Intro

Boltmade is a company that works closely with its clients to build quality web and native applications. One of the primary technologies used at Boltmade for building these applications is the Ruby on Rails web framework. Introduced in 2004, Ruby on Rails has become one of the leading choices for developing robust web applications as a result of its powerful features, ease of use, and committed following. The framework is written in the Ruby programming language and features a MVC (model-view-controller) architecture. A difficulty that often arises when building Rails applications is the design of dynamic HTML components. Interaction are handled with static components by making a request to the server to update the application's state and once this has completed, the server re-renders the current page. While this approach ensures that the page is accurately updated to reflect the change initiated by the user, it results in re-drawing parts of the page that did not need to change, and prevents the user from interacting with the interface until the page has finished reloading. Dynamic components overcome these problems by making asynchronous requests to modify the application's state and then manually updating visual components on the page using client-side JavaScript DOM manipulation. This approach becomes very difficult as the page's visual components become increasingly complex and often results in brittle implementations that are prone to errors. One proposed solution that Boltmade has explored on some of its most recent Rails projects is the use of Facebook's React library for building dynamic JavaScript components.

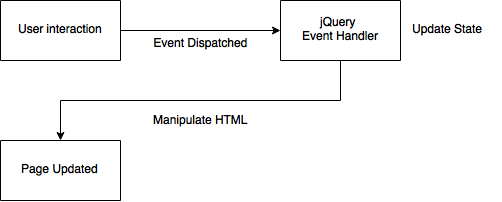
This report examines the potential for using the React library over plain JavaScript for handling complex changes to components in current and future Ruby on Rails applications at Boltmade. It first illustrates the basic implementation of a native JavaScript approach for handling simple user interactions. It then compares this model to the system used by the React library and examines the advantages of using React for creating more complex dynamic components. It concludes with an assessment of the overhead required to incorporate React in to new and existing Rails applications.

The Structure of the Rails FrameWork

The Ruby on Rails framework follows a simple MVC pattern that can be found in many common web frameworks. Models in Rails consist of Ruby classes that closely integrate with the application's data. The controllers are responsible for exposing the data from the models to the views and performing the logic for updating the data as a result of a user interaction. The views then take the data prepared in the controllers and present it in a way that is useful to users. The views can be written in several markup languages that all result in the output of HTML that is rendered to the browser. A common way of creating dynamic components is to use Ajax requests, which make asynchronous JavaScript requests to the server. Once the request has completed and returned any necessary data, the view can then be dynamically updated using client-side JavaScript to reflect this change. For making small changes to the page, it is often simplest to use plain JavaScript to update the elements of the DOM that need to be changed. As the complexity of these visual changes increate, it becomes challenging to build well-structured components without the use of additional libraries.

Dynamic Components with Native JavaScript and jQuery

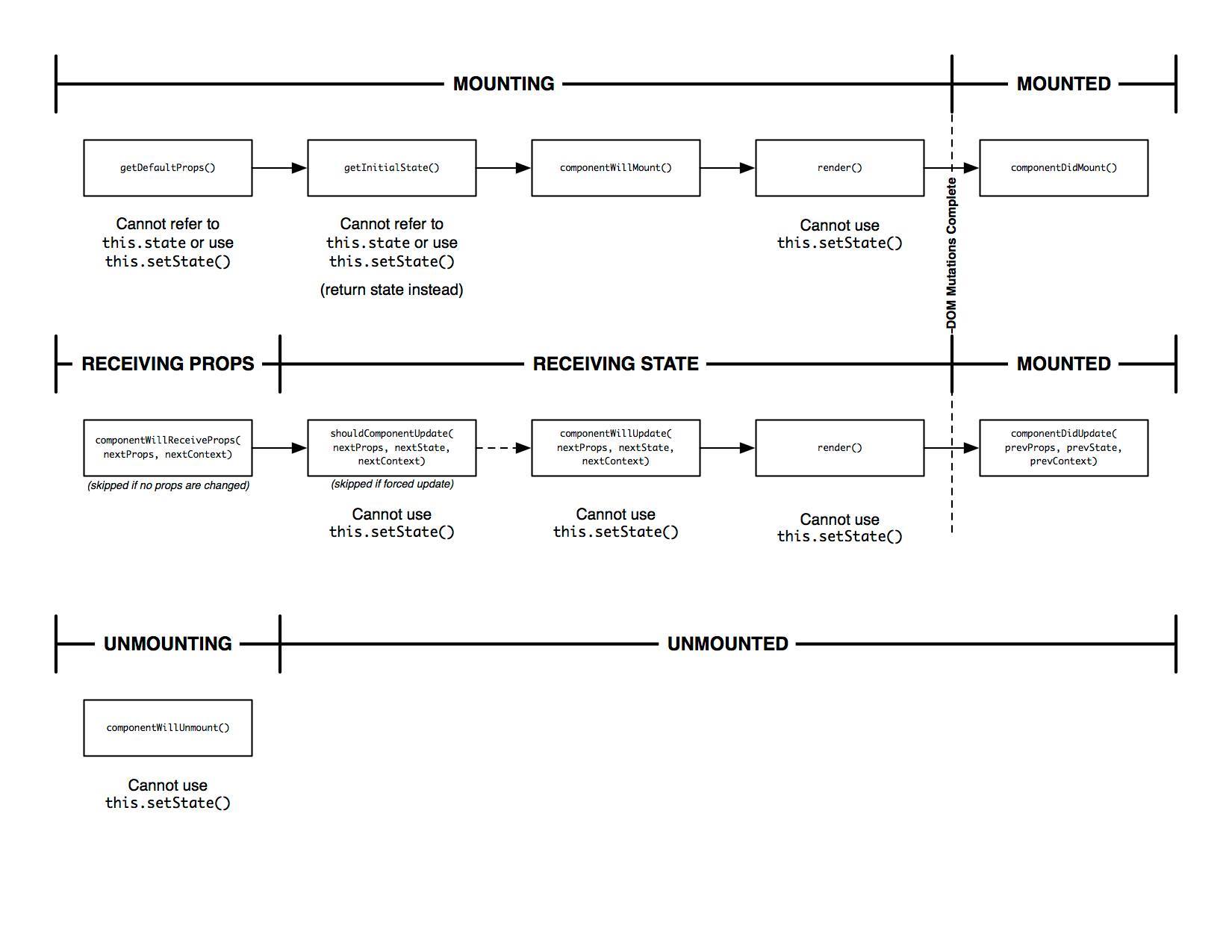
The most important factors to consider when designing many of these JavaScript components are performance, re-usability, and how the complexity of the code scales as the components are assigned more features and requirements. The only JavaScript tool that Boltmade consistently uses to help manage components is jQuery, a library that offers many convenience functions for manipulating the HTML document, adding and handling events, and making Ajax requests. In the current approach, HTML files describe the structure of the page and any dynamic updates to the visual components are performed using jQuery event handling.



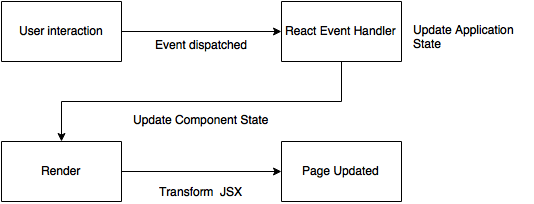
When the user interacts with an HTML element, an event is dispatched and interpreted by the jQuery event handler. The handler then makes an Ajax request to the server to update the application's state and once this request has completed, it modifies the page's visual appearance. While this approach is a simple model for performing small manipulations to the page, it is not a viable solution for more complex interactions. An immediate problem with this approach is that it mixes the logic for updating the application's state with the logic for updating its visual appearance. This practice results in an event handler that is bloated and prone to errors as a result of these mixed responsibilities. A larger concern with this implementation is its reliance on complex path knowledge. If this interaction had been handled by updating the application's data and re-rendering the entire page, then the page's visual structure would always be consistent with the state of the application. Since in this approach the page is not being reloaded, it is the event handler's responsibility to contain the knowledge of all the ways in which the page can change as a result of the possible updates to the application's state. As the number of paths that the event handler must consider increase, its implementation becomes brittle and difficult to maintain. As an example, one of the components that was designed on a recent project involved creating a set of toggles that each filter different content in a list of items. As the logic for what content each filter shows and hides became very complicated, it was necessary to find an alternative approach for managing these components. Through discussions with many teams on various past projects it is clear that manually executing changes to visual components is fundamentally a more complex operation than re-rendering a component given a new set of data. While handling small dynamic changes using event handlers is a common practice on Rails applications at Boltmade, an effort has been made to explore better solutions for handling these complex interactions.

Dynamic Components with the React Library

The React JavaScript library has been incorporated in to several recent Rails applications at Boltmade as an alternative method of handling dynamic updates to pages. React was designed by Facebook as a framework-agnostic solution for implementing the view portion of a MVC application. The library is focused around the creation of modular JavaScript components that each represent a section of the component hierarchy of the page. Using React, a page can easily be broken down in to a number of unique components corresponding to the natural divisions in its visual design. Different parts of the page such as a header, footer, or a group of inputs like a contacts list can each be rendered by separate React components using JavaScript rather than static HTML files. The React components can then be shared and re-used on any page that requires a particular component. This approach allows for a level of modularity and code re-use that was not often achieved on previous Rails projects, even through the use of re-usable HTML templates. In addition to its modularity, the largest benefit the team experienced after working with React was the library's simplified process for handling dynamic changes to the structure of a component.

React Life-cycle

Each React component has a component life-cycle that controls how the it should update its internal state and visual appearance. The first operation the component performs is to setup its initial states and props, short for properties, that are used to manage its behaviour. The props of the component are immutable attributes of any native JavaScript type that can be used to determine certain behaviours of the component, such as the generation of custom HTML element. One common use case for a prop is string title for an immutable header or label that the component renders. The states of the component are the same as the props, except that they are mutable. States are one of the most powerful advantages of using React components, as they are easy to set and allow components to internally track situations such as whether a component that renders a check-box currently has a state of checked or unchecked. The most important phase of the component's life-cycle is its render method. This method returns a description of all of the DOM elements that the component should generate, as well as any child React components that should be created and nested within the current component. The elements are specified using a XML-like syntax called JSX which is transformed in the browser into native JavaScript. The outputted JavaScript then adds all the of elements in the proper hierarchy to the DOM.



The render life-cycle event is triggered when the component is first created as well as when any of the component's states are updated. One advantage of this life-cycle system is that it immediately addresses the issue of complex path knowledge that hindered the previous approach. The difficulty with the simpler jQuery event handler had to know precisely how to update certain parts of the visual structure of the page based on the changes to the application's state. In the React version of the toggle filtering example discussed in the event handling approach, when a filter is clicked, React fires an event using its custom event system, which is then forwarded to a defined handler. The handler then updates the application's state through a request to the server, as well as the filters state of the React component itself. This update to the component's state triggers a re-render of the component, and the compiled JavaScript updates the page to reflect the state change. React is using a similar approach to the previously discussed model of reloading the page after every change to the application. While the advantage of completely reloading the page comes at the cost of performance and user experience, React uses a parallel concept for handling visual updates, but without either of these disadvantages.

React Virtual DOM

The React library uses an efficient client-side method of changing only the elements of the DOM that must be updated as a result of a state change. Instead of re-rendering the entire component, the library creates a diff between the JSX description of the DOM returned from the render method and an internal representation of the DOM that is stored in memory. By using this virtual DOM technology, React is able to achieve a level of performance which allows its components to be defined using a single visual structure in the component's render method.

Overhead of Adding React to Rails Applications

React is a simple JavaScript library that can be easily downloaded and incorporated into most web applications. The Ruby programming language has an extensive package manager that makes it simple to incorporate Ruby programs and libraries in the form of gems. The react-rails gem is available for installing React under Rails and includes a post-installation script that adds all of the necessary JavaScript files to the project. After running the post-installation script the application is then completely prepared for running React. The gem is available for Rails applications at version 3.2 and above, which was released in January of 2012. This makes the gem an effective choice for installing React on all recent and future Rails applications at Boltmade.

Conclusion:

In cases where there is a need for handling dynamic interactions it is clear that the React library offers significant advantages over plain JavaScript implementations. The comparison of jQuery event handlers to React components has determined that the React library should be used on future Rails applications at Boltmade that require the creation of complex dynamic components.