Seneca
SCHOOL OF INFORMATION AND COMMUNICATIONS TECHNOLOGY

BTI425

Web Programming for Apps and Services

Introduction to Angular

Recall the React intro notes. React was described as a *library for building user interfaces*. As you learned, it is actually a rich and vibrant ecosystem for client-side app building.

Angular is also a widely-used platform for building client-side apps. Like React, it's JavaScript-based, and assumes that it's targeting a browser or browser-like runtime on a device.

React introduced the idea of *component*-oriented architecture to you. Arguably, it was Angular that conceived this idea, many years before React was released.

Angular, and AngularJS

We must clarify something before continuing, and it's important.

In this course, you will learn *Angular*, the current and future platform for client-side app building.

We will *not* cover its family ancestor, *AngularJS*. While this framework is still being maintained, it is likely that it will not receive any more significant upgrades in the future. It is simply not the focus of improvement efforts any longer.

Relevant history

In September 2009 - a long time ago in web years - Misko Hevery of Google introduced AngularJS to the world, as a personal “side project”. At the time, Hevery was motivated by the need to make client-side Ajax development work better.

> *The original product name was Angular.*
> *It started to become known as AngularJS sometime in late 2012 or early 2013.*

Its design and development approach were new and different. However, they proved to be impactful, and became part of Google’s development techniques. This led to the project’s open source release in October 2010, and after wide adoption and uptake by the developer community, its “version 1” was officially launched on June 14, 2012.

Then, a series of typical and normal events happened in the early life of the platform. These included massive use in apps large and small, evolution at Google to current and future web topics, developer community feedback, and rapid improvement in the deployment ecosystem (HTML5, JavaScript engines, and browsers).

Obviously, work on “version 2” happened too. And, React happened, which contributed both subtle and overt competitive pressure. It became apparent only a year after the original release that the “version 2” successor was going to be a bit of a departure, and not simply an upgrade (with the assumed implications of smoothness, compatibility, and so on).

By mid-2013 or so, the community was buzzing about breaking changes and the next Angular version being effectively a rewrite. Much thrashing, discussion, prototyping, and the goal to get it right led to the September 2014 unveiling of “Angular 2” (and the subsequent renaming of the original to “AngularJS”). Much more tumult happened before its official release in September 2016, including the choice of programming language. The new Angular must be considered a complete rewrite, and a clean break away from AngularJS.

As a developer who is contemplating the design and coding of an app, the old and the new have a good amount to share in their philosophy and approach. However, in terms of the actual coding and organization of the app, including tooling, packaging, and deployment, the old and the new have little in common. While we will give some old-and-new comparisons in linked articles, we’re not going to delve into them here.

Your teacher team feels it was necessary to cover off this relevant history, so that you are well-prepared to compete for modern web developer positions. Knowing the differences between Angular and AngularJS is vital for your credibility and adapability.

What’s similar to React, and what’s different

There are many similarities, including:

- Component-based development and architecture
- Declarative style of programming, for the UI
- Extends HTML, manages the DOM, de-emphasize (decouple) direct/explicit DOM manipulation
- Decouples client-side app dev concerns from server-side app dev concerns

Here are a few differences:

- Some argue that React is more focused to only building UIs
- Angular describes itself as a *complete app dev platform*
- Angular's default programming language is TypeScript

Similar to React, the big difference from legacy-style client-side programming is that we will not work directly with the DOM. Yes, our work will affect the rendered DOM, but we cannot use that conceptual model as a foundation for building and modifying the DOM.

Big ideas in Angular

After the React topic coverage, you have a basis for understanding *modern* client-side app building.

It all starts with *components*, introduced recently. The app is composed of a hierarchy of nested or contained components, and each encapsulates is visible content and appearance, its interaction code, and interfaces to other components and services. This is arguably the most important concept in Angular.

In code, the component architecture is driven by the design and coding environment of the TypeScript programming language. The organization of a folder to hold the component's source code, as well as the all the other bits have been influenced as well.

TypeScript - should I be worried?

No.

As a brief introduction and paraphrasing from the documentation:

> *TypeScript is a primary language for Angular application development. It is a superset of JavaScript with rich design-time support for type safety and tooling. It's JavaScript that scales. It boosts expressiveness, productivity, efficiency, and effectiveness. In other words, it enables the dev to get more out of their efforts.*
>
> *The developer tools transpile TypeScript code into plain JavaScript.*

We're not going to obsess about the TypeScript differences from JavaScript. We're just going to simply begin using TypeScript, and incrementally and immediately explain anything that needs an explanation.

In summary… (take-aways)

The *browser* is the deployment target.

In the "old days", programmers targeted *operating systems*. We wrote *for Windows*, or *for Unix*. Even when we wrote for a managed runtime environment (e.g. Java), there was always part of our work - sometimes a big part - that had to respect the services and facilities offered by the platform and device. In other words, we couldn't install an app like the College's *Student Center* on a 2005-era cellphone. (In those days, "smart" cellphones often came with a Java runtime.)

Value proposition of client-side front end dev

There are several articles and opinions out there about the front end dev idea. Here are a few:

~ ~ ~ ~ ~

Alexandre Lombard wrote a very good article early in 2020: Do We Really Need a Front-end Framework? (on Medium).

> *And I found that, the short answer is no if you focus on the features, while the real answer is yes if you focus on the user experience (UX). Let's see why, and see what questions you should ask yourself before choosing a front-end solution.*

> *The choice of a front-end technology doesn't depend on the features you want to provide to your user, but on the user experience you want him/her to have*

~ ~ ~ ~ ~

In February 2020, in the Stack Overflow blog, Max Pekarsky wrote a very good lengthy article: Does your web app need a front-end framework?

> *A few years ago, while working on a side project, … my own front end became unwieldy. I had a loose sense that implementing a framework was the right next step, but I had little idea about what they did. [This article has] the explanation I wish had then.*

Good, comprehensive treatment of what we face as modern software developers.

~ ~ ~ ~ ~

Cory House wrote a very good article a few years ago:
Here's Why Client-side Rendering Won (on Medium).

Paraphrasing and summarizing, client-side front end dev "won", because it offered these massive advantages:

- No full page load
- Lazy loading
- Rich iteractions
- Cheap hosting
- Leverages CDN
- Easy deployment
- Enforced separation of concerns
- Learn once, write everywhere
- Same UI Technology for Web, Native Mobile, and Desktop

Does server-side rendering make sense anymore?
Sure. But in far fewer situations than before. If you're building a mostly static site, server-side rendering will be easier.
