Копируем библ.wp в корень домена (нашего проекта)

Далее необходимо создать БД wp1

Устанавливаем WordPress :  
<https://ru.wordpress.org/download/>

копируем содержимое в корень домена

создаем свою тему в проекте, в папке themes

создаем в в папке themes файл style.css

с содержимым:

@charset "UTF-8";  
/\*  
Theme Name: Тестовая тема  
Author: Smile  
Author URI: https://wordpress.org/  
Description: Описание тестовой темы  
Requires at least: WordPress 4.9.6  
\*/

создаем index.php для нашей автономной темы

в папку themes помещаем картинку png с именем screenshot (имя обязательно)

активируем тему (в начале так как index.php пустой мы увидим пустую страницу)

**Урок 4 - Импорт тестовых данных**

Заполним наш сайт тестовыми данными. Что бы не делать этого вручную, заходим на   
https://codex.wordpress.org/Theme\_Unit\_Test  
там качаем (либо с git либо прямо от туда <https://raw.githubusercontent.com/WPTRT/theme-unit-test/master/themeunittestdata.wordpress.xml>)

Далее: инструменты - импорт - (выбираем систему Wordpress) - установить - запустить импорт - выбираем скаченный файл : themeunittestdata.wordpress.xml - Загрузить и импортировать файл - ставим галку 'Скачать и импортировать файлы вложений' - Отправить (Ждем...)

**Урок 5 - Файлы шаблона header и footer**

Для дальнейшей работы имеет смысл использовать документация:  
<https://codex.wordpress.org/%D0%97%D0%B0%D0%B3%D0%BB%D0%B0%D0%B2%D0%BD%D0%B0%D1%8F_%D1%81%D1%82%D1%80%D0%B0%D0%BD%D0%B8%D1%86%D0%B0>

Начинаем создавать сайт:  
качаем bootstrap - в корне проекта создаем папку assets в ней папку boostrap и кладем папки css и js в папку bootstrap.

меняем источник стилей с CDN на локальный из папки assets

Делим на отдельные модули шапка - сайд бары - main - footer

создаем отдельные файлы в корне проекта - header.php ... (контент хранится в index.php) - footer.php . Название лучше оствлять стандрт. тогда некоторые функции wp будут видит эим файлы по умолч. вписываем в них соответств. названию логич.части страницы.

далее в index.php подключаем логич. части стр. при помощи get\_header() и get\_footer() - см.оф.док.

Если мы хотим создать доп. логич.части для стр. то их названия указваются с префиксом header - например:  
header-myNewHeader.php

**Урок 6 - Подключение стилей и скриптов.**

**Часть 1**

Файл функций (functions PHP) – Первое что подключается при вызове страницы, перед тем как отобразить весь контент страницы это functions.php

Не стоит помещать в этот файл какие-либо шаблоны view

! Вывод ошибок может быть отключен, настройки вывода см. в видео !

В файле functions.php мы можем описывать некую логику, например в wp принято подключать скрипты и стили при помощи спец. Функций

В wp не принято подключать стили и скрипты как это принято обычно, прямо в html коде страницы шаблона. Их необходимо подтягивать в момент определенного события в файле functions.php

Для этого использ. функции:

**wp\_enqueue\_style(); - функцию необходимо обернуть в свою польз. Функцию чтобы потом подцепить нашу функцию/обработчик на hook (повесить на событие), чтобы она была вызвана на определенное событие.**

\* регистрируем хук 'wp\_print\_styles'

add\_action('wp\_print\_styles', 'add\_my\_stylesheet');

\* Добавляем в очередь файл стилей, если он существует.

function add\_my\_stylesheet() {

$myStyleUrl = WP\_PLUGIN\_URL . '/myPlugin/style.css';

$myStyleFile = WP\_PLUGIN\_DIR . '/myPlugin/style.css';

if ( file\_exists($myStyleFile) ) {

wp\_register\_style('myStyleSheets', $myStyleUrl);

wp\_enqueue\_style( 'myStyleSheets');

}

}

В файле functions.php описываем нашу функц.

Так как вновь созданная нами функц. Могла быть ранее уже определенна, то в избежание конфликта рекомендуется именовать наши функции с префиксом(соответствует текущей теме), например test\_scripts(){.-.-.}

Далее при определении **wp\_enqueue\_style()** чтобы не писать длинный путь к файлу использ. функцию:

**get\_template\_directory\_uri**()- возвращает путь к папке с нашей темой

***~~wp~~***~~-~~***~~content~~***~~/~~***~~themes~~***~~/~~***~~test~~***~~/~~***assets***/***bootstrap***/***css***/***bootstrap***.***min***.***css***

get\_template\_directory\_uri() . **'/'** . ***'assets/bootstrap/css/bootstrap.min.css'***

**get\_stylesheet\_uri()** – ищет файл style.css в корне проекта

**wp\_enque\_script(); -** принцип похож, на wp\_enqueue\_style() – разве что имеет смысл передать аргумент TRUE в параметре $in\_footer, чтобы скрипты были загружены после основного html кода страницы.

Сейчас принято подключать скрипты перед закрывающим тегом body (например того требуют поисковые системы Google, для хорошей оценки нашего сайта).

**function** test\_scripts() {  
 *// Подключаем стили* wp\_enqueue\_style(**'test\_bootstrap'**, get\_template\_directory\_uri() . **'/'** .  
 **'assets/bootstrap/css/bootstrap.min.css'**);  
 wp\_enqueue\_style(**'test\_style'**, get\_stylesheet\_uri());  
  
 *// Подключаем скрипты* wp\_enque\_script(**'test\_popper'**, get\_template\_directory\_uri() . **'/'** . **'assets/bootstrap/js/popper.min.js'**);  
 wp\_enque\_script(**'test\_boostrap\_js'**, get\_template\_directory\_uri() . **'/'** . **'assets/bootstrap/js/bootstrap.min.js'**);  
}

Важно если **WP не подгружает style.css**, в этом случае необходимо:

Добавить в wp\_enqueue\_style параметр $ver. Можно даже в него подставлять дату изменения файла. Например:

**wp\_enqueue\_style**(**'test-style'**, get\_stylesheet\_uri(), **array**(), *filemtime*( get\_template\_directory()));

Особенность Chrome (в МСК надо проверить)

Источник: <https://ru.wordpress.org/support/topic/%D0%BD%D0%B5-%D0%BF%D0%BE%D0%B4%D0%BA%D0%BB%D1%8E%D1%87%D0%B0%D0%B5%D1%82%D1%81%D1%8F-style-css/>

**Урок 7 - Подключение стилей и скриптов. Часть 2**

После описания **function** test\_scripts() {} её необходимо повесить на hook/событие (повесить слушателя на событие)

**add\_action**(**'тип хука/тип события', 'имя функции обработчика'**);

Тип хука для скриптов - wp\_enqueue\_scripts

Тип хука для стилей - wp\_enqueue\_styles

Все хуки регистрируются при помощи **add\_action('тип хука/тип события', 'имя функции обработчика'**);

Для того чтобы окончательно подцепить наши скрипты и стили необходимы 2 важные функции:

**wp\_head();**

**wp\_footer();**

*Эти функции должны присутствовать в каждой теме !*

В header.php перед закр. Тегом </head> указываем **<?php** wp\_head() **?> и в footer.php в конце <?php** wp\_footer() **?>**

Эти вызовы/метки необходимы для различных плагинов wp

**Подключаем JQuery**

Если использовать wp\_enqueue\_script(**'jquery'**)- то подгрузится стандартная-старая версия JQuery, но так как нам нужна самая свежая версия JQuery, необходимо дерегистрировать старую стандартную версию jquery при помощи wp\_deregister\_script(), которая идет по умолч. в WP и после зарегистрировать новую версию jquery при помощи wp\_register\_script() и уже после вешать новую версию jquery на хук при помощи в wp\_enqueue\_script().

*// Подключаем Last.ver - JQuery*wp\_deregister\_script(**'jquery'**);  
wp\_register\_script(**'jquery'**, get\_template\_directory\_uri() . **'/assets/js/jquery-3.4.1.min.js', [], false, true**);  
~~wp\_enqueue\_script(~~**~~'jquery'~~**~~)~~;//Это можно вызвать как зависимость, см. Ниже

Вызов wp\_enqueue\_script(**'jquery'**) – можно убрать и подгрузить jquery как зависимость для poper.min.js, передав идентификатор jquery в 3й параметр wp\_enqueue\_script(**'test\_popper', '<PATH TO SCRIPT>', 'Dependencies'**) в виде элемента массива, там где подгружаем Poper.js

Т.е. если какой-то скрипт зависит от другого скрипта, не обязательно подключать каждый скрипт отдельно их можно подтянуть к проекту как зависимости для текущего скрипта.

Подключение скриптов в FOOTER:

Чтобы скрипт был подгружен в footer необходимо передать в параметре $in\_footer – TRUE. В параметре $ver можно указать false – тогда WP автоматом назначит версию.

~~По итогу: вешать скрипты напрямую в header или footer НЕ ПРАВИЛЬНО, правильно подключать с помощью хуков wp\_enequeue\_<. . .> на который вешаем соответствующие им функции/обработчики в которых использ~~

**Урок 8 - Цикл The Loop в WordPress**

Для вывода постов в WP есть такое понятие как the loop (или цикл WP):

**<?php if**(have\_posts()): **?>  
 <?php** $i = 1; **while**(have\_posts()): **?>  
 <?php** the\_post(); **?>** <**h3**>**<?=**$i;**?>**. **<?php** the\_title();**?>**</**h3**>  
 **<?php** $i++; **endwhile ?>  
<?php else**: **?>** <**p**>Постов нет ...</**p**>  
**<?php endif**; **?>**

**Пример из урока**

**<?php if**(have\_posts()): **?>** *<!--Проверяем есть ли на данной странице посты для вывода-->* **<?php** $i = 1; **while**(have\_posts()): **?>***<!--Пока есть посты, посты хранятся в массиве-->* **<?php** the\_post(); **?>***<!--Проверяет есть ли следующий пост и увелич. счетчик на 1, также помещает данные из текущего поста в спец. переменную-->* <**h3**>**<?=**$i;**?>**. **<?php** the\_title();**?>**</**h3**>*<!--Выводит заголовок поста-->* **<?php** $i++; **endwhile ?>***<!--св-->***<?php else**: **?>***<!--св-->* <**p**>Постов нет ...</**p**>  
**<?php endif**; **?>***<!--св-->*

Что бы постоянно не писать код цикла, в PhpStorm можно сделать заготовку ( это как трейты в PHP ) – это назыв. Live Template (Живой шаблон) позволяет сохранить кусок кода так называемый сниппет и вызывать его по некоторой аббревиатуре.

Что бы сохранить сниппет в PHPStorm: File – Settings – Editor – Live Templates – PHP

Далее справа + (add template) – даем название – вставляем кусок кода, apply.

После в коде набираем имя сниппета + Tab

Template: Sticky – Прилепленная запись - Эта запись выводится не зависимо от цикла, так как это закрепленный пост.

WP цикл по умолч. выводит 10 записей (см. настройки – чтение – На страницах блога отобр. …)

Циклом WP можно управлять, создавать собственные циклы и управлять там выборкой записей.

**Урок 9 -** [**Теги постов**](file:///D:\GEEK_BRAINS_TOSHIBA\PHP\WebForMySelf\%5bWebForMySelf%5d%20WordPress-%D0%9C%D0%B0%D1%81%D1%82%D0%B5%D1%80.%20%D0%A0%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B0%20%D1%82%D0%B5%D0%BC%20%D0%B4%D0%BB%D1%8F%20WordPress%20(2018)\videokurs\part1\part1_9.html)

Теги шаблона/функции (в этом уроке):

the\_title();

the\_content();

the\_excerpt();

the\_permalink();

В терминологии WP картинка в записи называется миниатюра.

*У функций с префиксом the\_....() есть функции близнецы с префиксом get\_the\_....(), отличие в том, что если функция с префиксом the\_....()выводит результат непосредственно в браузер, то get\_the\_....() возвращает значение которое можно сохранить в переменной.*

Функции *get\_the\_....() полезны* если например необходимо обработать title перед тем как вывести его в браузер:  
**<?php** $title = get\_the\_title(); **echo** обработка\_($title); **?>**

the\_content();

По задумке в записи, в списке записей, должен выводится не весь контент, а какая-то его часть, в терминологии WP отрывок (раньше называлось цитата). Иначе в списке карточек будет выведен контент из каждой карточки.

Чтобы этого избежать надо зайти в запись в админке, выбрать запись - открыть визуально и поставить тег-далее (разрыв строки) (если нет такого тега нажми + справа)

$more\_link\_text – Текст который будет показан вместо стд. текста “далее …”

the\_content($more\_link\_text, 2й можно опустить). Если передать пустую строку ‘’ в $more\_link\_text то данная ссылка будет опущена.

В админке настройки экрана

Еще один способ ограничить вывод контента записи использ. вместо the\_content()Функцию the\_excerpt() – выводит отрывок контента записи (по умолч. первые 55 слов), так же в админке в записи – отрывок можно указать конкретный отрывок. Помимо всего удаляет html теги и графику

Есть часть функций которые рекомендуется выводить только внутри цикла WP – см. оф. Док.

the\_permalink($post) – получает постоянную ссылку на запись, принимает аргумент $post - id поста, по умолч. id текущего поста в цикле.

Как правило принято делать ссылкой на полный текст статьи: заголовок и кнопку под отрывком.

По итогу: Берем статический шаблон блока и в нужных местах использ. функции шаблона в цикле WP, таким образом придавая динамику страничке.

**Урок 10 - Шаблон single**

Чтобы не мучиться с расстановкой тега <--MORE--> в каждой записи, самое простое и очевидное решение это использ. the\_excerpt()

Но есть побочный эффект, при переходе к полному тексту записи снова срабатывает the\_excerpt() и мы видим только первые 55 слов. Как этого избежать см. ниже:

Создаем новый файл single.php  
Теперь когда мы переходим к полному тексту записи, то использ. шаблон single.php, а не index.php

Шаблон single.php немного отличается от шаблона Index.php

<**div class = 'container'**>  
 <**div class = 'row'**>  
 **<?php while**(have\_posts()) : the\_post(); **?>** *<!-- post -->* <**div class="col-md-12"**>  
 <**div class="card"**">  
 <**img src="..." class="card-img-top" alt="..."**>  
 <**div class="card-body"**>  
 *<!-- Так как это единственная статья использ. h1-->* <**h1 class="card-title"**>**<?php** the\_title(); **?>**</**h1**>*<!-- Выводит в браузер-->  
 <!-- здесь вместо the\_excerpt() использ. the\_content()-->* <**p class="card-text"**> **<?php** the\_content(**''**); **?>**</**p**>  
 </**div**>  
 </**div**>  
 </**div**>  
 **<?php endwhile**; **?>**</**div**>  
</**div**>

Теперь когда мы переходим к полному тексту записи, то видим весь её контент.

Почему это все раньше работало на одном шаблоне Index.php без single.php см. в след. уроке.

**Урок 11 - Иерархия шаблонов**

<https://codex.wordpress.org/%D0%98%D0%B5%D1%80%D0%B0%D1%80%D1%85%D0%B8%D1%8F_%D1%88%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD%D0%BE%D0%B2>

Шаблон single.php использ. для показа отдельной статьи

В WP есть понятие ярлык(или слаг) в других CMS называется alias – на указывает на соответствующий шаблон.

Примеры

Если Ваш блог находится по адресу **http://domain.com/wp/** и посетитель переходит на страницу рубрики **http://domain.com/wp/category/your-cat/**, WordPress в папке текущей Темы в первую очередь ищет шаблон, который отвечает за вывод этой рубрики. Если ID данной рубрики 4, WordPress ищет шаблон с именем **category-4.php**. Если такой шаблон не найден, WordPress ищет общий для всех рубрик шаблон **category.php**. Если и такого файла-шаблона нет, WordPress ищет шаблон для вывода архива **archive.php**. Если нет и этого файла, WordPress будет использовать основной шаблон Темы **index.php**.

Если посетитель переходит на главную страницу сайта **http://domain.com/wp/**, WordPress ищет шаблон **home.php** и использует его для генерации запрошенной страницы. Если файла **home.php** в папке Темы нет, WordPress использует файл **index.php** для вывода главной страницы блога.

**Урок 12 - Работа с медиафайлами в WordPress. Часть 1**

Отключаем создание копий картинок в WordPress - <https://wp-kama.ru/id_4409/ubiraem-generatsiyu-kopiy-zagruzhaemyih-izobrazheniy-v-wordpress.html>

Медиа файлы лежат wp-content/uploads

По умолчанию WordPress создает четыре копии:

1. миниатюру (thumbnail)
2. средний размер (medium)
3. умеренный большой (medium\_large) с версии 4.4.
4. большой размер (large)

Чтобы изменить кол-во копий или размеры создаваемых копий необходимо зайти в админку далее Настройки – Медиафайлы.

Размер/вес файлов-копий с картинкой WP оптимизирует, поэтому они копии весят совсем не много.

Чтобы откл. Средний размер и Крупный размер необходимо выставить Ш х В значение 0.

Для Средний и Крупный размер масштаб рассчитывается из расчета 2Ш к 1В

умеренный большой (medium\_large) - откл. или настроить через админку нельзя, можно только откл. через программный код, вешая на хуки.

*## отключаем создание миниатюр файлов для указанных размеров*add\_filter( **'intermediate\_image\_sizes'**, **'delete\_intermediate\_image\_sizes'** );  
**function** delete\_intermediate\_image\_sizes( $sizes ){  
 *// размеры которые нужно удалить* **return** *array\_diff*( $sizes, **array**(  
 **'medium\_large'**,  
 **'large'**,  
 ) );  
}

Хук add\_filter под названием **'intermediate\_image\_sizes'**

ывмыв

**Урок 13 - Работа с медиафайлами в WordPress. Часть 2**

Крепим к посту миниатюру:

Данная опция вкл. программно в functions.php, иначе по умолч. она не отображается в админке.

*// Регистрирует поддержку новых возможностей темы в WordPress  
// (поддержка миниатюр, форматов записей и т.д.).*

*//* **'post-thumbnails'** *- для всех типов постов***function** test\_setup() {  
 add\_theme\_support(**'post-thumbnails'**);  
}  
  
*// Вызываем test\_setup() во время события after\_setup\_theme*add\_action( **'after\_setup\_theme'**, **'test\_setup'** );

Затем миниатура вставляется при помощи **<?php** the\_post\_thumbnail() **?>**

в нужный участок шаблона какой-либо страницы.  
По умолч. установ. ориг. изображение. тоже самое что и:  
the\_post\_thumbnail('full');

Зарезервированные названия размеров

thumb, thumbnail, medium, large, post-thumbnail

Названия "thumb" и "thumbnail" - это алиасы (синонимы) и относятся к одинаковым файлам.

Для записей без миниатуры можно установить миниатюру по умолч.

**<?php if**(has\_post\_thumbnail()): **?>  
 <?php** the\_post\_thumbnail() **?>  
<?php else**: **?>** <**img src="./wp-content/uploads/2019/07/Bear-1024x678.jpg"**>  
**<?php endif**; **?>**

**add\_image\_size()**- Регистрирует новый размер картинки (миниатюры).

Регистрация означает, что WordPress, при загрузке изображения, кроме базовых размеров будет создавать еще один файл - измененную копию оригинала с указанными размерами.

**function** test\_setup() {  
 add\_theme\_support(**'post-thumbnails'**);  
 add\_image\_size(**'my-thumb'**, 100, 100);  
}

**Урок 14 - Пагинация в WordPress**

Пагинация или постраничная навигация.

В веб-дизайне и дизайне программного обеспечения под пагинацией понимают постраничный вывод информации, то есть показ ограниченной части информации на одной (веб)-странице (например, 10 результатов поиска или 20 форумных тредов). Она повсеместно используется в веб-приложениях[1] для разбиения большого массива данных на странице и включает в себя навигационный блок для перехода на другие страницы — пагинатор.

ссылки для поста:

ЧПУ — это аббревиатура, расшифровывается как человеко понятные урл. Название говорит само за себя — это осмысленные URL адреса, которые может прочесть обычный человек.

Настройки -> постоянных ссылок

Добавим пагинацию на наш сайт.

the\_posts\_pagination( $args );

$args = array(

'show\_all' => false, // показаны все страницы участвующие в пагинации

'end\_size' => 1, // количество страниц на концах

'mid\_size' => 1, // количество страниц вокруг текущей

'prev\_next' => true, // выводить ли боковые ссылки "предыдущая/следующая страница".

'prev\_text' => \_\_('« Previous'),

'next\_text' => \_\_('Next »'),

'add\_args' => false, // Массив аргументов (переменных запроса), которые нужно добавить к ссылкам.

'add\_fragment' => '', // Текст который добавиться ко всем ссылкам.

'screen\_reader\_text' => \_\_( 'Posts navigation' ),

);

Настройка текста пагинатора в виде списка:

Обычно пагинацию делают в виде горизонт. списка.  
параметр type

#3 Удаляем заголовок H2 из шаблона:

<https://wp-kama.ru/function/the_posts_pagination>

**Урок 15 - Функция bloginfo - информация о сайте - работа с <title>странички</title>**

<https://wp-kama.ru/function/bloginfo>

Добавим меню

Выведим наименование нашего блога с ссылкой на главную страницу(home)

home\_url() - <https://wp-kama.ru/function/home_url> - Возвращает УРЛ главной страницы сайта (без / на конце). Учитывает протокол SSL.

**<?= 'My Title'?>**

тоже самое что и

**<?php echo 'My Title'**; **?>**

в WP принято использ. стиль кода из 2го примера.

Делаем динамичный <title> странички.

<**title**>**<?php echo** wp\_get\_document\_title(); **?>**</**title**>

**wp\_get\_document\_title()**- <https://wp-kama.ru/function/wp_get_document_title>

Если добавить теме поддержку ' [title-tag](https://wp-kama.ru/function/add_theme_support#title-tag)'

add\_theme\_support( **'title-tag'** );

то в этом случае в <head> части документа не нужно указывать тег <title>, он будет добавлен автоматически. Добавит его функция \_wp\_render\_title\_tag(), которая зацеплена на хук wp\_head, который ***вызывается функцией wp\_head()***, которая, в свою очередь, вызывается в <head> части темы.

**Урок 16 - Работа с меню в WordPress. Часть 1**

Стандарты кодирования в WP:  
<https://codex.wordpress.org/%D0%A1%D1%82%D0%B0%D0%BD%D0%B4%D0%B0%D1%80%D1%82%D1%8B_%D0%BA%D0%BE%D0%B4%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F_WordPress>

Меню в WP

register\_nav\_menus() - Это предпочтительнее, даже если планируется одно меню.

register\_nav\_menu() - регистрирует одно меню.

Сначала в админке см. слева Внешний вид, если в нем такого пункта нет, это говорит о том, что наша тема еще не поддерживает такую возможность.

Чтобы её вкл. нужно:

Зарегистрировать меню в functions.php

/\*Регистрация меню\*/  
register\_nav\_menus( **array**(  
 'header\_menu1' => 'Меню в шапке 1',  
 'footer\_menu2' => 'Меню в футере 2'  
) );

обычно поддержку меню вешают на хук' after\_setup\_theme'.

/\*  
 \* Вызываем test\_setup() во время события after\_setup\_theme  
 \* \*/  
add\_action( 'after\_setup\_theme', 'test\_setup' );

Далле приводим в соотетствие стили со стилями bootstrap,

~~в параметр container в wp\_nav\_menus() передаем аргумент 'container'=>'nav' в качестве тега обертки для меню.~~ в итоге ничего передават не надо, отсавляем значение по умолч.: 'div'

Урок 17 - Работа с меню в WordPress. Часть 2

Если нужно создать меню для нестандартной верстки, есть 2 варианта:

1. Так же как и в случае с пагинацией можно в style.css создать классы с соотетсвющими названиями классов для wp меню и скопировать правила стилей из bootastrap меню для классов wp меню.

2. Использ. параметр walker котором можно указать объект, который будет строить меню. В этом объекте можно описать HTML код получаемого меню, т.е. структуру меню.

в классе Test\_Menu extends Walker\_Nav\_Menu переопределяем метод public function start\_el(), для начала копируем содержимое метода из ядра WP (Что бы посмотреть API класса можно кликнуть по нему л.к.м. + ctrl)

Нам понадобятся некоторые из параметров объекта $item из функц. start\_el():

current - явл. текущ. элемент (нужна для выделения активного пункта меню)

Необход. переназначить классы вложженным элементам в wp меню в <ul>

в методе start\_el() класса Test\_Menu в конец массив $classes Добав. нужные нам классы.

/\*мои классы из BS\*/  
$classes[] = 'nav-item';

далее добавляем классы для <a> в разделе \* Filters a menu item's title

дописываем в строку с определнием атрибутов <a> наши классы:

$item\_output .= '<a' . $attributes . 'class="nav-link">';

Далее отмечаем активный пункт меню:

Для этого в массиве $classes в функц. start\_el() надо через if(по текущей стр.) добавить класс active для текущего <li>

По итогу, если нас не устраивает внеш.вид меню, мы можем добавить к классам, которые были сгенерированны WP для меню, классы стилей из Bootstrap например.

[start\_lvl( &$output, $depth = 0, $args = array() )](https://wp-kama.ru/function/Walker_Nav_Menu::start_lvl)

и  
[end\_lvl( &$output, $depth = 0, $args = array() )](https://wp-kama.ru/function/Walker_Nav_Menu::end_lvl)

Это теги <ul> и </ul> Нашего меню, $depth указывает уровень вложенности выбираемого меню или подменю (т.е. тега <ul>).

[start\_el( &$output, $item, $depth = 0, $args = array(), $id = 0 )](https://wp-kama.ru/function/Walker_Nav_Menu::start_el)  
и  
[end\_el( &$output, $item, $depth = 0, $args = array() )](https://wp-kama.ru/function/Walker_Nav_Menu::end_el)

**Урок 18 - Сайдбары и виджеты в WordPress. Часть 2**

создаем файл sidebar.php который будет содержат данную часть нашего сайта, т.е. это включаемая часть нашего шаблона страницы. В дальнейшем данный блок можно применять к различным страницам сайта (к single, page, category и т.д.)

создаем файл sidebar.php с шаблоном данной части страницы

<div class="col-md-3">  
 SIDEBAR  
</div>

функция регистрирующая sidebar вешается на хук 'widgets\_init'

после того как мы зарегили и вызвали sidebar в functions.php в меню становится доступным настройка виджетов во внешнем виде, там же мы увидим вновь созданный нами сайдбар, который мы можем запонить различным виджетами (в том числе встроенными).

Обратить внимание на не активные виджеты, туда можно поместить не использ.(которые подготовленны до поры до времени) виджеты без сброса их настроек.

Выводится сайдбар на экран при помощи dynamic\_sidebar( $index )

$index - ид-р ($id) зарегистрированного сайдбара в register\_sidebar()

Делаем чтобы если в сайдбаре нет виджетов, то контентная часть занимала все 12 колонок:  
is\_active\_sidebar() - проверит есть ли виджеты в данном сайдбаре, вернет true если в данной области есть хотя бы 1 виджет. Так же в index.php у контейнера надо указать: <!-- class="col" займет всю достпную ширину, достп. с BS 4-->

Далее вставляем данную часть шаблона страницы в нужное нам место на нужнюу страницу при помощи get\_sidebar();//см.названия параметра по умолч.

**Урок 20 - Кастомайзер тем. Часть 1**

Кастомайзер позволяет видит вносимые изменения в реальном времени, т.е. не надо заходит в админку вносить изменнения, потом сохранять и идти обратно на сайт.

Расширим наш кастомайзер:

добавим логотип для сайта, так чтобы его можно было корр. через кастомайзер:

вкл. поддержку лого в functions.php

/\*Вкл. настройку логотипа\*/  
add\_theme\_support('custom-logo', **array**(  
 'width' => '150',  
 'height' => '40'  
) );

После этого в кстмзре в \*св-ва сайта\* появ. опция Логотип, в нем загруж. картинку.

Для того чтобы лого вывелось на сайт надо внести правки в шаблон header.php:

<!-- Провереяем имееться ли Лого -->  
**<?php if** ( has\_custom\_logo() ) : the\_custom\_logo(); **?>  
 <?php else** : **?>** <a class="navbar-brand" href="**<?php echo** home\_url(); **?>**">**<?php** bloginfo('name'); **?>**</a>  
**<?php endif**; **?>**

has\_custom\_logo() -

the\_custom\_logo() -

***Кастомизируем Бэкгранд сайт***

В functions.php:

/\*Вкл. настройку background-color\*/  
add\_theme\_support('custom-background', **array**(  
 'default-color' => 'eeee22', // цвет по умолч. !цвет без знака #  
) );

После вкл. 'custom-background' в кастомайзере появл. возможность настройки background сайта, но на экране эти изменения не отображаются! В <head> генерируется код стиля, но <body> не назначается нужный класс кастомайзера, чтобы это исправить надо в необходимом месте вызвать функц. body\_class();  
. . .

<body **<?php** body\_class(); **?>** >

. . .

Добавление фонового изображения:

/\*Вкл. настройку background-color\*/  
add\_theme\_support('custom-background', **array**(  
 'default-color' => 'dbdbdb', // цвет по умолч. !цвет без знака #  
 'default-image' => get\_template\_directory\_uri() . '/assets/images/background.png' // Фоновое изображение по умолчанию  
) );

После этого в кастомайзере появ. опция ‘Фоновое изображение’, через админку это также будет доступно.

Если 'default-image' не отработает, то будет использ. 'default-color' .

Настройки фонового изображения, такие как position, repeat и т.д. делаются в кастомайзере.

**Урок 21. Кастомайзер тем. Часть 2**

Чтобы добавить фоновое изображение для шапки, нужно:  
/\*Вкл. background для шапки \*/  
add\_theme\_support('custom-header', **array**(  
 'default-image' => get\_template\_directory\_uri() . '/assets/images/coffee.jpg', // Фон.карт.по умолч.  
 'width' => '2000',  
 'height' => '1300'  
) );

Есть 2 вар. чтобы вывести background шапки на экран:  
1й – использ. функции header\_image(); Функц. отработает только если изображение установленно.  
get\_header\_image() – пытается проверить, а есть ли карт.для шапки.

2й – использ. get\_custom\_header() – возвращает объект с описанием картинки, нам нужно св-во url, так как хотим ипольз. 1 картинку по умолч.

В современных сайтах есть тренд выводить шапку (слайдеры) только на главной странице, на внутренних страницах Header не выводится (или выводится что-то другое) чтобы добиться этого в WP использ. след. функции:

is\_home(); - работает корректно только если в настройках глав. Стр. в кастомайзере выбран пункт ‘Ваши последние записи’ – см. оф. док.

is\_front\_page(); - отображается ли главная(домашняя) страница сайта(работает корректно в любом случае). Вернет true только если мы на главной странице.

Превью страниц должно показываться только в списке записей, если мы заходим через пункт меню в какую-либо запись, то контент должен отобразится полностью, для этого необходимо создать шаблон страницы page.php (использ. для показа отдельной страницы) он практически полностью дублирует single.php (для примера можно просто скопировать single.php) – *см. Урок 11. Иерархия шаблонов.*

**Урок 22 - Кастомайзер тем. Часть 3**

Посредством add\_theme\_support() мы подключали встроенный (предустановленные) возможности кастомайзера WP, но на самом деле мы можем добавлять собственные настройки в кастомайзер. Для полного понимания необходимо изучить оф.док. WP (по мнению автора самая оптим. статья на англ. яз. на оф. сайте):

При работе с кастомайзером мы будем оперировать след. Понятиями:

*Секция (Section):*



*Control (элемент управления):*



*Настройка/Св-во:*

Это то что находится в элементе управления, т.е. это значение которые мы можем вводить.



Если мы хотим добавить и использ. свои/пользовательские секцию(или control или настройку), её надо зарегить при помощи **my\_theme\_customize\_register()** и повесить на хук ‘customize\_register’,

// Пример из оф.док.  
function mytheme\_customize\_register( $wp\_customize ) {  
 //All our sections, settings, and controls will be added here  
}  
add\_action( 'customize\_register', 'mytheme\_customize\_register' );

Не обязательно создавать новую секцию для новой настройки, новую настройку (или изменения в уже сущ. настройке) можно добавить в уже сущ-щию секцию, если она подходит по смыслу для нашего функционала.

В качестве примера поработаем с цветом ссылок, дадим возможность пользователю их изменять через кастомайзер.

Так как это работа с цветом, логично использ. уже сущ-ию секцию ‘Цвета’ и в ней создать новый элемент управления для изменения цвета ссылок.

Некоторые методы объекта $wp\_customize:  
add\_setting() – позволяет добавить настройку.

add\_seсtion() – позволяет добавить новую секцию.

add\_control() – позволяет добавить элемент управления.

Для создания новой секцию (или контрола или настройки) надо использ. Методы объекта $wp\_customize:   
**$wp\_customize->add\_<section или control или setting>()**;  
затем данный объект передается в качестве параметра в my\_theme\_customize\_register();

Уже имеющиеся секции в WP:

* **title\_tagline** - Site Title & Tagline (and Site Icon in WP 4.3+)
* **colors** - Colors
* **header\_image** - Header Image
* **background\_image** - Background Image
* **nav** - Navigation
* **static\_front\_page** - Static Front Page

Пример $wp\_customize->add\_setting() из оф.док.

$wp\_customize->add\_setting( 'header\_textcolor' , array(  
 'default' => '#000000',  
 'transport' => 'refresh', // по умолч.'refresh'  
) );

**'transport'** Принимает 2 параметра => 'refresh' или 'post-message'

**'refresh'** – применяет изменения перезагрузкой фрейма ,работает с frame без JS.

**'postMessage'** – Работает асинхронно, применяет изменения через JS.

Все настройки кастомайзера хранятся в табл. wp-options в БД WP, получить доступ к этим св-вам/значениям можно при помощи:

**Get\_theme\_mods()** – список всех опций кастомайзера

**Get\_theme\_mod(‘название опции’)** – если надо получить конкретную опцию.

Приступим к написанию кода, обычно код для кастомайзера выносят в отдельный файл(по типу functions.php), но в учебных целях опишем его в functions.php

Поддержка настройки цвета ссылок через кастомайзер:

/\*  
 \* Customizer  
 \* Определим новую настройку и элемент управл. и добавим их в уже предустановленную в WP секцию 'colors'  
 \* Через данную функцию мы работаем с 3мя основными элементами кастомайзера секция - эл.управл. – настройка

\* Добавление и редактирование настроек осущ. посредством методов объекта  
\* $wp\_customize - объект класса WP\_Customize\_Manager

\* \*/  
**function** test\_customize\_register( $wp\_customize ) {  
 $wp\_customize->add\_setting( // добавляем в объект $wp\_customize настройку  
 'test\_link\_color',// Наименование настройки  
 **array**( // массив настроек  
 'default' => '#007bff', // цвет по умолч.  
 'sanitize\_callback' => 'sanitize\_hex\_color'// валидирует цвета - sanitize\_hex\_color - встроенная функц. ПРОЧИТАТЬ ПРО ЭТО В ОФ.ДОК. опционально  
 ) );  
 /\*  
 \* после добавления настройки 'test\_link\_color' она появится в Get\_theme\_mods() – список всех опций кастомайзера  
 \* или <?php //echo get\_theme\_mod('test\_link\_color'); // - отрывок из header.php ?>  
 \* \*/  
  
 /\* control(элемент управления) в который будет выведена настройка 'test\_link\_color' \*/  
 $wp\_customize->add\_control( // далее добавим элемент управления  
 **new** WP\_Customize\_Color\_Control( // объект этого класса использ. для настроек цвета  
 $wp\_customize, // объект класса WP\_Customize\_Manager  
 'test\_link\_color', // ID  
 **array**( // массив настроек  
 'label' => 'Цвет ссылок', // подпись данного поля  
 'section' => 'colors', // секция куда мы хотим добав. наш новый control; colors - предустановленная секция WP.  
 'setting' => 'test\_link\_color' // наименование настройки которая будет выведенна в данном control, определенна выше.  
 )  
 )  
 );  
}  
/\* вешаем 'test\_customize\_register' на хук 'customize\_register' \*/  
add\_action('customize\_register', 'test\_customize\_register');  
  
/\*  
 \* Для того чтобы вносимые изменения во вновь созданной настройке 'test\_link\_color'  
 \* применялись к нужному нам html элементу и отображались в браузере  
 \* необходимо еще описать функц. test\_customize\_css() в которой будут описаны стили css.  
 \* test\_customize\_css() необходимо повесить на хук 'wp\_head',  
 \* это нужно для того чтобы стили были выведенны в <header> и могли использоваться в <body>  
 \* \*/  
**function** test\_customize\_css() {  
 $test\_link\_color = get\_theme\_mod('test\_link\_color');  
 **echo** <<<HEREDOC  
<style type="text/css">  
a { color: $test\_link\_color; }  
</style>  
HEREDOC;  
}  
/\*  
 \* хук 'wp\_head' генерируется в том месте,  
 \* где стоит функц.метка wp\_head(); - см. в <header> в файле header.php  
 \* \*/  
add\_action('wp\_head', 'test\_customize\_css');

**Урок 23 - Кастомайзер тем. Часть 4**

В прошлом уроке при добавлении новой настройки add\_setting() мы использ. параметр 'transport' => 'refresh' , что влекло за собой перезагрузку всего фрейма, в этом уроке мы воспольз. параметром 'transport' => 'postMessage' что позволит нам воспольз. Live preview при внесении правок через кастомайзер, для этого требуется добавить параметр   
'transport' => 'postMessage'  
есть 2 варианта как добавить этот параметр:

*1й вариант, пример из оф.док:*

**$wp\_customize->get\_setting(‘назв. setting’)->transport = ‘postMessage’**

*2й вариант:*  
добавить transport = ‘postMessage’ сразу при инициализации настройки, в массиве настроек для новой настройки.

$wp\_customize->add\_setting( // добавляем в объект $wp\_customize настройку  
 'test\_link\_color',// Наименование настройки  
 **array**( // массив настроек  
 'default' => '#007bff', // цвет по умолч.  
 'sanitize\_callback' => 'sanitize\_hex\_color',// валидирует цвета - sanitize\_hex\_color - встроенная функц. ПРОЧИТАТЬ ПРО ЭТО В ОФ.ДОК. опционально  
 'transport' => 'postMessage' // асинхрон. перезагрузка части страницы  
) );

Чтобы вносимые изменения начали отображаться в браузере в реальном времени необходимо еще описать небольшой js скрипт в отдельном файле(<название\_темы>-customize.js) (пример можно взять в оф.док.):

Далее на хук ‘customize\_preview\_init’ необходимо повесить функц. в которой будет вызвана функц. wp\_enqueue\_script() которая подключает файл <название\_темы>-customize.js

/\* Подключение файла test-customize.js для асинхронной работы настроек в кастомайзере\*/  
**function** test\_customize\_js() {  
 wp\_enqueue\_script( // подключение js скрипта для асинхорон. работы кастомайзера  
 'test\_customizer', // ID  
 get\_template\_directory\_uri() . '/assets/js/test-customize.js', // путь к файлу  
 **array**( 'jquery', 'customize-preview' ), // массив зависимостей, которые будут полключ. перед загрузкой скрипта test-customize.js  
 '',  
 **true** // подключаем скрипт в футере  
 );  
}  
/\* вешаем на хук \*/  
add\_action('customize\_preview\_init', 'test\_customize\_js');

+++last string of conspect+++