What Is Angular?

AngularJS is a structural framework for dynamic web apps. It lets you use HTML as your template language and lets you extend HTML's syntax to express your application's components clearly and succinctly. Angular's data binding and dependency injection eliminate much of the code you would otherwise have to write. And it all happens within the browser, making it an ideal partner with any server technology.

Angular is what HTML would have been had it been designed for applications. HTML is a great declarative language for static documents. It does not contain much in the way of creating applications, and as a result building web applications is an exercise in *what do I have to do to trick the browser into doing what I want?*

The impedance mismatch between dynamic applications and static documents is often solved with:

- **a library** - a collection of functions which are useful when writing web apps. Your code is in charge and it calls into the library when it sees fit. E.g., `jQuery`.
- **frameworks** - a particular implementation of a web application, where your code fills in the details. The framework is in charge and it calls into your code when it needs something app specific. E.g., `durandal`, `ember`, etc.

Angular takes another approach. It attempts to minimize the impedance mismatch between document centric HTML and what an application needs by creating new HTML constructs. Angular teaches the browser new syntax through a construct we call directives. Examples include:

- Data binding, as in `{{}}`.
- DOM control structures for repeating/hiding DOM fragments.
- Support for forms and form validation.
- Attaching new behavior to DOM elements, such as DOM event handling.
- Grouping of HTML into reusable components.

A complete client-side solution

Angular is not a single piece in the overall puzzle of building the client-side of a web application. It handles all of the DOM and AJAX glue code you once wrote by hand and puts it in a well-defined structure. This makes Angular opinionated about how a CRUD application should be built. But while it is opinionated, it also tries to make sure that its opinion is just a starting point you can easily change. Angular comes with the following out-of-the-box:

- Everything you need to build a CRUD app in a cohesive set: data-binding, basic templating directives, form validation, routing, deep-linking, reusable components, dependency injection.
- Testability story: unit-testing, end-to-end testing, mocks, test harnesses.
- Seed application with directory layout and test scripts as a starting point.

Angular Sweet Spot

Angular simplifies application development by presenting a higher level of abstraction to the developer. Like any abstraction, it comes at a cost of flexibility. In other words not every app is a good fit for Angular. Angular was built with the CRUD application in mind. Luckily CRUD applications represent the majority of web applications. To understand what Angular is good at, though, it helps to understand when an app is not a good fit for Angular.

Games and GUI editors are examples of applications with intensive and tricky DOM manipulation. These kinds of apps are different from CRUD apps, and as a result are probably not a good fit for Angular. In these cases it may be better to use a library with a lower level of abstraction, such as `jQuery`.

The Zen of Angular

Angular is built around the belief that declarative code is better than imperative when it comes to building UIs and wiring software components together, while imperative code is excellent for expressing business logic.

- It is a very good idea to decouple DOM manipulation from app logic. This dramatically improves the testability of the code.
- It is a really, *really* good idea to regard app testing as equal in importance to app writing. Testing difficulty is dramatically affected by the way the code is structured.
- It is an excellent idea to decouple the client side of an app from the server side. This allows development work to progress in parallel, and allows for reuse of both sides.
- It is very helpful indeed if the framework guides developers through the entire journey of building an app: from designing the UI, through writing the business logic, to testing.
- It is always good to make common tasks trivial and difficult tasks possible.

Angular frees you from the following pains:

- **Registering callbacks:** Registering callbacks clutters your code, making it hard to see the forest for the trees. Removing common boilerplate code such as callbacks is a good thing. It vastly reduces the amount of JavaScript coding *you* have to do, and it makes it easier to see what your application does.
- **Manipulating HTML DOM programmatically:** Manipulating HTML DOM is a cornerstone of AJAX applications, but it's cumbersome and error-prone. By declaratively describing how the UI should change as your application state changes, you are freed from low-level DOM manipulation tasks. Most applications written with Angular never have to programmatically manipulate the DOM, although you can if you want to.
- **Marshaling data to and from the UI:** CRUD operations make up the majority of AJAX applications' tasks. The flow of marshaling data from the server to an internal object to an HTML form, allowing users to modify the form, validating the form, displaying validation errors, returning to an internal model, and then back to the server, creates a lot of boilerplate code. Angular eliminates almost all of this boilerplate, leaving code that describes the overall flow of the application rather than all of the implementation details.
- **Writing tons of initialization code just to get started:** Typically you need to write a lot of plumbing just to get a basic "Hello World" AJAX app working. With Angular you can bootstrap your app easily using services, which are auto-injected into your application in a Guice (https://github.com/google/guice)-like dependency-injection style. This allows you to get started developing features quickly. As a bonus, you get full control over the initialization process in automated tests.