Complex Forms

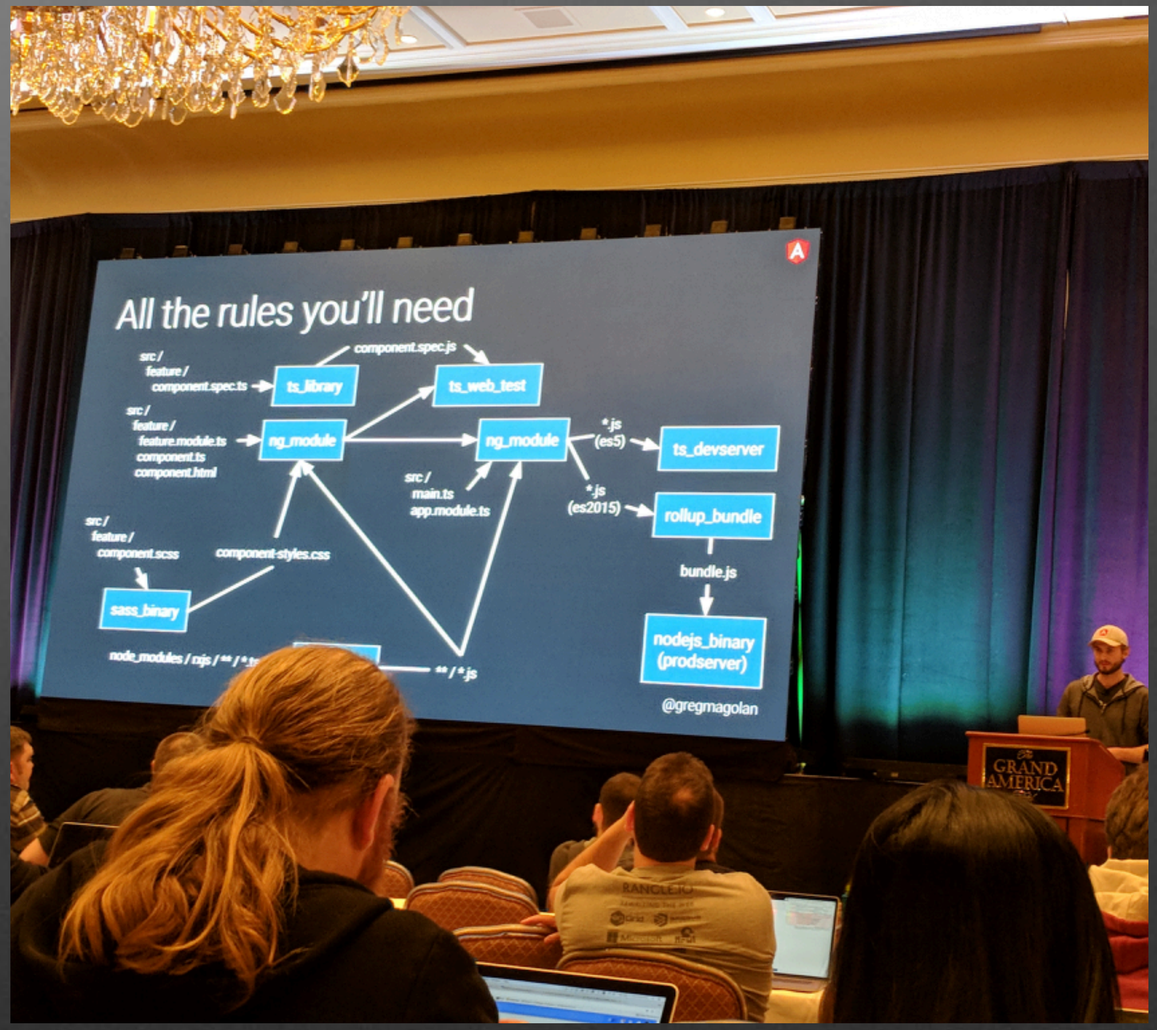
- One talk saying to use NgRx, one talk saying to not.
    - Argument to not use it was because FormGroup is already managed.
- Ionic using NgRx to store native state.
- Architecture patterns:
    - Publish / Subscribe pattern (pubSub) - eliminates that need for event emitters. I'm kind of doing that, but not in such an organized way.
    - Container gets data → Presentation patches values onto formGroup → Value Changes → Emit Event back to → Back to Container → Container decides when to add to store.
        - Container: Prevent unsaved data from going to the store.
        - Presentation: builds form and Output (Event Emitter) back to container
    - Custom classes to manage repeating forms / nested form groups

Bazel

I think we would need to approach this as an entire group. It is too much work for just the UI project.

Benefits of Bazel

- Fast at Scale
- Full-Stack - inputs are files, structure is the application, not “how” but “what”
- Builds depend on the size of the change, not the size of the app
- Customizable
  - Structure of the app
  - Paths for output
  - Plugins for the languages and tools
- Bazel rules
  - Input file → action → output fileIncremental Builds
  - Perform build steps/actions
  - Chained together rules and abstract complexities
- Roadmap:
  - Now via AngularLabs This Year: Ergonomics, Performance, Code-splitting
  - Later → converge with Angular CLI
- http://g.co/ng/abc http://g.co/ng/abc http://github.com/alexeagle/angular-bazel-example
- http://github.com/alexeagle/angular-bazel-example/wiki



Complex Forms

- One talk saying to use NgRx, one talk saying to not.
  - Argument to not use it was because FormGroup is already managed.
- Ionic using NgRx to store native state.
- Architecture patterns:
  - Publish / Subscribe pattern (pubSub) - eliminates that need for event emitters. I'm kind of doing that, but not in such an organized way.
  - Container gets data → Presentation patches values onto formGroup → Value Changes → Emit Event back to → Back to Container → Container decides when to add to store.
    - Container: Prevent unsaved data from going to the store.
    - Presentation: builds form and Output (Event Emitter) back to container
  - Custom classes to manage repeating forms / nested form groups