В этом документе я хочу собрать общие рекомендации к написанию кода во всех WEB-проектах компании.

Архитектура

Если мы пишем новые классы, функции или шаблоны - определите подходящее место в проекте, и следуйте принципам SOLID, KISS, DRY для ООП.

- **Шаблоны** - верхнеуровневые WP шаблоны должны находится в корне папки темы. Вложенные шаблоны или шаблоны подключаемые через плагины - в папке /templates/ и ее подпапках. Название подпапок и шаблона выбираем по сущностям, где:
  -родительская папка - объединяет их по отношению к определенному глобальному элементу сайта - **header**, **footer**, **отдельный шаблон страницы или custom post type**, **global** (другие sitewide элементы не входящие ни в одну из предыдущих категорий).
  -вложенная папка - объединяет шаблоны различных элементов вышеуказанного глобального шаблона.
  Примеры верхнеуровневых шаблонов:
  -page.php
  -header.php
  -footer.php
  -single.php
  -single-posttype.php
  -category.php
  -taxonomy.php.
  Подробнее в документации Wordpress.
  Примеры вложенных шаблонов:
  -/templates/header/mobile.php
  -/templates/header/desktop.php
  -/templates/page/liquidity-specifications/mainbanner.php
  -/templates/product/b2core/benefits.php
- **Классы** - в теме и плагинах мы складываем в /classes/ и ее подпапки, подключаем через spl_autoload проходом по папке. основной файл плагина также может являться классом.
  В теме functions.php подключается автозагрузка и создание статичных объектов из классов.
  В плагинах включенные плагины автоматически исполняют

одноименных файл php в корне, прямо под классом можно создать статичный объект.
В теме классы рекомендую создавать на сущности вордпресса - виджеты, типы постов, обработчики Ajax вызовов, волкеры Меню, Кэш,
В плагинах классы скорее должно относиться к сторонним интеграциям - youtube, модификации wpml, модификации acf, аналитика, црм.

- **Функции** - внутри классов можно называть без префикса, вне класса - обязательно добавляйте префикс, который не может случайно повториться при установке плагина или написанием функции другим разработчиком. Префиксом могут быть **b2broker_** или комбинация из нескольких префиксов **b2broker_wpml_,** в плагинах префиксом может быть название плагина (если вы назвали плагин wp - естественно не надо использовать префикс wp_ и по аналогии другие занятые префиксы - wpml_ acf_ и т.п.).

- **Форматирование** - используйте встроенные возможности IDE:
  -Indent - 4 Spaces
  -End of Line - LF
  -установите отдельные плагины для форматирования PHP, HTML, JS.
  -установите Autoprefixer или настройте в GULP

В корень проекта добавлен файл .editorconfig, который устанавливает базовые настройки вашего редактора. Для работы editorconfig необходимо установить плагин в редакторе. Некоторые IDE поддерживают работу без плагина. На сайте есть два раздела:

EditorConfig - перечислены редакторы которым плагин не нужен,
EditorConfig - перечислены редакторы которым плагин нужен и ссылка для установки (кликнуть по логотипу).
Синтаксис установки правил интуитивно понятен. В квадратных скобках указываются файлы, на которые действуют правила. Правила выглядят как ключ-значение. Более подробно настройки описаны в документации EditorConfig

```
[*] // для любых файлов
charset = utf-8
end_of_line = lf
```

```
4indent_style = space
5indent_size = 4
6trim_trailing_whitespace = true
7insert_final_newline = true
8
9[*.md] // для файлов с расширением .md
10trim_trailing_whitespace = false //переопределили правило
для .md
11
12[*.{json,yml,yaml}]
13indent_size = 2 // переопределили правило для файлов
json,yml,yaml
```

> Чтобы убедится в работоспособности плагина, можно в конце любого файла добавить несколько пустых строк, сохранить файл. Если после сохранения файла в конце останется одна пустая строка — плагин работает.

Настроить и описать работу с stylelint

Настроить и описать работу с eslint

- **Комментарии** - на данном этапе опциональны. Но желательно пояснять что передается в функцию и что возвращается в ответ. Также отдельные блоки внутри функции можно объяснить своими словами.

PHP

1. Конечно весь проект опирается на доступные методы для текущей версии языка, документацию можно посмотреть здесь:
   - PHP: Hypertext Preprocessor
2. Wordpress и плагины - методы вордпресса и используемых нами плагинов уже имеют множество функций, которые можно использовать.
   Попробуйте поискать, прежде чем изобретать велосипед.

- Reference | WordPress Developer Resources
- ACF | Resources, Documentation, API, How to & Tutorial Articles
- Документация - WPML
- и т.д.

3. Проекты переведенные на Bedrock также позволяют использовать Composer - используйте его, чтобы устанавливать нужные пакеты.

CSS / SCSS

1. **Архитектура.** По аналогии с архитектурой указанной для шаблонов, стили следует разделить по папкам и подпапкам - header, footer, global, page/page-type/page-element или product/product-type/product-element
   Во многих случаях даже не нужно создавать подпапки - разместите все стили для типа страницы или продукта в одном файле.
   Все стили находятся в папке /web/app/themes/……./frontend/src/css для темы, или в папке /css для плагинов. Для минификации и префиксов используется gulp, который кладет сбилженные версии в аналогичную папку без /src/ в пути.
   Рекомендую также почитать правила созданные для GULP касательно стилей, чтобы именно происходит или возможно предложить улучшения.
   Что положить в global?
   -подключение шрифтов
   -настройки .container
   -общие настройки для p,a,div,h1,h2,h3,h4,h5,h6,body,html ///TODO договориться об этом с QA и дизайнерами
   -формы
   -хеадер-футер
2. **Медиа запросы.** Для простоты поддерживания (KISS) и единообразия при тестировании - мы должны использовать всего 2 брейкпоинта - @media(max-width:992px) и @media(max-width:600px). Медиа запросы следует указывать внутри элемента, а не снаружи и тем более не в отдельном файле.
3. **Языковые версии** - прежде всего стараемся делать универсальную верстку, которая будет нормально смотреться без индивидуальных правок на другом языке. Обсуждайте это с дизайнерами. Если все же нужно делать альтернативный вариант (как например для RTL элементов) - добавляйте класс через php и закладывайте стили внутрь элемента, в котором описаны и обычные стили (они не должны быть снаружи элемента)

4. **Независимость** - не используйте классы которые могут случайно пересечься, используйте длинные префиксы по названию блока. например contactus_title, contactus_description, где в качестве префикса используется понятный референс на блок

JS / NODE.JS

1. **Node.js** используется только для менеджера пакетов и работы Gulp, поэтому все зависимости можно ложить в --save-dev.
   В целом в каждом проекте мы создаем несколько тасок - для images,scripts,styles и локальной разработки. От всех разработчиков хотелось бы - в свободное время читать о gulp, если необходимо вносить изменения в конфиг под измененные процессы, и в целом умение справляться с обновлением ноды и пакетов самостоятельно.
2. **Frontend JS** - по архитектуре следует ориентироваться на общие правила уже описанные выше - распределить весь код по папкам libs (полностью заимствованные библиотеки), global (скрипты которые используются на всех страницах сайта), page/page-type, product/product-type (для конкретных страниц), header, footer также отдельно.
   Если скрипт написанный для одной страницы, постепенно переходит в глобальный - используется на нескольких страницах или больше. Его можно обернуть в класс, дописать логику обрабатывающую фиксированные значения на динамически передаваемые переменные и положить в global.
   Мы используем jQuery и VanillaJs. Прошу не писать на React/Vue, т.к. это сузит круг разработчиков способных работать с проектом.
   Некоторые частые ошибки замеченные мной при мерж реквестах:
   - отсутствует проверка на null/undefined из-за чего падает весь остальной код
   - при подключении в wp_enqueue_script не указаны зависимости от других скриптов, из-за чего скрипт может загрузиться раньше чем зависимость
   - скрипт используемый на одной странице подключен глобально
   - логика для одной задачи расположена во множестве разных файлах

MYSQL и поля данных

Почти для всех запросов к БД есть готовые Wordpress функции, которые стоит посмотреть в доке. Но иногда приходится писать кастомные SQL запросы и передавать их через $wpdb. Поэтому вам пригодятся знания составления SQL запросов для текущей версии сервера баз данных. Также следует локально установить phpmyadmin, чтобы в визуальном интерфейсе полазать в структуре БД. Прочитать о структуре можно в официальной документации, но вот некоторые общие поинты о том где что хранится.

- **prefix_options** - таблица общих настроек сайта и глобальных полей используемых на всех страницах. в нее записываются все из Settings раздела админки, также плагины создают там свои строки с настройками плагина, также ACF поля ‘options’. У всех строк есть колонка autoload, которая определяет будет ли строка загружена при загрузке страницы.
  Слишком раздутый _options будет загружаться вместе с каждой страницей и замедлять страницу, старайтесь избегать сохранения данных в эту таблицу.
- **prefix_posts** - это таблица с индексом сущностей, в нее сохраняется минимальное количество данных о каждой сущности для более быстрой ее работы - в ней хранятся id, post_type, post_title, post_content и еще некоторые базовые поля сущностей.
- **prefix_postmeta** - это таблица в которой хранятся все дополнительные поля для каждой сущности - их могут быть тысячи. они привязаны к сущности по id. Движок вордпресса загружает всю постмету текущего поста в глобальную переменную $post при загрузке страницы. Когда вы выводите список постов циклом, он загружает всю постмету для каждого поста в цикле - поэтому все такие конструкции нужно оборачивать в объектный кэш, чтобы снизить нагрузку. больше про кэш ниже.
- **другие таблицы** - может быть множество таблиц которые создают плагины под свои нужды, читайте официальную документацию, и учитесь работать с ними, чтобы решать задачи по доработке плагинов.

Чтобы снизить объем БД имеет смысл не выводить все элементы для редактирования в админке. Закладывайте в шаблон все, что скорее всего будет редактироваться через разработчиков, а не контент-менеджеров.

Объектный кэш

Объектный кэш позволяет сохранять в памяти как данные из БД, так и полностью отрендеренные элементы с HTML, таким образом снижая нагрузку на сервер создаваемую PHP интерпретатором и запросами к БД.

У wordpress есть стандартные методы позволяющие работать с объектным кэшем - wp_cache_get, wp_cache_set, но чтобы ими пользоваться - необходимо установить Redis и подключить плагин Redis Object Cache - это уже сделано на некоторых прод сайтах на данный момент, также есть на тест сервере, и у меня на локальной машине.

Вам как разработчику, чтобы работать над задачами связанными с кэшем, потребуется установить их и локально.

Кэш может создавать аномалии, если вы не продумаете что и в каком кэше должно лежать.

Приведу пример:

1. На Prime of Prime Liquidity & Technology Provider - B2Broker вы создаете отдельный кэш для мобильной и десктопной шапки.
2. в зависимости от устройства пользователя тянется нужный кэш по ключу.
3. далее вы можете обнаружить, что определение текущего языка используется в формировании меню, и для всех языков закэшировался один и тот же язык.
4. На этом этапе вы добавляете к ключу кэша текущий язык. У вас получается уже не 2 раздельных кэша, а 2 * на количество языков.
5. При последующем тестировании однако вы обнаруживаете что хотя внутри одного языка шапка в целом одинаковая, но один элемент в шапке разный для каждой страницы - ссылки в языковом меню ведут всегда на разные страницы, поэтому их нельзя кэшировать.
6. В таком случае нужно рендерить языковое меню вне кэша, создавать для него отдельный кэш по другой логике и уже на фронтенде добавлять его скриптом в шапку.