State Management with Redux and @ngrx/store Best Practices

Alex Thalhammer

Best Pactices

#1: LoadStatus & Caching

- We have to distinguish between
  - „no data available“
  - „data not yet loaded“

- Possibilities
  - LoadStatus property in State
  - Rely that null means „not loaded“
  - ActivationGuard for Data Loading

#2: Container & Presentation Components

- Different types of Modules

- Dumb Component represent UI Components:
  - almost no injections (Exception like FormBuilder)
  - @Input & @Output)
  - Candidates for StoryBook

- Smart (Container) Component connects UI with Data-Access

#3: Facade

- NgRx is used only for data-access services
- Outside services and components shouldn‘t know anything about NgRx
- Hide all ngrx-specific elements and rely on plain Observables
- Each module should provide one Interface to communicate with the world

#4: StateModel vs. ViewModel

- Each UI component defines its own, individual ViewModel
- ViewModel can differ from State model (denormalization)
- Access to different States are required to generate the ViewModel
- We don’t want to render incomplete ViewModel
- Procedure:
    1. Wait until all stores return requested data
    2. Transform data
    3. Render UI component with full ViewModel

#5: Url as TRUE Source of Truth

- URL data always precedes Store data
- Expect your users (GoogleBot) to navigate in unexpected ways or enter via a deep link
- Always dispatch URL/State synchronisation
- Each component must send the relevant and current Url data to Store

Labs

NgRx Best Practices

Bonus Content

ngrx-normalizer

- Should be used when endpoints return denormalized data

- Can automatically transform denormalized data into normalized

ngrx-undo-redo

- Can revert actions
- Possibility to only allow certain actions to undo/redo
- Potential issues with
  - Changing routes
  - Localised state, e.g. state without ngrx