Agenda

1. ReactJS

2. Redux

3. Redux Saga - managing side effects

4. RESTful APIs - Lottoland API

5. DEMO

ReactJS

Reactive programming:

- is programming with asynchronous data streams.
- is to specify the dynamic behaviour of a value completely at the time of declaration
- separates the *how* from the *when* question

ReactJS

Props and State

- Props - *props* (short for *properties*) are a Component's **configuration,** its *options* if you may. They are received from above and are **immutable** as far as the Component receiving them is concerned.

Implementation:

```
import React, { Component } from 'react'

export default class Jackpots extends
    Component {
  render() {
    const jackpots = []

    this.props.data.forEach((jackpot) =>
    {
      jackpots.push(jackpot + '\n')
    })

    return (
      <div>
        {jackpots}
      </div>
    )
  }
}
```

Usage:

```
<li>Jackpots: <Jackpots data={this.props.jackpots} /></li>
```

ReactJS

State

- State - the *state* starts with a default value when a Component mounts and then **suffers from mutations in time (mostly generated from user events).** It's a serializable* representation of one point in time—a snapshot.

```
constructor (props) {
    super(props)

    this.state = {
      valid: true
    }
    this.validate = this.validate.bind(this)
  }
validate () {
    if (!this.props.validate) return
    this.setState({
      valid:
    this.props.validate(this.getValue())
    })
  }


  isValid () {
    return this.state.valid
render () {
    const common = {
      id: this.props.id,
      ref: 'input',
      onChange: this.validate
    }


    const classes = []


    if (this.state.valid === true) {
      classes.push('valid')
    } else if (this.state.valid === false) {
      classes.push('invalid')
    }
  }
```

ReactJS

Components and Containers

- Stateless Component (dumb) - Only *props*, no *state.* There's not much
  going on besides the
  `render()` function and all their
  logic revolves around
  the *props* they receive. This
  makes them very easy to
  follow (and test
  for that matter).

```jsx
import React, { Component } from 'react'
import { Route, Link } from 'react-router-dom'

import PrivateRoute from '../../containers/
    PrivateRoute'
import LoginDialog from '../../containers/
    LoginDialog'
import HomePage from '../pages/HomePage'
import LotteryPage from '../pages/LotteryPage'

class Header extends Component {

  render() {
    return (
      <div>
        <nav >
          <Link to="/">Home</Link>
          {" | "}
          <Link to="/lotteries">Lotteries</Link>
        </nav>

        <Route exact path='/' component={HomePage}/>
        <Route path='/login' component={LoginDialog}/
>
        <PrivateRoute path='/lotteries'
component={LotteryPage}/>
      </div>
    )
  }
}

export default Header
```

ReactJS

Components and Containers

- Stateful Component (container) - Both *props* and *state.* We also call these
  state managers. They are in charge of client-server communication
  (XHR, web sockets, etc.), processing data and responding to user events.

```
class LotteryPage extends Component {

  constructor() {
    super()
    this.logout = this.logout.bind(this)
  }
  componentDidMount() {
    this.props.dispatch(getData())
  }
  logout() {
    this.props.dispatch(logOut())
  }
  render() {
    const { lotteries } = this.props
    return (
      <Wrapper>
        <Lotteries lotteriesData={lotteries} />
        <div>
          <Button primary onClick={this.logout}
>Logout</Button>
        </div>
      </Wrapper>
    )
  }
}
LotteryPage.propTypes = {
  dispatch: propTypes.func.isRequired
}
const mapStateToProps = (state) => ({
  lotteries: getLotteriesData(state)
});

export default connect(mapStateToProps)(LotteryPage)
```

ReactJS

React Router v4

- Used to navigate through different pages of an app

- A pure React rewrite of the popular package. The required route configuration from previous versions has been removed and everything is now "just components".

- Installation - yarn add --dev react-router-dom.

- Rendering a <Router>

```
<Route exact path='/' component={HomePage}/>
<Route path='/login' component={LoginDialog}/>
<PrivateRoute path='/lotteries' component={LotteryPage}/>
```

```
import React from 'react'
import ReactDOM from 'react-dom'
import { BrowserRouter } from 'react-router-dom'

import App from './containers/App'
import { Provider } from 'react-redux'
import configureStore from './store/configureStore'

const store = configureStore()


ReactDOM.render((
    <Provider store={store}>
      <BrowserRouter>
        <App />
      </BrowserRouter>
    </Provider>
), document.getElementById('root'))
```

ReactJS

styled-components

- CSS styles embedded into JS

- No need of preprocessors

- High decoupling and able to re-use them in 'components like' style

- Check if interesting: https://styled-components.com/ / https://www.youtube.com/watch?v=jjN2yURa_uM

ReactJS

styled-components example

Usage: `<Button primary>View Lotteries</Button>`

Implementation:

```
import styled from 'styled-components'

const Button = styled.button`
  /* Adapt the colors based on primary prop */
  background: ${props => props.primary ? 'palevioletred' : 'white'};
  color: ${props => props.primary ? 'white' : 'palevioletred'};

  font-size: 1em;
  margin: 1em;
  padding: 0.25em 1em;
  border: 2px solid palevioletred;
  border-radius: 3px;

  &:hover {
    color: ${props => props.primary ? 'palevioletred' : 'white'};
    background: ${props => props.primary ? 'white' : 'palevioletred'};
  }
`

export default Button
```

ReactJS

yarn *vs* npm

Potential Problems with NPM

- *nested dependencies (fixed in npm 3)*
- *serial installation of packages*
- *single package registry (npmjs.com ever go down for you?)*
- *requires network to install packages (though we can create a makeshift cache)*
- *allows packages to run code upon installation (not good for security)*
- *indeterminate package state (you can't be sure all copies of the project will be using the same package versions)*

ReactJS

yarn *vs* npm

Yarn Solutions

• multiple registries - Yarn reads and installs packages from both `npmjs.com` as well as Bower. In the event one goes down, your project can continue to be built in CI without issue

• flat dependency structure - simpler dependency resolution means Yarn finishes faster and can be told to use a single version of certain packages, which uses less disk space

• automatic retries - a single network request failing won't cause an install to fail. Requests are retried upon failure, reducing red builds due to temporary network issues

• parallel downloads - Yarn can download packages in parallel, reducing the time builds take to run

• fully compatible with npm - switching from npm to Yarn is a no friction process

• Yarn.Lock - keeps dependencies locked to specific versions similar to Gemfile.lock in the Ruby world

ReactJS

React Dev Tools

- Chrome plugin useful for developing and debugging of React Apps

- Installation -
https://chrome.google.com/webstore/detail/react-developer-tools/
fmkadmapgofadopljbjfkapdkoienihi?hl=en
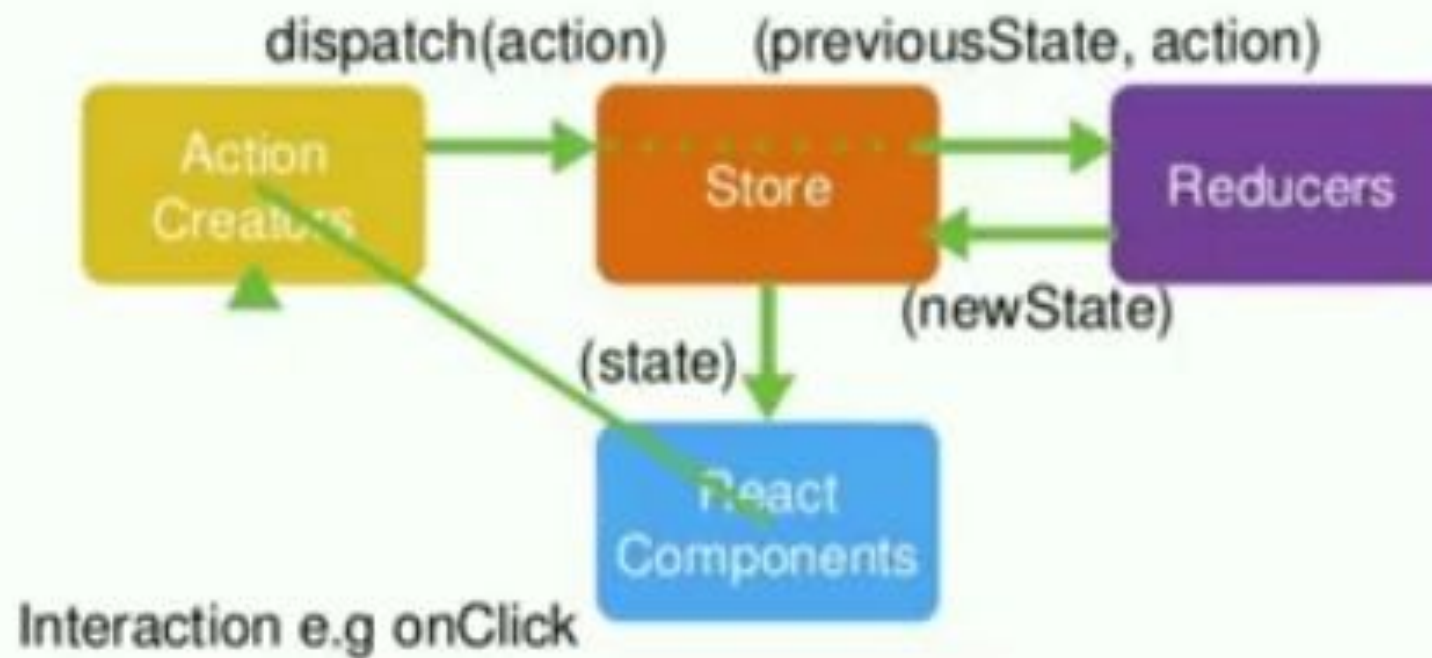
Redux

What is Redux?

- Provides predictable state management using actions and reducers.

- Can use *middleware* to manage **async**/side effects, such as Redux-Thunk or Redux-Saga

- *Single source of truth - the state of your whole application is stored in an object tree within a single store*

- react-redux is the package that allows you to use Redux in a React app

Redux

Redux Flow

Redux

Actions

- Describe something has (or should) happen, but they don't specify how it should be done
- Payloads of information that send data from your application to your store

- The only source of information for the store

- Plain JavaScript objects that must have *type* property that indicates the type of the action being performed

Example:

```
export const AUTH_REQUEST = 'AUTH_REQUEST'
{
    type: types.AUTH_REQUEST,
    payload: { username, password }
}
```

Redux

Action Creators

- Functions that create actions

- Redux action creators simply return an action

- In Redux usually are used with dispatch(), like this:

Example:

```
logout() {
    this.props.dispatch(logOut())
  }
```

```
export function logIn (username, password) {
  return {
    type: types.AUTH_REQUEST,
    payload: { username, password }
  }
}
```
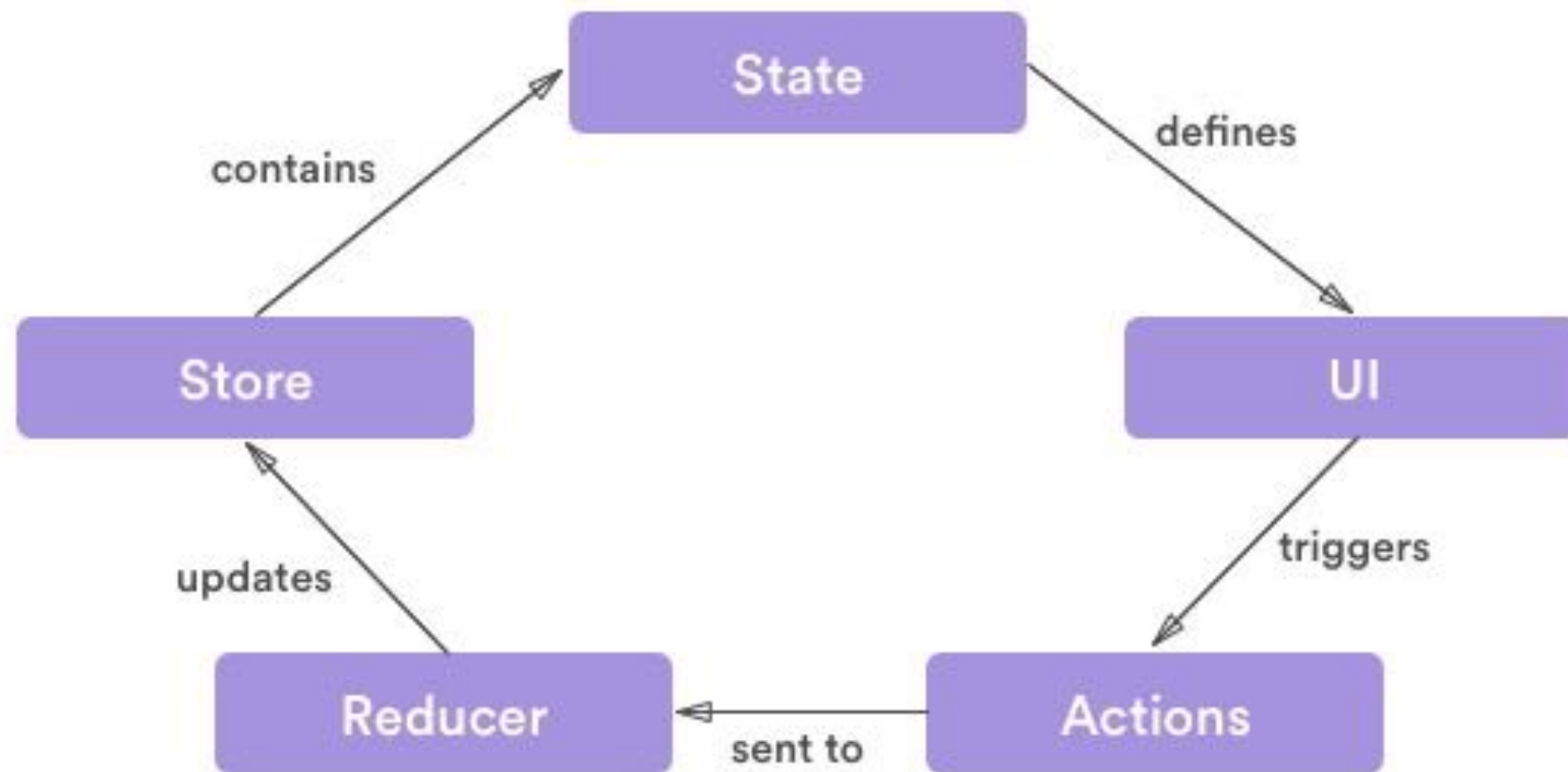
Redux

Reducers

• Pure functions that take the previous state and an action, an return the next state

• Actions describe the fact that *something happened*, but don't specify how the application's state changes in response, this is the job of reducers

• Reducers handle state transitions, but they must be done synchronously

*Things you should **never** do inside a reducer:*
- *Mutate its arguments;*
- *Perform side effects like API calls and routing transitions;*
- *Call non-pure functions, e.g. Date.now() or Math.random()*

Redux

Reducers

Redux

Reducers Examples

authReducer

```javascript
import * as types from '../constants/actionTypes'
import { Map, fromJS } from 'immutable'


export default function (state = Map(), action) {
  if (action.type === types.AUTH_RESPONSE) {
    return Map(state).mergeDeep(fromJS(action))
  }
  if (action.type === types.AUTH_LOGOUT) {
    return Map()
  }
  if (action.type === types.AUTH_ERROR) {
    return Map(state).set('error', action.payload)
  }
  return state
}
```

lotteriesDataReducer

```javascript
import * as types from '../constants/actionTypes'
import { Map, fromJS } from 'immutable'

export default function (state = Map(), action) {
  if (action.type === types.DATA_RESPONSE) {
    return Map(state).mergeDeep(fromJS(action))
  }
  if (action.type === types.DATA_ERROR) {
    return Map(state).set('error', action.payload)
  }
  return state
}
```

*Using _immutable js_ to prevent mutating the state by accident.
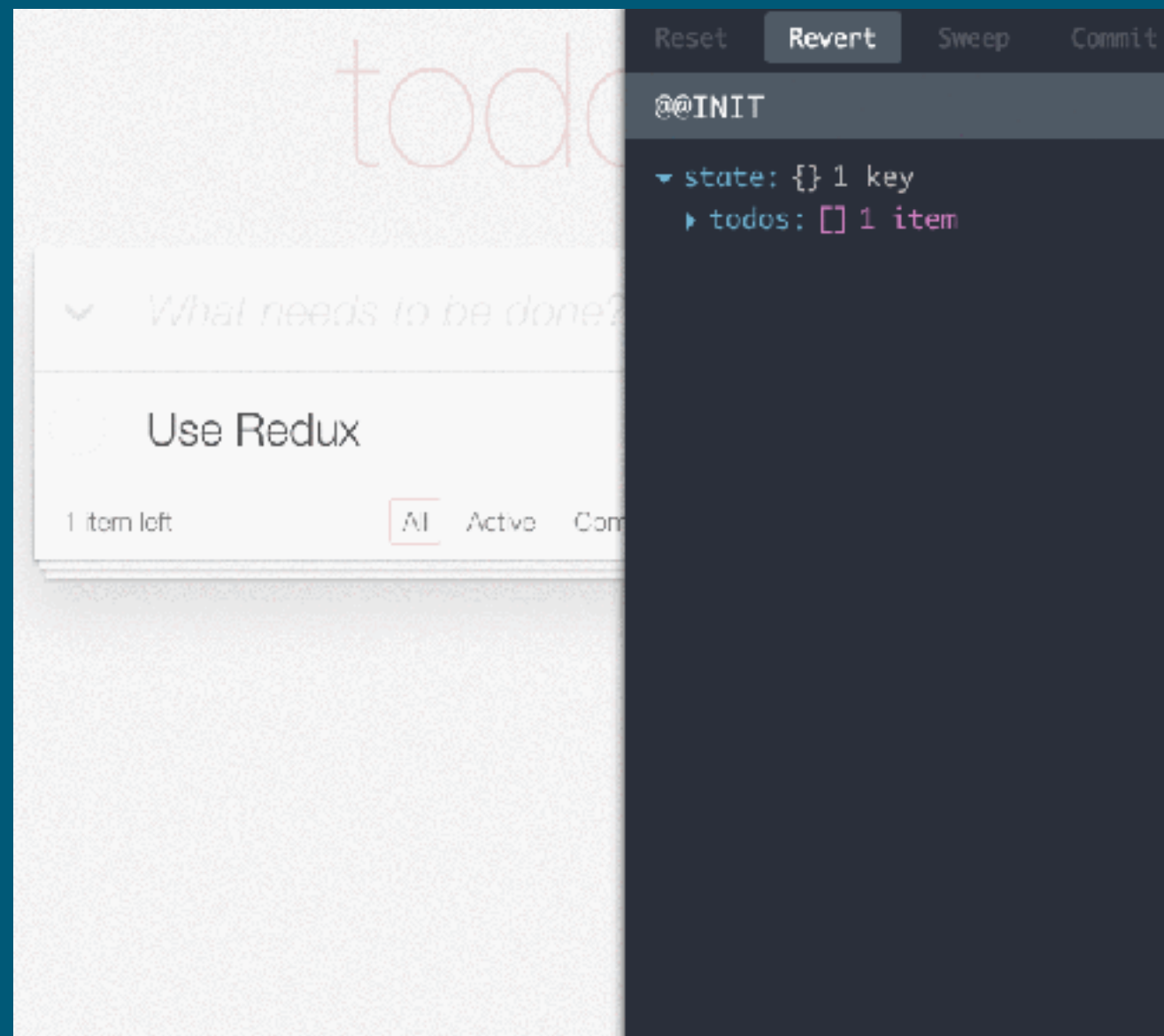
Redux
Store, Connected components

- Store is an object that brings actions and reducers together

- Store holds application state

- Store allows access to state via *getState()*

- Store allows state to be updated via dispatch(action)

- Store registers listeners via subscribe(listener)

- Store handles unregistering of listeners via the function returned by subscribe(listener)

- Use connect() to connect the *React part* of your app with the store, such components are also known as *connected components*

- It's important to note that you'll only have a single store in a Redux application

Redux

Redux Dev Tools

https://github.com/gaearon/redux-devtools

Redux-Saga

What is Redux-Saga?

- A library that aims to make side effects (i.e. asynchronous things like data fetching and impure things like accessing the browser cache) in React/Redux applications easier and better

- Uses ES6 Generators for making asynchronous flows easy to read, write and test

- Advantage against another famous *middleware* called 'redux-thunk' is that you don't end up in callback hell
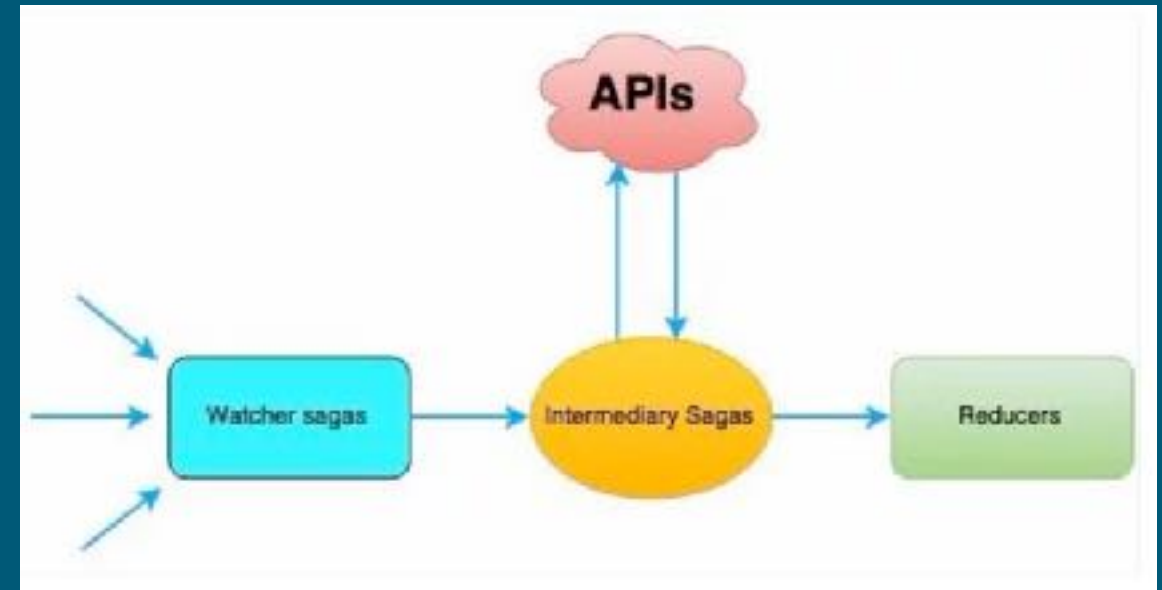
React - Redux - Saga Cycle
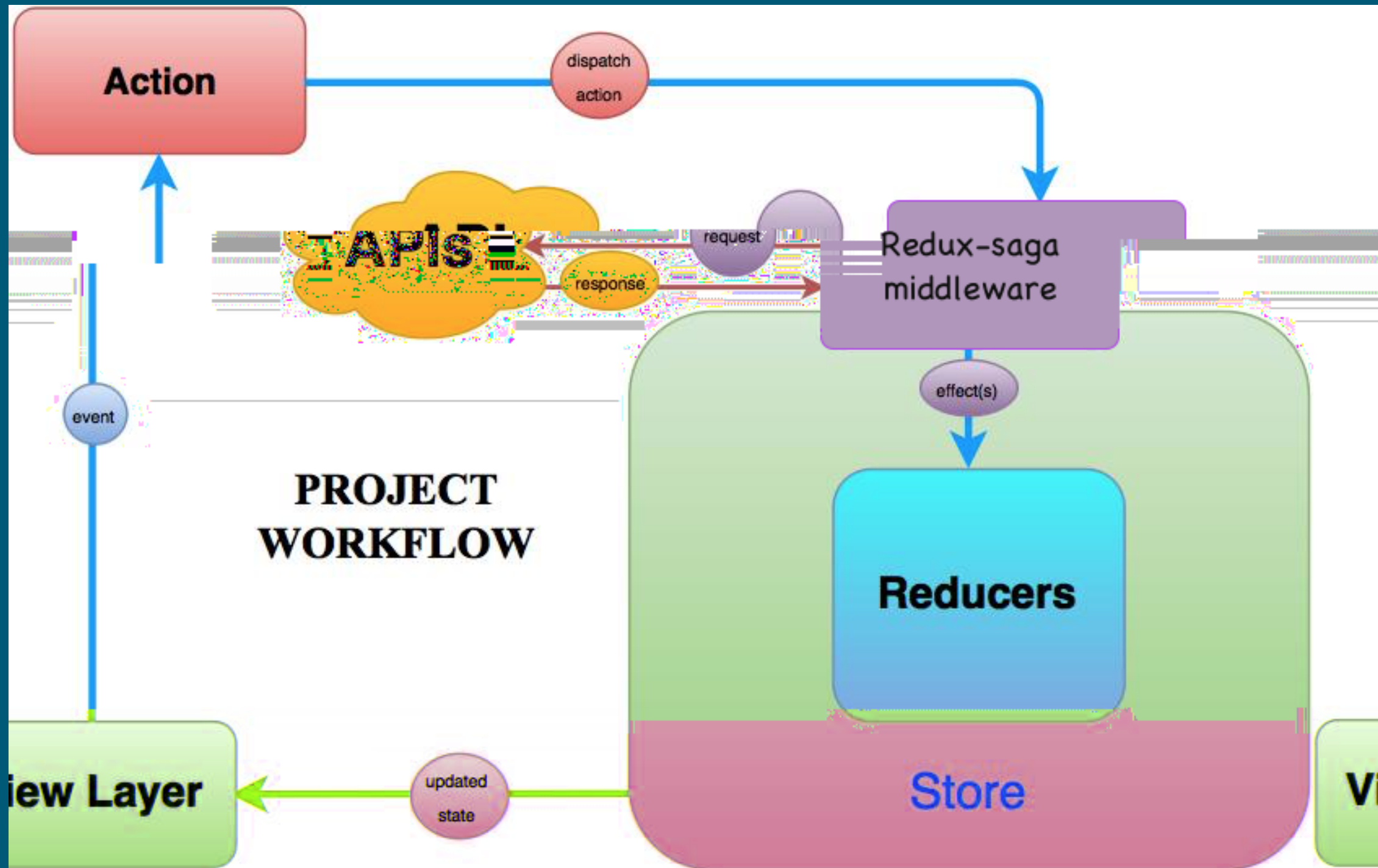
Redux-Saga

Examples

Sagas starter:

```
import { fork } from 'redux-saga/effects';
import watchAuthentication, { watchLotteriesData } from './watcher';

export default function* startForman() {
  yield fork(watchAuthentication);
  yield fork(watchLotteriesData);
}
```


authSaga

```
import { put, call } from 'redux-saga/effects'
import { login } from '../API/api'
import * as types from '../constants/actionTypes'

export default function* authSaga({ payload }) {
  try {
    const authInfo = yield call(login, payload)
    yield put({ type: types.AUTH_RESPONSE, authInfo })
  } catch (error) {
    yield put({ type: types.AUTH_ERROR, error })
  }
}
```

lotteriesSaga

```
import { put, call } from 'redux-saga/effects'
import { getData } from '../API/api'
import * as types from '../constants/actionTypes'

export default function* lotteriesSaga() {
  try {
    const data = yield call(getData)
    yield put({ type: types.DATA_RESPONSE, data })
  } catch (error) {
    yield put({ type: types.DATA_ERROR, error })
  }
}
```

RESTful APIs - Lottoland API

- *RESTful API - **Representational state transfer (REST)** or **RESTful** Web services are one way of providing interoperability between computer systems on the Internet. REST-compliant Web services allow requesting systems to access and manipulate textual representations of Web resources using a uniform and predefined set of stateless operations*

- The most common use of them is with JSON - they accepts http requests (GET, POST, PUT, etc) to a certain URL and provide responses like JSON objects.

- Lottoland APIs used in the demo app:
  - {{Server}}/api/client/v1/players/login - for logging in the user
  - {{Server}}/api/client/v1/drawings - for getting the info about the lotteries

RESTful APIs - Lottoland API

JSON Responses

Login

```
{
  "access_token":
"eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJqdGki0iI2YWYwMjE5NS0
wNGE2LTQ5ZTkt0WVjYS1hNGExYjY5MWEyNDIiLCJleHAiOjE0OTYzMDkyNjIs
ImlhdCI6MTQ5NTA50TY2MiwiYXVkIjoiZXh0ZXJuIiwicGFrIjoiOTN6eU5Me
EZSaHc9IiwiZm5hIjoiTWloYWlsIiwibG5hIjoiR2FiZXJvdiIsImN1ciI6Ik
VVUiIsInN1YiI6IjU5MDBhZTFhYWM3MzcxMTE0YWY3YmI4ZCJ9.U82W1rRTuw
vd4BNWy-taPYUbbIUd-C4JfNxsqKJws24",
  "token_type": "bearer",
  "expires_in": "2592000",
  "refresh_token":
"eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJqdGki0iJmNDcyMTM0Ny0
wYzA4LTQ5YmUt0WMxZi1iMzg1MjliYWJmY2MiLCJleHAiOjE0OTc20TE2NjIs
ImlhdCI6MTQ5NTA50TY2MiwiYXVkIjoicmVmcmVzaCIsInBhayI6IjkzenlOT
HhGUmh3PSIsInN1YiI6IjU5MDBhZTFhYWM3MzcxMTE0YWY3YmI4ZCJ9.qKE7E
t5wJkGxRAeLRM9EHP6RZ8fLYnPZhK6CYlBtAwg"
}
```

Drawings

```
[
  {
    "id": "austriaLotto_2629",
    "lotteryId": "austriaLotto",
    "drawingDate": "2017-05-21T16:30:00.000+0000",
    "closingDate": "2017-05-21T16:00:00.000+0000",
    "state": "IN_PLAY",
    "doubleJackpotAllowed": true,
    "jackpots": [
      {
        "lotteryId": "austriaLotto",
        "jackpot": 0,
        "marketingJackpot": 0
      }
    ],
    "drawingType": "SU"
  },
  {
    "id": "cash4Life_307",
  // ...more
]
```

DEMO

References

- *https://hackernoon.com/rapid-tips-for-your-react-redux-application-68f513a7cebf*

- *https://github.com/uberVU/react-guide/blob/master/props-vs-state.md*

- *https://scotch.io/tutorials/build-a-media-library-with-react-redux-and-redux-saga-part-2*

- *https://gist.github.com/staltz/868e7e9bc2a7b8c1f754*

- *http://stackoverflow.com/questions/42123261/programmatically-navigate-using-react-router-v4*

- *https://medium.com/@pshrmn/a-simple-react-router-v4-tutorial-7f23ff27adf*

- *https://github.com/reactjs/redux/blob/master/docs/introduction/ThreePrinciples.md*

- *https://redux-saga.js.org/*

- *React: Up and Running: Building Web Applications, book by Stoyan Stefanov*