Lessons learned from growing a React application

Cerize Santos

ReactJS Vancouver, June 26 2018

cerizesantos

About me

Software developer at Bananatag

All things Javascript - Node/React/Redux

Python/R/Data Science

Tech News

May, 2016

Cerize Joins Bananatag

Already considered the biggest news since Grumpy Cat meme! Many in Silicon Valley speculate she will work with React and Node.

Rer foll imp

cerizesantos

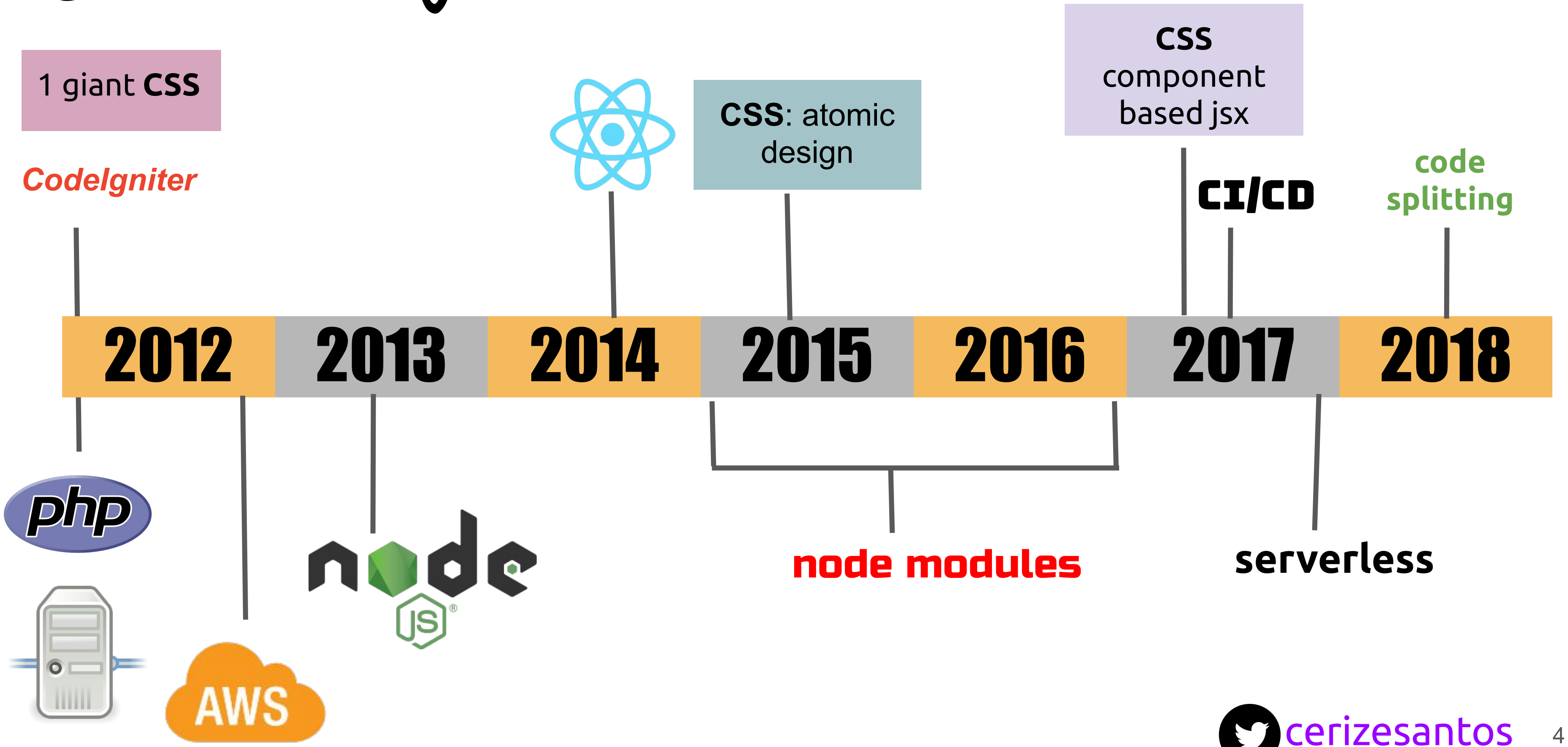
Disclaimer about this talk

~~Best Practices/ Should do~~ ✗

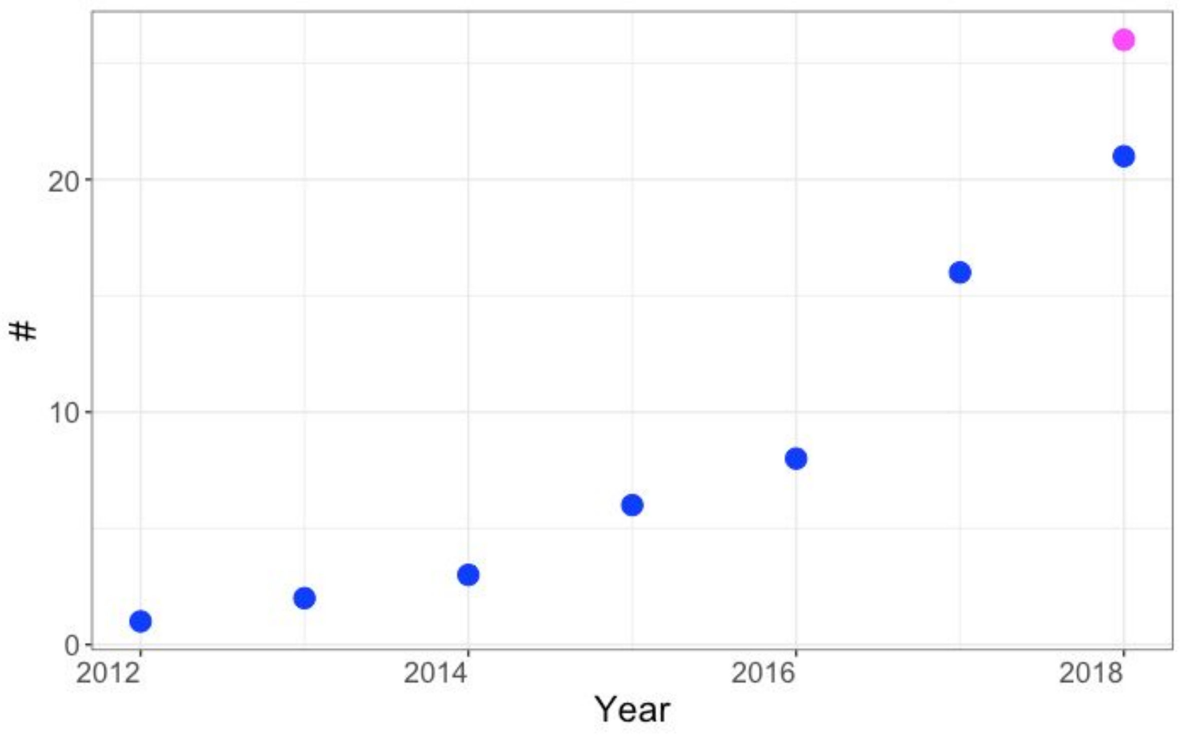
What worked for us

Mostly my own perceptions

Bananatag - stack timeline



Product Team

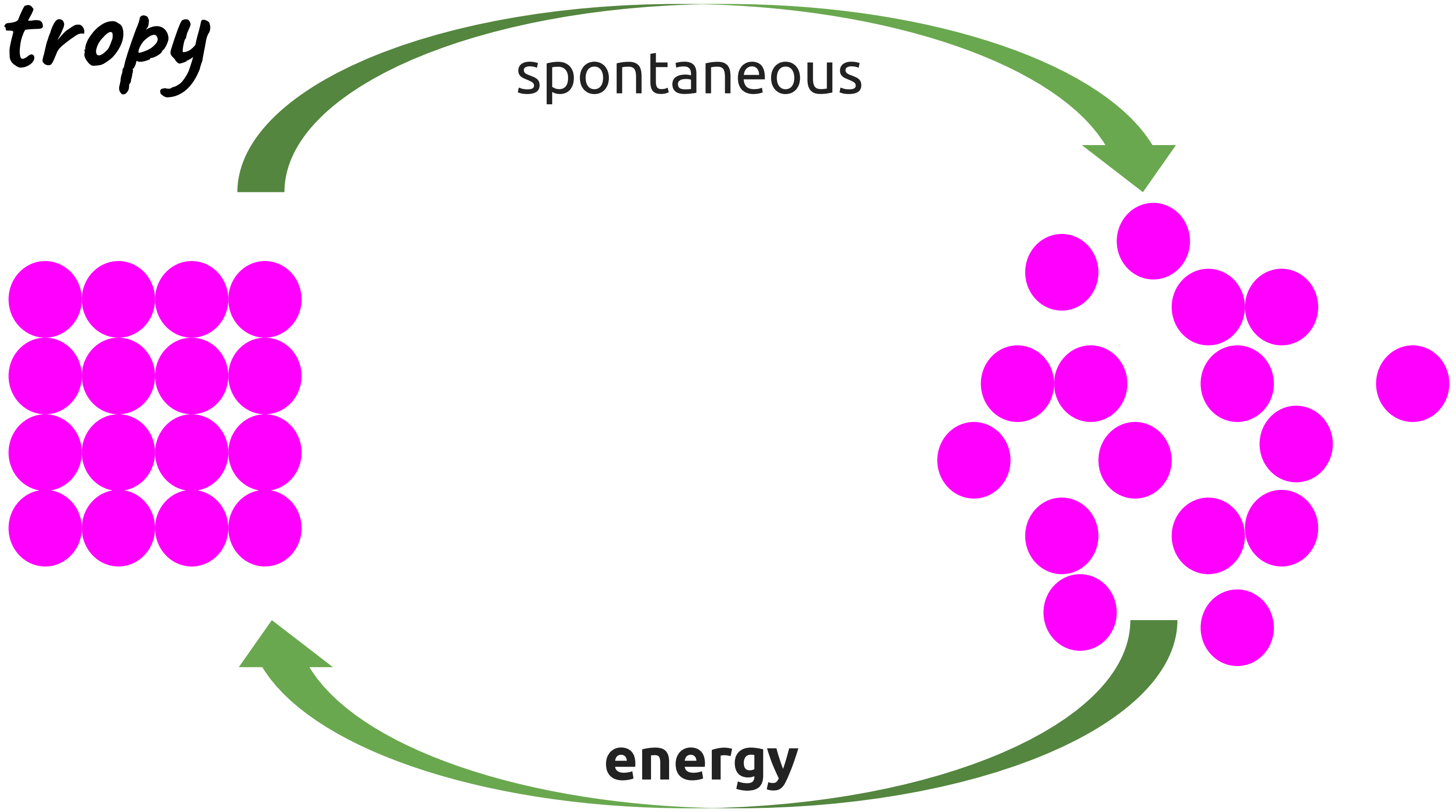


Choose one

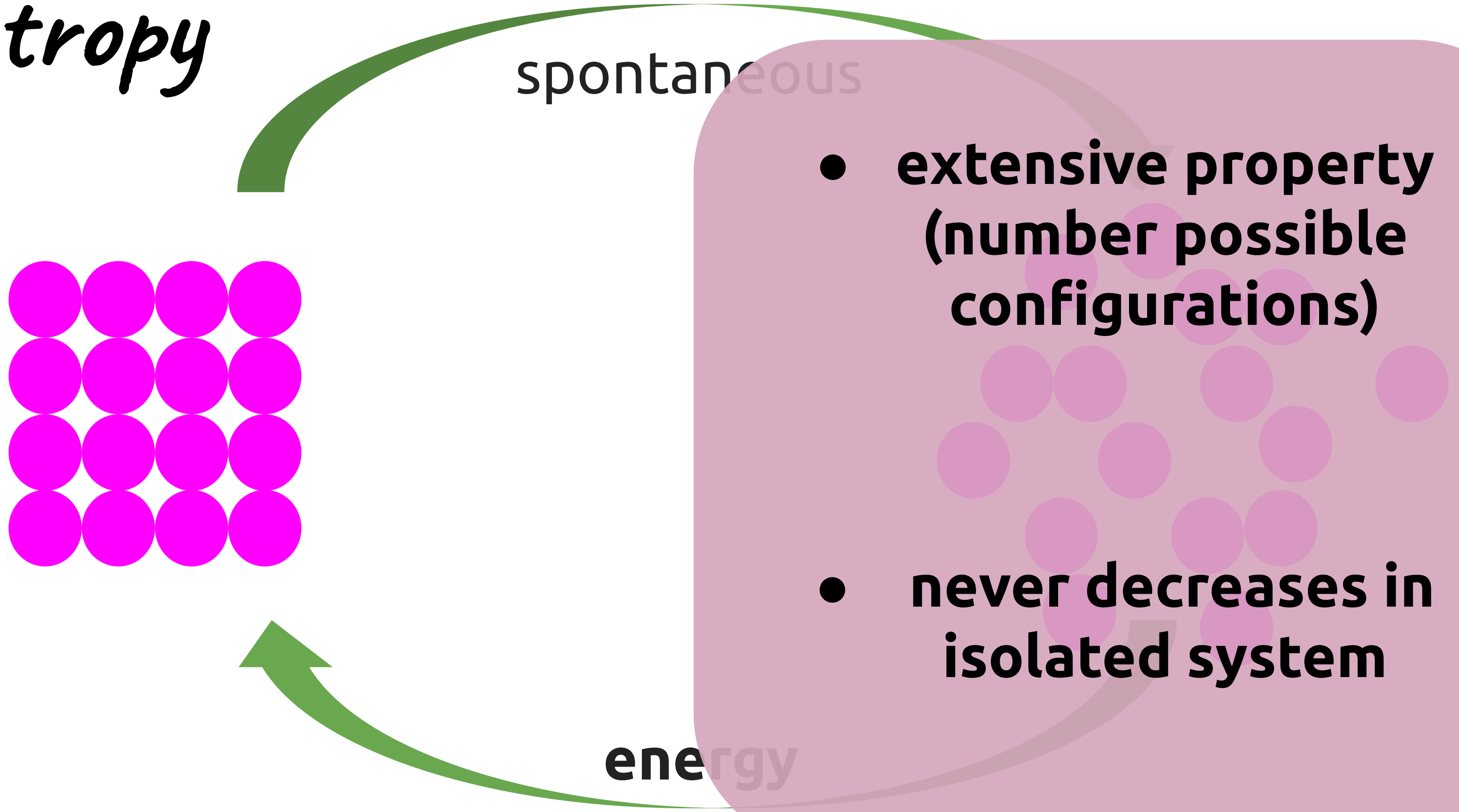
☐ Perfect architecture

☐ Stay in business

Entropy

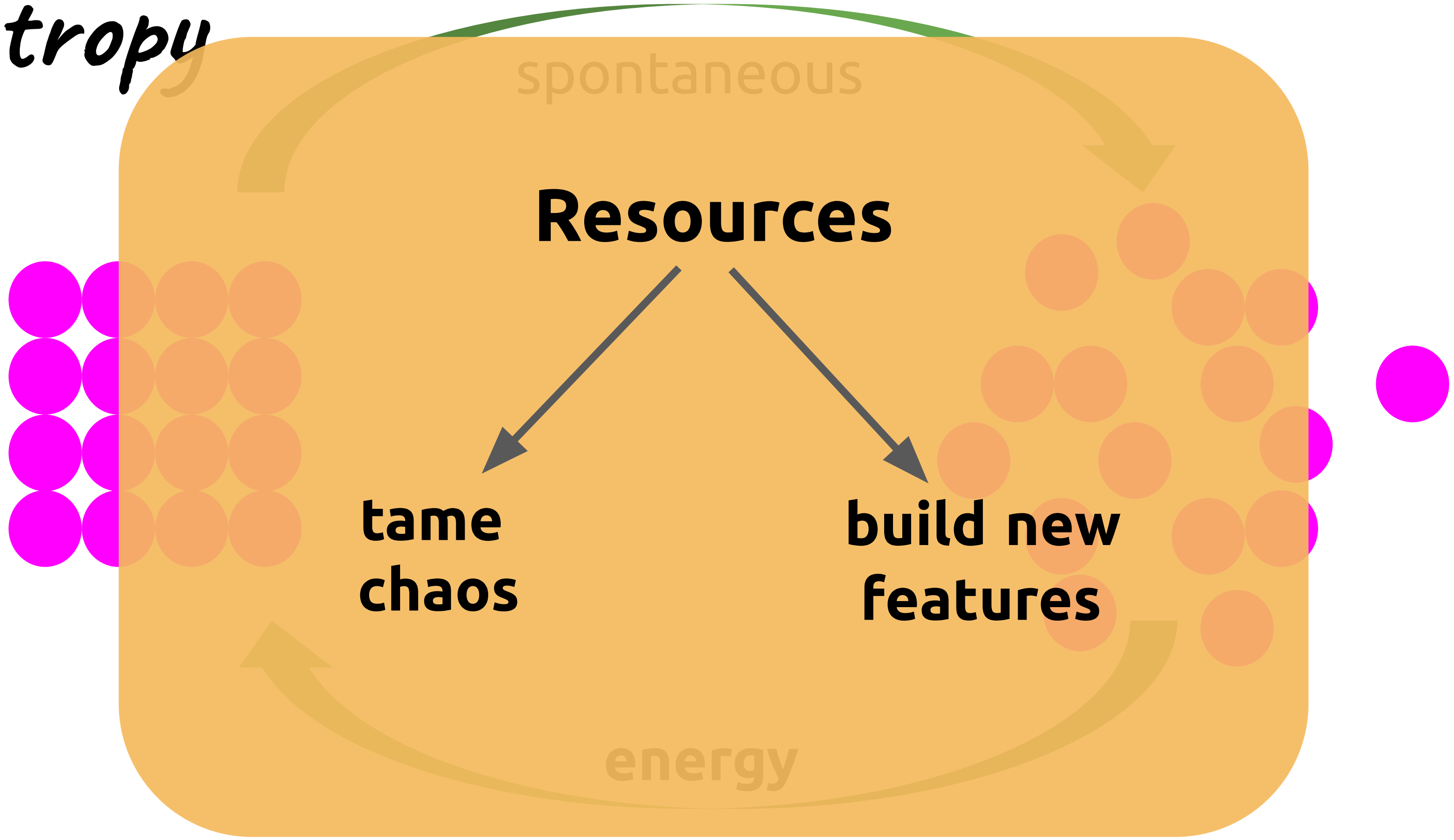


Entropy



- **extensive property (number possible configurations)**
- **never decreases in isolated system**

Entropy

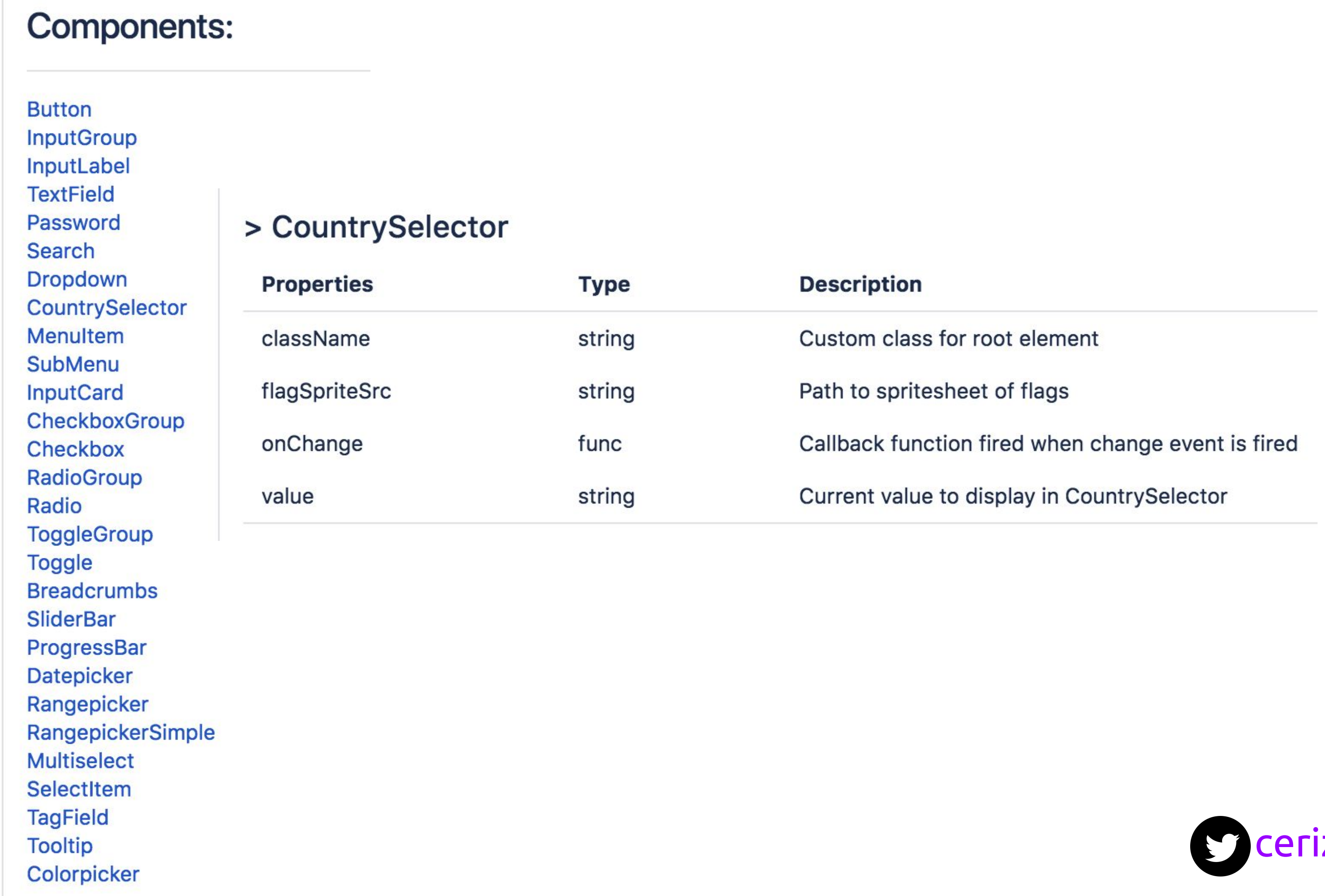

spontaneous
Resources
tame chaos
build new features
energy

1. Technical Debt

- know what good looks like
  - continued learning sessions
  - talented team

- know when to change
  - hiring - leadership
  - encourage proposals
  - support on implementation
  - communication across the team
  - healthy conflict

2. Component library



Components:

- Button
- InputGroup
- InputLabel
- TextField
- Password
- Search
- Dropdown
- CountrySelector
- MenuItem
- SubMenu
- InputCard
- CheckboxGroup
- Checkbox
- RadioGroup
- Radio
- ToggleGroup
- Toggle
- Breadcrumbs
- SliderBar
- ProgressBar
- Datepicker
- Rangepicker
- RangepickerSimple
- Multiselect
- SelectItem
- TagField
- Tooltip
- Colorpicker

> CountrySelector

Properties	Type	Description
className	string	Custom class for root element
flagSpriteSrc	string	Path to spritesheet of flags
onChange	func	Callback function fired when change event is fired
value	string	Current value to display in CountrySelector

Two distinct ways of seeing the components:

library components

- flexibility is key
- black boxes - keep interface
- independent of each other
- different components can follow different patterns

application components

- consistency is key
- keep things simple
- spend time on the state management and data flow patterns

State management

- Single source of truth
    - If something can be derived from either props or state, it probably shouldn’t be in the state

- lift up state vs put in the a global store (redux)

- local state vs redux store

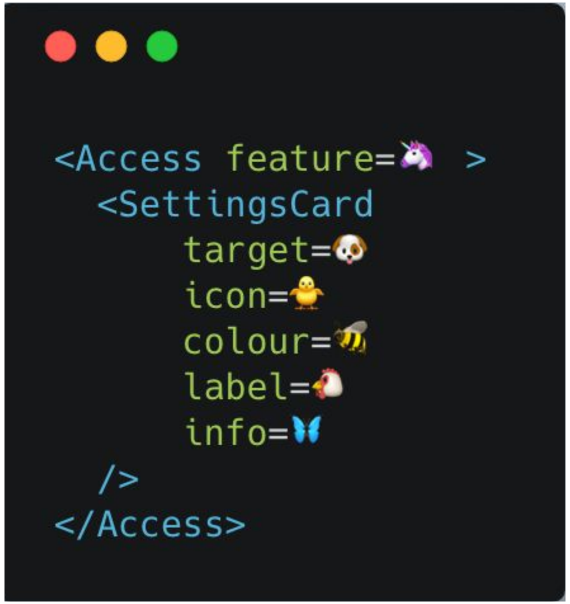
3. CSS

CSS

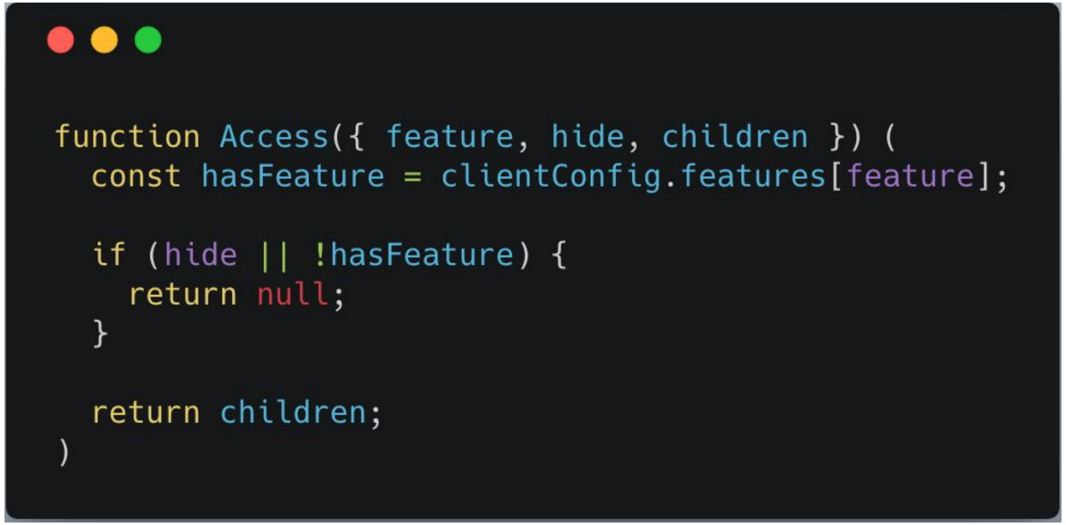
- Separate style from functions
- Designers are our bosses - responsible for the CSS architecture
- All designers can code

4. Abstraction vs Duplication

Access

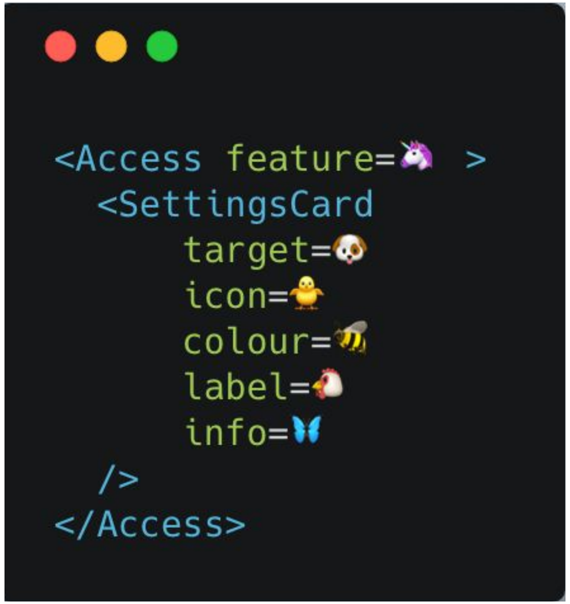


```
<Access feature=🦄 >
  <SettingsCard
      target=🐶
      icon=🐥
      colour=🐝
      label=🐔
      info=🦋
  />
</Access>
```

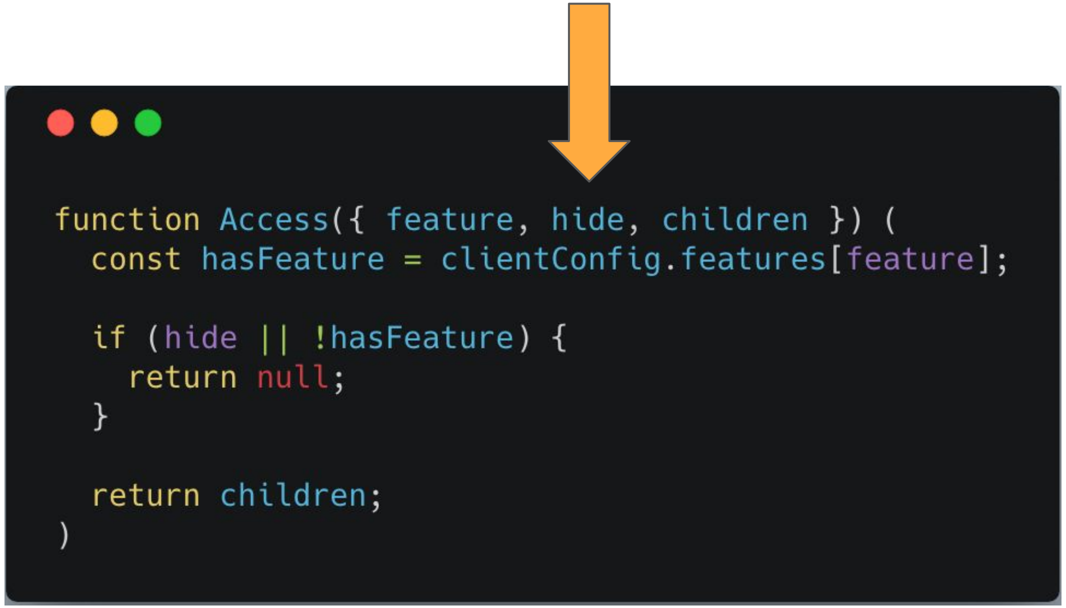


```
function Access({ feature, hide, children }) (
  const hasFeature = clientConfig.features[feature];

  if (hide || !hasFeature) {
    return null;
  }

  return children;
)
```

Access

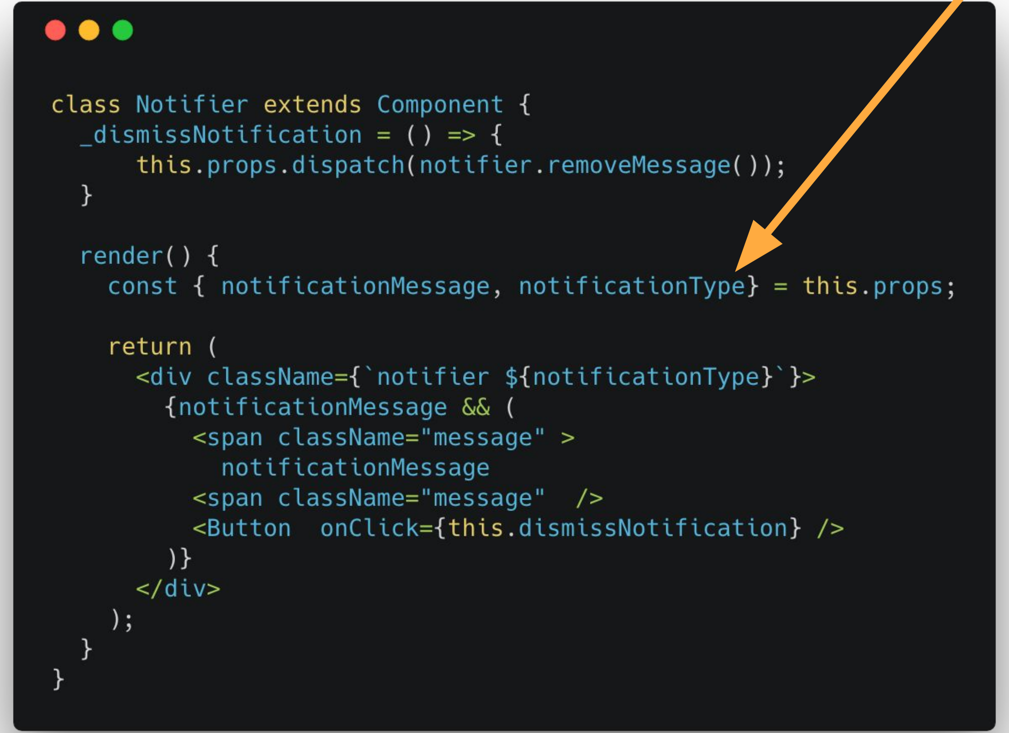


```
<Access feature=🦄 >
  <SettingsCard
      target=🐶
      icon=🐥
      colour=🐝
      label=🐔
      info=🦋
  />
</Access>
```



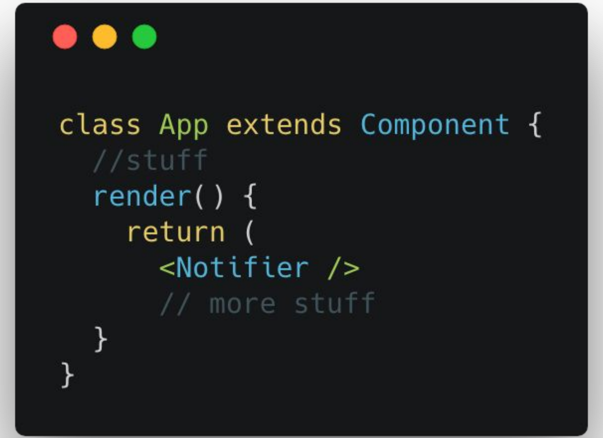
```
function Access({ feature, hide, children }) (
  const hasFeature = clientConfig.features[feature];

  if (hide || !hasFeature) {
    return null;
  }

  return children;
)
```

Notification - components



on Redux

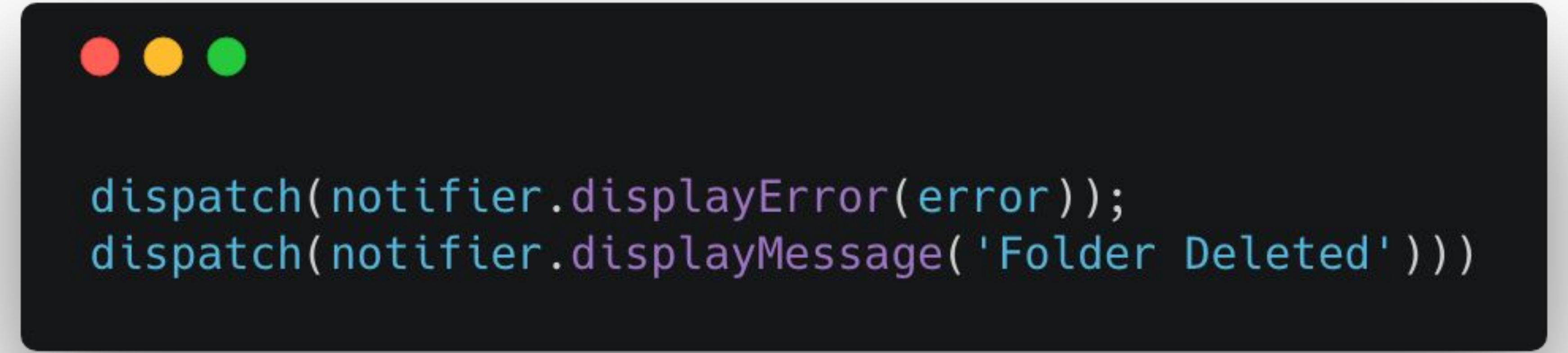
```
class Notifier extends Component {
  _dismissNotification = () => {
      this.props.dispatch(notifier.removeMessage());
  }

  render() {
    const { notificationMessage, notificationType} = this.props;

    return (
      <div className={`notifier ${notificationType}`}>
        {notificationMessage && (
          <span className="message" >
            notificationMessage
          <span className="message"  />
          <Button  onClick={this.dismissNotification} />
        )}
      </div>
    );
  }
}
```



```
class App extends Component {
  //stuff
  render() {
    return (
      <Notifier />
      // more stuff
  }
}
```

cerizesantos

Notification - actions

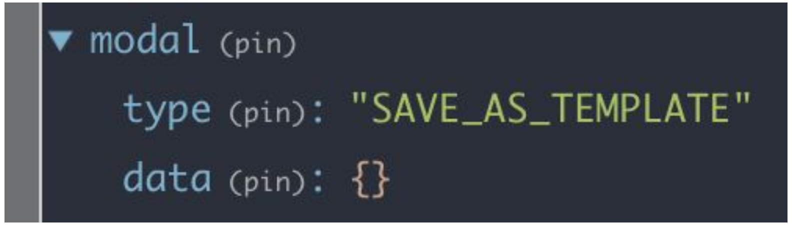


```
dispatch(notifier.displayError(error));
dispatch(notifier.displayMessage('Folder Deleted')))
```

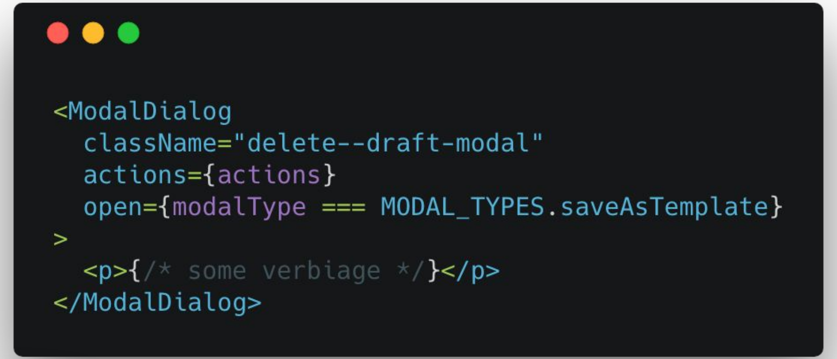
Modals

Component

Redux

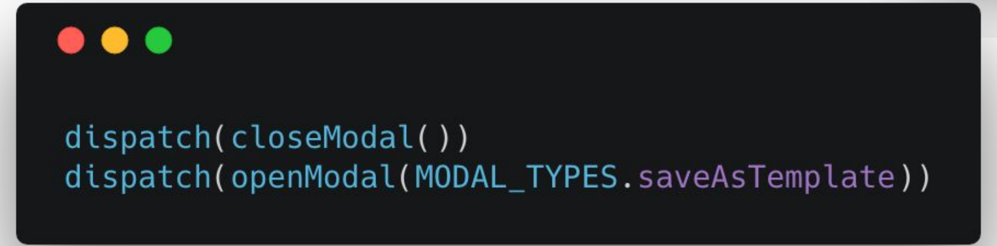


```
▼ modal (pin)
    type (pin): "SAVE_AS_TEMPLATE"
    data (pin): {}
```



```
<ModalDialog
  className="delete--draft-modal"
  actions={actions}
  open={modalType === MODAL_TYPES.saveAsTemplate}
>
  <p>{/* some verbiage */}</p>
</ModalDialog>
```

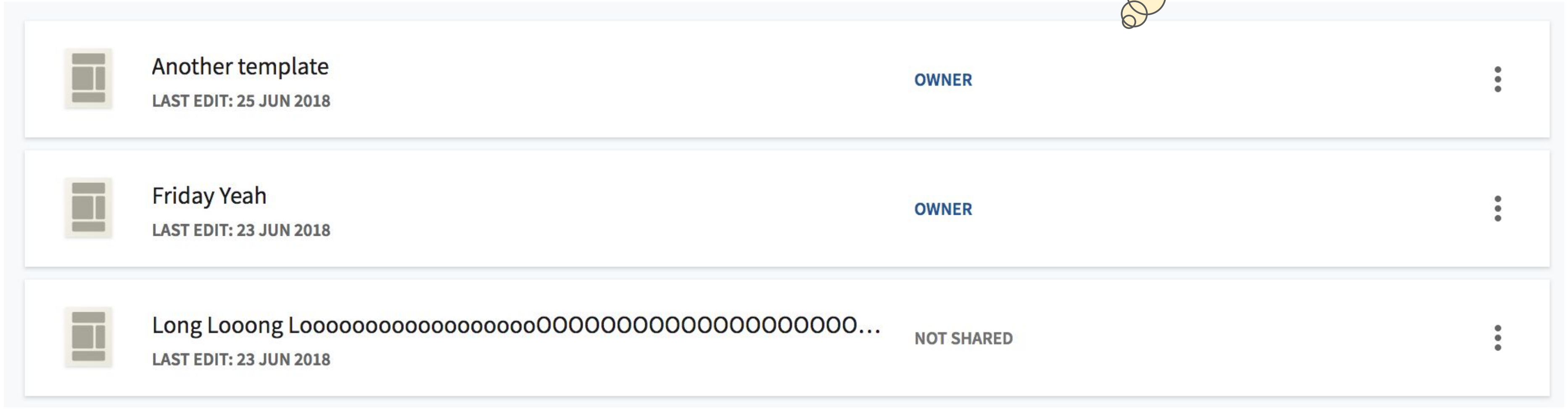
Actions



```
dispatch(closeModal())
dispatch(openModal(MODAL_TYPES.saveAsTemplate))
```

The Assets Table

Let’s have a component for all lists like that!

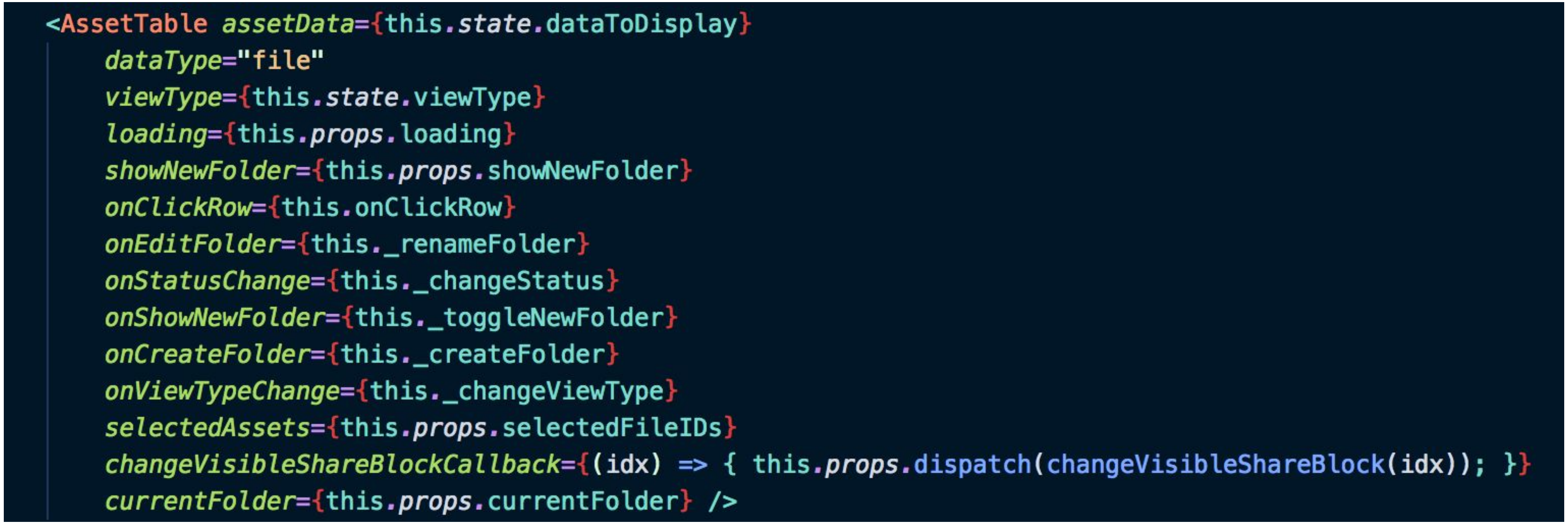


Another template
LAST EDIT: 25 JUN 2018
OWNER

Friday Yeah
LAST EDIT: 23 JUN 2018
OWNER

Long Looong LooooooooooooooooooOOOOOOOOOOOOOOOOOOOO...
LAST EDIT: 23 JUN 2018
NOT SHARED

The Assets Table - when things went wrong



```
<AssetTable assetData={this.state.dataToDisplay}
    dataType="file"
    viewType={this.state.viewType}
    loading={this.props.loading}
    showNewFolder={this.props.showNewFolder}
    onClickRow={this.onClickRow}
    onEditFolder={this._renameFolder}
    onStatusChange={this._changeStatus}
    onShowNewFolder={this._toggleNewFolder}
    onCreateFolder={this._createFolder}
    onViewTypeChange={this._changeViewType}
    selectedAssets={this.props.selectedFileIDs}
    changeVisibleShareBlockCallback={(idx) => { this.props.dispatch(changeVisibleShareBlock(idx)); }}
    currentFolder={this.props.currentFolder} />
```

5. Best Practices Guide for React

React/Redux

- Define the components architecture before start coding - make it a team decision when implementing new features
- All components use Immutable.js structures
- When to use local state?
    - That state is not used anywhere else (if that state means something to other part of the app, move it to redux)
    - Other part of the app does not affect that part of the dom (you want to coordinate components that are not parent-child via redux state. If you decide to lift the state to a parent so both components can inherit the state, don't do that if the prop has to be passed more than one level)
    - You don't need to keep the state when component unmounts (after unmount the state will reset to its initial state)

  PS: These are must haves for local state. It is not enforced to have local state even if all those check.
- If a component has more than one child, it should be put it in a separate folder
- Name actions according to current patterns. For example, for async actions:
    - GET_ALL_REQUEST
    - GET_ALL_SUCCESS
    - GET_ALL_FAIL

from React docs...

(talking about conditional rendering)

"Just like in JavaScript, it is up to you to choose an appropriate style based on what you and your team consider more readable. "

Bananatag is hiring!!!

bananatag.com

1 of Bananatag

many perks:

Watch Martha

the Raccoon

Thank you

Bananatag Team

Get in touch

cerizesantos

cerizesantos@gmail.com