% endif %}

        </div>
    </div>
75 </div>
</section>

<section class="our-services">
    <div class="container">
        {% if record %}

            <h1>{{ record.title }}</h1>

            {% for art in record.descr %}
85             <article>

                {% if art.descr_subtitle%}
                    <h5>{{art.descr_subtitle}}</h5>
90                {% endif %}

                {% if art.text_before %}
                    {{art.text_before | raw}}
                {% endif %}

                {% if art.descr_image %}
95                 <a data-fslightbox="gallery" href="/
                    storage/app/media{{ art.descr_image }}"
                    >
                        
                    </a>
                {% endif %}
            {% endfor %}
        {% else %}
            {{ notFoundMessage }}
        {% endif %}
    </div>
</section>

```

```

100         {% endif %}

        {% if art.text_after %}
            {{art.text_after | raw}}
        {% endif %}

105     </article>
    {% endfor %}

    {% else %}
110        {{ notFoundMessage }}
    {% endif %}
</div>
</section>

```

certain-service.htm:

```

url = "/services/:slug"
layout = "layout"
title = "certain-service"
5
[builderDetails]
modelClass = "Emil\Papabrand\Models\Services"
identifierValue = "{{ :slug }}"
modelKeyColumn = "slug"
10 displayColumn = "title"
    notFoundMessage = "Record not found"
    ==
    {% set record = builderDetails.record %}
    {% set displayColumn = builderDetails.displayColumn %}
15 {% set notFoundMessage = builderDetails.notFoundMessage %}

<section class="bread">
    <div class="container">
        <div class="breads-container">
20            <a href="/">Главная</a>
            <a href="/services">Услуги</a>
            <a href="{{ record.slug }}">{{ record.title }}</a>
        </div>
    </div>
25 </section>

<section class="services-slider">
    <div class="container">
        <div class="servicer-slider-container">
30

```

```

35     <div class="swiper-buttons-container">
        <div class="swiper-button-prev">
            
        </div>
        <div class="swiper-button-next">
            
        </div>
    </div>

40    <div class="swiper-wrapper">

        {% for image in record.service_images %}
        <div class="swiper-slide">
            <div class="service-descr-container">
                <span>{{image.description}}</span>
            </div>
            
            </div>
            {% endfor %}
50    </div>
</div>
<div class="servicer-slider-container-mini">
    <div class="swiper-wrapper">
        {% for image in record.service_images %}
55    <div class="swiper-slide">
            
            </div>
            {% endfor %}
        </div>
60    </div>
</div>
</section>

<section class="our-services">
65    <div class="container">
        <h1>{{record.title}}</h1>

        {% for desc in record.descr %}

70    <article>

```

```

        {% if desc.descr_subtitle %}
            <h5>{{desc.descr_subtitle}}</h5>
        {% endif %}

        {% if desc.text_before %}
            {{desc.text_before | raw}}
        {% endif %}

        {% if desc.descr_img %}
            <a data-fslightbox="gallery" href="/
                storage/app/media{{ desc.descr_img }}">
                
            </a>
        {% endif %}

        {% if desc.text_after %}
            {{desc.text_after | raw}}
        {% endif %}
    </article>

    {% endfor %}

</div>
</section>

```

#### branding.php:

```

url = "/branding"
layout = "layout"
title = "branding"
5
[builderList brandingList]
modelClass = "Emil\Papabrand\Models\Branding"
scope = "-"
scopeValue = "{{ :scope }}"
10 displayColumn = "id"
noRecordsMessage = "No records found"
detailsPage = "-"
detailsUrlParameter = "id"
pageNumber = "{{ :page }}"
15 ==
{% set records = brandingList.records %}
{% set displayColumn = brandingList.displayColumn %}
{% set noRecordsMessage = brandingList.noRecordsMessage %}

```





```

        
        </a>
        <div class="text-container">
            {{dsc.text_right |raw}}
        </div>
    </div>
    {% else %}

        {% if dsc.image_in_article %}
        <a data-fslightbox="gallery1" href="/
            storage/app/media{{ dsc.
            image_in_article }}">
            

            </a>
        {% endif %}

    {% endif %}

    {% if dsc.text_after %}
        {{dsc.text_after |raw}}
    {% endif %}
</article>

{% endfor %}

</div>
</section>

```

#### layout.htm:

```

[builderList partnersList]
modelClass = "Emil\Papabrand\Models\Partners"
scope = "-"
5 scopeValue = "{{ :scope }}"
displayColumn = "id"
noRecordsMessage = "No records found"
detailsPage = "-"
detailsUrlParameter = "id"
10 pageNumber = "{{ :page }}"

[genericForm]

```

```

messages_success = "Ваша форма была успешно отправлена!"
messages_errors = "В вашей заявке содержатся ошибки."
15 mail_enabled = 1
mail_recipients[] = "kyrbicanal2@gmail.com"
mail_template = "martin.forms::mail.notification"
inline_errors = "disabled"
js_on_success = "openModal();"
20 sanitize_data = "disabled"
anonymize_ip = "disabled"
recaptcha_theme = "light"
recaptcha_type = "image"
recaptcha_size = "normal"
25 emails_date_format = "Y-m-d"
==
{% set records = partnersList.records %}

<!DOCTYPE html>
30 <html lang="ru">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial
        -scale=1.0">
35 <title>{{ this.page.title }}</title>

    <link rel="stylesheet" href="{{ '/assets/swiper.css' |
        theme }}" />
    <link rel="stylesheet" href="{{ '/assets/style.css' |theme
        }}">
    <link rel="stylesheet" href="{{ '/assets/media.css' |theme
        }}">
40
    {% styles %}

    <link rel="icon" type="image/x-icon" href="{{ '/assets/
        images/favicon.svg' |theme }}">
</head>
45 <body>

    <div class="request-container dn">
        <div class="request-inner-container">
            
50 <h2>Мы приняли вашу заявку!</h2>

```

```

        <span>В скором времени мы с вами свяжемся...</span>
    >
</div>
</div>
55 <div class="top-header-container-outer">
    <div class="container">
        <div class="top-header">
            <a href="/" class="logo"></a>
            >
            <nav class="header-nav">
160         <ul>
                <li{% if this.page.title == 'Papa
                    Brand' %} class="underlined"{%
                    endif %}><a href="/">Главная</a></li>
                <li{% if this.page.title == 'services'
                    or this.page.title == 'certain-
                    service' %} class="underlined"{%
                    endif %}><a href="/services">Услуги
                    </a></li>
                <li{% if this.page.title == 'branding'
                    %} class="underlined"{% endif %}><
                    a href="/branding">Брендинг</a></li>
                >
                <li{% if this.page.title == 'our-
                    projects' or this.page.title == '
                    certain-project' %} class="
                    underlined"{% endif %}><a href="/
                    our-projects">Наши проекты</a></li>
165 <li><a href="/#brand">О нас</a></li>
                <li><a href="#footer">Контакты</a></li>
                >
            </ul>
            </nav>
            <div class="schedule"><span>Получить консульта
                цию:</span><a href="tel:{{ records[0].
                    phone_number }}">{{ records[0].phone_number
                    }}</a></div>
170 
            <div class="top-header-right">

```

```

<a href="{{ records[0].insta_link }}"
  class="socials"></a>
<a href="{{ records[0].face_link }}" class
="socials"></a>
</div>
</div>
</div>
</div>
<main>
{% page %}

<section class="our-partners">
  <div class="container">
    <h1>Наши партнеры</h1>
    <div class="swiper-partners-container">
      <div class="swiper-partners">
        <div class="swiper-wrapper">
          {% for record in partnersList.records
            %}
            {% for image in record.
              partner_logos %}

              <div class="swiper-slide">
                
              </div>

            {% endfor %}
          {% endfor %}

        </div>
      </div>
    </div>
  </div>
</section>

</main>

<footer id="footer" >
  <div class="container">

```

```

110 <a href="/" class="logo"></a>
<div class="contacts-container">
    <div class="contacts-item">
        <h4>Телефон</h4>
        <a href="tel:{{ records[0].phone_number }}"
            >{{ records[0].phone_number }}</a>
    </div>
115 <div class="contacts-item">
        <h4>Почта</h4>
        <a href="mailto:{{ records[0].phone_number
            }}">{{ records[0].email }}</a>
    </div>
120 <div class="contacts-item">
        <h4>Адрес</h4>
        <a href="{{ records[0].address_link }}">{{
            records[0].address }}</a>
    </div>
    <div class="contacts-item">
        <h4>График работы</h4>
125 <span>{{ records[0].schedule }}</span>
    </div>
</div>

<form data-request="{{ genericForm }}"::
onFormSubmit">
130 {{ form_token() }}
    <div id="{{ genericForm }}_forms_flash" style=
        "display: none;"></div>
    <textarea id="comment" name="Комментарий"
        placeholder="Комментарий"></textarea>
    <div class="name-and-phone">
        <input id="name" name="Имя" type="text"
            placeholder="Имя" required>
135 <input id="phone" name="Телефон" type="
            text" placeholder="Телефон" required>
    </div>
    <input id="email" name="Почта" type="text"
        placeholder="someandrey@gmail.com">
    <button type="submit">Отправить</button>
    </form>
140 </div>
</footer>

<script

```

```

src="https://code.jquery.com/jquery-3.6.0.min.js"
145 integrity="sha256-/xUj+30JU5yExlq6GSYGSk7tPXikynS7ogEvDej/
    m4="
crossorigin="anonymous"></script>
    {% framework extras %}
    {% scripts %}
    <script src="{{ '/assets/libs/swiper.js' |theme }}"></
        script>
150
    <script>
    //Burger
        window.addEventListener('resize', function() {
155
            allHfirst.forEach(element => {
                element.style.setProperty("--width", `${(
                    window.innerWidth - containerTopHeader.
                    offsetWidth)/2 + element.offsetWidth}px`);
            });

            if (window.innerWidth > 1230 && burger.
                getAttribute('style')) {
160
                if (burger.getAttribute('src') == '{{ '/assets
                    /images/close.svg' |theme }}') {
                    topHeaderBurger();
                }
                burger.removeAttribute("style");
165 headerNav.removeAttribute("style");
                schedule.removeAttribute("style");
                schedule.classList.remove('dn');
                logo.removeAttribute("style");
                getConsultText.removeAttribute("style");
170
            }

        }, true);

        function topHeaderBurger() {
175
            if (burger.getAttribute('src') == '{{ '/assets/
                images/burger.svg' |theme }}') {

                containerTopHeader.style.height = "auto";
                burger.setAttribute('src', '{{ '/assets/images/
                    close.svg' |theme }}');
180
                burger.style.gridColumn = "4 / 5";

```

```

burger.style.gridRow = "1 / 2";
headerNav.style.display = "block";
headerNav.style.gridColumn = "1 / -1";
schedule.style.gridColumn = "1 / 5";
185 schedule.style.gridRow = "2 / 2";
schedule.style.margin = "30px 0 13px 0";
schedule.style.justifyContent = "start";
schedule.style.justifySelf = "start";
schedule.style.maxWidth = "none";
190 schedule.classList.add('no-before-after');
schedule.classList.add('db');
schedule.classList.remove('dn');
logo.style.marginRight = "0";
backgroundTop.style.height = "100vh";
195 getConsultText.style.display = "inline";
} else {
    setTimeout(() => {
        containerTopHeader.style.height = "100%";
    }, 200);

    burger.setAttribute('src', '{ ' /assets/images/
        burger.svg' |theme }}');
    burger.style.gridColumn = "auto";
    burger.style.gridRow = "auto";
    headerNav.style.display = "none";
    headerNav.style.gridColumn = "auto";
    schedule.style.gridColumn = "auto";
    schedule.style.gridRow = "auto";
    schedule.style.justifyContent = "end";
    schedule.style.justifySelf = "end";
    schedule.style.margin = "0";
    schedule.classList.remove('db');
    getConsultText.style.display = "none";

    if (window.innerWidth < 550) {
        schedule.style.display = "none"
        schedule.classList.add('dn');
    }

    logo.style.marginRight = "20px";
    backgroundTop.style.height = "73px";
220 }
}
</script>

```

```

225 <script src="{ { '/assets/script.js' |theme } }"></script>

    {% if this.page.title == 'Papa Brand' %}
        {% partial "main-partial" %}
    {% endif %}

230

    {% if this.page.title == 'certain-service' %}
        {% partial "certain-service-partial" %}
    {% endif %}

235

    {% if this.page.title == 'branding' %}
        <script src="{ { '/assets/libs/fslightbox.js' |theme } }"
            "></script>
    {% endif %}

    {% if this.page.title == 'our-projects' %}
240        {% partial "our-projects-partial" %}
    {% endif %}

    {% if this.page.title == 'certain-project' %}
        {% partial "certain-project-partial" %}
245    {% endif %}

</body>
</html>

```

#### services.htm:

```

url = "/services"
layout = "layout"
title = "services"
5
[builderList]
modelClass = "Emil\Papabrand\Models\Services"
scope = "-"
scopeValue = "{ { :scope } }"
10 displayColumn = "slug"
noRecordsMessage = "No records found"
detailsPage = "certain-service"
detailsKeyColumn = "slug"
detailsUrlParameter = "slug"
15 pageNumber = "{ { :page } }"

[builderList servicesPage]
modelClass = "Emil\Papabrand\Models\ServicesPage"
scope = "-"

```



```

20 scopeValue = "{{ :scope }}"
   displayColumn = "title"
   noRecordsMessage = "No records found"
   detailsPage = "-"
   detailsUrlParameter = "id"
25 pageNumber = "{{ :page }}"
   ==
   {% set records = builderList.records %}
   {% set displayColumn = builderList.displayColumn %}
   {% set noRecordsMessage = builderList.noRecordsMessage %}
30 {% set detailsPage = builderList.detailsPage %}
   {% set detailsKeyColumn = builderList.detailsKeyColumn %}
   {% set detailsUrlParameter = builderList.detailsUrlParameter
      %}

   {% set services = servicesPage.records %}
35
   <section class="bread">
       <div class="container">
           <div class="breads-container">
               <a href="/">Главная</a>
40               <a href="/services">Услуги</a>
           </div>
       </div>
   </section>

45 <section class="services-page">
   <div class="container">
       <h1>Наши услуги</h1>
       <div class="services-page-container">
           {% for record in records %}
50             {% if detailsPage %}
               <a href="{{ detailsPage|page({ (
                   detailsUrlParameter): attribute(record,
                   detailsKeyColumn) }) }}" class="
                   services-page-item" style="{% for img
                   in record.service_images %} {% if loop.
                   first %} background-image: url({{img.
                   path}});{% endif %}{% endfor %}" ><span
                   >{{ record.title }}</span></a>
               {% endif %}
           {% endfor %}
       </div>
55   </div>
   </section>

```

```

<section class="services-page-description">
  <div class="container">
    <article>
60      <h5>{{services[0].title}}</h5>
        {{services[0].descr | raw}}
    </article>
  </div>
</section>

```

our-projects.htm:

```

url = "/our-projects"
layout = "layout"
title = "our-projects"
5
[builderList projectsPageList]
modelClass = "Emil\Papabrand\Models\ProjectsPage"
scope = "-"
scopeValue = "{{ :scope }}"
10 displayColumn = "id"
    noRecordsMessage = "No records found"
    detailsPage = "-"
    detailsUrlParameter = "id"
    pageNumber = "{{ :page }}"
15
[builderList projectsList]
modelClass = "Emil\Papabrand\Models\Projects"
scope = "-"
scopeValue = "{{ :scope }}"
20 displayColumn = "slug"
    noRecordsMessage = "No records found"
    detailsPage = "certain-project"
    detailsKeyColumn = "slug"
    detailsUrlParameter = "slug"
25 pageNumber = "{{ :page }}"

[builderList servicesList]
modelClass = "Emil\Papabrand\Models\Services"
scope = "-"
30 scopeValue = "{{ :scope }}"
    displayColumn = "id"
    noRecordsMessage = "No records found"
    detailsPage = "-"
    detailsUrlParameter = "id"
35 pageNumber = "{{ :page }}"
==

```

```

{% set recordsPage = projectsPageList.records %}
{% set displayColumn = projectsPageList.displayColumn %}
{% set noRecordsMessage = projectsPageList.noRecordsMessage %}
40 {% set detailsPage = projectsPageList.detailsPage %}
    {% set detailsKeyColumn = projectsPageList.detailsKeyColumn %}
    {% set detailsUrlParameter = projectsPageList.
        detailsUrlParameter %}

{% set records = projectsList.records %}
45 {% set displayColumn = projectsList.displayColumn %}
    {% set noRecordsMessage = projectsList.noRecordsMessage %}
    {% set detailsPage = projectsList.detailsPage %}
    {% set detailsKeyColumn = projectsList.detailsKeyColumn %}
    {% set detailsUrlParameter = projectsList.detailsUrlParameter
        %}

50 {% set servs = servicesList.records %}

<section class="bread">
55     <div class="container">
        <div class="breads-container">
            <a href="/">Главная</a>
            <a href="/our-projects">Наши проекты</a>
        </div>
60     </div>
</section>

<section class="our-projects">
    <div class="container">
65         <h1>Наши проекты</h1>

        {% if recordsPage[0].show_categories %}

            <div class="buttons-swiper">
70                 <div class="swiper-pagination"></div>
                <div class="swiper-wrapper button-group
                    filters-button-group">
                    <button class="swiper-slide button is-
                        checked" data-filter="*">Все категории<
                        /button>

                    {% for serv in recordsPage[0].
                        services_to_show %}

```

```

75         <button class="swiper-slide button"
            data-filter=".{{ serv }}">{% for
            servv in servs %} {% if serv ==
            servv.slug %} {{ servv.title }} {%
            endif %}{% endfor %}</button>
            {% endfor %}
        </div>
    </div>
80     {% endif %}

</div>
<div class="our-projects-container grid">
    <div class="grid-sizer"></div>
85     <div class="gutter-sizer"></div>

    {% for proj in records %}
        <div class="project-item {% for serv in proj.
            services_belong %}{{serv}} {% endfor %}">
            <a class="project-item-img-container" href="{{
                detailsPage|page({ (detailsUrlParameter):
                attribute(proj, detailsKeyColumn) }) }}">
90                
            </a>
            <h5>{{proj.title}}</h5>
        </div>
    {% endfor %}
95
</div>
</section>

```