Modules and Components

Modules and Components are the reusable elements of the design that can be used to construct new pages. Part of planning the development of the site should be to figure out what parts of the design are going to be part of the template, what parts are components and what parts are modules. This will depend on the requirements of the design, and what level of customisation is required.

What's the difference?

Modules and Components are very similar, they both contain the Twig template, JS files, SCSS files required for them to work. In that sense, they are self-contained which makes it easy to transfer them from one site to another. The difference is that Modules rely on the ACF flexible content field, and Components do not. Modules are used within the page's module loop, and components are generally used once on a page. An example of a component is `Header`, `Footer`, `Sidebar`, `HomeBanner`. An example of a module is `TextImage`, `CallToAction`, `ProductGrid`.

Components

In the JuiceBox WordPress Boilerplate, components are defined in the `src/JuiceBox/Components` directory. This directory is scanned by Twig, which allows you to easily include components in your templates. For example, the header can be included use the following syntax `{% include 'Header/template.twig' %}`.

To ensure that the component's styling is also included, you must manually include the component's SCSS file in the `scss/main.scss` file. If the component includes any JavaScript code, you must also import the corresponding JavaScript file into the `js/main.js` file.

By following these steps, you can ensure that your components are properly styled and functional within your WordPress theme.

Modules

In the JuiceBox WordPress Boilerplate, modules are defined in the `src/JuiceBox/Modules` directory. These modules are rendered in the module loop, which is by default included within the `afterMain` Twig block:

```
{% block afterMain %}
    {% include 'partials/modules.twig' %}
{% endblock %}
```

To ensure that the module's styling is applied correctly, you must manually include the corresponding SCSS file in the `scss/main.scss` file. If the module also includes any JavaScript code, you need to import the corresponding JavaScript file into the `js/main.js` file.

By following these steps, you can ensure that your modules are properly styled and functional within your WordPress theme.

Module Fields

A special ACF field group, called Modules, serves as an ACF Flexible Content field, where you can create modules. Each module is considered a "Layout" within the ACF flexible content field. Once you save the field configuration for the modules, the module folder within `src/JuiceBox/Modules` is automatically created by duplicating `src/JuiceBox/Modules/__template` and naming the folder in line with the convention used.

It also creates a `fields.json` file within the module, which contains the field configuration for that module. This approach has the advantage that all field configurations are committed to source control, eliminating the need for syncing between environments.

It is crucial to bear in mind that any alterations made to ACF fields on staging or production environments through the WordPress admin backend will be overwritten when the code is redeployed from the repository. Therefore, all modifications to ACF fields must be committed to the repository to ensure that the changes are propagated correctly across all environments.

Naming Modules

When creating page modules in a WordPress website, it is recommended to use generic names for the modules instead of specific names. This helps ensure that the module can be used in other contexts without requiring renaming or customization, which can save development time and reduce the likelihood of errors or issues arising from naming inconsistencies.

For example, if you are creating a module to display a grid of products, you might name the module "ProductGrid" rather than "PieGrid" to indicate its generic purpose. Similarly, if you are creating a module to display text and images side-by-side, you might name it "ImageText" instead of "PieIngredients" to keep the module more flexible and reusable.

By following a consistent naming convention for modules, it becomes easier to locate and use the appropriate module for a given purpose, and it helps ensure that the website remains organized and maintainable over time.