redux

State manager

What we will cover

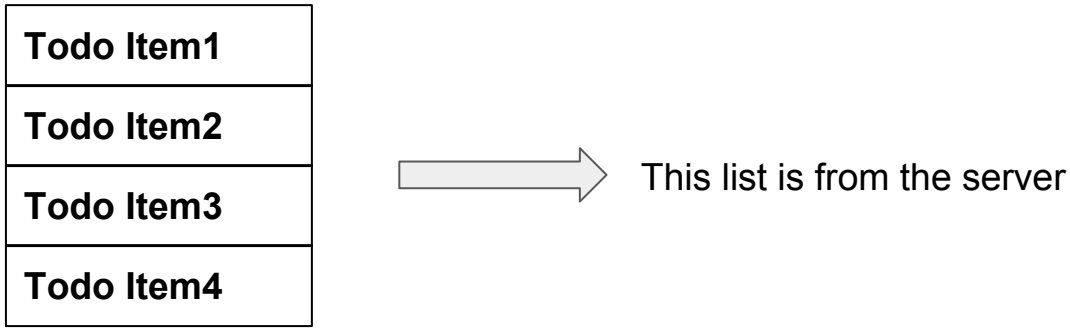
- what is a state
- what is redux
- redux architecture
  - reducers
  - actions
  - store
- middlewares
- installing our first middleware
- dev tools

What is a state

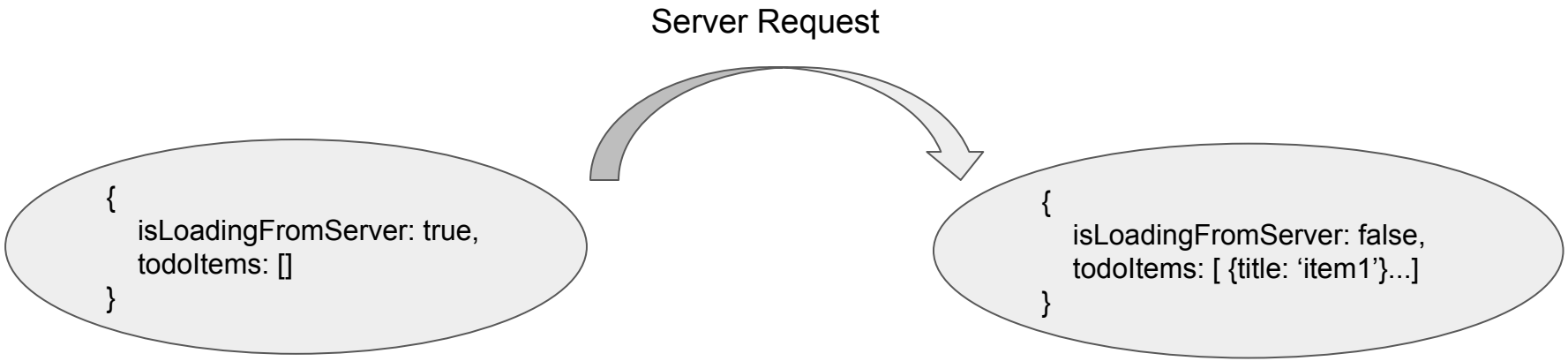
- in college at automation course we learned about Finite-state machine (FSA)
- FSA is a mathematical model of computation
- FSA can be in one state out of a group of final states at any given time
- FSA can transition to a new state based on certain change in input
- Can we look at our frontend application as an FSA?
- What can be the cause of a transition in state in a frontend web app?

Todo app state

- in our todo app we have a component that displays the list of todo items
- at the beginning the list is empty and we query the server for the list of items
- after the server return our response we display the list of tasks
- the todo list component looks like this:



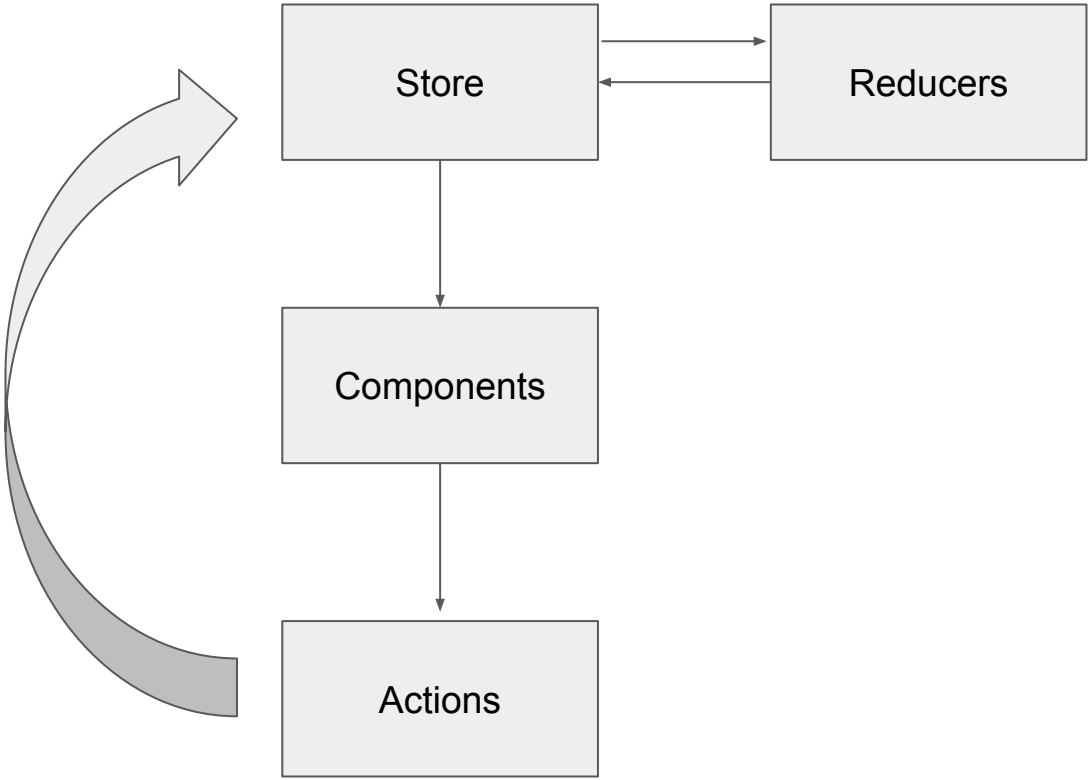
State for todo app



What is Redux

- predictable state container for JS apps
- using redux we have a single object holding the state of our app which is called **store**
- the state in the store can be an Object, Array, Primitive but it's highly recommended that it will be immutable
- The only way to change the state is by calling a method on the store called **dispatch**
  - **store.dispatch(action)**
- An action is a simple object describing **what happened**
- The store decides how the state will change using a pure function called reducer
- The reducer answers the questions **How does the state change?**
  - **(state, action) => state**

Redux Architecture - UniDirectional Data Flow



Redux Core

Redux core concepts are:

- Actions
- Reducers
- Store

Let’s try to demonstrate redux usage using the state of our todo application

Actions / Action Creators

- action is a simple JS Object
- the action needs to hold the following information
  - identifier of the action (string)
  - data associated with the action
  - example: **{type: ‘SET_LOADING’, payload: true }**
- action creator is a pure function which returns an action

```
function setIsLoading(isLoading) {

    return {type: ‘SET_LOADING’, payload: isLoading};

}
```

- What action creators do you think our todo app requires?

Reducers

- reducers are pure functions
- reducers get the current state and action as arguments
- the reducer need to decide based on the action and the current state what the next state will be
- the state has to be immutable to guarantee that no one will mutate the state in the reducer we will use a library called **immutable.js**
- when our state grows if we use only a single reducer the reducer will be huge so it is better to split the state to sections and also split the reducers
- redux has a function called **combineReducers** to help us split the app and make each reducer in charge of certain section of the state
- Before we dive into reducers let’s do a small intro on **immutable.js**

immutable.js

- immutable data cannot be changed after created
- immutable.js offer us api to change the data but instead of changing the data it will yield a new immutable object
- immutable.js offer us immutable data structures we will cover the popular ones:
  - Map
  - List
- We can install immutable.js with npm
  - **npm install immutable --save**
- let’s try and create a map and a list and see the common api for those data structures

Todo Reducer

- Our todo app will contain a single reducer
- When initiated redux will call the reducer with the state undefined so we can utilize this fact to set our initial state
- let’s create the reducer for our todo app

combineReducers

- when our state grows so will our single reducer
- We want to split the state to logical sections
- let’s say our todo application will need to hold information about the current user
- we will create another reducer that will handle the user info and combine those reducers with the **combineReducers** function

Store

- there is a single store in redux application
- the store holds the state of our app
- to create the store we use **createStore**
  - **createStore(reducers, initialState)**
- we need to pass to the **createStore** the combined reducers and optional initial state
- Let’s create our store

Redux Middlewares

- the only way to change the state is using the **store.dispatch** method
- a middleware will decorate the **dispatch** method and add logic before and after the store calls the reducer
- there is a function in redux called **applyMiddleware** which takes the list of middlewares we want to apply
- the result is enhanced **dispatch** method which we place as the third argument in the **createStore**
- we won't cover how to create middleware of our own cause in common use cases we will use community middlewares
- let's install our first middleware

redux-thunk - async actions and query server

- redux-thunk is a redux middleware
- it allows action creators to return a function
- if action creator is returning a function it will be run by thunk middleware
- the function returned will be called with 2 arguments
  - **dispatch**
  - **getState**
- we can use redux thunk to delay store dispatch, condition the dispatch, run async code like server communication
- we can install it with npm: **npm install redux-thunk --save**
- we need to install it as a middleware to our redux store
- let’s create an action which queries the server for todo items

Redux dev tools

- one of the strong features of redux is the easy of testing and easy of development
- you have a state and the app should behave according to the state
- for development purpose it’s better if we can easily examine the current state and all the actions that got us to this state
- there is an easy to use browser extension:
  - https://github.com/zalmoxisus/redux-devtools-extension
- to install the devtool extension: **npm install redux-devtools-extension --save-dev**

```
const store = createStore(reducer, composeWithDevTools(
  applyMiddleware(...middleware),
  // other store enhancers if any
));
```

Summary

- we covered the core concepts of redux
- redux has it’s core package and also framework specific packages to connect it to different frameworks
- the next lesson will focus how to combine redux with angular and the benefits of using redux with angular