Redux

You don't need Redux lots of the time.

You'd be surprised how far the React fundamentals will get you.

One data store where all your app's state lives.

Dispatch actions to change the state

Ask the store for its current state.

Actions

Plain JS objects that describe changes you want to make to the state:

```
{ type:
'LOG_IN_USER', name:
'jack' }
```

Reducer

A function that takes an action along with the current state, and returns the new state.

A Counter app

1. The store: { count = 0 }

2. Dispatch actions to increment: { type: 'INCREMENT' }

3. Reducer gets called with (state, action)

4. Reducer updates state

5. store.getState() returns { count = 1 }

The reducer

```
const defaultState = { count: 0 }

const reducer = (state = defaultState, action) => {
  switch (action.type) {
    case 'INCREMENT':
      return { count: state.count + 1 }
    default:
      return state
  }
```

The store

```
import { createStore } from 'redux'

const reducer = (state, action) => {...}

const store = createStore(reducer)

store.getState() // { count : 0 }
```

Remember: open the dev tools!

1. The store: { count = 0 }

2. Dispatch actions to increment: { type: 'INCREMENT' }

3. Reducer gets called with (state, action)

4. Reducer updates state

5. store.getState() returns { count = 1 }

```
npm run exercise redux 1
```

react-redux

<Provider>

Provides access between your React components and the Redux store

connect()

Connects an individual React component to the Redux store.

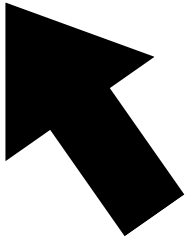
react-redux provider

```
const store = createStore(reducer)

import { Provider } from 'react-redux'

ReactDOM.render(
  <Provider store={store}>
    <App />
  </Provider>
)
```

by default no components can read from the store

you connect them

```
import { connect } from 'react-redux'

class SomeComponent extends Component {...}

const ConnectedComponent = connect(reduxStoreState => {
  return {
    count: reduxStoreState.count
  }
})(Component)
```



here we are explicitly saying which parts of our redux store this component is allowed to access

a connected component can also dispatch actions

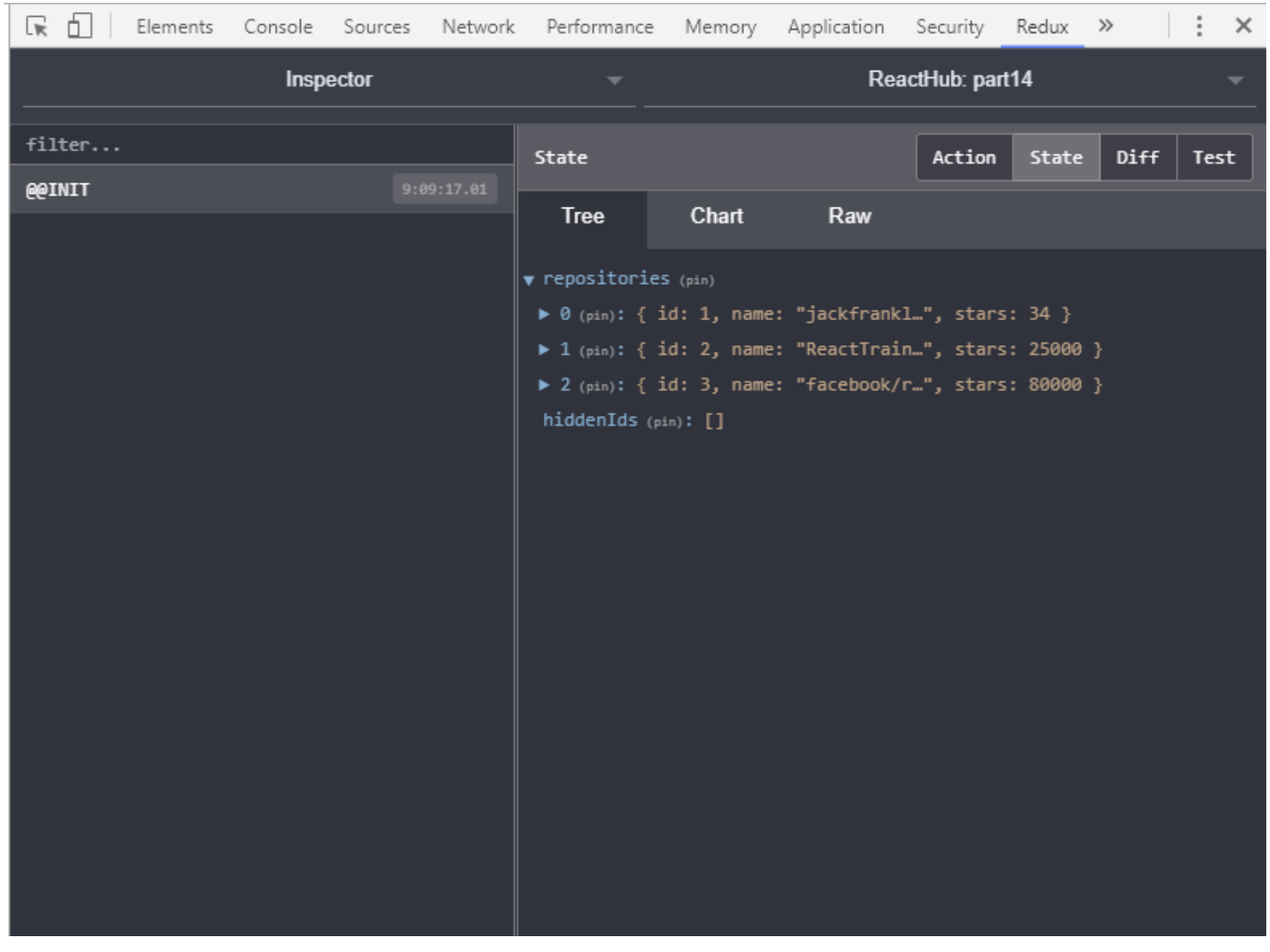
```
// inside a connected component

this.props.dispatch({ type: 'INCREMENT' })
```

Exercise 2!

```
const ConnectedCounter = connect(storeState => {

})(Counter)


export default ConnectedCounter
```

```
npm run exercise redux 2
```

Redux dev-tools



Exercise 3: you're on your own!

Task: allow the user to enter a number into an input field which will act as the number to increment or decrement by. So if the user enters 5 in the box, and presses the "+" button, the number should go up by 5.

Exercise 3: you're on your own!

Question: should the state of the input box go into Redux, or stay as state in the Counter component?

```
npm run exercise redux 3
```

Exercise 3: you're on your own!

Prefer local, component state when you can.

Task: allow the user to enter a number into an input field which will act as the number to increment or decrement by. So if the user enters 5 in the box, and presses the "+" button, the number should go up by 5.

```
npm run exercise redux 3
```

Redux best practices

```
// store.js

case 'INCREMENT': ...

// counter.js

this.props.dispatch({ type: 'INCEMENT' })
```

spot the bug?

```
import { INCREMENT } from './actions'

case INCREMENT:

this.props.dispatch({ type: INCREMENT })
```

action creators

```
const increment = () => {
  return { type: 'INCREMENT' }
}


import { increment } from './actions'

this.props.dispatch(increment())
```

Tidy up our Redux

1. Create a text box for letting the user set the number (just like in part 3), but this time create an action-creator for it and use constants like we have for INCREMENT.

```
npm run exercise redux 4
```

Multiple components reading the state

We want to add a header:

Counter App: Incrementing by: 5

Now two components need to know the increment by value

So it's time for Redux!

Moving incrementBy into Redux

1. Add new key (incrementBy) to state with default value of 1

2. Create action and action creator for setting incrementBy

3. Connect the <Header /> component so it can read the value

4. Connect the input in <Counter /> to the store so it can read the value

5. Make the value of incrementBy be updated when the user updates the input box.

```
npm run exercise redux 5
```

APIs and Redux

http://github-proxy-api.herokuapp.com/users/jackfranklin/repos

Redux Thunk

https://github.com/reduxjs/redux-thunk

thunk?!

```
// calculation of 1 + 2 is immediate
// x === 3
let x = 1 + 2;

// calculation of 1 + 2 is delayed
// foo can be called later to perform the calculation
// foo is a thunk!
let foo = () => 1 + 2;
```

redux-thunk allows action creators to dispatch other actions

redux-middleware

configuring thunk

```
import { createStore, applyMiddleware } from 'redux'
import thunk from 'redux-thunk'

return createStore(reducer, applyMiddleware(thunk)
```

```
// a regular action creator
export const setIncrementBy = (amount) => {
  return { type: 'SET_INCREMENT_BY', amount: amount }
}

// a thunk creator
export const fetchRepositories = () => {
  return dispatch => {
    // in here we do some async work
    // and then dispatch
    dispatch({ type: 'FETCHED_REPOSITORIES', data: ... })

  }
}
```

a thunk action creator is an action creator
that can dispatch other actions

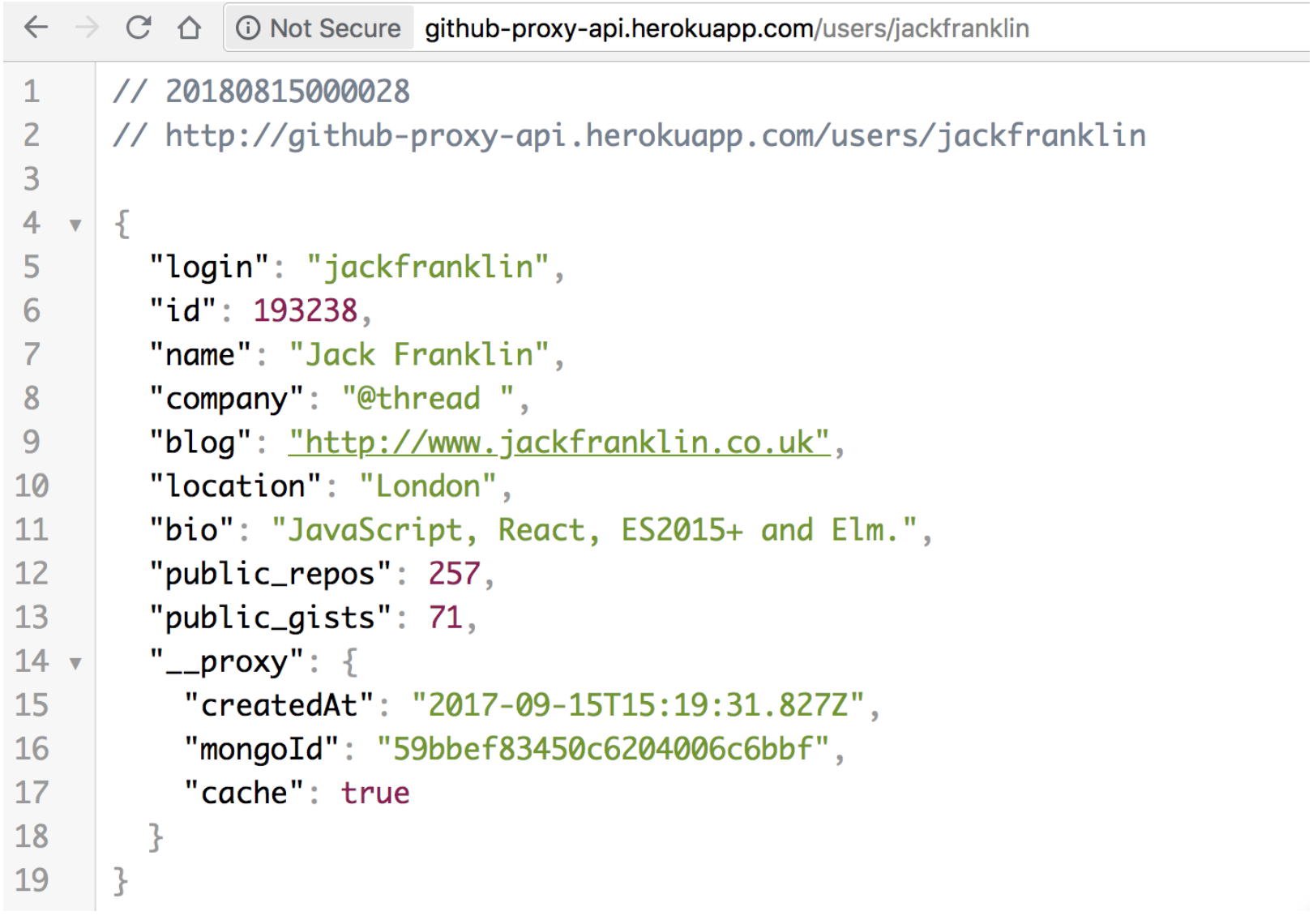
```
export const fetchRepositories = () => {
  return dispatch => {
    fetch('/users/jackfranklin/repos').then(result => {
      dispatch({
        type: 'FETCHED_REPOSITORIES',
        data: result.data
      })
    })
  }
}

// in a component

this.props.dispatch(fetchRepositories())
```

Fetching github repos with Redux

```
npm run exercise redux 6
```

More reducers

Let's fetch not only the repos, but the user information.


Not Secure
github-proxy-api.herokuapp.com/users/jackfranklin
// 20180815000028
// http://github-proxy-api.herokuapp.com/users/jackfranklin
{
"login": "jackfranklin",
"id": 193238,
"name": "Jack Franklin",
"company": "@thread ",
"blog": "http://www.jackfranklin.co.uk",
"location": "London",
"bio": "JavaScript, React, ES2015+ and Elm.",
"public_repos": 257,
"public_gists": 71,
"__proxy": {
"createdAt": "2017-09-15T15:19:31.827Z",
"mongoId": "59bbef83450c6204006c6bbf",
"cache": true
}
}

```
export const FETCHED_REPOSITORIES = 'FETCHED_REPOSITORIES'

export const fetchRepositories = () => {
  ...
}

export const FETCHED_USER = 'FETCHED_USER'

export const fetchUser = () => {
  return dispatch => {
    return fetch(
      'http://github-proxy-api.herokuapp.com/users/jackfranklin/repos'
    ).then(result => {
      dispatch({ type: FETCHED_USER, user: result.data })
    })
  }
}
```

two bits of data

= two keys in our store

= two reducers

we have one reducer per key in our store

and create a new reducer to deal with this bit

```
// STORE:

{ repositories: [], user: {} }
```

we'll make our current reducer only care about this part of the state

note the name change!

```
const repositoryReducer = (state = [], action) => {
  switch (action.type) {
    case FETCHED_REPOSITORIES:
      return action.data.items
    default:
      return state
  }
}
```

and this reducer now only cares about the repositories array *and knows nothing about the rest of the state.*

this reducer only cares about the user part of the state

```
const userReducer = (state = {}, action) => {
  switch (action.type) {
    ...
  }
}
```

combineReducers

takes reducers that are all responsible for a part of the state, and creates one reducer for the entire state

```
const reducer = combineReducers({
  repositories: repositoryReducer,
  user: userReducer,
})
```

combining reducers!

```
npm run exercise redux 7
```

now you're on your own.

build on exercise 7 and add an input that lets the user type in a username. When they hit a "go" button, fetch repositories and information for that user. Put the username into redux and pass it through when you dispatch the fetchRepositories() and fetchUser() function.

```
npm run exercise redux 8
```