# REDUX & ASYNCHRONICITY

*by Emily Intersimone*

# REDUX

*A predictable state container*

# REDUX GLOSSARY

- **Store**: a read-only **object** representing state
- **Action**: an **object** with a type, and optionally, some other info
- **Dispatch**: a store **method** that takes in an **action**, and passes that action to the reducer
- **Reducer**: a **function** that takes in **state** and an **action** and returns **new state**
- **Subscribe**: a store **method** that registers callbacks to execute when the store updates

USER INTERACTION → DISPATCH(ACTION)

REDUCER(OLD STATE, ACTION) → NEW STATE

# TO BE PREDICTABLE, REDUX HAS CERTAIN LIMITATIONS...

...most notably, that "changes are made with **pure functions**".

Where in Redux can we perform asynchronous side effects like API calls?

. . . DISPATCH(ACTION) →

MIDDLEWARE →

REDUCER(OLD STATE, ACTION) . . .

# REDUX-THUNK

- Popular, accessible, versatile.
- Dispatch actions that are functions, aka **thunks**, not just objects.
- Thunks return functions can dispatch regular actions.

USER INTERACTION → DISPATCH(THUNK)

THUNK() → ASYNC CALL → DISPATCH(ACTION)

REDUCER(OLD STATE, ACTION) → NEW STATE

# THUNK MIDDLEWARE INTERNALS (SLIGHTLY SIMPLIFIED)

```
function createThunkMiddleware() {
  return ({ dispatch, getState }) => next => action => {
    if (typeof action === 'function') {
      return action(dispatch, getState);
    }

    return next(action);
  };
}
```

# ADDING THUNK MIDDLEWARE TO YOUR STORE

```
import reducer from './reducers';
import { createStore, applyMiddleware } from 'redux';
import thunkMiddleware from 'redux-thunk';

const store = createStore(
  reducer,
  applyMiddleware(
    thunkMiddleware
  )
);
```

# REACT COMPONENT W/ THUNK

```
const CatButtons = ({ allCatNames }) => (
   <div>
   {
     allCatNames.map((catName, index) =>
       <button
         key={index}
         onClick={() => store.dispatch(fetchCat(catName))}>
          Load {catName}
        </button>
     )
   }
);
```

# ACTION CREATORS

```
const fetchCat = catName => {
   return dispatch => {
      fetch(`/api/cats/${catName}`)
      .then(res => res.json())
      .then(cat => {
         dispatch(loadCat(cat));
      })
      .catch(err => console.error(err.stack));
   }
};

const loadCat = cat => ({
  type: ‘LOAD_CAT’,
  cat
})
```

# MAIN TAKEAWAYS

- Out of the box, Redux doesn't have a place for asynchronous side effects.
- Middleware allows us to do some extra business in between an action being dispatched and it reaching the reducer.
- Redux Thunk middleware lets us dispatch functions ("thunks") instead of just actions.
- Thunks are invoked by the middleware, and passed dispatch as an argument - because of this, they can do some async work and then dispatch regular actions.

# OTHER MIDDLEWARE TO USE

- Redux Pack or Redux Promises (promises!)
- Redux Saga (generators!)
- Redux Observable (observables!)

# RESOURCES

- Slides: http://github.com/intersim/redux-async/
- Redux Middleware
- How can I represent side effects such as AJAX calls?

# THANKS!