GrapeCity®

DESKTOP TO WEB:

MIGRATING FROM WINFORMS TO JAVASCRIPT

A migration guide for development teams building enterprise applications.

Table of Contents

Migrating from WinForms to JavaScript

Many companies today are moving from desktop software to web applications. As part of that process, desktop developers sometimes find themselves scrambling to learn how to build enterprise applications with a new technology stack — specifically the modern JavaScript ecosystem.

Building enterprise applications with web technologies is the future. In the long run, you have more options for development tools and technologies. Your ability to quickly deploy fixes, new features, and entire apps is greatly simplified. And web technologies lets you easily deploy to almost any desktop or mobile platform from a single codebase.

Web development has been on the rise for many years, and it is now overtaking desktop apps in both market share and richness of features.

Web development has been on the rise for many years, and it is now overtaking desktop apps in both market share and richness of features. When an enterprise begins a new development project today, the web is usually the target platform. But the library and framework options available from the JavaScript ecosystem can be overwhelming. Common questions for many organizations are: Where do I start evaluating a new technology stack? What is our best migration path from desktop to JavaScript?

This is a guide for development teams and software dev leads to find the best approach to migrating. We want to help guide you and give you a structured way of bringing existing Windows Presentation Foundation (WPF) and WinForms apps to modern web technologies.

Here, we will explain the issues you should consider when migrating from desktop to the web, then walk you through the key elements of our recommendation as to the best path today: **Angular and TypeScript**.

Desktop Enterprise Applications

WinForms applications has been the framework of choice for desktop development for many years. There are several good reasons: the dominance of Windows on enterprise desktops, rich development tools, a vast third-party control ecosystem, and the perceived security of self-contained applications inside company firewalls. With WinForms, desktop developers could build feature-rich applications quickly with the power of Visual Studio and the .NET Framework.

WinForms became the desktop platform of choice for data-entry and utility applications, replacing many legacy mainframe applications in the years following the Y2K scramble.

The introduction of WPF in the mid-2000s provided Windows desktop developers with a UI platform that could create more visually appealing apps. While WinForms had that classic Windows, utilitarian look-and-feel, WPF let developers style UI elements to have almost any appearance.

WinForms remained popular for building simple, stand-alone utilities and in many client/server and 3-tier architected systems. WPF developers adopted the Model-View-ViewModel (MVVM) pattern to create unit-testable apps with more complex, rich UIs, sometimes backed by web services.

Shifting to the Web

As business users began to use smartphones, enterprises were presented a choice to support a mobile workforce: create native mobile apps for their users or shift to the web. Desktop apps were not an option on mobile devices, and no desktop development frameworks have seen long-term success directly migrating to cross-platform web or mobile use.

Building applications with web technologies such as HTML, CSS, and JavaScript gives companies the ability to create both a mobile app and a web app. First, web apps can be used on any platform with a browser. (Headache-inducing problems like juggling browser compatibility are mostly a thing of the past today.) But second, the majority of popular, modern cross-platform mobile development frameworks use HTML, CSS, and JavaScript.

Enterprises that choose to stay with their desktop apps and create native mobile apps are forced to maintain separate code bases for desktop, Android and iOS. They may share some business logic, services, and data (assuming the legacy desktop app was not a tightly coupled monolith), but each will need individual client apps.

Companies weighing these factors are overwhelmingly choosing to start building with web technologies. Modern web applications leverage components and modules for maximum code reuse and scalability. Staffing is also easier when you can draw from a huge base of experienced web developers.

The rise of web development also brought about advances in DevOps practices such as Continuous Integration (CI), Continuous Delivery (CD), microservices, and the ability to rapidly build, test, and deploy web applications using commodity cloud infrastructure.

Understanding The Web Ecosystem

Most desktop applications are complex beasts behind the code and don't seem like they could be easily recreated in a web app. But in reality, we're usually just translating from one set of patterns to another. In many cases the patterns are the same, just called something else.

Migrating to web technologies can seem more complex than it really is. For all the apparent confusion of libraries and frameworks, it's really pretty simple. The early web brought many limitations to developers: issues like limited memory, performance limitations for complex scripts, and browser compatibility issues. Web apps running in browsers are sandboxed with no direct access to client file systems, printers, or local networks.

However, each unit of web app development is simply HTML, CSS, and JavaScript — and in the case of server-side functions or some frameworks, just JavaScript.

Today developers write server-side JavaScript functions with Node.js, use mobile frameworks, and their work is abstracted from differences in browsers by current JavaScript frameworks and libraries.

With all of these frameworks and libraries to choose from, which one is the best for today's enterprise? The choice can be overwhelming. There seems to be a new, hot JavaScript library or framework release every week. How can anyone know which framework is the right one, and that the framework chosen today will continue to be supported 3-5 years from now?

Our Recommendation: **Angular and TypeScript**

Our guidance for migrating your team from WPF or WinForms development to a web technologies revolves around two central concepts: adopting a language and runtime that leverages your team's existing knowledge and toolchain, and choosing a solid framework with a proven track record and broad industry support.

Our recommendation is to take a look at TypeScript as the scripting language and Angular as the web framework.

TypeScript is a superset of JavaScript. What does that mean? It means that all valid JavaScript code is also valid TypeScript code.

The TypeScript language was first released publicly in 2012 and reached version 1.0 status in 2014. It was created by Anders Hejlsberg at Microsoft, where it is developed as open source. Hejlsberg also created the C# language. TypeScript continues to mature and, as we write this, is at version 3.5.

TypeScript takes JavaScript and builds on it by adding features that make it an object-oriented language like C#. It is becoming the language of choice for large-scale web development projects. In fact, beginning with version 2.0, Angular itself is built on TypeScript.

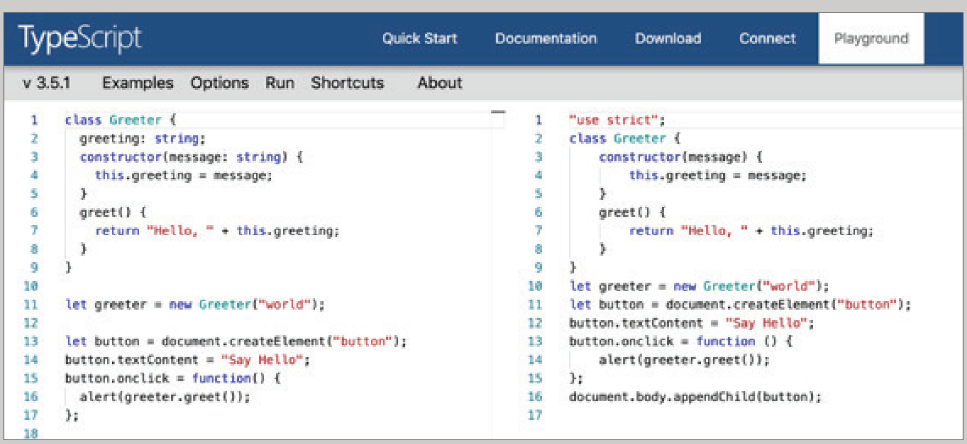
TypeScript is a strongly-typed language. Traditionally, JavaScript was dynamic and had a very loose type system. This has been changing and is becoming closer to the type system in TypeScript since the introduction of the ECMAScript 6 (ES6) specification of JavaScript. TypeScript, as the name implies, adds a strong type system. Developers can also define their own classes and interfaces. The type system provides developers with compile-time checking and IntelliSense while coding.

Here is an example of a class defined and used in TypeScript:

```
class Car {
  color: string;
  constructor(color: string) {
    this.color = color;
  }
  Honk() {
    return “The “ + this.color + “ car honked.”;
  }
}
let car = new Car(“green”);
let sound = car.Honk();
// sound will contain the string “The green car honked.”
```

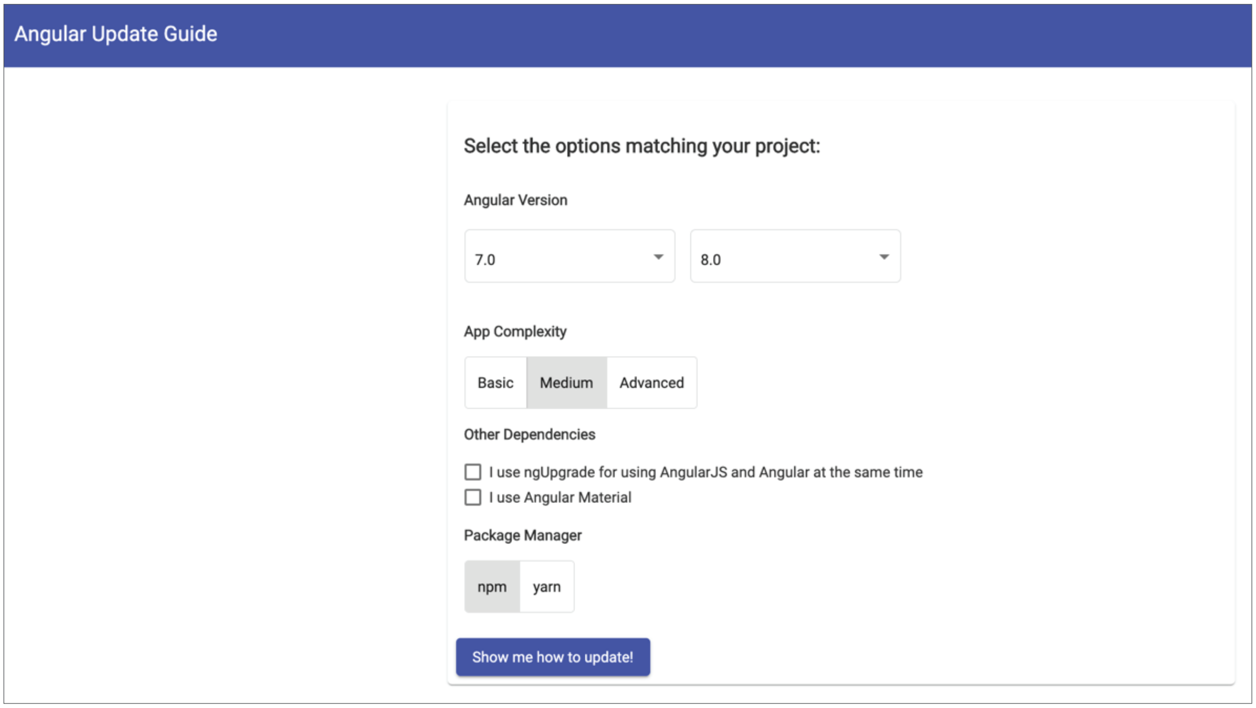
The syntax looks like JavaScript. It compiles to pure JavaScript, but it should also feel familiar to C# developers. The type system is similar to that in .NET, which will ease the transition to web development for veterans of desktop development.

Want to try TypeScript before installing it? Visit the TypeScript Playground. There you can write TypeScript in the editor and see it compiled to JavaScript in real time.
There are some great examples to get started with TypeScript.



Desktop WPF and WinForms developers write most of their business logic in C# targeting the .NET Framework, which provides a rich set of libraries and helpers. When migrating to web technologies, developers are going to need a comparable JavaScript framework to provide the same kinds of libraries and helper functions.

That's where Angular comes into play.



AngularJS was first released in 2010 as an open-source JavaScript framework. AngularJS released several iterations before being completely rewritten in TypeScript and released as Angular 2 in 2016.

The current version of Angular as we're writing this is version 8.2.0, and version 9 is planned for release before the end of 2019. As you can see, the Angular team has been iterating quickly since the launch of Angular 2. While the update from AngularJS to Angular 2 introduced many breaking changes (enough for many web developers to re-write their apps), each release beyond version 2 has seen far fewer significant changes.

Angular has an upgrade guide to help web developers update their applications when changing versions of Angular. Enter a source and target version of Angular to see a list of recommendations when updating.

While migrating a well-architected WPF or WinForms application to an Angular app will involve writing new client-side code, many desktop controller layers will logically translate to Angular components. We will talk more about Angular components, modules, and other design concepts in a later section.

So, what's the best way to get started with Angular and TypeScript and plan your application's migration?

The Angular/TypeScript Toolchain

Let's start the discussion of an Angular application's toolchain with a comparison of the desktop and web application toolchain components.

	WinForms	WPF	JavaScript
IDE/Editor	Visual Studio	Visual Studio	Visual Studio / VS Code
Language	C# or VB	C# or VB	TypeScript
Framework(s)	.NET Framework	.NET Framework	JS Frameworks
Package Manager	NuGet	NuGet	npm
Controller Logic	Components or Controllers	ViewModels	Components
Styling	Property based styling	XAML Styles	CSS
Data Access	SQL or Web Services	SQL or Web Services	Web Services (XML, JSON)

Notice how many of the concepts and tools are exactly the same or at least very similar. This is why so many developers find that the move to web development can be a relatively smooth transition.

For developers who love building applications in Visual Studio today, there is no reason to switch IDEs. There are a few ways that Visual Studio 2017 or later can be used to develop with Angular.

First, Visual Studio can create a new ASP. NET Core Web Application and select the API template. This will create a project with WebAPI controllers and an Angular front-end.

Second, developers can install and use the Node Package Manager (npm) and the Angular CLI to create a new Angular application. After the application has been created, Visual Studio can open the folder containing the Angular app. Visual Studio 2017 and later can open a folder instead of a project or solution file.

VS Code has extensions for nearly every development language and framework. It also recently introduced Extension Packs, which are bundles of related extensions. Install an Extension Pack for your framework of choice to get up and running quickly. We recommend the Angular Essentials pack by John Papa, which includes extensions for Angular Language Service and Angular Snippets, TSLint (a TypeScript linter), and Angular console, Angular inline templates and stylesheets, and much more.

Web applications can use a number of different JavaScript frameworks. While Angular is the primary framework, another installed by default when creating a new application with the Angular CLI is Jasmine for unit testing your components.

.NET Framework developers are probably familiar with the NuGet package management system. The package management system used by most web developers today is Node's npm. After installing npm, packages can be installed at the command line with a simple command like:

```
npm install <package-name>
```

By default, packages are installed locally to each project, but you can also install a package globally.

Angular applications use CSS familiar to all web development by default when creating a new application, but other options, including LESS and SASS, can also be used. CSS3 is a rich, powerful styling system that can apply style to any HTML element and supports inheritance and multiple methods of linking styles to visuals. When used in Angular, styles can be applied from the CSS file for the current component or inherited from a parent component, module, or application-wide style.

Data Access in Web Applications

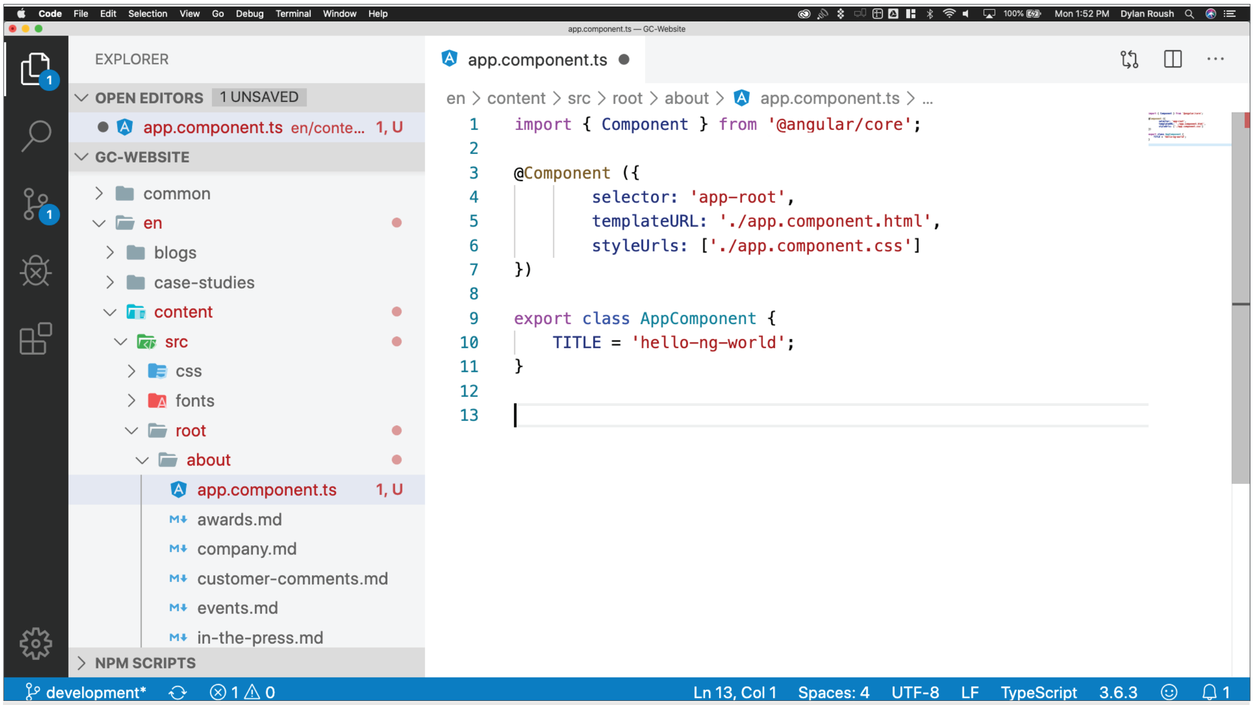
Desktop developers build apps in an environment where the entire workstation and network are available when reading data. When running on the client-side within a web browser, there are fewer options for data access.

We recommend building web services to return data formatted as XML or JSON. Visual Studio developers will feel the most comfortable building ASP.NET WebAPI projects for this purpose.

In Angular, web service calls will be wrapped in functions inside "Service" classes. These Service classes can then be used across components in the library.

NgRx (Reactive State for Angular) is a powerful library used commonly inside Angular service classes. .NET Framework developers who have used Microsoft Reactive Extensions will find that NgRx concepts feel familiar.

Already have web services that return XML or JSON to your desktop application? There is a good chance you can re-use those in your Angular application.



While Visual Studio is the primary tool for Windows desktop developers, Microsoft released a new lightweight editor called Visual Studio Code (VS Code) in 2015. Thanks to the speed, flexibility, and extensibility of VS Code, it has become the editor of choice for many modern web developers on every platform (VS Code runs on Windows, Mac and Linux). Here is a new Angular project in VS Code on a Mac.

Starting the Migration Process

Because the scope of moving an enterprise application from the desktop to the web is non-trivial, often spanning an organization's release cycles or even fiscal years, it is best to plan a phased migration. Migrating each layer of the application in separate phases is a good logical separation of work.

By starting with the data layer, moving next to the controller (business logic) layer, and finally finishing with the UI layer, developers can minimize the pain of a large scale migration. If the desktop application has these existing logical layers, it can be updated to leverage the migrated layer as each phase of the migration completes.

The layers that will be executing on the servers, in this case the data and services, are referred to as the back end of the application by web developers. The UI layer and its controllers are known as the front end.

Angular Libraries, Modules, and Components

Angular breaks applications down into libraries, modules, and components. Think of a library as a .NET assembly or DLL. Angular libraries can be versioned and shared among multiple Angular applications, containing one or more modules and components.

An Angular module is a grouping of related components in the application.

A component may correspond to a screen in the web application, which in turn will be composed of components for each of the composite controls on the screen. WPF developers can think of Angular components as similar to View Models and Angular modules as user controls grouping multiple, interconnected View Models.

Each Angular visual component consists of an HTML file, a TypeScript file, and a CSS file. The HTML contains definitions of the visual elements, the TypeScript contains the logic, and the CSS contains the style to be applied to the HTML elements.

Migrating the Data Layer

In most migrations, the data layer migration will require the least amount of change to support a web app. The data layer for Windows desktop applications in many enterprise environments is usually a SQL Server database, possibly with a large collection of stored procedures.

The good news is you can keep your database.

If the database is running in a local data center, you can leave it untouched (provided the web services will be running on a server with access to the database). You might take the opportunity to migrate to a cloud provider like Microsoft Azure. This decision is something that should be evaluated during migration planning.

Instead of querying data directly from the desktop application, the web services in the back end (controllers) will be querying the data and returning it to the UI. Let's take a look at the difference.

If the desktop application already uses web services for all data access, this will streamline the migration as this first step is already complete!

If the desktop application queries data directly from the client workstation, but has a good separation of the business logic and data access layers, much of this data access code can be ported or re-used directly on the back end.

Migrating the Controller (Business Logic) Layer

The controller layer is where the real work in the application takes place. This is where the business logic is typically hosted. Desktop developers typically create their business logic modules in C# or Visual Basic.

When migrating to the web with Angular, these could be migrated to TypeScript components. An alternative approach would be to keep the .NET code running on the server instead of rewriting in TypeScript. This would keep the code closer to the data layer and potentially incur less work.

There are several strategies for migrating desktop business logic to TypeScript, depending on the existing design and the availability of documentation and tests.

If the layer is well designed using the SOLID principles, you can follow some of these guidelines for translating .NET Framework concepts of logical separation to TypeScript.

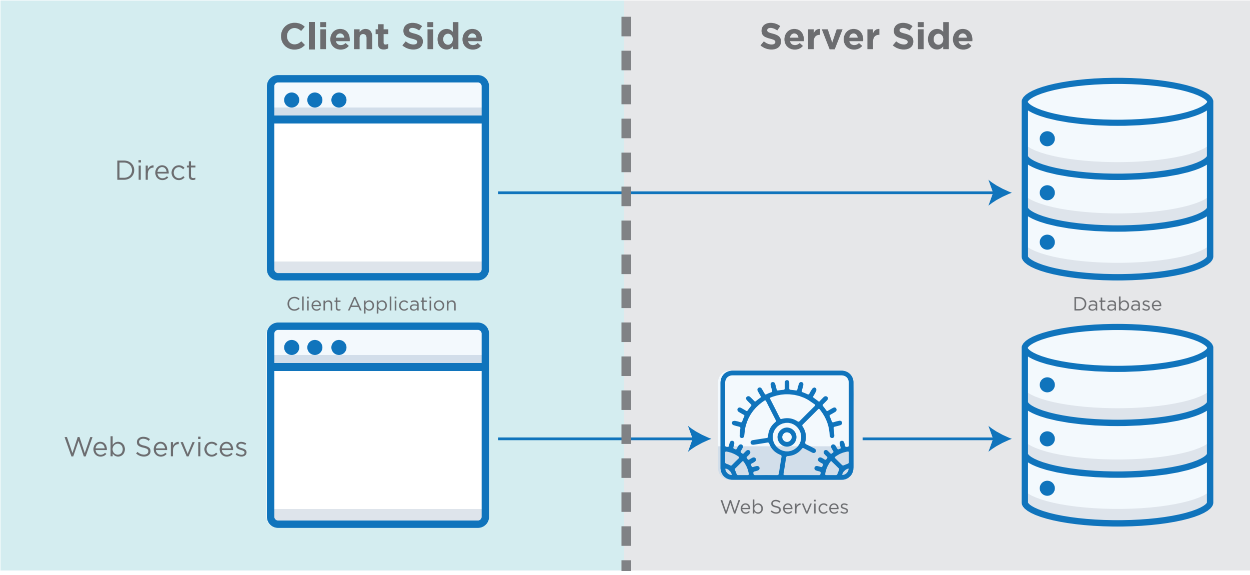
- *When translating .NET assemblies to TypeScript libraries, these libraries can be packaged and shared across applications, like assemblies. They can also be shared publicly as npm packages, similar to NuGet packages.*

- *.NET namespaces can be translated to TypeScript modules or namespaces. Modules in TypeScript are logical groupings of components that share state and are available externally. Namespaces are modules that are only available internally.*

- *.NET classes can be translated into TypeScript classes. There can typically be a 1-to-1 translation here, barring any refactoring.*

- *.NET interfaces relate directly to TypeScript interfaces. TypeScript supports the concept of interfaces, and they are used commonly in Angular applications.*

Data Access



If the existing controllers have unit tests, these should be migrated to JavaScript unit tests with the corresponding class under test using the Jasmine test framework. Tests can execute via npm directly or while building the application.

When actually creating the TypeScript code, it is possible to convert some simple C# classes. There is a Visual Studio extension called TypeScriptSyntaxPaste that is a time-saver when migrating some simple business logic, DTO classes, and interfaces.

Migrating the UI Layer

Desktop application developers who build UI components are accustomed to having a visual designer available in Visual Studio to lay out the UI. WinForms and WPF have robust designers in Visual Studio. JavaScript UI frameworks do not have the benefit of visual designers to rapidly lay out and visualize the UI when working in VS Code. However, there are a few options for visually designing components.

Your .NET UI components and controls will be replaced with Angular components. Many of these will be created with simple HTML controls such as buttons, checkboxes, inputs, and datepickers. Controls can also come from third-party libraries like Wijmo. These controls, like .NET controls, save you the time of writing them from scratch.

To create a modern-looking, consistent UI design, Angular applications can leverage a UI library like Bootstrap or Material to style the UI components. Most native HTML and third-party components are supported by these styling libraries to provide a strong foundation of UI design on which to build. Both Bootstrap and Material support theming to easily apply company branding to the application.

CI is the practice of multiple developers contributing to a source code repository for an application and merging their changes into a single master branch or repository.

Design Resources

Here are three top design resources for your JavaScript conversion project.

- Storybook creates individual UI components for Angular, React, or Vue-based applications in the browser. It is open-source with several add-ons and tutorials available.
- Wijmo Designer for VS Code is available to visually create Wijmo components for Angular.
- Sketch is a popular application for designing web UI components. It is not a free tool, but it is powerful and widely used.

These merged changes are compiled, and automated test suites are run against them to validate the changes.

Build and Deploy

One of the benefits of moving to the web is the relative ease of deploying releases. Desktop applications must be deployed across many client workstations. Web applications are deployed either to a web server. This ease of deployment has led to the rise of Continuous Integration/ Continuous Delivery (CI/CD) and modern DevOps practices.

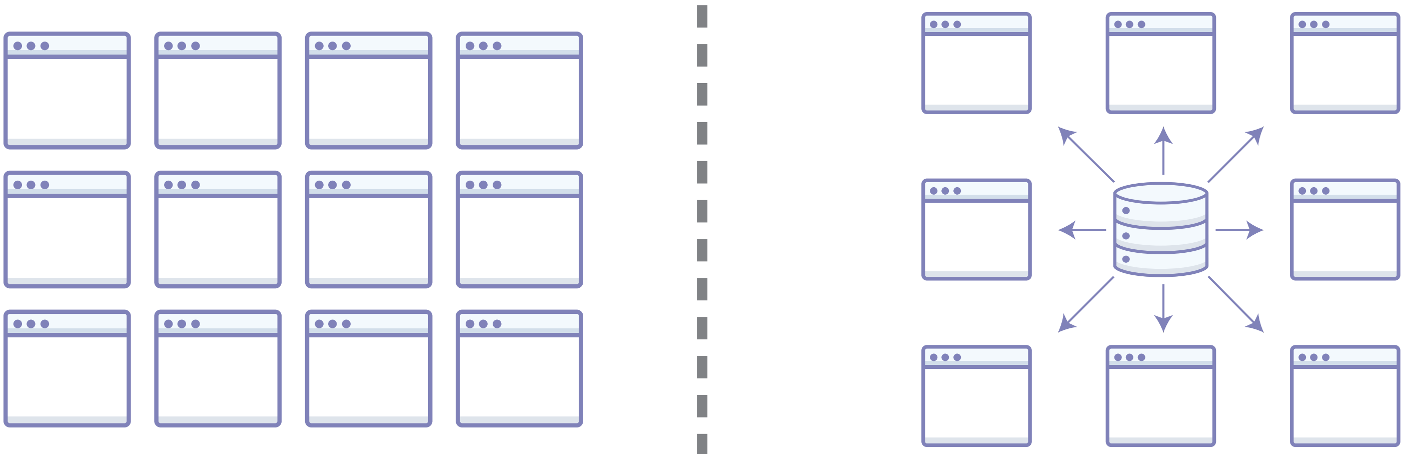
CD is the practice of frequently releasing and deploying the application. This is usually a staged deployment (Dev, Test, Production) to allow manual and automated test suites to be executed before deploying to production.

When using a cloud service like Azure DevOps Services for CI/CD, these steps for integration, build, test, and deployment can be automated and orchestrated in a way that would be much more challenging with desktop application build-and-deploy processes.

Cloud service providers like Azure or AWS can also make it easier to automatically scale different layers of the application as needed. The UI, web services and database can each be scaled up or out to keep the application performing smoothly during times of peak demand.

Another method of designing services for scale is by creating microservices. Microservice architecture is achieved by separating each service into its own self-contained component that can be deployed and scaled independently. Removing external dependencies and their stateless nature are the keys to this independence.

Traditional Development vs. CI/CD Methodology



The Future

It is well known that no technology lives forever, especially web development technologies. So, what does the future hold for web developers? Will migration to the web become easier?

WebAssembly

WebAssembly (or Wasm) is native compiled code that runs in modern web browsers. Wasm 1.0 is now available and runs on all of the major desktop browsers including Chrome, Firefox, Edge, and Safari. That means that languages like C++, Rust, and Java can be compiled to download and run inside the browser on the client machine. Even .NET Framework languages can be run in the browser with Wasm and Blazor.

This will mean that developers can continue to write web application code in the programming language they know best. As Wasm matures, it seems likely that these applications will out-perform scripted languages like JavaScript and TypeScript while maintaining the security and safety of sandboxed web apps in the browser.

Blazor

Blazor is Microsoft's framework for building .NET UI components in Wasm. This means that all server-side and client-side application code can be written in C# rather than JavaScript.

Does this mean that you can take all of your desktop .NET Framework components and re-compile them targeting Blazor? Unfortunately, no. Blazor is part of ASP.NET Core 3.0, so all Blazor components must target .NET Core 3.0. This also means that Blazor components are not quite ready for production applications as .NET Core 3.0 is still in beta and is targeted for a late 2019 release. However, the migration from .NET Framework to .NET Core will be less painful than a move to a JavaScript framework.

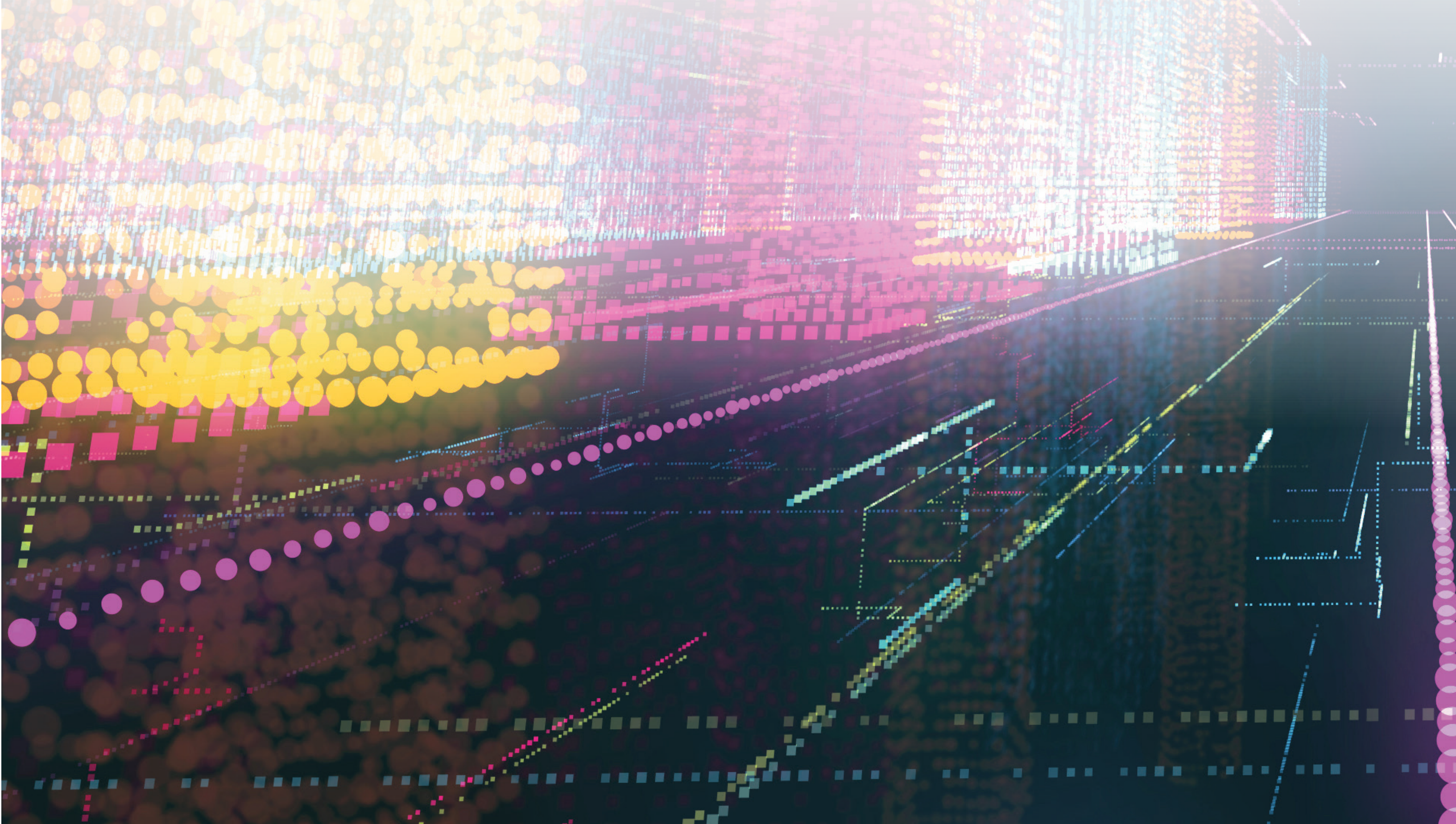
Wrapping Up

As you've seen in this guide, building new applications or migrating enterprise applications to use web technologies is the future. The technology available is diverse and mature, and your ability to quickly develop and deploy fixes, new features, and entire apps is becomes much more streamlined.

Hopefully this guide gave you an overview the best approach to migrating your development process from WPF and WinForms apps to modern web technologies.

Our recommendation as to the best path today is to take a close look at Angular and TypeScript.

While technologies like WebAssembly and Blazor may open up new opportunities to re-use C# code down the road, they are still immature. Starting to adopt technologies like Angular, TypeScript, and JavaScript today will open a world of opportunity.



The transition from desktop to web applications can be overwhelming, especially if you're new to the modern JavaScript ecosystem. With a multi-faceted approach to addressing developer-specific needs, our offerings continue to expand and address the needs of the .NET, JavaScript mobile, and web developer.

If you are considering this migration, you can future-proof your apps with cross-platform UI controls.

Our ComponentOne.NET offerings continue to grow and address the needs of .NET, mobile, and web developers. GrapeCity also offers Wijmo, a suite of 60+ JavaScript UI controls with all ComponentOne subscriptions. We offer these tools as a package, along with a mixture of resources written by developers to help layout the best framework and tools to simplify the migration process.

TRY WIJMO TODAY