Advanced Component Patterns

siddharthkp

bundlesize

Keep your bundle size in check

JavaScript ★ 2.3k 100

cost-of-modules

Find out which of your dependencies are slowing you down 🐢

JavaScript ★ 2k 45

auto-install

Install dependencies as you code ⚡

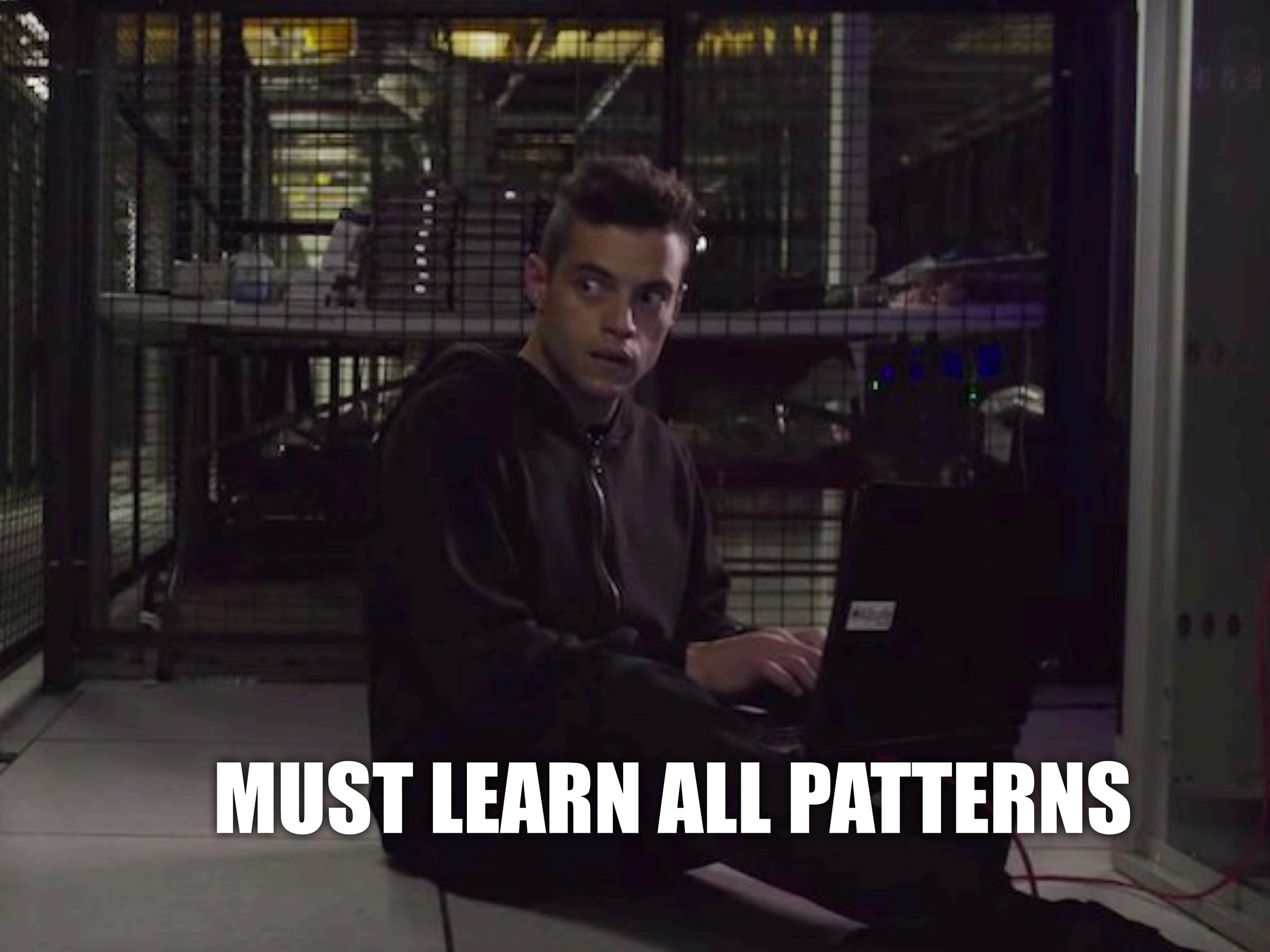
JavaScript ★ 787 35

notella

No fluff notes app

JavaScript ★ 183 20

I teach React


MUST LEARN ALL PATTERNS

ADVANCED COMPONENT PATTERNS

7 component patterns

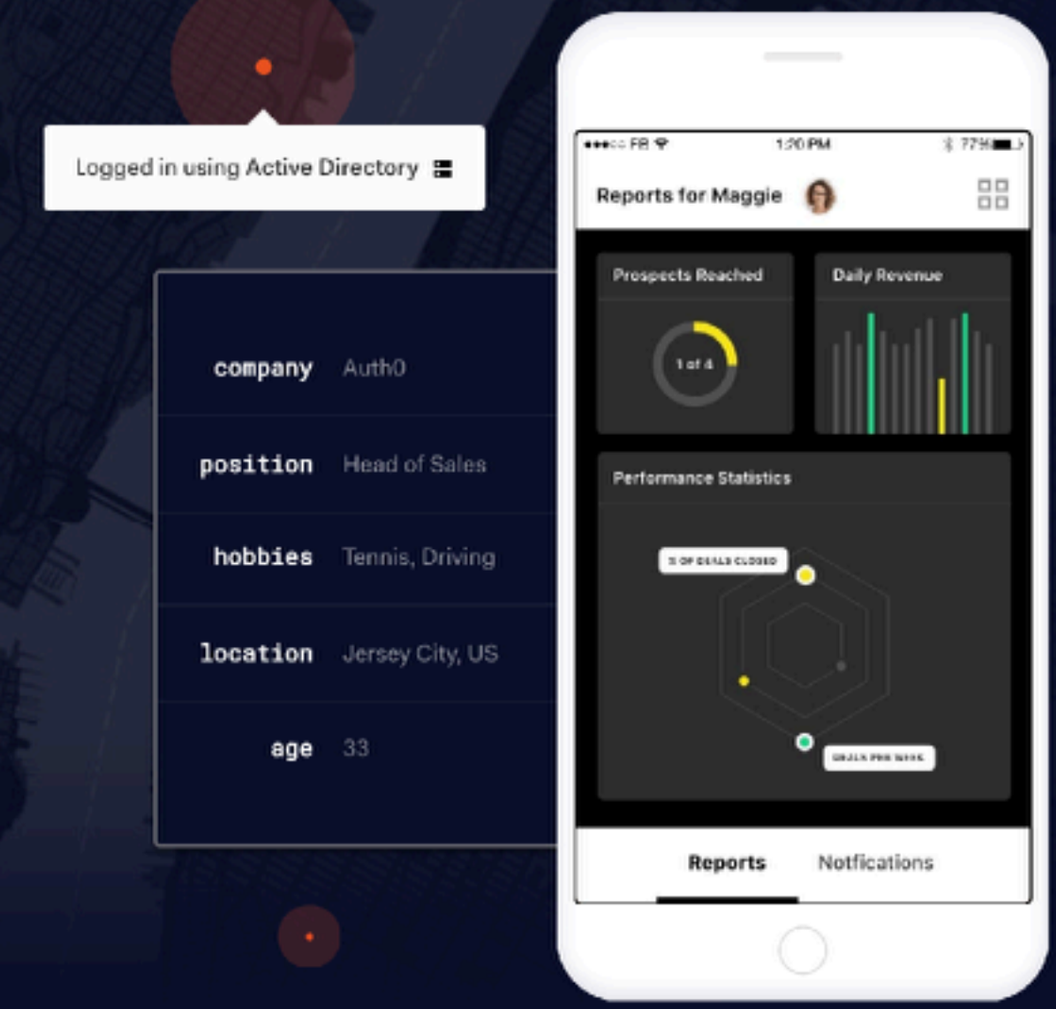
The new way to solve Identity

We solve the most complex identity use cases with an extensible and easy to integrate platform that secures billions of logins every month

USE AUTH0 FOR FREE

WATCH VIDEO

No credit card required



A Powerful Platform for Developers

DESIGN

DEVELOPMENT

Horizontal (Default)

Name API Explorer Client

Description Add a description in less than 140 characters

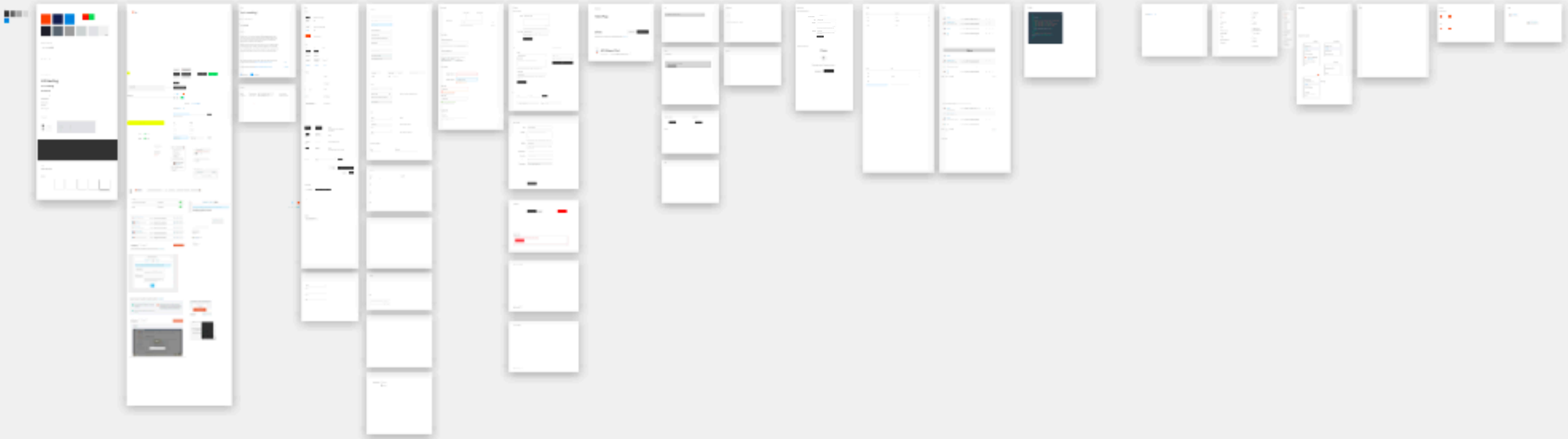
1 / 140

A free text description of the client. Max character count is 140.

Client Type Non Interactive

The type of client will determine which settings you can configure from the dashboard.

SAVE CHANGES




IF YOU CAN SOME EMPATHY TO THE BUTTON
THAT WOULD BE GREAT
imgflip.com

Enter some text


```
const TextInput = props => {
   return (
      <input
         className="input"
         type="text"
         placeholder={props.placeholder}
      />
   )
}

render(
   <TextInput placeholder="Enter some text" />
)
```

This input is readonly

Enter some text


```
const TextInput = props => {
   return (
      <input
         className="input"
         type="text"
         placeholder={props.placeholder}
      />
   )
}

render(
   <TextInput placeholder="Enter some text" />
)
```


```
const TextInput = props => {
   return (
      <input
         className="input"
         type="text"
         placeholder={props.placeholder}
      />
   )
}

render(
   <TextInput readOnly placeholder="Enter some text" />
)
```


```
const TextInput = props => {
   let classes = 'input'
   if (props.readOnly) classes += ' readonly'
   return (
      <input
         className={classes}
         type="text"
         placeholder={props.placeholder}
      />
   )
}

render(
   <TextInput readOnly placeholder="Enter some text" />
)
```


```
const TextInput = props => {
   let classes = 'input'
   if (props.readOnly) classes += ' readonly'
   return (
      <input
         className={classes}
         type="text"
         placeholder={props.placeholder}
         readOnly={props.readOnly}
      />
   )
}

render(
   <TextInput readOnly placeholder="Enter some text" />
)
```

Functional component


```
const TextInput = props => {
   let classes = 'input'
   if (props.readOnly) classes += ' readonly'
   return (
      <input
         className={classes}
         type="text"
         placeholder={props.placeholder}
         readOnly={props.readOnly}
      />
   )
}

render(
   <TextInput readOnly placeholder="Enter some text" />
)
```

This input is readonly

Enter some text

0 chars

Enter some text

This input is readonly

Enter some text

0 chars

Enter some text

```
const TextInput = props => {
   return (
      <input
         className="input"
         type="text"
         placeholder={props.placeholder}
      />
   )
}
```

0 chars

Enter some text

```
const TextInput = props => {
   return (
      <div>
         <span>0 chars</span>
         <input
            className="input"
            type="text"
            placeholder={props.placeholder}
         />
      </div>
   )
}
```


```
class TextInput extends React.Component {
   render() {
      return (
         <div>
            <span>0 chars</span>
            <input
               className="input"
               type="text"
               placeholder={props.placeholder}
            />
         </div>
      )
   }
}
```


```
class TextInput extends React.Component {
   render() {
      return (
         <div>
            <span>0 chars</span>
            <input
               className="input"
               type="text"
               placeholder={this.props.placeholder}
            />
         </div>
      )
   }
}
```

0 chars

Enter some text

```
class TextInput extends React.Component {
   constructor(props) {
      super(props)
      this.state = { length: 0 }
   }
   render() {
      return (
         <div>
            <span>0 chars</span>
            <input
               className="input"
               type="text"
               placeholder={this.props.placeholder}
            />
         </div>
      )
   }
}
```


```
class TextInput extends React.Component {
   constructor(props) {
      super(props)
      this.state = { length: 0 }
   }
   render() {
      return (
         <div>
            <span>{this.state.length} chars</span>
            <input
               className="input"
               type="text"
               placeholder={this.props.placeholder}
            />
         </div>
      )
   }
}
```


```
class TextInput extends React.Component {
   constructor(props) { ... }

   render() {
      return (
         <div>
            <span>{this.state.length} chars</span>
            <input
               className="input"
               type="text"
               placeholder={this.props.placeholder}
            />
         </div>
      )
   }
}
```


```
class TextInput extends React.Component {
   constructor(props) { ... }
   onChange(event) {
      this.setState({ length: event.target.value.length })
   }
   render() {
      return (
         <div>
            <span>{this.state.length} chars</span>
            <input
               className="input"
               type="text"
               placeholder={this.props.placeholder}
            />
         </div>
      )
   }
}
```

3 chars

Abc

```
class TextInput extends React.Component {
   constructor(props) { ... }
   onChange(event) {
      this.setState({ length: event.target.value.length })
   }
   render() {
      return (
         <div>
            <span>{this.state.length} chars</span>
            <input
               className="input"
               type="text"
               placeholder={this.props.placeholder
              onChange={this.onChange.bind(this)}
            />
         </div>
      )
   }
```

Class component

3 chars

Abc

```
class TextInput extends React.Component {
   constructor(props) { ... }
   onChange(event) {
      this.setState({ length: event.target.value.length })
   }
   render() {
      return (
         <div>
            <span>{this.state.length} chars</span>
            <input
               className="input"
               type="text"
               placeholder={this.props.placeholder
               onChange={this.onChange.bind(this)}
            />
         </div>
      )
   }
```

3 chars

Abc

```
class TextInput extends React.Component {
   constructor(props) { ... }
   onChange(event) {
      this.setState({ length: event.target.value.length })
   }
   render() {
      return (
         <div>
            <span>{this.state.length} chars</span>
            <input
               className="input"
               type="text"
               placeholder={this.props.placeholder
              onChange={this.onChange.bind(this)}
            />
         </div>
      )
   }
```

3 chars

Abc

```
class TextInput extends React.Component {
   constructor(props) { ... }
   onChange(event) {
      this.setState({ length: event.target.value.length })
   }
   render() {
      return (
         <div>
            <span>{this.state.length} chars</span>
            <input
               className="input"
               type="text"
               placeholder={this.props.placeholder
              onChange={this.onChange}
            />
         </div>
      )
```

3 chars

Abc

```
class TextInput extends React.Component {
   constructor(props) { ... }
   onChange = (event) => {
      this.setState({ length: event.target.value.length })
   }
   render() {
      return (
         <div>
            <span>{this.state.length} chars</span>
            <input
               className="input"
               type="text"
               placeholder={this.props.placeholder
              onChange={this.onChange}
            />
         </div>
      )
```

3 chars

Abc

```
class TextInput extends React.Component {
   constructor(props) { ... }

   onChange = (event) => { ... }
   render() {
      return (
         <div>
            <span>{this.state.length} chars</span>
            <input
               className="input"
               type="text"
               placeholder={this.props.placeholder
              onChange={this.onChange}
            />
         </div>
      )
```

3 chars

Abc

```
class TextInput extends React.Component {
   constructor(props) {
      super(props)
      this.state = { length: 0 }
   }
   onChange = (event) => { ... }
   render() {
      return (
         <div>
            <span>{this.state.length} chars</span>
            <input
               className="input"
               type="text"
               placeholder={this.props.placeholder
              onChange={this.onChange}
            />
         </div>
```

3 chars

Abc

```
class TextInput extends React.Component {

  state = { length: 0 }

  onChange = (event) => { ... }
  render() {
    return (
      <div>
        <span>{this.state.length} chars</span>
        <input
          className="input"
          type="text"
          placeholder={this.props.placeholder
          onChange={this.onChange}
        />
      </div>
```

3 chars

Abc

```
class TextInput extends React.Component {
  state = { length: 0 }
   onChange = (event) => { ... }
   render() {
      return (
         <div>
            <span>{this.state.length} chars</span>
            <input
               className="input"
               type="text"
               placeholder={this.props.placeholder
              onChange={this.onChange}
            />
         </div>
      )
   }
}
```

BABEL

Class properties transform

Installation

Shell Copy

```
npm install --save-dev babel-plugin-transform-class-properties
```

Usage

.babelrc

JSON Copy

```
{
  "plugins": ["transform-class-properties"]
}
```

3 chars

Abc

```
class TextInput extends React.Component {
  state = { length: 0 }
   onChange = (event) => { ... }
   render() {
      return (
         <div>
            <span>{this.state.length} chars</span>
            <input
               className="input"
               type="text"
               placeholder={this.props.placeholder
              onChange={this.onChange}
            />
         </div>
      )
   }
}
```

email siddharth@gmail.com

0 chars Enter some text

This input is readonly

Enter some text

email


```
class TextInput extends React.Component {
   render() {
      return (
         <input
            type="text"
            placeholder={this.props.placeholder}
         />
      )
   }
}
```

Uncontrolled component

email SIDDHARTH

```
class TextInput extends React.Component {
   render() {
      return (
         <input
            type="text"
            placeholder={this.props.placeholder}
         />
      )
   }
}
```

email

Enter some text

```
class TextInput extends React.Component {
   state = { value: '' }
   render() {
      return (
         <input
            value={this.state.value}
            type="text"
            placeholder={this.props.placeholder}
         />
      )
   }
}
```

email s

```
class TextInput extends React.Component {
   state = { value: '' }
   render() {
      return (
         <input
            value={this.state.value}
            type="text"
            placeholder={this.props.placeholder}
         />
      )
   }
}
```

email s

```
class TextInput extends React.Component {
   state = { value: '' }
   render() {
      return (
         <input
            value={this.state.value}
            type="text"
            placeholder={this.props.placeholder}
         />
      )
   }
}
```

email


```
class TextInput extends React.Component {
   state = { value: '' }
   onChange = (event) => {
      this.setState({ value: event.target.value.toLowerCase() })
   }
   render() {
      return (
         <input
            onChange={this.onChange}
            value={this.state.value}
            type="text"
            placeholder={this.props.placeholder}
         />
      )
   }
}
```

email SIDDHARTH@Gmail.com

```
class TextInput extends React.Component {
   state = { value: '' }
   onChange = (event) => {
      this.setState({ value: event.target.value.toLowerCase() })
   }
   render() {
      return (
         <input
            onChange={this.onChange}
            value={this.state.value}
            type="text"
            placeholder={this.props.placeholder}
         />
      )
   }
}
```

address

Enter some text

email

siddharth@gmail.com

0 chars

Enter some text

This input is readonly

Enter some text

address


```
render(
   <TextArea placeholder="Enter some text" />
)
```

address


```
render(
   withLabel('address', <TextArea placeholder="Enter some text" />)
)
```

address


```
const withLabel = (label, Component) => {

}

render(
   withLabel(‘address', <TextArea placeholder="Enter some text" />)
)
```

address

Enter some text

```
const withLabel = (label, Component) => {
   return (
      <div className="form-field">
         <label>{label}</label>
         <Component/>
      </div>
   )
}

render(
   withLabel('address', <TextArea placeholder="Enter some text" />)
)
```

Higher Order Component

address

Enter some text

```
const withLabel = (label, Component) => {
   return (
      <div className="form-field">
         <label>{label}</label>
         <Component/>
      </div>
   )
}

render(
   withLabel('address', <TextArea placeholder="Enter some text" />)
)
```

address

Enter some text

```
const withLabel = (label, Component) => {
   return (
      <div className="form-field">
         <label>{label}</label>
         <Component/>
      </div>
   )
}

render(
   <WithLabel label="address">
      <TextArea placeholder="Enter some text" />
   </WithLabel>
)
```

address

Enter some text

```
const WithLabel = (props) => {
   return (
      <div className="form-field">
         <label>{label}</label>
         <Component/>
      </div>
   )
}

render(
   <WithLabel label="address">
      <TextArea placeholder="Enter some text" />
   </WithLabel>
)
```

address

Enter some text

```
const withLabel = (props) => {
   return (
      <div className="form-field">
         <label>{label}</label>
         {props.children}
      </div>
   )
}

render(
   <WithLabel label="address">
      <TextArea placeholder="Enter some text" />
   </WithLabel>
)
```

email Enter some text

address Enter some text

accept? OFF

SAVE CHANGES CLEAR

email


```
render(
   <form>
   </form>
)
```

email

Enter some text

```
render(
   <form>
      <WithLabel label="address">
         <TextInput placeholder="Enter some text" />
      </WithLabel>
   </form>
)
```

email

Enter some text

```
render(
   <form>
      <WithLabel label="address">
         <TextInput placeholder="Enter some text" />
      </WithLabel>
      <WithLabel label="address">
         <TextArea placeholder="Enter some text" />
      </WithLabel>
      <WithLabel label="accept?">
         <Switch />
      </WithLabel>
   </form>
)
```

email

Enter some text

```
render(
   <form>
      <WithLabel label="address">
         <TextInput placeholder="Enter some text" />
      </WithLabel>
      <WithLabel label="address">
         <TextArea placeholder="Enter some text" />
      </WithLabel>
      <WithLabel label="accept?">
         <Switch />
      </WithLabel>
      <div className="actions">
         <Button primary onClick={this.save}>Save</Button>
         <Button onClick={this.clear}>Clear</Button>
      </div>
   </form>
)
```

email Enter some text

address Enter some text

accept? OFF

SAVE CHANGES CLEAR

email

Enter some text

```
render(
   <form>
      <WithLabel label="address">
         <TextInput placeholder="Enter some text" />
      </WithLabel>
      <WithLabel label="address">
         <Form.TextArea placeholder="Enter some text" />
      </WithLabel>
      <WithLabel label=“accept?">
         <Form.Switch />
      </WithLabel>
      <div className="actions">
         <Button primary onClick={this.save}>Save</Button>
         <Button onClick={this.clear}>Clear</Button>
      </div>
   </form>
)
```


```
render(
   <Form>
      <Form.TextInput label="email" placeholder="Enter some text" />
      <Form.TextArea label="address" placeholder="Enter some text" />
      <Form.Switch label="accept?" />
      <Form.Actions
         primary={{ label: 'Save Changes', method: this.save }}
         secondary={{ label: 'Save Changes', method: this.save }}
      />
   </Form>
)
```

Compound Component❤️

compo
compo


```
render(
   <Form>
      <Form.TextInput label="email" placeholder="Enter some text" />
      <Form.TextArea label="address" placeholder="Enter some text" />
      <Form.Switch label="accept?" />
      <Form.Actions
         primary={{ label: 'Save Changes', method: this.save }}
         secondary={{ label: 'Save Changes', method: this.save }}
      />
   </Form>
)
```

email

Enter some text

```
<Form>
  <Form.TextInput label="email" placeholder="Enter some text" />

  <Form.TextArea label="address" placeholder="Enter some text" />
```

email

Enter some text

```
const Form = (props) => {
   return (<form>{ props.children }</form>)
}
```

```
<Form.TextInput label="email" placeholder="Enter some text" />
```

```
<Form.TextArea label="address" placeholder="Enter some text" />
```

email

Enter some text

```
const Form = (props) => {
   return (<form>{ props.children }</form>)
}
```

```
Form.TextInput = props => {
   return (
      <WithLabel label={props.label}>
         <TextInput placeholder={props.placeholder} />
      </WithLabel>
   )
}

   <Form.TextArea label="address" placeholder="Enter some text" />
```

email

Enter some text

```
const Form = (props) => {
   return (<form>{ props.children }</form>)
}

Form.TextInput = props => {
   return (
      <WithLabel label={props.label}>
         <TextInput placeholder={props.placeholder} />
      </WithLabel>
   )
}

Form.TextArea = props => {
   return (
      <WithLabel label={props.label}>
         <TextArea placeholder={props.placeholder} />
      </WithLabel>
```

email

Enter some text

```
render(
   <Form>
      <Form.TextInput label="email" placeholder="Enter some text" />
      <Form.TextArea label="address" placeholder="Enter some text" />
      <Form.Switch label="accept?" />
      <Form.Actions
         primary={{ label: 'Save Changes', method: this.save }}
         secondary={{ label: 'Save Changes', method: this.save }}
      />
   </Form>
)
```

email Enter some text

address Enter some text

accept? OFF

SAVE CHANGES CLEAR

email

Enter some text

```
render(
   <Form>
      <Form.TextInput label="email" placeholder="Enter some text" />
      <Form.TextArea label="address" placeholder="Enter some text" />
      <Form.Switch label="accept?" />
      <Form.Actions
         primary={{ label: 'Save Changes', method: this.save }}
         secondary={{ label: 'Save Changes', method: this.save }}
      />
   </Form>
)
```

email Enter some text

address Enter some text

accept? OFF

SAVE CHANGES CLEAR

email

Enter some text

address

Enter some text

accept?

OFF

SAVE CHANGES

CLEAR

0 chars

Enter some text

PROJECT NAME

Abc really long name, omg goes on and on

34 chars

PROJECT NAME

Abc really long name, omg goes on and on 34 chars

```
class TextInput extends React.Component {
   constructor(props) { ... }
   onChange(event) {
      this.setState({ length: event.target.value.length })
   }
   render() {
      return (
         <div>
            <input onChange={this.onChange.bind(this) />
            <span>{this.state.length} chars</span>
         </div>
      )
   }
}
```

PROJECT NAME

Abc really long name, omg goes on and on 34 chars

```
class TextInput extends React.Component {
   constructor(props) { ... }
   onChange(event) {
      const error = event.target.value.length > 30
      this.setState({ length: event.target.value.length, error })
   }
   render() {
      return (
         <div>
            <input onChange={this.onChange.bind(this) />
            <span className={...}>{this.state.length} chars</span>
         </div>
      )
   }
}
```

PROJECT NAME

Abc really long name, omg goes on and on 34 chars

```
class TextInput extends React.Component {
   constructor(props) { ... }
   onChange(event) {
      const error = event.target.value.length > this.props.limit
      this.setState({ length: event.target.value.length, error })
   }
   render() {
      return (
         <div>
            <input onChange={this.onChange.bind(this) />
            <span className={...}>{this.state.length} chars</span>
         </div>
      )
   }
}
```

PROJECT NAME

Abc really long name, omg goes on and on 34 chars

```
class TextInput extends React.Component {
   ...
}

render(<TextInput limit={30} />)
```

PROJECT NAME

Abc really long name, omg goes on and on

34 chars

Abc really long name, omg goes on

28 chars

PROJECT NAME

Abc really long name, omg goes on and on 34 chars

```
class TextInput extends React.Component {
   ...
}

render(<TextInput limit={30} />)
```

PROJECT NAME

Abc really long name, omg goes on and on 34 chars

```
class TextInput extends React.Component {
   ...
}

render(<TextInput renderLabel={ renderLabel } />)
```

PROJECT NAME

Abc really long name, omg goes on and on 34 chars

```
class TextInput extends React.Component {
   ...
}

render(<TextInput renderLabel={ renderLabel } />)
```

```
const renderLabel = (length) => {
  return <span> {length} chars </span>
}
```

Render prop

PROJECT NAME

Abc really long name, omg goes on and on 34 chars

```
class TextInput extends React.Component {
   ...
}

render(<TextInput renderLabel={ renderLabel } />)
```

```
const renderLabel = (length) => {
  return <span> {length} chars </span>
}
```

PROJECT NAME

Abc really long name, omg goes on and on 34 chars

```
class TextInput extends React.Component {
   ...
}

render(<TextInput renderLabel={ renderLabel } />)
```

```
const renderLabel = (length) => {
  let className
  if (length > 25) className = 'warn'
  else if (length > 30) className = 'error'

  return <span className={className}> {length} chars </span>
}
```

PROJECT NAME

Abc really long name, omg goes on and on 34 chars

```
class TextInput extends React.Component {
   constructor(props) { ... }
   onChange(event) { ... }
   render() {
      return (
         <div>
            <input onChange={this.onChange.bind(this) />
            <span>{this.state.length} chars</span>
         </div>
      )
   }
}
```

PROJECT NAME

Abc really long name, omg goes on and on 34 chars

```
class TextInput extends React.Component {
   constructor(props) { ... }
   onChange(event) { ... }
   render() {
      return (
         <div>
            <input onChange={this.onChange.bind(this) />
            { this.props.renderLabel() }
         </div>
      )
   }
}
```

PROJECT NAME

Abc really long name, omg goes on and on 34 chars

```
class TextInput extends React.Component {
   constructor(props) { ... }
   onChange(event) { ... }
   render() {
      return (
         <div>
            <input onChange={this.onChange.bind(this) />
            { this.props.renderLabel(this.state.length) }
         </div>
      )
   }
}
```

PROJECT NAME

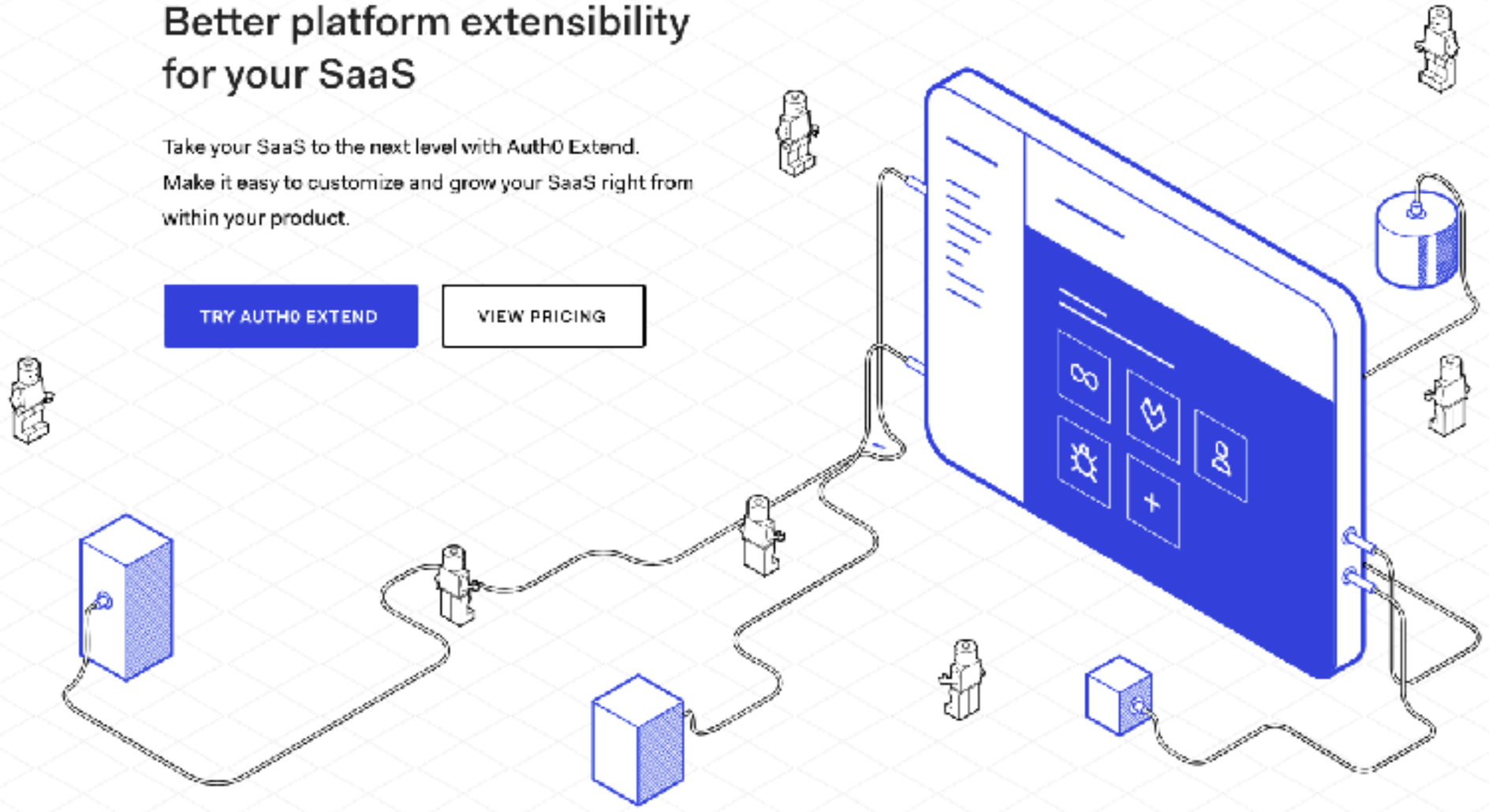
Abc really long name, omg goes on and on 34 chars

```
class TextInput extends React.Component {
   constructor(props) { ... }
   onChange(event) { ... }
   render() {
      return (
         this.props.render(this.state.length)
      )
   }
}
```

Better platform extensibility for your SaaS

Take your SaaS to the next level with Auth0 Extend.
Make it easy to customize and grow your SaaS right from within your product.

TRY AUTH0 EXTEND VIEW PRICING



Power up your SaaS

Auth0 Extend lets your users create extensions right within your SaaS. You get an embedded customizable code editor in your app and a cloud-based service for running extensions on demand, securely, and at scale.

Watch this short video to discover how Extend can help you:

email

Enter something

address

Enter something

accept?

ON

SAVE CHANGES

CLEAR

email

Enter something

address

Enter something

accept?

ON

SAVE CHANGES

CLEAR

email Enter something

address Enter something

accept? ON

SAVE CHANGES CLEAR

email Enter something

address Enter something

accept? ON

SAVE CHANGES CLEAR

email

Enter something

address

Enter something

accept?

ON

SAVE CHANGES

CLEAR

```
<Form>
   <Form.TextInput label="email" placeholder="Enter some text" />
   <Form.TextArea label="address" placeholder="Enter some text" />
   <Form.Switch label=“accept?" theme="manage" />
   <Button theme="manage">Save Changes</Button>
</Form>
```

email

Enter something

address

Enter something

accept?

ON

SAVE CHANGES

CLEAR

```
<Form>
   <Form.TextInput label="email" placeholder="Enter some text" />
   <Form.TextArea label="address" placeholder="Enter some text" />
   <Form.Switch label=“accept?" theme="extend" />
   <Button theme="manage">Save Changes</Button>
</Form>
```

email

Enter something

address

Enter something

accept?

ON

SAVE CHANGES

CLEAR

```
<Form>
   <Form.TextInput label="email" placeholder="Enter some text" />
   <Form.TextArea label="address" placeholder="Enter some text" />
   <Form.Switch label=“accept?" theme="extend" />
   <Button theme="manage">Save Changes</Button>
</Form>
```

email

Enter something

address

Enter something

accept?

ON

SAVE CHANGES

CLEAR

```
<Form>
   <Form.TextInput label="email" placeholder="Enter some text" />
   <Form.TextArea label="address" placeholder="Enter some text" />
   <Form.Switch label=“accept?" theme="extend" />
   <Button theme="extend">Save Changes</Button>
</Form>
```

email

Enter something

address

Enter something

accept?

ON

SAVE CHANGES

CLEAR

```
render(
    <App />
)
```

email

Enter something

address

Enter something

accept?

ON

SAVE CHANGES

CLEAR

```
render(
  <ThemeProvider name="manage">
    <App />
  </ThemeProvider>
)
```

email

Enter something

address

Enter something

accept?

ON

SAVE CHANGES

CLEAR

```
render(
  <ThemeProvider name="extend">
    <App />
  </ThemeProvider>
)
```

Provider Component

Why Not To Use Context

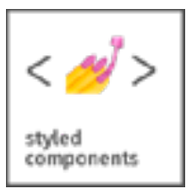
✅ The vast majority of applications do not need to use context.

✅ If you aren't familiar with state management libraries like Redux or MobX, don't use context. For many practical applications, these libraries and their React bindings are a good choice for managing state that is relevant to many components. It is far more likely that Redux is the right solution to your problem than that context is the right solution.

✅ If you aren't an experienced React developer, don't use context. There is usually a better way to implement functionality just using props and state.

👍 If you want your application to be stable, don't use context. It is an experimental API and it is likely to break in future releases of React.

👍 If you insist on using context despite these warnings, try to isolate your use of context to a small area and avoid using the context API directly when possible so that it's easier to upgrade when the API changes.



email

Enter something

address

Enter something

accept?

ON

SAVE CHANGES

CLEAR

```
render(
  <ThemeProvider name="extend">
    <App />
  </ThemeProvider>
)
```

```
render(
  <ThemeProvider name="extend">
    <App />
  </ThemeProvider>
)
```

```
class ThemeProvider extends React.Component {
  render() {
    return this.props.children
  }
}
```

```
render(
  <ThemeProvider name="extend">
    <App />
  </ThemeProvider>
)
```

```
class ThemeProvider extends React.Component {
  getChildContext() {
    return { theme: this.props.name }
  }
  render() {
    return this.props.children
  }
}
```

```
render(
  <ThemeProvider name="extend">
    <App />
  </ThemeProvider>
)
```

```
class ThemeProvider extends React.Component {
  getChildContext() {
    return { theme: this.props.name }
  }
  render() {
    return this.props.children
  }
}

ThemeProvider.childContextTypes = {
  theme: PropTypes.string
}
```

```
render(
  <ThemeProvider name="extend">
    <App />
  </ThemeProvider>
)
```

```
class ThemeProvider extends React.Component {
   ...
}

ThemeProvider.childContextTypes = {
  theme: PropTypes.string
}
```

```
render(
  <ThemeProvider name="extend">
    <App />
  </ThemeProvider>
)
```

```
class ThemeProvider extends React.Component {
   ...
}

ThemeProvider.childContextTypes = {
  theme: PropTypes.string
}

const Button = (props) => {
  return <button className="button">{props.children}</button>
}
```

```
render(
  <ThemeProvider name="extend">
    <App />
  </ThemeProvider>
)
```

```
class ThemeProvider extends React.Component {
   ...
}

ThemeProvider.childContextTypes = {
  theme: PropTypes.string
}

const Button = (props) => {
  return <button className="button">{props.children}</button>
}

Button.contextTypes = {
  theme: PropTypes.string
}



render(
  <ThemeProvider name="extend">
    <App />
  </ThemeProvider>
)
```

```
class ThemeProvider extends React.Component {
   ...
}

ThemeProvider.childContextTypes = {
  theme: PropTypes.string
}

const Button = (props, context) => {
  return <button className="button">{props.children}</button>
}

Button.contextTypes = {
  theme: PropTypes.string
}



render(
  <ThemeProvider name="extend">
    <App />
  </ThemeProvider>
)
```

```
class ThemeProvider extends React.Component {
   ...
}

ThemeProvider.childContextTypes = {
  theme: PropTypes.string
}

const Button = (props, context) => {
  let className = 'button' + 'button-' + context.theme
  return <button className={className}>{props.children}</button>
}

Button.contextTypes = {
  theme: PropTypes.string
}



render(
  <ThemeProvider name="extend">
    <App />
  </ThemeProvider>
)
```

email

Enter something

address

Enter something

accept?

ON

SAVE CHANGES

CLEAR

```
render(
  <ThemeProvider name="extend">
    <App />
  </ThemeProvider>
)
```

email

Enter something

address

Enter something

accept?

ON

SAVE CHANGES

CLEAR

```
render(
  <ThemeProvider name="manage">
    <App />
  </ThemeProvider>
)
```

Why Not To Use Context

👍 If you insist on using context despite these warnings, try to isolate your use of context to a small area and avoid using the context API directly when possible so that it’s easier to upgrade when the API changes.

```
class ThemeProvider extends React.Component {
   ...
}

ThemeProvider.childContextTypes = {
  theme: PropTypes.string
}

const Button = (props, context) => {
  let className = 'button' + 'button-' + context.theme
  return <button className={className}>{props.children}</button>
}

Button.contextTypes = {
  theme: PropTypes.string
}



render(
  <ThemeProvider name="extend">
    <App />
  </ThemeProvider>
)
```

```
class ThemeProvider extends React.Component {
   ...
}

ThemeProvider.childContextTypes = {
  theme: PropTypes.string
}

const Button = (props, context) => {
  let className = 'button' + 'button-' + context.theme
  return <button className={className}>{props.children}</button>
}
```

```
addContext(Button)
```

```
render(
  <ThemeProvider name="extend">
    <App />
  </ThemeProvider>
)
```

```
class ThemeProvider extends React.Component {
   ...
}

ThemeProvider.childContextTypes = {
  theme: PropTypes.string
}

export function addContext (Component) {
  Component['contextTypes'] = {
    theme: PropTypes.string
  }
  return Component
}
```

```
const Button = (props, context) => {
  let className = 'button' + 'button-' + context.theme
  return <button className={className}>{props.children}</button>
}

addContext(Button)
```

Why Not To Use Context

✅ If you insist on using context despite these warnings, try to isolate your use of context to a small area and avoid using the context API directly when possible so that it’s easier to upgrade when the API changes.

7 component patterns

I hope you learned something today

siddharthkp