Generalized Featured Product Page

Final Report

By: Andrew Pirondini

6 - 8 - 2018

Contents

1. Introduction

The goal of my senior project was to generalize the process of creating featured product pages with iFixit. Featured product pages are pages with custom content describing an individual product such as text blurbs, images, and even some responsive components. These do not have a simple template like products you might see on Amazon but instead involve scrolling through a variable amount of content.

In order to generalize these, each part of the page was broken into discrete components that contain custom content. There are components for images, paragraph blurbs, lists, footers and more. These components are implemented with React and can have their custom content passed in to the individual components. To make the page then involves rendering a series of components in order with whatever custom content desired for that product.

This project solves the problem of how to create featured product pages quicker by providing a cleaner and more straightforward way for programmers to create new ones. With less hard coding, programmers can rely on previous examples more easily and change only parts of the page that are required to be custom. Also, the components will be implemented in a way that is more general and will require less style tweaking. This final process should allow for many pages to be created with minimal code duplication and minimal time spent programming.

2. Background

2.1 Research

The fundamental question I needed to answer before starting to work on this project was how we were going to break the page up into components. While breaking things up into classes in the PHP backend was pretty straightforward, we wanted a better frontend library than vanilla HTML, CSS, and JavaScript. The two main technologies we initially considered on this frontend were Angular, maintained by Google, and React, maintained by Facebook. During the research process we uncovered there was a third possibility, Vue which we also considered when making our decision.

Although it was hard to find much technical documentation highlighting the tradeoffs between these languages I was able to find several articles of people debating this very decision. In an article by Jens Neuhaus about this subject, “React vs Angular vs Vue: A 2017 comparison” he highlights the important information needed for people making this decision. A similar article by Dominick Tarnowski titled “Angular vs React — the DEAL BREAKER”, presents the same comparison but with much less depth.

One important concept that advocates for React use in defense for React’s methodology is the idea of “separation of concerns”. This idea is a first principle that states the reason why we maintain code in separate files or classes is in order to maintain a separation of concerns that allows us to think more clearly about each individual concern. Applying this to frontend, the division between HTML, CSS, and JavaScript does not meet this first principle because HTML tags are littered with classes that the JavaScript uses or the CSS uses and changing any of these will break the functionality or style of the element. Similarly, JavaScript is often used to apply CSS conditionally to elements. Thus the true separation of concerns should not be on the type of file (template, style, or functionality) but rather at the element level since all three apply to a single element. React is built around this mindset and provides the ability to do all three in one file.

Along with the better organization of code detailed above some other important aspects React possessed were its flexibility with outside JavaScript, its maturity when compared to Vue, and the better framework existing for testing with React. Due to all of these factors React seemed like a pretty clear cut choice for the frontend framework to use on this project.

2.2 Underlying Technologies

The main technology we ended up using for the generalized featured product page is the aforementioned React. React revolves around creating individual components that contain their own logic and templating in them. These components can then be exported for use inside of other React components. Components can take various number of named arguments called ‘props’ (short for properties) that can allow for customized use. One of the most important and difficult aspects of React is in its state management. When done correctly, state is only contained in top-level components and passed down through props to all children components. Working with this in our codebase involves rendering the top-level React component with all the data from our backend in PHP and making sure to update its state appropriately for children components. The specifics of how React works are all explained in their main page, reactjs.org, which also features various examples and tutorials.

On top of the base React framework, the generalized featured product page uses glamorous, a React library for adding CSS-like style to a React component. With this library the style, templating, and functionality of a component are all contained in the same file. Glamorous works by creating unique CSS class names for each set of rules defined. These classes are then added to the elements of the React component that they apply to. Props can be used to determine which rules to generate allowing for more dynamic rules than standard CSS. This method replaces such practices as adding CSS classes to an element through JavaScript events. Glamorous has its own landing page with documentation and tutorials located at glamorous.rocks.

The last major technology used in the project is the asset management tool -- webpack. Webpack handles the process of determining which assets (in this case just js files with React components in them) to send to the client for a given page load. This is done by specifying entry points for webpack to start with. Webpack will then follow the import statements recursively until all assets needed for that entry point are included. The assets generated from an entry point are bundled into an asset group that can be selectively added to pages that need them.

3. Process Planning

3.1 Understanding the Process

The implementation of the generalized featured product page came in several parts, the first of which was figuring out which parts of the page needed to be generalized. This assessment was difficult at first because there was only one featured product page, iFixit's Pro Tech Toolkit, and this was almost completely hard coded into template files.

To understand the process I was trying to improve of creating these pages, I first went through the process myself and created a second featured product page for iFixit's Manta Driver Kit. This allowed me to gain insight into the process I was trying to speed up, as well as providing more examples to use for generalization. During the process of making this page, I kept all the logic out of the templates and mirrored the existing page as much as possible. Places where I had to deviate were useful examples of things that wouldn't be as easy to generalize.

After completing this second Featured Product Page, we discussed what future pages were going to look like, and where the key bottlenecks for the process were. I found that the main bottlenecks revolved around the style of individual components since they had to be designed for all screen sizes and browser compatibility.

3.2 Generalizing the Page

While working through this second featured product page I discovered some things were incredibly similar to the original page. Text blurbs in the design were almost always identical, containing a heading of two possible sizes, and a more detailed paragraph. On the opposite end of the spectrum, the landing view for the page was dramatically different with a slideshow on the Pro Tech Toolkit's page, as opposed to an image with accompanying text and a video background for the Manta's page. These comparisons were useful in determining what parts of the page we wanted to generalize.

Another step I did while working on generalizing the page was to move the logic out of the template files and into the PHP responders. This cleaned up the templates quite a bit and allowed me to create an initial FeaturedProductPage base class. This class contained variables common to all featured products such as title, price, itemcode, and a custom url.

3.3 Breaking into Components

The last step for planning out the process involved breaking the page up into components. To do this I grouped the templates and style into component folders signifying which aspects of the page were going to be turned into React components. The templates in these folders were rendered in sequence in the main body of the concrete class implementing the FeaturedProductPage. After this organization was complete we had four clear components that would belong to all product pages and one that would apply to the Manta and some other future pages but not all of them. These are described in detail in the next section.

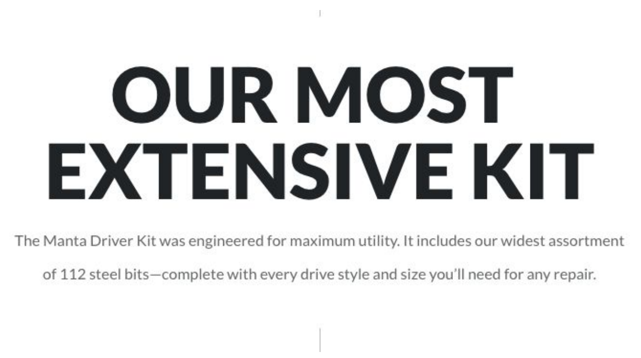
Creating these components involved taking some chunk of template code that was used on both pages, sometimes with multiple instances, and putting these into the render functions for a React component. Then, functionality was added through use of callbacks for buttons as well as conditionally rendering elements based on component props. Lastly style was added by using existing style rules and converting them to glamorous rules. Some additional conditions based on the props of the component were also added in the style rules to account for using the same component in what was previously separate places with different styles.

Rendering the React components involved adding a render call for the desired component inside of the template file that it was replacing. This call allows for variables computed in the PHP responder to be passed in as props to the individual component. Since most of the logic of the templates had been previously moved into PHP, it was relatively seamless to pass these variables into the React component.

Adding React components to an individual page was handled through use of webpack. The base FeaturedProductPage class added the asset group for the components that should be common to all featured product pages. The entry point for this asset group simply consists of a file that contains a series of imports for all the main components. Components that may not apply to all featured product pages are added in the specific classes that extend the FeaturedProductPage class. This prevents having multiple asset groups that are always included on the same page while also avoiding importing components that won't be used.

4. Components

4.1 Text

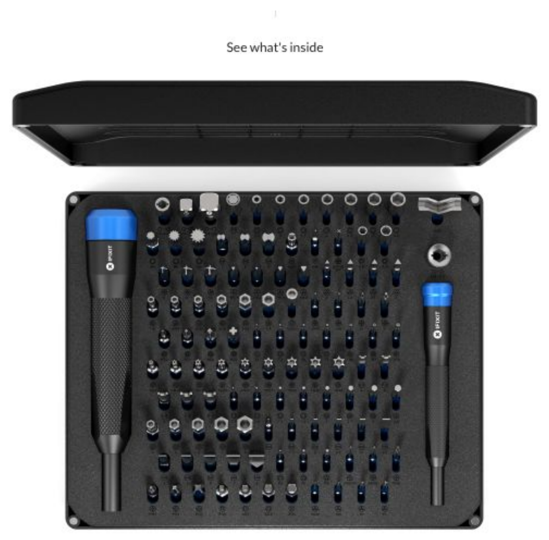


OUR MOST
EXTENSIVE KIT

The Manta Driver Kit was engineered for maximum utility. It includes our widest assortment of 112 steel bits—complete with every drive style and size you'll need for any repair.

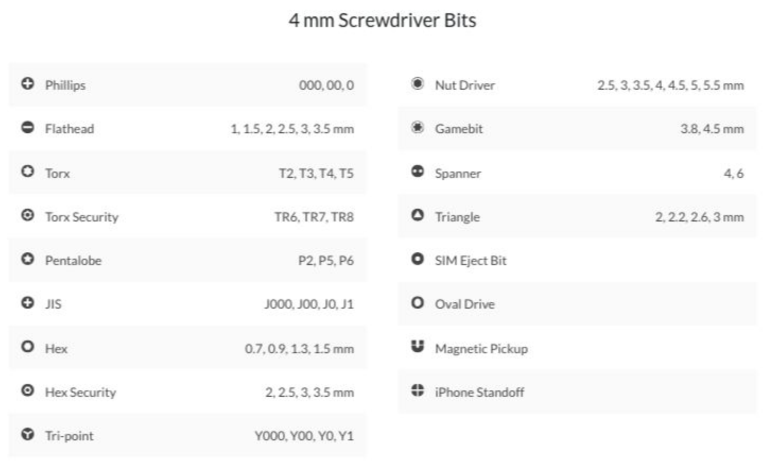
The first component created, the TextComponent was by far the simplest. The text takes as its properties heading, subheading, body, and max width. The heading and subheading are exclusive to each other (so one should always be null) and are separate props to determine which one of two existing styles to use. The body of the text is simply rendered below the heading with a few style tweaks. The size of the text shrinks on mobile screens so as to prevent headings from spanning many lines. The creation of this component, while simple, took some time because it was the first one and the surrounding asset management had to be figured out first.

4.2 Image



The second component created was the ImageComponent. This component takes as its properties heading, dimensions, src, srcset and a flag to determine whether or not to lazy load the image. Most images on these pages are lazy loaded due to size. The implementation of lazy loading used 'react-lazyload' and is an example of the ability React provides for adding functionality on top of the templating.

4.3 List

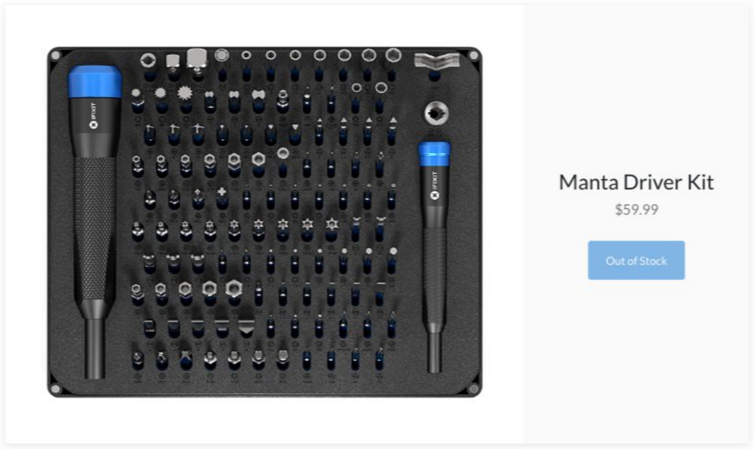


4 mm Screwdriver Bits

Phillips	000, 00, 0
Flathead	1, 1.5, 2, 2.5, 3, 3.5 mm
Torx	T2, T3, T4, T5
Torx Security	TR6, TR7, TR8
Pentalobe	P2, P5, P6
JIS	J000, J00, J0, J1
Hex	0.7, 0.9, 1.3, 1.5 mm
Hex Security	2, 2.5, 3, 3.5 mm
Tri-point	Y000, Y00, Y0, Y1
Nut Driver	2.5, 3, 3.5, 4, 4.5, 5, 5.5 mm
Gamebit	3.8, 4.5 mm
Spanner	4, 6
Triangle	2, 2.2, 2.6, 3 mm
SIM Eject Bit	
Oval Drive	
Magnetic Pickup	
iPhone Standoff	

The third component created was the ListComponent. This component takes as its properties simply a heading and a list of items. The items each include an optional icon, a term, and an optional description to allow the most flexibility. These are rendered to a description list html tag ,dl, and are set to have multiple columns through use of CSS column-count property. The heading of the ListComponent is created by rendering the TextComponent within this component using only the subheading. This was the first time we put a component inside another component and decided it was useful in preserving consistency of style.

4.4 AddToCart

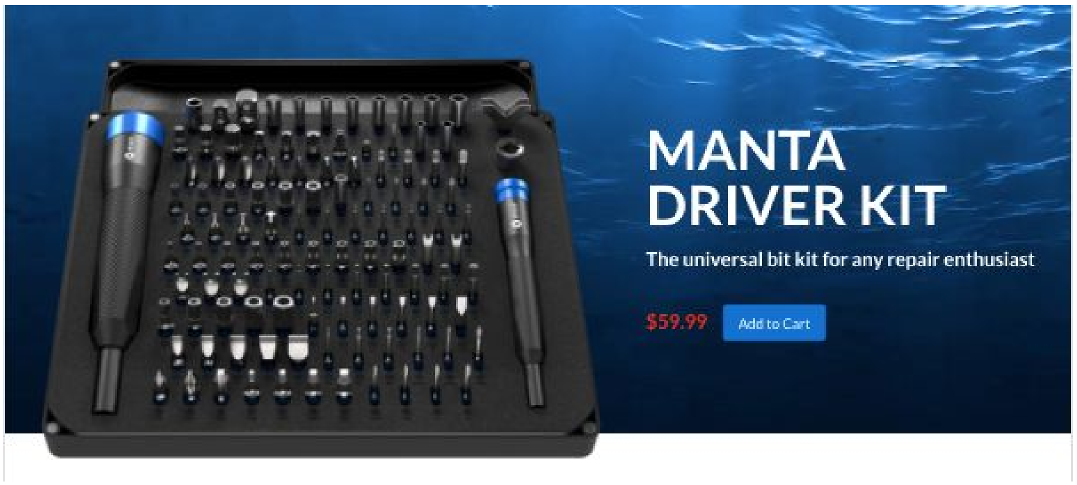


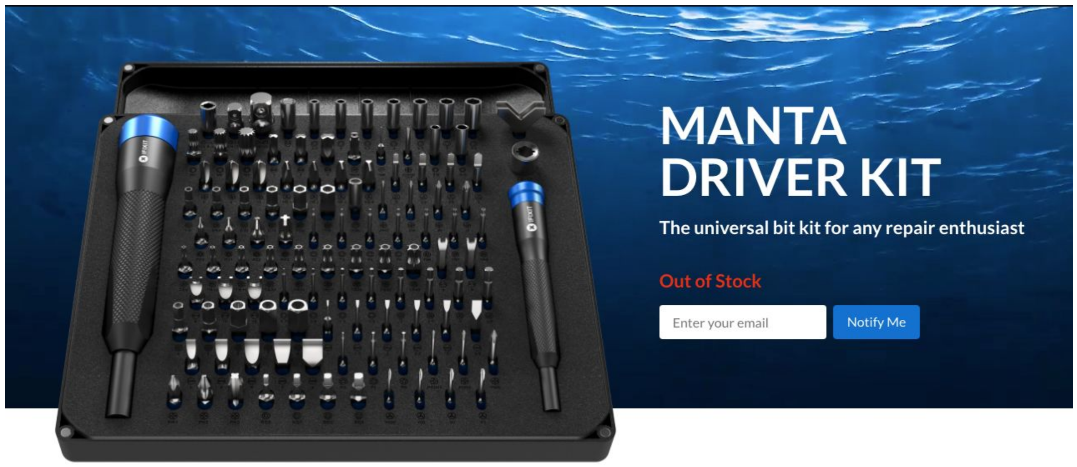
Manta Driver Kit

$59.99

The last of the general purpose components created was the AddToCartComponent. This component, originally just referred to as the footer, appears at the bottom of every featured product page. It has a large number of properties including product data such as the itemcode and price as well as the properties needed to construct the image. It is much more complex than the previous components and uses the ImageComponent inside of it to handle some of this logic. It also imports AddToCart functionality from a separate, non React javascript file. This proved to be one of the more challenging components but also showed the most promise in reducing future work for programmers using the component.

4.5 Hero





The last of the components created was the HeroComponent. The term "hero" has generally been thrown around at iFixit to refer to the flashy fullscreen view the user sees when they first load the page. For the HeroComponent, we took the hero we used for the Manta product and decided that we would want to use it on future product pages. This is the only component that is not also used in the Pro Tech Toolkit. This proved to be by far the most complex of all the components created. It supports three possible backgrounds, an image, color, or a video like the one shown above. On top of this it renders both an ImageComponent for the product and a TextComponent for the product name and tagline. It imports AddToCart functionality similar to the AddToCartComponent but goes even further. In the HeroComponent only, if the product is out of stock it will display a form for notifying the user when it is back in stock. This functionality was implemented in its own smaller React component NotifyProduct.

5. Evaluation

The initial goal with this project was to design an interface for creating featured product pages without the need for programmer intervention. This interface was intended to be used by technical writers so that featured product pages could be created much quicker than before. As the project progressed and the large scope of it was uncovered we decided to scrap the user interface and focus just on making the process of created featured product pages quicker but still requiring programming work.

In this regard we can compare the difficulty I had in implementing the second featured product page with what is left for programmers to do in future uses of my project. Creating the second featured product page involved a decent amount of copying and pasting of code, copying in the raw text and images, significant changes of style, and the creation of one new component in the hero of the page. This whole process took about a month of me working part time (about 16 hours a week) including the code review and QA process.

With the work I have done in componentizing the process this should be much shorter. A programmer needs to extend the base class by overriding a few variables such as productcode, name, and url. Then the programmer will need to take the raw text and images, as well as any other necessary assets and pass them into the appropriate React components. To create a page similar to the Manta Driver Kit's page this would involve the Hero component, a series of Text and Image components, and a finishing AddToCart component. Assuming some style tweaks may be necessary and some unfamiliarity with the process this should be accomplishable in a week or two with code review and QA. Unfortunately, we have not yet had time to go through this process to get accurate metrics and this is merely a thought experiment.

Another metric of success could be looked at in terms of code clarity. Previously, most of the code was hard coded constants and significant logic in template files. With React components, the code clarity is pretty clearly improved. Each component contains style, templating, and functionality in a way that is fairly straightforward for people that have had some experience with React. With multiple pages using the same components, the amount of code duplication is also reduced resulting in less lines of code overall.

Overall the project can be seen as a minor success. If the original goal of making a user interface and no programmer input was reached it would have been a remarkable success but this proved a bit too challenging for the time frame. As is, the project clearly improved code clarity and should reduce the time required to create future featured product pages.

6. Conclusions

Throughout this project I gained a lot of experience using React and learned the difficulties inherent in trying to generalize a frontend code. With the creation of the second featured product page I was able to see just how often the pages could differ from each other in code or style, even when they appear remarkably similar to the user.

Creating individual React components was a process that started slow but built upon itself pretty quickly. Having code review suggestions from other programmers that have used React before allowed me to reach an end result of an initial component that was very clear and efficient. This process took quite a while but then subsequent React components were able to use this example and were developed much quicker.

During this project I also made the mistake of starting on the user interface too early. I created a form for tech writers to use for modifying featured product pages but then realized the implementation of this form would be too difficult. This ended up being wasted time and created code that will most likely be scrapped and never used. It was hard to figure out where to start when I began on this project but now I understand that getting a better understanding of the problem by trying to generalize existing pages would have been a far better strategy.

There is a lot of room for future work on this project since its scope ended up being so large. One main improvement would be in storing props for each component in somewhere outside of code. This was not done because creating a DB schema to describe the props we were passing in would be a large undertaking. However, to make the system more robust and able to be changed without need for programming, this should be done. Along with this, the base featured product page could provide more default behavior with overrides available. For example, we expect all pages to contain an AddToCart component which only requires product level information available to the base class. Having default components on the page would allow for even quicker creation of pages in the future.

References

"Maintainable CSS with React." *Glamorous - React Component Styling Solved*, glamorous.rocks/.

Neuhaus, Jens. "Angular vs. React vs. Vue: A 2017 Comparison – Unicorn.supplies – Medium." *Medium*, Unicorn.supplies, 28 Aug. 2017.

"React." *ReactJS*, reactjs.org/.

Tarnowski, Dominik. "Angular vs React - the DEAL BREAKER – Hacker Noon." *Hacker Noon*, Hacker Noon, 17 Jan. 2017.