Janelle Wong

Follow 23 Followers About

Atomic Design Pattern: How to structure your React application

Design Patterns

Janelle Wong Dec 11, 2017 · 3 min read

As we build out scalable applications in React, we often face challenges in maintaining the growing complexity of component structures. What I want you to take away from this post is to understand why implementing an archetypal design pattern — *Atomic Design*, is crucial for the readability, scalability and flexibility of your application code. The Atomic design pattern has proved to be remarkably suited for the componentised nature of React.

Atomic Design

Atomic design, developed by Brad Frost and Dave Olsen, is a methodology for crafting design systems with five fundamental building blocks, which, when combined, promote consistency, modularity, and scalability. In this post, we're going to explore how these principles are a natural fit for building interfaces in React, and how we can extend the Atomic metaphor in useful ways such that we can map out components which have a dynamic lifecycle inside of an abstract ecosystem.

. . .

Atomic Development

The five distinct levels of atomic design — atoms > molecules > organisms > templates > pages — map incredibly well to React's component-based architecture.

Atoms:

Basic building blocks of matter, such as a button, input or a form label. They're not useful on their own.

Molecules:

Grouping atoms together, such as combining a button, input and form label to build functionality.

Organisms:

Combining molecules together to form organisms that make up a distinct section of an interface (i.e. navigation bar)

Templates:

Consisting mostly of groups of organisms to form a page — where clients can see a final design in place.

Pages:

An ecosystem that views different template renders. We can create multiple ecosystems into a single environment — the application.

. . .

File Structure

Since React follows a component-based architecture, it's pretty common to organise your components based on the type, rather than feature. What if we built a sub-ecosystem for each component feature?

Each component or service has its own isolated environment — everything needed to work on its own instance. You can see that each component ***/Buttons*** & ***/Form*** has its own set of styles, actions, and unit or integration tests that act like an independent piece of feature in your app. (*You can also add its own set of images and other local variables.)*

This makes it much easier, and reduces your efforts, to test your code consistently and effectively.

This type of organisation allows for nesting components into another component. Note that if you define a new component inside */Delete, /Submit, /Login, or /Register*, the nested component can only be used by its direct parent, and not its cousins.

. . .

Why would you want to do that?

The main purpose of following the atomic design pattern when organising a React file structure is to isolate the environments of each feature component. When side-effects are isolated, code becomes a lot more readable and modular. A single instance of a feature will make testing more straightforward, thus improving the overall quality assurance of your application. As the complexity of your application and state management begin to grow, organising your file structure in this pattern will help you easily determine and handle state.

. . .

Below are some great resources that I've found to be extremely insightful:

Atomic Design Methodology | Atomic Design by Brad Frost

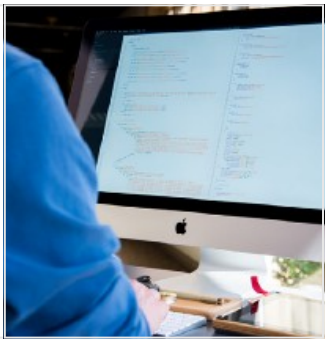
Learn how to create and maintain digital design systems, allowing your team to roll out higher quality, more consistent...

atomicdesign.bradfrost.com

Atomic Design - Your Ultimate Guide to Scalable & Modular CSS (Sass) - Alex Devero Blog

Creating scalable and maintainable CSS is hard. On the other hand, it is very easy to create bloated stylesheets nobody...

blog.alexdevero.com



A Better File Structure For React/Redux Applications

Most of the examples I could find about React/Redux applications (either client side or universal) are very simple...

marmelab.com

React File Management Design Patterns Atomic Design JavaScript