2.11. БЭМ — это про компоненты

~ 19 минут

Мы разобрались с правилами в методологии БЭМ, выяснили, как нужно именовать классы в проекте. Но БЭМ не совсем об этом. Главная его идея — это создание независимых блоков, независимых компонентов. Если вкратце, вам нужно измельчить макет до такой степени, чтобы у вас получились самые маленькие кусочки, какие только можно использовать отдельно без потери осмысленности, а затем нужно из этих кусочков собрать макет. Эти кусочки и есть ваши БЭМ-блоки, они же компоненты.

Если компоненты разработаны в отдельных файлах, их могут ещё называть модулями.

Модуль — изолированный и законченный фрагмент программы, имеет вид отдельного файла с исходным кодом, может применяться в других программах. Компонент, оформленный в отдельный файл, мы тоже можем называть модулем. У них много общего: изолированность, возможность повторного использования, возможность замены/изменения без связи с остальной системой.

Компоненты включают в себя всё необходимое, не зависят от внешних компонентов (блоков), могут разрабатываться отдельно и подключаться в разные части сайта, переиспользоваться. В идеале, компоненты можно использовать в разных проектах. Но на практике это встречается очень редко, разве что в коробочных решениях.

> Коробочные решения

Итак, что такое компонент? В чём идея «компонентного мышления»? Есть ли сложности с тем, чтобы сформировать и развить у себя «компонентное мышление»? Как эти сложности решаются? Как выработать навык создания компонентов?

Часто у начинающих разработчиков возникает желание сразу верстать сайты целиком, от начала до конца. Сам *процесс разработки* этому способствует. Сначала вы всё размечаете, затем накидываете общие стили (устанавливаете параметры шрифта, задаёте цвет фоновой заливки), строите сетки и так далее. Это напоминает принципы работы прогрессивного *JPG*, когда сначала делаются наброски, а затем шаг за шагом прорабатываются детали. Но при таком подходе сложно правильно вычленить компоненты. Велика вероятность, что компоненты будут сильно связаны между собой и «размазаны» по макету, не будет ясно, где заканчивается один компонент и начинается другой, или компоненты будут слишком большие, включающие в себя несвязанные части, или, наоборот, будут слишком маленькими и использоваться только в своём окружении.

Усугубляют ситуацию нативные технологии («чистые» HTML и CSS, без надстроек, шаблонизаторов, библиотек, фреймворков, препроцессоров и так далее). Сложно разрабатывать каждый компонент отдельно. Если стили мы ещё можем разделить по блокам и собрать их с помощью правила `@import`, то с разметкой это не получится сделать. Вся разметка находится в одном файле.

Мы постарались собрать небольшую инструкцию, которую можно использовать как руководство по делению макета на компоненты. Это не набор жёстких правил, для которых существует только так, а не иначе, это скорее рекомендации. Некоторые утверждения удостаивались длительных холиваров.

Перед тем как сформулировать правила, сделаем небольшое отступление для пары определений.

Композиция — это процесс сборки проекта из отдельных компонентов. Компоненты собираются вместе, вкладываются один в другой.

Это похоже на такой пазл для детей, в котором нужно вкладывать отдельные кирпичики и кубики в формочки большего размера, а потом собирать уже их в ещё большую форму.

Пазл

Декомпозиция — это обратный процесс, деление макета на множество кусочков, каждый такой кусочек макета это компонент или БЭМ-блок.

Наша версия правил создания хорошей архитектуры по БЭМ

Правило 1

Лучше перемельчить, чем недомельчить. Когда перестать мельчить, зависит от проекта.

У БЭМ нет точных указаний, что будет блоком, а что элементом, это зависит от проекта. Вопрос, который поможет нам отличить одно от другого: *может ли элемент существовать самостоятельно, вне своего родителя?* Да — блок, нет (или нет в рамках этого сайта) — элемент.

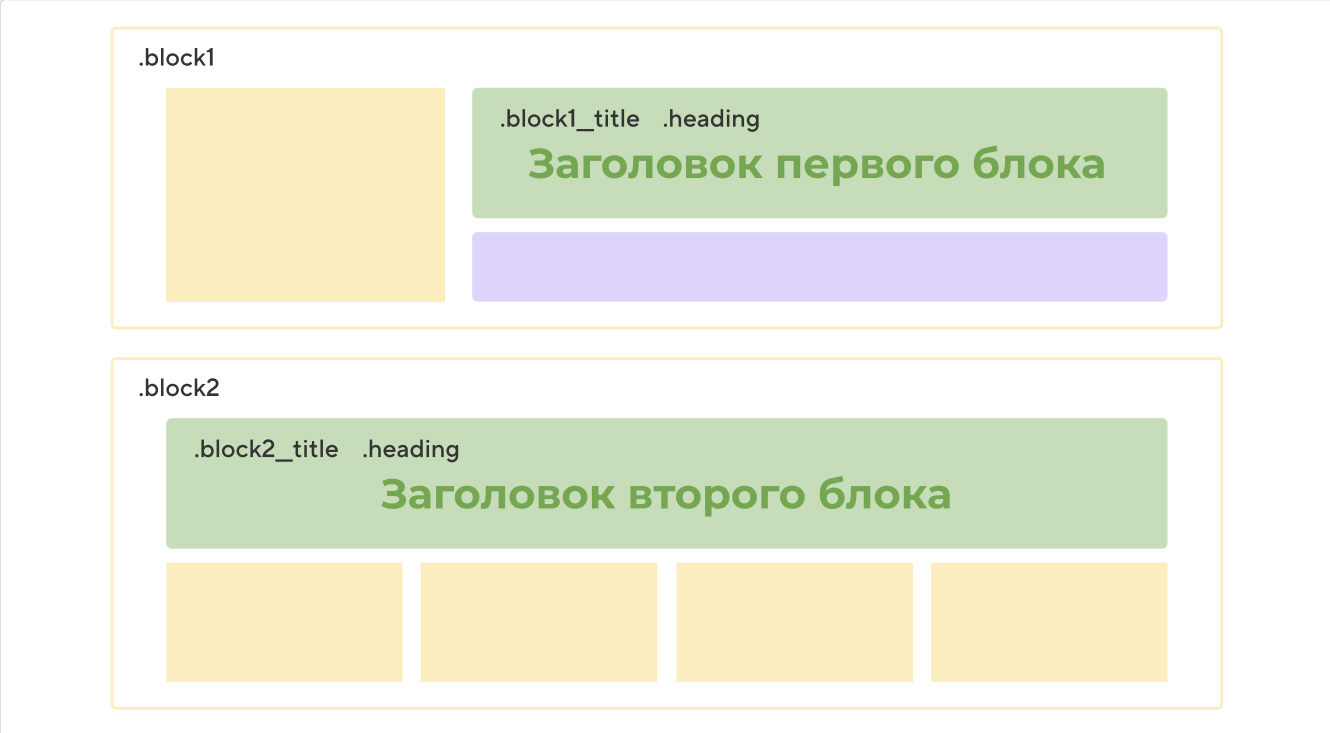
Но всегда ли всё так однозначно?

Вот такой пример попался нам в документации:

```
<button class="button">
  <span class="button__inner">
    <span class="icon"></span>
  </span>
</button>
```

Казалось бы, иконка отдельно от кнопки не существует. Но если мы уверены, что у нас по одной системе (положение, размер и так далее) располагаются иконки во всех кнопках, то почему бы даже этой иконке не быть блоком.

Есть и другой пример. Есть небольшой лендинг, с двумя-тремя разделами, в каждом из которых есть заголовки, которые выглядят почти одинаково, различаться могут цвет шрифта, выравнивание. Стоит ли такой заголовок выносить в отдельный БЭМ-блок? Может заголовок существовать отдельно от раздела? Вроде бы, нет. А если проект большой и у нас есть для заголовков свои стили, которые определяются в дизайн-системе? При изменении стиля заголовка в дизайн-системе нужно будет зайти в огромное количество модулей и поменять свойства. А если что-то пропустили? В этом случае лучше вынести заголовок в отдельный блок.



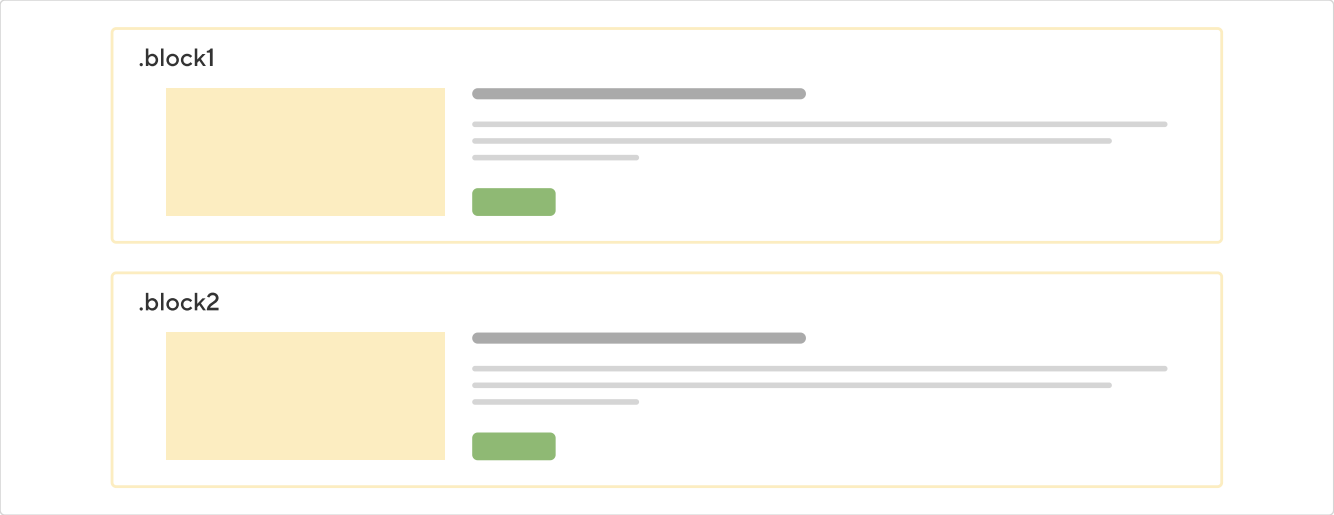
Лендинг с типовыми блоками

Поэтому говорят: «Лучше перемельчить, чем недомельчить».

Правило 2

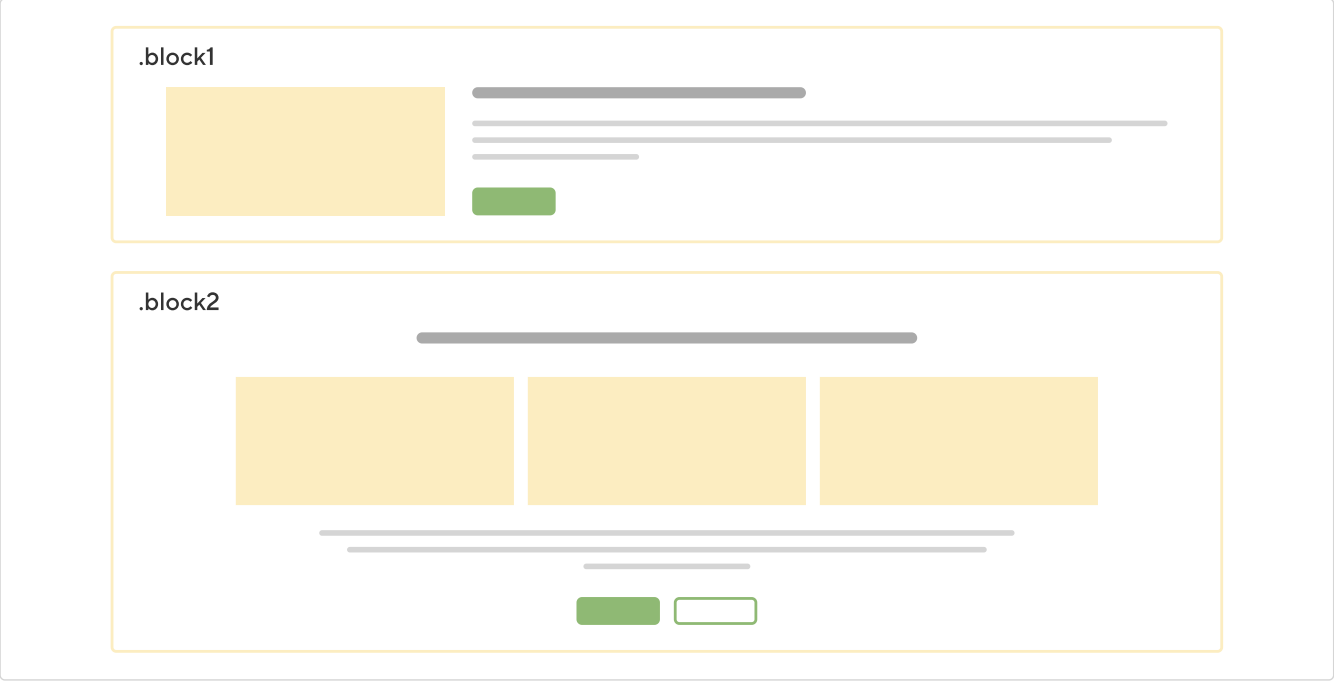
Сначала лучше, чтобы все элементы дизайна были отдельными блоками, а только потом выделяем общие части и переиспользуем их. Уровень абстракции зависит от проекта.

Все мы знаем о принципах *KISS (Keep It Simple, Stupid)*, *DRY (Don't Repeat Yourself)* и нам хочется их применить. Часто находим общее, там где его на самом деле нет. К примеру, у нас есть всё тот же небольшой лендинг. Большинство разделов выглядят одина различается только их содержимое.



Лендинг с типовыми блоками

Теоретически, каждый из этих блоков при развитии проекта может начать жить своей отдельной жизнью, и изменения в одном не должны повлиять на другой.



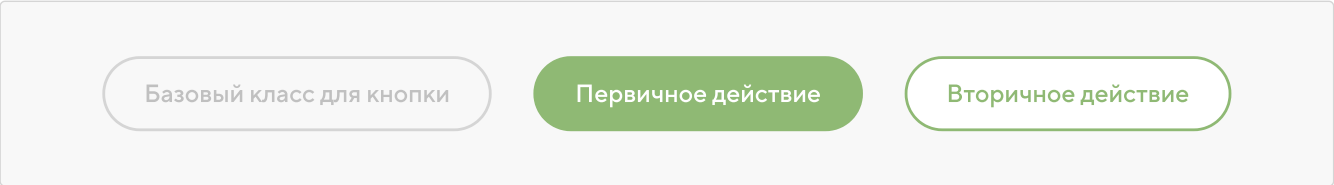
Развитие проекта. Меняется вид разделов

Пусть в начале каждый блок в макете будет отдельным БЭМ-блоком, не стараемся сразу его сделать универсальным и переиспользовать.

Плохая абстракция, неверно выделенная общая часть неизбежно приведёт к мудрению в коде. Мы пытаемся объединить в один БЭМ-блок сущности, разные по смыслу (блок с рекламой и раздел для выбора курса). Не факт, что такая композиция приживётся.

Допустим, вы убедились, что этот блок может быть переиспользован. Тогда смотр различается ли реализация блока от места к месту. Если да, значит определяем б состояние (базовый класс) этого компонента и все его модификации.

Бывает так, что базовое состояние компонента не используется отдельно без модификации. Стратегия следующая: допустим, есть кнопка; смотрим, какие в принципе кнопки есть на сайте, собираем все варианты. Если кнопки сильно отличаются друг от друга, возможно, для базового класса нужно выбрать кнопку с минимальным количеством стилей. А все варианты кнопок — это модификации базового класса.



Базовое состояние и все модификации блока Кнопка

Правило 3

Правильный блок знает всё о своих элементах, но ему всё равно, что находится во вложенных в него блоках. Если блок хочет управлять содержимым своих внутренних/вложенных блоков, значит, что-то пошло не так.

Стараемся убирать излишнюю связанность БЭМ-блоков. Помним, что один БЭМ-блок, равно как и его стили, не должен влиять на другой БЭМ-блок (его стили) *никогда, никогда, никогда, никогдашечки* ©.

Если перевести это правило в CSS — не используется межблочный каскад, к примеру, такой:

```
.menu .logo {
  ...
}
```

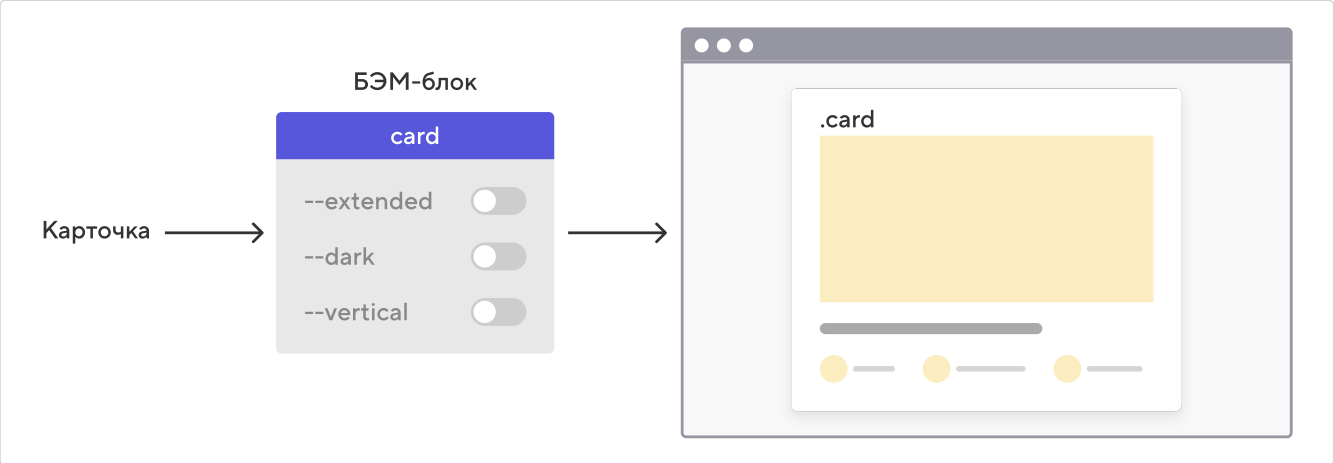
Если вам нужно изменить поведение/отображение для логотипа внутри блока с меню, можно сделать его элементом меню и стили добавить для этого элемента, к примеру, так:

```
<nav class="menu">
  <a class="menu__logo logo" href="index.html">
    <img class="logo__img" src="img/logo.svg" width="120" height="30" alt="">
  </a>
</nav>
```

```
.menu__logo {
  padding-top: 20px;
  padding-bottom: 35px;
}
```

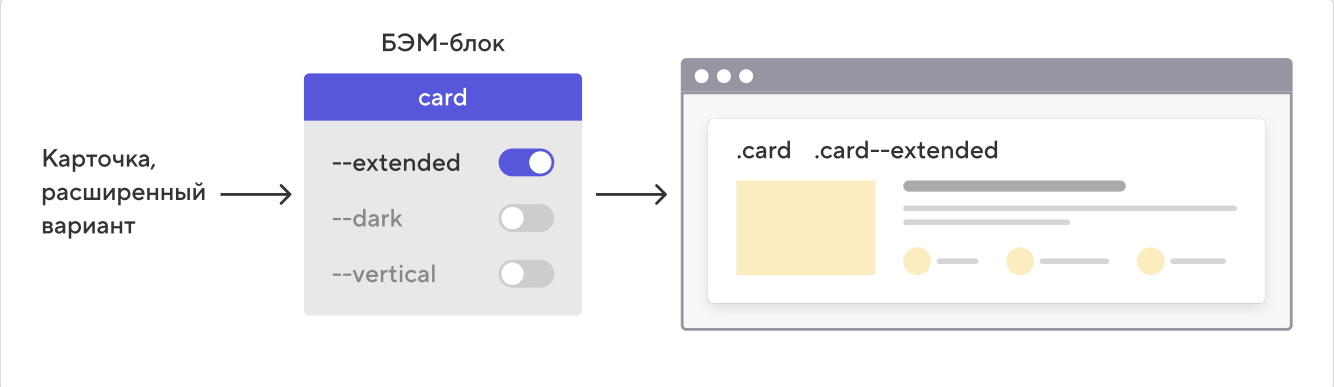
У каждого блока есть свои зоны ответственности, блок не должен хотеть знать о соседних блоках и элементах, нижних или верхних уровнях абстракций дерева вложенности.

На разметку и стили блока извне мы повлиять не можем.



Обычный блок

Если мы хотим получить модификацию блока, то на вход мы подаём параметры, в нашем случае — модификатор.



Блок с параметрами

По заданным параметрам блок изменяет своё отображение на странице.

Кроме как через модификаторы, больше никак на внутренности блока мы не можем воздействовать.

Если составной блок всё-таки воздействует на стили вложенных блоков, это означает, что композиция составлена неверно. Нужно поделить на блоки по-другому.

> Внешняя и внутренняя модификация блока

Правило 4

БЭМ-блок должен иметь только одну причину для изменения.

Это уже *высшая* ступень при составлении композиции. То, к чему нужно стремиться.

Что за «одна причина»? Формулировка уже устоялась в таком виде, хотя следовало бы сказать: «чтобы изменить один параметр в интерфейсе, потребуется изменить только один БЭМ-блок (или минимальное их количество)».

Имеется в виду, что сайт — это интерфейс, прослойка между пользователем и кодом. И каждый блок — это интерфейсный кусок макета, и речь идёт об изменениях в интерфейсе (поменялся размер шрифта в каком-то элементе, заменили фон в секции или, возможно, немного пересобрали карточку, сделали кнопки поярче). Вот это атомарное, одно изменение должно в идеале исправляться правкой свойств в одном блоке (компоненте, модуле, файле), или хотя бы в минимальном их количестве.

Мы сейчас не говорим о редизайне, это глобальное изменение, обычно затрагивающее переработку всего интерфейса.

Например, блок с подвалом сайта. В подвале может быть что угодно: логотип, контактная информация, сервисное меню, подписка на рассылку и другое. Ошибкой будет сделать весь подвал одним простым блоком, в котором есть только элементы. Блок подвала может измениться? Да. А как? Добавятся ещё элементы и блоки со своими стилями, изменится состав меню, появится копирайт и ссылка на пользовательское соглашение, поменяется логотип. Нужно ли его разбивать на ещё более мелкие блоки? Да. Лучше разбить на более мелкие частицы: отдельный блок для логотипа, отдельный блок для формы на рассылку и так далее. Теперь проверим, как, к примеру, может измениться блок с логотипом. Может измениться его внешний вид, либо есть два варианта логотипа, светлый для тёмного фона и тёмный для светлого фона. Это блок с одной причиной для правки, а значит — это правильный отдельный БЭМ-блок.

Подвал сайта с вложенными блоками

Так же мы можем выделить БЭМ-блок *Контакты*, БЭМ-блок с вспомогательной навигацией и так далее.

Правило 5

Используем композицию вместо наследования.

Это правило поможет нам избавиться от «жадных» блоков, которые хотят управлять всем сразу (декором, сеткой, отступами).

Один и тот же HTML-элемент сразу может быть несколькими БЭМ-блоками. Это помимо того, что мы можем миксовать, и один и тот же HTML-элемент будет БЭМ-блоком и БЭМ-элементом. Не бойтесь добавлять много классов для одного HTML-элемента.

```
<section class="product card">
  <div class="product__description-column card__description-column offer">
    <h2 class="offer__title">Дом как произведение искусства</h2>
    <p class="offer__description">Расположенный в самом центре Москвы квартал
класса De luxe «Театральный Дом»</p>
    <button class="offer__button button button--accent"
type="button">Подробнее</button>
  </div>
  <div class="product__img-column card__img-column">
    <img class="product__img" src="img/photo.jpg" width="470" height="332"
alt="">
  </div>
</section>
```

Правило 6

Создаём БЭМ-блоки в изоляции (правило необязательное, но с ним, особенно поначалу, ваша жизнь станет легче).

Проще, когда вы создаёте каждый компонент отдельно от контекста его применения. Это напоминает разработку компонента в вакууме или «песочнице». А затем страница формируется как конструктор из таких компонентов.

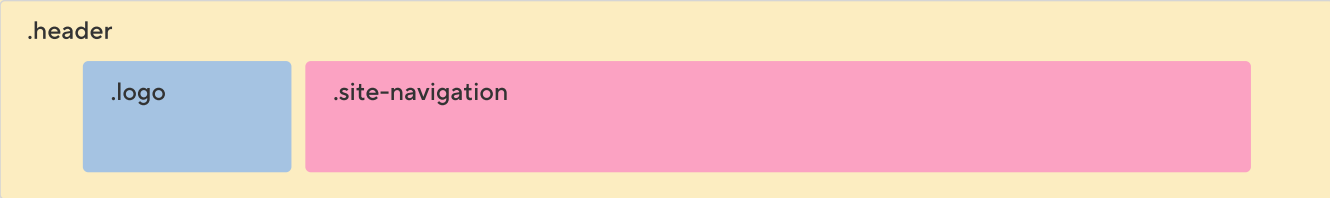
Технически это можно сделать следующим образом: для каждого компонента создаётся отдельный html-файл. В нем выполняется разметка компонента. Вы увидите, как ведёт себя и как выглядит компонент в отрыве от всей страницы.

```
<!DOCTYPE html>
<html lang="ru">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Компонент</title>
  </head>
  <body>
  <!-- тут разметка для компонента -->
```

```
  </body>
</html>
```

В реальной разработке для создания компонентов в вакууме используют HTML-препроцессоры или шаблонизаторы (*Pug/Jade, Haml, Handlebars*). При разработке с помощью современных библиотек и фреймворков также нет проблем с созданием отдельных независимых компонентов. Так, при разработке *React*-приложений каждый компонент — это отдельный модуль системы.

В файле, где описывается разметка для простого БЭМ-блока, не должно быть других блоков и их кусков. Это поможет нам не смешивать компоненты и правильно определять зависимости.



Создание сложного блока, в который вложены два других простых блока

Блок с логотипом:

```
<a class="logo" href="index.html">
  <img src="img/logo.svg" width="120" height="35" alt="">
</a>
```

Блок с меню:

```
<ul class="site-navigation">
  <li class="site-navigation__item">
    <a class="site-navigation__link site-navigation__link--active"
href="#">Обувь</a>
  </li>
  <li class="site-navigation__item">
    <a class="site-navigation__link" href="#">Новинки</a>
  </li>
  <li class="site-navigation__item">
    <a class="site-navigation__link" href="#">Мужчинам</a>
  </li>
  <li class="site-navigation__item">
    <a class="site-navigation__link" href="article.html">Женщинам</a>
  </li>
</ul>
```

Композиция блоков:

```
<header class="header">
  <!-- Блок с логотипом -->
  <!-- Блок с меню/навигацией по сайту -->
</header>
```

В свою очередь блок с шапкой — это вложенный блок для блока со страницей сайта:

```
<body class="page">
  <!-- Блок с шапкой -->
  <!-- ... другие блоки -->
</body>
```

Если блокам внутри родительского блока нужно выравнивание, определение размера, внешние отступы, то вложенные блоки заворачиваем в элементы, или используем *миксование*.

Ссылка на профиль пользователя и одновременно пункт меню в пользовательском меню

Композиция блоков:

```
<header class="header">
  <!-- Элемент шапки страницы -->
  <div class="header__logo">
    <!-- Блок с логотипом (код блока) -->
  </div>
  <!-- Элемент шапки страницы -->
  <div class="header__nav">
    <!-- Блок с меню/навигацией по сайту (код блока) -->
  </div>
</header>
```

Или так:

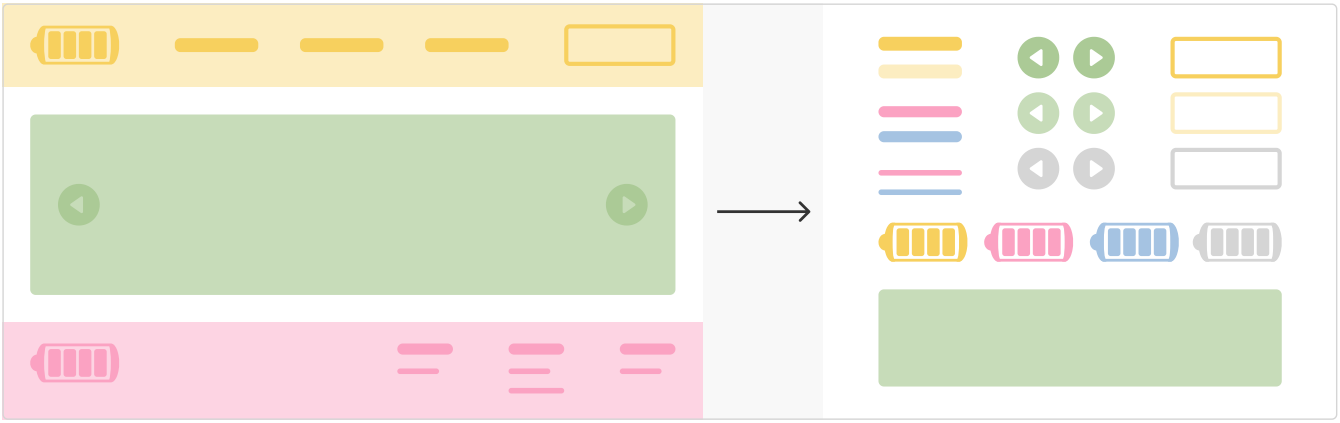
```
<header class="header">
  <!-- Элемент шапки страницы и Блок с логотипом -->
  <a class="header__logo logo" href="index.html">
```

```
    <img src="img/logo.svg" width="120" height="35" alt="">
  </a>
  <!-- Элемент шапки страницы и Блок с меню/навигацией по сайту -->
  <ul class="header__nav site-navigation">
    <li class="site-navigation__item">
      <a class="site-navigation__link site-navigation__link--active"
href="#">Обувь</a>
    </li>
    <li class="site-navigation__item">
      <a class="site-navigation__link" href="#">Новинки</a>
    </li>
    <li class="site-navigation__item">
      <a class="site-navigation__link" href="#">Мужчинам</a>
    </li>
    <li class="site-navigation__item">
      <a class="site-navigation__link" href="article.html">Женщинам</a>
    </li>
  </ul>
</header>
```

Что мы получим в итоге?

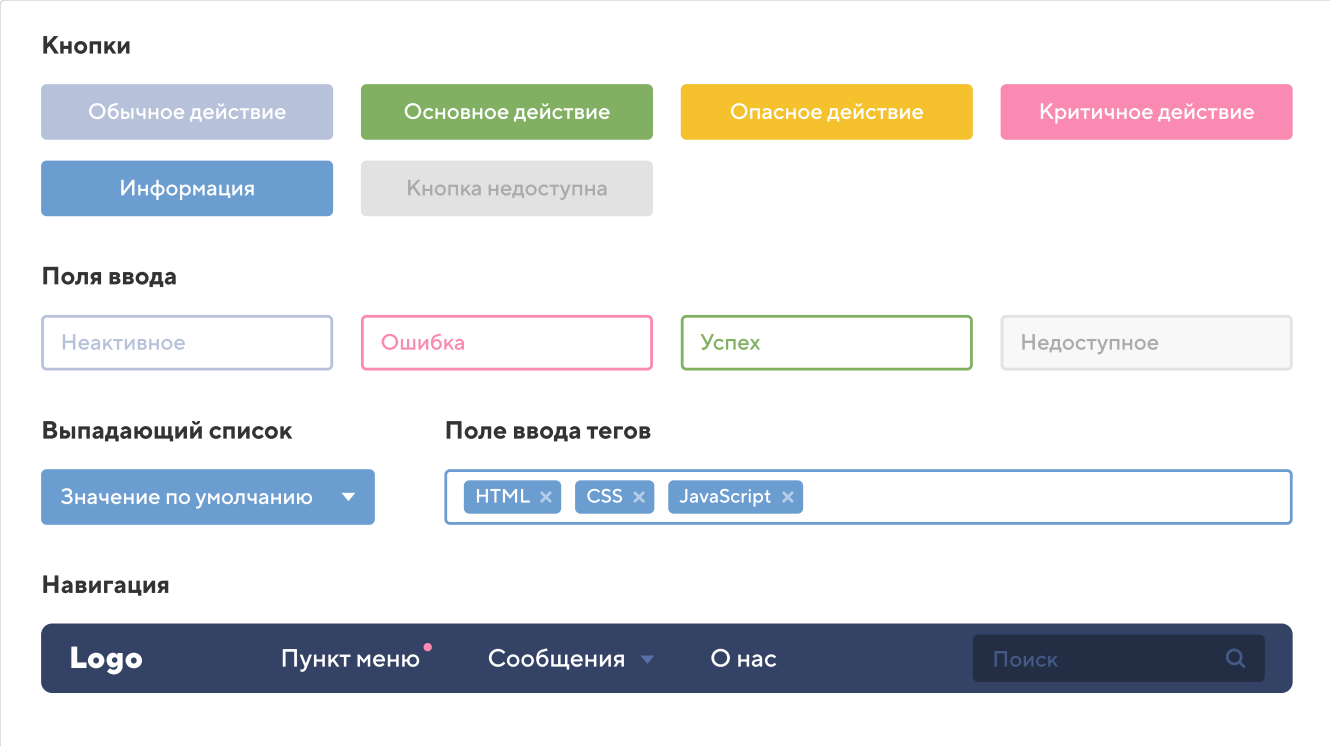
Нам нужно весь дизайн декомпозировать в *плоское* дерево компонентов, где компоненты не зависят друг от друга, а только вкладываются друг в друга. Зависимые и связанные компоненты — это плохой результат декомпозиции, такого быть не должно.

Все простые компоненты собираются в так называемый *UI-kit*. Причём в *UI-kit*-е помещаются не только базовые блоки, но и все их модификации. Всё это будет строительным материалом для нашего проекта. Мы будем собирать проект из компонентов, составлять их композицию.



UI-kit для сайта интернет-магазина

Понятие *UI-kit* пришло из дизайна, и выглядит *UI-kit* примерно так:



UI-kit

> *UI-kit* — это набор всех элементов, на которых строится *UI* (*user interface*). Это могут быть кнопки, поля ввода, «хлебные крошки», меню, переключатели, формы — все те элементы, что помогают пользователям взаимодействовать с сайтом или приложением. Тут же могут быть более крупные элементы интерфейса, к примеру, карточки товара, слайдеры и другие.

Работа с макетом у дизайнеров

Для верстальщиков *UI-kit* — это не только внешний вид компонентов, но и реализация этих компонентов. И это тоже способ получить единообразный, понятный и переиспользуемый код. Не нужно изобретать велосипед, вы просто подключаете нужный компонент на страницу. Если выясняется, что какой-то компонент отсутствует в *UI-kit*, его добавляют.

В веб-студиях, как правило, уже есть готовый *UI-kit* типовых компонентов, и процесс создания сайтов заключается в сборке этих компонентов с небольшой модификацией под стиль заказчика. В результате готовый проект или сайт получается в разы быстрее по сравнению с обычной вёрсткой с нуля. Более того, в крупных компаниях дизайнеры не просто создают макет, а сами верстают компоненты для *UI-kit*-а.

Компонентный подход сейчас поддерживают и графические редакторы. К приме в *Фигме* можно создать отдельную библиотеку с компонентами и уже оттуда под готовые компоненты в макет.

Важно при создании компонентов выработать системность, даже если в дизайне этой системности нет (например, если отступы везде разные). Подгонка модификаторами, если захочется подправить элемент — не выход.

Прочитали главу?

Нажмите кнопку «Готово», чтобы сохранить прогресс.

Готово

Если вы обнаружили ошибку или неработающую ссылку, выделите ее и нажмите Ctrl + Enter

Поиск по материалам

Git

Все материалы

В самом начале

- [ ] Пройдите опрос
- [ ] Укажите персональные данные
- [ ] Изучите регламент
- [ ] Прочитайте FAQ
- [ ] Добавьте свой Гитхаб
- [ ] Выберите наставника
- [ ] Создайте проект

Мой наставник

Выбрать наставника

Работа с наставником

У вас осталось **10** из 10 консультаций.

История




Практикум

Тренажёры

Подписка

Для команд и компаний

Учебник по PHP

Профессии

Фронтенд-разработчик

JavaScript-разработчик

Фулстек-разработчик

Курсы

HTML и CSS. Профессиональная вёрстка сайтов

HTML и CSS. Адаптивная вёрстка и автоматизация

JavaScript. Профессиональная разработка веб-интерфейсов

JavaScript. Архитектура клиентских приложений

React. Разработка сложных клиентских приложений

Node.js. Профессиональная разработка REST API

Node.js и Nest.js. Микросервисная архитектура

TypeScript. Теория типов

Алгоритмы и структуры данных

Паттерны проектирования

Webpack

Vue.js 3. Разработка клиентских приложений

Git и GitHub

Анимация для фронтендеров

Блог

С чего начать

Шпаргалки для разработчиков

Отчеты о курсах

Услуги

Работа наставником

Для учителей

Стать автором

Информация

Об Академии

О центре карьеры

Остальное

Написать нам

Мероприятия

Форум

Соглашение

Конфиденциальность

Сведения об образовательной организации

Лицензия № 4696

Sk
Участник