THOMAS ST.

FAMO.US

Famous-Angular

Goals:

- Allow famo.us components to work seamlessly with other components inside existing or future Angular apps.
- Maintain the same high standards of code-quality, developer experience, and performance as the core Famo.us project.
- Allow an Angular developer to write a Famo.us app while separating its view logic from the rest of its logic
- Create that view declaratively, using the DOM to represent the tree structure of an app -- bringing the declarative magic of Angular to Famo.us.
- Work with the Famo.us team to make sure Famous-Angular supports Famo.us' long-term goals and vision.

Approach:

- **AMD:**
    - **Challenge**: Having AMD in the core of the library requires apps built on that library to use AMD as well. Although Angular apps *can* be built around AMD, it adds an unnecessary level of complexity to authoring an app, since Angular already has a powerful module-loading system through dependency injection. These two systems needed to be consolidated.
    - **Solution**: To work with AMD, modules are loaded asynchronously at bootstrap time, and then passed into an angular provider in RequireJS's require callback. This allows all of the references brought in by RequireJS to be accessible throughout the angular app simply by injecting the `famous` service and calling it as a hash: `var Surface = famous['famous/core/surface'];`

      We can also use the Angular module system plus Bower to reduce the weight of Famous-angular projects by allowing developers to install individual Famo.us AMD modules, and load them as Angular modules.

- **Rendering:**
    - **Challenge**: Angular has strong opinions about how it renders its views. So does

famo.us. This needed to be managed.

- **Solution**: Famo.us does views better than Angular. Fortunately, thanks to Angular's modular design, there's no reason that Angular needs to be involved in the view rendering directly. Our approach to solve this was to create parallel DOM elements for the Angular and the Famo.us components inside a given famo.us app. Angular's view is hidden from the DOM (`display: none;`.) Angular manages structure, communication, and gluing together references, while Famo.us handles everything related to rendering--there is no point at which Angular's watch mechanisms or template interpolation interfere with or hijack Famo.us's render cycle.

- **Simplicity:** Creating a Famo.us app inside angular is as simple as adding a `<fa-app></fa-app>` tag to an existing Angular app. Behind the scenes, our adapter traverses the Angular DOM and determines parent-child relationships, adding the necessary elements with the appropriate hierarchy to the famo.us context. Developer-defined controllers can be added to any `<fa-...>` element, allowing the developer to continue to use Angular's controllers as the atomic unit of component logic.

Development Roadmap:

- Work with Famo.us engineering team to establish standards for code quality, developer experience, and performance so that the quality of Famous-Angular is as high as the core Famo.us project.
- Refactor the existing proof-of-concept code, abstracting duplicated code, and positioning the library for future extension.
  - With the knowledge we've gained from building this proof-of-concept, spend some extra time at the whiteboard and ensure that our path forward is clear and can progress side-by-side with Famo.us as it grows.
  - Abstract redundant directive features into a hybrid factory/delegate pattern
    - A service holds the shared behavioral logic for different components
    - Components that want to inherit logic pass their scope into the factory/delegate service. That service then mutates the scope, adding appropriate members to it. This allows shared logic to exist in exactly one place, easing maintenance and extension.
  - Clarify and clean up scope creation and structure to eliminate calls to $parent scopes from views, which is brittle in the face of refactoring/growth. Careful scope management on our part will eliminate the need for those calls.
- Continue creating declarative wrappers around as every famo.us component for which it makes sense to have a declarative representation --GridLayout, LightBox, Sync, EventHandler, Transitions, Physics, and on and on. No boilerplate should be needed for out-of-the-box, simple interactions between standard components.