ARTSY

Javascriptures:

State Management: Local State / Redux

React has championed the concept of 1-way data flow

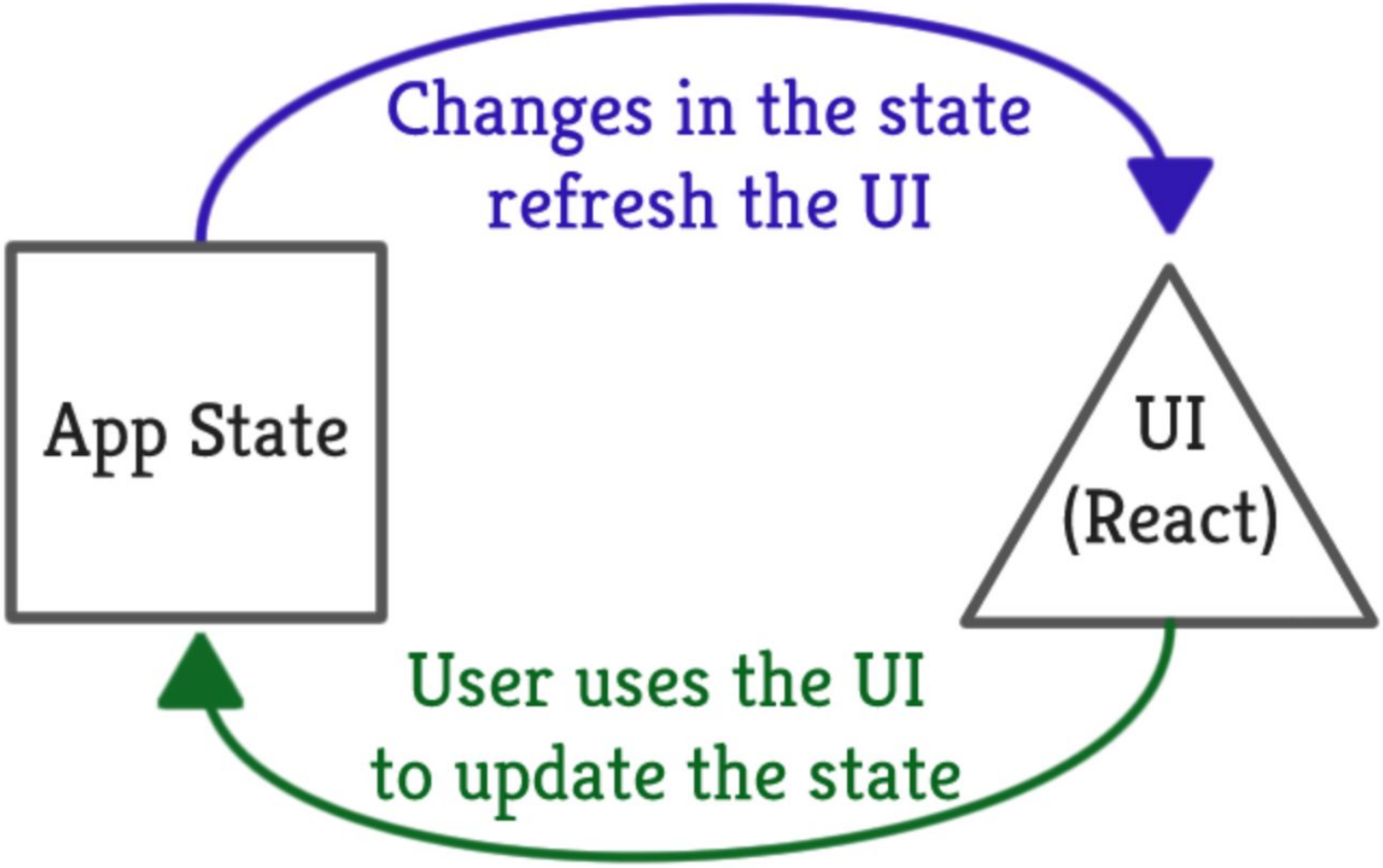
Data is passed into a component as props, and components then pass that data to other components as props

When data is passed in it is immutable and cannot change

```
function App () {
  return (
    <Kitty name='leif' />
  )
}

function Kitty (props) {
  props.name = 'Some other lil cat!' // Error! attempting to mutate props

  return (
    <div>
      {props.name}
    </div>
  )
}

ReactDOM.render(<App/>, mount)
```

If you can't change the data that’s passed into a component, how does one manage state?

Via `setState()`


Changes in the state
refresh the UI
App State
UI
(React)
User uses the UI
to update the state

setState can only be used in React class components, via `React.Component`

```
class User extends React.Component {
  state = {
    isAdmin: false,
  }

  render() {
    return <div>Is user an admin? {this.state.isAdmin}</div>
  }
}
```

```
class User extends React.Component {
  constructor(props) {
    super(props)

    this.state = {
      isAdmin: props.isAdmin
    }
  }

  render() {
    return <div>Is user an admin? {this.state.isAdmin}</div>
  }
}
```

Can modify state through calls to `setState`, for example on UI interactions

```
class User extends React.Component {
  state = {
    isAdmin: false,
  }

  toggleAdminStatus = () => {
    this.setState({
      isAdmin: !this.state.isAdmin,
    })
  }

  render() {
    return (
      <div>
        <div>Is user an admin? {this.state.isAdmin}</div>

        <button onClick={this.toggleAdminStatus}>Toggle Admin Status</button>
      </div>
    )
  }
}
```

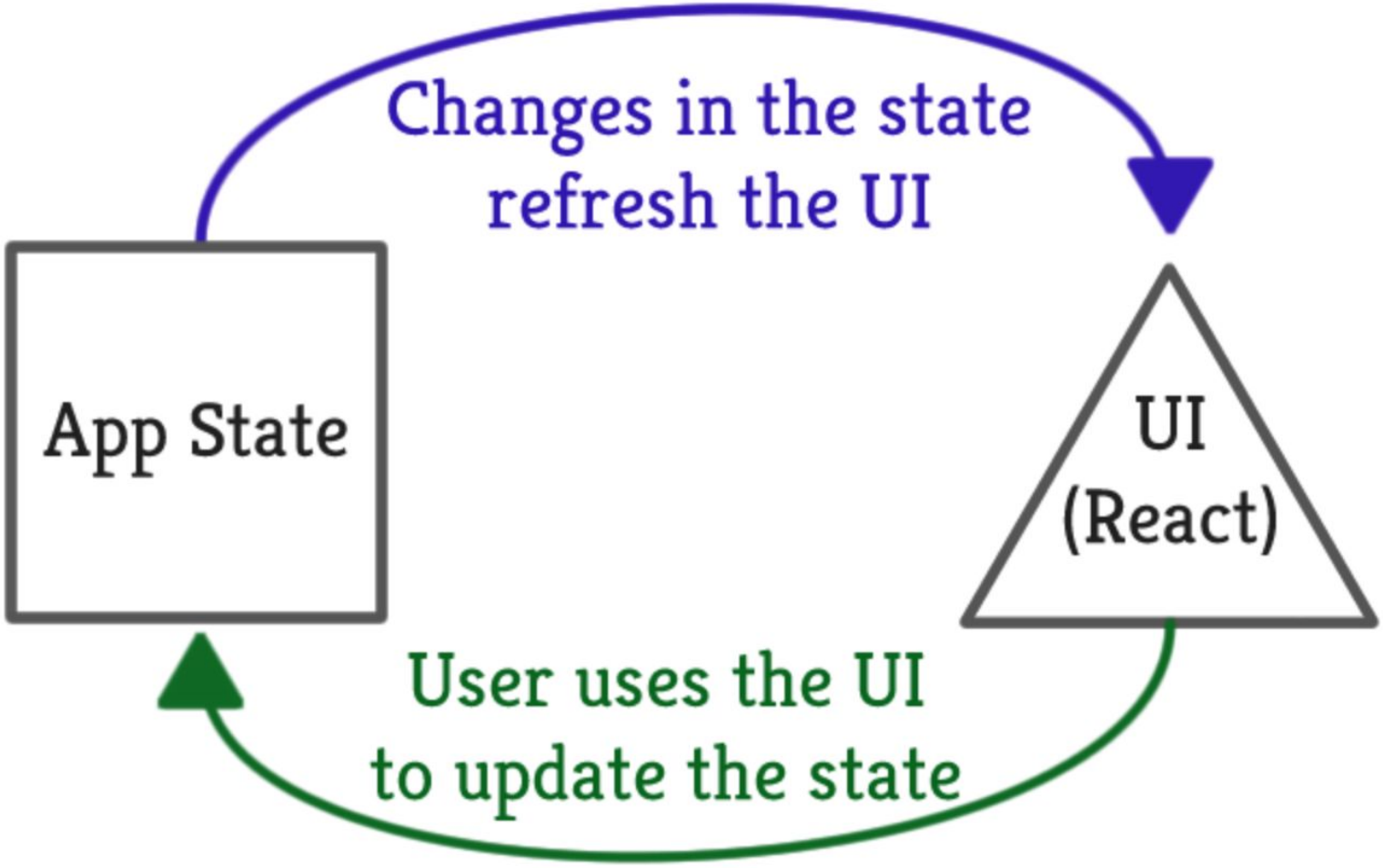
Simple UI interaction can be easily managed within a single class -- think, button toggles, show / hide, basic http list views

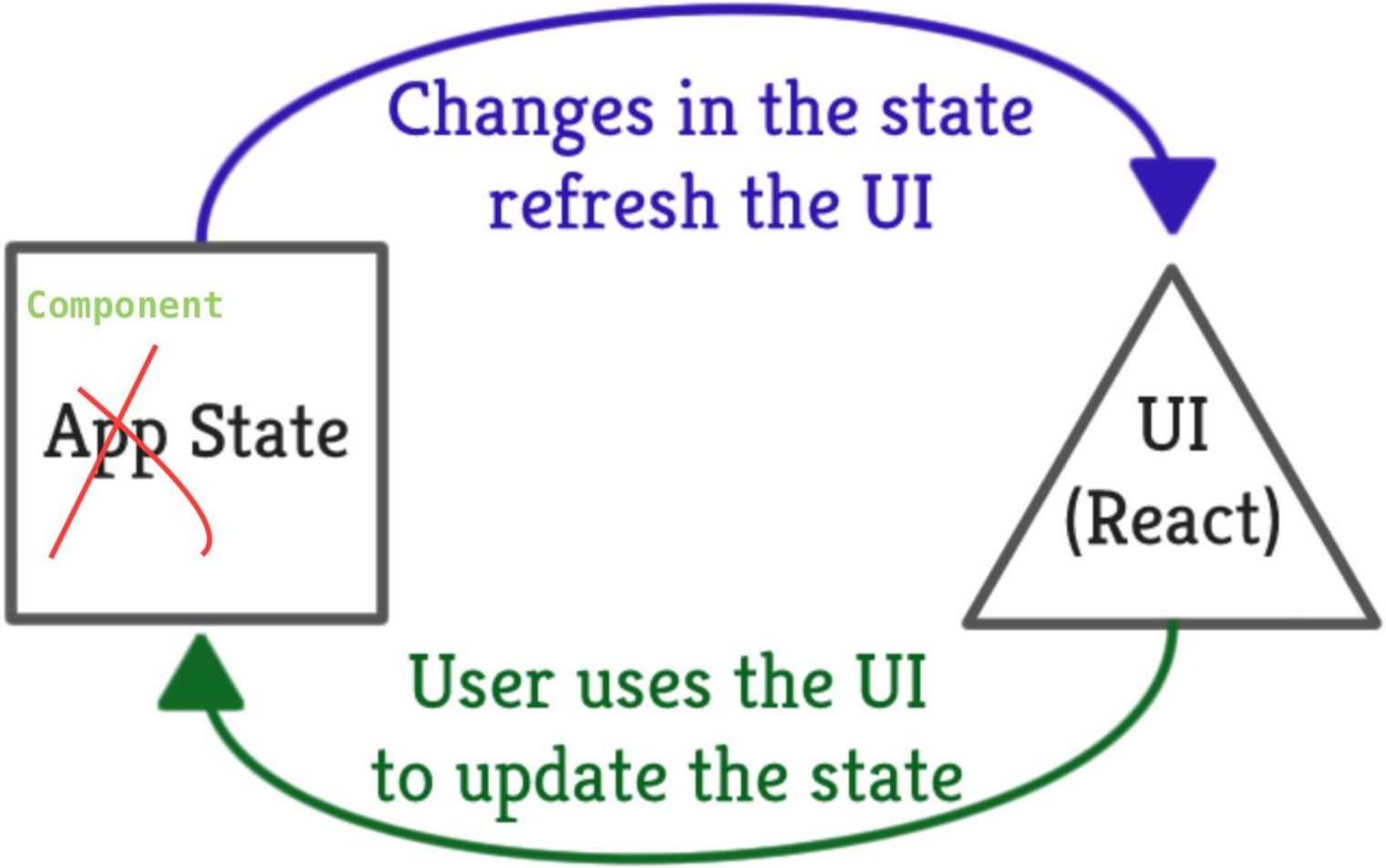
```
class UserList extends React.Component {
  state = {
    users: [],
  }

  async fetchUsers() {
    const users = await http.get("/users")

    this.setState({
      users,
    })
  }
```

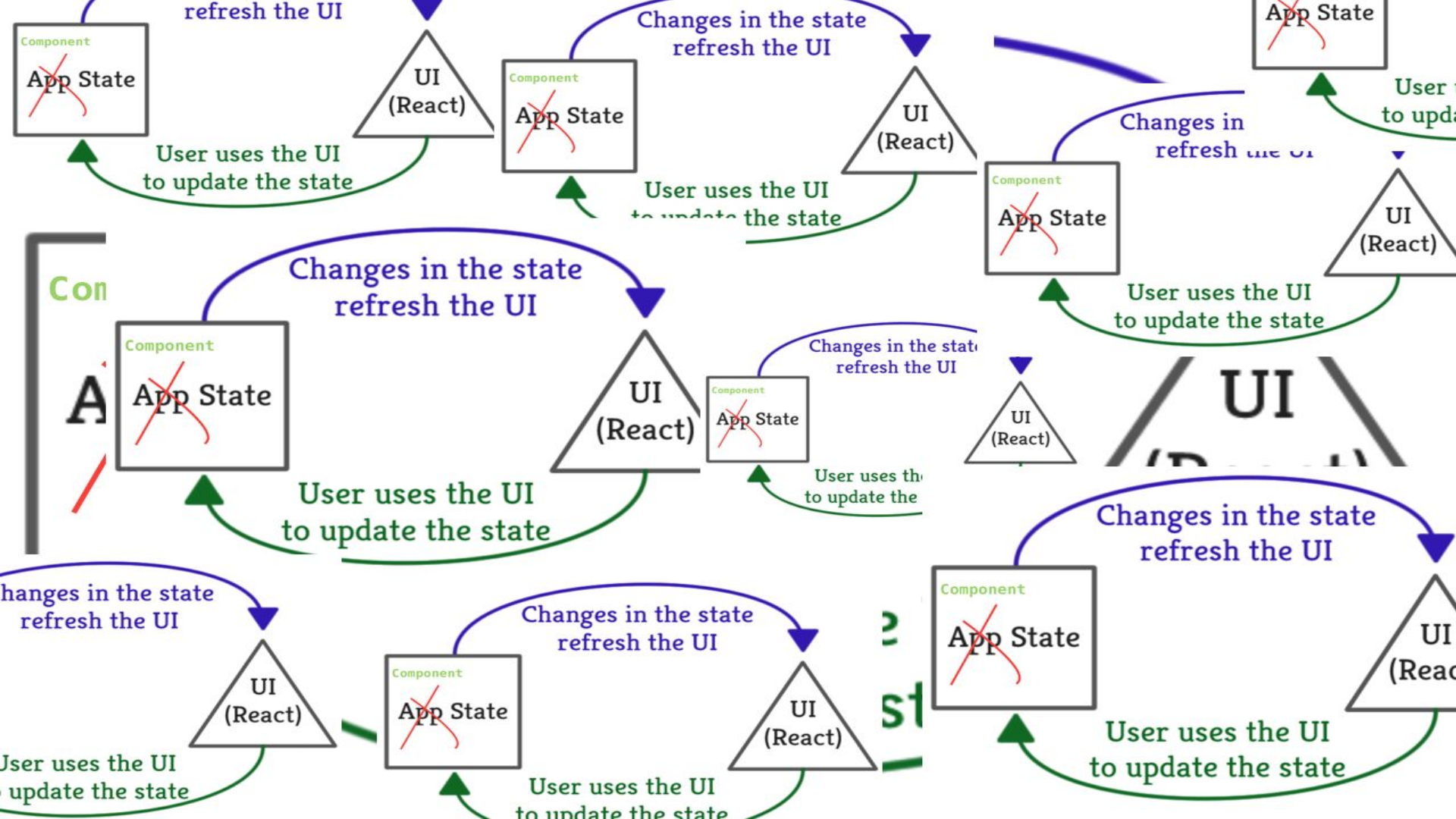
```
  render() {
    const hasUsers = Boolean(this.state.users.length)

    return (
      <div>
        <h1>User List</h1>

        {hasUsers && (
          <ul>
            {this.state.users.map(user => {
              return (
                <div>
                  <h3>{user.name}</h3>
                  <div>{user.bio}</div>
                </div>
              )
            })}
          </ul>
        )}

        <button onClick={this.fetchUsers}>Fetch Users!</button>
      </div>
    )
  }
}
```

This works well most of the time!

`setState` can take you far, but problems begin to occur when complex UI interactions need to be managed across multiple components


Changes in the state
refresh the UI
App State
UI
(React)
User uses the UI
to update the state


Changes in the state
refresh the UI
Component
App State
UI
(React)
User uses the UI
to update the state


Component
App State
UI
(React)
Changes in the state
refresh the UI
User uses the UI
to update the state

When component setState starts to manage *App* state and is scattered across a codebase it begins to feel suspiciously like MVC and becomes increasingly hard to scale

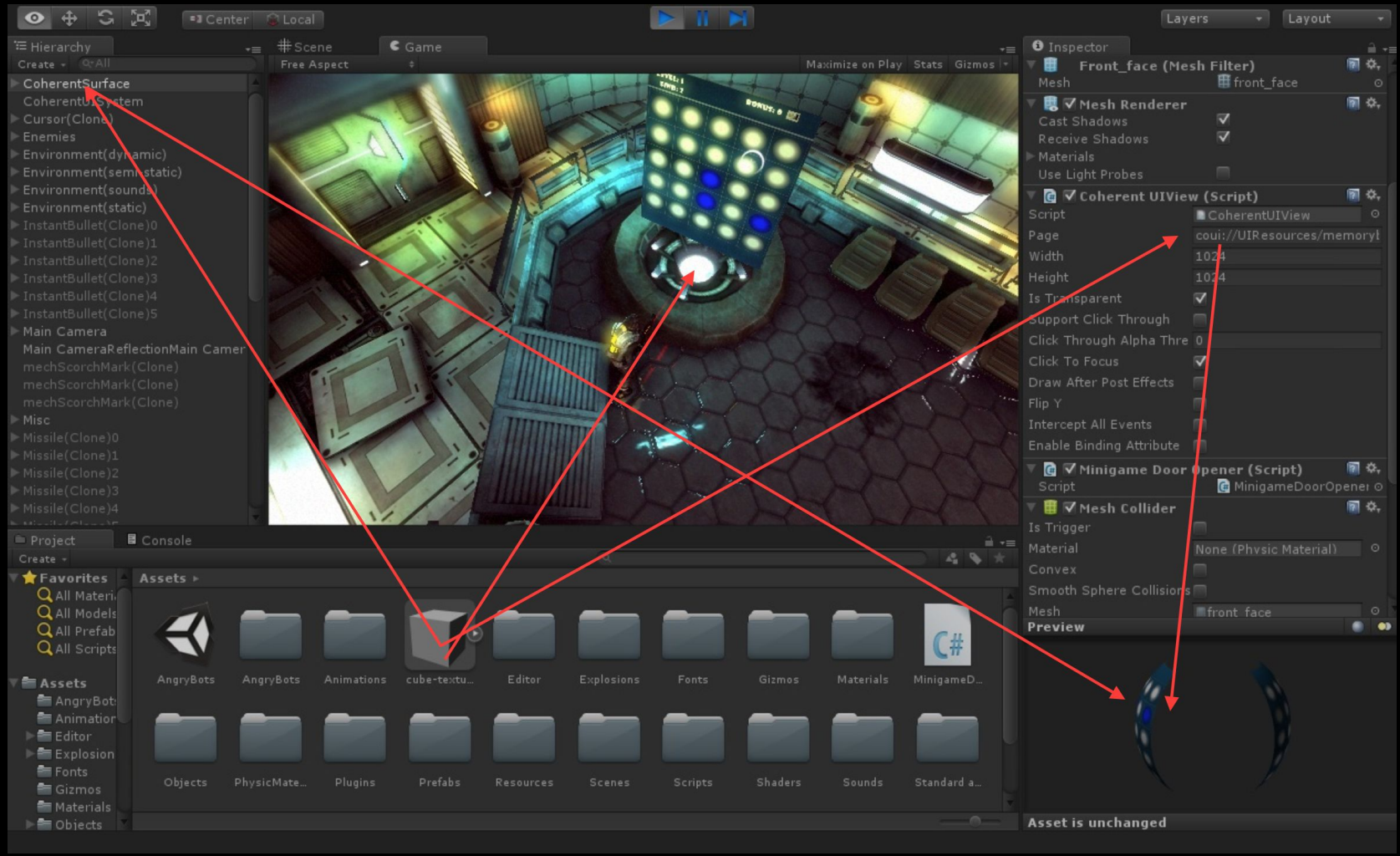
Redux

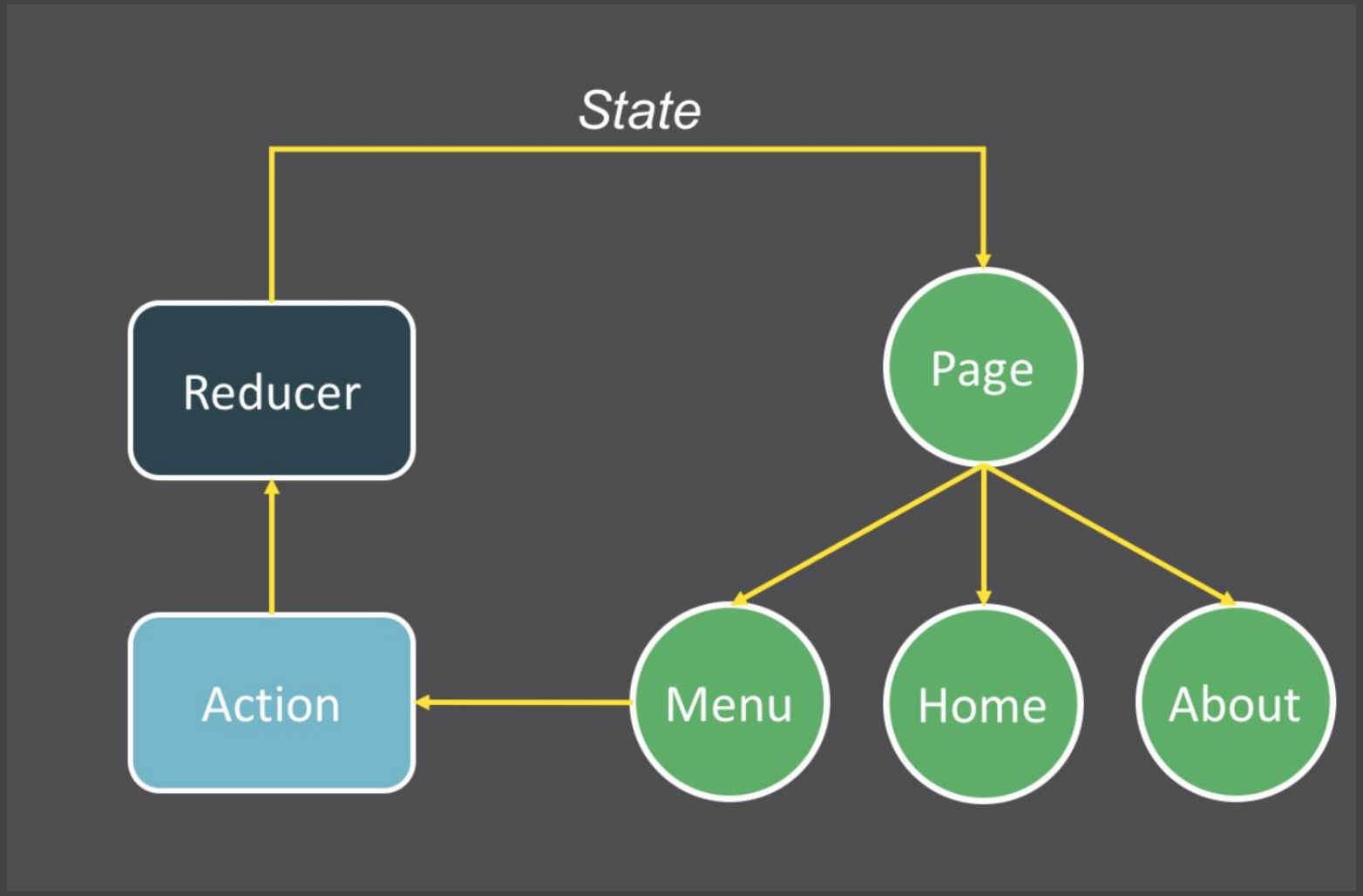
Redux is a “predictable state container for JavaScript apps”

“It helps you write applications that behave consistently, run in different environments (client, server, and native), and are easy to test.”

It’s not just about state management...

...it’s about *data flows*.


Center
Local
Layers
Layout
Hierarchy
Create
CoherentSurface
Cursor(Clone)
Enemies
Environment(dynamic)
Environment(semi-static)
Environment(sounds)
Environment(static)
InstantBullet(Clone)0
InstantBullet(Clone)1
InstantBullet(Clone)2
InstantBullet(Clone)3
InstantBullet(Clone)4
InstantBullet(Clone)5
Main Camera
Main CameraReflectionMain Camer
mechScorchMark(Clone)
mechScorchMark(Clone)
mechScorchMark(Clone)
Misc
Missile(Clone)0
Missile(Clone)1
Missile(Clone)2
Missile(Clone)3
Missile(Clone)4
Scene
Game
Free Aspect
Maximize on Play
Stats
Gizmos
Inspector
Front_face (Mesh Filter)
Mesh
front_face
Mesh Renderer
Cast Shadows
Receive Shadows
Materials
Use Light Probes
Coherent UIView (Script)
Script
CoherentUIView
Page
coui://UIResources/memoryl
Width
1024
Height
1024
Is Transparent
Support Click Through
Click Through Alpha Thre
0
Click To Focus
Draw After Post Effects
Flip Y
Intercept All Events
Enable Binding Attribute
Minigame Door Opener (Script)
Script
MinigameDoorOpener
Mesh Collider
Is Trigger
Material
None (Physic Material)
Convex
Smooth Sphere Collisions
Mesh
front_face
Preview
Project
Console
Create
Favorites
All Materi
All Models
All Prefab
All Scripts
Assets
AngryBot
Animatior
Editor
Explosion
Fonts
Gizmos
Materials
Objects
Assets
AngryBots
AngryBots
Animations
cube-textu...
Editor
Explosions
Fonts
Gizmos
Materials
MinigameD...
Objects
PhysicMate...
Plugins
Prefabs
Resources
Scenes
Scripts
Shaders
Sounds
Standard a...
Asset is unchanged


State
Reducer
Page
Action
Menu
Home
About

UI engineering is hard, but there are tools that make things simpler -- particularly at scale and over time

Three Principles

1. Single source of truth

The state of your whole application is stored in an object tree within a single store.

2. State is read-only

The only way to change the state is to emit an action, an object describing what happened.

3. Changes are made with pure functions

To specify how the state tree is transformed by actions, you write pure .reduce(rs)

Reducers are just functions that take the previous state and an action, and return the next state

$$state2 = f(state1, action)$$

<App /> = state

As your app grows, reducers can be split off into smaller functions that manage parts of your state tree

Side-effects are discouraged except through official channels (actions)

Because of the functional nature of Redux architecture, devs can achieve highly granular control over how data flows through an application in a consistent way

Developer tooling can be built up around this

And historically difficult things like undo / redo and time-travel become much less complex

Which makes debugging stateful applications like...


Score: 12,500
4
30
Moves left
18
0
9
12
0
19
Level 402
12

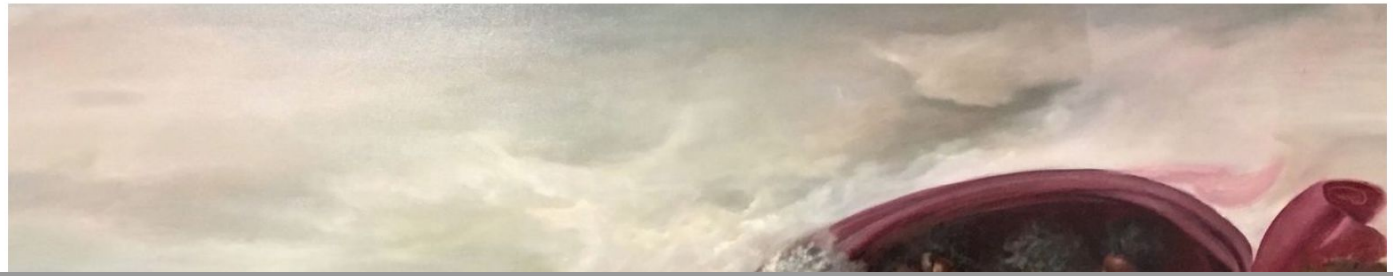
...much easier.

Use Case: Positron (aka Writer)

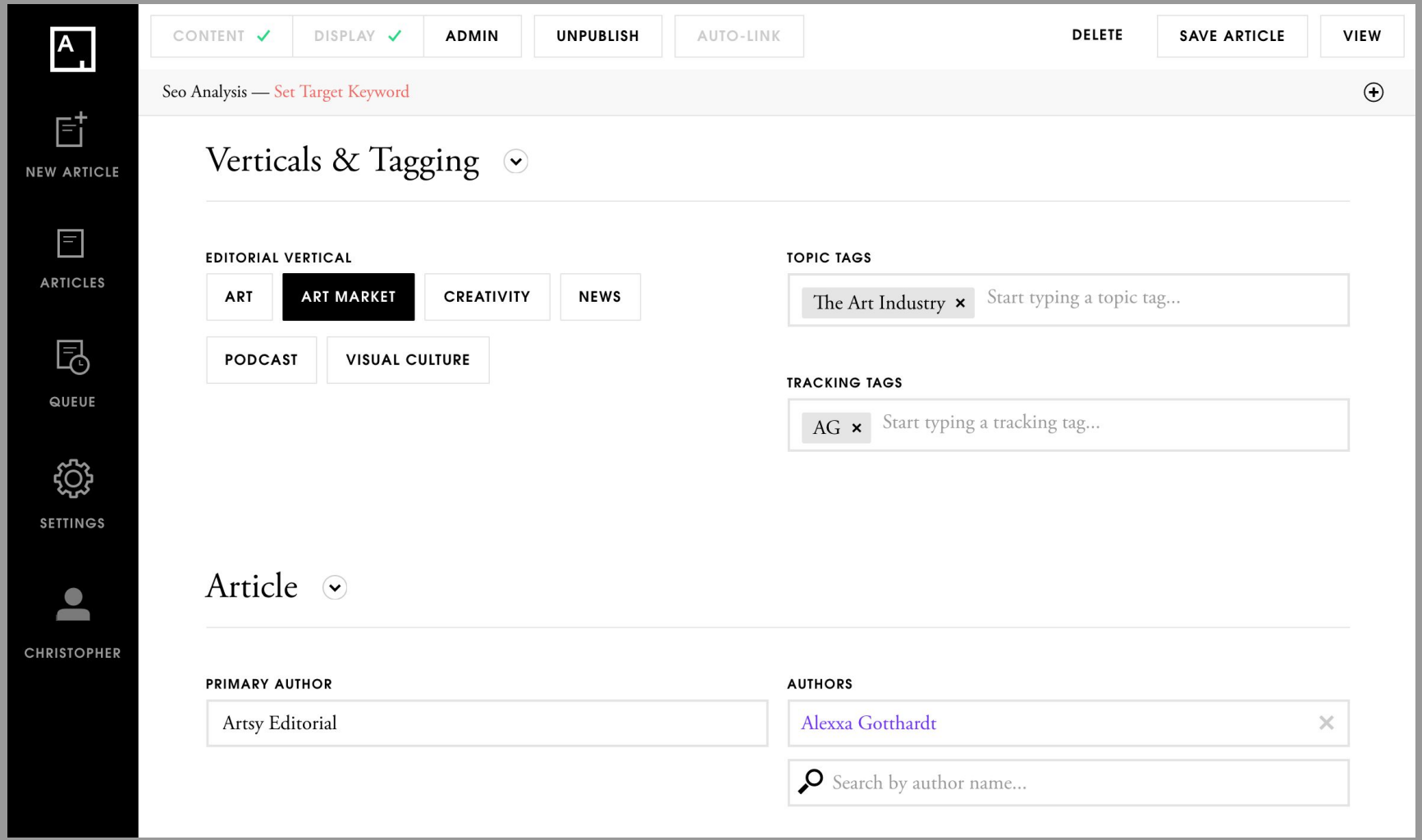
Art Market

This New $50k Prize Is Just for Emerging Female Figurative Painters

By Artsy Editors Apr 11, 2018 6:19 pm



Positron manages a *lot* of state


CONTENT ✓
DISPLAY ✓
ADMIN
UNPUBLISH
AUTO-LINK
DELETE
SAVE ARTICLE
VIEW
NEW ARTICLE
ARTICLES
QUEUE
SETTINGS
CHRISTOPHER
Seo Analysis — Set Target Keyword
Verticals & Tagging
EDITORIAL VERTICAL
ART
ART MARKET
CREATIVITY
NEWS
PODCAST
VISUAL CULTURE
TOPIC TAGS
The Art Industry ×
Start typing a topic tag...
TRACKING TAGS
AG ×
Start typing a tracking tag...
Article
PRIMARY AUTHOR
Artsy Editorial
AUTHORS
Alexxa Gotthardt
Search by author name...

NEW ARTICLE

ARTICLES

QUEUE

SETTINGS

CHRISTOPHER

Canvas

OVERLAY | IMAGE/VIDEO | SLIDESHOW

HEADLINE 0 Characters

Daniel Arsham The Angle of Repose Till Dec 23

CTA TEXT 0 Characters

Go to the exhibition page

CTA LINK

https://www.artsy.net/show/perrotin-daniel-arsham-the-angle-o

DISCLAIMER (OPTIONAL) 150 Characters

Enter legal disclaimer here

LOGO

IMAGE 1

IMAGE 2

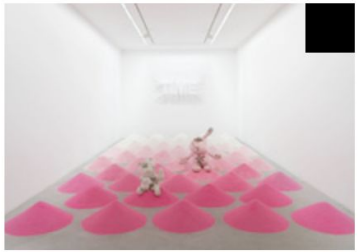


IMAGE 3

Positron’s Legacy Setup

- Backbone for models
- React components rendered inside Backbone parent views
- Mixture of Backbone and React in UI components
- React components used Backbone’s set/get methods for all data mutations

Pitfalls of Legacy Setup

- React components and Backbone views in same UI could have separate instances of shared data
    - React and Backbone lifecycles exist independently from each other
    - Listeners for Backbone ‘change’ events were scattered throughout app to trigger re-renders

- Props and onChange functions were passed down the entire component tree
    - Bug fixing is hard when tracking through deep layers of nesting

- Opportunities for data loss and for views becoming out-of-sync arose when components unmounted
    - Bug fixing is hard when tracking through deep layers of nesting
    - Rather than conditionally mounting components, we used jQuery to hide them

Benefits of Redux

- A single source of truth: Centralized data store is available to all components
    - A consistent source for form data, but also other info like saved status or client errors

- All components that access centralized store re-render automatically when data changes
    - And this operation can be memoized for highly performant update cycles

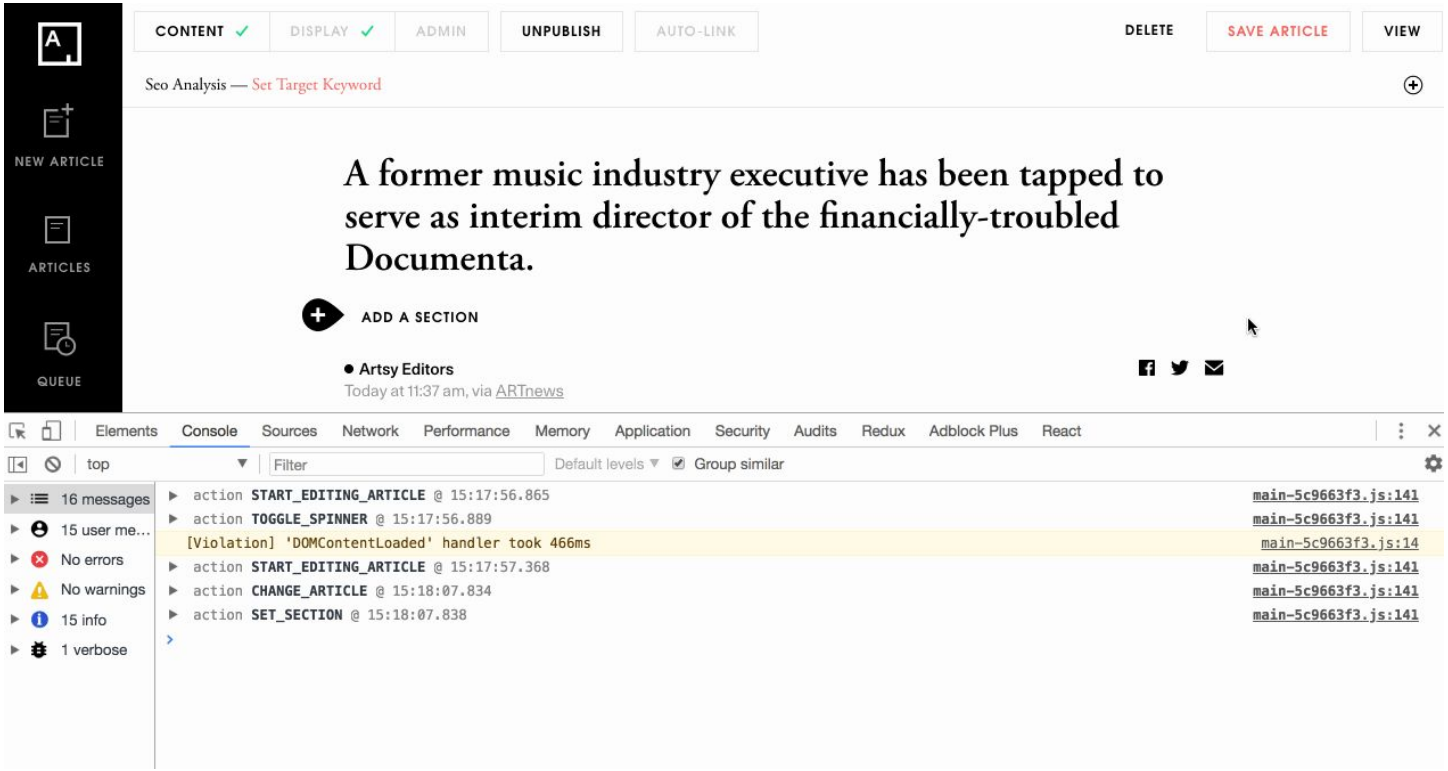
- Functions for mutating and saving data are consolidated to a centralized set of actions
    - Makes it easy to jump into a new codebase and see what’s possible

```
export const actions = keyMirror(
  'CHANGE_SAVED_STATUS',
  'CHANGE_VIEW',
  'CHANGE_SECTION',
  'CHANGE_ARTICLE',
  'UPDATE_ARTICLE',
  'START_EDITING_ARTICLE',
  'STOP_EDITING_ARTICLE',
  'DELETE_ARTICLE',
  'ERROR',
  'NEW_SECTION'
)
```

- Stop passing props!
    - Via `connect` devs can “pull” portions of state from “db” and inject into component as props

```
-   } else {
-     return (
-       <DropDownList
-         className='EditDisplay admin-form-container max-width-container'
-         activeSections={[0]}
-         openMany
-         sections={sections}
-       >
-         <DisplayMagazine
-           article={article}
-           onChange={onChange}
-         />
-         <DisplaySocial
-           article={article}
-           onChange={onChange}
-         />
-         <DisplaySearch
-           article={article}
-           onChange={onChange}
-         />
-         <DisplayEmail
-           article={article}
-           onChange={onChange}
-         />
-       </DropDownList>
-     )
-   }
+
+         : <DropDownList
+             className='admin-form-container max-width-container'
+             activeSections={[0]}
+             openMany
+             sections={sections}
+           >
+             <DisplayMagazine />
+             <DisplaySocial />
+             <DisplaySearch />
+             <DisplayEmail />
+           </DropDownList>
+       }
+     </EditArticleContainer>
+   )
```

Track dispatched actions in the console



Should I be using this?

Dan Abramov Follow
Working on @reactjs. Co-author of Redux and Create React App. Building tools for humans.
Sep 19, 2016 · 3 min read

You Might Not Need Redux

People often choose Redux before they need it. “What if our app doesn’t scale without it?” Later, developers frown at the indirection Redux introduced to their code. “Why do I have to touch three files to get a simple feature working?” Why indeed!

Blair Anderson Follow
Amazon Manufacturer's Representative @ https://www.AndersonAssociates.net/
Jul 16, 2017 · 3 min read

You Probably Don't Need Redux

> ***edit:*** *got some great comments below so i've added some links to the bottom of the article—Blair*

This is your daily reminder to listen to your gut.

Do not believe the hype.

Do not over-optimize.

Do not add libraries because they're popular.

Dan Abramov
@dan_abramov

Follow

Replying to @thecamjackson

I would like to amend this: don't use Redux until you have problems with vanilla React. See github.com/petehunt/react… @CamJackson89

petehunt/react-howto

react-howto - Your guide to the (sometimes overwhelming!) React ecosystem.

github.com

Dissecting Twitter’s Redux Store

https://medium.com/statuscode/dissecting-twitters-redux-store-d7280b62c6b1

Thank you!

Bonus!

“State so simple, it goes without saying”

https://github.com/jamiebuilds/unstated