Package Manager for Angular

*Status: slightly outdated | Authors:* [*igor@angulajs.org*](mailto:igor@angularjs.org)

*Created: Feb 2014 | Last update: March 2014 | Url:* [*http://goo.gl/vEnvQ5*](http://goo.gl/vEnvQ5)

# Objective

Select a reliable, secure and client-side focused package manager for Angular 2 core, Angular dependencies, as well as third-party components. If no existing solution meeting our criteria exists, this document should select the one that is the closest to our ideal and suggest improvements and a plan for implementing them.

# Backgroun

AngularJS was originally created when modularity wasn't a mainstream best practice in client-side development. In 2012 [bower](http://bower.io/) came out and we started using it because it seemed to be the best choice at the time. While bower did a decent job at serving our needs, it wasn't without complications. We discovered several design flaws that pushed us to reconsider our options for Angular 2. Most of these issues are reflected as requirements in this doc.

# Prior Art

## bower

[bower](http://bower.io/%20) is a node.js powered package manager that is module system agnostic, uses a custom namespace registry and distributed git (and other SCMs) as its file repository and transport layer. It's heavily focused on client-side components.

Roadmap:

* [Next-gen registry & architecture doc](https://github.com/bower/registry/issues/73) ( Jul 13, 2014)
* [Design doc for Bower 2 from Jun 2013](https://docs.google.com/document/d/17Nzv7onwsFYQU2ompvzNI9cCBczVHntWMiAn4zDip1w/edit?usp=docslist_api)
* [signs](https://github.com/bower/registry/pull/50) of [progress](http://gyazo.com/374a120dd4214e2c4a07d55d56293a2e)

## npm

[npm](https://npmjs.org/) is the default package manager for Node.js. It's module system agnostic but is currently heavily dominated by server-side packages using the CommonJS module system targeting Node.js APIs.

## pub

[pub](https://pub.dartlang.org/doc/) is the default package system for Dart. While it's not an ideal choice for JavaScript applications, the design philosophy behind pub aligns well with some of our requirements.

## component

[component](https://github.com/component/component) is a node.js powered package manager that uses the CommonJS module system for file loading and GitHub as repository / registry service.

## duo

d[uo](http://duojs.org/) is a new/up-and-coming package manager that the [component](https://github.com/componentjs/component) (see above) team are recommending:

“*Newcomers should use Duo instead of this implementation. Duo is more feature-complete and adds additional goodies while still supporting component.json, so check it out!*”

## maven/rubygems/etc

Pretty much every modern (web) development platform has a package management solution. Some are better than others, but it's clear that package management is an important piece of a platform’s ecosystem. For example it was only after Maven took off that code reusability and sharing on the Java platform became a real thing.

## jspm

[jspm](http://jspm.io/) is a package manager for the SystemJS universal module loader which supports AMD, CommonJS, globals and ES6 modules, the latter built on top of the dynamic ES6 module loader. It uses a flat dependency management which is a major requirement for version-management in the browser. jspm isn't also a package manager in classical sense. It's more like a package manager of package managers.

# 

# Detailed Design

## Requirements

A package repository usually consists of the following components: the registry (metadata repository), file storage (package file repository), transport layer, client

1. registry
   1. must support replication[[1]](#footnote-0)
   2. must have an API for querying packages by any metadata
   3. must have web ui that enables package discoverability
   4. should support package ownership transfer[[2]](#footnote-1)
   5. should support good namespacing or have have clear rules that prevent name squatting[[3]](#footnote-2)
   6. could support query/download/usage stats[[4]](#footnote-3)
2. file repository (storage mechanism for files in the package)
   1. must be at least as available as the registry[[5]](#footnote-4)
   2. must support replication[[6]](#footnote-5)
   3. must guarantee immutability of files once published[[7]](#footnote-6)
   4. should scale well during traffic spikes[[8]](#footnote-7)
   5. should be separate from project's source repository[[9]](#footnote-8)
3. transport layer
   1. must be secured using encryption (e.g. with TLS)[[10]](#footnote-9)
   2. must use compression to reduce the amount of transferred data[[11]](#footnote-10)
4. client
   1. must have minimal startup cost. (A noop update must take less than 0.5s)[[12]](#footnote-11)
   2. must support offline invocation of `<pkgcmd> update` command if no new files need to be fetched via the network[[13]](#footnote-12)
   3. should have a [version constraint solver](http://pub.dartlang.org/doc/versioning.html#constraint-solving) and not rely on module isolation to deal with conflicting transitive dependencies[[14]](#footnote-13)
   4. must support freezing package version definition into the project (a.k.a. shrinkwrap, lockfile)[[15]](#footnote-14)
   5. should be capable of answering the question "how many more bytes will adding dependency X and its transitive dependencies add to my project" (aptitude model)[[16]](#footnote-15)
   6. should support mechanisms to prevent tampering and detect file corruption (signing files, file checksums)[[17]](#footnote-16)
   7. should have hooks for integration with build systems and template generators[[18]](#footnote-17)
5. the package format
   1. must support custom metadata associated with each package[[19]](#footnote-18)
   2. must be module loader agnostic (we primarily care about ES6 modules, but don't require direct support).[[20]](#footnote-19)
   3. must support non-javascript assets (images, css, etc)[[21]](#footnote-20)
   4. must allow defining dependencies using rich semver based version matching[[22]](#footnote-21)
6. community / development team
   1. must have an active and diverse community around the project[[23]](#footnote-22)
   2. must have committed and responsive development team[[24]](#footnote-23)
   3. must have open/transparent development practices[[25]](#footnote-24)
   4. should have a roadmap aligned with our requirements[[26]](#footnote-25)
   5. should be open to our contributions if needed[[27]](#footnote-26)

## Comparison existing solutions

This table mirrors the requirements described above

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **requirement / solution** | **bower** | **npm** | **pub** | **component** |
| **1. registry** | - | - | - | - |
| a. replication | no | yes[[28]](#footnote-27) | no? | no |
| b. rich api | [yes](http://bower.io/#programmatic-api) | [yes](https://www.npmjs.org/api/npm.html) | ??? | ??? |
| c. web ui | limited[[29]](#footnote-28) | [decent](https://www.npmjs.org/) | [great](https://pub.dartlang.org/) | [broken](http://component.io/)[[30]](#footnote-29) |
| d. name transfers | manual[[31]](#footnote-30) | yes | ??? | yes (via GH) |
| e. namespacing / anti-squatting | no | [anti-squatting](https://www.npmjs.org/doc/misc/npm-disputes.html) only[[32]](#footnote-31) | ??? | yes (via GH) |
| f. stats | [yes](http://bower.io/stats) (client-based) | [yes](https://www.npmjs.com/) (server-based) | no | no |
| **2. file repository** | - | - | - | - |
| a. high availability | usually[[33]](#footnote-32) | usually[[34]](#footnote-33) | yes | github |
| b. replication | no | yes | no? | no |
| c. immutability | no[[35]](#footnote-34) | yes | ??? | no |
| d. scalability | package dependent | usually[[36]](#footnote-36) | yes[[37]](#footnote-37) | github |
| e. isolated from source repo | no | yes | yes | no |
| **3. transport layer** | - | - | - | - |
| a. secure by default | package dependent[[38]](#footnote-38) | yes[[39]](#footnote-39) | ??? | ??? |
| b. compression | usually[[40]](#footnote-40) | yes[[41]](#footnote-41) | ??? | ??? |
| **4. client** |  |  |  |  |
| a. noop update under 0.5s | no[[42]](#footnote-42) | yes | ??? | ??? |
| b. transparent offline mode | no[[43]](#footnote-43) | yes | ??? | ??? |
| c. version constraint system | [no](https://github.com/bower/bower/issues/578) | no?[[44]](#footnote-44) | [yes](http://pub.dartlang.org/doc/versioning.html) | no |
| d. dependency freezing | [no](https://github.com/bower/bower/issues/505) | yes[[45]](#footnote-45) | [yes](http://pub.dartlang.org/doc/versioning.html) | no |
| e. dependency weight analysis | no | no | no | no |
| f. tampering / corruption prevention | no | yes[[46]](#footnote-46) | ??? | no |
| g. tooling integration hooks | no | no[[47]](#footnote-47) | ??? | no |
| **5. package format** | - | - | - | - |
| a. custom metadata | partially[[48]](#footnote-48) | partially[[49]](#footnote-49) | [no](https://pub.dartlang.org/doc/pubspec.html) | ??? |
| b. module loader agnostic | yes | yes[[50]](#footnote-50) | ??? | no[[51]](#footnote-51) |
| c. non-javascript assets | yes | yes | ??? | yes |
| d. semver version matching | yes | yes | yes | yes |
| **6. community / core team** | - | - | - | - |
| a. active & diverse community | yes | yes | yes[[52]](#footnote-52) | no |
| b. committed/responsive team | varies[[53]](#footnote-53) | [yes](http://blog.npmjs.org/post/76320673650/funding) | yes | ??? |
| c. open and transparent | yes | [yes](http://blog.npmjs.org/) | ??? | ??? |
| d. roadmap | unclear | [positive](http://blog.npmjs.org/post/76320673650/funding) | ??? | unclear |
| e. "contributable" | iffy[[54]](#footnote-54) | ??? | ??? | ??? |

|  |  |  |
| --- | --- | --- |
| requirement / solution | jspm | duo |
| 1. registry | - | - |
| a. replication | yes[[55]](#footnote-55) |  |
| b. rich api | yes |  |
| c. web ui | no |  |
| d. name transfers | yes[[56]](#footnote-56) |  |
| e. namespacing / anti-squatting | yes[[57]](#footnote-57) |  |
| f. stats | no |  |
| 2. file repository | - | - |
| a. high availability | \*[[58]](#footnote-58) |  |
| b. replication | \* |  |
| c. immutability | \* |  |
| d. scalability | \* |  |
| e. isolated from source repo | \* |  |
| 3. transport layer | - | - |
| a. secure by default | \*[[59]](#footnote-59) |  |
| b. compression | \*[[60]](#footnote-60) |  |
| 4. client | - | - |
| a. noop update under 0.5s | \*[[61]](#footnote-61) |  |
| b. transparent offline mode | yes[[62]](#footnote-62) |  |
| c. version constraint system | yes[[63]](#footnote-63) |  |
| d. dependency freezing | yes |  |
| e. dependency weight analysis | no |  |
| f. tampering / corruption prevention | no |  |
| g. tooling integration hooks | partly[[64]](#footnote-64) |  |
| 5. package format | - | - |
| a. custom metadata | yes |  |
| b. module loader agnostic | no[[65]](#footnote-65) |  |
| c. non-javascript assets | yes[[66]](#footnote-66) |  |
| d. semver version matching | yes |  |
| 6. community / core team | - | - |
| a. active & diverse community | yes |  |
| b. committed/responsive team | yes |  |
| c. open and transparent | yes |  |
| d. roadmap | ?[[67]](#footnote-67) |  |
| e. "contributable" | yes |  |

# 

# Caveats

Because the package repository will be used by many people, it should be developed and maintained by the JavaScript community at large. Unfortunately, package repository discussions run the risk of turning into religious debates. We need to be sensitive to the preferences and requirements of others without compromising the requirements set out above. Because Angular is such a widely-used and influential project, our choice will affect how other developers publish their code. Because of this, we want to pick a solution that benefits the whole JavaScript community.

# Security Considerations

Developers should be able to trust the package repository system they use. The repository maintainers should be security focused and take any proactive measures to secure the artifacts during publishing, storage, and retrieval. Because a repository is potentially vulnerable through these three vectors and contains trusted code artifacts, it is an attractive target for attackers.

# Performance Considerations / Test Strategy

TBD

# Work Breakdown

Preliminary plan:

1. finish the analysis of the existing solutions
2. pick the final solution
3. break down things that need to be done

1. The registry should not be single point of failure, we should be able to run our own replica at repo.angularjs.org and use that for our CI builds and releases. [↑](#footnote-ref-0)
2. Maintainers might change during a lifetime of a package, the registry should handle this real life scenarios without too much hassle. [↑](#footnote-ref-1)
3. Most registries create a new namespace in a world full of namespaces (DNS, github, twitter). This attracts squatters. We need to be able to deal with them. [↑](#footnote-ref-2)
4. It's good to know what's popular! Stats make deciding between two comparable packages easier. [↑](#footnote-ref-3)
5. This requirement is driven by an issue with Bower where files for each package can be stored on different servers, each maintained by unique parties. This means that in order for bower install to succeed, all of these servers need to be available at the same time. N points of failure without any redundancy are much worse than one point of failure. [↑](#footnote-ref-4)
6. Replication is critical for ensuring that we (or anyone else!) can operate our CI builds and day-to-day development without relying on availability of third-party services like the package repository. In the past npm and bower outages had significant impact on our productivity. When an outage happens, there is really very little that can be done unless we prepare for it ahead of time. [↑](#footnote-ref-5)
7. A package repository is the source of truth for shared code. If the repository is compromised, gets corrupted or if the package owner simply replaces or deletes the package files, everyone depending on that particular version of files will be affected. In the past we had a problem with a package maintainer deleting an npm package without any warning. This should not be allowed. It's ok to have grace period (1h?) after publishing to correct publishing of wrong build, but once people start using that version of a module, no further modifications should be allowed. [↑](#footnote-ref-6)
8. The repository should be built to scale. We should review how the design and architecture of various solutions are prepared to deal with spikes. Good replications story (2b) usually (but not always) means ability to scale. [↑](#footnote-ref-7)
9. Mixing source and build artifacts is not a good practice because it results in big source repos, messy code reviews and general development overhead. [↑](#footnote-ref-8)
10. Encryption prevents tampering with data and provides privacy. [↑](#footnote-ref-9)
11. Because communication with pository results in lots of transferred bytes, the data should be compressed either on the fly at the HTTP layer or ahead of time by compressing individual packages. [↑](#footnote-ref-10)
12. We need to ensure that we are using the right dependencies every time we create a local or CI build of Angular. Being able to do this with minimal overhead is critical to ensure speedy build. [↑](#footnote-ref-11)
13. Offline support usually translates to good caching strategy. This requirement enables us create builds in environments that don't have an internet connection (airplanes, submarines, locked down networks). [↑](#footnote-ref-12)
14. Having more than one version of a package loaded in the browser context at a time is troublesome for many reasons. The main two are [interoperability](http://pub.dartlang.org/doc/versioning.html#shared-dependencies-and-unshared-libraries) and performance. For this reason it is important that the package manager facilitates the process of picking one version of each module for the app given version constraints for transitive dependencies expressed via semver patterns. Doing this by hand is too time consuming. [↑](#footnote-ref-13)
15. semver and version constraint resolver make patch release upgrades of dependencies easy, but we need to be able to know exactly what version is being used by teammates or in production. In these scenarios locking down dependencies to exact versions or even sha's is a must. [↑](#footnote-ref-14)
16. Bytes matter. Transitive dependencies can add bytes very quickly even during minor dependency updates. We need the package manager to help us keep file size under control and allow us to understand the impact a dependency has on the payload of our apps when choosing a dependency or during a (minor) upgrade. [↑](#footnote-ref-15)
17. We put a ton of trust into the package manager. Checksums are a good way to ensure that we are not using files that were modified due to network/storage corruption, republishing (authorized or unauthorized). [↑](#footnote-ref-16)
18. The archetypical example is adding a <script> tag to an existing index.html as you install new packages. The functionality shouldn't depend on any particular generator system, we really just need a hook like "invoke some function after fetching a package". [↑](#footnote-ref-17)
19. Query-able metadata will allow us to build tools and apps on top of the repo. For example a catalog of reusable components. Metadata also allows integration with other tools (e.g. built tools). [↑](#footnote-ref-18)
20. We expect that ES Modules will be THE module system for all of client-side and server-side JS in the future. Any solution that is hard-wired for a different module system will fall out of favor. In the meantime, metadata can be used to specify what module system a particular package supports. [↑](#footnote-ref-19)
21. It's useful to be able distribute non-js files via the package repository. For example we currently fetch Closure Compiler and minErr compilation pass via bower even though they are Java binaries. [↑](#footnote-ref-20)
22. [semver](http://semver.org/) is the way to go. It's the industry standard and it works reasonably well. We do need to watch out for how the standard is implemented though to avoid issues that we experienced in [the](https://github.com/bower/bower/issues/782) [past](https://github.com/bower/bower/issues/1017). [↑](#footnote-ref-21)
23. In the Angular 2 world, interoperability with modern reusable client-side-friendly non-Angular code is important, so the repository must be used by both Angular and non-Angular communities. [↑](#footnote-ref-22)
24. Does someone monitor issue tracker / mailing list? Are significant issues being promptly investigated and resolved? Is the project being developed on an ongoing basis? [↑](#footnote-ref-23)
25. Do dev discussions take place or get posted on a public forum? Is the code developed in the open (e.g. on GitHub?) [↑](#footnote-ref-24)
26. Is there a project roadmap? Is the roadmap going to preserve or enhance things we care about? [↑](#footnote-ref-25)
27. In case we need to get a fix in, will we be able to? [↑](#footnote-ref-26)
28. The repo is couchdb based, so [the regular couchdb replication](http://clock.co.uk/tech-blogs/how-to-create-a-private-npmjs-repository) can be used to replicate the [main couchdb instance](https://www.npmjs.org/doc/misc/npm-registry.html). [↑](#footnote-ref-27)
29. Recently got refreshed, but still has very poor ranking, limited package info, etc. [↑](#footnote-ref-28)
30. Limited package info and functionality. [↑](#footnote-ref-29)
31. One has to contact the bower developers and hope that they have the time to do it. [↑](#footnote-ref-30)
32. It seems [unlikely that npm will add support for namespacing](https://github.com/npm/npm/issues/798) using the github model. We could follow the grunt model and declare ng- prefix to be only for core packages, and angular- prefix for anything associated with Angular. [↑](#footnote-ref-31)
33. Bower primarily depend on [Heroku](https://status.heroku.com) & [Github](https://status.github.com)’s uptime ([reference](https://dataclips.heroku.com/byvmrorsycxmclubeuzxtlckegvm#Bower-packages-git-hosting-services)). In order for bower install to succeed all repositories that dependencies are being fetched from must be available. This means that the system has N points of failure. [↑](#footnote-ref-32)
34. There have been issues, but replication can mitigate them. [↑](#footnote-ref-33)
35. Force-pushing to a repo or deleting a tag is enough to break everyone depending on the affected version of the package. [↑](#footnote-ref-34)
36. There have been issues in the past, there is an effort to deal with them. [↑](#footnote-ref-36)
37. Uses appengine, bigtable, but has yet to be tested with really massive traffic [↑](#footnote-ref-37)
38. usually **no**, because most packages use git:// protocol for lookup which is not secured, https:// is commonly used for file download though [↑](#footnote-ref-38)
39. uses https:// for all the communication [↑](#footnote-ref-39)
40. Users compressed tarballs for github hosted packages and git (less efficient) for others. [↑](#footnote-ref-40)
41. All packages are distributed as tar.gz archives. [↑](#footnote-ref-41)
42. Even very simple projects take 1.5-6s to "update". [↑](#footnote-ref-42)
43. Keeps on trying to check the git repos. [↑](#footnote-ref-43)
44. There is [dedupe](https://www.npmjs.org/doc/cli/npm-dedupe.html) package which might work, but we need to investigate. [↑](#footnote-ref-44)
45. [shrinkwrap](https://www.npmjs.org/doc/shrinkwrap.html) has a reputation for being buggy, but [lockdown](https://www.npmjs.org/package/lockdown) looks like a good alternative.

    update: lockdown is buggy too, but shrinkwrap [is usable](https://github.com/angular/angular.js/pull/6653). [↑](#footnote-ref-45)
46. [npm registry contains "shasum" for all tarballs, npm-seal](https://github.com/zaach/npm-seal) and [lockdown](https://www.npmjs.org/package/lockdown) support checksum verification. [↑](#footnote-ref-46)
47. npm's [postinstall](https://www.npmjs.org/doc/misc/npm-scripts.html) and postupdate only work for installing or updating a package, but they don't get invoked when a dependency is installed (via `install --save`) or updated. [↑](#footnote-ref-47)
48. It is not well documented, but it is possible to query the repo by keywords ([example](https://bower-component-list.herokuapp.com/keyword/web-components)). It could be possible to modify [bower-search](https://github.com/bower/search-server) server to enable better querying. [↑](#footnote-ref-48)
49. It's not clear if we can query by any metadata. `npm search` doesn't seem to support that. [↑](#footnote-ref-49)
50. It's usually used with CommonJS modules but there is no requirement to use it. There is a discussion to [support ES6 modules](https://github.com/npm/npm/issues/4323). [↑](#footnote-ref-50)
51. Designed to work with CommonJS. However it's possible that it's not a hard requirement. [↑](#footnote-ref-51)
52. But not a JS community :-) [↑](#footnote-ref-52)
53. In our experience the team is sometimes super responsive other times not sensitive to major issues affecting many production applications. :-( [↑](#footnote-ref-53)
54. We tried several times to propose major changes and even contribute resources, but we had no significant success. [↑](#footnote-ref-54)
55. Registry itself is available on Github but the jspm registry server hasn’t been yet open sourced but its planned to do so. [↑](#footnote-ref-55)
56. Via PR on Github<https://github.com/jspm/registry/> [↑](#footnote-ref-56)
57. jspm allows linking in any registry like github or npm and there won't be any naming conflicts between them e.g. github:angular/angular.js won't be conflicting with a npm:angular [↑](#footnote-ref-57)
58. jspm has no own file repository. Instead it allows adding enpoints so that you can use any file repository of your choice. jspm comes by default with an endpoint for npm and github. Hence the availability, replication immutability, scalability etc. strongly depends on the endpoint you’re using. [↑](#footnote-ref-58)
59. The npm and github endpoints are using https but the transport layer security depends strongly on the endpoint you’re using. [↑](#footnote-ref-59)
60. Depends on endpoint but the github endpoint downloads archives instead of cloning the repos. [↑](#footnote-ref-60)
61. Depends on endpoint. [↑](#footnote-ref-61)
62. jspm supports caching of packages and an offline mode flag (--offline) will be soon available. [↑](#footnote-ref-62)
63. The version constraint system is based on constraint solving the freshest versions possible with minimal forks. [↑](#footnote-ref-63)
64. You can write an own endpoint which performs special actions after installing a package but adding hooks to existing endpoints is not yet possible. [↑](#footnote-ref-64)
65. jspm uses SystemJS as module loader which supports by default es6, cjs and amd. But via plugins other module loaders can be added similar to RequireJS. [↑](#footnote-ref-65)
66. jspm is coupled to the SystemJS, but you can also just use URLs directly e.g. ./jspm\_packages/angular@1.2.3/some.css or any other asset. Plugin-based asset bundling build workflows are currently in development. [↑](#footnote-ref-66)
67. https://groups.google.com/forum/#!topic/jspm-io/Sm2H3mdE7WI [↑](#footnote-ref-67)