**Архитектура и организация CSS-кода**

**11.1. Зачем нужна архитектура в CSS?**

Когда сайт растёт — вместе с ним растёт и CSS: десятки компонентов, сотни классов, пересекающиеся стили, конфликтующие селекторы…

**Без архитектуры:**

* Возникают конфликты и дублирование.
* Стили становятся неуправляемыми.
* Любое изменение ломает сайт.

**Архитектура CSS** — это способ писать читаемый, масштабируемый, модульный код, с которым удобно работать в команде и который легко развивать.

**11.2. Принципы хорошей CSS-архитектуры**

* **Модульность**: стили каждого компонента живут изолированно.
* **Предсказуемость**: изменения в одном месте не ломают другое.
* **Повторное использование**: классы можно применять повторно.
* **Поддерживаемость**: легко находить, читать и изменять код.

**11.3. Структура проекта: организация файлов**

Пример организации CSS-файлов:

arduino

КопироватьРедактировать

/css

├── base.css (сброс, базовые стили)

├── layout.css (контейнеры, сетка, структура)

├── components.css (карточки, кнопки, формы)

├── utilities.css (мелкие утилиты: .text-center, .hidden)

└── main.css (импорт всех остальных)

Или с использованием SCSS:

bash

КопироватьРедактировать

/scss

├── \_variables.scss

├── \_mixins.scss

├── \_reset.scss

├── \_layout.scss

├── \_components/

│ ├── \_header.scss

│ └── \_card.scss

└── main.scss

**11.4. Методологии CSS: BEM, OOCSS, SMACSS**

**BEM (Block Element Modifier)**

Самая популярная методология для модульной вёрстки.

**Синтаксис:**

lua

КопироватьРедактировать

.block {}

.block\_\_element {}

.block--modifier {}

**Пример:**

html

КопироватьРедактировать

<div class="card card--highlighted">

<h2 class="card\_\_title">Заголовок</h2>

<p class="card\_\_text">Описание</p>

</div>

Плюсы:

* Читабельность
* Минимум каскада
* Лёгкая модификация

**OOCSS (Object-Oriented CSS)**

Принцип "поведения важнее структуры".

* Разделение структуры и оформления.
* Повторное использование "объектов" (например, .btn, .media, .box).

**SMACSS (Scalable and Modular Architecture)**

Стили делятся на категории:

* Base
* Layout
* Module
* State
* Theme

SMACSS — философия гибкой организации, часто используется в крупных проектах.

**11.5. Утилитарные классы (Utility-first)**

Подход, при котором каждый класс — это одна конкретная функция.

Пример:

html

КопироватьРедактировать

<div class="text-center bg-light p-3 rounded">...</div>

Этот подход реализован в **Tailwind CSS**. Позволяет писать быстро и избегать каскада.

**11.6. Как избегать каскадных конфликтов**

* Используй **только классы** (избегай tag, id, .class tag).
* Сохраняй **глубину селекторов ≤ 2** уровней.
* Никогда не стилизуй через !important, кроме случаев с утилитами или reset.
* Избегай глобальных переопределений: не пиши .btn { ... } если .card\_\_btn { ... } логичнее.

**11.7. CSS-переменные и их архитектура**

Используй root-переменные для цветов, шрифтов, размеров и анимаций:

css

КопироватьРедактировать

:root {

--primary-color: #337ab7;

--border-radius: 8px;

--font-size-base: 16px;

}

И применяй их по всему проекту:

css

КопироватьРедактировать

.button {

background: var(--primary-color);

font-size: var(--font-size-base);

}

**11.8. Линтеры, автопрефиксы и автоматизация**

* **Autoprefixer** — добавляет вендорные префиксы.
* **Stylelint** — анализирует и находит ошибки в CSS.
* **Prettier** — форматирует код.
* **SCSS/Less/PostCSS** — позволяют использовать переменные, вложенность, миксины.

**11.9. Частые ошибки и анти-паттерны архитектуры CSS**

* Глобальные переопределения (.container, .btn, h1)
* Использование ID для стилизации
* Глубокие селекторы .header .menu ul li a
* Отсутствие структуры
* Копипаст и дублирование стилей

**11.10. Глоссарий архитектуры CSS**

| **Термин** | **Определение** |
| --- | --- |
| BEM | Методология именования классов |
| SMACSS | Гибкая архитектура организации CSS |
| OOCSS | Объектно-ориентированный подход к CSS |
| Utility-first | Утилитарный подход (один класс = одна функция) |
| SCSS/Less | Препроцессоры CSS |
| Root-переменные | Переменные на уровне :root для глобальных значений |

**11.11. Вопросы для самопроверки**

1. Что такое BEM и зачем он нужен?
2. Как избежать каскадных конфликтов в CSS?
3. Чем отличается utility-first подход от классической архитектуры?
4. Какие плюсы у SCSS или CSS-переменных?
5. Как правильно структурировать стили по файлам?