* Two Players, One Goal
  + The web API interface provided by ASP.NET Core is a programmatic set of server-side tools
    - This is used to expose hooks/endpoints to a defined request-response system
    - Typically, this is done through structured markup languages (XML), language-independent data formats (JSON), or query languages (GraphQL)
    - This is achieved by exposing application programming interfaces (APIs) via HTTP/HTTPS protocols through a publicly available web sever (Node.js, IIS, etc.)
  + Angular is a modern, feature-rich, client-side framework
    - This pushes HTML and ECMAScript’s features and browser’s capabilities to their full extent
    - This is done by binding HTML input/output areas into a flexible, testable, and reusable model
* The ASP.NET Core Revolution
  + ASP.NET and the .NET framework more generally have undergone drastic changes in tune with MS’s changes to become a driving force for open source
  + The 2014 MS Build conference was when MS first announced public release of the .NET compiler (Roslyn) and the start of the non-profit .NET Foundation
  + Since that conference, MS has published numerous large, open-source projects (e.g. RyuJIT, MSBuild, VS Code, the .NET Standard, etc.)
  + ASP.NET Core 1.x
    - Public release of ASP.NET Core 1.0 occurred in Q3 2016
    - This was a complete reimplementation of the ASP.NET Framework that had been around since 2002 without any significant changes
    - This new framework united all the components (e.g. MVC, Web API, web pages) into a single module (formerly known as MVC6)
    - This also introduced .NET Core, the new fully featured, cross-platform component and several new open-source tools (the CoreCLR cross-platform runtime, the improved RyuJIT x64 JIT complier, etc.)
    - ASP.NET Core was originally named ASP.NET 5; the renaming was meant to emphasize the complete separation from previous versions
    - The old Web API 2 tooling was a dedicated framework for building HTTP services returning JSON or XML
      * This was originally intended as an alternative to MVC and has since been merged into the general purpose web application MVC6
      * MVC6 is now shipped as a separate module for ASP.NET Core
    - ASP.NET Core 1.1 followed and introduced several new features like MVC pipeline middleware, a URL rewrite module, etc.
  + ASP.NET Core 2.x
    - This version came out in Q3 2017 and focused on standardizing shared APIs between .NET Framework, .NETE Core, and the .NET Standard
    - This made it easier for older ASP.NET devs to adopt the new framework and retain their previous knowledge
    - A follow-up was released soon after that added security and performance improvements
    - There were also several new features added in the 2.1 version
      * **SignalR**: an open-source library simplifying adding real-time web functionality to .NET Core apps
      * **Razor Class Libraries**: an improvement to the Razor SDK that allows devs to build views/pages into reusable class libraries
      * **Identity UI Library and Scaffolding**: allows adding identity to any app and customize it to need user needs
    - 2.1 also included built-in support for Global Data Protection Regulation (GDPR) and updated SPA templates for Angular and ReactJS
    - The updated default templates for Angular and React were actually big advancements for new devs
      * These new templates used the standard project templates and build pipelines for these frameworks
      * These tools replaced external task runners (e.g. Gulp, Grunt), module builders (e.g. webpack), and toolchains (e.g. Babel)
    - Version 2.2 included improved endpoint routing, updated templates, new health checks, new SignalR Java client, etc.
  + ASP.NET Core 3.x
    - This release came with many performance and security improvements and a bunch of new features
    - Security features included TLS 1.3, OpenSSL 1.1.1, improvements in the cryptography namespace, etc.
    - There were many improvements in the performance and reliability for the framework when used in a containerized environment
    - This release was the first to make big changes to make CoreCLR more efficient and generally improve performance in Docker
    - This version also introduced Blazor!
    - A huge size difference is noteworthy for macOS and Linux (less noticeable for Windows due to new WPF and Windows Forms platform-specific libraries)
    - Version 3.1 had many improvements to the Windows desktop development and fixes related to Blazor apps
* What’s New in Angular?
  + GetAngular and AngularJS
    - Angular started out as GetAngular and was a side project of Misko Hevery and Adam Abrons
    - AngularJS 1.0.0 was release in June 2012 and became hugely popular for many of the following reasons
      * **Dependency Injection**: this was the first client-side framework to have DI to support loosely coupled and easily testable components
      * **Directives**: marks on specific DOM items (elements, attributes, styles, etc.) to specify custom, reusable HTML
      * **Two-Way Data Binding**: binds data in the view with data in the model
      * **Single-Page Approach**: Angular kickstarted the SPA craze; it was the first to do this and demonstrate the benefits of fewer and smaller network requests and a smoother client experience
      * **Cache-Friendly**: because all AngularJS sites were client-side, they could be cached and made available anywhere (e.g. CDNs)
    - Angular 2 made several key advancements and also broke backwards-compatibility with previous versions
      * **Semantic Versioning**: first introduced here
      * **TypeScript Support**: an MS-made superset of JavaScript that adds typing
      * **Server-Side Rendering**: this is where the server runs your application on the back-end, renders the HTML, and provides them to the client (instead of the client rendering them)
      * **Angular Mobile Toolkit (AMT)**: a set of tools for building applications
      * **Command-Line Interface**: a new CLI could be used to generate Components, routes, services, pipes via the terminal
      * **Components**: these are the main building block of Angular and allow building of app data, business logic, templating, styling, etc.
    - Version 3 was skipped to align the SemVer for Angular and the various Angular Components that had been developed separately to that point
    - Angular 4 added some other breaking changes
      * **Ahead-of-Time (AOT) Compilation**: this compiles templates during the build phase and generates the appropriate JS; an improvement over the JIT compilation used previously
      * **Animations npm Package**: all animations were moved to a separate package and allowed apps that didn’t need them to be much smaller
    - Angular 5 had more breaking changes and security/reliability improvements
      * **New HTTP Client API**: this replaced the older package and included better JSON support, etc.
      * **State Transfer API**: an API that allows the server and client to transfer app state
      * **Additional Router Events**: this allows for more granular control over HTTP lifecycle
    - Angular 6 was mostly a maintenance release
    - Angular 7 was a more major release with several new, key features for the platform
      * **Easy Upgrade**: an interactive tool that eased the path for devs to upgrade Angular apps from older versions
      * **CLI Update**: a command to attempt an automatic upgrade for your Angular application
      * **CLI Prompts**: additional prompts during CLI commands to help devs discover new features
      * **Angular Material and CDK**: improvements that generally improve user experience in apps (e.g., virtual scrolling, drag-and-drop)
      * **Partner Launches**: aligning releases for several tools that support app dev (e.g., StackBlitz improvements, Angular Console)
      * **Updated Dependencies**: added support for newer versions of TS, RxJS, and Node
    - Angular 8 followed shortly and mostly focused on officially providing Ivy (a new, faster compiler/runtime for Angular), in addition to other improvements
      * **Bazel Support**: this is a free tool for the automation of build/test pipelines
      * **Routing**: new syntax for lazy-loading routes
      * **Service Workers**: more fidelity of control for when/if to register service workers when starting their apps
      * **Workspace API**: a better way to modify the Angular workspace config instead of via the JSON
    - Angular 9 was released in Q4 2019 and brings several additional features
      * **JS Bundles and Performance**: this was apparently an attempt to fix the very large bundle files that affect Angular and actually decreased performance due to increased download times
      * **Ivy Compiler**: this is now the default rendering engine
      * **Selector-less Bindings**: now added to Ivy
      * **Internationalization**: allows the creation of files for translators and publishing Angular apps in multiple languages
* Reasons for Choosing .NET Core and Angular
  + This is because they are stable, fast, and provide a slew of cross-platform features that can be leveraged by devs
* A Full-Stack Approach
  + Full-stack means working on the following to be able to deliver a full products
    - Back-end programming
    - Front-end programming
    - UI styling and UX design
    - DB design, modeling, config, and administration
    - Web server config and administration
    - Web app deployment
  + Note that we do not and will not be experts in all these fields
  + The goal is to become familiar with all of them so you are cognizant of the implications across the full stack during your development
* SPAs, NWAs, and PWAs
  + Single-Page Application (SPA)
    - This is a web-based application that attempts to offer the same user experience as a desktop application
    - This is obviously difficult due to the nature of web apps and common patterns of DOM management, UI, etc. must be updated
    - SPAs ideally load all their resources (HMTL, JS, CSS, etc.) either in one page load or dynamically fetch them as needed
    - The page should never reload/refresh; it should dynamically call the server to respond to user actions
    - Some of the main features of a good SPA are:
      * **No Server-Side Roundtrips**: SPAs should be able to redraw any part of the UI without needing a full roundtrip to retrieve a full HTML page; this is primarily done via a Separation of Concerns (SOC) separating data, business logic, and presentation
      * **Efficient Routing**: SPAs should be able to track a user’s state and location in your app using efficient JS-based routers
      * **Performance and Flexibility**: SPAs should transfer all UI rendering to the client via their JS SDK of choice (Angular, Vue, etc.); this generally increases performance and allows devs to change the UI without needing to change much, if anything, on the server
    - These features among others are part of the reason for SPAs replacing many traditional Multi-Page Applications (MPAs)
  + Native Web Application (NWA)
    - These are basically your traditional multi-page applications that serve several small single-page modules instead of one monolithic SPA
    - Note that this pattern is still widely popular and very effective; it is a solidified pattern in web app dev
  + Progressive Web Application (PWA)
    - The concept of a PWA was introduced in 2018 by a Google dev and a freelance dev
    - The idea was to take advantage of the modern concepts of service workers and manifest files to provide a better experience
    - Specifically, this means having apps that provide features like a mobile app (e.g., limited offline mode, push notifications, permission based HW access)
    - This style of app has become more and more popular as modern browsers have implemented support for service workers
    - The main technical features of a PWA are:
      * **Progressive**: progressive principles allow any user to use the app, regardless of browser
      * **Responsive**: fits to any form factor including those that have not yet been developed
      * **App-Like**: feels like a native application due to app-like features and navigation
      * **Fresh**: kept up-to-date due to update service worker constantly running in the background
      * **Safe**: served over an HTTPS URL endpoint
      * **Discoverable**: the app can be identified as an app via the manifest file, a registered service worker; it is discoverable by search engines
      * **Re-Engageable**: promotes user engagement via e.g. push notifications
      * **Linkable**: easily sharable via a URL and does not require complex installation
      * **Installable**: provides home screen icons without requiring installation from an app store
    - Note that these features are not mutually exclusive from NWAs or SPAs
    - PWAs are more a set of design principles, whereas SPAs and NWAs seem to be more fundamental design choices
    - A web app can implement as much or as little of these PWA features as it needs based on its design, frameworks, and end goals
* Product Owner Expectations
  + The Product Owner is the one in Agile methodologies who represents the customer expectations that devs struggle to satisfy
  + They prioritize functionality and are empowered by management to make tough calls for the project
  + Curtly, they are the ones in charge of the project, so it is imperative that we understand their vision
  + Here are some common expectations for typical web-based SPAs
    - **Early Releases**: customers want the ability to see what they are getting; for Scrum this is releasable products at the end of each sprint, for Waterfall this is the milestones for your project
    - **GUI Over Back-End**: customers really only ever see the front end of the product, so it makes sense that they are most concerned with it; because of this, front-end and GUI are often the first things that we need to work on, and this isn’t necessarily a bad thing; indeed it lets us verify the design with a front end and mocked back end before potentially coding ourselves into a corner
    - **Fast Completion**: you obviously want to be able to get things done quickly, so choosing front-end and back-end frameworks that work well together and allow reliable, consistent growth is important
    - **Adaptability**: this is the key of the Agile manifesto; we need to be able to respond and adapt to changes in requirements or design; indeed if we can’t do this we may consider the project a failure