High Level Design Considerations

*Author:* [*rob@bluespire.com*](mailto:rob@bluespire.com)

*This document serves as a place to put cross-cutting design discussions. The author has seeded the document with a few items he feels strongly about. Please comment and add other important design considerations or ideas about framework “aesthetics”, usability or extensibility.*

# Modular

Within reason, Angular 2.0 should be constructed as a set of loosely coupled modules. A developer who wants to use only the observable object system or only the templating system should be able to do that. If they like the DI library only, let them take just that.

# Unobtrusive

Common application code should look as “plain” as possible. Essentially, this means that when looking at the majority of application code, we should not see “angularisms”. The number of application classes/objects that need to directly interact with the Angular API surface should be low. When a developer looks at their code...they should see *their code* and not Angular.

# Teachable

Angular’s built-in services should have sensible default behaviors. However, when designing these services we should consider the breadth of development cultures and software design methodologies which may be employing their use. We should consider exposing key “hooks” for each service to allow developers to selectively “teach Angular new tricks.” An example of one such key point is….

## Separate Metadata Model from Metadata Discovery

In Angular 2.0 much of the framework is controlled by metadata. This includes things such as how directives operate or instructions for the DI system concerning object lifetime. Currently, the default mechanism for providing this metadata is through annotations on classes and class members. However, there are other valid ways that developers may wish to provide this information. Two common ways are through centralized configuration or through conventions. It would be nice to see Angular 2.0 allow for “pluggable” metadata discovery mechanisms. To support this, each service that uses metadata could support a metadata discovery delegate. The default implementation would return data from annotations. But, a custom delegate could be supplied to obtain the metadata from elsewhere. Examples of services that might support this are: compiler, DI and router.