1}Foundation, Bulma, Materialize и другие

**Плюсы использования CSS фреймворков:**

* увеличивается продуктивность, избегаются частые ошибки;
* код становится более правильным и понятным;
* хорошо документированный css фреймворк позволяет эффективней работать в команде;
* в случае хорошего фреймворка, код будет правильно работать во всех браузерах;
* на выходе получается хорошо структурированный код.

**Минусы использования CSS фреймворков:**

* чтобы разобраться с фреймворком, нужно какое-то время;
* можно столкнуться с ошибкой разработчика фреймворка;
* разрабатывая сайт, используются готовые решения, а не свои знания CSS;
* из виду упускается уникальность каждого проекта.

### 2) Polymer

### React

### Vue

* 1. Разработка отнимает много времени, т.к. большинство функционала придётся создавать самостоятельно с нуля. Даже банальные функции вроде корзины товаров и админки, которые доступны в CMS «из коробки», на фреймворк-базированном сайте придётся прописывать вручную.  Фреймворк – инструмент для профи, новичкам создать полноценный сайт с его использованием будет весьма сложно
* Полная свобода творчества. Заключается в том, что создание веб-ресурса с помощью фреймворка – это написание его кода с нуля, а не пользование «тем, что дают», как в случае с CMS.
* Определённые рамки, естественно, всё же придётся выдерживать, т.к. framework в переводе с языка оригинала и означает «рамки», но они будут заключаться лишь в использовании определённых синтаксических конструкций и общей структуры приложения.
* В CMS же, повторюсь, помимо указанных ограничений вы будете иметь дело с готовым кодом сайта, который бывает не всегда читабельным (сколько людей, столько и стилей кодирования)
* 2. Разработка заключается именно в написании нового функционала, а не в урезании имеющегося, как в случае с CMS.
* В статье с подробным обзором [плюсов и минусов CMS](http://cccp-blog.com/sozdanie-saytov/plyusy-i-minusy-cms) в числе одного из главных недостатков я называл огромную массу возможностей CMS, которые идут «из коробки», большая часть которых будет не востребована на будущем сайте.
* В то же время вся эта масса дополнений создаёт дополнительную нагрузку на железо сервера и снижает производительность сайта, т.е. он попросту тормозит.

3

**Поток**

По мере уменьшения размера экрана контент начинает занимать большую высоту экрана, и нижележащие элементы при этом могут отодвигаться все дальше от нижней границы области просмотра. Это называется «*потоком*». Вам необходимо освоиться с этой концепцией, чтобы научить ваш сайт правильно реагировать на изменение ширины экрана:

**Относительные единицы измерения**

Сайт должен одинаково хорошо выглядеть не только на больших десктопных мониторах и на крошечных телефонах, но и на всём многообразии устройств, находящихся по своим характеристикам где-то между этими крайностями.

**Максимальные и минимальные размеры**

Использовать для отображения контента всю ширину экрана без остатка – отличная идея, если мы имеем дело с экраном смартфона. Но если использовать аналогичным образом широкий экран монитора, контент просто перестанет восприниматься. Здесь нам на помощь приходит **CSS**-атрибут ‘**max-width**’.

Например, если мы задаём ширину блока контента в 100%, а максимальную ширину – в 1000 пикселей, то мы можем быть уверены, что контент будет занимать весь экран, но не выйдет за предел в 1000 пикселей по ширине:

**Вложенные объекты**

Имея на странице слишком много объектов с относительным позиционированием, сложно добиться их правильного расположения друг относительно друга.

Но можно поместить группу элементов в блок-контейнер, и в дальнейшем управлять позицией и размером контейнера. Этот приём также упрощает работу с элементами, имеющими абсолютные размеры, например, логотипами и кнопками:

**Встраиваемые или системные шрифты?**

Хотите использовать на вашем сайте крутую типографику? Используйте встраиваемые шрифты. Но учтите, что каждая гарнитура, загружаемая из Сети, увеличивает время загрузки страницы. Шрифты, уже имеющиеся на устройстве клиента, загружаются мгновенно, зато их внешний вид и соответствие дизайнерскому замыслу может оставлять желать лучшего:

**Растровые или векторные изображения?**

Если логотип вашего сайта содержит множество деталей и графических эффектов, то используйте его растровую версию. В противном случае попробуйте сохранить его в **SVG**. Для мелких иконок вы также можете использовать встраиваемый декоративный шрифт.

Каждый подход имеет свои преимущества и недостатки. В любом случае необходимо учитывать размер изображения. Все ресурсы, используемые на сайте, должны соответствующим образом оптимизироваться.

Векторные изображения «*плоских*» объектов обычно имеют крошечные размеры, но псевдо реалистичные векторные изображения со множеством кривых и градиентов могут весить больше, чем их растровые копии. И имейте в виду, что некоторые старые браузеры не поддерживают формат **SVG**:

4

### Какие правила необходимо соблюдать, делая верстку по БЭМ методологии?

1. Придумывая название блоку или элементу, ответьте себе на вопрос – *Что это?* (карточка: **card**)
2. Имя элемента соответствует схеме: *имя-блока\_\_имя-элемента* (через двойное подчеркивание).
3. Давая имя модификатору, ответьте на вопрос – *Какой?*
4. Имя модификатора отделяется от блока или элемента одним подчеркиванием: *имя-блока\_модификатор* или *имя-элемента\_модификатор*.
5. В стилях блока с элементами не должны присутствовать свойства, влияющие на внешнее окружение и размеры (отступы, границы, позиционирование). Создайте внешний блок-обертку, которому и будете задавать опасные свойства.
6. Не используйте в качестве селекторов названия тегов и **id**, только классы. Это дает взаимозаменяемость тегов, без необходимости изменять названия селекторов.

* 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.

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.

#### **Блок**

Логически и функционально независимый компонент страницы. Блок полностью самодостаточен: у него может быть свое поведение, шаблоны, стили, документация и не только. Блоки могут использоваться в любом месте страницы, повторно, даже в другом проекте.  
  
Одни блоки можно вкладывать в другие, компоновать, использовать для создания более сложных блоков.

#### **Элемент**

Часть блока, которая не может использоваться в отрыве от него и имеет смысл только в рамках своего родителя. Элементы могут быть обязательными и опциональными.  
  
Работая с элементами, важно помнить правило: не рекомендуется создавать элементы элементов. Если вложить один элемент в другой, будет невозможно изменить внутреннюю структуру блока: элементы нельзя будет поменять местами, удалить или добавить без корректировки существующего кода.

#### **Модификатор**

Свойство блока или элемента, которое меняет их внешний вид, состояние или поведение.  
Модификатор имеет имя и может иметь значение. Использование модификаторов опционально. У блока/элемента может быть несколько разных модификаторов одновременно.  
  
Так, например, с помощью модификатора можно изменить не только цвет меча, но и его функциональность (как показано в случае с красным мечом):

Ещё

Atomic CSS

# **SMACSS, FUN**