Содержание:

Введение

БЭМ (Блок, Элемент, Модификатор) — компонентный подход к веб-разработке. В его основе лежит принцип разделения интерфейса на независимые блоки. Он позволяет легко и быстро разрабатывать интерфейсы любой сложности и повторно использовать существующий код, избегая «Copy-Paste»

БЭМ-методология: с чего всё начиналось и зачем это всё нужно

Конечно, все началось с собственных потребностей Яндекса. Вместе с тем, как он рос, росло и количество сотрудников, которые занимаются фронтендом. Постепенно команда увеличилась настолько, что стало очевидно — без единых стандартов работать будет сложно. К тому же, мы находимся в офисах Яндекса в разных городах. Возникла идея создать общую методологию, которая поможет организовать процессы в большой команде, работающей над разными проектами. А главное то, что мы хотели не только упорядочить и ускорить разработку, но и снизить порог входа в проект для нового разработчика.

Для чего нужна БЭМ-методология

- *Разработчик должен понимать свой код (даже вернувшись к нему через год) и код любого программиста в команде БЭМ-проекта.*
- *Любой блок кода может быть использован повторно: необходимо создать общую базу знаний и не писать каждый раз всё с нуля, а использовать готовые наработки.*
- *Работая в одной команде, разработчики, менеджеры, дизайнеры и верстальщики должны называть одни и те же вещи одинаково. То есть говорить на одном языке.*
- *Команды могут обмениваться специалистами для реализации какой-то конкретной функциональности.*

- *Порог входа при переходе на новый проект должен быть снижен за счет одинаковой структуры организации всех БЭМ-проектов и одинаковых правил именования всех сущностей.*

Блок

Функционально независимый компонент страницы, который может быть повторно использован. В HTML блоки представлены атрибутом class.

Особенности:

- Название блока характеризует смысл («что это?» — «меню»: menu, «кнопка»: button), а не состояние («какой, как выглядит?» — «красный»: red, «большой»: big).

Пример

<!-- Верно. Семантически осмысленный блок `error` -->

<!-- Неверно. Описывается внешний вид -->

- Блок не должен влиять на свое окружение, т. е. блоку не следует задавать внешнюю геометрию (в виде отступов, границ, влияющих на размеры) и позиционирование.
- В CSS по БЭМ также не рекомендуется использовать селекторы по тегам или id.

Таким образом обеспечивается независимость, при которой возможно повторное использование или перенос блоков с места на место.

Принцип работы с блоками

Вложенность

- Блоки можно вкладывать друг в друга.
- Допустима любая вложенность блоков.

Пример

<!-- Блок `header` -->

<header class="header">

<!-- Вложенный блок `logo` -->

<!-- Вложенный блок `search-form` -->

<form class="search-form"></form>

</header>

Элемент

Составная часть блока, которая не может использоваться в отрыве от него.

Особенности:

- Название элемента характеризует смысл («что это?» — «пункт»: item, «текст»: text), а не состояние («какой, как выглядит?» — «красный»: red, «большой»: big).
- Структура полного имени элемента соответствует схеме: имя-блока__имя-элемента. Имя элемента отделяется от имени блока двумя подчеркиваниями (__).

Пример

<!-- Блок `search-form` -->

<form class="search-form">

<!-- Элемент `input` блока `search-form` -->

<input class="search-form__input">

<!-- Элемент `button` блока `search-form` -->

<button class="search-form__button">Найти</button>

</form>

Принципы работы с элементами

- Вложенность
- Принадлежность
- Необязательность

Вложенность

- Элементы можно вкладывать друг в друга.
- Допустима любая вложенность элементов.
- Элемент — всегда часть блока, а не другого элемента. Это означает, что в названии элементов нельзя прописывать иерархию вида block__elem1__elem2.

Пример

<!--

Верно. Структура полного имени элементов соответствует схеме:

`имя-блока__имя-элемента`

-->

<form class="search-form">

<input class="search-form__input">

<button class="search-form__button">Найти</button>

</form>

<!--

Неверно. Структура полного имени элементов не соответствует схеме:

`имя-блока__имя-элемента`

-->

<form class="search-form">

<!--

Рекомендуется:

`search-form__input` или `search-form__content-input`

-->

<input class="search-form__content__input">

<!--

Рекомендуется:

`search-form__button` или `search-form__content-button`

-->

<button class="search-form__content__button">Найти</button>

</form>

Имя блока задает пространство имен, которое гарантирует зависимость элементов от блока(block__elem).

Блок может иметь вложенную структуру элементов в DOM-дереве:

Пример

Однако эта же структура блока в методологии БЭМ всегда будет представлена плоским списком элементов:

Пример

.block {}

.block__elem1 {}

.block__elem2 {}

.block__elem3 {}

Это позволяет изменять DOM-структуру блока без внесения правок в коде каждого отдельного элемента:

Пример

Структура блока меняется, а правила для элементов и их названия остаются прежними.

Принадлежность

Элемент — **всегда часть блока** и не должен использоваться отдельно от него.

Пример

<!-- Верно. Элементы лежат внутри блока `search-form` -->

<!-- Блок `search-form` -->

<form class="search-form">

```
<!-- Элемент `input` блока `search-form` -->
<input class="search-form__input">
<!-- Элемент `button` блока `search-form` -->
<button class="search-form__button">Найти</button>
</form>
<!-- Неверно. Элементы лежат вне контекста блока `search-form` -->
<!-- Блок `search-form` -->
<form class="search-form">
</form>
<!-- Элемент `input` блока `search-form` -->
<input class="search-form__input">
<!-- Элемент `button` блока `search-form` -->
<button class="search-form__button">Найти</button>
```

Необязательность

Элемент — необязательный компонент блока. Не у всех блоков должны быть элементы.

Пример

```
<!-- Блок `search-form` -->
<div class="search-form">
<!-- Блок `input` -->
<input class="input">
<!-- Блок `button` -->
<button class="button">Найти</button>
```

Когда создавать блок, когда — элемент?

Создавать блок

Если фрагмент кода может использоваться повторно и не зависит от реализации других компонентов страницы.

Создавать элемент

Если фрагмент кода не может использоваться самостоятельно, без родительской сущности (блока).

Исключение составляют элементы, реализация которых для упрощения разработки требует разделения на более мелкие части — подэлементы. В БЭМ-методологии нельзя создавать элементы элементов. В подобном случае вместо элемента необходимо создавать служебный блок.

Микс

Прием, позволяющий использовать разные БЭМ-сущности на одном DOM-узле.

Миксы позволяют:

- совмещать поведение и стили нескольких сущностей без дублирования кода;
- создавать семантически новые компоненты интерфейса на основе имеющихся.

Пример

<!-- Блок `header` -->

<!-- К блоку `search-form` примиксован элемент `search-form` блока `header`-->

В данном примере совместили поведение и стили блока search-form и элемента search-form блока header. Такой подход позволяет нам задать внешнюю геометрию и позиционирование в элементе header__search-form, а сам блок search-

form оставить универсальным. Таким образом, блок можно использовать в любом другом окружении, потому что он не специфицирует никакие отступы. Это позволяет нам говорить о его независимости.

История развития БЭМ

Как верстали 10 лет назад

Все верстали сайты, складывая CSS в один файл project.css, скрипты, которых было очень мало, — в project.js, а картинки — в папку images.

В 2005 году обычный, с точки зрения интерфейса, проект был набором статических HTML-страниц. Вот такой была типичная структура проекта того времени:

about.html *# Для каждой страницы создавался отдельный HTML-файл*

index.html

...

project.css *# Стили находились в одном файле для всего проекта*

project.js *# Скрипты хранились в одном файле для всего проекта*

images/ *# Картинки складывались в отдельную директорию*

yandex.png

В CSS использовались id, классы и теги.

Пример

#foot div div div div

{

```
background-position: 54%;

background-image: url(../i/foot-4.png);

}
```

Типичный CSS того времени в большинстве случаев содержал длинный каскад.

Малейшие изменения требовали длительного рефакторинга. Свёрстанные статические HTML-страницы нарезались в шаблоны. Если HTML изменялся, все правки было необходимо переносить вручную в шаблон.

Вёрстка в больших проектах была неуправляемой.

Основы БЭМ-методологии

Технологии (HTML, CSS, JavaScript), которые мы использовали, изменялись в зависимости от требований проекта, а принципы БЭМ должны были быть универсальны.

Мы сформулировали основные правила, по которым будут жить и развиваться наши проекты, и которые никак не будут зависеть от технологий и инструментов.

Чтобы ускорить разработку, необходимо было облегчить поддержку HTML и CSS отдельных компонентов страницы, сделать код менее связанным. Для этого мы разбили страницу на части. Так появилось новое понятие - блок. Блок мог состоять из различных элементов, которые не использовались вне самого блока. Состояния и поведение блока и элемента можно было задавать с помощью модификатора. Это были три ключевых понятия, на которых основывалось большинство правил. Аббревиатура от трех слов **Б**лок, **Э**лемент и **М**одификатор стала названием методологии — **БЭМ.**

Правила именования CSS-селекторов

Все принципы БЭМ формировались и внедрялись постепенно. Мы начали с того, что сформулировали жесткие правила именования CSS-селекторов.

По БЭМ-методологии блоки не уникальны и их всегда можно использовать повторно, поэтому в описании CSS-правил отказались от использования id.

Блок не должен зависеть от окружающих его блоков и сам не должен влиять на соседние блоки, поэтому в CSS отказались от:

- тегов;
- вложенных селекторов;
- глобального сброса правил для всей страницы.

Важной определяющей сущностью в именовании селекторов стал блок:

- Полное имя элемента/модификатора формируется так, чтобы из него можно было определить принадлежность данного элемента/модификатора к конкретному блоку.
- По имени модификатора элемента должно быть возможно определить принадлежность данного модификатора к конкретному элементу конкретного блока.

Правила формирования имени БЭМ-сущности

- Каждая БЭМ-сущность должна иметь свой класс.
- CSS-свойства для блоков, элементов и модификаторов описываются только через классы.
- Для разделения слов в именах используется дефис (-).
- Элемент отделяется от блока двумя подчеркиваниями (__).
  Модификатор — одним (_).
- Имена БЭМ-сущностей записываются с помощью цифр и латинских букв в нижнем регистре.

Пример

- Имя блока — header.
- Имя элемента блока — header__search-form — элемент search-form блока header.
- Имя модификатора блока — header_theme_green-forest — модификатор theme в значении green-forest блока header.
- Имя модификатора элемента — header__search-form_disabled — булев модификатор disabled элемента search-form блока header.

HTML

- …

CSS

- .header { color: red; }

Существует ряд альтернативных схем именования. Выбор всегда остается за разработчиком.

Но я рекомендую придерживаться описанной выше схемы, так как инструменты БЭМ-платформы умеют работать именно с данным вариантом именования.

Заключение

Заготовка проекта

Быстро начать разработку БЭМ-проекта можно с помощью project-stub — проекта с заранее предустановленными технологиями и инструментами. Начинать знакомство с ним стоит с помощью быстрого старта по БЭМ.

И в заключение

БЭМ-методология — это набор правил и рекомендаций по организации работы над проектом.

В какой-то момент мы отделили методологию от ее практической реализации — платформы.
БЭМ-платформа — это частный случай реализации общих принципов БЭМ-методологии. Так как все технологии создавались с учетом требований наших проектов и развивались постепенно, БЭМ-платформа наиболее полно охватывает все возможности, которые предоставляет БЭМ-методология.

Все части БЭМ-платформы интегрированы для совместной работы, но могут быть использованы и по отдельности. Каждая часть решает конкретную задачу и её можно настраивать под свой процесс и заменять на другие.

Список используемой литературы

Varvara Stepanova: A New Front-End Methodology: BEM

Maxim Shirshin: Scaling Down The BEM Methodology For Small Projects

Harry Roberts: MindBEMding – getting your head ’round BEM syntax

Robin Rendle: BEM 101

Edward Eluson: BEM on habr