React, Redux, Async, and Servers

Objectives

- Students will make a request from their app to their server
- Students will use thunk middleware for async actions in action creators

React "Ropes"

- When something goes wrong in your application you've got to "run the ropes" to figure exactly where the error occurs.

Consider everything that happens when you click a button:

- Verify things render correctly
- HTML renders the button
- The button is attached to a click (or submit) handler
- The handler may accept parameters (verify these!)
- The handler may process data (verify this!)
- The handler dispatches an action
- It runs through an action creator
- The action arrives at the reducer
- Verify it passes through the reducer without errors!

Use your knowledge of these lifecycles to debug your apps!

Attaching Middleware (reminder)

```
import { createStore, combineReducers, applyMiddleware } from 'redux'

const todoApp = combineReducers(reducers)
const store = createStore(
  todoApp,
  applyMiddleware(logger, crashReporter)
)
```

Example Logger (reminder)

```
const logger = store => reducer => action => {
  console.log('dispatching', action)
  let result = reducer(action)
  console.log('new state', store.getState())
  return result
}
```

Example Logger (reminder)

```
const logger = store => next => action => {
  console.log('dispatching', action)
  let result = next(action)
  console.log('next state', store.getState())
  return result
}
```

Example Crash Reporter (reminder)

```
const crashReporter = store => next => action => {
  try {
    return next(action)
  } catch (err) {
    console.error('Caught an exception!', err)
    throw err
  }
}
```

Example Timeout (reminder)

- Schedules actions with { meta: { delay: N } } to be delayed by N milliseconds.
- Makes `dispatch` return a function to cancel the timeout in this case.

```
const timeoutScheduler = store => next => action => {
  if (!action.meta || !action.meta.delay) {
    return next(action)
  }

  const timeoutId = setTimeout(
    () => next(action),
    action.meta.delay
  )

  return function cancel() {
    clearTimeout(timeoutId)
  }
}
```

Example Thunk (reminder)

- The `thunk` middleware intercepts functions that are dispatched as actions and executes them.
- It allows anything that's not a function to pass through as it regularly would.
- `thunk` is great for running code like `fetch` that downloads data from the internet asynchronously, then, when it receives the data, has access to dispatch to dispatch an action to update and display results.

```
const thunk = store => next => action =>
  typeof action === 'function'
    ? action(store.dispatch, store.getState)
    : next(action)
```

```
import {showResults} from './actions/search-actions';

dispatch((dispatch, store) => {
  fetch('http://www.reddit.com/r/movies.json')
  .then(res => res.json())
  .then(json => {
    dispatch(showResults(json));
  });
})
```

New Tools!! yarn and parcel (optional)

Web Development goes at a fast pace. People make new tools quickly. Here are two new tools some people are finding useful. Decide for yourself!

```
npm install -g yarn
npm install -g parcel
```

- yarn
  - replaces `npm`
  - "Fast, reliable, and secure dependency management."
  - why? *much* faster than `npm`
- parcel
  - replaces `webpack`
  - "blazing fast, zero configuration web application bundler"
  - why? *zero configuration*. truly. write your app, not webpack!

Parcel Dependencies

- Parcel also came out with a new feature where it automatically installs dependencies!