My name is Cory Rylan. Google Developer Expert and Front End Developer for VMware Clarity. Angular Boot Camp instructor. I specialize in creating fast progressive web applications.

Follow @coryrylan

Angular Design Patterns: Feature and Presentation Components

Cory Rylan
Jul 13, 2017
Updated Feb 18, 2018 - 4 min read

> This article has been updated to the latest version of Angular 8 and tested with Angular 7. The content is likely be applicable for older Angular 2 or other previous versions.

In this post, we will cover the Feature and Presentation Component Design pattern. This pattern has been called many things such as

- Smart/Dumb Components
- Stateful/Stateless Components
- Container Components

just like to give my perspective on this useful pattern.

The Component

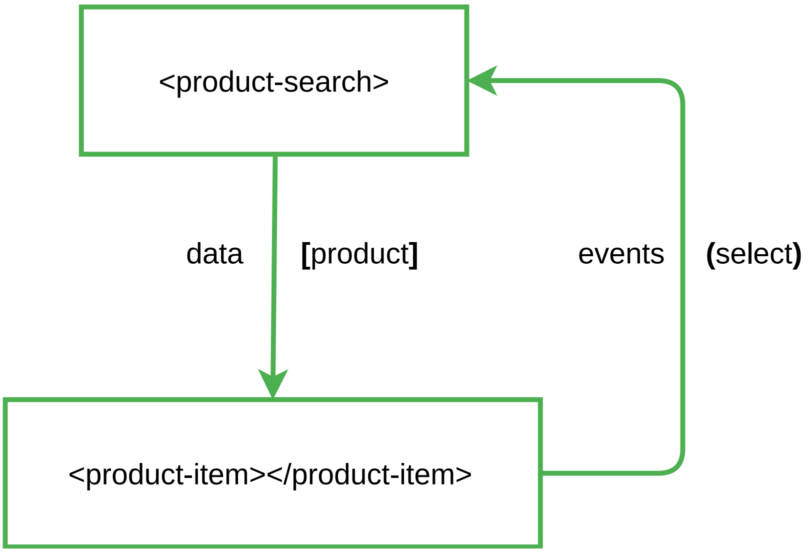
What is a Feature or Presentation Component? First let's define what a component is. A component is simply a small encapsulated piece of a user interface. Many frameworks and libraries use components including Angular, React, Ember, and Vue. Components allow us to break a large complex UI into smaller easier to understand pieces. This is similar to the way we break our code up into smaller functions. This makes our UI easier to understand and reuse in our applications.

When building large UI systems with components our application it becomes a tree-like structure. Components get used inside of other components. Most of the time these parent-child relationships need to communicate back and forth. In Angular components, we communicate via Input properties and Output events. Parent components can set a property of a child component, and a child component can emit events up to parent components.

This makes it clear where data is flowing from, but it brings up a few questions. Who should fetch data from the server? Which component is responsible for saving the data? This is where Feature and Presentation Components come in.

Feature vs. Presentation

In our web applications, we can take advantage of the Feature and Presentation Component pattern. Let's look at the example app below.



Here we have a product search page. The user can search for products with filters. The search view shows a list of product items.

Let's define the Feature component first. A Feature component is a top-level component that contains all other components in our feature. This commonly is a routed component in Angular. Our feature components are responsible for gathering data from various services for our feature to use. If our user saves data, the feature component is responsible for passing that data to our Angular Services to save the data to our server API. Feature components are still very slim with the amount of application logic. We try to defer this logic to Services if possible. For this example, the `product-search` is our Feature Component and will be composed of many Presentation components.

Where do Presentation Components fit in? As our Search Feature component grows in complexity, we can refactor our component into smaller reusable Presentation Components. Presentation Components behave like pure functions taking in the data via `@Input` and emitting data via `@Output`. This allows the majority of our UI not to know the underlying implementation detail of where the data came from. For example, a `product-item` component takes in a `@Input` of a product to display. This allows the `product-item` component to have the only responsibility of rendering the product when the data is passed to it.

```
<app-product-item
  [item]="itemData"
  (select)="selectProduct($event)"></app-product-item>
```

Our `product-item` could also have a single `@Output` to notify the feature component when a user clicks a product in the list. This is useful because we can now use the item component for different situations whether that should display a different view or save some information to our API.

Presentation components do not have to be reusable. Keeping the pure Inputs and Outputs makes it easier to debug features. Because our Feature Component is the only component communicating with Services it is easy to determine where the data is coming from and in turn, making it easier to debug.

a time, we will have to pass up to each parent component manually. Sometimes introducing other sub feature or container components can help elevate this. At this point, it brings into question how to manage state which can be for another blog post :)

Presentation Components do take more time to create and hook up than just having large single Feature Components but in the long term can help the maintainability and reusability of our components. Many if not most Presentation Components can be abstracted into a style guide or UI library for the project. To get ideas of component design and style guide maintainability I recommend Brad Frost's fantastic book Atomic Design.

♡ Recommend 6 Tweet Share Sort by Best

Start the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

Be the first to comment.

ALSO ON CORYRYLAN.COM

Keeping your Angular CLI project up to date

3 comments • 2 years ago

Avatar Sébastien Côté — You can also look at this repository : https://github.com/cexbraya...It indicates you which files have been updated

2017 Review

1 comment • 2 years ago

Avatar Maye Edwin — This is awesome

Creating a Dynamic Checkbox List in Angular

41 comments • a year ago

Avatar Hặršh Ḿặĩŝhẹrĩ — How Do i add up the selected checkboxes value?
Like in your example the value is ***[100, 300]***

Using Web Components in React

4 comments • 6 months ago

Avatar Cory Rylan — Updated post, thanks!

Subscribe to get the latest dev posts and content!

email address

SUBSCRIBE FOR FREE