Необходимо использовать нашу сборку для верстки. Требования к верстке указаны ниже:

Требования к разметке (HTML)

Общие правила

`<meta charset="UTF-8">`

`<meta name="viewport" content="width=device-width, initial-scale=1.0">`

HTML-синтаксис <img> ~~<img/>~~

В атрибутах всегда используем двойные кавычки, а не одинарные.

Семантические блоки header, footer, article, aside.

Если не оговорено обратное, футер прибиваем к низу страницы.

Избегаем дублирующей разметки. Например, если это технически возможно, используем одно меню для мобильной и десктопной версии.

Проверяем собранные страницы в валидаторе. Если макет не проходит 100%-ную валидацию, это должно было оправданно.

Телефоны заключаем в тег `<a href="tel:">`, чтобы они были кликабельными на мобильных устройствах.

Все скрипты и стили подключаем непосредственно перед закрывающим тегом `body`. В блоке `head` допустимо прописать стили прелоадера, если он предусмотрен.

Самостоятельно или с помощью дизайнера готовим favicon и все, что с ней связано (webmanifest, browserconfig). Используем realfavicongenerator
.

Активные элементы, которые не предполагают перехода на другие страницы, не делаем ссылками `<a href="#">`, используем `span` или `button`
.

Никакого флеша.

Формы

Используем `input type="tel|email|url"` для подключения соответствующей клавиатуры на мобильных устройствах и проверки ввода средствами браузера.

Если не оговорено обратное, предусматриваем элементы для вывода ошибок валидации и стили для неправильно заполненных полей.

Проверяем tabindex.

Изображения

Используем ленивую загрузку для изображений (lazyload).

Шаблонные изображения (те, которые используются в дизайне и не будут меняться в дальнейшем) сжимаем вручную или с помощью npm-пакетов.

Для фоновых изображений используем дополнительные контейнеры с абсолютным позиционированием и image/picture внутри. Если не поддерживаем устаревшие браузеры, смело используем object-fit.

Не забываем про alt и title.

Используем современные форматы (webp) и адаптивные изображения там, где это возможно.

Пример со всем вышеперечиленным:

```
<picture>
    <source type="image/webp" media="(min-width: 768px)" data-srcset="image-lg.webp" srcset="image-preview.webp"
>
    <source type="image/webp" media="(max-width: 767px)" data-srcset="image-sm.webp" srcset="image-preview.webp"
>
    <source type="image/png" media="(min-width: 768px)" data-srcset="image-lg.png" srcset="image-preview.png">
    <source type="image/png" media="(max-width: 767px)" data-srcset="image-sm.png" srcset="image-preview.png">
    <img class="lazyload" data-src="image.png" src="image-preview.png" alt="alt text photo" title="alt text">
</picture>
```

Ссылки

Для внешних ссылок `target="_blank"` прописываем `rel="noopener"`.

Для ссылок, на которые не хотим передавать вес (соц. сети, месседжеры и пр.) прописываем `rel="nofollow noreferrer"`.

Шаблонизаторы

Используем шаблонизаторы для экономии времени и сил. Например, pug.

Дополнительно по согласованию с менеджером

- микроформаты
- accessibility
- работоспособность без js

Требования к стилям (CSS)

Общие правила

Согласовываем с менеджером, что мы хотим получить на выходе - один общий файл стилей или несколько для разных разделов сайта.

Фреймворки используем только при необходимости. Если используем, то последней стабильной версии. Не нужно для одностраничного сайта тянуть bootstrap. Если все же решили использовать, то подключаем только то, что действительно нужно.

Используем полный ресет стилей браузера (см. файл _custom-reset.scss в заготовке или из используемого css-фреймворка).

Для всех кнопок и инпутов, которые используются как кнопки, не забываем прописать `cursor: pointer`.

Верстаем mobile-first. Тема холиварная, но вроде как с точки зрения производительности так правильнее.

Не забываем отслеживать переходы между breakpoints. Т.е. если мобильное меню было открыто и пользователь увеличил окно, меню должно корректно закрыться, кнопки активации меню должны вернуться в исходное состояние.

Никогда не навешиваем стили на html-теги. Недопустимо: `ul li a span { color: black; }` Разработчику может понадобиться добавить еще один span внутри a.

В тоже время, для контентных блоков, которые будут заполняться клиентом через текстовый редактор системы управления контентом, стили навешиваем непосредственно на html-теги.

Не используем #id для стилизации.

Для именования классов используем методологию БЭМ с допущениями. Модификаторы для элемента пишем без перечня названий блоков-элементов, т.е. `product-list__item _popular`, а не `product-list__item product-list__item_popular`. Модификатор располагаем непосредственно после названия класса элемента, к которому он относится. Модификатор без этого класса никак не должен влиять на элемент.

Никакого транскрипта в названиях классов (`kartochka-tovara`), логически понятные английские названия (`product-card`).

Если используем background-image, то проверяем, умеет ли работать с такими изображениями используемый js-плагин «ленивой» загрузки. Если не работает, используем другие варианты. Помним, что изображения, жестко прописанные в стилях, сложнее заменить через систему управления контентом, не создаем дополнительных проблем разработчику.

Табличную верстку применяем только для таблиц.

Цвет фона для body всегда прописываем, даже если он белый.

Ресайз textarea (если он не отключен) не должен ломать вёрстку.

Проверяем блоки на «переполнение» текстом. Если в макете нарисована идеальная ситуция, когда все названия товаров занимают в листинге ровно две строки, то это не наш случай. В реальной жизни такого не бывает. Независимо от количества текста, карточки товаров должны быть одинаковой высоты. Все непонятные моменты согласовываем с дизайнером/менеджером.

Все анимации, какие только можно, реализуем через css3 transition.

Для всех ссылок/кнопок и других активных элементов предусматриваем состояния `:hover`, `:active` и `:focus`, а также реакцию на `:visited`. При необходимости требуем от дизайнера предоставить макет этих состояний.

Для иконок используем спрайты или иконочный шрифт.

Препроцессоры

Используем препроцессоры (scss).

Все переменные выносим в отдельный файл. Стараемся максимально использовать переменные для простоты изменения макетов.

Часто используемые конструкции (колонки, media-queries) выносим в mixins.

В отдельный файл выносим базовые стили для html, body и контейнера, если он есть. Сюда же пишем универсальные классы, которые могут пригодится разработчику/контентщику (hidden, nowrap, text-center и пр.). Предусматриваем классы для body, которые позволят зафиксировать скролл при открытии модальных элементов. Помним про эпл.

Стили для каждого логического блока описываем в отдельном файле и подключаем в конечный файл. Файлы называем по названию блока (`product-card.scss`), при необходимости пишем комментарии в начале файла и описываем, что это за блок и где используется.

Для вендорных префиксов используем gulp-autoprefixer. Список поддерживаемых браузеров согласовываем с менеджером и прописываем в `package.json`, используем browserslist.

Шрифты

Локальные шрифты складываем в fonts и подключаем через @import в файле fonts.scss. Внешние подключаем с помощью webfontloader. Подключаем только нужные семейства и стили шрифтов для минификации трафика. Если необходимый шрифт есть в свободном доступе на Google Fonts – используем Google Fonts.

Не забываем прописывать базовые font-family `'My Cool Font', sans-serif`. Проверяем верстку «на прочность» при отсутствии шрифтов.

Прописываем `font-display: swap` для `@font-face`.

Шрифты с иконками собираем с помощью fontello и складываем также в fonts (сохраняя всю структуру архива с демо и конфигом).

Названия шрифтов выносим в variables, в том числе и fontello.

Если шрифты платные, то не ищем их в свободном доступе, а запрашиваем у менеджера проекта.

Дополнительно по согласованию с менеджером

- Версия для печати

Требования к скриптам (JS)

Если в проекте интерактива будет больше, чем простое скрытие/отображение блоков – не боимся использовать jQuery.

Не используем cdn для загрузки библиотек.

Используем все современные возможности языка. Для обратной совместимости используем Babel.

Webpack для сборки билда.

Проверяем файлы линтером.

Не используем для выборки элементов классы, которые использованы для оформления и стилизации. Только собственные классы с префиксом `js-`.

Не засоряем глобальное пространство имен своими переменными.

Разбиваем код на логические блоки/функции/методы. Каждый метод выполняет определенный функционал.

Если метод инициализирует слайдер, код не должен вызывать ошибку, если слайдер отсутствует на странице.

Не изобретаем велосипеды. Используем проверенные решения и библиотеки.

Рекомендуемые библиотеки

- jQuery
- Попапы, полноэкранный слайдер Fancybox
- Маски ввода для инпутов Inputmask
- Слайдеры Swiper, Slick
- Кастомные селекты Select2
- LazyLoad