SHOPIFY

MODERN WORKFLOW

WE'RE GOING TO COVER

- EXISTING VS CUSTOM THEME
  - TOOLS
  - SLATE
  - THINK IN COMPONENTS
  - FOUNDATION 6
  - AUTOMATE EVERYTHING W/ GULP
  - PERFORMANCE OPTIMIZATION INTRO
  - HOW I APPROACH PROJECTS

EXISTING THEME

- Pros
  - Fully-functional code base
  - Potentially quicker, depending on the scope.
- Cons
  - It’s not your code, so it can be difficult to understand
  - Advanced features can be very hard to implement
  - Not modular (no access to theme developer’s build process)
  - Limited to Sass v3.2

CUSTOM THEME W/ A BUILD PROCESS

- Pros
  - Complete control
  - Can be as modular as you want
  - It’s [mostly] your code, so if you run into problems, you don’t have to learn how someone else thinks
  - Extremely organized
  - Faster `slate watch` save uploads
  - Modern version of Sass and all tools related, like Compass
- Cons
  - Hard to set up initially
  - Can’t easily hand off a build process upon completion

LOVE YOUR SCAFFOLD THEME

- Your scaffold theme is your baby. It should be in a repo and all commits should be clear. Someone should manage it. It might become your favorite topic
- frontendchecklist.io - is your compass. Complete as much of this as possible with your scaffold before starting your first project.
- The hardest parts - DO THESE ONCE and continuously improve them as needed in your scaffold theme.
  - Ajax cart - Study hard: Slate's product.js, variants.js, and Timber's ajax-cart
  - Variant selection
- When you start a new project, before coding any particular page, you want to adjust all your settings (Foundation) and perfect your styles page (base styles, standard elements, common modules, etc)
  - Let's walk through a mockup and strategize an approach

TOOLS I USE

- OS: Mac OS
- Code editor: Visual Studio Code
- Snippet Management: Snippets Lab (integrates with Alfred)
- Project Management: Trello
- Design: Sketch
- Git: Command line or Tower
- Code Diff: Kaleidoscope
- Build automation: Gulp (previously CodeKit)
- Time zones: The Clock
- Password management: 1Password (integrates with Alfred)
- Clipboard: Alfred
- Others:
  - Alfred w/ workflows
  - Sip
  - Dash
  - iTerm 2
  - Total Finder
- Various cheat sheets

VISUAL STUDIO CODE

- Emmet is built-in
- Dual/split terminals (essential for this workflow)
- User snippets - Automate your most common actions
- Extensions
  - Project manager - Allows you to save and quickly switch between project workspaces
  - Liquid Languages Support - Will semi-correctly highlight Liquid syntax. I don't thing a perfect version of this exists in any code editor
  - Sass - Sass highlighting
  - Settings Sync - Allows you to backup/restore your VSC settings/extensions/snippets using a GitHub gist
  - Shopify Liquid Template Snippets - Collection of snippets for common Shopify code
  - htmltagwrap - Automatically surround selected text with tags
  - HTML Snippets - Collection of snippets for common HTML
  - Dash - Integrates with Dash app, allowing you to access API documentation in the code editor
  - Auto Rename Tag - When renaming a tag, automatically renames closing tag
  - Paste and Indent - When pasting code from another source, it will try to auto-indent correctly

WHAT SO GREAT ABOUT SLATE?

- Allows you to split your SASS into partials for better organization and concatenates them
- Concatenates your JavaScript also
- In /icons, will optimize svgs, strip out unneeded info, and save them as snippets with special classes
- Comes with a bunch of JS functions to help us out
- Comes with a very helpful page.styles.liquid template
- Calls your scripts as deferred (which creates a new challenge)

CONSIDERATIONS FOR THIS WORKFLOW

- For max compatibility, you'll want to approach building using progressive enhancement.
  - The first goal of your scaffold theme will be to make sure that the entire shop functions if JavaScript is disabled
- You'll want to have a conversation with your client regarding agreed-upon browser compatibility to manage expectations (for that moment when you're client is freaking out because he has problems viewing his site on IE 9 and demands you fix it)
- As you introduce functionality/features, check against caniuse.com and make sure there's fallback functionality if not all devices are supported (if client has analytics, they can be plugged into this site for a visual representation)

CONSIDERATIONS FOR THIS WORKFLOW. . . CONTINUED

- Avoid .scss.liquid and .js.liquid files
  - Keep js, css, and liquid separate
  - When you need to store JS variables from Liquid, do so in a dedicated snippet that's called in <head>
  - Using our gulp Sass pre-processing, there's a trick to passing through Liquid values: **border-color: #{'{{ settings.message-color }}'};** This tells the compiler to ignore what's inside #{''} and pass it through
  - You won't be able to use Liquid logic, however. Minor trade-off.
- Forget about Slate's /scripts and /styles. We won't need those folders
  - Slate's Gulp process compiles SCSS and JS using these folders before copying them to /dist. However, all files need to upload if only one file is changed. That adds up when you save often.
  - Our Gulp process will save our compiled JS and SCSS directly to /src, which is then copied to /dist. So when you save any part of your SCSS, only one file needs to be uploaded on save.

WORKING WITH A DESIGNER

- Educate your designers: To reduce your feedback loop and execute projects faster, consider creating a Shopify design guide for your designer, which should include:
    - Customer pages - a written or visual guide to all of the customer pages that may need to exist, when the required elements for each page
    - All the required elements for the cart page
    - Branded password page for under-construction store
    - 404 page
    - Etc…. Anything else that you find tends to be forgotten or not considered when design is happening. Remember, designers are likely not familiar with Shopify's limitations.

- Here's an example of how I help my designer put together a collection template design:

Collection

Things to consider

- Each collection can have a single featured image or no image at all.
- Each collection can have a description in any spot or no description at all. If metafields will be used, any collection can have more associated information, like extra collection images or extra descriptions.
- The max number of products pulled from Shopify per page is 50.
- If above 50 products, pagination under the grid, in this form:
    - "Previous" "1" "2" "3" "..." "last" "next"
    - Unless infinite scrolling is used, in which case no pagination guides are needed.

THINKING IN COMPONENTS

- Adopt a team-wide system of agreed-upon syntax, and try your absolute hardest to stick to it.
- Standardize your practices across your entire time, including the way you think about implementing components.

- You've encountered BEM all over the place with Shopify themes. It's used with CSS Wizardry Grid which was part of Timber.
- My favorite is rscss.io because it's super light and clean.
- Really make Shopify's snippets work for you!

EXAMPLE: THE PERFECT IMAGE SNIPPET

- Tired of typing in for every image that you use?
- What if there was a single snippet that you could use for every image in your theme? What would it look like? It would:
  - Be dynamic
  - Accomodate lazy loading
  - Use srcset and therefore be responsive
  - Take advantage of Shopify image object filters
  - Accomodate dynamic data attributes for linking with JS
  - Accomodate alt tag
  - Built-in accessibility
- Optimize this snippet over time!

INVERTED TRIANGLE CSS (ITCSS)

- Mental framework for manageable CSS, particularly at scale
- Specificity, if left unchecked, can be the bane of our existence, causing all sorts of problems that can't really be fixed later.
- Uses specificity to its advantage
- https://bit.ly/2Gj58fQ for a great description of the problem and how this solves it.

DOCUMENTATION NOT PERFECT, HERE ARE SOME THINGS THAT TOOK A WHILE TO LEARN

Initializing Foundation Correctly

```
// Inititialize Foundation
$(document).ready(function() {
    Foundation.addToJquery($);
    $(document).foundation();
});
```

There’s a css class that’s incredibly important

```
.foundation-mq{font-family:”small=0em&medium=40em&large=64em&xlarge=75em&xxlarge=90em”}
```

Foundation Events

```
$(document).on('open.fndtn.reveal', '#modalCart[data-reveal]', function () {
    console.log("modal revealed”);
});

$(document).on('opened.fndtn.offcanvas', '#offCanvasCart[data-off-canvas]', function () {
    console.log("canvas revealed");
});
```

NODE

- Start with Node Version Manager (NVM) before installing node.js (OSX) - manages multiple versions of node
- For me, the sweet spot version of Node is v8.6.0 to use with Slate and Gulp
- NVM installs Node. NPM depends on Node
- Node Package Manager (NPM) is used to install packages to use with your application.
- The preferred alternative to NPM is Yarn, which is installed using NPM.

GULP

- Things you can do with Gulp
  - Compile Sass (modern versions)
  - Concatenate JavaScript files
  - Transpile ES6 using Babel
  - Build JS using different module formats
  - Optimize images
  - Convert fonts
  - Generate critical css
  - ..... and anything else you can think of probably

PERFORMANCE OPTIMIZATION - BUILD WITH THESE IN MIND

- Google PageSpeed Insights warnings
  - Optimize images - Gulp
  - Resize image - The right image snippet
  - Minify JS and CSS - Gulp
  - Enable compression - Built into Shopify's servers (no control over 3rd party app resources)
  - Leverage browser caching - Same as above
- Eliminate render-blocking JavaScript and CSS in above-the-fold content - This is the big bad boy that takes some Gulp magic. If you build a theme with tackling this in mind:
  - Every page will need to have critical CSS inline
  - You'll need to consider when a single page has multiple above-the-fold scenarios. A good example is when your theme settings allow the client to choose between either using a hero slider or a hero video. Only one of these elements can exist when running the critical analysis.

PERFORMANCE OPTIMIZATION - BUILD WITH THESE IN MIND

- Always use SVGs when possible
- Think about the resources that are being loaded into the browser when the page loads when you're building elements. Some culprits I commonly come across when optimizing sites:
  - Quick shop functionality - All images for each grid item are loaded into the browser.
  - Not using Liquid image filters correctly to properly size user-uploaded images
  - Inefficient use of image formats (png vs jpg)

HOW I APPROACH A NEW PROJECT

- Conversation with client about browser compatibility, setting up expectations.
- Study the mockup in detail and look for aspects (and scenarios that are missing from the mockup).
- Export all assets from the mockup in both @1x and @2x versions
- Spend maybe a solid week focusing solely on the page.styles.liquid template. This is another new best friend! (And it will help you build modularly)
    - Slate comes with this template. Make a page and apply this template. You will be adding to it.
    - Get all the basics right in foundations settings file. Make all existing content on the template exactly how it should be, matching the designs, down to every single detail. If this is done first, everything falls into place later!
    - Study the mockup and recognize your modular elements (These elements' css should be in the global scope, not within something like .template-product)
    - Build them into page.styles.liquid one by one, using team-standardized rscss.io rules.
    - Consider turning each of these elements into a snippet, making use of snippet variables for…..
    - …..If most elements are built into this one page (page.styles.liquid), you just copy/paste the code when needed, then apply variant-css, build in schema Liquid, etc.
    - This approach is incredibly useful because you may need to duplicate these elements into different spots.
- When using this method, focusing your efforts on pre-emptive, defensive coding, the individual pages just fly by