redux-sagas

A short introduction

Sophie Koonin
@type__error

🐶 Dogs of the Week 🐶

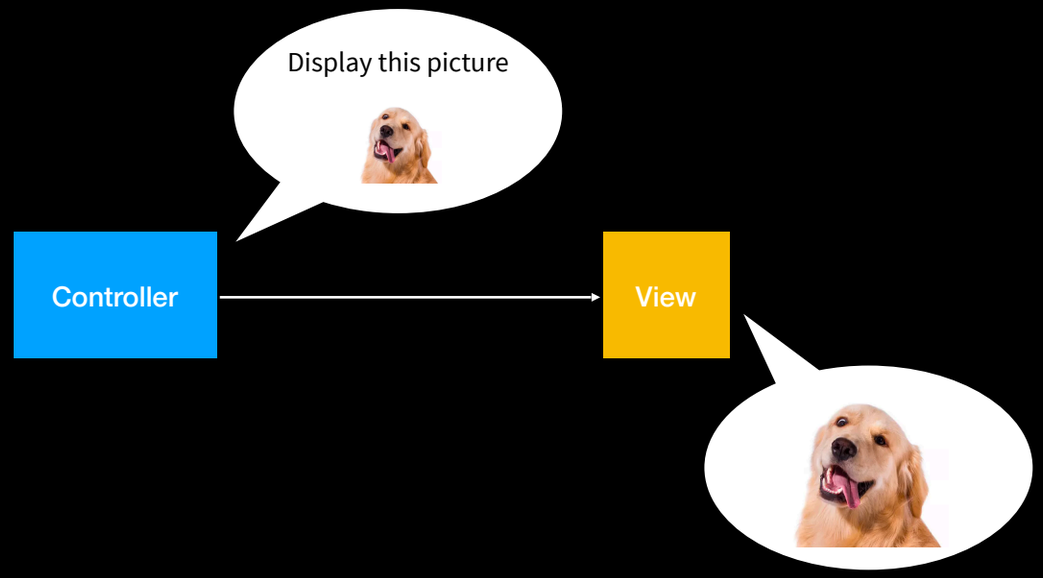
Name: Dennis
Age: 1 month
Breed: Golden Retriever
Favourite Toy: Squeaky pig

Name: Boris
Age: 3
Breed: Staffy
Favourite Toy: Ball

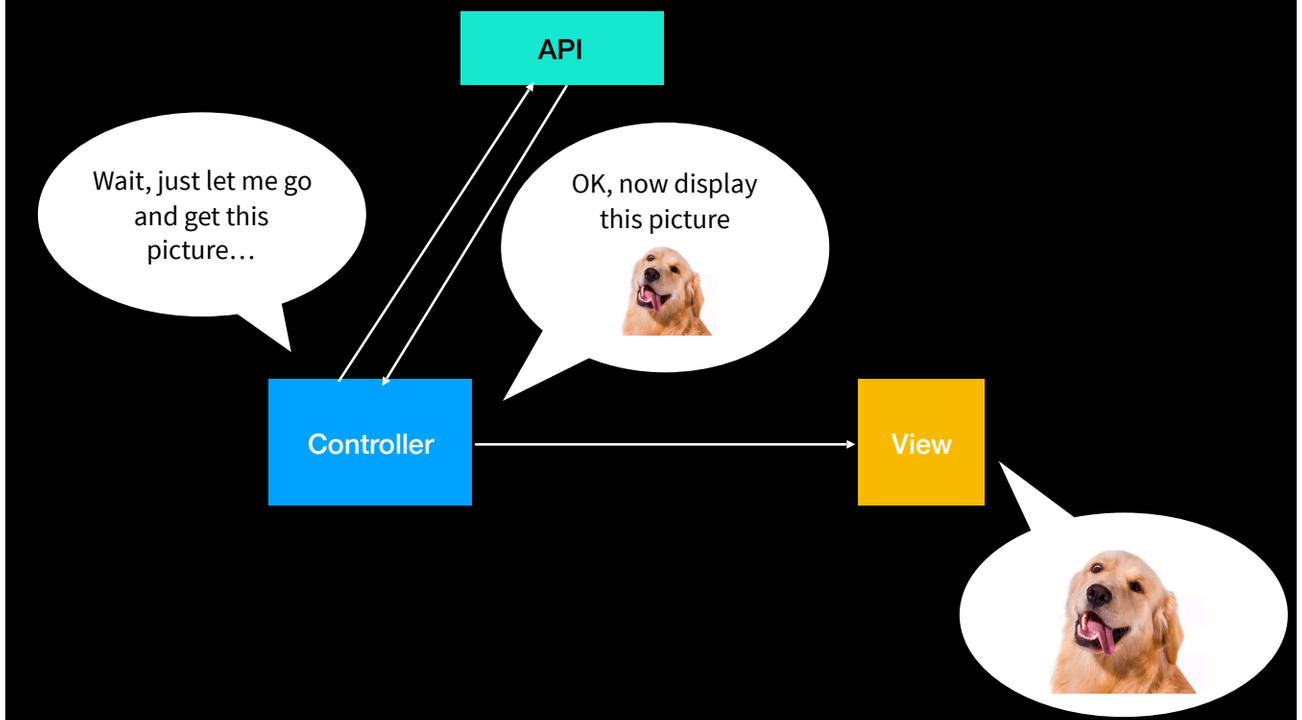
Name: Gandalf
Age: 1
Breed: Miniature Schnauzer
Favourite Toy: Stuffed bear

Name: Pepsi
Age: 6
Breed: Labrador
Favourite Toy: Rubber bone

Without side effects



With side effects



Fetching data from inside a component

```
export default class DogList extends React.Component {
    constructor(props) {
        super(props)
        this.state = {
            dogs: []
        }
    }

    componentDidMount() {
        api.fetchDogs()
        .then(response => this.setState({ dogs: response.body.dogs }))
    }

    ...
}
```

Fetching data with redux-thunk

Thunk: a function returned by another function in order to delay execution until it is needed

```
const fetchDogs = dispatch => api.fetchDogs()
          .then(({dogs}) => dispatch(fetchDogsSucceeded(dogs)))
```

Fetching data with redux-thunk

```
const fetchDogs = dispatch => {
    dispatch(fetchDogsRequested())
    api.fetchDogs()
    .then(({dogs}) => Promise.all(dogs.map(dog => fetchFavouriteToy(dog)(dispatch))))
    .then(dogs => dispatch(fetchDogsSucceeded(dogs)))
}
const fetchFavouriteToy = dog => dispatch => {
    dispatch(fetchToysRequested(dog))
    return api.fetchToy(dog).then(({toy}) => ({...dog, toy}))
}
```

Testing thunks can get complicated...

```
jest.mock('../fetch-toys');
jest.mock('../fetc
jest.mock('../fetc
    ok: true,
    body: {
        dogs: [ 'G
    }
}));

test('calls fetchD
    const dispatch
    fetchDogs(disp
    expect(dispatc
})
```


```
test('dispatches fetchFavouriteToysRequested with the fetched dogs if response is ok', () => {
    const dispatch = jest.fn()
    fetchDogs(dispatch)
    expect(dispatch).toHaveBeenCalledWith(fetchFavouriteToy)
})
```

Introducing redux-saga

“`redux-saga` is a library that aims to make application side effects ... easier to manage, more efficient to execute, simple to test, and better at handling failures.”

Fetching data with sagas

```
function* getDog = () => {
   try {
     const res = yield call(api.fetchDogs)
     const dogsWithToys = yield all(res.dogs.map(dog => fetchFavouriteToy(dog)))
     yield put(fetchDogsSuccessful(dogsWithToys))
   } catch (error) {
     yield put(fetchDogsFailed(error))
   }
}
```

```
function* fetchFavouriteToy({ dog }) {
  try {
    const response = yield call(api.fetchFavouriteToy, dog)
    return {...dog, favouriteToy: response.toy}
  } catch (error) {
    yield put(fetchFavouriteToyFailed(error))
    }
}
```

Triggering sagas

```
function* watchForFetchDogsRequested() {
    yield takeLatest('FETCH_DOGS_REQUESTED', fetchDogs)
}
```

How sagas work

ES6 Generators

```
function* getDog = () => {
   try {
      const res = yield call(api.fetchDogs)
      const dogsWithToys = yield all(res.dogs.map(dog => fetchFavouriteToy(dog)))
      yield put(fetchDogsSuccessful(dogsWithToys))
   } catch (error) {
      yield put(fetchDogsFailed(error))
   }
}
```

Function call... objects?

```
const res = yield call(api.fetchDogs)

// return value
{
  CALL: {
    context: null,
    fn: [Function: fetchDogs],
    args: []
  }
}
```

Effects

```
const { response, timeout } = yield race({
  response: call(api.patDog, dogId),
  timeout: delay(1000, ‘oh no’)
})

const [
    favouriteFood,
    favouriteToy
] = yield all([
    select(getFavouriteFood),
    select(getFavouriteToy)
])
```

Testing is easy with sagas!

```
// Sagas
import { call } from 'redux-saga/effects'
test('calls fetchDogs API', () => {
    const iterator = fetchDogs()
    expect(iterator.next().value).toEqual(call(api.fetchDogs))
})

// Thunks
jest.mock('../fetch-dogs');
jest.mock('../fetch-toys');
test('calls fetchDogs API', () => {
    const dispatch = jest.fn()
    fetchDogs(dispatch)
    expect(dispatch).toHaveBeenCalledWith(fetchDogs)
})
```

Testing branched logic

```
test('dispatches fetchDogsFailed if the call fails', () => {
   const iterator = fetchDogs()
    iterator.next()
    expect(iterator.throw('oh no').value)
        .toEqual(call(fetchDogsFailed('oh no')))
})
```

To wrap up...

- Easy to test
- Good for managing complicated data fetch flows
- Keep your actions as plain objects
- Keep business logic out of your display components

Thank you!