Choosing a Responsive Email Framework: MJML vs. Foundation for Emails

Author
Paolo Mioni

Last Updated
Apr 23, 2018

Implementing responsive email design can be a bit of a drag. Building responsive emails isn't simple at all, it is like taking a time machine back to 2001 when we were all coding website layouts in tables using Dreamweaver and Fireworks.

But there's hope! We have tools available that can make building email much easier and more like coding a modern site. Let's take a look at a couple of different frameworks that set out to simplify things for us.

(#first-the-situation) First, the Situation

You have to be compatible with lots of old email clients, many of which don't even support the most basic web standards (floats, anyone?). You also have to deal with all sorts of webmail clients which, for security or technical reasons, have made their own opinionated choices about how to display your precious email.

Furthermore, now emails are read from different devices, with very different viewports and requirements.

The best solution, as is often the case, would be to keep things simple and stick to basic one-column designs, using multiple columns only for menus or simple interface elements of known width. You can produce great, effective designs using only one column, after all.

However end-users and clients, who are used to the "normal" browser-based web, may hold their email-viewing experience to the same high standards they do for viewing web pages in terms of graphics and responsiveness. Therefore, complex designs are expected: multiple columns, different behaviors on mobile devices as opposed to desktops, lots of images, etc., all of which have to be implemented consistently and pixel-perfect across all email clients. What options are available to make all that happen?

(#option-1-build-from-scratch) Option 1: Build From Scratch

One option you could choose is coding each email by hand, keeping it simple, and testing it while you go. This is a viable option only if:

1. You have a very simple design to implement.
2. You have direct control of the design, so you can eventually simplify things if you can't come out with a consistent solution for what you intended to do.

3. You can accept some degree of degradation on some older clients: you don't mind if your email won't look exactly the same (or even plain bad) in old Outlook clients, for example.
4. You have a lot of time on your hands.

Obviously, you need to test your design heavily. Campaign Monitor has a great CSS guide (https://www.campaignmonitor.com/css/) you can reference during the build process and is sort of like Can I Use, but for email. After that, I recommend using automated test suites like Litmus (https://litmus.com/) or Email on Acid (https://www.emailonacid.com/) . Both offer you a complete testing suite and previews of how your email will look like on a huge variety of email clients. Expect some surprises, though, because often the same design does not look correct even on the same email client, if viewed on different browsers, or different operating systems. Fonts will render differently, margins will change, and so on.

Screenshot of the same email design tested on different devices on Email on Acid.

(#option-2-use-a-boilerplate-template) Option 2: Use a Boilerplate Template

Another option is to use one of the various boilerplates available, like Sean Powelll's, which you can find here (https://github.com/seanpowell/Email-Boilerplate) . Boilerplates already address many of the quirks of different email clients and platforms. This is sensible if:

1. You are working alone, or on a small team.
2. You have lots of experience, so you already know most of the quirks of email formatting because you've met them before.
3. You have set up your own tools for managing components (for different newsletters which share some pieces of design), inlining styles (https://litmus.com/blog/a-guide-to-css-inlining-in-email) (and yes, you should keep inlining your styles (https://emails.hteumeuleu.com/should-we-stop-inlining-styles-in-emails-8c3b64f0d407) if your emails target an international audience), and have some kind of development toolkit in place (be it Gulp, Grunt or something similar) which will automate all of that for you.
4. You have the kind of approach where you'd like to control everything in the code you produce and don't like to rely on external tools.
5. You prefer to solve your own bugs instead of having to solve possible bugs caused by the tool you are using.

(#option-3-use-a-framework) Option 3: Use a Framework

However, if any of the following points are valid for you:

1. You have to produce a lot of email templates with shared components.
2. The job has to be carried out by a team of developers, who might change and/or have a variable degree of proficiency and experience with email implementation.
3. You have little or no control on the original design.

...then you will likely benefit a lot from using a framework.

Currently, two of the most interesting and popular frameworks available are MJML (https://mjml.io/) and Foundation for Emails (https://foundation.zurb.com/emails.html) . The biggest advantages in using either framework is that they have already solved for you most of the quirks of email clients. They also provide you with an established workflow you can follow, both alone and as a team. They also handle responsive design very well, albeit differently from one another.

Let's look at an overview of both frameworks and compare the best use cases for each before drawing some conclusions.

(#mjml) MJML

MJML (https://mjml.io/) is a project that was created internally by Mailjet (https://www.mailjet.com) , a company specializing in email marketing tools. It was then open-sourced. It works with its own custom, HTML-like templating language, using its own tags. For example:

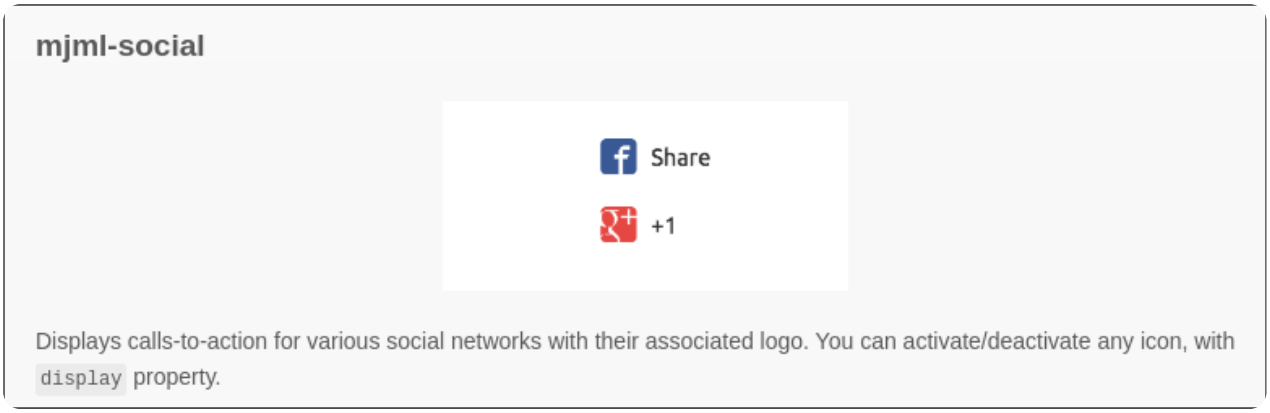
```HTML
<mj-text>Your text here</mj-text>
```

When you compile the final code, MJML will convert their tags into HTML for everything from tables to custom components they have created for you, all using its internal engine. It takes out the heavy lifting for creating complex markup and it's all been tested.

MJML has a set of standard components (https://mjml.io/documentation/#components) . They include sections, columns, groups, buttons, images, social links (which are very easy to create), tables, accordions, etc. They even include a pre-styled carousel, which should work in most clients. All of these components translate well into responsive emails, apart from the "invoice" component which I could not get to work in my tests. All of these components have parameters you can pass in your code to customize their appearance.

For example, the social links component allows you to stack icons vertically and horizontally, and to choose background colors for the related icons. There are actually a lot more parameters (https://mjml.io/documentation/#mjml-social) you can play with, all with the intent of allowing for greater flexibility. Even the logo image files are already included in the package, which is a great plus.

Here's a preview of a simple configuration of the social links component:



Screenshot from the MJML website (https://mjml.io/documentation/#mjml-image) .

You can also create your own custom components (https://mjml.io/documentation/#creating-a-component) , or use components created by the community. There is just one community component available at the moment, however.

MJML handles responsiveness automatically, meaning that components will switch from multi-column (more items in the same row) to single-column (items are put one under the other instead of side-by-side) without any active intervention from the developer. This is a very flexible solution, and works fine in most cases, but it gives the developer a little less control over what happens exactly in the template. As the docs mention (https://mjml.io/documentation/#getting-started) , it's worth keeping mind that:

> *"For aesthetics purposes, MJML currently supports a maximum of 4 columns per section."*

This is most likely not only an aesthetic preference but also about limiting possible drawbacks of having too many columns. The more columns you have, the more unpredictable the output, I guess.

(#how-to-work-with-mjml) How to Work With MJML

MJML can work as a command line tool, which you can install with npm, and output your files locally, with commands like:

Commande Line

```
$ mjml -r index.mjml -o index.html
```

This can be integrated in your build procedure via Gulp or the like, and in your development work by using a watch command, which will update your preview every time you save:

Commande Line

```
$ mjml --watch index.mjml -o index.html
```

MJML has plugins for Atom and Sublime Text. In Atom, it even supplies a real-time preview of your layout, which can be regenerated on every save. I haven't tried it personally, but it seems very interesting:

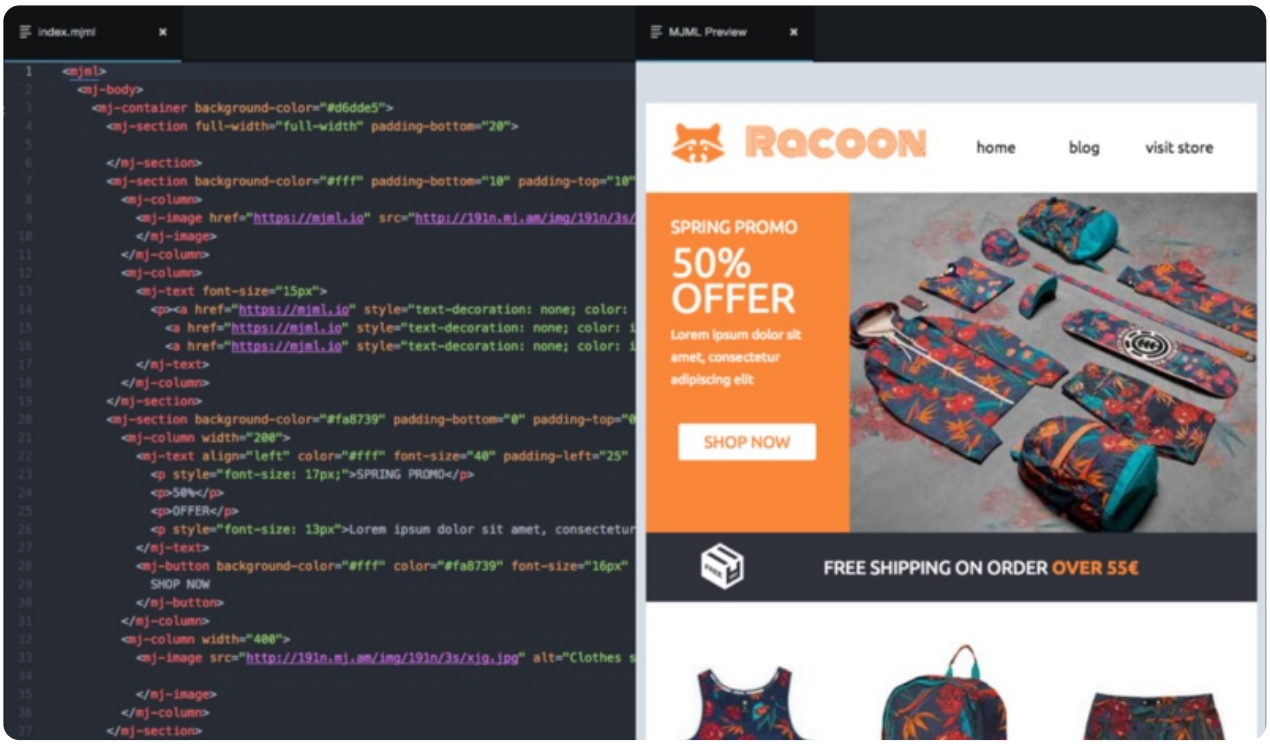
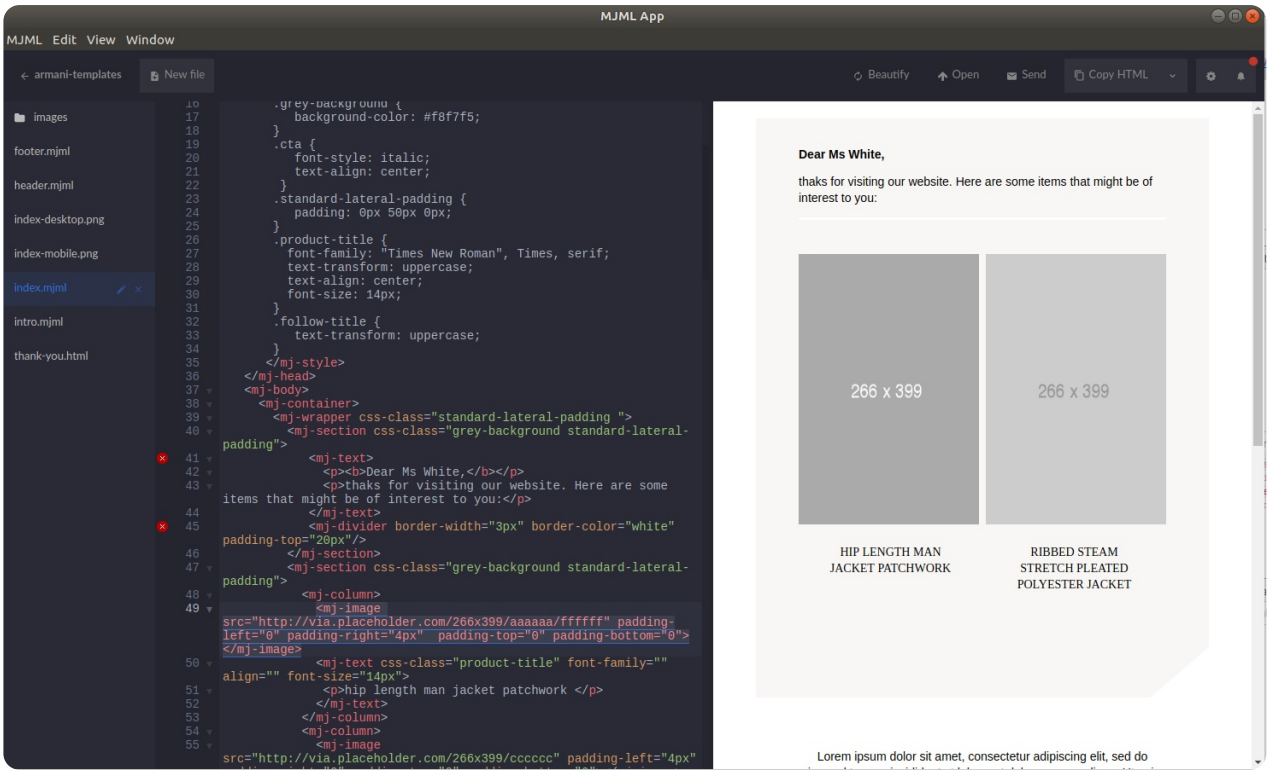


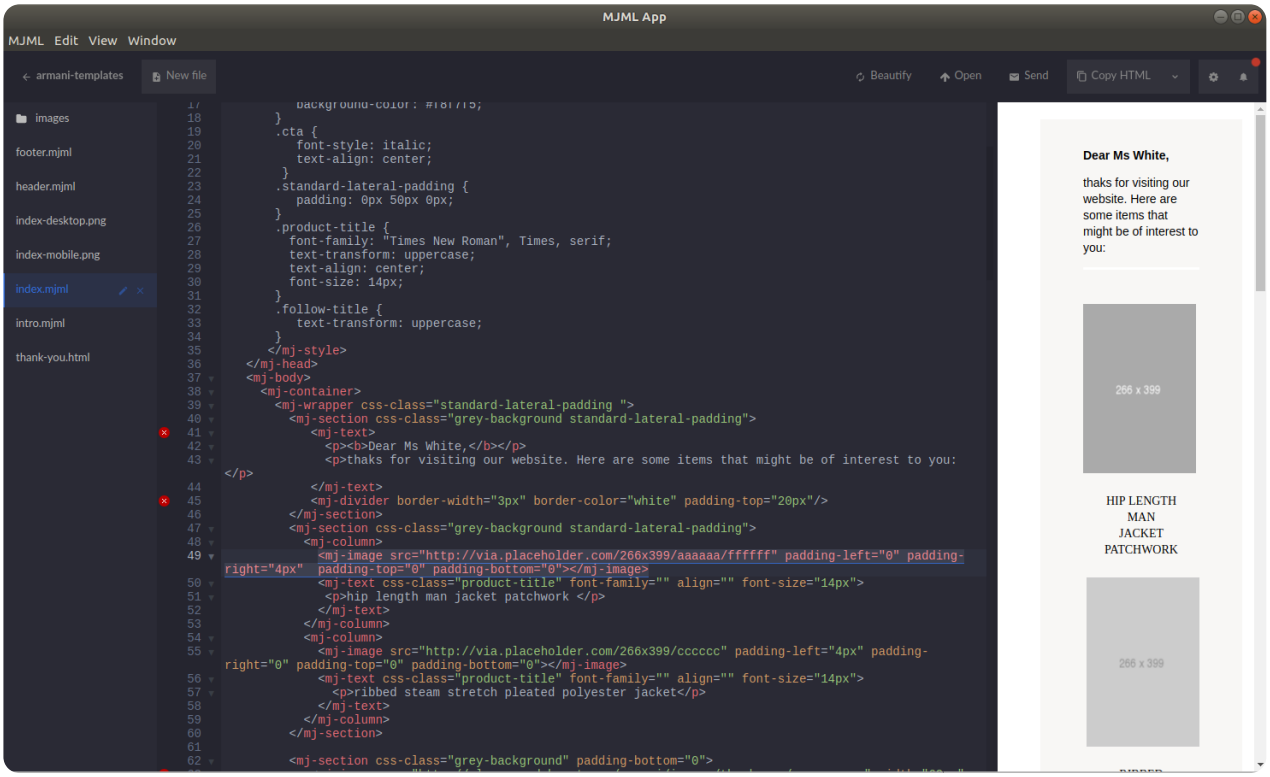
Image from Atom.io (https://atom.io/packages/mjml-preview)

MJML also has its own simple desktop app (https://mjmlio.github.io/mjml-app/) , and it's free. You can set up your code and components, have it build your designs for you, and get a real-time preview of the results, both for mobile and for desktop. I find that it works pretty well on Ubuntu, but the only drawback I've found is that you should never, never, *never* open your files with another editor while they're open on the app, because the app crashes and the content of the file gets lost.

Here are some screenshots of the previews at work:



Desktop preview of email



Mobile preview of email

The app can also be integrated with a free Mailjet account, which allows you to send test emails immediately to yourself. This is very handy to quickly check problematic clients if you have them available directly. (I would suggest taking out that old Windows machine you have in the storage room to check things in Outlook, and to do it as often as possible.) However, I would still recommend using either Litmus or Email on Acid to test your emails cross-client before deploying them because you can never be too careful and email standards can change just like they do in browsers.

Overall, I have found MJML very easy to use. However, when I tried to make a pixel-perfect template which was identical to the design our client requested, I had some difficulties dealing with custom margins for some images. Not all of the component parameters available worked as expected from their description in the documentation. In particular, I had some

problems customizing image margins and padding between desktop and mobile.

(#advantages) Advantages

- Pre-built components
- Integration with Mailjet
- Easy to use, with instant preview of your work (on Atom and Desktop App)

(#disadvantages) Disadvantages

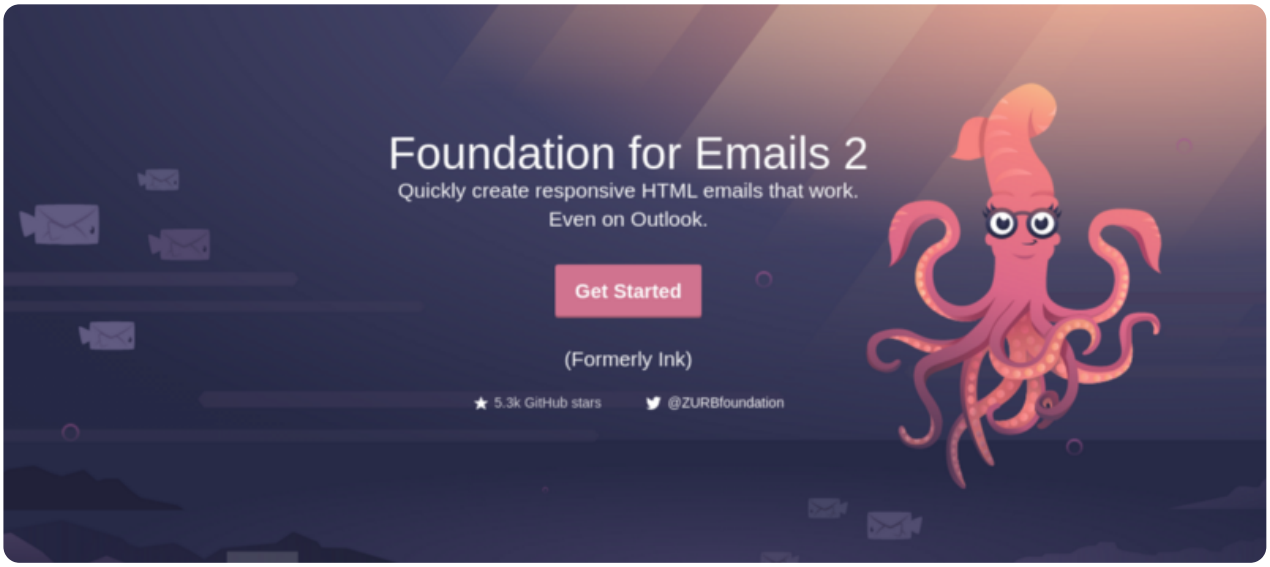
- A bit less reliable than Foundation for Emails if you have to do pixel-perfect designs
- Some components have parameters that don't work as expected
- Desktop App not perfectly stable
- Does not come with a structured set of folders for your content (see Foundation below)

(#foundation-for-emails) Foundation for Emails

Foundation for Emails (https://foundation.zurb.com/emails.html) (formerly known as Ink — insert obligatory Prince quote here) is a framework by Zurb, the same folks who brought us the responsive front-end framework, Foundation for Sites (https://foundation.zurb.com/sites.html) .

When you download the Starter Kit (https://foundation.zurb.com/emails/getting-started.html) you get a full development environment, complete with Node.js commands to run your project. It will setup a Node routine and even open your browser to give you an immediate preview of your work.

The files you have to use are already organized in folders and subfolders, with a clear indication of where to put your stuff. For example, it has directories specifically for partials, templates and images. I find this feature very important, because it is very easy to end up using different folders when you work on a team, and this leads to a lot of lost time looking for stuff that isn't where you expect it to be. Enforcing conventions is not a limitation; when you work in a team it is indispensable.



TFFKAI — The Framework Formerly Known As Ink

Foundation for Emails comes with a boilerplate template, which is the starting point for your code. It is fully annotated, so you know how to extend it with your code. It comes with

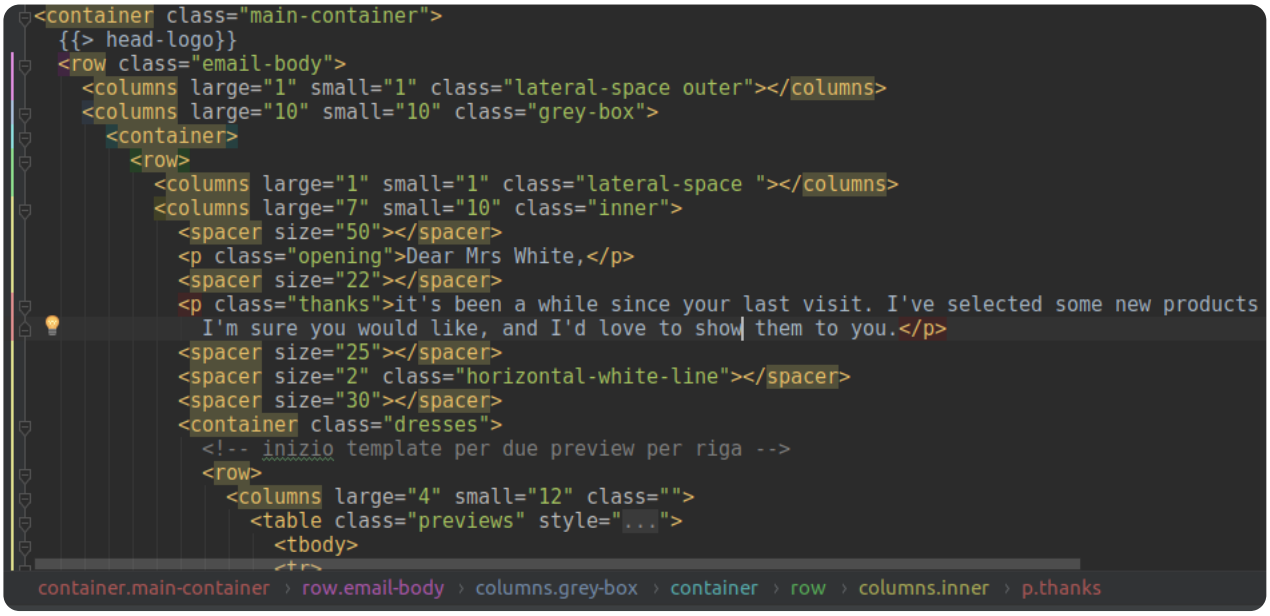
SASS/SCSS support, which is very very handy for complex projects. It also comes with its own inliner, so you don't have to worry about converting all your CSS (or SASS/SCSS) into inline styles.

There's a template engine behind this framework called Inky. And, just like MJML, it has custom tags that will automatically convert to HTML when it's compiled. There are tags like `<container>`, `<row>`, `<column>`, which will be used to build your grid. The grid is based on a 12-column system, which allows you to subdivide your layout very accurately. Why 12? Because it is divisible by 2, 3, 4 and 6, making it very easy to make a 2-column, 3-column, 4-column, or 6-column layout.

Foundation for Emails uses Panini (https://foundation.zurb.com/emails/docs/panini.html) to compile the code. Panini is a custom library which compiles HTML pages using layouts. It supports Handlebars (http://handlebarsjs.com/) syntax and there are several helpers you can use to customize the behavior of components depending on where they're being used. You can also create your own helpers if you need to and set each template's custom variables with custom data. This is very useful if you have several templates sharing the same components.

There are several pre-built email templates available (https://foundation.zurb.com/emails/email-templates.html) you can download, which cover many of the standard use cases for email, like newsletters and announcements. There are also a few pre-built components (with their own custom tags),

including buttons, menus and callouts (which, I have to admit, I don't see a purpose for in emails, but never mind).

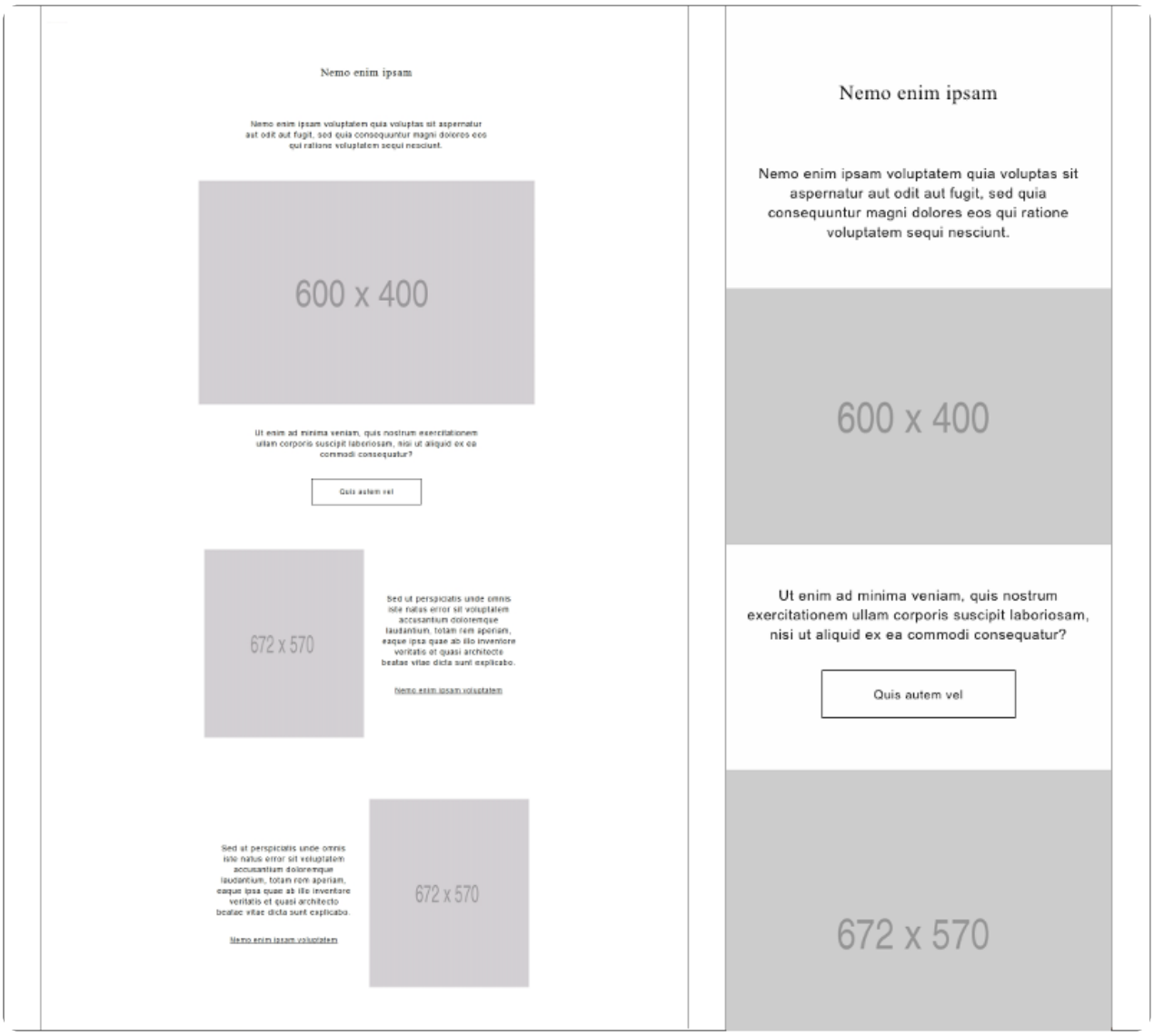


A code sample from a Foundation for Emails template.

The main difference between Foundation for Emails with MJML is that Foundation for Emails defaults to desktop view, then scales for smaller screens. According to the docs, this is because many desktop clients do not support media queries and defaulting to the large screen layout makes it more compliant across email clients. That said, it only manages one breakpoint. You create the desktop version and the mobile version, and the mobile version switches under a certain number of pixels, which can be configured.

You can decide whether some components will be visible only on large or small screens using handy pre-defined classes like `.hide-for-large` and `.show-for-large` (although `.hide-for-large` might not be supported by all clients (https://foundation.zurb.com/emails/docs/visibility.html)). You can also decide how much space a column will take by using specific classes. For example, a class of `.large-6` and

`.small-12` on a div will make a column that occupies half the screen on desktop and the whole screen width on mobile. This allows for very specific and predictable layout results.



A preview of the same e-mail template, developed with Foundation for Emails, on Outlook 2007 (left) and iPhoneX (right).

Foundation for Emails is a bit clunkier to use than MJML, in my opinion, but it does allow for much tighter control on the layout. That makes it ideal for projects where you need to reproduce pixel-perfect designs, with very specific differences between mobile and desktop layouts.

(#advantages) Advantages

- A more precise control over end results
- Pre-built templates
- Sass support
- Great documentation

(#disadvantages) Disadvantages

- The project file size is heavy and takes a lot of disk space
- A little less intuitive to use than MJML's pre-defined parameters on components
- Fewer components available for custom layouts

(#conclusions) Conclusions

Producing email templates, even less than front-end development, is not an exact science. It requires a lot of trial and error and a LOT of testing. Whatever tool you use, if you need to support old clients, then you need to test the hell out of your layouts — especially if they have even the smallest degree of complexity. For example, if you have text that needs to sit beside an image, I recommend testing with content at different lengths and see what happens in all clients. If you have text that needs to overlap an image, it can be a bit of a nightmare.

The more complex the layout and the less control you have over the layout, then the more useful it is to use a framework over hand-coding your own emails, especially if the design is handed to you by a third party and has to be implemented as-is.

I wouldn't say that one framework is better than the other and that's not the point of this post. Rather, I would recommend MJML and Foundation for Emails for different use cases:

- MJML for projects that have a quick turnaround and there is flexibility in the design.
- Foundation for Emails for projects that require tighter control over the layout and where design is super specific.

(#resources-and-links) Resources and Links

- The MJML website (https://mjml.io/)
- The Foundation for Emails website (https://foundation.zurb.com/emails.html)
- Litmus Email Testing (https://litmus.com/)
- Email on Acid Testing (https://www.emailonacid.com/)
- An interesting conversation (https://litmus.com/community/discussions/5407-what-is-your-email-framework-of-choice) on the Litmus forum, which was in some ways the starting point for this article.
- Another article (https://www.rivalhound.com/blog/post/inkyvsmjml) by James Luterek that compares these frameworks