Angular and NgRx Development Reference Documentation

Release

Henrik Palmlund Wahlgren @ Palmlund Wahlgren Innovative Tech

Nov 13, 2017

Contents:

1 Angular **3**
- 1.1 Getting started . . . 3
- 1.2 Angular Cheat sheet . . . 3
- 1.3 Style Guide . . . 3
- 1.4 Animations . . . 3

2 Application State **5**
- 2.1 State Best Practices . . . 5

3 State Management with NgRx **9**
- 3.1 Organizing your state using NgRx Store . . . 9

4 TSLint Settings **11**

5 Indices and tables **13**

A guide to our new developers on how we build and structure our Angular projects

CHAPTER 1

Angular

Additional information on using Angular in our projects.

1.1 Getting started

There are tons of resources on how to learn Angular but start with the Angular Introduction to get a base knowledge of how Angular works: https://angular.io/guide/quickstart

1.2 Angular Cheat sheet

Angular has a lot of special syntax that you will have to learn. Use this cheat sheet as a quick reference: https://angular.io/guide/cheatsheet

1.3 Style Guide

Angular.io style guide is well made and should be used: https://angular.io/guide/styleguide

1.4 Animations

When building more complex animations try to use Angulars animations instead of pure css animations. When you start writing conditionals and extra code in components and templates it is a good point to start with Angular animations. https://angular.io/guide/animations

CHAPTER 2

Application State

2.1 State Best Practices

2.1.1 Structure the state after the frontend application, not the backend API.

The state should help you structure the data in the front end so it might not be the best to model it after how the data is represented in the backend.

2.1.2 Make the state as flat as possible

Having a nested state will introduce complexity. By making the state flat we will have a more maintainable solution

2.1.3 Use maps instead of arrays

We can treat our applications state as an in memory database. We want to query the database and then it is not efficient to go through arrays to find the data we need. But if it represented with a map we can use it easier.

```
// with Map
 objects = {
    1: {'name': 'Norrlands Guld', 'type': 'lager'},
    2: {'name': 'Guinness', 'type': 'stout'},
    3: {'name': 'Tuborg', 'type': 'lager'}
}

// with Array
objects = [
   {'id': 1, 'name': 'Norrlands Guld', 'type': 'lager'},
   {'id': 2, 'name': 'Guinness', 'type': 'stout'},
   {'id': 3, 'name': 'Tuborg', 'type': 'lager'}
]
```

2.1.4 Don't model the state for use in views

It might be a quickfix to model the state of your application after how you want to present it to the user. Different views might want to display the same data in different ways. By modeling your data for the views you might end up duplicating data in your state

2.1.5 Don't duplicate data in your state

You want your state to be the single source of truth. When we have dupliated data in the state will have to update several parts every time and eventually this will increase complexity and introduce errors and bugs. Make use of selector functions and custom filters to get the data you need.

2.1.6 Don't store derived data in your state

Again with the single source of truth. If we store derived state we have to keep it updated and this will increase complexity and might intrdduce bugs.

2.1.7 Normalize your data

We want to reduce the complexity and keep the state flat. It will be easier to maintain and add new functionality.

```
// denormalized
 users = {
  '1': {'name': 'Oscar',
       'registered_at': '2017-02-01T09:45:32',
       'groups': [
          {'id': 1, 'name': 'admin', 'description': 'Group for admins'}]
        },
  '2': {'name': 'Sara',
       'registered_at': '2017-05-04T10:20:42',
       'groups': [
          {'id': 1, 'name': 'admin', 'description': 'Group for admins'},
          {'id': 2, 'name': 'editor', 'description': 'Group for editors'}   ]
        },
  '3': {'name': 'Robert',
       'registered_at': '2017-12-01T14:45:40',
       'groups': [
          {'id': 2, 'name': 'editor', 'description': 'Group for editors'}]
        },
}

// normalized
users = {
  '1': {'name': 'Oscar',
       'registered_at': '2017-02-01T09:45:32',
       'groups': ['1']
        },
  '2': {'name': 'Sara',
       'registered_at': '2017-05-04T10:20:42',
       'groups': ['1', '2']
        },
  '3': {'name': 'Robert',
       'registered_at': '2017-12-01T14:45:40',
       'groups': ['2']
```

```
        },
}

groups = {
   '1': {'name': 'admin', 'description': 'Group for admins'},
   '2': {'name': 'editor', 'description': 'Group for editors'},

}
```

CHAPTER 3

State Management with NgRx

Our take on using NgRx. There are a lot of instructions on how to organize your state online and this is resources on how we decided to do it since there is many ways to implement it.

3.1 Organizing your state using NgRx Store

3.1.1 File structure

Todo

separate store folder MainReducer Folder per state slice, files per state slice. Add note to say it might be wrapped. make it ORMy with .objects

CHAPTER 4

TSLint Settings

CHAPTER 5

Indices and tables

- genindex
- modindex
- search