

# Planning an Angular Application

By Todd Motto

---

WHITEPAPER

# Table of Contents

Project Management / 3

Accessibility, i18n and Environments / 5

Development Process Methodology / 6

Tooling and Development / 6

Testing Methodologies / 11

Codebase Distribution Strategies. / 12

Mobile and Desktop / 13

Style Guide, Architecture and State Management / 16

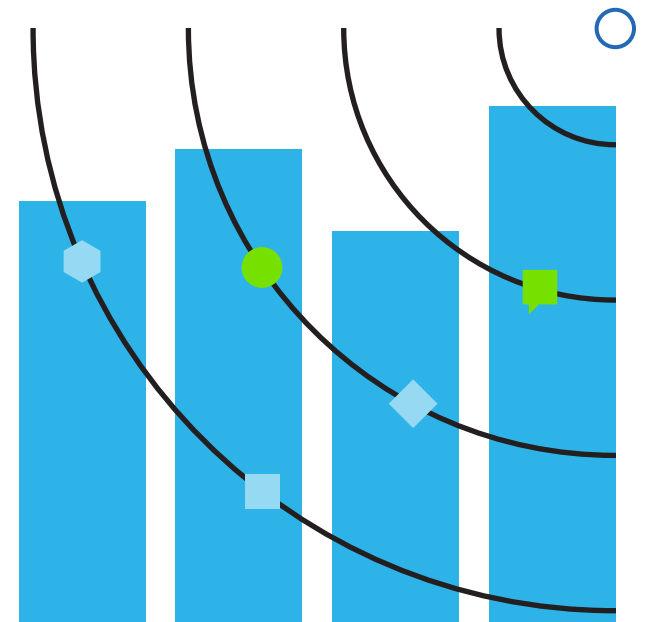
Backend API / 18

Performance Strategies / 19

Planning an Angular (version 2 and above) application is something you may have already done, or will be soon attempting. This whitepaper documents a high-level outline of things to consider when planning an Angular application, from tooling choices during development all the way through to deployment and performance strategies. There's certainly a lot more to it than meets the initial eye.

## Project Management

Before you get started, you need to consider how you're going to get the ball rolling - and keep it rolling. This usually starts with project management and discussing and agreeing upon particular toolchains to accomplish your next application.



## Software Management Tools

To manage the development of the front-end application, you'll minimally need to select the following software management tools to manage code, assets, and team members' tasks:

Ensure that you and your team adopt the tools you choose, and frequently assess and improve your workflow. New applications and tools are released to the public all the time and you may wish to address new tools that coincide with features or things you feel are missing - and you'll naturally adopt new tools as time progresses.

Software management tools	Examples
Issues and feature tracker	GitHub, BitBucket, JIRA
Version control system	GitHub, BitBucket
Document/asset storage	Slack, internal network storage, cloud
Team communication	Slack, HipChat, IRC, Google Hangouts
Task manager	GitHub Org Tasks, Trello, JIRA

# Accessibility, i18n and Environments

Accessibility, i18n (internationalisation) and building for the correct environments are an essential part of any application. It's not just deciding what to build, but how you're going to build and support it. Addressing these considerations at the beginning of a project will enable you to clearly vision how you'll implement the said features, such as accessibility concerns and i18n for example.

Software management tools	Examples	Links
Internationalisation / Globalisation	Translations for different languages Culture differences	<a href="https://angular.io/docs/ts/latest/cookbook/i18n.html">https://angular.io/docs/ts/latest/cookbook/i18n.html</a>
SEO	Yes, server-side render	<a href="https://universal.angular.io/">https://universal.angular.io/</a>
Browser support	IE10+	
Accessibility	WAI-ARIA	<a href="https://www.w3.org/WAI/intro/aria">https://www.w3.org/WAI/intro/aria</a>
Offline-first	•	<a href="https://developers.google.com/web/fundamentals/getting-started/primers/service-workers">https://developers.google.com/web/fundamentals/getting-started/primers/service-workers</a>
Progressive Web App	•	<a href="https://developers.google.com/web/progressive-web-apps/">https://developers.google.com/web/progressive-web-apps/</a>
Native Mobile App	•	<a href="https://docs.nativescript.org/angular/start/introduction">https://docs.nativescript.org/angular/start/introduction</a>

Above are some examples for consideration when deciding baseline standards and types of support your application can offer. These details may differ per project, for things such as i18n and offline strategies, it comes down to the goals of your project.

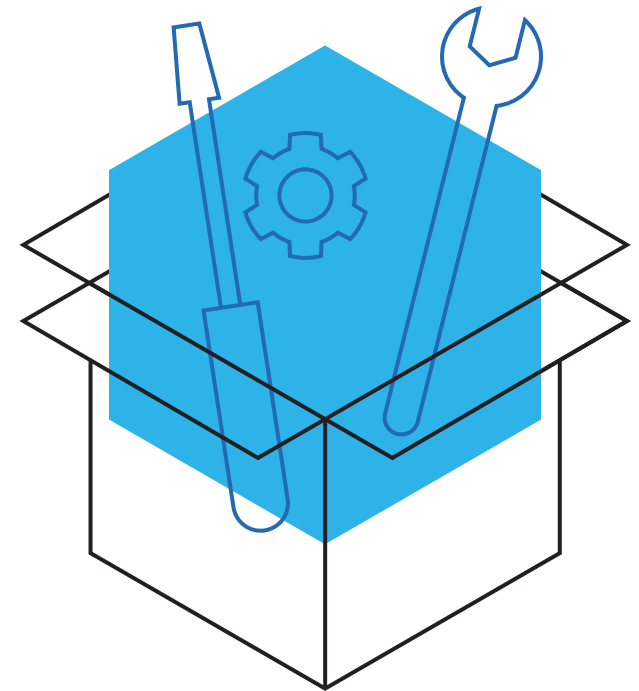
# Development Process Methodology

Typically there are a few different approaches to development, such as Agile, Waterfall, Scrum, Kanban and likely many more adaptations.

Whichever approach you take, it's important to remain consistent. The processes I've found to be ideal are the custom, loosely enforced, agile-like processes that can be wrapped around the uniqueness of the project and team members.

## Tooling and Development

Tooling has been increasingly important when developing any kind of application for the web or other platforms. With Angular, there are a vast amount of tooling options available. System.js was introduced first, however WebPack has seemingly become the de facto standard across the JavaScript ecosystem. Let's dive into some tooling a little further.



## Package Managers

Package managers allow you to grab dependencies from an external host, for example using npm to fetch your dependencies for development and also any dependencies that will make it into production.

An example of this would be using a development dependency such as TypeScript, which will never make its way into the browser as it's pre-compiled locally during development and for project builds before deployment. An example of a dependency that needs to make its way into production would be parts of Angular itself, such as Http, core and other modules.

Here are a few examples when considering a package manager.



## Task Runners

Task runners will allow you to configure particular tasks depending on what you're aiming to achieve. Managing third party code and their dependencies should not be a manual task performed by a human, it's not productive.

For example, you can use a particular command from a task runner to start a local server, compile all assets and then serve those assets in your browser with ease. WebPack has become the default standard with Angular as it can run your tasks, tests, compile your code and serve locally - as well as much more.





## Linters and Enforcement

When working on a team, the goal should be that each file is written as if it were coded from a single developer's mind in regards to error prevention, formatting, and style. Three main types of tools (i.e. code linters/hinters, code style checker, and a code editor config file) aid this process and should be properly implemented and configured before coding begins.

## Angular CLI

The Angular CLI will allow you to do most of the above, all in a single environment. Using the CLI will allow you to create new components, directives, services and more via commands in your terminal. You can also serve the app, run local servers, build and compress styles (Sass) and compile the application for best possible performance. I'd highly recommend checking it out and building applications with it.

Tools	Examples
Linters / Hinters	Codelyzer, CSSLint, ESLint
Code style checker	ESLint
Code editor formatting/style	.editorconfig

## UI Components

Building web applications means that you are likely going to require some additional 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 great options. You can choose either commercial or open-source components. The important thing is to pick a component library

which is built on Angular, not wrapped with it. If the underlying components are not built specifically for Angular, you will lose much of the advantages that Angular offers, such as Ahead of Time Compilation, tree shaking, server-side rendering and more.

Tools	Examples
<a href="#">Kendo UI</a>	Popular commercial components built specifically for Angular. Fully supported.
<a href="#">Angular Material</a>	An open-source library containing many of the components needed to create applications which adhere to the <a href="#">Material Design specification</a> .
<a href="#">Bootstrap</a>	A baseline CSS framework that is often used for application layout and it's popular grid system.

# Testing Methodologies

How you test and what you test is less important than the fact that you test something. It's likely the case that you'll want to test each module or unit of code by writing unit tests. When all of the units of code unite to form a running application, you'll want to do functional end-to-end testing. Below I detail the tools required (tasking tool facilitate all this) to do cross-browser unit and functional testing.

Tools	Purpose
Jasmine	The Jasmine test framework. provides everything needed to write basic tests. It ships with an HTML test runner that executes tests in the browser.
Angular Testing Utilities	The Angular testing utilities create a test environment for the Angular application code under test. Use them to condition and control parts of the application as they interact within the Angular environment.
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. This chapter describes how to setup and run tests with karma.
Protractor	Use protractor to write and run end-to-end (e2e) tests. End-to-end tests explore the application as users experience it. In e2e testing, one process runs the real application and a second process runs protractor tests that simulate user behavior and assert that the application responds in the browser as expected.

You can read more about Angular via the [documentation](#).

# Codebase Distribution Strategies

Gone are the days where we can just build an application purely for the browser environment. We're at the stage where, without necessarily knowing it, we're writing code in a format that can run pretty much nearly anywhere. Under the hood, language parsers such as Babel or TypeScript convert our code into an AST (Abstract Syntax Tree). An AST is a chain of commands that describe our code, at a higher level. This means that we're not limited to the original language it was written in. People can then (and already have for most cases) write programs that interpret these ASTs and spit them out in whatever language is needed.

Via an AST, things like [NativeScript](#) exist to transform that AST into native code on mobile for impeccable performance and native experience.

For your application, you need to consider the initial environments you'll be deploying to, as well as any future considerations - such as NativeScript for native mobile applications (it'll compile your Angular code for you, reusing 90%+ on average of your existing codebase).

## Browser only

If your application will only run in a browser, then your strategy will be fairly simple. You'll be able to deploy to a single environment and the code will run in the browser like any other web app that's "browser only".

## Server-side rendering

Server-side rendering has a huge performance and SEO benefit to loading an Angular application directly in the browser. Because the Angular rendering core is split from the framework itself, we can essentially render a view on the server. Once the server has rendered the initial payloads, the client-side part of Angular can pick up where the server left off, hydrating it with JavaScript once Angular is ready. For this, you'll need [Angular Universal](#).

# Mobile and Desktop

Let's talk about mobile and desktop. Here are some considerations when thinking about using your Angular codebase on non-browser platforms.

Tools	Purpose
<a href="#">NativeScript</a>	Open source framework for building truly native mobile apps with Angular, TypeScript or JavaScript.
<a href="#">Ionic</a>	Open source framework for hybrid apps, Angular + TypeScript.
<a href="#">Electron</a>	Build cross platform desktop apps with JavaScript, HTML, and CSS.

## Progressive Web Apps (PWA)

Progressive Web Apps use modern web capabilities to deliver an app-like user experience. They evolve from pages in browser tabs to immersive, top-level apps, maintaining the web's low friction at every moment. Here's some further information as to characteristics of PWAs:

- Progressive - Work for every user, regardless of browser choice because they're built with progressive enhancement as a core tenant .
- 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.
- 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.

Angular makes these much more easy to integrate, and I'd encourage you to check out [Angular Mobile Toolkit](#).

Or check out this four-part blog series explaining [the basics, intricacies and importance of PWAs](#).



## Define deployment strategy

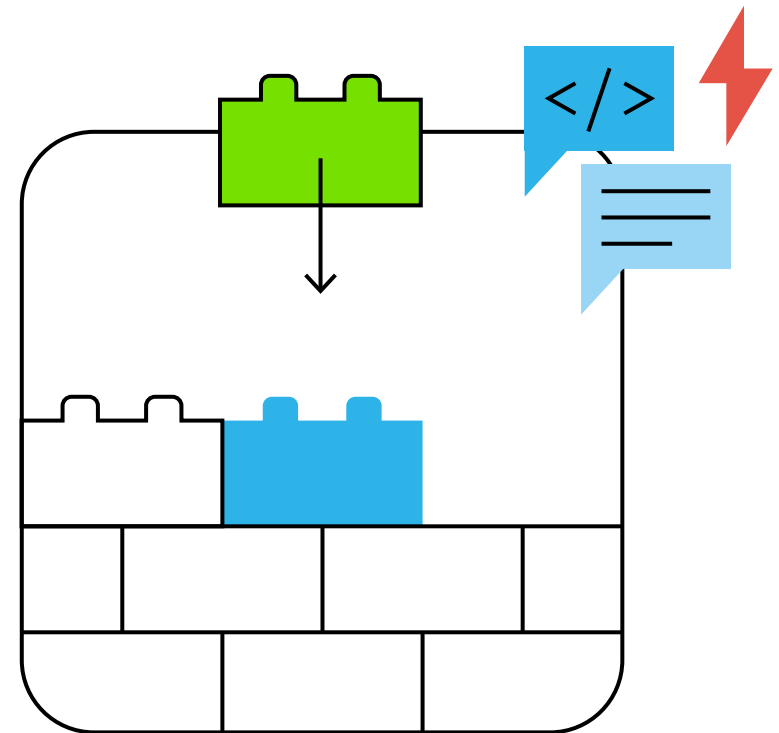
You need a plan to get your local code to staging and production (i.e. deploy local code to public servers). After all, not everyone can always see your locally running application, even if you use some magic to make it happen. A continuous integration server is an ideal solution for crafting deployments regardless of whether you intend to deploy to production on every code push.

Let me step back for a moment from CI concepts and talk about local development. Ideally, anything you do during CI, you should be able to do locally. That means, building and testing your application should be crafted first to run locally. In fact, I will often run a second server locally that serves staging code on my local machine (i.e. what gets outputted during CI process). It's this local testing and build process that becomes automated once you set up your CI integration. I've loosely mapped out below what this might look like.

## JavaScript error monitoring

A JavaScript error monitoring tool needs to be selected to capture run-time errors occurring in staging and production code. Typically, this monitoring tool is not used on development code. Pick one, use it.

This step is probably the most commonly skipped step in building JavaScript applications. Don't overlook capturing runtime errors.

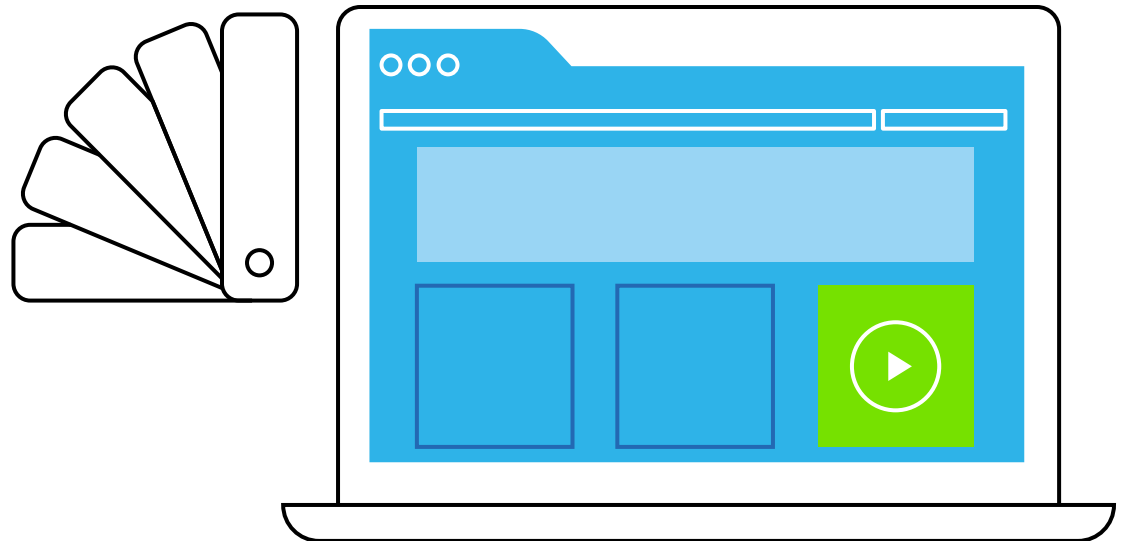


# Style Guide, Architecture and State Management

Defining a style for your team and project is essential, as is deciding what architectural practices you'll be using.

## Angular style guide

[The Angular style guide](#) should be a go-to consideration to get familiar with architectural and style best practices before diving into any Angular project. It offers a lot of help towards sensible app structure, offers common “gotchas” and recommendations for you.





## State management

State management comes easy to Angular with uni-directional dataflow, however using a Redux approach to Angular development is something to highly consider.

Tools	Purpose
ngrx/store	RxJS powered state management for Angular applications, inspired by Redux
ngrx/effects	Side effect model for @ngrx/store.
ng2-redux	Ng2Redux lets you easily connect your Angular components with Redux, while still respecting the Angular idiom.

# Backend API

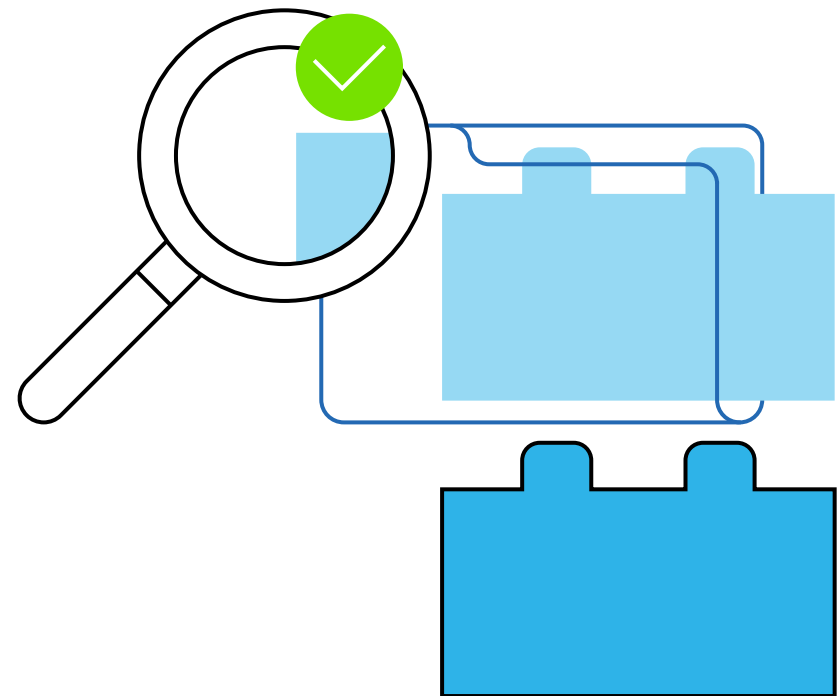
The first step assumes API-first development, which is an excellent method that I highly recommended.

In a nutshell, API-first development means that you document, build, and test your API first. This means you have a relatively stable API before you write any application code. Note that during API construction, front-end developers should be prototyping minimal viable features using the API and providing feedback to the API engineers.

The main reason to follow API-first development is to reduce the possible deficiencies in the API from being amplified in the data layer. You want to do everything in your power up front to avoid having to make up for your API's deficiencies in your application logic. Having a documented and mostly solidified data API before a line of application code is written can go a long way towards reducing pain and misery in the future. Build your API first. Document it, test it, and

then be ready to evolve it as you build out the applications that use it.

It's worth noting that it may be assumed that security and authentication details will accompany the API. It is also assumed that the API developers will provide a development API to be used for development. Using the same data API for both development and production should never be an option.



# Performance Strategies

It's worth investigating how to get the most out of your Angular application before you've even set foot in the codebase. Let's investigate some approaches.

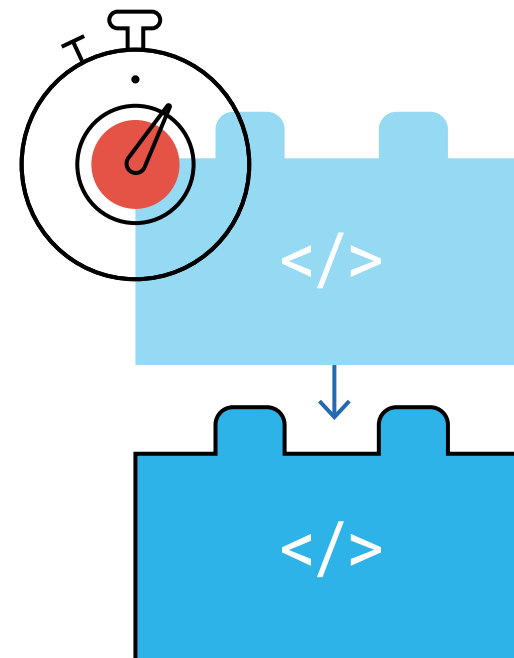
## Ahead-of-Time compiling

Around half of Angular's codebase makes up the internal compiler, which you can absolutely strip out with offline compilation called Ahead-of-Time compiling. This can help you achieve very

small payloads that, combined with module lazy-loading, can significantly improve performance.

If you don't AoT compile, you'll ship the compiler to the browser, which means the codebase is heavier, thus by default you adopt a "Just-in-Time" compilation strategy. The AoT approach is also similar to React's story with JSX - it's all preprocessing.

Read more about [AoT](#).



## Bundling

Bundling code allows us to eliminate dead code and minify our build before we deploy, as well as minimising payload overheads when the browser requests the initial JavaScript file. This also includes code mangling to rename variables, functions and properties to get the minimum payload size possible.

## Tree-shaking

Tree-shaking allows us to remove unused imports from our codebase, thus (potentially) reducing the bundle drastically in size. The Angular team have put together a guide on tree-shaking that walks you through it using [Rollup.js](#). [Webpack](#) supports tree-shaking for ES2015 modules as well.

## Lazy-loading

The Angular router allows lazy-loading of feature modules, which essentially will allow you to request large module chunks on-demand rather than include it any initial payloads. It can be removed before deployment and requested before accessing a particular route that a user may not have visited yet.

# Brought to You by Progress Kendo UI

We're engineering true Angular UI components, not just wrapping existing components like other vendors. We're dedicated to delivering pure, high-performance Angular UI components without any jQuery dependencies because we won't settle for anything less and we don't think you should either. [The Components page](#) contains the most up-to-date list of included UI components.



## About the Author

Todd is a front end engineer from England, UK. Runs Ultimate Angular, Founder of Voux, Developer Expert at Google, conference speaker and open source evangelist.

Try Kendo UI for Angular

## 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 1700 independent software vendors, 80,000 enterprise customers, and 2 million developers rely on Progress to power their applications. Learn about Progress at [www.progress.com](http://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](http://www.progress.com)

Find us on  facebook.com/progresssw  twitter.com/progresssw

 youtube.com/progresssw

For regional international office locations and contact information, please go to [www.progress.com/worldwide](http://www.progress.com/worldwide)

Progress and Kendo UI are trademarks or registered trademarks of Progress Software Corporation and/or one of its subsidiaries or affiliates in the U.S. and/or other countries. Any other trademarks contained herein are the property of their respective owners.

© 2018 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.

Rev 18/10 | 171002-0040 / RITM0029821