NgRx - Reducing Boiler Plate

NgRx Schematics works with the CLI. Blueprints that make it very easy to use.

NgRx Data

- Becoming an official NgRx package soon

- Really nice way to build out NgRx Schematics. Should look into it.

What belongs in the store: SHARI

- Shared - shared state is accessed by many components and services
- Hydrated - state that is persisted and hydrated from storage
- Available - state that needs to be available when re-entering routes
- Retrieved - state that needs to be retrieved with a side effect
- Impacted - state that is impacted by actions from other sources

What doesn't go in the store:

- Forms... WTF?
- Data that can't be serialized
- State that is solely owned by a component doesn't belong in the store.

Ideally

Application can be booted up with a specific state to recreate any scenario.

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

NgRx - Reducing Boiler Plate

NgRx Schematics works with the CLI. Blueprints that make it very easy to use.

NgRx Data

- Becoming an official NgRx package soon
- Really nice way to build out NgRx Schematics. Should look into it.

What belongs in the store: SHARI
- **Shared** - shared state is accessed by many components and services
- **Hydrated** - state that is persisted and hydrated from storage
- **Available** - state that needs to be available when re-entering routes
- **Retrieved** - state that needs to be retrieved with a side effect
- **Impacted** - state that is impacted by actions from other sources

What doesn’t go in the store:
- Forms… WTF?
- Data that can’t be serialized
- State that is solely owned by a component doesn’t belong in the store.

Ideally
Application can be booted up with a specific state to recreate any scenario.