Angular and SASS

Препроцессор стилей SASS

Использование **препроцессоров стилей** позволяет разработчику более удобно и эффективно создавать стилизацию приложения.

Создание стилей теперь частично действительно напоминает язык программирования.

Почему «нужно брать»:

- Возможность объявления переменных
- Наследование
- Модульность
- Математические операции
- Типы данных
- Структуры для работы с данными
- Синтаксическая чистота и сахар

Подключение SASS

Для начала роботы с **SASS**, не нужно устанавливать дополнительные зависимости. Возможность использования препроцессоров стилей **уже включена** в приложение которое генерирует **Angular CLI**.

Но поскольку возникает необходимость компиляции **scss** вне проектов **Angular**, наилучшим решением будет установка этого модуля глобально.

- **npm install -g node-sass**

Использование с консоли:

- Компиляция**: node-sass scss/your-style-names.scss css/your-style-names**
- Watcher**: node-sass -w scss/your-style-names.scss css/your-style-names.css**

Подключение SASS

Далее необходимо внести несколько изменений, для того чтобы **angular-cli** понимал что мы работаем с **SCSS**.

Используйте команду:

- **ng set defaults.styleExt scss**

Или мануально измените раздел **defaults/stylesExt** в файле **angular-cli.json**

```
"defaults": {
  "styleExt": "scss",
  "component": {
  }
}
```

В этом же файле, в параметре «**styles**» измените расширение **styles.css** на **styles.scss**

Измените все расширения файлов стилей на **.scss**

Для генерации нового проекта с scss включенным по умолчанию, используйте команду: **ng new <project-name> --styles=scss**

Типы данных

SASS поддерживает восемь типов данных:

- числа (1.2, 13, 10px)
- строки ("foo", 'bar', baz)
- цвета(blue, #04a3f9, rgba(255, 0, 0, 0.5))
- boolean (true, false)
- nulls
- массивы (листы) (e.g. 1.5em 1em 0 2em, Helvetica, Arial, sans-serif)
- мапы (ключ-значение)
- функциональные ссылки

Переменные

Одно из огромных преимущества использование **SASS** – это возможность создавать переменные.

Для объявления переменной используется специальный знак $ (sigil).

```
$template-dark-color: #041474;
$template-main-color: rgba(213, 255, 64, 0.79);

$template-full-height: 100vh;
$template-full-width: 900px;

$template-main-font-size: 12px;
$template-font-width: 400;


body {
  background: $template-dark-color;
  font-size: $template-main-font-size;
  margin-left: 20px;
  margin-top: 100px;
}
```

Вложенные классы

Использование препроцессора позволяет создавать вложенные классы и элементы.

SCSS

```scss
body {
  font-size: $template-main-font-size;
  margin-left: 20px;
  margin-top: 20px;

  .custom-parent {
    display: flex;
    flex-direction: column;
    justify-content: center;
    align-items: center;
    background: $template-main-color;
    height: 500px;
    width: 500px;
    font-size: $template-main-font-size;

    .inner-example {
      display: flex;
      flex-direction: column;
      justify-content: center;
      align-items: center;
      background: green;
      width: 400px;
      height: 400px;
    }
  }
}
```

CSS

```css
body {
  font-size: 25px;
  margin-left: 20px;
  margin-top: 20px;
}

body .custom-parent {
  display: flex;
  flex-direction: column;
  justify-content: center;
  align-items: center;
  background: rgba(213, 255, 64, 0.79);
  height: 500px;
  width: 500px;
  font-size: 25px;
}

body .custom-parent .inner-example {
  display: flex;
  flex-direction: column;
  justify-content: center;
  align-items: center;
  background: green;
  width: 400px;
  height: 400px;
}
```

Mixins

Миксины позволяют создавать блоки кода, который можно использовать повторно при необходимости.

SCSS

```scss
@mixin flex-column-all-center {
  display: flex;
  flex-direction: column;
  justify-content: center;
  align-items: center;
}

body {
  font-size: $template-main-font-size;
  margin-left: 20px;
  margin-top: 20px;

  .custom-parent {
    @include flex-column-all-center;

    background: $template-main-color;
    height: 500px;
    width: 500px;
    font-size: $template-main-font-size;

    .inner-example {
      @include flex-column-all-center;

      background: green;
      width: 400px;
      height: 400px;
    }
  }
}
```

CSS

```css
body {
  font-size: 25px;
  margin-left: 20px;
  margin-top: 20px;
}

body .custom-parent {
  display: flex;
  flex-direction: column;
  justify-content: center;
  align-items: center;
  background: rgba(213, 255, 64, 0.79);
  height: 500px;
  width: 500px;
  font-size: 25px;
}

body .custom-parent .inner-example {
  display: flex;
  flex-direction: column;
  justify-content: center;
  align-items: center;
  background: green;
  width: 400px;
  height: 400px;
}
```

Передача параметров в mixin

Миксин так же представляет собой подобие функции, которая может выполнять разные операции и применять соотвествующий стиль

SCSS

```scss
@mixin flex-column-all-center {
  display: flex;
  flex-direction: column;
  justify-content: center;
  align-items: center;
}

@mixin if-example($local-var) {
  @if $local-var {
    @include flex-column-all-center;
  } @else {
    display: block;
  }
}


@mixin padding($values...) {
  @each $var in $values {
    padding: #{$var};
  }
}
```

SCSS

```scss
body {

  a {
    @include padding(2px 4px 6px);
  }

  @include if-example(true);

  font-size: $template-main-font-size;
  margin-left: 20px;
  margin-top: 20px;

  .custom-parent {
    @include flex-column-all-center;

    background: $template-main-color;
    height: 500px;
    width: 500px;
    font-size: $template-main-font-size;

    .inner-example {
      @include flex-column-all-center;

      background: green;
      width: 400px;
      height: 400px;
    }
  }
}
```

CSS

```css
body {
  display: flex;
  flex-direction: column;
  justify-content: center;
  align-items: center;
  font-size: 25px;
  margin-left: 20px;
  margin-top: 20px;
}

body a {
  padding: 2px 4px 6px;
}

body .custom-parent {
  display: flex;
  flex-direction: column;
  justify-content: center;
  align-items: center;
  background: rgba(213, 255, 64, 0.79);
  height: 500px;
  width: 500px;
  font-size: 25px;
}

body .custom-parent .inner-example {
  display: flex;
  flex-direction: column;
  justify-content: center;
  align-items: center;
  background: green;
  width: 400px;
  height: 400px;
```

Наследование

Наследование – дополнительная возможность выделять общие стили и повторно их переиспользовать.

SCSS

```scss
.row {
  width: 50px;
}
.new-row {
  @extend .row;
}
.another-row {
  @extend .row;
}
```

CSS

```css
.row, .new-row, .another-row {
  width: 50px;
}
```

Важно понимать, что наследование создает **взаимоотношение**, а миксин является **блоком кода**!

SCSS

```scss
@mixin row {
  width: 50px;
}
.new-row {
  @include row;
}
.another-row {
  @include row;
}
```

CSS

```css
.new-row {
  width: 50px;
}

.another-row {
  width: 50px;
}
```

Media query

Media query – создают определенные правила отображения по заданым параметрам.

Задача: реализовать отображение шаблона А при размере экрана больше 1200px и реализовать отображение шаблона Б при размере экрана меньше 1200 px.

```
@media only screen and (min-width: 1200px) {
  .mobile-layout {
    display: none;
  }

  .desktop-layout {
    display: block;
  }
}

@media only screen and (max-width: 1199px) {
  .mobile-layout {
    display: block;
    text-align: -webkit-center;
  }

  .desktop-layout {
    display: none;
  }
}
```

```
<!DOCTYPE html>
<html lang="en">
<head...>
<body>

<div class="desktop-layout"...>

<div class="mobile-layout"...>

<div class="reduced-gutter"></div>
</body>
</html>
```

Импорт стилей

Иногда возникает необходимость импортировать содержимое одного стиля в другой и эта необходимость может возникать несколько раз.

- **@import "foo.css"** - прямое подключение стиля (допустимо расширение .scss)
- **@import "http://foo.com/bar";** - импорт стиля с внешнего источника
- **@import url(foo)** – если имя файла, это url;

Multiply: @import "first-style", "second-style";

Импорт стилей – одна из возможностей, которая частично позволяет обойти инкапсуляцию.

Важно понимать, что после компиляции, мы получаем **один файл** стилей, который по сути склеен с остальных частей

Подключение bootstrap через Angular

Установите bootstrap через npm:

- **npm install bootstrap --save**

Установите библиотеку tether, необходимую для bootstrap:

- **npm install tether --save**

Внесите изменения в файл **angular-cli.json** в раздел **styles** и **script:**

```
"styles": [
  "styles.scss",
  "../node_modules/font-awesome/css/font-awesome.min.css",
  "../node_modules/primeng/resources/primeng.min.css",
  "../node_modules/primeng/resources/themes/omega/theme.css",
  "../node_modules/bootstrap/dist/css/bootstrap.min.css"

],
"scripts": [
  "../node_modules/jquery/dist/jquery.min.js",
  "../node_modules/tether/dist/js/tether.min.js",
  "../node_modules/bootstrap/dist/js/bootstrap.min.js"
],
```

Порядок скриптов **важен**!

Задание

1. Создайте новый проект, который изначально настроен на работу с **scss**.

2. Создайте компонент (регистрации или обратной связи)

3. Создайте базовую верстку (только грид и необходимые контейнера, используя Bootstrap)

4. Создайте необходимую на данный момент стилизацию, используя **scss**