УДК 004

БЭМ. КОМПОНЕНТНЫЙ ПОДХОД К ВЕБ-РАЗРАБОТКЕ

Попков И.В. [1], Курзаева Л.В. [2]

[1] *ФГБОУ ВО «Магнитогорский государственный технический университет имени Г.И. Носова»Магнитогорск, Россия email: iliailiaq2@mail.ru.*
[2]*ФГБОУ ВО «Магнитогорский государственный технический университет имени Г.И. Носова» Магнитогорск, Россия email: lkurzaeva@mail.ru.*

Для высоконагруженных проектов, содержащих большое количество строк кода и модулей самого приложения, одной из насущных проблем является сложность их поддержки. Поиск путей разрешения данной проблемы актуален как для обеспечения работоспособности таких приложений, так и повышения понятности и читабельности кода в ходе дальнейшего их развития и, вероятной, смены разработчиков. Так, зачастую, придя в уже готовый проект, разработчик тратит очень много времени на то, чтобы разобраться с текущим состоянием приложения, его структуре и коде. Одним из способов решения данной проблемы является использование компонентного подхода «Блок, элемент, модификатор» (БЭМ), заключающегося в приведении всего написанного кода к единой структуре, в обеспечивающий для нового разработчика минимальный порог вхождения в проект. В статье подробно рассмотрены основные составляющие БЭМ дефиниции: блок, элемент и модификатор, особенности их построения и способы работы с ними. Кратко приведена методика реализации проекта с использованием данного подхода: описана файловая структура проекта при использовании БЭМ подхода, выделены основные принципы разработки с использованием данной структуры, описана технология сборки проекта из разрозненных файлов. Подробно объяснена суть деклараций, списка, позволяющего делать сборку проекта только по определенным параметрам и /или только определенных блоков, что дает возможность избавиться от дублирования кода. Статья представляет интерес для разработчиков веб-приложений.

Ключевые слова: БЭМ, веб-разработка, компонентный подход, структура проекта.

BEM. COMPONENT APPROACH TO WEB DEVELOPMENT

Popkov I.V. [1], Kurzaeva L.V. [2]

[1] *FGBOU VO "Magnitogorsk State Technical University named after G.I. Nosov », Magnitogorsk, Russia email: iliailiaq2@mail.ru.*
[2] *FGBOU VO "Magnitogorsk State Technical University named after G.I. Nosov », Magnitogorsk, Russia email: lkurzaeva@mail.ru.*

For highly loaded projects that contain a large number of lines of code and modules of the application itself, one of the most pressing problems is the complexity of their support. The search for ways to solve this problem is relevant both for ensuring the operability of such applications, as well as for improving the comprehensibility and readability of the code in the course of their further development and, probably, the change of developers. So, often, having come to an already ready project, the developer spends a lot of time trying to figure out the current state of the application, its structure and code. One of the ways to solve this problem is to use the component approach "Block, element, modifier" (BEM), which consists in bringing all the written code to a single structure, which provides the minimum threshold for the new developer to enter the project. The article considers in detail the main components of the BEM definition: the block, the element and the modifier, the features of their construction and the ways of working with them. Briefly, the methodology for implementing the project using this approach is described: the file structure of the project is described using the BEM approach, the basic principles of development using this structure are described, and the technology for assembling the project from disparate files is described. The essence of the declarations, the list, allowing to do the project assembly only on certain parameters and / or only certain blocks, that allows to get rid of duplication of the code is explained in detail. The article is of interest to web application developers.

Key words: BEM, web development, component approach, project structure.

При разработке веб-приложений, особенно в команде, у многих веб-разработчиков возникают ряд проблем, связанных с вроде бы тривиальной

задачей «понимания кода». Причины этого могут быть различны, но они, прежде всего, связаны со структурой самого приложения и полхода к разработке кода.

Приведем некоторые типовые ситуации. CSS-правила, которые не явно указывают на блок, которому они заданы, в следствие чего при изменении этого класса веб-страницы могут «разъехаться» и сломаться.

Другой пример связан с трудностями в использовании кода повторно и взаимного влияния компонентов друг на друга. Зачастую, при использовании одного и того же компонента, например, menu, на нескольких страницах может серьезно нарушить работу всех компонентов при каком-то изменении самого компонента, либо добавления в него еще одного блока. В случае если компонент страницы нужен для повторного использования, код просто копируется и вставляется в нужное место, а затем редактируется. Это увеличивает кодовую базу проекта и усложняет отладку кода при выявлении ошибки.

При появлении нового разработчика на проекте, ему приходится тратить очень много времени на то, чтобы понять, как все работает, как устроен код, разобраться в архитектуре проекта. Зачастую именно на это уходит большая часть времени, а не на устранения ошибки или внесение правок.

Для решения данных проблем был разработан БЭМ. Расшифруется данная аббревиатура так: «Блок, элемент, модификатор». Это некий набор абстракций, на которые можно разбить интерфейс и разрабатывать всё в одних и тех же терминах. БЭМ предлагает единые правила написания кода, помогает его масштабировать и повторно использовать, а также упрощает работу в команде и увеличивает производительность.[1]

Основная идея БЭМ это независимые блоки, суть данной идеи заключается в том, чтобы использовать минимум глобальных стилей и каждый элемент страницы делать полностью блоком со своими уникальными стилями, классами, которые его описывают. Также все селекторы по названиям элементов и ID элементов заменяются на селекторы со своими уникальными

классами. Каждый класс в стилях это блок, данная идея позволяет легко менять блоки местами, вкладывать друг в друга и не бояться влияния одного блока на другой или конфликтов между стилями блоков.

Рассмотрим каждый элемент БЭМа отдельно.

Блок это функционально независимый компонент страницы, может быть использован повторно.

Особенности блока:

1) Его название характеризует смысл, а не его состояние(например блок меню должен иметь класс menu, а не yellow или small);
2) Блок не влияет на окружающие его блоки, то есть не задает внешнюю геометрию и позиционирование;
3) CSS свойство не может определяться по ID или названию html-тега;
4) Блоки можно вкладывать друг в друга;
5) Вложенность блоков не ограничена.

Элемент это главная, составная часть блока, не может использоваться без него.

Особенности элемента:

1) Название характеризует чем является этот элемент(например класс item, если элемент является частью меню);
2) Название задается при помощи двойного подчеркивания, сначала идет название блока, а затем через два подчеркивания название самого элемента, например menu__item;[2]

При работе с элементами необходимо соблюдать три основных принципа:

1) Вложенность. Допускается вкладывать несколько элементов друг в друга, элемент часть блока, а не другого элемента, то есть в названии класса элемента нельзя писать block1__elem1__elem2, только block1__elem1 и block1__elem2. На рисунке 1 представлен пример вложенности.

```
<!-- Блок `header` -->
<header class="header">
    <!-- Вложенный блок `logo` -->
    <div class="logo"></div>

    <!-- Вложенный блок `search-form` -->
    <form class="search-form"></form>
</header>
```

Рисунок 1 – Пример вложенности блоков

2) Принадлежность. Элемент всегда часть блока, следовательно не может использоваться в отрыве от него.
3) Необязательность. Элемент не является обязательным компонентом блока, следовательно, в блоке может и не быть компонента.

Модификаторы. Определяет внешний вид, поведение или состояние блока.

Особенности модификаторов:

1) Название модификатора характеризует внешний вид(размер, тему), состояние(отличия от других модификаторов, например «отключен») и поведения(как себя ведет, как взаимодействует с пользователем);
2) Имя модификатора отделяется от имени блока при помощи одного подчеркивания.

Существует два типа модификатора, булевый и ключ-значение. [3]

Булевый используют когда важно наличие или отсутствие модификатора при этом его значение не существенно. Например «отключенный» модификатор обозначается как «disabled». При наличии булевого модификатора его значение равно true. В таком случае структура модификатора соответствует следущей записи: название-блока_название-модификатора или, если есть элемент, тогда: название-блока__название-элемента_название-модификатора.

Ключ-значение используется если значение модификатора важно. Тогда структура имени модификатора будет выглядеть следующим образом: название-блока__название-элемента_название-модификатора_значение-модификатора. В случае, если элемент отсутствует то таким образом: название-блока_имя-модификатора_значение-модификатора. Пример использования ключ-значения представлен на рисунке 2.[4]

```
<!-- Блок `search-form` имеет модификатор `theme` со значением `islands` -->
<form class="search-form search-form_theme_islands">
    <input class="search-form__input">

    <!-- Элемент `button` имеет модификатор `size` со значением `m` -->
    <button class="search-form__button search-form__button_size_m">Найти</button>
</form>

<!--
    Невозможно одновременно использовать два одинаковых модификатора
    с разными значениями
-->
<form class="search-form
             search-form_theme_islands
             search-form_theme_lite">

    <input class="search-form__input">

    <button class="search-form__button
                   search-form__button_size_s
                   search-form__button_size_m">
        Найти
    </button>
</form>
```

Рисунок 2 – Пример использования ключ-значения

Нельзя использовать модификатор самостоятельно, его нельзя отрывать от модифицируемого блока или элемента. Модификатор может лишь изменять вид, состояние сущности или ее поведение, а не полностью заменять ее.

Существует один интересный прием, позволяющий комбинировать различные БЭМ-сущности на одном DOM-узле. Этот прием называется «миксы», он позволяет разработчику совмещать стили нескольких сущностей и их поведение без необходимости дублировать код. Для понимания приведен код на рисунке 3.

```
<!-- Блок `header` -->
<div class="header">
    <!-- К блоку `search-form` примиксован элемент `search-form` блока `header`-->
    <div class="search-form header__search-form"></div>
</div>
```

Рисунок 3 – Пример использования миксов

На данному рисунке совмещено поведение и стили блока search-form и элемента search-form из блока header. Это позволяет задать позиционирование и внешнюю геометрию в элемента header__search-form, оставив блок search-form универсальным. Это дает возможность использовать блок search-form в другом окружении, так как он не задает никакие отступы, следовательно этот блок является независимым.

Помимо работы с компонентами, БЭМ позволяет организовывать файловую структуру проекта. Реализация элементов, блок, модификаторов делится на независимые файлы-технологии, что позволяет их подключить по необходимости.

Особенности работы с файловой структурой в понимании БЭМ:

1) На каждый отдельный блок создается отдельная директория;

2) Название директории и блока, который находится в ней должны совпадать;

3) Реализация самого блока делится на отдельные файлы-технологии, то есть создаются css и js файлы по названию блока и эти файлы описывают только этот блок;

4) Директория с блоком является содержащей для элементов и модификаторов, которые в нем находятся;

5) Имена директория модификаторов и элементов начинаются с одинарного и двойного подчеркивания, соответственно;

6) Различные реализации элементов и модификаторов делятся на отдельные файлы-технологии.

Использование данной организации хранения позволит легко поддерживать и повторно использовать код. При этом стоит понимать, что для production версии продукта, все это будет собрано в единые файлы css и js. Пример файловой структуры представлен на рисунке 4.

```
project
    common.blocks/
        input/                              # Директория блока input
            input.css                       # Реализация блока input в технологии CSS
            input.js                        # Реализация блока input в технологии JavaScript
        popup/                              # Директория блока popup
            popup.css                       # Реализация блока popup в технологии CSS
            popup.js                        # Реализация блока popup в технологии JavaScript
```

Рисунок 4 – Пример файловой структуры

Разделение файлов на множество несвязанных друг с другом порождает проблему использования проекта на production, данную проблему решает «сборка» проекта. Сборка проекта позволяет собирать исходные файлы из различных директорий, подключать только необходимые блоки, обрабатывать исходный код файлов, например, преобразовывать sass в css или js в jsx. Для наглядности схема работы сборщика проекта представлена на рисунке 5. [5]

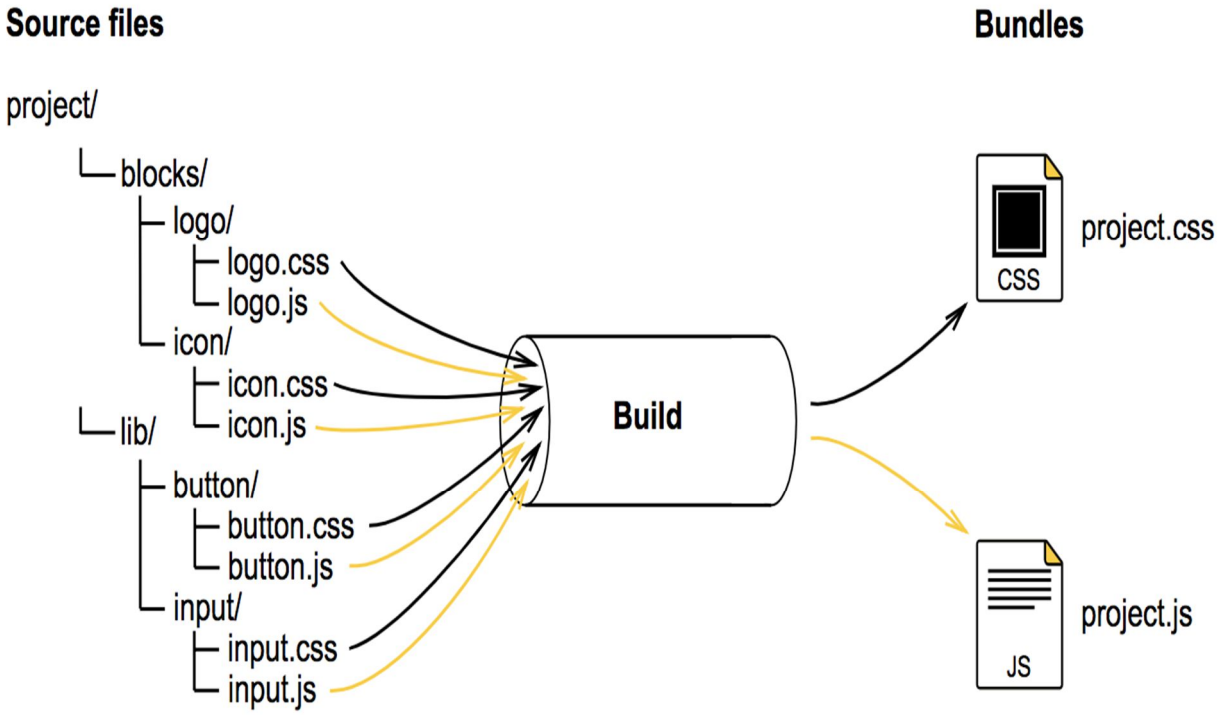

Рисунок 5 – Пример работы сборщика проекта

Весь процесс сборки можно поделить на три этапа: составление списка БЭМ-сущностей, установления зависимостей между ними, порядок их подключения. Для включения в сборку проекта только необходимых БЭМ-сущностей, составляется список всех используемых на странице элементов, данный список называется декларацией. Благодаря нему, можно избавиться от лишнего кода. Так как блоки могут строятся на основании зависимости друг от друга, необходимо указывать зависимости одних блоков от других, что также избавит разработчика от ненужного кода.

В результате сборки будут получены два файла, в одном будет собран весь javascript проекта, а в другом все стили проекта. Но, в случае, если проект очень большой, то можно делать специальные сборки под каждую отдельную страницу, и загружать их только при необходимости, это ускоряет работу продукта в разы.

Самым сложным при сборке проекта является создание декларации. Декларация это список блоков, элементов и модификаторов на странице. Инструмент, которым вы собираете проект, на основании этого файла ограничивает сущности, попадающие в сборку, что позволяет не подключать все блоки сразу, а только те, которые нужны. Следовательно, вытекает основная задача декларации – определение того, что должно подключаться и в каком порядке. Можно создать данный файл вручную, создав массив объектов, в которых хранится информация о блоках.

Таким образом, в данной статье было рассмотрено использование компонентного подхода БЭМ к веб-разработке, были описаны все основные сущности данного подхода, их особенности, правила работы с ними и основные идеи данных сущностей. Использование данного подхода позволит сократить время разработки, за счет повторного использования кода и упрощенной структуры самого проекта. Также, применяя данный подход масштабирование будет в разы упрощено, а также позволит создать модульное приложение.

Список литературы

1. БЭМ [Электронный ресурс] – Режим доступа: https://ru.bem.info/
2. Зачем нужен БЭМ? [Электронный ресурс] – Режим доступа: https://htmlacademy.ru/shorts/5
3. Методология БЭМ [Электронный ресурс] – Режим доступа: https://ru.bem.info/methodology/
4. Рудинский И. Д. Технология проектирования автоматизированных систем обработки информации и управления [Электронный ресурс]: учеб.пособие / И.Д.Рудинский. - М.: Горячая линия – Телеком, 2011, 304 с., - Режим доступа: http://ibooks.ru/reading.php?productid=334027 – Загл. с экрана .- ISBN 978-5-9912-0148-3
5. Google Developers. Python NDB Overview [Электронный ресурс] / Google компания разработчик ПО. Режим доступа https://developers.google.com/appengine/docs/python/ndb/, свободный (дата обращения 15.06.2018 г.)