Progress®

Planning a React Application

Brandon Satrom

WHITEPAPER

Table of Contents

Progress®

Planning a non-trivial web application that utilizes React is something you may have done in the past or are in the midst of doing. This whitepaper documents a high-level process to use when planning a React application, from project organization, collaboration considerations and tooling choices during development, all the way through deployment and performance strategies. There are a lot of moving pieces to creating a real-world application in React, but this guide will help get you started.

Project Management

Before you write a single line of code, you need to decide how you're going to get things set up. This typically starts with project management, including discussing and agreeing upon the approaches, tools and services you'll use to deliver your next application.

Software Management Tools

To manage the development of any front-end application, consider the following tools to version code, store assets and oversee team member tasks:

SOFTWARE MANAGEMENT TOOL	EXAMPLES
Issues and feature tracker	GitHub, BitBucket, JIRA
Distributed version control system (DVCS)	Git, Mercurial
Cloud-based DVCS repository	GitHub, BitBucket
Document/asset storage	Internal Network, Google Docs, Basecamp, Office365
Team communication	Slack, HipChat, IRC, Google Hangouts, MS Teams
Task management	GitHub Issues, GitHub Project Boards, Trello, JIRA

No matter which tools you choose, it's essential that your team adopt and use the ones you select. Also, don't be afraid to monitor your use of these tools and improve your workflow if opportunities for improvement arise. New tools are released all the time and you may wish to adopt emerging tools that provide features you find missing in your current process. It's natural to adopt different tools as your team matures and your application grows.

Progress®

Accessibility, i18n and Environments

Accessibility, i18n (internationalization), and targeting the correct execution environment for your app are essential parts of any development effort. More than just what to build, it's essential to consider how you're going to build your app, who you intend to use it, and how you're going to support it. Addressing these considerations at the start of your project will help you clearly articulate how you'll address key aspects of your app that are ancillary to the code itself, but essential for certain audiences (accessibility and i18n, for instance).

The following table summarizes some of these considerations and provides some helpful resources applicable to addressing them for React-based applications.

APP CONSIDERATION	EXAMPLES	RESOURCES
Internationalization / globalization	UI and UX translations for multiple languages	https://github.com/yahoo/react-intl
SEO	Server-side rendering to enable search indexing	https://reactjs.org/docs/react-dom-server.html, https://preactjs.com
Cross-browser support	If your site must support IE10+ and all modern browsers (Edge, Chrome, Safari)	http://Babeljs.io
Accessibility	WAI-ARIA, WCAG	https://www.w3.org/WAI/intro/aria, https://www.w3.org/WAI/intro/wcag
Offline-first	Service workers	https://developers.google.com/web/fundamentals/primers/service-workers/
Progressive web apps (PWA)		https://medium.com/@addyosmani/progressive-web-apps-with-react-js-part-i-introduction-50679aef2b12
Cross-platform native mobile app	React Native, NativeScript	https://facebook.github.io/react-native/, https://nativescript.org/

The resources above are examples for consideration when deciding baseline standards and the types of support your application can offer. There are other approaches and new options crop up all the time. What's more, if your app won't benefit from an offline-first or PWA approach, don't build one in. Always consider the goals and intended audience of your app.

Progress®

Development Process Methodology

There are a number of different approaches to software development that have evolved over the last 50+ years. Waterfall, Agile, Scrum and Kanban are among the most notable.

Whichever project methodology you select, it's important to remain consistent, and to ensure that you have buy-in and support from key stakeholders beyond your development team. This includes management, executives and project customers. Some methodologies—Scrum, for example—require active engagement from non-engineering resources. Securing the support of these stakeholders is essential to a successful project.

Over the course of my career, I've found success with the more agile learning methodologies like Scrum and Kanban. For each company and in each case; however, I've been careful to adapt those processes where needed to fit within the culture and context of the team and organization in which the team operates.

Tooling and Development

Tooling has grown in importance among web application developers in the last decade. As the complexity of web applications has grown, so has the variety, scope and scale of the tools developers use to create these applications. In the context of a web application, package managers, module loaders and bundlers, linters, task runners, online collaboration tools and UI frameworks are the key building blocks of developing a robust React application.

Let's take a look at some of the current popular tooling for React applications.

Package Managers

Package managers help you manage dependencies for an application and ensure that these are available for every environment in which your app will run. For example, npm is often used to fetch dependencies for development, in addition to those needed for production.

Development dependencies are tools that you need during the creation of your app, but which are not required in production. Examples include unit testing tools, code linters or transpilation libraries like TypeScript, which produces your production code assets at build-time, and is not needed in production. Production dependencies are those dependencies that are required for your app to run in production, like React itself, CSS and UI libraries or utility tools like moment.js.

Progress®

Here are a few tools to consider when choosing a package manager:

PACKAGE MANAGERS
npm
Yarn
jspm.io
Bower

Task Runners

JavaScript task runners allow you to automate many tasks common to complex web application development and deployment. Managing and performing these types of tasks are error-prone when left to humans, but task runners make these simple, and speed up application development and deployment.

Task runners can be used to start a local development server, to compile, minify/uglify assets, run test suites and more. In recent years, webpack has become the de facto standard in the React community, though there are other solid options available.

TASK RUNNERS
webpack
npm
Grunt
Gulp
Tree.js

Linters and Enforcement

When working as a part of a team of engineers, a common goal is to ensure that each piece of code authored is written as if it were coded by a single person. The "common voice" idea extends from things like application structure and error-handling, all the way down to formatting and code styles.

There are three types of tools that aid in enforcing a consistent coding style within a team, and should be configured before coding begins.

Progress®

TOOL	EXAMPLES
Code linters	ESLint, CSSLint, Standardjs
Code style checkers	ESLint, Standardjs
Code editor formatting/style	.editorconfig

Create-React-App

Many developers using modern front-end frameworks can quickly become overwhelmed with all the setup and configuration required to get a simple application up-and-running. What used to take minutes in the early JavaScript and jQuery days now seems to require hours to get package managers, linters, task runners and testing tools all working together. To combat this tooling fatigue, the React team supports a tool called Create-React-App, a command-line utility that provides all the app set-up and configuration for you, so you can get to coding in minutes. It's an extensible tool that is perfect for most early projects, and you can "eject" into a full configuration at any point if your needs expand beyond what the tool provides, out-of-the-box.

Online Collaboration Tools

Online collaboration tools enable you to easily develop and test your ideas. Furthermore, you can easily share your code with others.

Here are a few tools to consider when choosing an online playground:

ONLINE COLLABORATION TOOLS
Stackblitz
Plunker

UI Components

Building any non-trivial web application is going to require you to create UI components beyond what the browser itself has to offer. Textboxes, labels and dropdown lists will only get you so far.

When it comes to UI components, there are a lot of solid options, both open source and commercial. The key is to pick a library that is built on React and uses React APIs, not one that merely wraps React. Wrapper libraries will provide a familiar syntax, but you often lose many of

Progress®

the performance and functional benefits of React when implementing these libraries.

TOOL	DESCRIPTION
KendoReact	Native React UI component library that includes grid, charts, dropdowns and many other customizable components.
Material-UI	An open source library containing many of the components needed to create applications which adhere to the Material Design specification.
Semantic-UI-React	Official React components for the Semantic UI framework. Designed to help create responsive layouts with "human-friendly HTML."
React-Bootstrap	React components for this popular CSS framework that is often used for application layout and its popular grid system.

Testing Methodologies

How you test, the tools you choose for testing and the ways you decide to implement tests is less important than the fact that you prioritize some form for testing in your application. It's likely that you'll want to test each module or unit of your code with unit tests. As you start to string code units together into a complete application, you'll want to consider functional, end-to-end testing. The list below contains some popular unit and functional test tools for React applications.

TOOL	DESCRIPTION
Jest	The Jest testing framework is a zero-configuration test framework that works out of the box with React applications.
Enzyme	Enzyme is a JavaScript testing utility for React that makes it easier to assert, manipulate, and traverse your React Components' output.
Karma	The karma test runner is ideal for writing and running unit tests while developing the application. It can be an integral part of the project's development and continuous integration processes.
Navalia	Navalia is an end-to-end test-runner and browser automation framework with a simple API and support for complex user interactions.

Progress®

Codebase Distribution Strategies

The days of building web-based applications solely for the browser are well-behind us. In today's day and age, it's possible to use web technologies to build desktop and fully-native mobile applications. Modern language interpreters and transpilers like Babel and TypeScript make this possible by converting the JavaScript we create into an Abstract Syntax Tree, or AST. An AST is a series of commands that describes our code, but which is written at a higher level than our code itself. ASTs make our code portable, meaning that other programs can take these AST representations of our web code and output whatever code is needed for another platform or target.

For example, NativeScript, a popular cross-platform native mobile application framework, uses an AST to convert JavaScript and TypeScript code into native code that delivers a fully native application experience.

For your own application, you need to consider both your initial target and any future platforms on which you'll want to serve your app.

Browser Only

If your app will only run in a browser, then your strategy is simple: deploy to a single server environment and your code will be served to the browser like a traditional web app.

Server-Side Rendering

Server-side rendering provides huge performance and SEO gains over solely rendering React applications from the browser. Because React rendering in the DOM is separate from the core engine, it's possible to render views on the server and merely send HTML to the browser to represent the initial state of the application. Once the server has rendered these initial payloads, React picks up on the client-side, hydrating JavaScript and application logic when the app is ready. Server-side rendering is simple in React with ReactDOMServer.

Mobile and Desktop

If you're considering targeting mobile devices or the desktop for your app, here are some tools to consider for leveraging your React codebase on non-browser platforms

Progress®

TOOL	DESCRIPTION
Electron	Build cross platform desktop apps with JavaScript, HTML and CSS.
NativeScript	Open source framework for building native mobile apps with Angular, TypeScript or JavaScript.
React Native	React Native lets you build mobile apps using only JavaScript. It uses the same design as React, letting you compose a rich mobile UI from declarative components.

Progressive Web Apps (PWAs)

Progressive Web Apps use modern web capabilities to deliver app-like user experiences, particularly on mobile devices. They evolve from pages in browser tabs to immersive, top-level apps, maintaining the web's low friction at every moment. Some of the key characteristics of PWAs include:

- Progressive - Work for every user, regardless of browser choice because they're built with progressive enhancement from the start

- Responsive - Fit any form factor, desktop, mobile, tablet, or whatever is next

- Connectivity independent - Enhanced with service workers to work offline or on low quality networks

- App-like - Use the app shell model to provide app-style navigation and interactions

- Fresh - Always up-to-date thanks to the service worker update process

- Safe - Served via Transport Level Security to prevent snooping and ensure content hasn't been tampered with

- Discoverable - Are identifiable as "applications" thanks to W3C manifests and service worker registration scope allowing search engines to find them

- Re-engageable - Make re-engagement easy through features like push notifications.

Progress

- Installable - Allow users to "keep" apps they find most useful on their home screen without the hassle of an app store

- Linkable - Easily share via URL and not require complex installation

React has a lot of features that make creating PWAs easy. For more information, check out this blog series by Addy Osmani.

Define Your Deployment Strategy

When you're ready to start getting your application closer to test, staging or production environments, you need to plan for regularly moving your code between environments. A continuous integration (CI) server is the ideal solution for managing your deployments, regardless of whether you intend to deploy to a live environment with every push.

Setting up for CI also improves your approach to local development, especially when you think about your CI approach from the start. Ideally, anything you do during CI you should first do locally to ensure that other developers on the team can easily replicate your setup, and that your CI system is properly configured to obtain dependencies, run tests and the like.

For React applications, I recommend considering one of the following CI tools:

CONTINUOUS INTEGRATION TOOLS
Travis CI
Jenkins
Codeship

JavaScript Error Monitoring

A JavaScript error monitoring tool should be used to capture runtime errors that occur in your staging and production environments. Typically, you won't use this tool in development once you get it configured.

This is one of the most commonly-skipped steps in the creation of web apps, but should not be overlooked. A quality tool, like one of the tools below, will save you countless hours tracking down production issues that are difficult to replicate in a local environment.

Progress®

ERROR MONITORING TOOLS
Track:js
Sentry
Raygun

Style Guide, Architecture and State Management

Defining a style for your team and project is essential, as is deciding the architectural styles you'll implement.

Thinking in React

As a mature application framework, React is very opinionated about the guidance it provides and the patterns you're encouraged to use. Before starting your React application, consider a careful review of the Thinking in React guide for some pointers, recommendations and common patterns to consider. Doing so will help you scale your app as it becomes more mature and well-trafficked.

State Management

State management is built into React via component internal state and can be shared between components via Props. For more complex state management, I recommend considering a library like Redux or MobX, both of with can easily be plugged into React applications and used to manage application state and mutations.

TOOL	DESCRIPTION
React State & setState()	Native React mechanisms for managing the internal state of a component and batching updates to view elements that depend on that state.
Redux	Robust state container that lives apart of React components and can help manage complex application state.
MobX	Simple state manager for React apps. Preserves some of the concepts of Redux, but is a bit simpler to start with.

Progress®

Backend API

When building your web application, you'll want to make sure you consider data storage and access from the start. Even if you're working with an existing data repository, I highly recommend wrapping that store in an API and taking an API-first approach to your development project.

API-first development means that you document, build and test your API first. The end result is a relatively-stable API before you write any dependent application code. This doesn't mean front-end development has to wait, however. During API construction, front end developers can build prototypes as early consumers of the API and provide valuable feedback to API developers.

The strongest argument in favor of API-first development is to reduce the chances that API bugs or weaknesses end up propagating into or are amplified by the data later. As much as possible, you don't want your front-end to have to bend over backwards to work around or mask deficiencies in your API later. Having a documented and solid API before a line of production code is written can go a long way to saving you headaches in the future.

So build your API first. Document it, test it and be ready to evolve it as you build against it.

A few key details to remember when taking an API-first approach is that security and authentication need to be embedded in the API, and that data environments should be kept separate. When developers are coding against the API, they should be working with development data, never live production resources.

Performance Strategies

It's worth investigating how to get the most out of your React application—from a performance standpoint—early-on in the development process. Let's investigate some ways that you can ensure that your app runs well once you get it live.

Polyfills and Browser Support

Modern frameworks like React owe some of their popularity to the fact that they allow you to use cutting edge JavaScript language features (usually referred to as ES6, ES7, etc. or ECMAScript 2016, 2017) without having to worry too much about browser support. This has certainly sped up language feature adoption and allowed the TC-39 committee (which oversees the ECMAScript standard) to move fast when it comes to shipping new features for the language.

Progress

That said, when targeting modern JavaScript language features and browser capabilities, you want to be sure to only load polyfills or additional code when needed by a browser, and not for every user of your application. Using the tools below, you can ensure that your app visitors on modern browsers get the fastest experience and native use of modern features.

TOOL	DESCRIPTION
Babel-preset-env	An npm extension to Babel that allows you to specify the browsers and versions you want to support; ensures that Babel transpiles your source to code required by the browsers you support.
Polyfill.io	A utility library that loads polyfills at runtime when a user visits your site or app. Polyfills loaded are only those needed by the browser, meaning that modern browsers won't be hit with the network cost of downloading code they don't need.
Core.js	Modular standard library for JavaScript. Includes polyfills for ECMAScript 5, ECMAScript 6: promises, symbols, collections, iterators, typed arrays, ECMAScript 7+ proposals and more.

Bundling

Bundling code allows us to remove unused or "dead" code and minify our builds before we deploy, as well as shrinking the overhead of that first set of JavaScript resources. Bundling tools also include capabilities to rename variables, functions and properties in order to obtain the smallest payload size possible that your servers will have to transfer over the network.

Tree-Shaking

Tree-shaking allows you to remove unused imports from your codebase, thus reducing the size of your bundle and the final assets shipped down to the browser. As long as you are using ES2015 module syntax in your React application (i.e. import and export), Webpack 2 and above, along with a minifier like UglifyJS, Rollup or Babel will take care of tree-shaking on your behalf.

Lazy-Loading

Lazy-loading is an approach that loads dependent modules only when you need them. For instance, an About component on a Home page would only be fetched when the page is accessed by a user. This keeps initial application payloads small and speeds up the loading experience of your app. While React does not provide lazy-loading capabilities by default in its core API, it is possible to use Promises or npm libraries to implement this functionality, should you decide to do so.

Progress / Kendo UI

Conclusion

React is one of the hottest new web technologies, and there is a rich ecosystem of tools and libraries to support it, including the KendoReact components. The strategies we've presented here for tooling, testing, performance, style and distribution will help you get started on the path to success. Once you have mastered the basics, there is a rich and diverse set of tools, libraries and methodologies for you to explore.

Progress®

About KendoReact

KendoReact helps you build better React apps faster. Our library includes a wide array of powerful components ranging from complex grids, charts and dropdowns to basic elements such as buttons and inputs. Kendo UI supports a selection of popular themes out-of-the-box and also includes an easy-to-use theme editor to allow you to quickly match any custom web theme. Take a look for yourself with a [free trial of KendoReact.](#)

Get started

About the Author
Brandon Satrom

Brandon is the founder of Carrot Pants Press, a maker education and publishing company, the founder and CEO of Tangible Labs and an avid tinkerer.

About Progress

Progress (NASDAQ: PRGS) offers the leading platform for developing and deploying mission-critical business applications. Progress empowers enterprises and ISVs to build and deliver cognitive-first applications that harness big data to derive business insights and competitive advantage. Progress offers leading technologies for easily building powerful user interfaces across any type of device, a reliable, scalable and secure backend platform to deploy modern applications, leading data connectivity to all sources, and award-winning predictive analytics that brings the power of machine learning to any organization. Over 1,700 independent software vendors, 100,000 enterprise customers and 2 million developers rely on Progress to power their applications. Learn about Progress at www.progress.com or +1-800-477-6473.

Worldwide Headquarters

Progress, 14 Oak Park, Bedford, MA 01730 USA
Tel: +1 781 280-4000 Fax: +1 781 280-4095
On the Web at: www.progress.com
Find us on (f) facebook.com/progresssw twitter.com/progresssw youtube.com/progresssw
For regional international office locations and contact information, please go to www.progress.com/worldwide