352 lines (278 sloc) | 6.83 KB

Enforce a defaultProps definition for every prop that is not a required prop (react/require-default-props)

This rule aims to ensure that any non-required prop types of a component has a corresponding `defaultProps` value.

> **Note**: You can provide types in runtime types using PropTypes and/or statically using TypeScript or Flow. This rule will validate your prop types regardless of how you define them.

One advantage of `defaultProps` over custom default logic in your code is that `defaultProps` are resolved by React before the `PropTypes` typechecking happens, so typechecking will also apply to your `defaultProps`. The same also holds true for stateless functional components: default function parameters do not behave the same as `defaultProps` and thus using `defaultProps` is still preferred.

To illustrate, consider the following example:

With `defaultProps`:

```
const HelloWorld = ({ name }) => (
  <h1>Hello, {name.first} {name.last}!</h1>
);
```

```
HelloWorld.propTypes = {
  name: PropTypes.shape({
    first: PropTypes.string,
    last: PropTypes.string,
  })
};

HelloWorld.defaultProps = {
  name: 'john'
};

// Logs:
// Invalid prop `name` of type `string` supplied to `HelloWorld`, expect
ReactDOM.render(<HelloWorld />,  document.getElementById('app'));
```

Without `defaultProps`:

```
const HelloWorld = ({ name = 'John Doe' }) => (
  <h1>Hello, {name.first} {name.last}!</h1>
);

HelloWorld.propTypes = {
  name: PropTypes.shape({
    first: PropTypes.string,
    last: PropTypes.string,
  })
};

// Nothing is logged, renders:
// "Hello,!"
ReactDOM.render(<HelloWorld />,  document.getElementById('app'));
```

Rule Details

Examples of **incorrect** code for this rule:

```
function MyStatelessComponent({ foo, bar }) {
  return <div>{foo}{bar}</div>;
}

MyStatelessComponent.propTypes = {
  foo: PropTypes.string.isRequired,
  bar: PropTypes.string
};
```

```
var Greeting = createReactClass({
  render: function() {
    return <div>Hello {this.props.foo} {this.props.bar}</div>;
  },

  propTypes: {
    foo: PropTypes.string,
    bar: PropTypes.string
  },

  getDefaultProps: function() {
    return {
      foo: "foo"
    };
  }
});



class Greeting extends React.Component {
  render() {
    return (
      <h1>Hello, {this.props.foo} {this.props.bar}</h1>
    );
  }
}

Greeting.propTypes = {
  foo: PropTypes.string,
  bar: PropTypes.string
};

Greeting.defaultProps = {
  foo: "foo"
};



type Props = {
  foo: string,
  bar?: string
};

function MyStatelessComponent(props: Props) {
  return <div>Hello {props.foo} {props.bar}</div>;
}
```

Examples of **correct** code for this rule:

```
class Greeting extends React.Component {
```

```
  render() {
    return (
      <h1>Hello, {this.props.foo} {this.props.bar}</h1>
    );
  }

  static propTypes = {
    foo: PropTypes.string,
    bar: PropTypes.string.isRequired
  };

  static defaultProps = {
    foo: "foo"
  };
}



function MyStatelessComponent({ foo, bar }) {
  return <div>{foo}{bar}</div>;
}

MyStatelessComponent.propTypes = {
  foo: PropTypes.string.isRequired,
  bar: PropTypes.string.isRequired
};



function MyStatelessComponent({ foo, bar }) {
  return <div>{foo}{bar}</div>;
}

MyStatelessComponent.propTypes = {
  foo: PropTypes.string.isRequired,
  bar: PropTypes.string
};

MyStatelessComponent.defaultProps = {
    bar: 'some default'
};



type Props = {
  foo: string,
  bar?: string
};

function MyStatelessComponent(props: Props) {
  return <div>Hello {props.foo} {props.bar}</div>;
}
```

```
MyStatelessComponent.defaultProps = {
  bar: 'some default'
};


function NotAComponent({ foo, bar }) {}

NotAComponent.propTypes = {
  foo: PropTypes.string,
  bar: PropTypes.string.isRequired
};
```

Rule Options

```
...
"react/require-default-props": [<enabled>, { forbidDefaultForRequired: <
...
```

- `enabled` : for enabling the rule. 0=off, 1=warn, 2=error. Defaults to 0.
- `forbidDefaultForRequired` : optional boolean to forbid prop default for a required prop. Defaults to false.
- `ignoreFunctionalComponents` : optional boolean to ignore this rule for functional components. Defaults to false.

`forbidDefaultForRequired`

Forbids setting a default for props that are marked as `isRequired` .

Examples of **incorrect** code for this rule:

```
class Greeting extends React.Component {
  render() {
    return (
      <h1>Hello, {this.props.foo} {this.props.bar}</h1>
    );
  }

  static propTypes = {
    foo: PropTypes.string,
    bar: PropTypes.string.isRequired
  };

  static defaultProps = {
    foo: "foo",
    bar: "bar"
```

```
  };
}


function MyStatelessComponent({ foo, bar }) {
  return <div>{foo}{bar}</div>;
}

MyStatelessComponent.propTypes = {
  foo: PropTypes.string.isRequired,
  bar: PropTypes.string
};

MyStatelessComponent.defaultProps = {
  foo: 'foo',
  bar: 'bar'
};
```

Examples of **correct** code for this rule:

```
class Greeting extends React.Component {
  render() {
    return (
      <h1>Hello, {this.props.foo} {this.props.bar}</h1>
    );
  }

  static propTypes = {
    foo: PropTypes.string,
    bar: PropTypes.string.isRequired
  };

  static defaultProps = {
    foo: "foo"
  };
}


function MyStatelessComponent({ foo, bar }) {
  return <div>{foo}{bar}</div>;
}

MyStatelessComponent.propTypes = {
  foo: PropTypes.string.isRequired,
  bar: PropTypes.string.isRequired
};
```

ignoreFunctionalComponents

When set to `true`, ignores this rule for all functional components.

Examples of **incorrect** code for this rule:

```
class Greeting extends React.Component {
  render() {
    return (
      <h1>Hello, {this.props.foo} {this.props.bar}</h1>
    );
  }

  static propTypes = {
    foo: PropTypes.string,
    bar: PropTypes.string.isRequired
  };

  static defaultProps = {
    foo: "foo",
    bar: "bar"
  };
}
```

Examples of **correct** code for this rule:

```
function MyStatelessComponent({ foo, bar }) {
  return <div>{foo}{bar}</div>;
}

MyStatelessComponent.propTypes = {
  foo: PropTypes.string,
  bar: PropTypes.string
};



const MyStatelessComponent = ({ foo, bar }) => {
  return <div>{foo}{bar}</div>;
}

MyStatelessComponent.propTypes = {
  foo: PropTypes.string,
  bar: PropTypes.string
};



const MyStatelessComponent = function({ foo, bar }) {
  return <div>{foo}{bar}</div>;
}
```

```
MyStatelessComponent.propTypes = {
  foo: PropTypes.string,
  bar: PropTypes.string
};
```

When Not To Use It

If you don't care about using `defaultsProps` for your component's props that are not required, you can disable this rule.

Resources

- Official React documentation on defaultProps