

Главная / Моё обучение / Разметка по БЭМ / Углублённая теория

Частые проблемы из-за неправильной CSSархитектуры проекта

Мы уже знаем, что из-за неправильной CSS-архитектуры проекта всё может сломаться, сейчас докажем.

Ситуация первая

На странице есть несколько похожих элементов, но в одном (или больше) случае элемент выглядит немного иначе.

Как обычно поступают: ищут родительский элемент (либо создают такой элемент искусственно), и для исключительного случая описывают следующие правила.

```
/* повторяющийся элемент */
.button {
  background-color: yellow;
  border: 1px solid black;
  color: black;
  width: 50%;
}

/* первый исключительный случай */
#sidebar .button {
  width: 200px;
}

/* второй исключительный случай */
body.homepage .button {
  background-color: white;
}
```

Какие проблемы могут появиться? В разметке элемент с классом **button** везде одинаковый, но при этом он почему-то выглядит по-разному в сайдбаре и на главной странице. То есть элемент ведёт себя непредсказуемо, другие разработчики не понимают, почему так происходит. Такой элемент плохо масштабируется: если такой виджет нужен в другом месте, то придётся добавлять ещё один селектор, чтобы реализовать это. А если дизайн или формат виджета изменится, стили нужно будет поменять в нескольких местах. В самом пессимистичном варианте эти места будут раскиданы по стилевому файлу.

К тому же такой код противоречит одному из принципов *SOLID* — открытости/закрытости, который говорит о том, что части *ПО* (программного обеспечения) должны быть открыты для расширения, но закрыты для изменения. Зачем родительскому блоку знать, какие блоки в него вложены? Если мы спроектируем систему так, что в родительский блок можно поместить что угодно, это будет более универсально и избавит нас от лишних связей.

Ситуация вторая

Усложняем селекторы, чтобы поднять специфичность правила. Такие селекторы, как правило, очень сильно зависят от HTML, что не очень хорошо, так как разметка в любой момент может измениться, а значит, придётся менять селекторы в стилях.

```
#main-nav ul li ul li div { }
#content article h1:first-child { }
#sidebar > div > h3 + p { }
```

Ситуация третья

Названия классов для элементов слишком общие.

На больших проектах высока вероятность, что простое и распространённое имя класса, как, например, title, будет использовано в другом контексте или даже само по себе. В примере ниже для заголовка карточки товара используется класс title. Но в проекте, вполне, может быть заголовок раздела с классом title. Много разных элементов с одинаковыми классами — верный путь к тому, чтобы забыть о каком-то из них при изменениях и получить настоящий баг.

```
.title {
   color: #353a5a;
}
...
.card {...}
.card .title {
   font-size: 14px;
}
.card .contents {...}
.card .action {...}
```

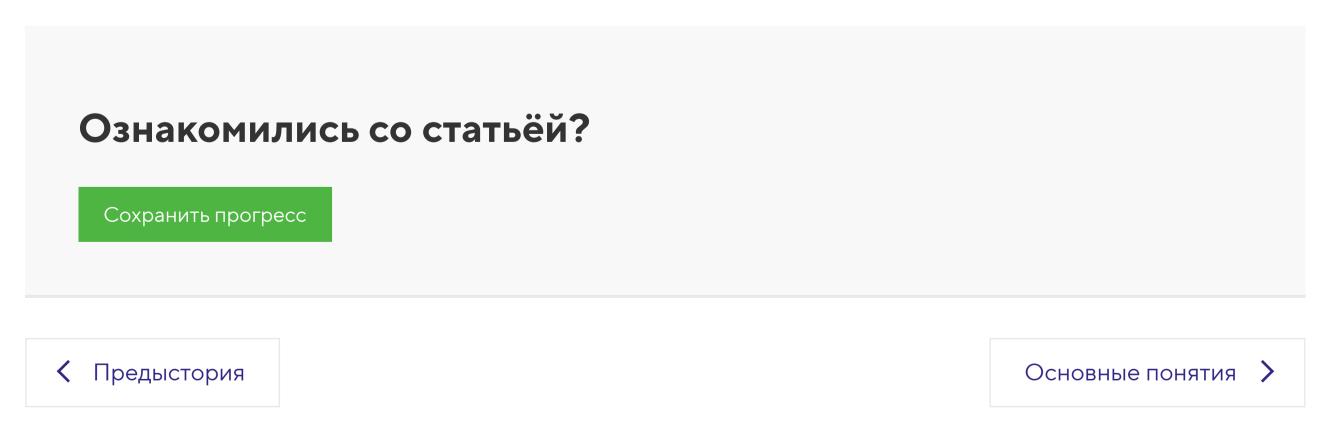
Ситуация четвёртая

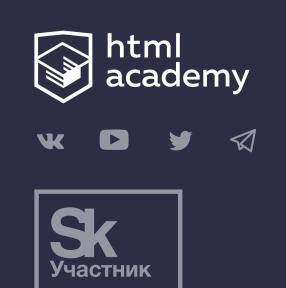
В одном правиле слишком много что определяется: и шрифт, и фон, и позиционирование, и другие параметры. Внешний вид можно использовать повторно, а раскладку и позиционирование уже нет, и если на странице (проекте) появится элемент с похожим оформлением, то придётся копировать стили. А это сложно поддерживать: чтобы изменить внешний вид, нужно вносить правки в множество правил вместо того, чтобы поправить одноединственное правило, ответственное за внешний вид. Лучше использовать больше правил, которые отвечают за что-то одно (разделение ответственности).

```
.card {
  position: absolute;
  top: 20px;
  left: 20px;
  background-color: red;
  font-size: 1.5em;
  text-transform: uppercase;
}
```

CSS определяет, как выглядит ваш компонент, а HTML применяет этот вид к элементам на странице. Чем меньше CSS «знает» про структуру HTML, тем лучше.

Если проект планируется «одноразовым», например, для мероприятия, которое пройдёт и больше не повторится, то не так важно, как вы сделаете его сайт. Но если планируется развитие, расширение и поддержка, то лучше подстелить соломки, и сделать проект так, чтобы не получить кучу проблем из-за плохой архитектуры.





Практикум
Тренажёры
Подписка
Для команд и компаний
Учебник по РНР

Профессии
Фронтенд-разработчик
React-разработчик
Фулстек-разработчик

Бэкенд-разработчик

УслугиРабота наставником
Для учителей
Стать автором

КурсыHTML и CSS. Профессиональная вёрстка сайтов
HTML и CSS. Адаптивная вёрстка и автоматизация

JavaScript. Профессиональная разработка веб-интере

JavaScript. Профессиональная разработка веб-интерфейсов JavaScript. Архитектура клиентских приложений React. Разработка сложных клиентских приложений PHP. Профессиональная веб-разработка

РНР и Yii. Архитектура сложных веб-сервисов

Node.js. Разработка серверов приложений и API

Анимация для фронтендеров

Вёрстка email-рассылок

Анимация для фронтендеров
Вёрстка email-рассылок
Vue.js для опытных разработчиков
Регулярные выражения для фронтендеров
Шаблонизаторы HTML
Алгоритмы и структуры данных
Анатомия CSS-каскада

БлогС чего начать
Шпаргалки для разработчиков
Отчеты о курсах

Информация
Об Академии
О центре карьеры
Остальное

Написать нам Мероприятия Форум