### Redux

Managing Client State with Angular/React

### MiQ Motivations

- Common client state management between Angular and React
- Consolidate client state into a common global store

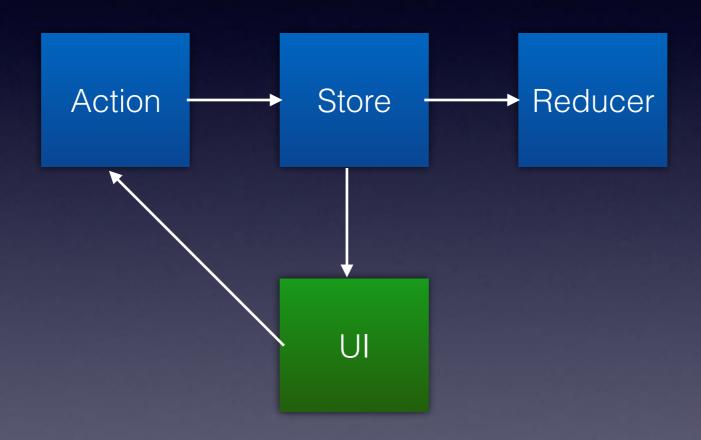
## Components Downside

- Component architectures spawn distributed state in components everywhere — Component Spaghetti
- Angular components only know about their @input passed into them
- This is further complicated by component mutation via events propagation (@output EventEmitter) at any time (async in no dependable order) - leads to Heisenbugs
- After clicking around several screens we really don't know what state our application is in anymore (or every screen has to re-request everything to know we are consistent)
- New features introduce new Events which introduce unpredictable behavior for the UI

## Redux Key Concepts

- Immutable State Management
- Reducers are Pure functions (functional, no side effects)
- Reducers don't change the state they return a new state: Reducer(state, action) -> Store
- Action is a type and optional payload
- State change is observed by subscribing to Observables (Rx.js)

## Redux Key Concepts



# Conceptual

 Conceptually the **store** can be pictured as a hierarchical tree of data. And any part of the tree can be subscribed to when it changes

# Component Code

```
export class AppComponent {
  title = 'This is a sample app';
  @select counter;
  constructor(private ngRedux: NgRedux<IAppState>) {
  }
  increment() {
    this.ngRedux.dispatch({type: INCREMENT});
  }
}
```

```
<h1>
{{title}}
</h1>
Counter: {{counter | async}}
<button (click)="increment()">Increment</button>
```

### Store Code

```
export interface IAppState {
   counter: number;
}

export const INITIAL_STATE: IAppState = {
   counter: 0,
};

export function rootReducer(state, action): IAppState {
   switch (action.type) {
     case INCREMENT: return tassign(state, { counter: state.counter + 1});
   }
   return state;
}
```

### How Can It Help

- UI Framework isolation via common Actions versus framework centric Services
- Consolidate distributed component state into a single immutable source
- Possible to serialize all or parts of the store to a DB or browser DB to resume that state from previous session

#### Pros

- Consistent application behavior
- Single source of truth (only one immutable **store**, *instead* of lots of mutable components responding to events)
- Performance, common data may not need to re-queried because its already available in the store
- Undo/Redo is simple
- Actions are simple data structures with no logic
- State is serializable and reproducible at any point

### Cons

- Another Layer of code (more code/more moving parts)
- Normally not necessary to start out with Redux
- Different Mental Model to learn

## Debugging

- Time Traveling Debugger (Awesome!)
  - Consistently reproduce your application state for any point in time
- QE can even dump the state.json to reproduce exact state of application when bug occurred
- Testing is much easier without the need for Mocks or spies

### Is Redux Needed

- How much state does my application have and how many ways can that state be changed?
- How much data is shared among other views (think common headers/sidebars in different screens)

# Angular 2+

- Angular 2+ has a tunable changed state detection mechanism
- Since we know some components don't internal state
  we can turn this off and get a huge performance
  boost by not checking changed components state
  since it is external to the component with Redux
- changeDetectionStrategy.onPush Turns dirty checking off - ignore dirty checking until we have observed a new value from Redux

### Implementations

- ngRx/store: Written as an Angular 2 implementation of Redux
- ng2-redux: Angular 2+ bindings for Redux. Uses
  Redux and is compatible with much of the Redux
  ecosystem. This is the implementation that we need
  to use for React/Angular integration.
- Both use Rx.js
- Comparison: (<u>https://github.com/ngrx/store/issues/</u> <u>169</u>)

### Summary

- Redux is a useful tool to consolidate client state management between different UI technologies (such as Angular/React)
- The Redux Architecture is a big win for large Angular applications by organizing state into a single source of truth
- Keeps the application state predictable

### Resources

- https://github.com/ngrx
- https://github.com/angular-redux/store