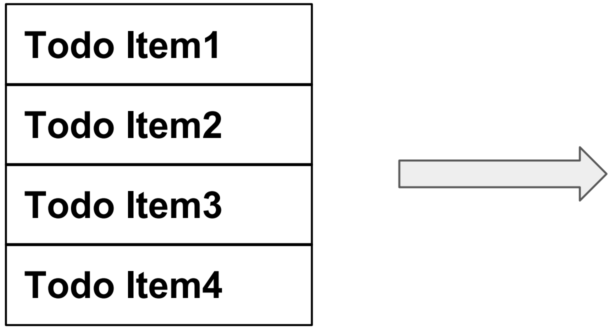
@ngrx/store

Reactive redux for angular

Our objective

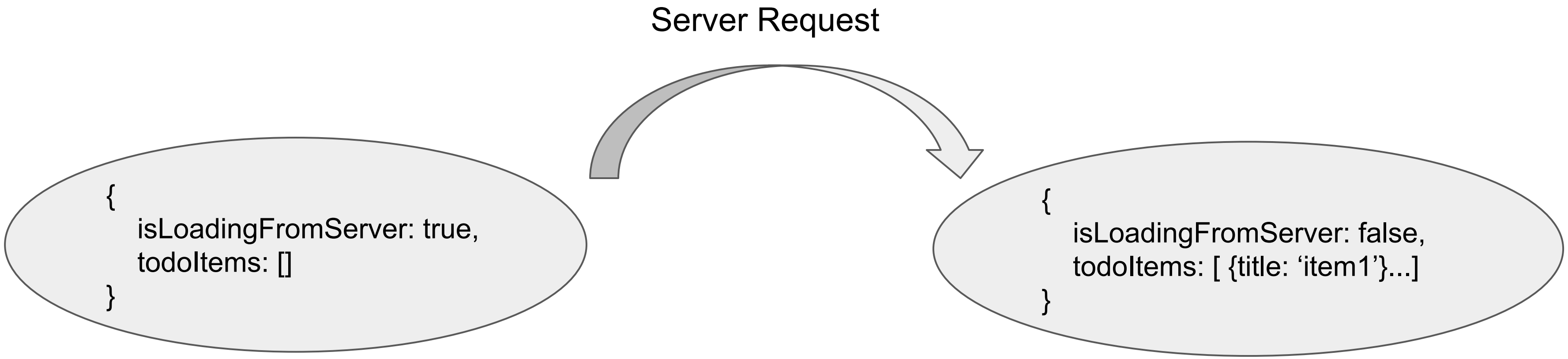
- following our redux lesson our objective:
  - create a todo list app
  - entire app with chage detection strategy OnPush

How our app will look



This list is from the server

How does our state look like



```
{
    isLoadingFromServer: true,
    todoItems: []
}
```

```
{
    isLoadingFromServer: false,
    todoItems: [ {title: ‘item1’}...]
}
```

What is @ngrx/store

- state management for angular applications inspired by redux
- we select items from the state and get them as observables
- we then use async pipe to display them in the components
- async pipe will cause rerender even if component is in change strategy on push
- install with npm:
  - **npm install @ngrx/store --save**

Task model and interface

- let’s start by creating our task model
- we will also create an interface for our model

Todo Service

- lets create a service with a method to grab all the tasks from the server

store as module

- similar to how we placed routing in separate modules, we will do the same with our state
- each module which needs to add items to our state will have that logic in a store module
    - **app-store** - will contain store module for the root module
    - **user-store -** will contain store module for the feature module called user
- inside each module we can split the reducers based on logical sections

Actions

- let’s start by implementing our actions
- we will the string types of the actions in a class with static properties called TodoActionsTypes
- each action will implement Action interface
- we will also use type alias to name the type of actions

Reducer

- we will define an interface that describes the section of the state that this reducer is in charge of
- we will define our initial state
- create our reducer as a pure function with switch and case
- the pure function will use the types defined in the action file

combining reducers

- we will create a file called **reducers**
- in this file we will describe the entire state of the app-store module
- we will create **ActionReduceMap<AppInterface>** which is a dictionary which maps keys to reducers
-

Store Module

- we will call **StoreModule.forRoot** to create our store
- it will get the reducer map we created before
- for lazy loaded module their state will be lazy loaded as well and we will call the function **StoreModule.forFeature**
- we will add our store module to the app module
- we can now inject the store to our services and components

modifying the service

- we can inject in out todo service the store
- in the **map** operator when we get the tasks from the server we can call the **dispatch** method with our action to add the tasks to the state
- in the app component call the service get tasks to initiate the state with the tasks from the server

todo list component

- create a component to display the list of todo tasks
- inject the store to the component
- we will use the select method on the store to select certain property from the state
- the property will return to us as observable
- we will use the async pipe to display that observable

Summary

- with redux we can manage the state of our app
- the only way to change the state is by calling **store.dispatch(action)**
- reducers will decide how the state changes
- combining angular and redux is a powerful tool which gives us
    - more testable app
    - better performance
    - easier management of the data of our app