REDUX & ANGULAR WORKSHOP

Nir Kaufman
Nir Kaufman
Head of Angular Development @ 500Tech
nir@500tech.com
@nirkaufman
il.linkedin.com/in/nirkaufman
github.com/nirkaufman
Agenda

Understanding Redux
Quick Angular overview
Implementing common scenarios
Best practices & next steps
THE CHALLENGE
SPA BECOME INCREASINGLY COMPLICATED
THERE IS NOTHING WRONG WITH THE MVC PATTERN
IF YOU ARE BUILDING A **CRUD** APPLICATION
<table>
<thead>
<tr>
<th>First name</th>
<th>Last name</th>
<th>Email</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gabrielle</td>
<td>Patel</td>
<td><a href="mailto:gabrielle@patel.com">gabrielle@patel.com</a></td>
<td>Imported</td>
</tr>
<tr>
<td>Brian</td>
<td>Robinson</td>
<td><a href="mailto:brian@robinson.com">brian@robinson.com</a></td>
<td>Customer</td>
</tr>
<tr>
<td>Eduardo</td>
<td>Haugen</td>
<td><a href="mailto:eduardo@haugen.com">eduardo@haugen.com</a></td>
<td>Contacted</td>
</tr>
<tr>
<td>Koen</td>
<td>Johansen</td>
<td><a href="mailto:koen@johansen.com">koen@johansen.com</a></td>
<td>Contacted</td>
</tr>
<tr>
<td>Alejandro</td>
<td>Macdonald</td>
<td><a href="mailto:alejandro@macdonald.com">alejandro@macdonald.com</a></td>
<td>Contacted</td>
</tr>
<tr>
<td>Angel</td>
<td>Karlsson</td>
<td><a href="mailto:angel@karlsson.com">angel@karlsson.com</a></td>
<td>Contacted</td>
</tr>
<tr>
<td>Yahir</td>
<td>Gustavsson</td>
<td><a href="mailto:yahir@gustavsson.com">yahir@gustavsson.com</a></td>
<td>Contacted</td>
</tr>
<tr>
<td>Haiden</td>
<td>Svensson</td>
<td><a href="mailto:haiden@svensson.com">haiden@svensson.com</a></td>
<td>Imported</td>
</tr>
<tr>
<td>Emily</td>
<td>Stewart</td>
<td><a href="mailto:emily@stewart.com">emily@stewart.com</a></td>
<td>Contacted</td>
</tr>
<tr>
<td>Corinne</td>
<td>Davis</td>
<td><a href="mailto:corinne@davis.com">corinne@davis.com</a></td>
<td>Customer</td>
</tr>
<tr>
<td>Ryann</td>
<td>Davis</td>
<td><a href="mailto:ryann@davis.com">ryann@davis.com</a></td>
<td>NotContacted</td>
</tr>
<tr>
<td>Yurem</td>
<td>Jackson</td>
<td><a href="mailto:yurem@jackson.com">yurem@jackson.com</a></td>
<td>Closed</td>
</tr>
<tr>
<td>Kelly</td>
<td>Gustavsson</td>
<td><a href="mailto:kelly@gustavsson.com">kelly@gustavsson.com</a></td>
<td>Contacted</td>
</tr>
<tr>
<td>Eileen</td>
<td>Walker</td>
<td><a href="mailto:eileen@walker.com">eileen@walker.com</a></td>
<td>NotContacted</td>
</tr>
</tbody>
</table>

**Edit customer**

- **First name**: Brian
- **Last name**: Robinson
- **Email**: brian@robinson.com
- **Birth day**: 10/20/58
- **Gender**: Male
- **Status**: Customer

**Buttons**

- Save
- Cancel
BUT WE ARE PUSHING THE ENVELOPE AS MUCH AS WE CAN
MANAGING AN EVER-CHANGING STATE IS A HARD TASK
EVERYTHING IS CONNECTED TO EVERYTHING
CHANGING SOMETHING BREAKS SOMETHING SOMEWHERE ELSE
ENTER REDUX
REDUX IS A LIBRARY FOR IMPLEMENTING A DESIGN PATTERN
REDUX ATTEMPTS TO MAKE STATE MUTATIONS PREDICTABLE
INSPIRED BY
FLUX, CQRS &
EVENT SOURCING
REDUX INTREDUCES
THREE PRINCIPLES
A SINGLE SOURCE OF TRUTH

The state of your whole application is stored in an object tree within a single store.
THE TRUTH IS OUT THERE
class SideBarComponent {

    private visible: boolean;

    toggle() {
        this.visible = !this.visible;
    }
}
class TabsComponent {

    private activeTab: Tab;

    activateTab(tab) {
        this.activeTab = tab;
    }
}

```typescript
class Accounts {

    private accounts: Account[];

    getAccounts() {
        return this.accounts;
    }
}
```
const state = {
  tabs: [],
  accounts: [],
  sidebar: {}
};
stateless UI

app state
THE STATE IS READ-ONLY

The only way to mutate the state is to emit an action - an object describing what had happened.
class Store {

    private state: Object;

    getState() {
        return this.state;
    }
}
getState()
dispatch(action)

getState()
PURE FUNCTIONS

To specify how the state tree is transformed by actions, you write pure functions.
PURE FUNCTION

The return value is only determined by its input values, without observable side effects.
Calculate the next state
Calculate the next state

Current State

Action

PURE FUNCTION

Next State

Reducer
Unidirectional data flow

UI → actions → STORE

state
ENTER THE STORE
THE STORE IS THE HEART OF Redux
TO CREATE A STORE
WE NEED A REDUCER
import { createStore } from 'redux';

const store = createStore(reducer);
import { createStore } from 'redux';

const store = createStore(reducer);
REDUCE METHOD

Apply a function against an accumulator for each element of the array (from left-to-right) to reduce the array to a single value.
function sum (previousVal, currentVal) {
    return previousVal + currentVal;
}

[0, 1, 2, 3, 4].reduce(sum);
// => 10
EVENT SOURCING

Capture all changes to an application state as a sequence of events.
Simple counter app

```javascript
function counter (state, action) {
    switch (action) {
        case 'up':
            return state + 1;
        case 'down':
            return state - 1;
        default:
            return state;
    }
}

['up', 'up', 'down'].reduce( counter, 0 );
```
THE REDUCER RETURNS THE NEXT **STATE**

BASED ON A SEQUENCE OF **ACTIONS**
THE SAME SEQUENCE OF ACTIONS
WILL PRODUCE THE SAME STATE
PREDICTABLE
STATE CONTAINER
STORE API

- dispatch(action)
- subscribe(listener)
- getState()
- replaceReducer(reducer)
HANDS ON!

Implementing a working store in less than 30 lines of code.
function createStore(reducer) {
  let state = null;
  const listeners = [];

  function getState() {
    return state;
  }

  function dispatch(action) {
    state = reducer(state, action);
    listeners.forEach(listener => listener());
  }

  function subscribe(listener) {
    listeners.push(listener);
    return function unsubscribe() {
      let index = listeners.indexOf(listener);
      listeners.splice(index, 1)
    }
  }

  return { getState, dispatch, subscribe };
ASYNC DATA FLOW
MIDDLEWARE

An *extension point* between dispatching an action, and the moment it *reaches the reducer*. 
UI \[\text{action}\] \rightarrow \text{MIDDLEWARE} \[\text{action}\] \rightarrow \text{STORE}

\[\text{state}\]
export const middleware = store => next => action => {
  return next(action);
};

- get the current state from the store
- pass an action to the next middleware
- access the provided action
ANGULAR & REDUX
AN ANGULAR APP IS MADE UP FROM A TREE OF COMPONENTS
WE MAP PROPERTIES TO THE STATE
WE DISPATCH ACTIONS AS A REACTION TO EVENTS
COMPONENT

STORE

actions

state

(events)

[properties]
ANGULAR 2.0 ENCOURAGES AN OOP APPROACH
WE WRAP EVERYTHING IN CLASSES
WE USE ANGULAR DI FOR WIRING
import {createStore} from 'redux';
import {RootReducer} from './reducers/root';

export class Store {

    private store = createStore(RootReducer);

    get state() {
        return this.store.getState();
    }

    dispatch(action) {
        this.store.dispatch(action);
    }
}
WE REGISTER OUR STORE CLASS AS A PROVIDER
@NgModule({
  declarations: [AppComponent],
  providers : [Store],
  bootstrap : [AppComponent]
})
export class AppComponent {

  constructor(store: Store) {
    // use the store
  }
}
HANDS ON!

- Create a provider for the store
- Wire the redux dev tool extension
- Test store to the app component
git checkout 00_setup

wire a store, root reducer and the debugger
COMMON SCENARIOS
THE SYNC CRUD

- Action creator provider
- List reducer
- List component
- Wiring component to the state
git checkout 01_crud
THE CRUD FEATURE MODULE

- Create a module for the list
- Create an Item component
- Set change detection strategy
git checkout 02_onPush
ASYNC FLOW - LOGIN FORM

- Module for the login form (with FormModule)
- User action creator
- Auth middleware
- User reducer
- Login form
- (optional) chaining middlewares
git checkout 03_login
APPLICATION STATE

- Module for shared components
- Application Action creators
- Application reducer
- Loader component (spinner)
- Alerts component (optional)
git checkout 04_loader
NEXT STEPS
RESOURCES

REDUX
http://redux.js.org/
https://egghead.io/series/getting-started-with-redux

CQRS & EVENT SOURCING

ANGULAR 2
angular-2-change-detection-explained.html
https://github.com/ngrx/store
https://github.com/angular-redux/ng2-redux
THE COMPLETE REDUX BOOK