oyster.engineering /post/124868558323/emails-oy-vey-render-emails-with-react

Emails, oy vey! Render emails with React

5-6 minutes

Emails, oy vey! Render emails with React

Vivek Patel

Today we're excited to open-source Oy, a set of utilities to render email templates server-side with React. Oy provides a set of utilities to:

- Move backwards in time to render HTML4, since React only supports HTML5.
- Validate attributes based on best-practices.
- Render your templates server-side.

Background

This is the latest step in a number of evolutions in how we've approached email template construction.

We started out building our templates directly in our email service provider's template UI. As a small team with a simple and limited set of emails, this worked.

Very quickly, we hit walls: Simple design changes like using a new header color meant going into each template manually and updating the code. This led to designs becoming outdated and inconsistent, which was unacceptable.

Modularity would help us keep development speed high with some useful guarantees for consistency. We broke our emails into modules and moved development to an internal templating toolchain which used Nunjucks `include` blocks to chain together modules to quickly construct emails. This drastically cut down development time for emails. This system—called Bosun—is what we've used for the past year.

Why change? A few things have changed in the last year: different developers building emails and more interesting layouts meant that quality became a serious issue. Buggy layouts and wildly inconsistent rendering were a growing occurrence.

Architecture

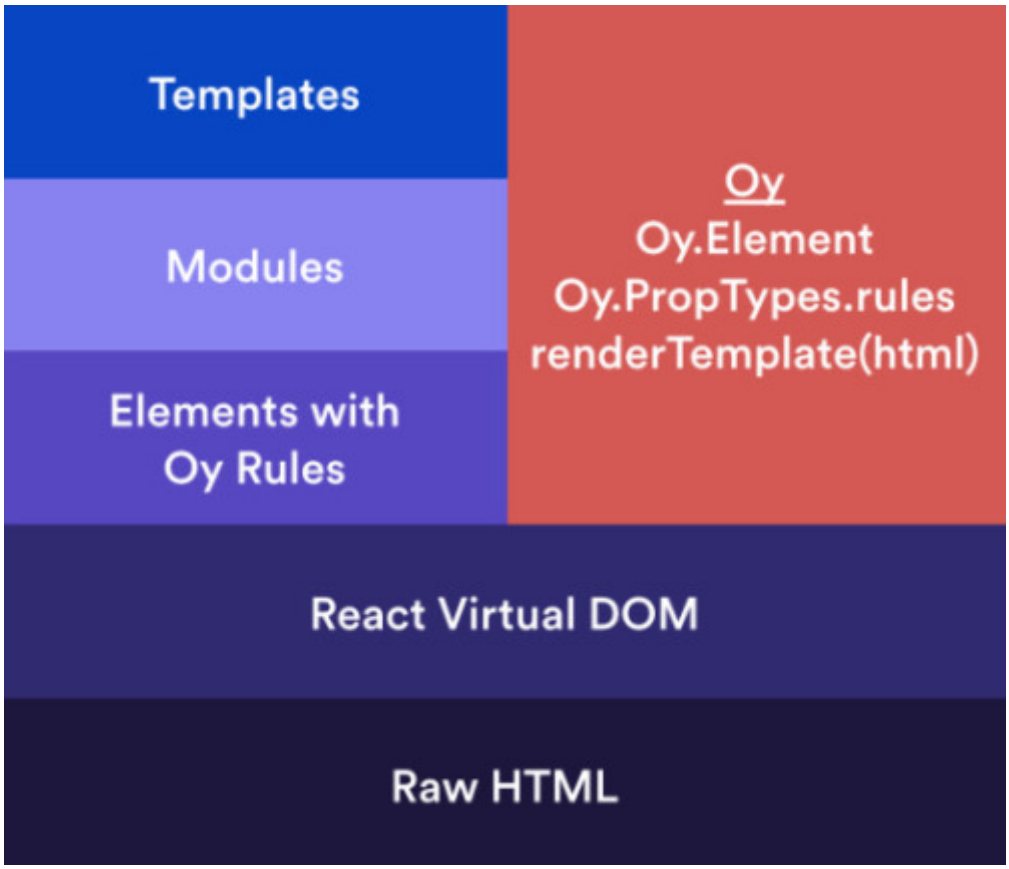
React has helped us address quality problems, and in certain cases has wiped out entire classes of problems.

Writing JS via JSX ensures no malformed HTML can exist, dealing with much headache and manual validation work. The virtual DOM allows us to render templates server-side. And React's component philosophy allows us to reuse and update common styles and layouts efficiently and with some consistency guarantees. React's propType validations also help notify the developer of mistakes.

React by itself has some limitations when it comes to email rendering, however:

- React only supports HTML5 tags and attributes, where we need to be able to render HTML4 tags and attributes for backwards-compatibility reasons.
- React mounts to an element and needs a base template to insert its rendered HTML string into.
- React provides helpful warnings for a modern website, but not for emails written for the 90s.

This is where Oy comes in:



Oy provides the tools necessary to fill the gaps that React can't (and probably shouldn't) to help us render email templates on the server.

Design Decisions

In designing Oy, we wanted to channel the principles that make React great: explicitness, modularity, and simplicity.

Early on in development, we felt autocorrection would help abstract away esoteric email rules: for example, 3-digit hex background colors aren't entirely supported, so to be safe you should probably use the 6-digit hex; also, you shouldn't use shorthand CSS styles for fonts; also, it's a good practice to set the line-height and font-size to 1px for empty spacer elements—the list goes on. After developing a handful of email templates, we noticed we didn't actually need autocorrection—warnings were good enough. If we wanted defaults, we could update our Oy wrapper components with default props. We felt hiding rules from the developer was a slippery slope to go down, especially from the outset.

This simplified things, making integration with React's already idiomatic propTypes clear.

Good API design is critical to any system. React's handful of component lifecycle methods mean developers can get up and productive in very little time. We aimed for a similarly simple API.

Usage

Using Oy means wrapping OyElement components with your own.

```
import React from 'react';
import Oy from 'oy';

export default React.createClass({
  propTypes: {
    border: Oy.PropTypes.rules(['TableBorderCellPaddingCellSpacingRule']),
    bgColor: Oy.PropTypes.rules(['SixCharacterHexBackgroundColorRule'])
    style: Oy.PropTypes.rules(['NoCSSShorthandRule'])
},
```

```
  render: function() {
    return React.createElement(Oy.Element, React.__spread({type: 'table'},
this.props);
  }
});
```

With these, you can then use HTML4 attributes like align and bgcolor on table and td. You could also provide default props, like border: 0, cellPadding: 0, cellSpacing: 0. At Oyster, we use these custom components to create modules (i.e. headers, footers, text elements, etc.), and then create templates out of modules.

Once ready to render the entire HTML, just call `Oy.renderTemplate` with an options object.

```
Oy.renderTemplate({
  bodyContent: '...',
  headCSS: [cssReset, customCSS].join(' '),
  previewText: 'This is test preview text.',
  title: 'Oyster'
});
```

Contributing

We're releasing Oy with just a few rule validations—there are so many more out there that people need to know about. This project is about codifying all the esoteric "best-practices" out there.

Another important next step is to allow developers to provide a custom base template.

The codebase is quite approachable, so please, dive in!

Interested?

Here's Oy on Github.

We're always hiring exceptional engineers—if rethinking how we discover and read books interests you, reach out!