Client-side framework selection criteria for progressive decoupling

This document's goal is to aid the Drupal community in choosing a JavaScript framework best-suited for **progressive decoupling**, where only parts of the page are rendered through the framework while key integration capabilities with Drupal configuration, modules, and themes are maintained. In short, Drupal-rendered components ought to coexist gracefully with framework-rendered components.

Importance legend

M (must have)
S (should have)
N (nice to have)

Further reading: "Selecting a client-side framework for Drupal" — "Should we decouple Drupal?" — "The future of decoupled Drupal"

Criteria		Most promising			Promising	Least promising		
Criterion (importance)	**Why it matters**	**Angular 2** (MVC, 2.0.0-beta.0)	**Ember** (MVC, 2.2.0)	**React** (V, 0.14.3)	**Elm** (lang, compile-to-JS)	**Backbone** (MV, 1.2.3; in core)	**Angular 1** (MVC, 1.4.8)	**Knockout** (MVVM, 3.4.0)
Server-side rendering of templates (M)	Server-side rendering of framework templates is important for SEO and performance. Over time, it allows us to migrate to unified templates across client and server and a server-side JavaScript-based render, whether via in-PHP JS execution or via a thin layer of Node.js.	Built-in	Add fastboot	Built-in	DIY with Nashorn, Rhino, or Node.js	Add rendr	Add -server	Add prerendered
Rehydration / seamless state transfer (M)	Does the client-side code discover and reuse HTML rendered during server-side framework execution without incurring an additional rerender?	Planned	In progress	Yes (add fluxible)	No (DIY)	No (DIY)	In progress	Yes (add prerendered)
Server-side rendering of app itself (M)	Can an app written to run in the client be rendered in the server without code changes? Doing this requires ecosystem-wide coordination around e.g. data-fetching and build tools.	Yes (built-in)	Yes (add fastboot)	No (add Flux impl and DI tool)	DIY with Nashorn, Rhino, or Node.js	Yes (add rendr)	Yes (add -server)	Yes (add prerendered)
Small payload size: TodoMVC JS as of 12/29 (M)	Large frameworks incur an initialization cost and deplete mobile batteries faster. Smaller file sizes mean less JS to download, interpret and execute, whether on the client or the server. All frameworks are deeply concerned with this (esp. globally-focused Google and Facebook), so this may be a non-issue in the medium term.	168KB (min+gzip@level=9); code size is p1 before end of beta	217KB (min+gzip@level=9)	No data (React needs add-ons)	37KB (min+gzip@level=9)	45KB (min+gzip@level-9)	49KB (min+gzip@level=9)	25KB (min+gzip@level=9)

Execution performance: TodoMVC (M)	An ideal framework should execute common application tasks (such as those found in TodoMVC) quickly. Leo Horie (Mithril) benchmarked some of these against a single TodoMVC app, but this relies on outdated releases. Read more about benchmark reliability.	No data	780ms (1.4.0 with Handlebars 1.3.0)	308ms (0.10.0)	266ms (0.16.0)	204ms (1.1.2)	344ms (1.2.14)	131ms (3.1.0)
Interoperability (M)	Frameworks that encompass the entire page rather than only encompassing portions of the page are less well-suited to a progressively decoupled approach, as Drupal would need to cede all control over renders and would be unable to render parts of the page in PHP/Twig.	Entire page or parts of page	Entire page or parts of page	Entire page or parts of page	Entire page or parts of page	Entire page or parts of page	Entire page or parts of page	Entire page or parts of page
Template engine friendliness for Drupal themers (M)	A declarative approach could be beneficial to progressively decouple UIs that are still migrating, while string-based templates are ideal for larger page components. How friendly is the templating system for Drupal themers? Does it work well for interpolation into existing Twig files?	Declarative within DOM (ng-) **and** string templates	Declarative within DOM (data-ember) **and** Handlebars (string templates)	JSX (string templates); JSX is optional but strongly rec'd	Declarative syntax through elm-html	Choose your own (string templates)	Declarative within DOM (ng- in flat HTML)	Declarative within DOM or string templates
Code structure unopinionated-ness (M)	A framework's opinionatedness about application structure means easy optimization, but it may be overly restrictive for an approach that favors "pick and choose" (library) over "the whole nine yards" (framework).	More opinionated (framework approach)	Most opinionated (framework approach)	Less opinionated (library approach)	Not opinionated (language). Suggested project structure.	Less opinionated (library approach)	More opinionated (framework approach)	Less opinionated (library approach)
Software licensing (M)	Drupal is free software using a GPLv2+ license. An ideal framework would be compatible with this licensing, insofar as it can be distributed singly with Drupal.	Apache 2.0 (compatible w/ GPLv3+)	MIT	BSD (free forks not required)	BSD3 (free forks not required)	MIT	MIT	MIT
Patent rights (M)	An ideal framework should lack a restrictive patent clause that prevents its use under unrelated conditions.	None	None	Restrictive	None	None	None	None
Client-side routing (M)	Client-side routing provides full URL support for SEO, back button functionality, and bookmarking. The lack of this makes navigation less easily introspected and breaks a fundamental aspect of the web.	Built-in	Built-in	react-router (3rd-party library)	elm-router-hash elm-route-parser	Built-in	Built-in	DIY (various 3rd-party libraries)

Nestable components (M)	Nestable components are important for elegant decomposition of complex UIs into manageable hierarchies of smaller portable, encapsulated pieces. Also see “Future readiness” below for discussion of Web Components support.	Yes	Yes	Yes	Yes	No (marionette)	Yes (.component method)	Yes
Robust state management (M)	The framework’s state system should not trigger a full DOM rerender, which is bad for performance. Instead, it should perform partial rerenders (only those components that have changed). Using DOM diffs off the page instead of model diffs may be a performance bottleneck.	Model diffing (with JIT compilation)	Value diffing (Handle-bars)	DOM diffing (Virtual DOM)	DOM diffing (virtual-dom)	Manual rerendering	Model diffing	View model diffing (with observables)
Robust REST support (M)	Frameworks either have built-in syntax for REST calls or enforce the use of jQuery (dependency) to fetch data from a service. Some eschew optimistic feedback by saving client-side data only once the server request is sent, not ideal for apps in disconnected environments (mobile or offline-first).	Built-in (in progress, better after beta)	Built-in	Add fetch or isomorphic-fetch	Through elm-http (maintained by author of Elm).	Built-in but syncs with server (no optimistic feedback; no offline; overridable)	Built-in ($http for broad AJAX, $resource for RESTful APIs)	Manual AJAX (knockout.mapping or $.ajax)
Testability (M)	Can we test our code using small, fast unit tests, using standard off-the-shelf tools, without excessive mocking? How well does it work with Drupal testing?	Good	Good	Good (also unexpected-react)	Good	Poor (DIY)	Good	Poor (DIY)
Data binding (M)	Two-way data binding allows for data updated from either the view or the model to be reflected in the view, but it often has a detrimental impact on performance. One-way data binding allows for a solely unidirectional flow and is usually adequate for most apps.	One-way data binding	Two-way data binding (one-way data binding will be default in 2.6)	One-way data binding (ReactLink for two-way)	One-way	None (getters and setters; DIY)	Declarative two-way data binding	Two-way data binding
Large community, ecosystem (S)	Corporate sponsorship or a large backing community help maintain a robust framework. A large ecosystem entails extensions, plugins, and other incidental projects that aid developers.	Large (Google)	Large	Large (Facebook)	Small	Medium	Large (Google)	Medium
Maturity (S)	Has the framework seen substantial adoption from many large enterprises? Also, does it have a long history of effective use in production?	Least mature	Mature	Most mature	Least mature	Most mature	Most mature	Mature

API docs and learnability (S)	Not only does the framework need to have an accessible learning curve for Drupal developers; front-end developers need to be able to use the framework efficiently and to integrate easily into the Drupal community.	Average (better by end of Q1 2016)	Good	Good	Good	Good	Good	Good
Debugging experience (S)	Developers desire a pleasant debugging experience such as a tool that aids not only in isolating errors and warnings but also a comprehensive inspector for the structure and execution of the application.	batarangle	Ember Inspector	-devtools	Time-travel debugger	-Debugger	batarang	chromeextensions-knockoutjs (low usage)
Error handling and reporting (S)	Developers require robust error reporting (e.g. compiler errors, runtime errors) to aid their debugging process. An ideal framework would provide exhaustive and helpful error reporting that minimizes blocked tasks.	TypeScript (statically typed): both compile-time and runtime errors	Built-in .onerror method (also -cli-honeybadger)	DIY (3rd-party libraries)	Strongly and statically typed, excellent compile-time errors, practically no runtime errors	DIY (no error handling OOTB)	Built-in error handling	Built-in .onError method
Native app support (N)	Frameworks increasingly have as part of their ecosystem the capability of compiling single-page JavaScript applications into native mobile applications written in Java and Objective-C. While this does not affect progressive decoupling (unless there is server-side JS), it is useful for full decoupling.	NativeScript Ionic 2 React Native	-cli-cordova (HTML5 to native)	React Native	In progress	None (DIY)	Ionic	None (DIY)
Future readiness (N)	An ideal framework should have a plan in place to either provide a polyfill for or directly support Web Components, Shadow DOM, and upcoming versions of JavaScript (ES6, ES7).	Excellent (ES6 support, WC-like syntax)	Excellent (ES6 support, WC-like syntax)	Average (Maple.js for WC)	Excellent	Poor	Poor	Good (WC-like syntax)
Backwards compatibility (N)	Ideally, a framework should be backwards-compatible with all previous versions in order to avoid incurring significant development costs later. In addition, an ideal framework should be using semantic versioning (semver).	First version is current; full semver	Fully backwards compatible; full semver	No backwards compatibility (0.x.x); full semver	No backwards compatibility (0.x.x); full semver	Full backwards compatible; full semver	No backwards compatibility; semver only recently	Full backwards compatibility; full semver
Release cadence (N)	Less frequent releases will alleviate the need for modules like jQuery Upgrader,	First version in beta 2	Minor every ~6w	Minor every ~6m	Minor every ~6m	Minor every ~6-12m	Minor every ~2w	Minor every ~6m

	which addressed the inability to retain easy dependency management in core. However, faster releases that are backwards-compatible are good for the framework; it simply creates friction in Drupal's management of dependencies.		(long-term support every ~6m)					
Totals		16 green 4 yellow 2 red 3 n/a	16 green 7 yellow 2 red 0 n/a	14 green 5 yellow 5 red 1 n/a	15 green 6 yellow 4 red 0 n/a	11 green 6 yellow 8 red 0 n/a	16 green 6 yellow 3 red 0 n/a	13 green 9 yellow 3 red 0 n/a

Other considerations for standardization:

- Build and deploy tools (Gulp, Grunt, Ember CLI)
- Dependency injection (Webpack, RequireJS, Browserify)
- Development dependency management (NPM)
- Health of module/add-on/plugin ecosystem (Ember Observer)

Special thanks to the following experts who provided review and input:

- Miško Hevery (creator of Angular; Google)
- Igor Minar (technical lead for Angular; Google)
- Ed Faulkner (core maintainer for Ember)
- Amitai Burstein (Drupal and Elm contributor; Gizra)
- Sebastian Siemssen (Drupal contributor, Elm and React developer; Zensations)
- John Albin Wilkins (Drupal 8 mobile initiative lead)
- Alex Bronstein (Drupal core maintainer; Acquia)
- Wim Leers (Drupal core contributor; Acquia)
- Dries Buytaert (Drupal project lead; Acquia)
- Preston So (Drupal contributor; Acquia)