Angular 2.0

As we’re starting into the implementation of Angular 2.0, we thought we should put pen to paper and give you some insight into how we’re thinking about the design and why we’re making the changes we are. We’re sharing it with you now so you can help make the right choices.

Angular 2.0 is a framework for mobile apps. It is for desktop as well, but mobile is the hard bit that we will get right first. The Angular you know and, hopefully, love will still be there with data-binding, extensible HTML, and a focus on testability.

All the design docs for Angular 2 are publicly available on [Google Drive](https://drive.google.com/?pli=1#folders/0B7Ovm8bUYiUDR29iSkEyMk5pVUk). This document contains a summary of our approach and design principles with links to specific designs in context.

**CAVEAT:** We’re still in design and prototyping stage for Angular 2.0. Some of this will happen differently or not at all in the final product.

# Designed for the future

We’re designing Angular 2 for the way the world will look when we believe folks will use it. In particular, this means targeting modern browsers and using ECMAScript 6.

## Modern Browsers

Modern browsers means the set of browsers known as ‘evergreen’ or always automatically updated to the latest version. The set currently includes Chrome, FireFox, Opera, Safari, and IE11. On mobile, we’ll support something close to Chrome on Android, iOS 6+, Windows Phone 8+ and Firefox mobile. We’re looking into supporting older versions of Android, but the jury is still out. Building for these browsers let us drop many hacks and workarounds that make Angular harder to use and develop on than it needs to be.

Yes, there are still older browsers in use, but we’re targeting these newer models as they will be the primary ones in use by the time Angular 2.0 is ready and you good developers have had time to build apps on it.

## ECMAScript 6+ ([design](https://docs.google.com/document/d/1uhs-a41dp2z0NLs-QiXYY-rqLGhgjmTf4iwBad2myzY))

All code in Angular 2 is already being written in ES6. As ES6 doesn’t run in browsers today, we’re using the [Traceur compiler](https://github.com/google/traceur-compiler) to generate the nice ES5 that runs everywhere. We’re working with the Traceur team to build support for a few extensions like annotations and [assertions](http://vojtajina.github.io/assert).

Don’t worry. Though Angular will be in ES6, you can still write in ES5 if you don’t want to upgrade. The compiler generates readable JS and there are human-sensible analogs for the extensions. See the [design doc](https://docs.google.com/document/d/1uhs-a41dp2z0NLs-QiXYY-rqLGhgjmTf4iwBad2myzY) for more info.

# Faster

## Faster change detection ([design](https://docs.google.com/document/d/10W46qDNO8Dl0Uye3QX0oUDPYAwaPl0qNy73TVLjd1WI))

Angular apps are built around data-binding between DOM and JS objects. The speed of Angular apps is driven largely by the underlying change detection mechanism we use. We’ve talked at length on how we hoped to make this faster by using [Object.observe()](http://updates.html5rocks.com/2012/11/Respond-to-change-with-Object-observe) when it becomes available in Chrome M35. And we will!

After some hard analysis (detailed in the doc), however, we’ve also found that we can make this much faster and more memory efficient even before this native change detection hits browsers. Buttery-smooth app buttery-butterness is around the corner.

## Instrumented ([design](https://docs.google.com/document/d/1gGUEODxxDjY7azF8InqtN1pRcLo3WhGb8BcoIihyI80))

## The other half of the performance equation is the code that you write. For this, we’ll provide high resolution timing details of where time gets spent in your application. You can see in that this is a goal in the Change Detection design, but we’ll directly support this through a new Angular-wide logging service called diary.js.

## Modular

When we released Angular 1.0, the features were in a take-it-or-leave-it single package. You incur the download price for every piece whether you use it or not. Though the whole of Angular is very small by desktop app standards, this can make a big difference on mobile devices.

One interesting thing we noticed was that when we split out the $route into a separate library, several innovative replacements popped up.

For performance and to enable innovation, our goal is that almost every piece of Angular should be optional, replaceable, and even used in other non-Angular frameworks. You’ll be able to pick and choose the parts you like and write or select others that you like better.

## Other Performance topics

Though there are no designs yet, there are many other areas of performance optimization we’ll include in Angular. From increasing first-load time with pre-parsed templates to ensuring silky smooth 60 frames per second animations, we’ll invest heavily on a fully optimized user experience. Please help us by highlighting areas where we should focus and techniques you can contribute to the shipping solution.

# Simpler

## Dependency Injection ([design](https://docs.google.com/document/d/1fTR4TcTGbmExa5w2SRNAkM1fsB9kYeOvfuiI99FgR24))

Dependency Injection is still a key differentiator between Angular and other client side frameworks in eliminating much of your application’s wiring code and making testability-by-default a reality. Though we’ve enjoyed these benefits in building our applications, we’re dissatisfied with the current implementation. We can make it less complex and more capable at the same time.

We’ll see a less complex DI by eliminating the config phase, simplifying the syntax by using declarative-style ES6+ annotations instead of imperative. We’ll enjoy greater capabilities by integrating DI with module loading through ES6 Modules. We’ll also see the ability to lazily-load parts of our JS through child injectors.

The doc linked above has our initial thoughts, but this is one part of Angular 2 that you can try today. See [the repo](https://github.com/angular/di.js) for current details on the implementation.

## Templating and Directives ([design](https://docs.google.com/document/d/1f5VWROeTI2kJwVKbNsrHuEz5IqtZe14OpoxM9fEYJNU))

The ability to specify templates directly in HTML and extend its syntax are the bread and butter of Angular. We have some lofty goals for Angular’s template compiler in 2.0. Namely:

* Simplify the directive API
* Integrate with other component frameworks using web standards
* Improve performance
* Allow tools like IDEs to analyze and validate templates

We’re so excited about these new bits that we can hardly wait to show them off. There’s too much good stuff to extract into a summary, so please just go straight to the [design doc](https://docs.google.com/document/d/1f5VWROeTI2kJwVKbNsrHuEz5IqtZe14OpoxM9fEYJNU) to find out more.

# More capable

## Touch animations ([design](https://docs.google.com/document/d/16Z6Lun15DoWNrE2imk7N-2WiRAaqc954LOfU2-2JSoI))

Users are accustomed to certain touch aware usage patterns. E.g. scroll through a list using their finger, circle through pictures in a carousel, remove list entries by swiping them away. However:

* Current implementations of carousel, infinite scrolling, … don’t share a common core and by this have a lot of redundancy and different approaches.
* Current implementations mostly don't provide an option to use native scroll events, as older browsers and also some current browsers don't support them well. However, by this they prevent the optimal performance on new devices.

We want to give these scenarios first-class support for the best user experiences possible in your applications.

## Router ([design](https://docs.google.com/document/d/1I3UC0RrgCh9CKrLxeE4sxwmNSBl3oSXQGt9g3KZnTJI))

The initial Angular router was designed to handle just a few simple cases. As Angular grew, we've slowly added more features. However, the underlying design is ill-suited to be extended much further.

We’re taking a careful look at known use cases and other router implementations across many application frameworks so we can deliver a simple yet extensible router that should fit the widest set of applications.

A few cases that we’re particularly keen on supporting include:

* State-based routing
* Integration with authentication and authorization
* Optionally preserving state of some views. Particularly needed for mobile!

## Persistence ([design](https://docs.google.com/document/d/1DMacL7iwjSMPP0ytZfugpU4v0PWUK0BT6lhyaVEmlBQ))

Beyond Angular’s humble $http, many developers have desired a higher level abstraction for working with data from servers and local persistent data in the browser.

Mobile apps need to work in an “always offline” mode with syn to the server. RESTful services need more than we’ve provided in $resource. Some data can be updated in batches while some needs a continuous streaming model.

In this new persistence layer, we’ll provide clean structure for these cases and take much more out of the boilerplate currently necessary.

## 

# Q&A

## When will it be done?

If you’ve been with us through the 1.2 announcement, you know that we don’t know. :) Though we’re just posting the design docs now, we’re already prototyping many of the modules. Dependency Injection and Zone.js appear to be usable. All the work will be done on GitHub and we’ll continue to capture [weekly meeting notes](https://docs.google.com/document/d/150lerb1LmNLuau_a_EznPV1I1UHMTbEl61t4hZ7ZpS0) so you can follow along.

## What will it be like to port an Angular 1.x application to Angular 2.0?

As Angular 2 is still in development, we honestly don’t know. In our imagination, porting will be fairly straightforward but not free. Taking advantage of ES6 will likely be the largest part of the job. Templates will be largely the same with some mostly-mechanical find and replace exercises. Your controllers contain your business logic and not much if any Angular API today. Those should port easily. The parts that will require the most thinking will be your use of Modules and Directives.

## Is Angular 2.0 a replacement for mobile tech like PhoneGap or Ionic Framework?

No, Angular is still just the core framework. You might still want libraries like Ionic for mobile-optimized CSS/JS components and tools like PhoneGap for app packaging and native API access.

## How does Angular 2.0 relate to AngularDart?

When porting AngularJS to the Dart language, we build a new version of Angular using all the learnings we’d acquired to date. Many of the improvements discussed in this document like improved Directive concepts and syntax and class/annotation-based DI are already present there.

While the implementation isn’t what we’ll finally arrive at for 2.0, it’s a great preview of what’s to come.

We’ll be upgrading AngularDart as we build AngularJS 2.0 so folks who prefer the Dart language can enjoy the same benefits as folks on JS. Our goal is that there will be a single framework with your choice of language.

# Other documents of interest

* [High Level Design Considerations](https://docs.google.com/document/d/1xaBPLLkk1gUI66-3OhNT8mR_QDrJL4EFqw28_5s0WMs)
* [Parser Abstraction Notes](https://docs.google.com/document/d/1gtPjO4BA_8naLeIM8NCFdNOB-atq7TEA-UsJJP33HAs)
* [Templating Implementation Ideas](https://docs.google.com/document/d/1tSrXqMiFd8lAdTkp78Do-6V5nNjQ8b2P01AlYG9FOWQ)
* [Templating Examples](https://docs.google.com/document/d/1je68EfKqPH9FcOrSqShZzVSYQpvV93p8ZEljBzDsHDw)