# 33 Redux Middleware

## Objectives

- Students will be able to create middleware for redux
- Students will be able to add third party middleware to redux

# Logging

Wouldn't it be nice if we logged every action that happens in the app, together with the state computed after it? When something goes wrong, we can look back at our log, and figure out which action corrupted the state.

# Attaching Middleware

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

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

# Example Logger

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

# Example Logger

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

# Example Crash Reporter

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

# Example Timeout Scheduler

- 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

- 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));
  });
})
```

# Reminder: Ternary Statements

The ternay expression syntax `expr ? truthyResult : falsyResult` is like an if-statement, but it **evaluates** the truthy or falsy expressions (it can store them in a variable immediately).

```
let canRideRide = height <= 60 ? "can't ride ride." : "can ride ride";
```

Compare this to if-else statements, which are purely structural and **do not evaluate** into a value.

```
let canRideRide = undefined;
if (height <= 60) {
  canRideRide = "can't ride ride.";
} else {
  canRideRide = "can ride ride";
}
```

This is illegal JavaScript:

```
let foo = if (something) { "value"} else { "other value"};
```

# Ternary vs if-else in React

- Ternary statements often appear inside React apps because templates depend on evaluation.
- Ternary statements evaluate into things
- if-else statements redirect the flow of your program, but evaluate into nothing themselves
- just like it doesn't make sense to say `x = for loop` it doesn't make sense to write `y = if statement`

React likes things to evaluate into things that are rendered on the page.

```
{this.props.isSending ?
  <button onClick={this.cancel}>Cancel</button> :
  <button disabled>Cancel</button>
}
```

React doesn't like if-statements because they don't inherently evaluate into something that can be returned on a page.

```
{if(this.props.isSending) {
  <button onClick={this.cancel}>Cancel</button>
} else {
  <button disabled>Cancel</button>
}}
```