Xaviju Add install instruction ebbfa01 10 days ago

1 contributor

168 lines (131 sloc) 5.37 KB Raw Blame History

Advanced Sass

To test this part you should install a Sass compiler:

- Download and install KoalaApp
- Create a new folder named `04_advanced_ui` in your classes folder
- Create a file named `main.scss` .
- Copy and paste the code from sassmeister you just created.
- Open `Koala` and on the `+` symbol. Search the folder we just created and add it.
- You should see your `main.scss` file. Select it.
- Click on `compile` button.

This process should create two files: `main.css` and `main.map.css` . Forget about the last one for now, its just a sourcemap

Open you `main.css` . It should be the CSS compiled as Sassmeister did. Now you are able to compile your own files. Remember that you should modify only on the `main.scss` but your HTML should include the automatically generated `main.css` file.

The process is the following:

1. Edit `main.scss`
2. Compile using koala.
3. Link the generated `main.css` to your html `head`
4. Confirm that the CSS works.
5. Modify and compile again `main.scss`

Partials

CSS files on large applications can become difficult to read because they can become reaaaaaally long. Sass allows to separate the files into small chunks of code that can be imported into other files to reduce the size. Files can be separated by responsibility.

Architecting a CSS project is probably one of the most difficult things you will have to do in a project's life. Keeping the architecture consistent and meaningful is even harder.

For example, a folder structure according to Sass Guildelines

```
styles/
|
|- abstracts/
|   |- _variables.scss    # Sass Variables
|   |- _mixins.scss       # Sass Mixins
|
|- base/
|   |- _reset.scss        # Reset/normalize
|   |- _typography.scss   # Typography rules
|   …                     # Etc.
|
|- components/
|   |- _buttons.scss      # Buttons
|   |- _carousel.scss     # Carousel
|   |- _cover.scss        # Cover
|   |- _dropdown.scss     # Dropdown
|   …                     # Etc.
|
|- layout/
|   |- _navigation.scss   # Navigation
|   |- _header.scss       # Header
|   |- _footer.scss       # Footer
|   |- _sidebar.scss      # Sidebar
|   |- _forms.scss        # Forms
|   …                     # Etc.
|
|- pages/
|   |- _home.scss         # Home specific styles
|   |- _contact.scss      # Contact specific styles
|   …                     # Etc.
|
|
`- main.scss              # Main Sass file
```

Import

From the main file, we can import the chunks of CSS into a single file that wil lbe compiled into a CSS file. Magic!

The `main.scss` file could look somethink like this:

```
@import 'abstracts/variables';
@import 'abstracts/mixins';

@import 'base/reset';
@import 'base/typography';

@import 'layout/navigation';
@import 'layout/grid';
@import 'layout/header';
@import 'layout/footer';
@import 'layout/sidebar';
@import 'layout/forms';

@import 'components/buttons';
@import 'components/carousel';
@import 'components/cover';
@import 'components/dropdown';

@import 'pages/home';
@import 'pages/contact';

@import 'themes/theme';
@import 'themes/admin';
```

Rules for import

Notice that files that ar chunks that will be imported are preceded by the simbol `_` as in `_variables.scss` but, when you import them, you should omit the underscore symbol. This symbol is telling sass that this file is just a part of a larger file and will not try to compile it.

Remember:

- one file per `@import` ;
- one `@import` per line;
- no new line between two imports from the same folder;
- a new line after the last import from a folder;
- file extensions and leading underscores omitted.

Order of imports

And remember, in CSS the order of the CSS is important. If two rules have the same name, rules will be merged and the last one will overwrite the previous one. be careful with your project naming.

Mixins

Mixins are one of the most used features from the whole Sass language. They are the key to reusability and DRY components. And for good reason: mixins allow authors to define styles that can be reused throughout the stylesheet.

They can contain full CSS rules and pretty much everything that is allowed anywhere in a Sass document. They can even take arguments, just like functions. Needless to say, the possibilities are endless.

This is how a mixin looks like

```
// Define the mixin
// This mixin is named `size` and receives two parameters: width and height
// If the height parameter its not set, it will be equal to width
// This mixin returns a piece of CSS with width and height.
@mixin size($width, $height: $width) {
  width: $width;
  height: $height;
}

// And use it anywhere
.box {
    @include size(1rem);
}

// Equals
/*
.box {
    width: 1rem;
    height: 1rem;
*/
```

Argument-less mixins

A mixin might not receive arguments.

```
@mixin large-font {
    font-family: 'Roboto', Arial;
    font-weight: 600;
    font-size: 2rem;
}
```

```
.selector {
    @include large-font;
}
```

Hands on

Organize your code as shown in the file structure. Divide your CSS into small chunks and import them from a single CSS main file. Remember the importance of the order. Add mixins to your project and avoid repeating code. Remeber that mixins have a specific file.

Bonus points

A proffesional guide to read with all the information about Sass: https://sass-guidelin.es