# Требования к верстке 2025

## Цели

Данный регламент написан исходя из многолетней практики и нацелен на получение верстки, которую можно в дальнейшем легко поддерживать, добавлять новые блоки, элементы не ломая прежних, для легкого переиспользования блоков между проектами

## Основные требования кратко

**Браузеры**: последние 2 версии: Хром, Edge, Yandex, Opera, Safari, FF

**Поддержка технологий:** не менее 90% по CanIUse https://caniuse.com/ если меньше - уточнять

**PixelPerfect** - нужен, но не должно быть видимых отклонений по размерам

**Компонентный подход**

**Стек:** Сброс стилей Эрика Мейера,Bootstrap 5+ (только сетка, некоторые утилитарные класса, элементы форм), БЭМ, нативный JS

**Подход:** mobile-first

Подробнее будет ниже

## Порядок работ

Внимательно **изучить макет** (в режиме демонстрации). Сразу уточнить все спорные моменты. Учесть то, что в дизайне может быть многое не показано или не продумано.

PixelPerfect не нужен, лучше соблюдать общую стилистику сайта и баланс между разными блоками и элементами (напр. отступы и поля использовать bootstrap), при этом не должно быть явных, видимых различий с макетом

**Перед началом верстки составить представление о проекте**

* Найти закономерности и повторяющиеся паттерны, свойства. Найти похожие блоки и прикинуть как из одного состояния получить другое (горизонтальная и вертикальная новость). Придумать универсальные понятные названия блокам и элементам.
* Представить мобильную версию и промежуточные состояния, исходить из подхода mobile-first
* Продумать верстку так, чтобы было минимальное количество стилей и html тегов, вложенности
* Учитывать, что конечному продукту необходимо будет проходить PageSpeed Insight тесты. Значит исходить из минимализма, универсализма, оптимизации.
* Составить карту нужных плагинов/библиотек
* Прикинуть план-порядок действий

Затем приступить непосредственно к верстке в порядке: шапка, футер, простая текстовая страница, каталог объектов, главная. По ходу верстки пополнять по необходимости утилитарные классы и стили UI элементов.

Блоки верстаем как правило на flex. Помним про grid. Используем для отступа между элементами gap, затем margin. Отступы и поля, центровку обычно решаем при помощи утилитарных классов.

**Семантика**

Используем тэги под контекст. Верхняя/нижняя часть блока - header/footer. Списки - ul/ol. Тело статьи/новости - article и т.д. Сразу обращаем внимание где у нас реально h1, h2, h3, а где нужно использовать только стилизацию классами div.h1, div.h2, …

Для инпутов прописываем **inputmode**, чтобы корректно отображать на телефоне соответствующую клавиатуру

Если инпут - email, то указываем соответствующий тип

**БЭМ**

Работаем по БЭМ-методологии

https://ru.bem.info/methodology/quick-start/

Элемент внутри элемента не указываем, пример

.slider > .slider\_\_arrows > .slider\_\_arrow-left

.slider

.slider\_\_arrows

.slider\_\_arrow-left, но не .slider\_\_arrows\_\_arrow-left

Каждый блок страницы (который во всю ширину) - это секция. Секции должны свободно перемещаться со страницы на страницу

**Стиль именования классов должен быть постоянным по проекту**

**Стандарт** - block-name\_\_elem-name\_mod-name\_mod-val

**Не стесняемся использовать помощников/ИИ**

Не знаем или сомневаемся как пишется / переводится слово - google translate / ChatGPT / …

Надо макаронный JS код оформить в класс, почистить, переименовать, переоформить, сгруппировать, рефакторинг, best practice и т.п. - ChatGPT / …

## Общие требования, сдача проекта

* Стили, JS не сжимать
* JS библиотеки использовать последних версий и подключать как файлы, а не как ссылки на CDN
* В консоли не должно быть ошибок
* Тест скорости LightHouse должен быть максимально достижимым (стремимся к 100 на PageSpeed Insights, зеленая зона - минимум)
* Соблюдать единство стиля кода, именования по проекту, именования файлов
* Форматирование**.** Общая опрятность и аккуратность кода. Индентация табуляция размером 4. Не должно быть более одной пустой строки. Соблюдаем постоянный порядок атрибутов (id, class, type, name, value, data-, required).
* Называть все в коде целиком, не сокращать или сливать слова, без аббревиатур, транслитерации, пишем грамматически правильно, понятно по фактическому содержанию, правильно указываем число (форма - tour-search-**form**.js. формы - **forms**.js)
* Все блоки (html, стили, скрипты) должны быть отбиты комментариями начала и конца. Пример   
  /\*\* Block Name \*/   
  …  
  /\*\* Block Name END \*/  
  <!-- Block Name --!>  
  …  
  <!-- Block Name END --!>
* Всю верстку проверить на адаптируемость путем сжатия окна браузера, ничего не должно ломаться
* Проверить все блоки на переполнение текстом, длинные слова. Не должны выходить за пределы. Аналогично если есть - таблицы.
* В CSS стилях свойство height/width/flex-basis использовать только там, где это реально необходимо. Возможно лучшим свойством окажется min-height / min-width
* CSS размеры стараемся указывать в em/rem/%, в частности width, margin
* Не использовать глобальные стили типа h1, делать через классы .h1

## Общий подход

* Сомневаешься? Не знаешь? - уточни
* Лучше (пере)спросить, чем потом переделывать
* Начало что-то получаться? Покажи что получается и получи добро
* Типовые проблемы должны уже иметь типовые готовые решения. Что непонятно - лучше свое не изобретать, а спросить что использовать, возможно уже есть готовое решение

## Сетка

Использовать сетку Bootstrap (flex).

Делаем отдельный файл. По ходу верстки копируем себе только самое необходимое.

Если .col-12 нужен всегда, то .offset может не пригодиться вовсе или .mt-3 нужен, а вот .mt-lg-3 - нет.

Пример https://safanto.ru/template/css/bootstrap-grid.css

## Цветовое решение

Вынести все цвета в css переменные в common.css.

Название переменной желательно, где имеет смысл, не должно быть привязано к цвету. Представьте, что попросят сделать переключаемую или настраиваемую цветовую схему.

Пример: main, brand-01, primary, secondary, font, alternate-font, error, success

## UI элементы

Разбить макет на повторяющиеся компоненты, выделить UI элементы.

.form

.popup

.close-button

.btn .btn-primary

.input .input.error .input.success .radio

.tabs\_\_head .tabs\_\_tab .tabs\_\_content

.slider\_\_dots .slider\_\_dot .slider\_\_arrow

.label (у карточки ярлык “Хит”) .tag

.rating .rating\_\_star

.address

.avatar

Первым делом сделать на отдельной странице UI-kit (кнопки, элементы форм) по аналогии с Bootstrap. Показать все варианты кнопок во всех цветах и размерах, элементов форм, цветовое решение, систему теней, закруглений, типографика, система иконок. Так предотвратим “изобретение велосипедов" в дальнейшем.

Пример <https://svoy-hotel.ru/ui>

## Утилитарные классы

Стили с классами поместить в utility.css

Файл делим на две части: bootstrap и наши классы. bootstrap выделяем в блок

/\*\* bootstrap \*/

…

.text-center {

…

/\*\* bootstrap END \*/

.color-error {

Пример как из утилитарных классов мы должны иметь возможность изменять элемент можно посмотреть здесь <https://tailwindcss.com/>

Сделать утилитарные классы для всех случаев данного дизайна.

Верстка должна строится так, чтобы

а) программно (при отрисовке блока на php или действия скрипта js) можно было менять вид и свойства блока и элементов

б) в дальнейшем создавать новые блоки можно было используя существующие классы с минимальным обращением в файл стилей для добавления визуальных свойств

Условно утилитарные классы можно разделить на группы

1. Единичные классы. В стилях и наименованиях ориентируемся на bootstrap, доработать под проект. Обычно один класс - одно свойство

* Цвета - color-primary, color-success
* Фон - bg-primary, bg-success
* Фон с прозрачностью color-primary-08
* Текст - text-center
* Вид - d-none d-flex flex-wrap align-items-center justify-content-center gap-3
* Отступы и поля - mb-1 mx-1 p-2 py-3
* и т.п.

Хотим расстояние между инпутами внутри колонки: добавим их контейнеру .d-flex.gap-3

Нужно отцентрировать кнопку: оборачиваем ее в .d-flex.justify-content-center

1. Особые стили дизайна

Если есть какой-то уникальный стиль, который постоянно используется, то не нужно обязательно использовать класс, закрепленный в bootstrap.

Пример:

Если по всему сайту используется один border-radius: 6px, достаточно .border-radius

Если используется 3px и 10px .border-radius-small .border-radius-large

Если по стандарту 3px, но иногда 10px .border-radius .border-radius-large

1. Сгруппированные стили

Если в верстке постоянно встречаются элементы, где утилитарных классов 4, 5 и более или ряд свойств повторяется, выносим их в отдельный класс/элемент.

Это могут быть как более простые повторяющиеся паттерны дизайна, например

* граница основного цвета .border-primary
* плашки, баннеры и карточки со скруглением краев и наложение вторичного полупрозрачного цвета фона .bg-secondary-02-radius или если это частый основной паттерн даже просто .bg-item

Так и более сложные

Напр. часто нужно, чтобы картинка заполняла фон какого-то элемента. Создадим для нее класс

.img-cover {

object-fit: cover;

display: block;

width: 100%;

height: 100%;

position: absolute;

z-index: 1;

inset: 0;

}

вместо .position-absolute.z-index-1.d-block.inset-0 и т.д.

Или если в дизайне часто используется блок с вертикальной и горизонтальной центровкой элементов, то имеет смысл вместо .flex-wrap.align-items-center.justify-content-center.gap-3 создать .container-centered

## Компонентный подход

Нужно исходить из универсальности блока, а не его специфичности.

Если блоки между собой похожи, нужно их “группировать” в плане верстки, стилей, скриптов и создавать модуль, который может “мутировать”.

Например блоки: блог, новости, статьи, идеи, а **в некоторых случаях** и отели, города, страны, наша команда, наши награды - все будут по сути являться “каталогом”, “статьями раздела” или “блогом”. При этом между собой они могут отличаться только цветом, или ориентацией элементов карточки (фото, описание, кнопка) горизонтально или вертикально, или оформлением фона текста и т.д. Если каждый блок расписывать отдельно, получится много стилей дублировано, а блоки нельзя будет переключить или мутировать из одного вида в другой.

Поэтому **когда это имеет смысл**,

создаем общий класс для “элементов каталога” - .cards

добавляем ему его специфику - .cards.news

элемент - .cards\_\_item

добавим отличие по ориентации - .cards.cards-vertical / cards-horizontal

добавим отличие по цвету .cards.cards-primary

добавим отличие по цвету элементу - .cards\_\_item.cards-secondary

Это позволит одну и ту же верстку вывести как

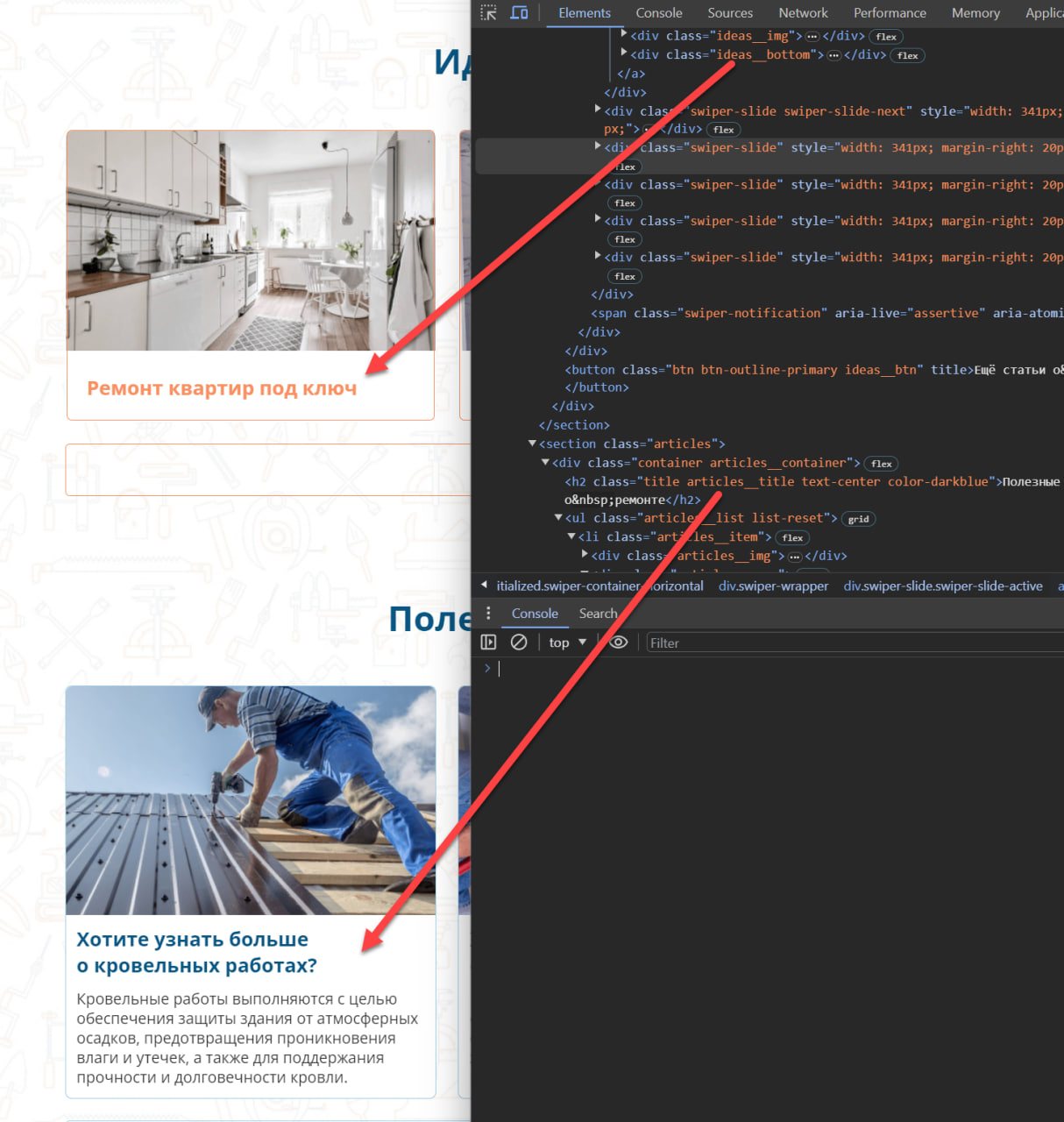
.cards.news.card-vertical.cards-primary

вертикальные синие карточки с особенностями новостей

так и

.cards.articles.card-horizontal.cards-secondary

горизонтальные оранжевые карточки с особенностями статей



*неправильный подход*

## Иконки

* Иконки сделать в SVG формате
* Собрать в икон-пак
* Иконки минифицировать, через SVGO (онлайн <https://jakearchibald.github.io/svgomg/>)
* Иконки должны быть монохромными, без прописанного цвета внутри иконки. Т.е. изначально она черная, цвет задается либо через собственный класс, либо через утилитарный класс (.color-primary)
* Все иконки должны иметь класс .icon в котором заданы по умолчанию размеры (наиболее часто встречающийся размер в проекте) и цвет (наиболее часто встречающийся размер в проекте или нейтральный, серый)
* Иконка должна наследовать цвет от родительского блока, а если это не подходит - добавляем собственный класс или класс-модификатор и в нем уже уточняем цвет
* Иконкам в явном виде прописать размеры
* Иконка, выполняющая роль кнопки должна быть именно кнопкой button > svg
* Пример вставки иконки  
  <svg class="**icon properties\_\_icon**" **width=”24” height=”24”**>  
   <use xlink:href="/template/image/icons.svg#icon-location"></use>  
  </svg>
* Иконки в иконпаке должны иметь id, начинающийся на icon-
* В проекте используется несколько иконок звезд (избранное, звезды отеля, рейтинг) По факту должна быть одна иконка звезды на все случаи (допустимо еще одну иконку - пустая звезда, только контур)
* В отзывах звезды можно реализовать простым способом: нижний слой - 5 звезд пустых, сверху слой 5 звезд закрашенных. Шириной верхнего слоя определяем рейтинг с любой точностью. Процент рейтинга задать в инлайновом css стиле в виде переменной (style=”--rate: 76”) а в стилях задать значение по умолчанию (width: var(--rate, 0)%)

## Картинки

* Разделить картинки, которые принадлежат шаблону и которые заполнены через админку
* Те, что шаблонные необходимо сжать в формат wepb (качество 90). Таких картинок меньшинство. Складывать в /template/img/
* Те, что в дальнейшем будут загружены через админку (почти все остальные) - в формате png. Складывать в /assets/images/.
* Лого - делать в формате png, должен быть картинкой, а не фоном,т.е. его можно скачать через меню правой кнопки
* Всем картинкам явно прописать высоту, ширину, исходя из размеров для десктопа.
* Всем картинкам прописать alt=””
* Всем картинкам, ниже первого экрана указать ленивую загрузку
* Не вставлять картинки фоном, если это не фон
* Не давать картинкам имена, которые вырежут баннерорезки! (banner и тп)
* Большим картинкам во всю ширину, например баннер на главной делать отдельные версии картинок для моб устройств см <https://www.visit-spa.com/>

Пример вставки

<img src="/assets/cache/images/foobar.webp" alt="Camelia Villas" class="card-object\_\_image" width=”200” height=”300” loading="lazy">

<picture>

<source media="(max-width: 320px)" srcset="assets/cache/images/de/kur-wellnes-reisen-320x750-308.webp">

<source media="(max-width: 576px)" srcset="assets/cache/images/de/kur-wellnes-reisen-576x750-aec.webp">

<source media="(max-width: 768px)" srcset="assets/cache/images/de/kur-wellnes-reisen-768x750-3c4.webp">

<img src="assets/cache/images/de/kur-wellnes-reisen-1903x750-e62.webp" alt="Reisen für Kur” title="Reisen für Kur &amp; Wellness zu Top-Konditionen" class="img-cover" width=”1903” height=’750’>

</picture>

## Ссылки

* Должны быть интерактивными: курсор - pointer, при наведении цвет меняется
* Ссылка должна иметь title=””
* Телефоны в шапке, футере должны быть в виде ссылок href=”tel:79….”

## Кнопки

* Каждая стандартная кнопка имеет класс .btn, в котором хранятся основные стили
* Цвета задаются по назначению кнопки (первичная, второстепенная и тп) .btn-primary .btn-secondary, где внутри класса прописывается цвет фона и шрифта если необходимо через переменные
* С обводкой btn-outline-primary, .btn-outline-secondary

В интерфейсах где нужно применяем user-select:none (.no-select), чтобы при двойном клике не выделялись слова

## Хлебные крошки

Взять верстку с <https://safanto.ru/penthausy> **Своего не изобретать!**

## Микроразметка

Для header и страницы контактов должна быть выполнена валидная микроразметка

## Адаптация под мобильные

* Делаем как минимум до 360px
* Верстка резиновая
* Для меню гамбургера использовать библиотеку <https://github.com/AndreyMyagkov/mobilePanel>
* Медиазапросы прописывать рядом с адаптируемым блоком, а не где-то отдельно
* Исходить из подхода mobile-first

## Типографика

Для простых текстовых страниц (типа о компании т.п ) использовать стили https://github.com/AndreyMyagkov/Typography

Есть тестовая страница с текстом рыбой и всевозможными элементами. Берем этот текст как есть

https://github.com/AndreyMyagkov/Typography/blob/master/index.html

https://github.com/AndreyMyagkov/Typography/blob/master/style.css, адаптируем под себя стили - цвета, размеры.

# Функционал

## Избранное

Взять механизм из <https://safanto.ru/> **ничего своего не изобретать**

<https://safanto.ru/template/js/favorite.class.js>

* На кнопках "в избранное" добавить рандомные id объектов data-id="270" и классы .js-to-favorite
* Дальше вешается скрипт https://safanto.ru/template/js/favorite.class.js который закидывает id в куки и меняет кол-во в избранном в шапке
* Для счетчика в шапке класс js-favorite-count
* Рекомендую звезду взять прямо с <https://safanto.ru/> Там она в единственном виде и может быть как пустой, так и залитой через свойства fill и stroke

## Переключение отображения вида карточек объектов

Горизонтальная/вертикальная карточка - взять механизм из <https://safanto.ru/> **Своего не изобретать!**

<https://safanto.ru/template/js/cardLayout.class.js>

## Формы. Отправка и валидация

Есть готовое решение. Код можно взять тут

<https://safanto.ru/template/js/formProcess.class.js>

Используем html валидацию полей. Соблюдаем типы полей.

Форма должна иметь скрытые поля для корректной работы, примеры на [https://safanto.ru](https://safanto.ru/template/js/formProcess.class.js)

## Уведомления о куках

Взять с <https://safanto.ru/>

# Прочее

## Библиотеки

**Пишем на чистом/vanilla JS. Подключение фреймворков по согласованию!**

* **Не использовать** jQuery
* **Vue** использовать только для тех блоков, которые не критичны для SEO: админка, калькуляторы, виджеты, корзина, визарды, сложные формы, конструкторы, т.е. там, где есть активное взаимодействие с пользователем, данные генерируются пользователем с нуля, либо на основе данных с сервера  
  Решение обосновано тем, что Гугл будет индексировать JS версию сайта только на свое усмотрение и это потребует дополнительный “поисковый бюджет”, а так же есть лимиты времени на генерацию контента, ориентировочно это 3 сек. Яндекс делает первые шаги в индексации JS контента и по умолчанию не индексирует. Индексация включается дополнительно в настройках веб-мастера (на 23-й год бета-версия)
* Типографика <https://github.com/AndreyMyagkov/Typography>
* Мобильное меню “Габургер” <https://github.com/AndreyMyagkov/mobilePanel>
* Галерея - Fancybox
* Слайдер - Swiper или Splide
* Датапикер - как вариант litepicker <https://litepicker.com/>
* Отправка форм - <https://safanto.ru/template/js/formProcess.class.js>
* Табы <https://safanto.ru/template/js/tabs.js>
* Вставка видео с Youtube https://github.com/paulirish/lite-youtube-embed
* Autocomplete - <https://tarekraafat.github.io/autoComplete.js/#/>
* Cookie - <https://github.com/js-cookie/js-cookie>

**Остальное по согласованию!**

## Файловая структура

index.html - главная

**На главной** добавить простейший текстовый блок, где списком **выведены ссылки на все сверстанные страницы**

* text.html - простая текстовая страница
* catalog.html - каталог объектов

/assets/images/ - демонстрационные фото в верстке (фото в новости), на рабочем сайте загружаемые через админку (потом заменяются реальными)

* /template/img - картинки шаблона (обычно это лого в шапке, футере, фоны страниц, блоков)
* /template/svg/ - svg
* /template/css/ - стили
* /template/js/animations.js - скрипты и модули
* /template/js/Menu.js - классы с большой буквы
* /template/lib/fancybox/ - библиотеки
* /template/block/search.css, search.js - файлы блоков, если много файлов и есть смысл - делаем папку /search - тут реши как делать

## Порядок подключения ассетов

Стили подключить в <head>. Каждый блок сайта - отдельным css файлом.

Порядок подключения

* reset.css - сброс стилей <https://meyerweb.com/eric/tools/css/reset/>
* bootstrap-grid.css - сетка (оставить только нужные стили!)
* ui.css - ui-kit (кнопки, формы и тп)
* utility.css - утилитарные стили блок bootstrap + собственные
* mobile-panel.css - мобильное меню гамбургер и другие библиотеки
* common.css - общие стили (шапка, футер, body, цвета),   
  проверить/добавить   
  \* {box-sizing: border-box}

Далее стили остальных блоков:

* search.css
* hotels.css

…

Скрипты подключать перед </body>

Порядок:

* Библиотеки - fancybox
* Классы - Tabs.js, ProductFilter.js
* main.js - основной скрипт, где запускаются общие необходимые скрипты
* Скрипты блоков - news.js, tour-search.js

Большие логические куски кода разделять в отдельные файлы модули. Код может быть в main.js, но если типового сгруппированного кода становится свыше 50-100 строк - выделить в отдельный файл и возможно класс

* menus.js
* animations.js
* banners.js
* sale.js

Ждем хорошего продукта, с которым будет приятно работать, легко читать и изменять, удобно использовать программистам

Есть вопросы? - Спроси :)