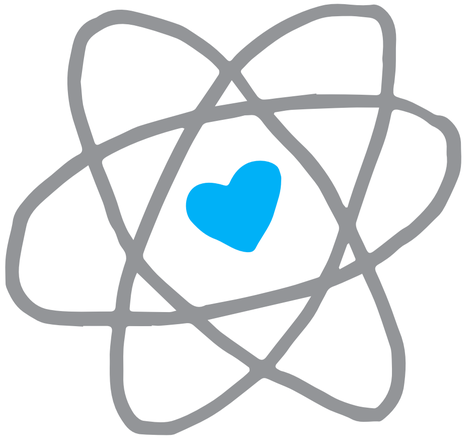
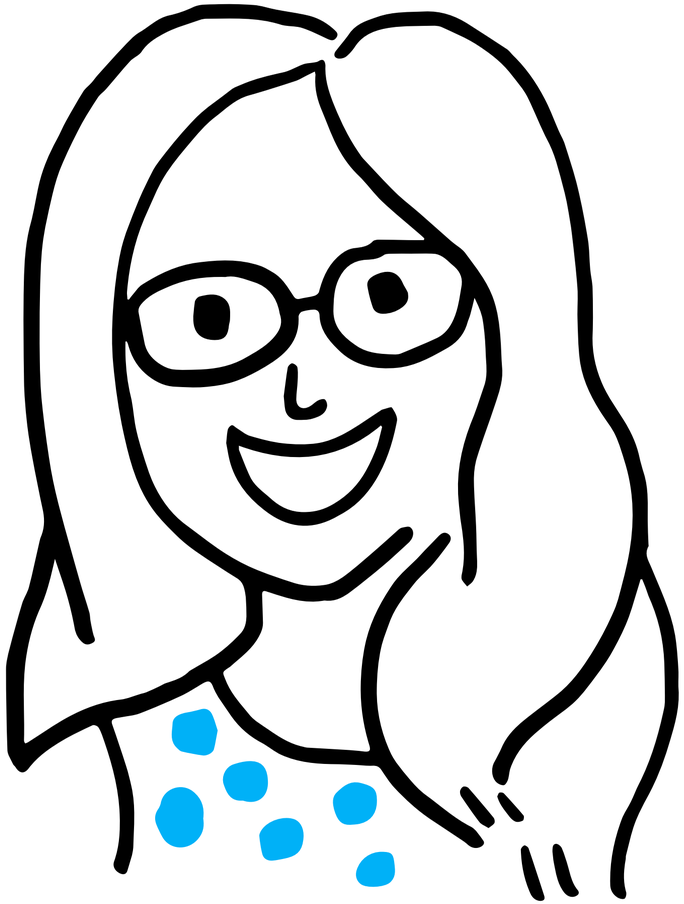
THE HOW AND WHY OF FLEXIBLE REACT COMPONENTS

Hi! I'm Jenn.
Senior Frontend Engineer, ClassPass

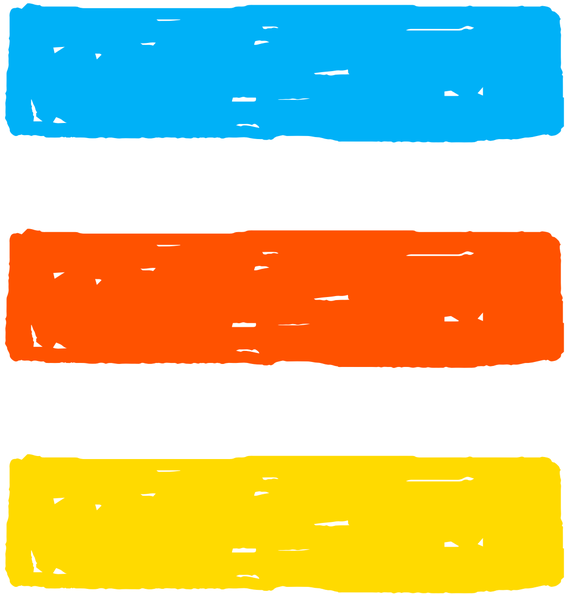
@gurlcode



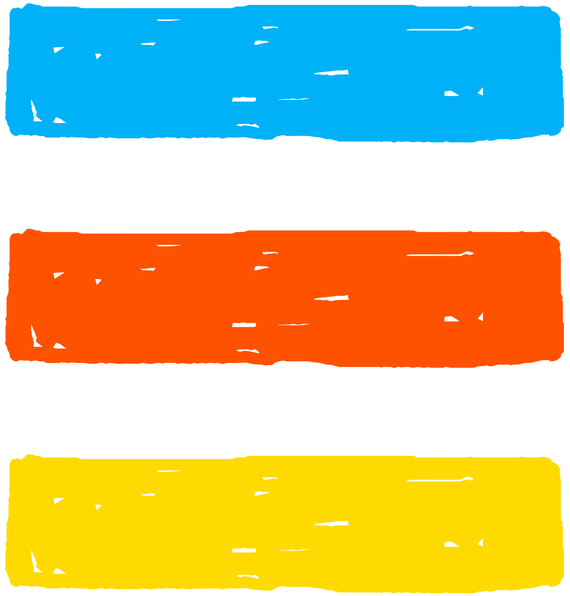


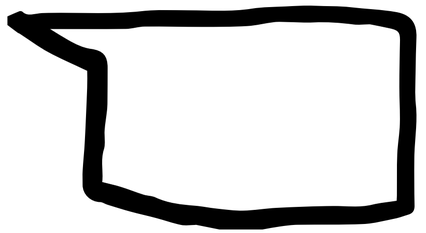
@gurlcode

flexible



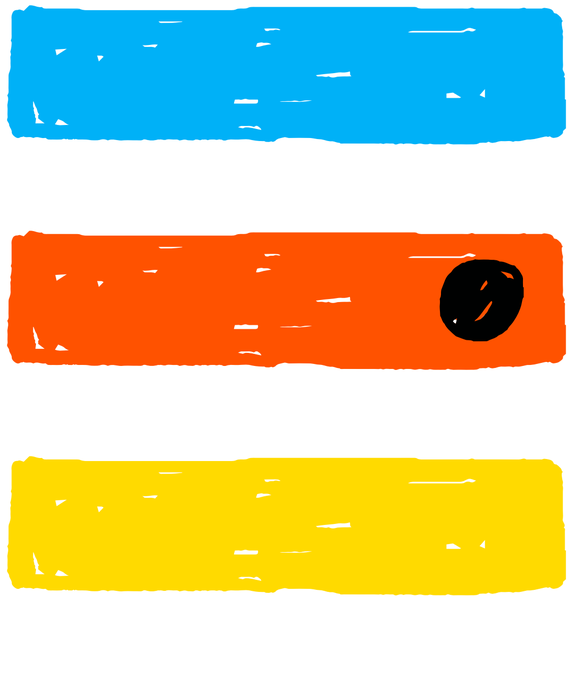
withTooltip

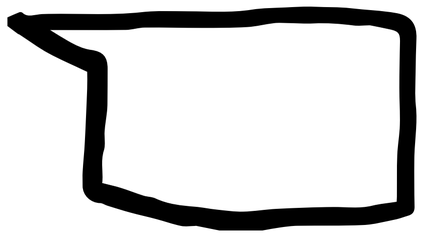




withDot

withTooltip



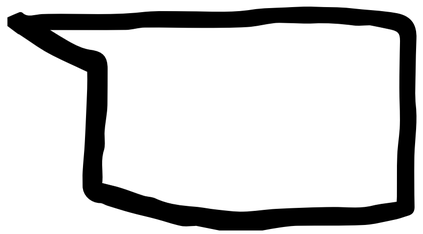


@gurlcode

withModal

withDot

withTooltip

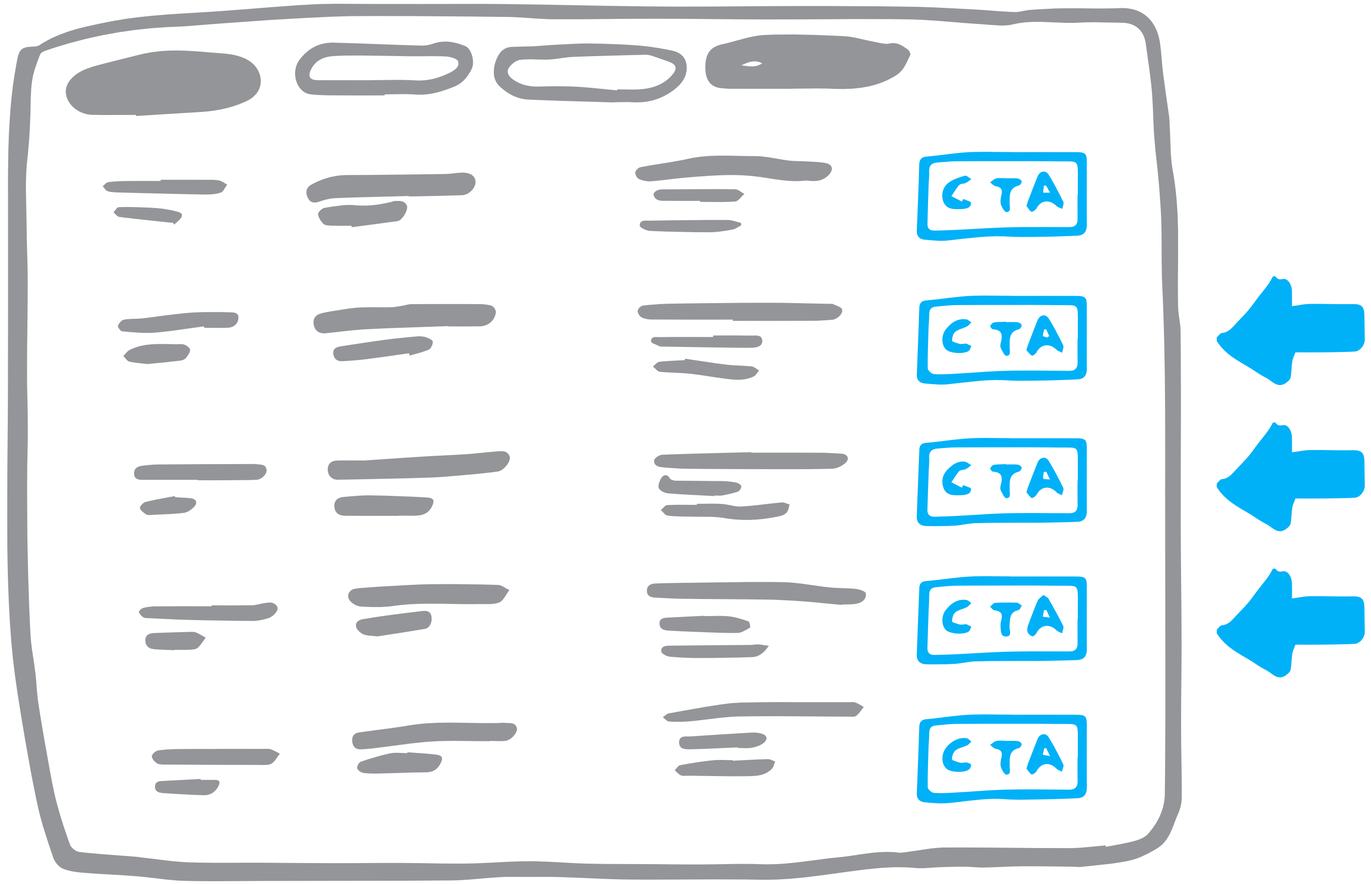


@gurlcode

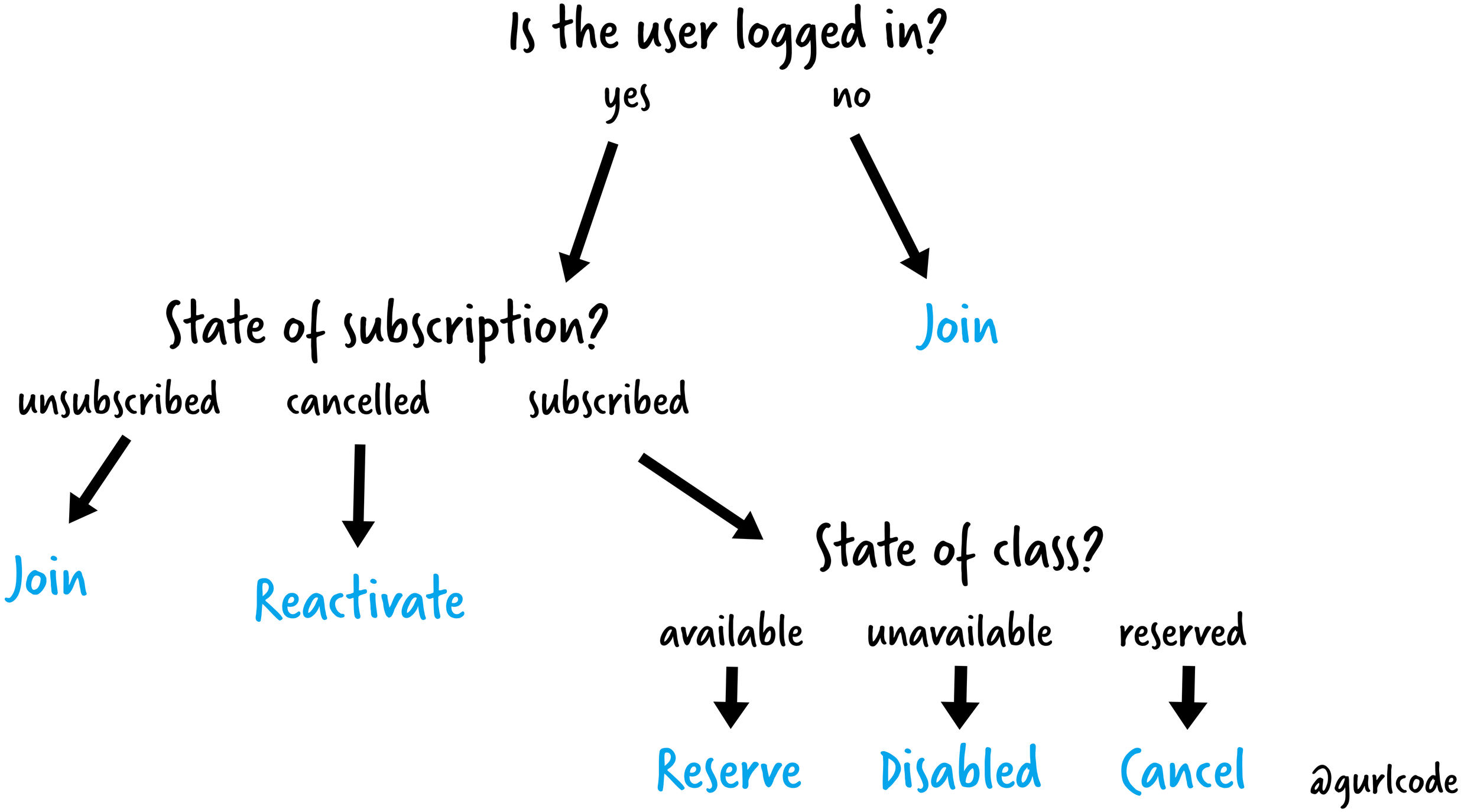
```
<BlockComponent
  className="block block—md block—orange"
  withTooltip={false}
  withDot={false}
  withModal
  modalProps={modalProps}
  onModalClose={onModalClose}
  onMouseOver={onMouseOver}
  withExperiment
  experimentId={experimentId}
  userName={userName}
  userId={userId}
  analyticsProps={analyticsProps}
  roundedCorners
/>
```

BEWARE THE APROPSCALYPSE

@gurlcode



@gurlcode

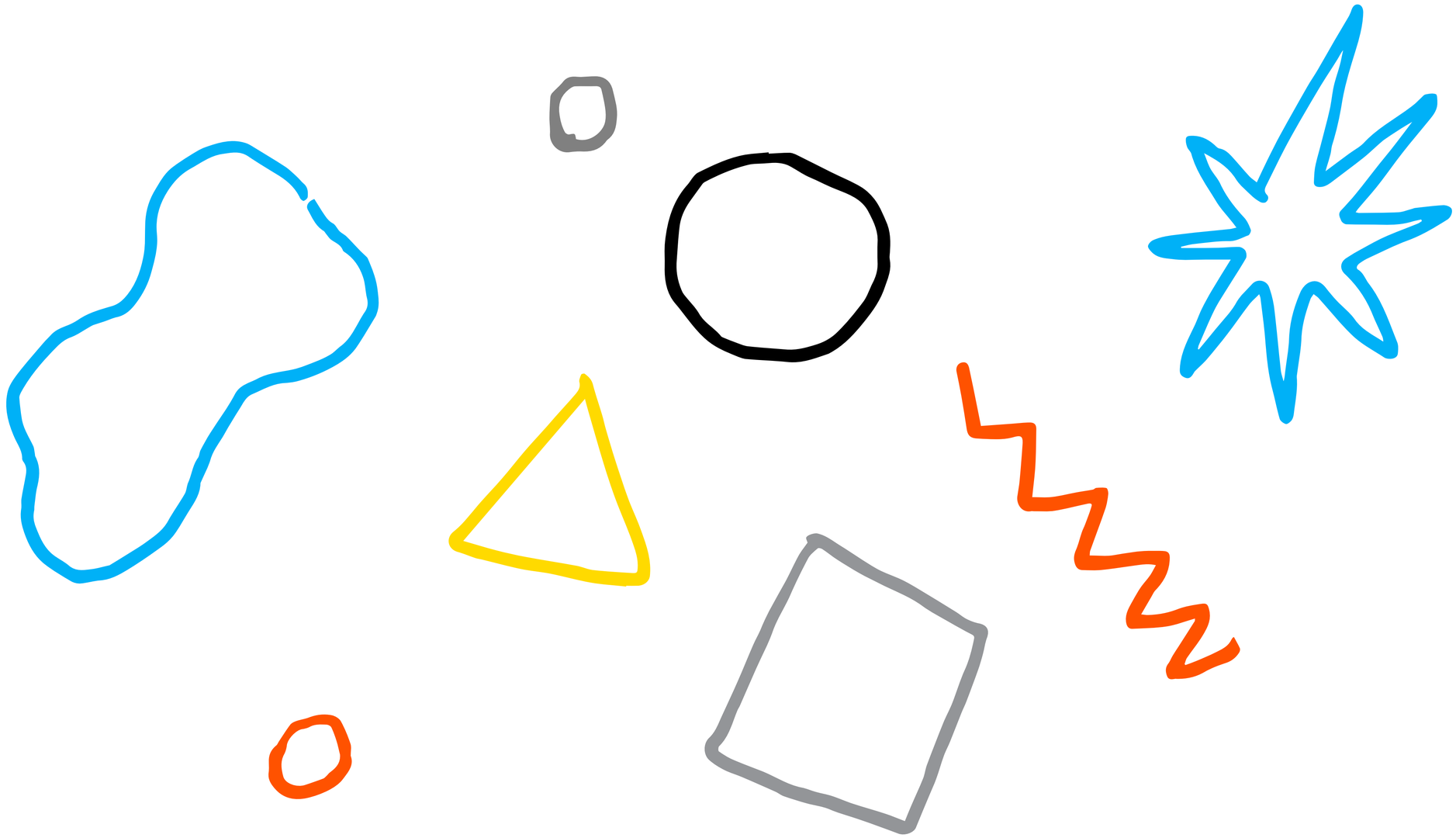

Is the user logged in?
yes
no
State of subscription?
Join
unsubscribed
cancelled
subscribed
Join
Reactivate
State of class?
available
unavailable
reserved
Reserve
Disabled
Cancel
@gurlcode

```
import PropTypes from 'prop-types';
import React from 'react';
import classNames from 'classnames';
import { FormattedMessage } from 'react-intl';
import { Link } from 'react-router';
import AVAILABILITY_REASONS from 'constants/availability-reasons';
import memberStatusEnum from 'enums/memberStatus';
import { ga } from 'helpers/multitrack';
import { ScheduleCtaAlcPurchase } from './ScheduleCtaAlcPurchase';

export const ScheduleCtaVisitor = ({
  startTime,
  ctaDestination,
  scheduleId,
  className,
}) => startTime * 1e3 > Date.now() && (
  <Link
    className={classNames(className, 'bt bt--lg bt--width-full')}
    to={ctaDestination}
    onClick={() => ga('Venue Details', 'Schedule Join Click', scheduleId)}
  >
    <FormattedMessage
      id="schedule.row.cta-join"
      defaultMessage="Join"
    />
  </Link>
);

ScheduleCtaVisitor.propTypes = {
```

```
<ScheduleCta
  availabilityLabel={availabilityLabel}
  availableToPurchase={availableToPurchase}
  availableToReserve={availableToReserve}
  className="tooltip tooltip--right tooltip--md-center"
  creditPrice={creditPrice}
  creditsReasons={creditsReasons}
  memberStatus={memberStatus}
  onReserveClicked={onReserveClicked}
  purchaseAmount={purchaseAmount}
  purchaseCurrency={purchaseCurrency}
  reasonCannotReserve={reasonCannotReserve}
  scheduleId={scheduleId}
  showCreditPricing={showCreditPricing}
  startTime={startTime}
/>
```

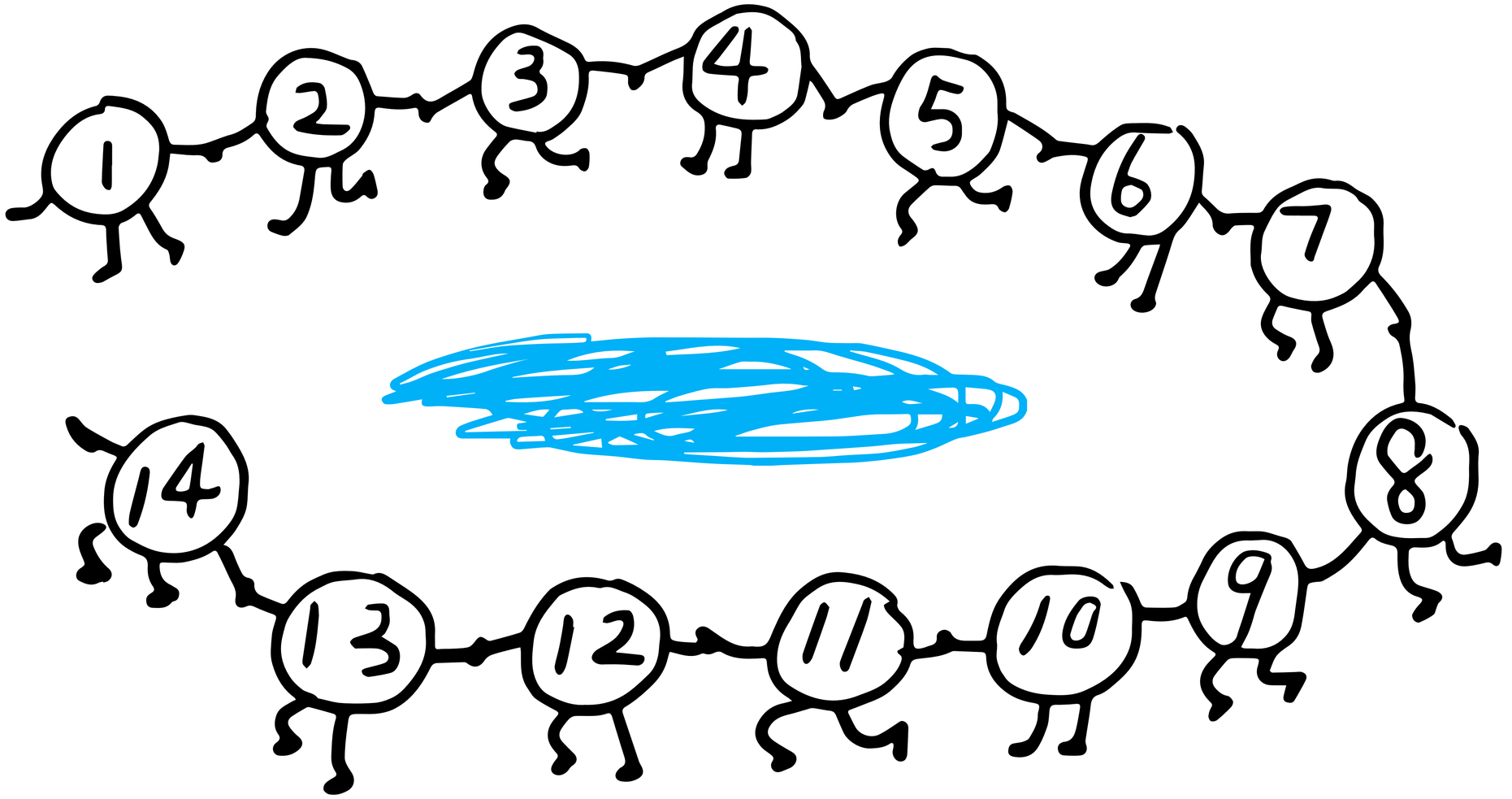
reusable !== flexible

Flexibility is about more than reusability.
It's about the ability understand and augment.

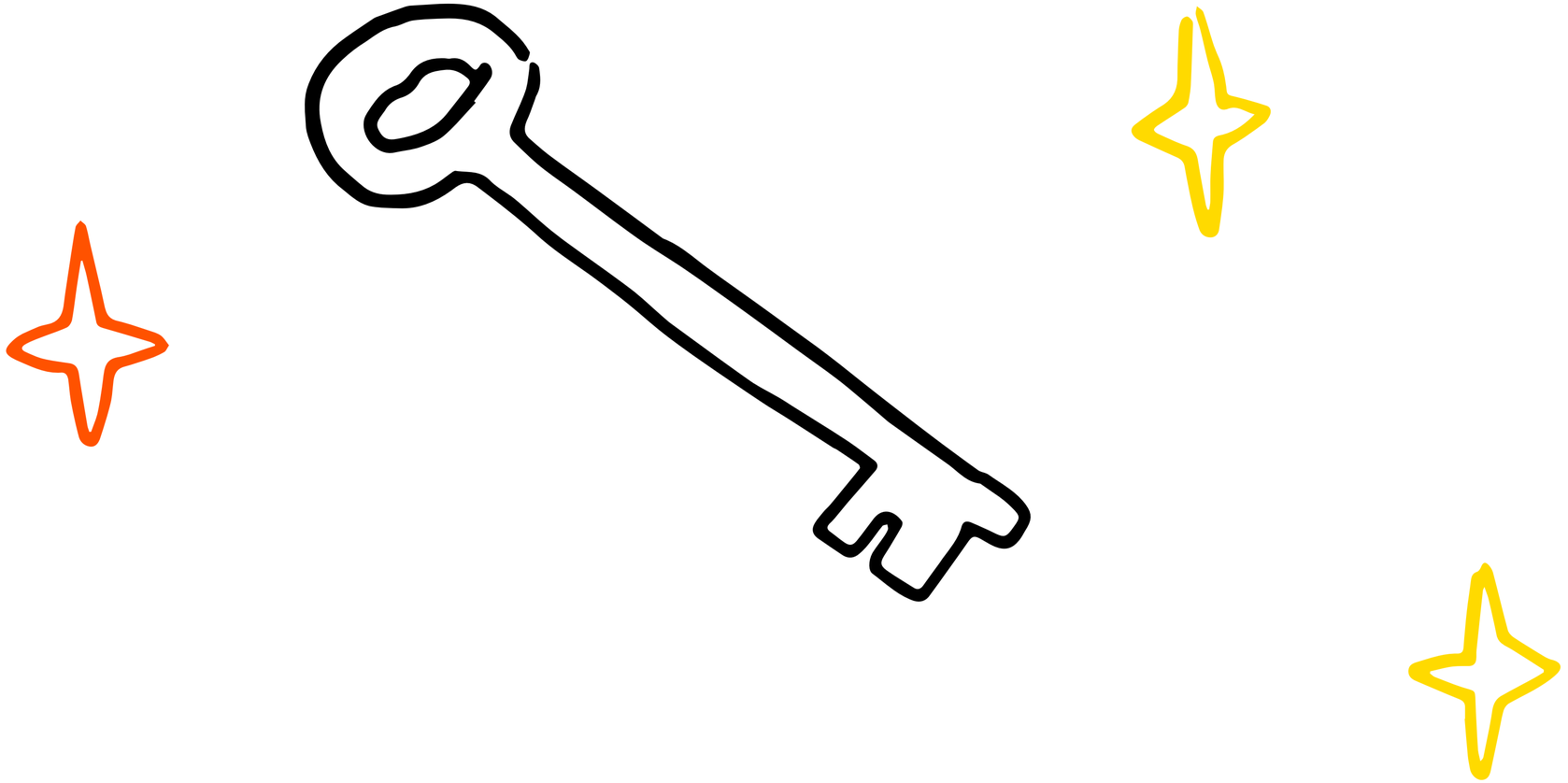


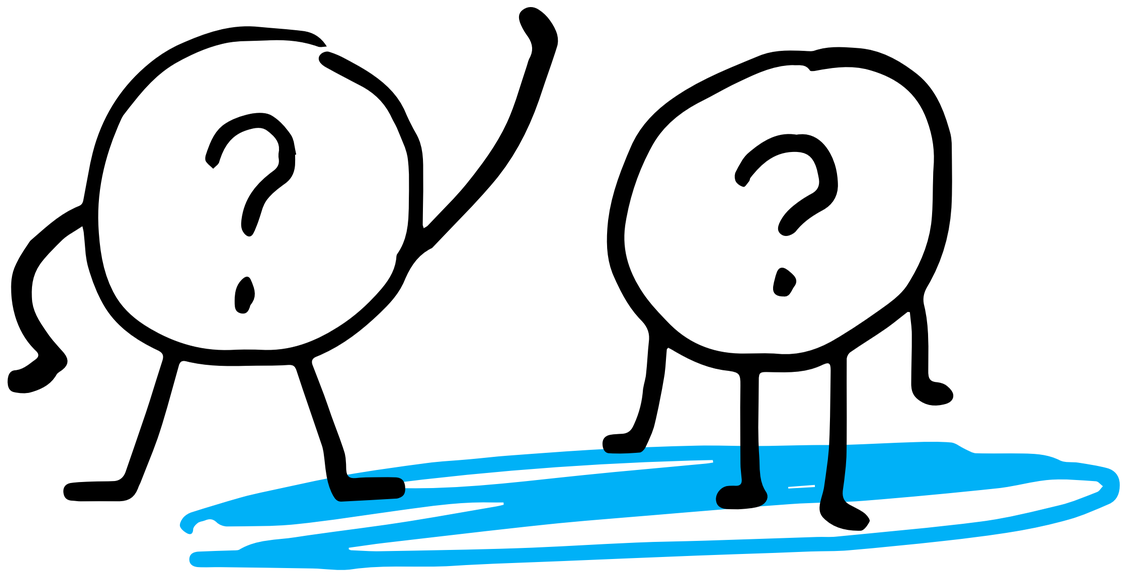
@gurlcode

<ScheduleCta type={?} />



@gurlcode





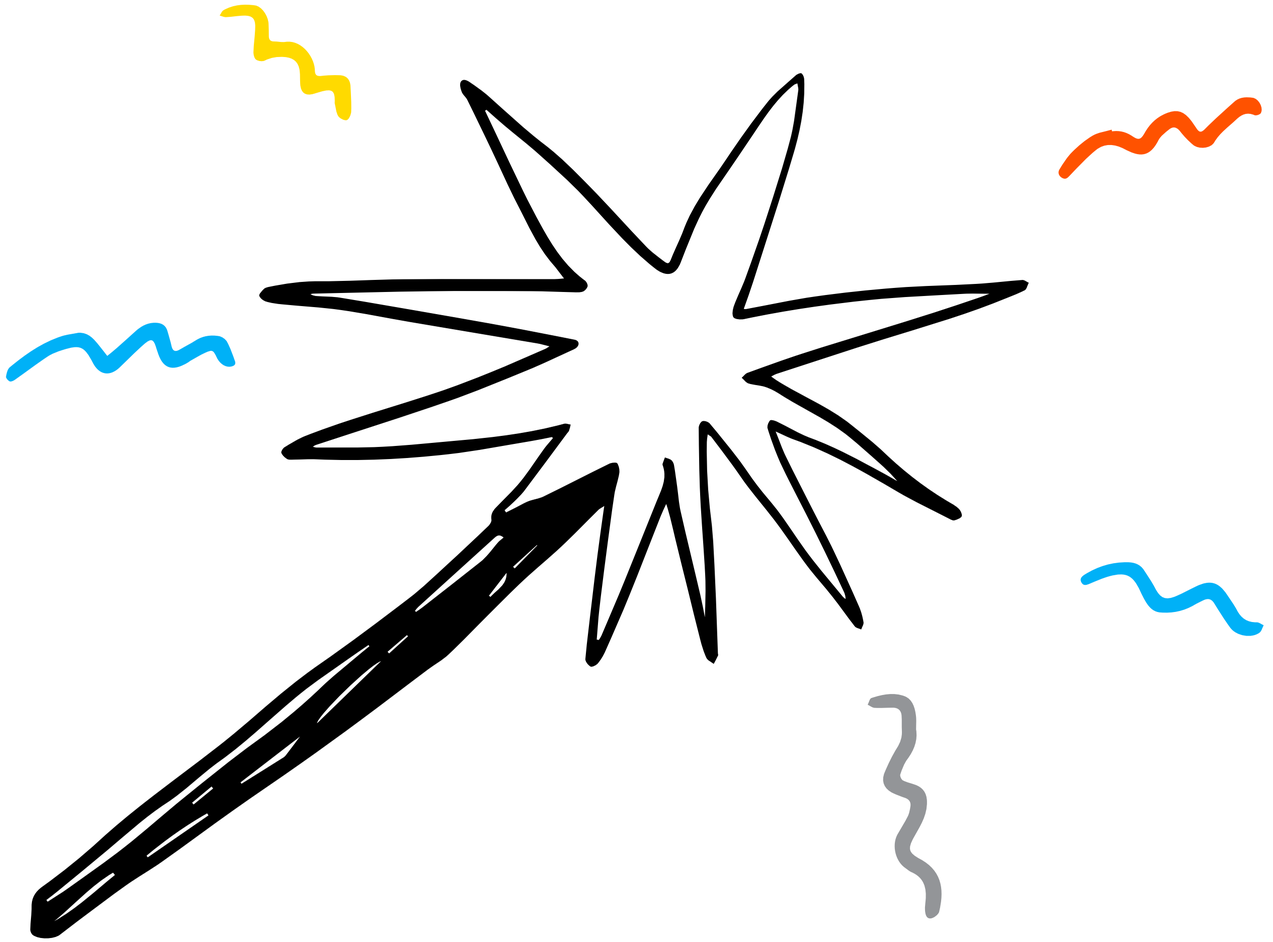
Keep business logic out of your components.

```
const getScheduleAction = (memberStatus, schedule, permissions = {}) => {
  const {
    availableToPurchase,
    availableToReserve,
    reasonCode,
  } = ScheduleDeriver.availabilityData(schedule);

  const NOT_A_MEMBER = 'NOT_A_MEMBER';

  switch (memberStatus || NOT_A_MEMBER) {
    case memberStatusType('UNPAID'):
    case memberStatusType('ACTIVE'):
      if (reasonCode === RESERVED) {
        return 'cancel';
      }

      if (!availableToReserve && availableToPurchase) {
        return 'buy';
      }

      if (!availableToReserve && reasonCode === PLAN_DOES_NOT_ALLOW_RESERVATIONS) {
        return 'signup';
      }

      return availableToReserve ? 'reserve' : 'disable';
    case memberStatusType('CANCELLED'):
      return 'reactivate';
    case memberStatusType('UNSUBSCRIBED'):
    case NOT_A_MEMBER:
      return permissions.isAlcEnabled ? 'purchase' : 'join';
    default:
      return 'join';
  }
};
```

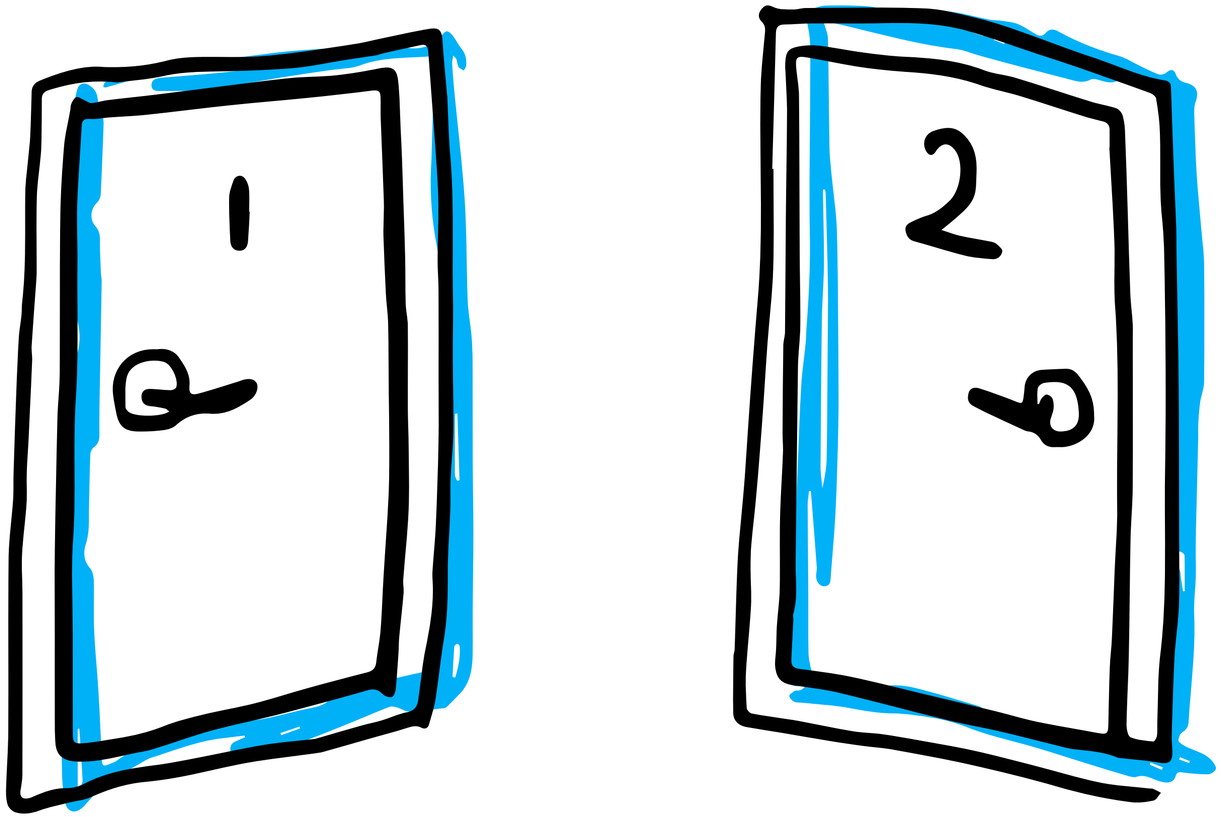
@gurlcode

Centralized logic is easier to reason about

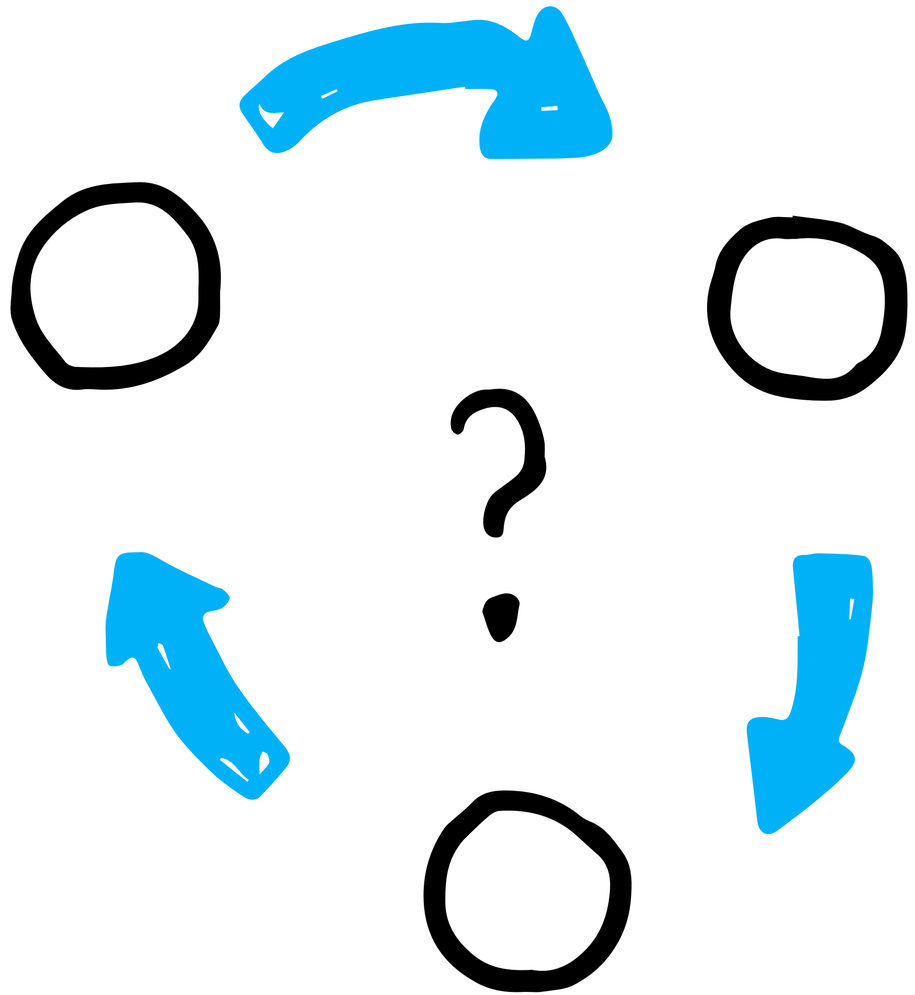
Test the component separate from the logic

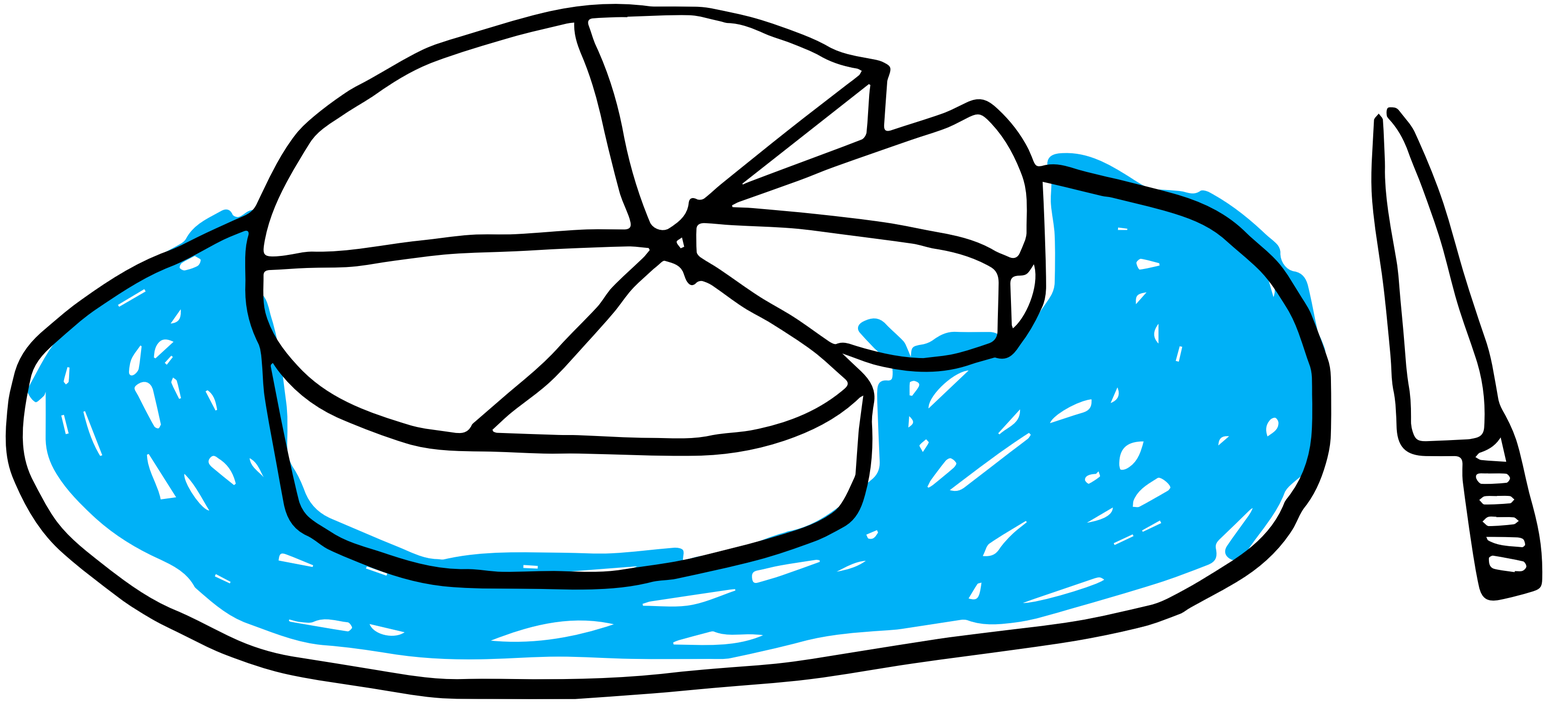


<ScheduleCta type={?} />

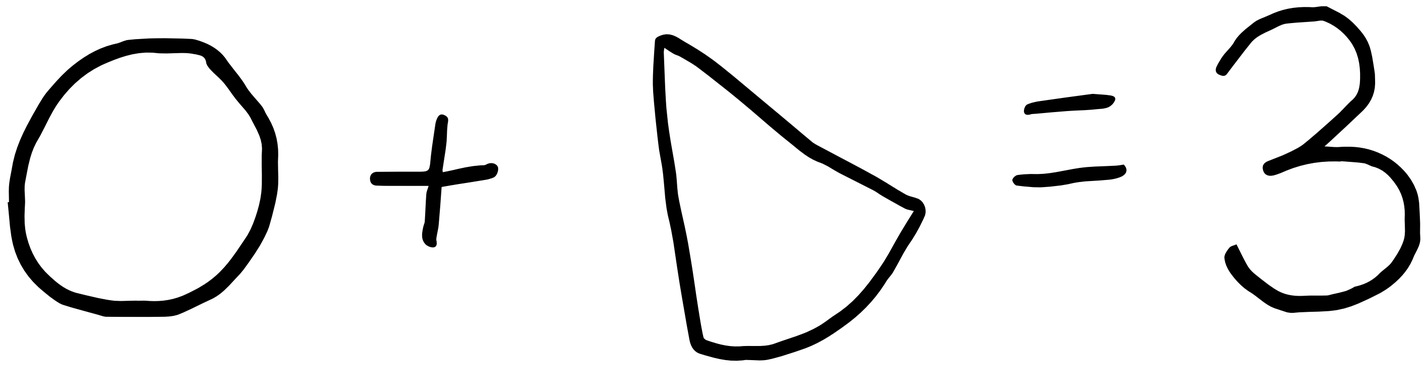


@gurlcode





@gurlcode



@gurlcode

```
const BaseCta = ({
  children,
  className,
  tag = 'button',
  ...otherProps
}) => {
  const Tag = tag;

  return (
    <Tag
      className={classNames('bt', className)}
      {...otherProps}
    >
      {children}
    </Tag>
  );
};
```

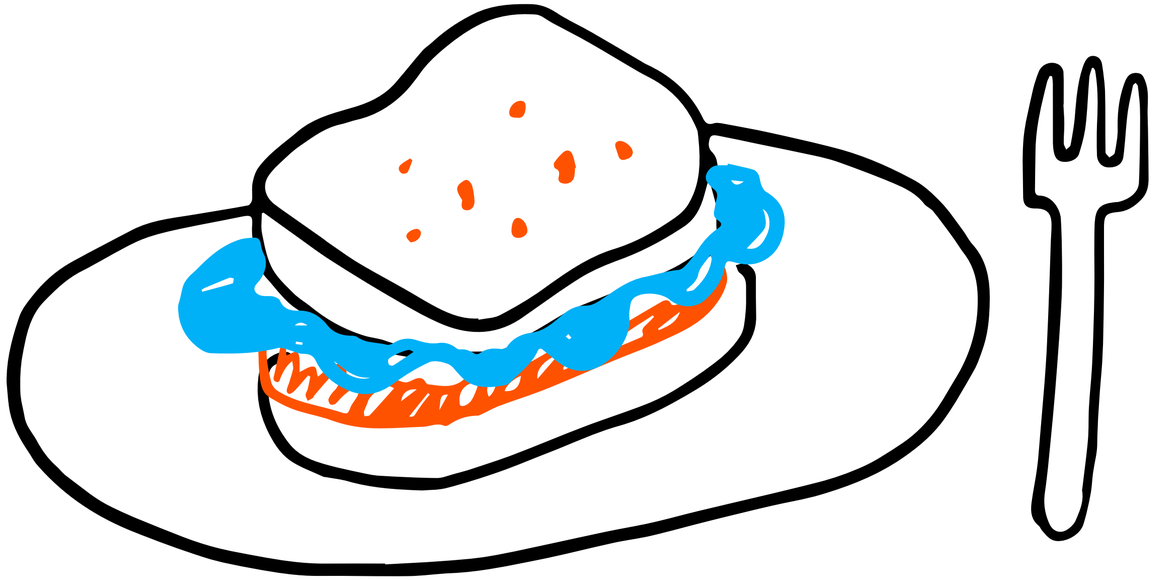
```
const BaseCta = ({
  children,
  className,
  tag = 'button',
  ...otherProps
}) => {
  const Tag = tag;

  return (
    <Tag
      className={classNames('bt', className)}
      {...otherProps}
    >
      {children}
    </Tag>
  );
};
```

```
const BaseCta = ({
  children,
  className,
  tag = 'button',
  ...otherProps
}) => {
  const Tag = tag;

  return (
    <Tag
      className={classNames('bt', className)}
      {...otherProps}
    >
      {children}
    </Tag>
  );
};
```

```
class Bread extends Component {
  // ...methods

  render() {
    return (
      <div className="bread">
        {this.props.children}
      </div>
    );
  }
}
```

@gurlcode

```jsx
import Bread from './bread';

class Burger extends Component {
  // ...methods

  render() {
    return (
      <Bread>
        <Lettuce />
        <Tomato />
        {props.withCheese && <Cheese />}
        <BurgerPatty />
      </Bread>
    );
  }
}
```



@gurlcode

```jsx
const BaseCta = ({
  className,
  tag = 'button',
  message,
  ...otherProps
}) => {
  const Tag = tag;

  return (
    <Tag
      className={classNames('bt', className)}
      {...otherProps}
    >
      {message}
    </Tag>
  );
};
```

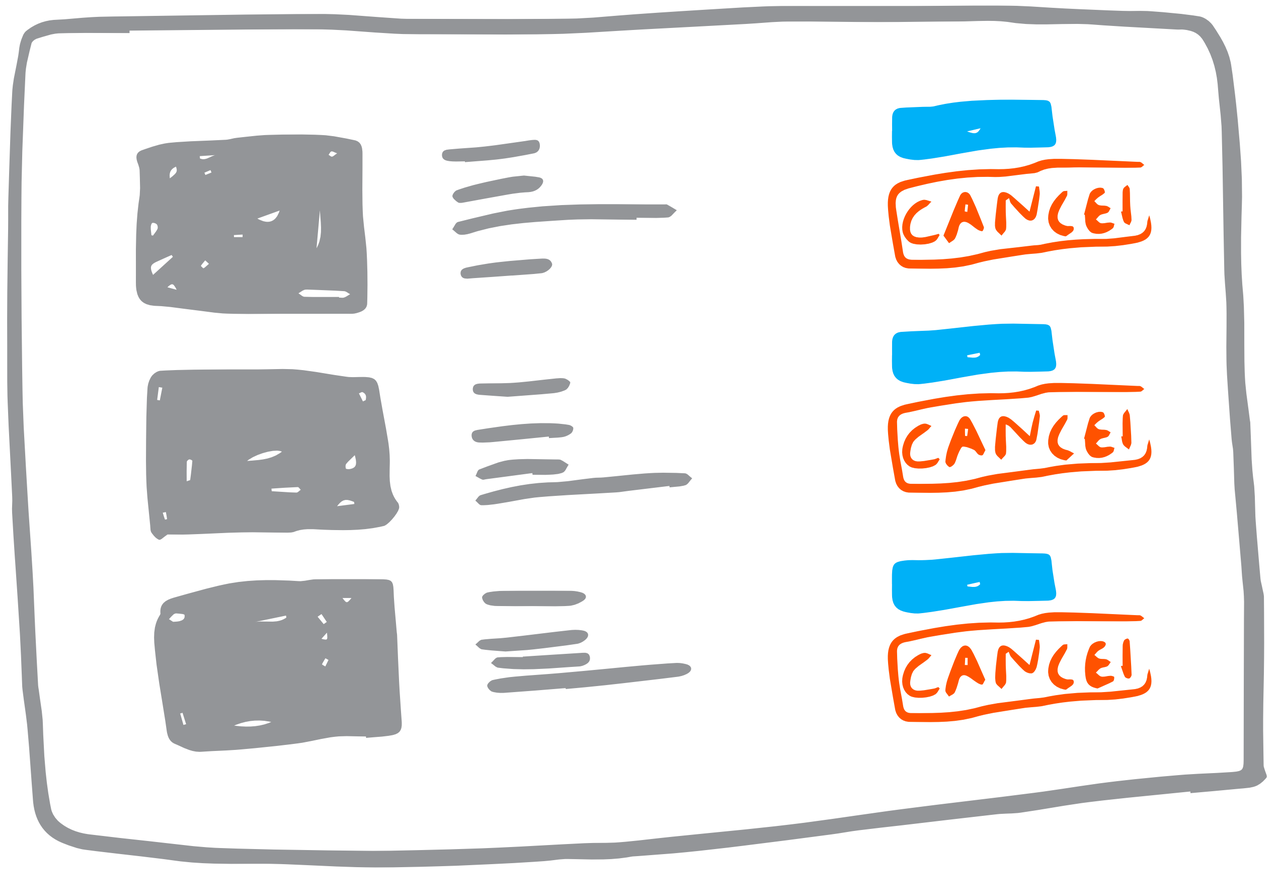
```
const ReserveCta = ({
  className,
  creditPrice,
  onClick,
  tooltip,
}) => (
  <BaseCta
    className={classNames(className, 'bt--primary')}
  >
    <span>{creditPrice}</span>
    <FormattedMessage
      id="schedule.cta.reserve"
      defaultMessage="credits"
    />
  </BaseCta>
);
```

12 credits

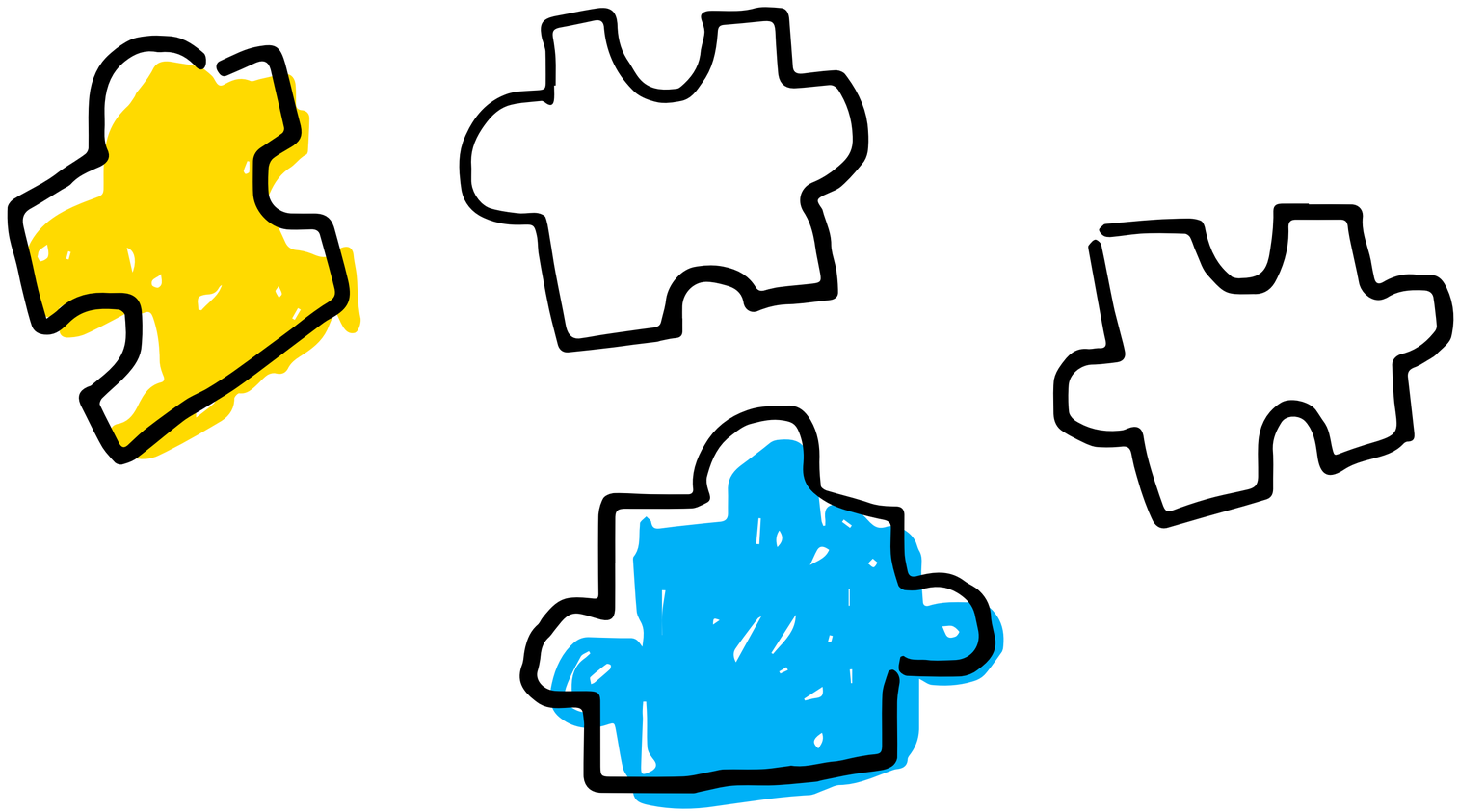
@gurlcode

```
const DisableCta = ({
  className,
  ...otherProps,
}) => (
  <BaseCta
    className='bt--disabled'
    {...otherProps}
    onClick={() => { /* noop */ }}
  >
    <FormattedMessage
      id="schedule.cta.disabled"
      defaultMessage="Reserve"
    />
  </BaseCta>
);
```

@gurlcode



@gurlcode



<ScheduleCta type={?} />

components[type]

```
components = {
    reserve: ReserveCta,
    join: JoinCta,
    cancel: CancelCta,
    disable: DisableCta,
    reactivate: ReactivateCta
}
```

```
const Component = components[type];
```

```
const Component =`${left || right}Sidebar`;
```

```
const Component = props.isCard ? (
   <Card>
     <Content />
   </Card>
) : (<Content />);
```

```
class ScheduleCta extends PureComponent {
  components = {
    buy: BuyCta,
    cancel: CancelCta,
    disable: DisableCta,
    join: JoinCta,
    purchase: PurchaseCta,
    reactivate: ReactivateCta,
    reserve: ReserveCta,
    signup: SignUpCta,
  }

  render() {
    const { type, ...props } = this.props;
    if (!type) {
      return null;
    }

    const Component = this.components[type];
    return (<Component {...props} />);
  }
}
```

@gurlcode

```
class ScheduleCta extends PureComponent {
  components = {
    buy: BuyCta,
    cancel: CancelCta,
    disable: DisableCta,
    join: JoinCta,
    purchase: PurchaseCta,
    reactivate: ReactivateCta,
    reserve: ReserveCta,
    signup: SignUpCta,
  }

  render() {
    const { type, ...props } = this.props;
    if (!type) {
      return null;
    }

    const Component = this.components[type];
    return (<Component {...props} />);
  }
}
```

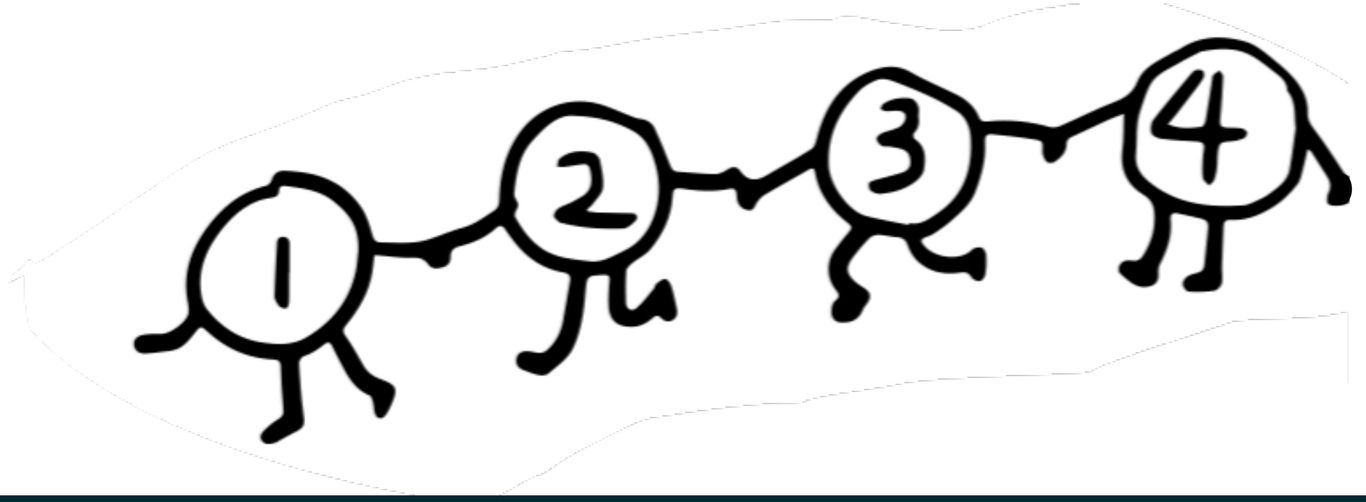
```
class ScheduleCta extends PureComponent {
  components = {
    buy: BuyCta,
    cancel: CancelCta,
    disable: DisableCta,
    join: JoinCta,
    purchase: PurchaseCta,
    reactivate: ReactivateCta,
    reserve: ReserveCta,
    signup: SignUpCta,
  }

  render() {
    const { type, ...props } = this.props;
    if (!type) {
      return null;
    }

    const Component = this.components[type];
    return (<Component {...props} />);
  }
}
```

@gurlcode

```
class ScheduleCta extends PureComponent {
  components = {
    buy: BuyCta,
    cancel: CancelCta,
    disable: DisableCta,
    join: JoinCta,
    purchase: PurchaseCta,
    reactivate: ReactivateCta,
    reserve: ReserveCta,
    signup: SignUpCta,
  }

  render() {
    const { type, ...props } = this.props;
    if (!type) {
      return null;
    }

    const Component = this.components[type];
    return (<Component {...props} />);
  }
}
```

```
class ScheduleCta extends PureComponent {
  components = {
    buy: BuyCta,
    cancel: CancelCta,
    disable: DisableCta,
    join: JoinCta,
    purchase: PurchaseCta,
    reactivate: ReactivateCta,
    reserve: ReserveCta,
    signup: SignUpCta,
  }

  render() {
    const { type, ...props } = this.props;
    if (!type) {
      return null;
    }

    const Component = this.components[type];
    return (<Component {...props} />);
  }
}
```

```
<ScheduleCta
   type={?}
   className=“”
/>
```




```
<ScheduleCta
  type={type}
  className="bt-small"
  schedule={schedule}
  tooltip={!availableToReserve && reason}
/>
```

@gurlcode

Use whatever logic we want

Use individual CTAs alone

Add & remove CTAs with ease

CTAs added and removed for A/B tests

Removing Disable CTA on some pages

Classnames changed

Messaging changed for ReserveCta

Ask for specific CTA on specific page

Hmm...

```
const getScheduleAction = (memberStatus, schedule, permissions = {}) => {
  const {
    availableToPurchase,
    availableToReserve,
    reasonCode,
  } = ScheduleDeriver.availabilityData(schedule);

  const NOT_A_MEMBER = 'NOT_A_MEMBER';

  switch (memberStatus || NOT_A_MEMBER) {
    case memberStatusType('UNPAID'):
    case memberStatusType('ACTIVE'):
      if (reasonCode === RESERVED) {
        return 'cancel';
      }

      if (!availableToReserve && availableToPurchase) {
        return 'buy';
      }

      if (!availableToReserve && reasonCode === PLAN_DOES_NOT_ALLOW_RESERVATIONS) {
        return 'signup';
      }

      return availableToReserve ? 'reserve' : 'disable';
    case memberStatusType('CANCELLED'):
      return 'reactivate';
```

@gurlcode

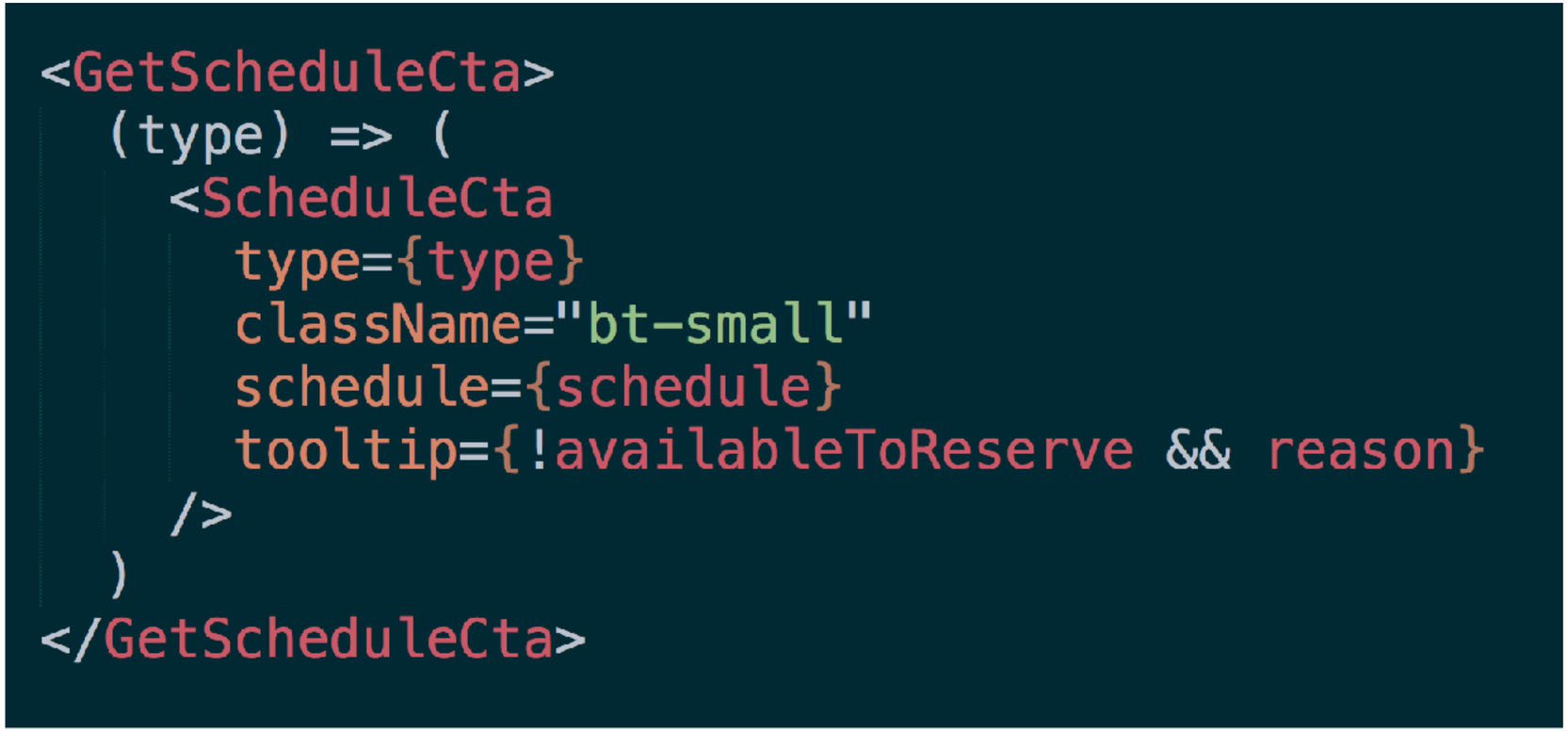
Render Props*

*children as a function

... so what if children were a function?

```
<GetScheduleCta>
  (type) => (
    <ScheduleCta
      type={type}
      className="bt-small"
      schedule={schedule}
      tooltip={!availableToReserve && reason}
    />
  )
</GetScheduleCta>
```

@gurlcode


<GetScheduleCta>
(type) => (
<ScheduleCta
type={type}
className="bt-small"
schedule={schedule}
tooltip={!availableToReserve && reason}
/>
)
</GetScheduleCta>

@gurlcode

```
<GetScheduleCta>
  (type) => (
    <ScheduleCta
      type={type}
      className="bt-small"
      schedule={schedule}
      tooltip={!availableToReserve && reason}
    />
  )
</GetScheduleCta>
```

@gurlcode

```
<GetScheduleCta>
  (type) => (
    <ScheduleCta
      type={type}
      className="bt-small"
      schedule={schedule}
      tooltip={!availableToReserve && reason}
    />
  )
</GetScheduleCta>
```

@gurlcode

```
<GetScheduleCta>
  (type) => (
    <ScheduleCta
      type={type}
      className="bt-small"
      schedule={schedule}
      tooltip={!availableToReserve && reason}
    />
  )
</GetScheduleCta>
```

@gurlcode

```
<GetScheduleCta>
  (type) => (
    <ScheduleCta
      type={type}
      [redacted]bt-small"
      schedule={schedule}
      tooltip={!availableToReserve && reason}
    />
  )
</GetScheduleCta>
```

```
class GetScheduleCta extends Component {
  render() {
    const type = getScheduleType();
    this.props.children(type);
  }
}
```

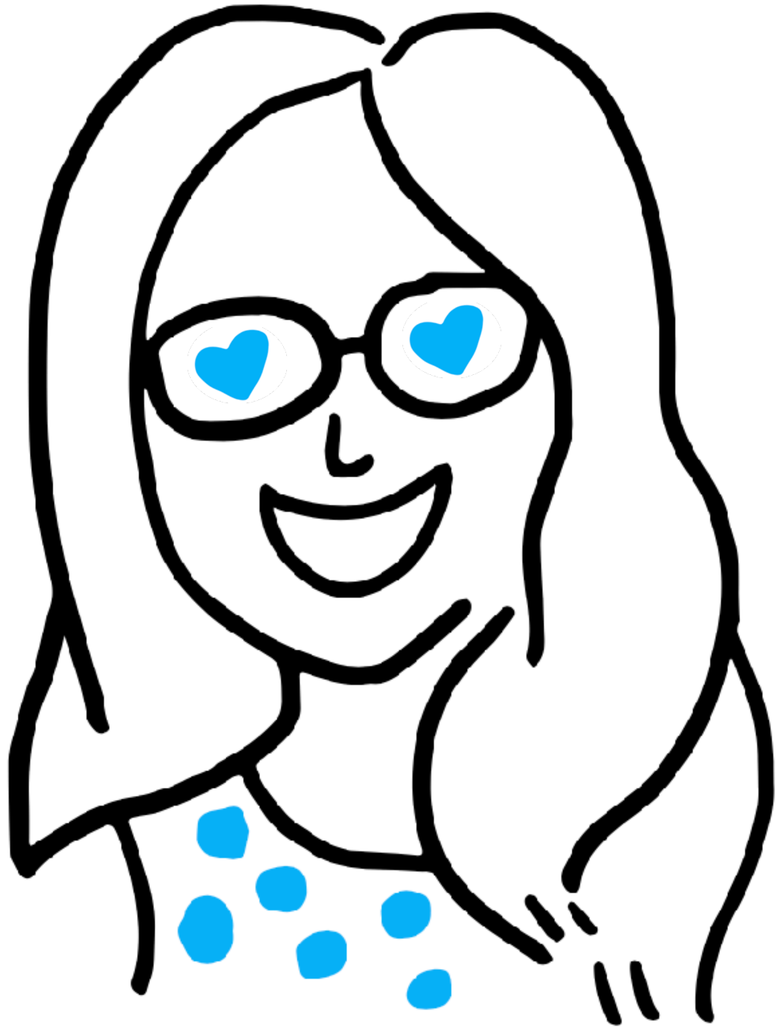
```
class GetScheduleCta extends Component {
  render() {
    const type = getScheduleType();
    this.props.children(type);
  }
}
```

Render Props === state delegate

children function === UI handler

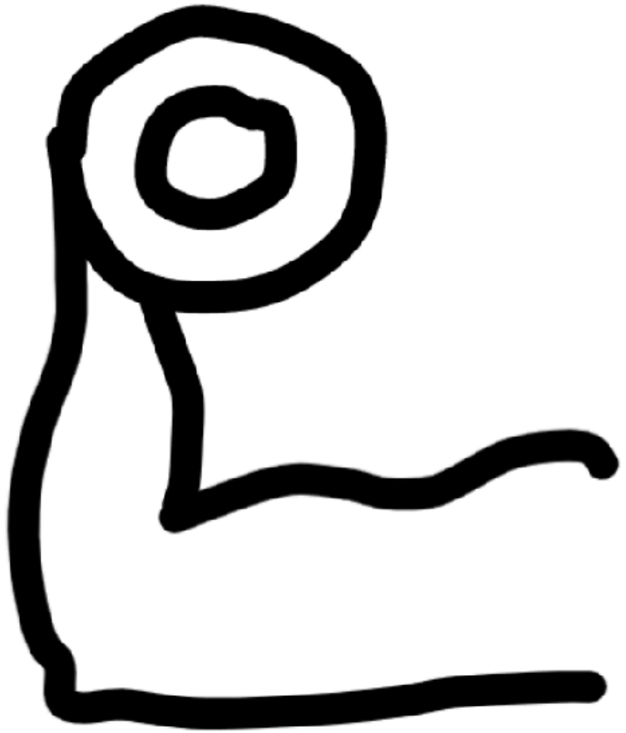
https://reactjs.org/docs/render-props.html

https://github.com/paypal/downshift



@gurlcode

you === flexible



Know your traps.

Kill your darlings.

Start with the ideal API

Let the API inform the component design

Decide what level of magic you want

Business logic does not belong in components

Render Props is awesome (use it!)

Be flexible as an engineer

(know your traps, kill your darlings)

@gurlcode

Thank you!

Content by Jenn Creighton

Illustrations by Janny Ji (jannyji.com)

@gurlcode