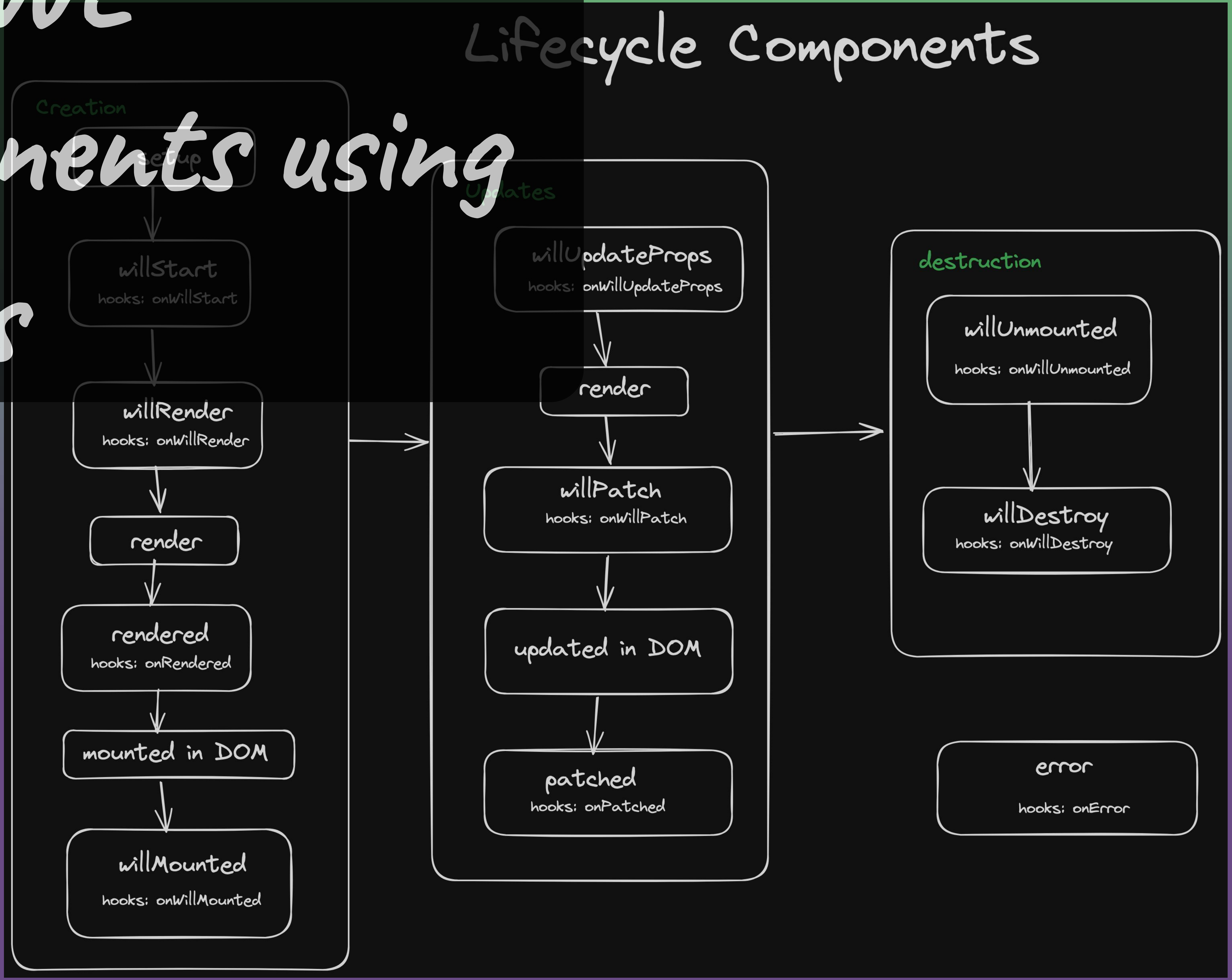
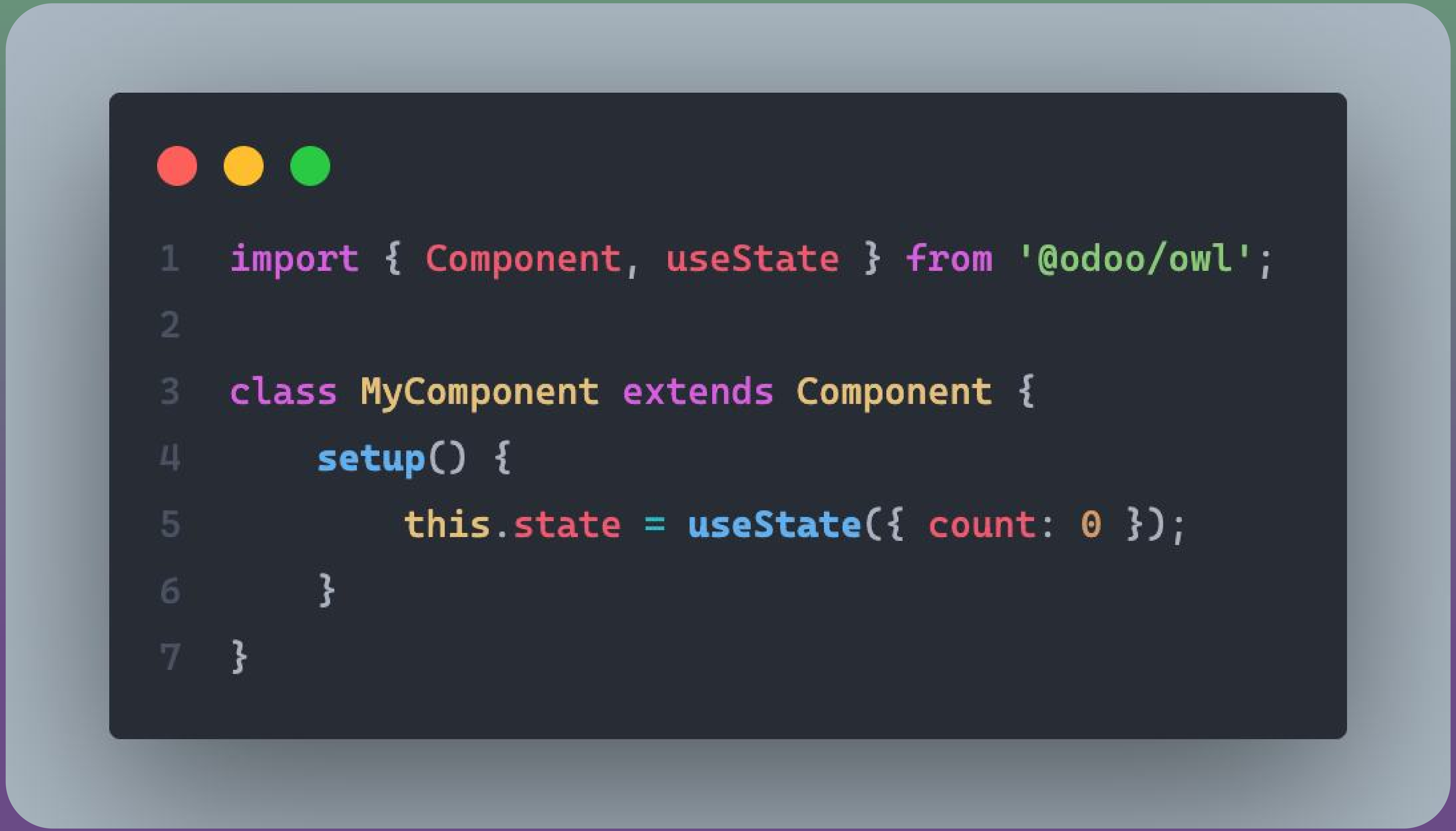
Odoo OWL Lifecycle Components using hooks



setup()

hook: None

Initializes the component by setting the state and properties. This method is called once, before any rendering or updating is done. In setup, you can define the initial state, bind reactive data, or prepare any logic that needs to be ready before the component starts interacting with the user.



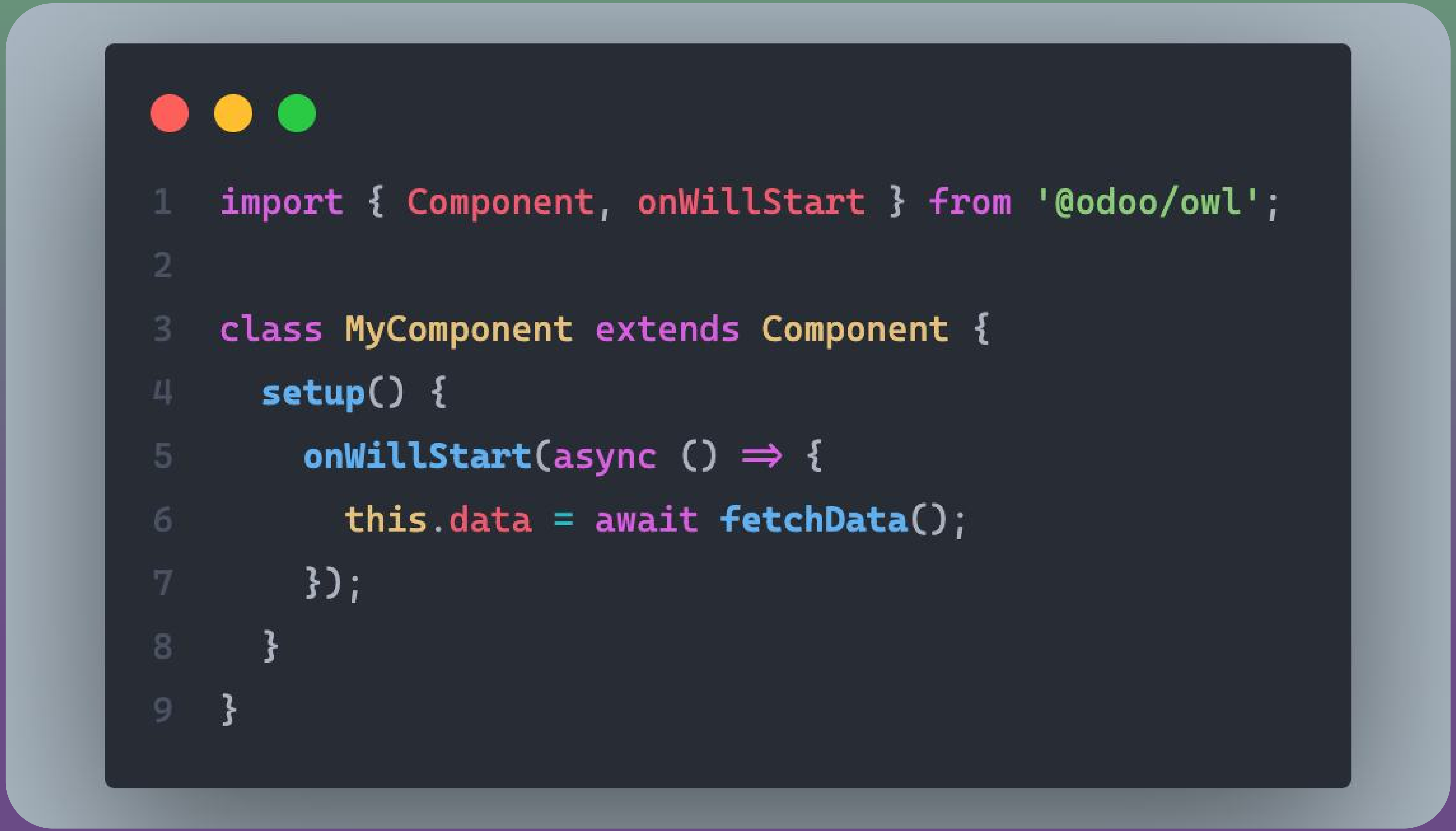
```
import { Component, useState } from '@odoo/owl';

class MyComponent extends Component {
    setup() {
        this.state = useState({ count: 0 });
    }
}
```

willStart

hook: onWillStart(async callback)

It is used to perform asynchronous tasks before the first render. It is ideal for initializing data that is needed for the first render of the component.



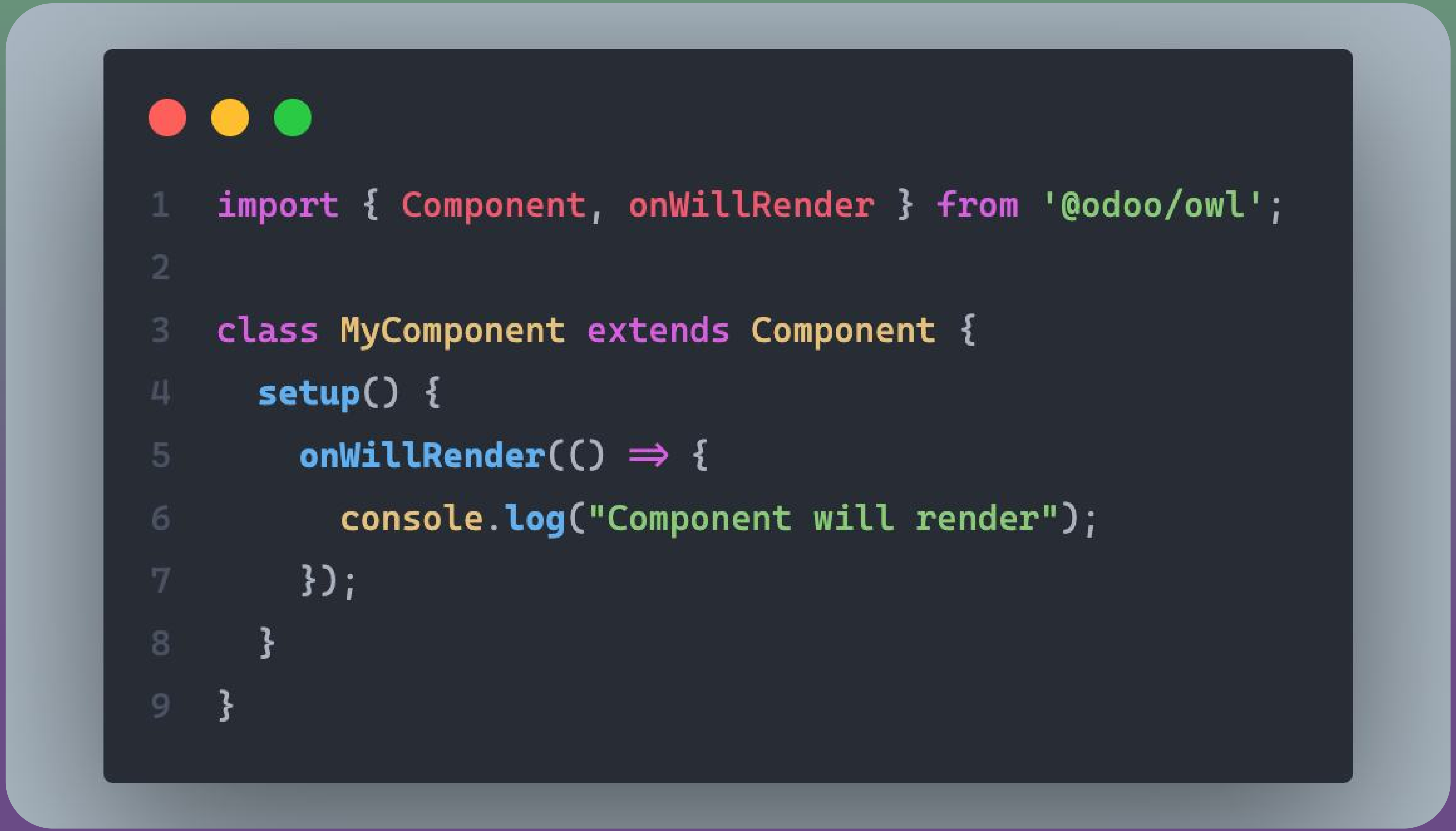
```
import { Component, onWillStart } from '@odoo/owl';

class MyComponent extends Component {
  setup() {
    onWillStart(async () => {
      this.data = await fetchData();
    });
  }
}
```

willRender

hook: onWillRender(callback)

Called just before the component is rendered. You can use this method to prepare any data or logic that needs to be ready just before the component is drawn.



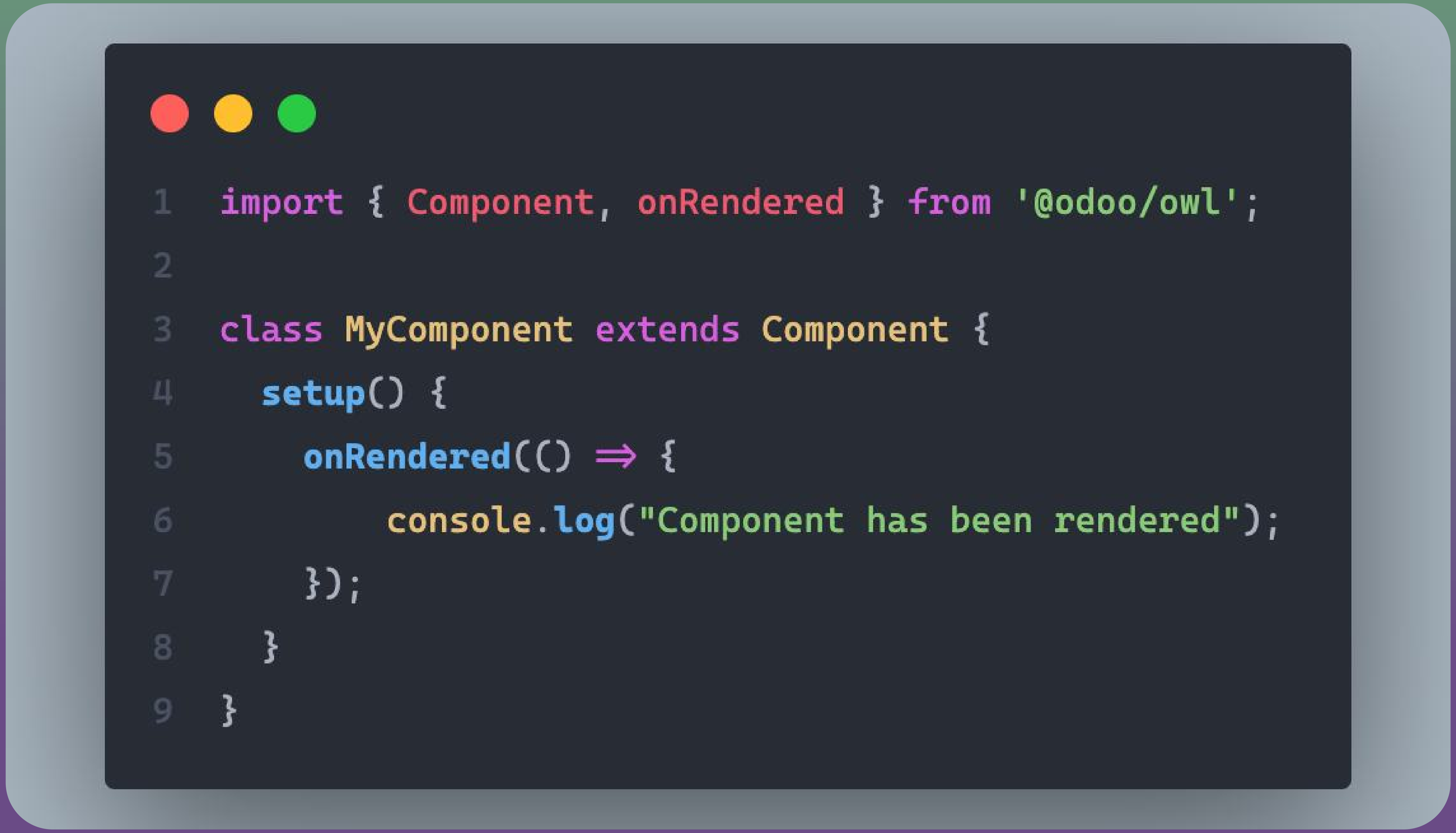
```javascript
import { Component, onWillRender } from '@odoo/owl';

class MyComponent extends Component {
  setup() {
    onWillRender(() => {
      console.log("Component will render");
    });
  }
}
```

rendered

hook: onRendered(callback)

Invoked right after the component has been rendered. You can use this method to perform tasks that depend on the existence of the component's DOM, such as initializing libraries that interact with the DOM.



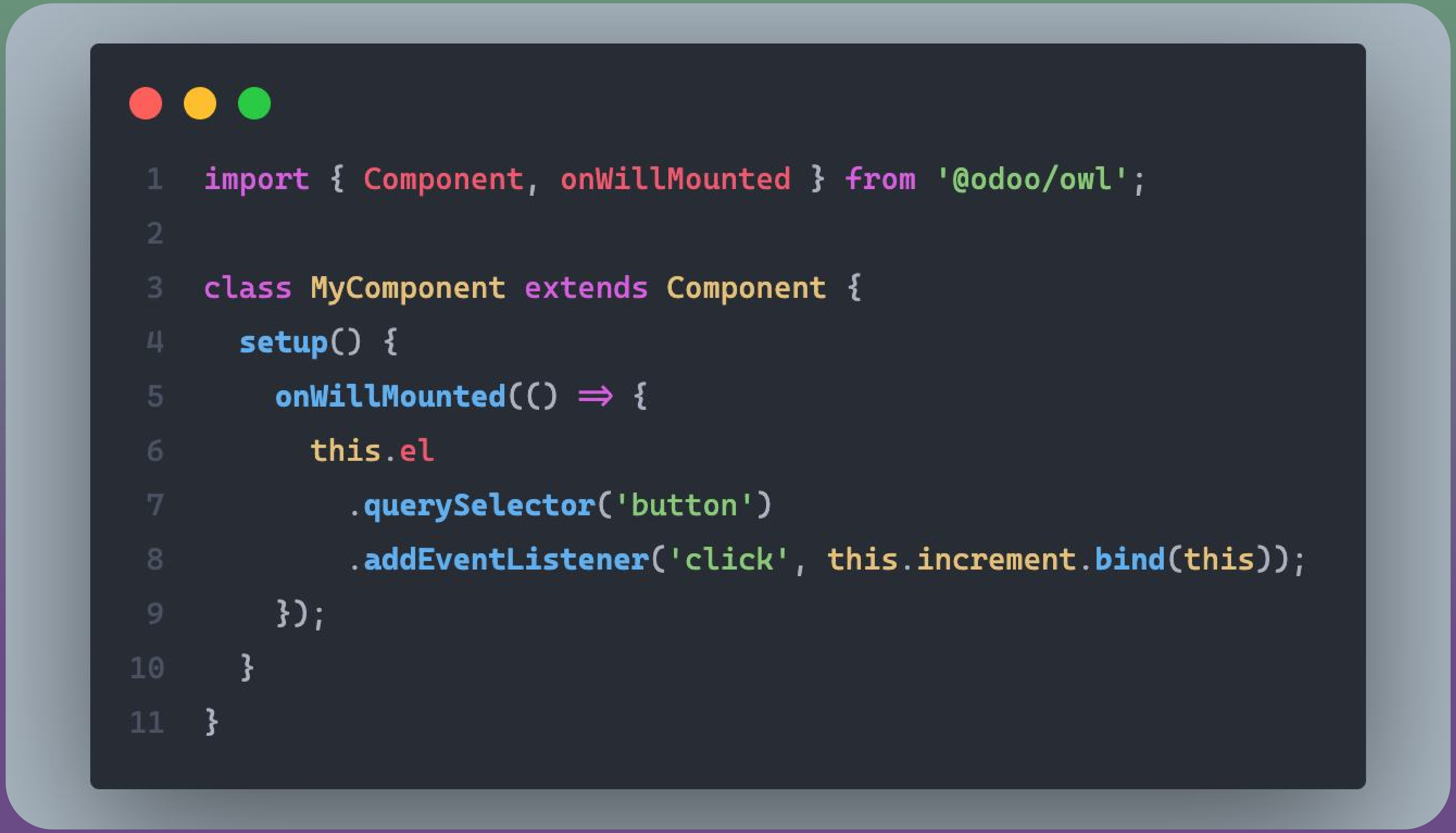
```
import { Component, onRendered } from '@odoo/owl';

class MyComponent extends Component {
  setup() {
    onRendered(() ⇒ {
        console.log("Component has been rendered");
    });
  }
}
```

mounted

hook: onWillMounted(callback)

Called after the component has been rendered and added to the DOM. It is useful for performing DOM manipulations or initializing integrations that require the component to be completely in the DOM.



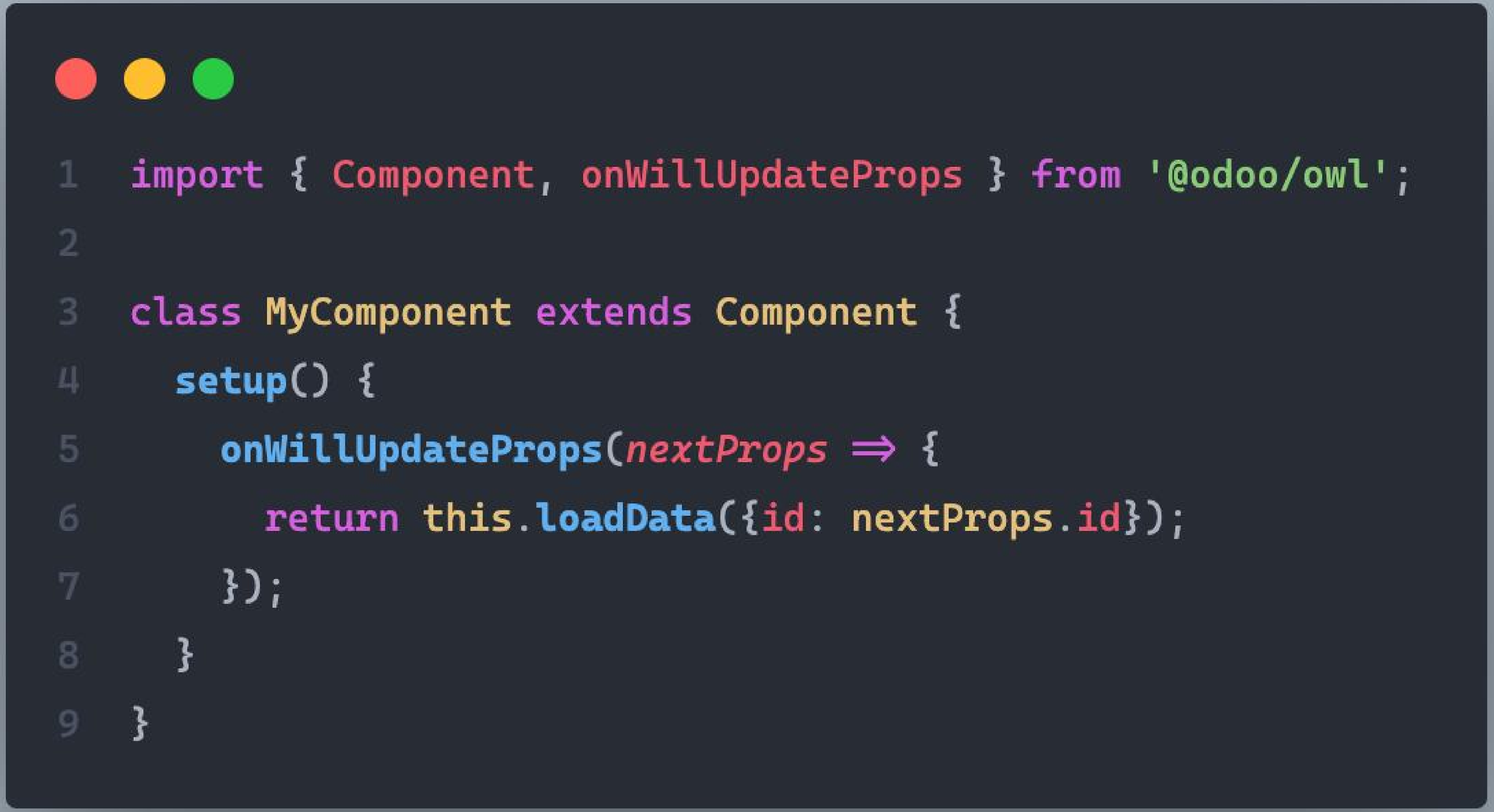
```javascript
import { Component, onWillMounted } from '@odoo/owl';

class MyComponent extends Component {
  setup() {
    onWillMounted(() => {
      this.el
        .querySelector('button')
        .addEventListener('click', this.increment.bind(this));
    });
  }
}
```

willUpdateProps

hook: onWillUpdateProps(callback)

Called before the component properties are updated. Here you can implement logic that reacts to property changes before the component is re-rendered.



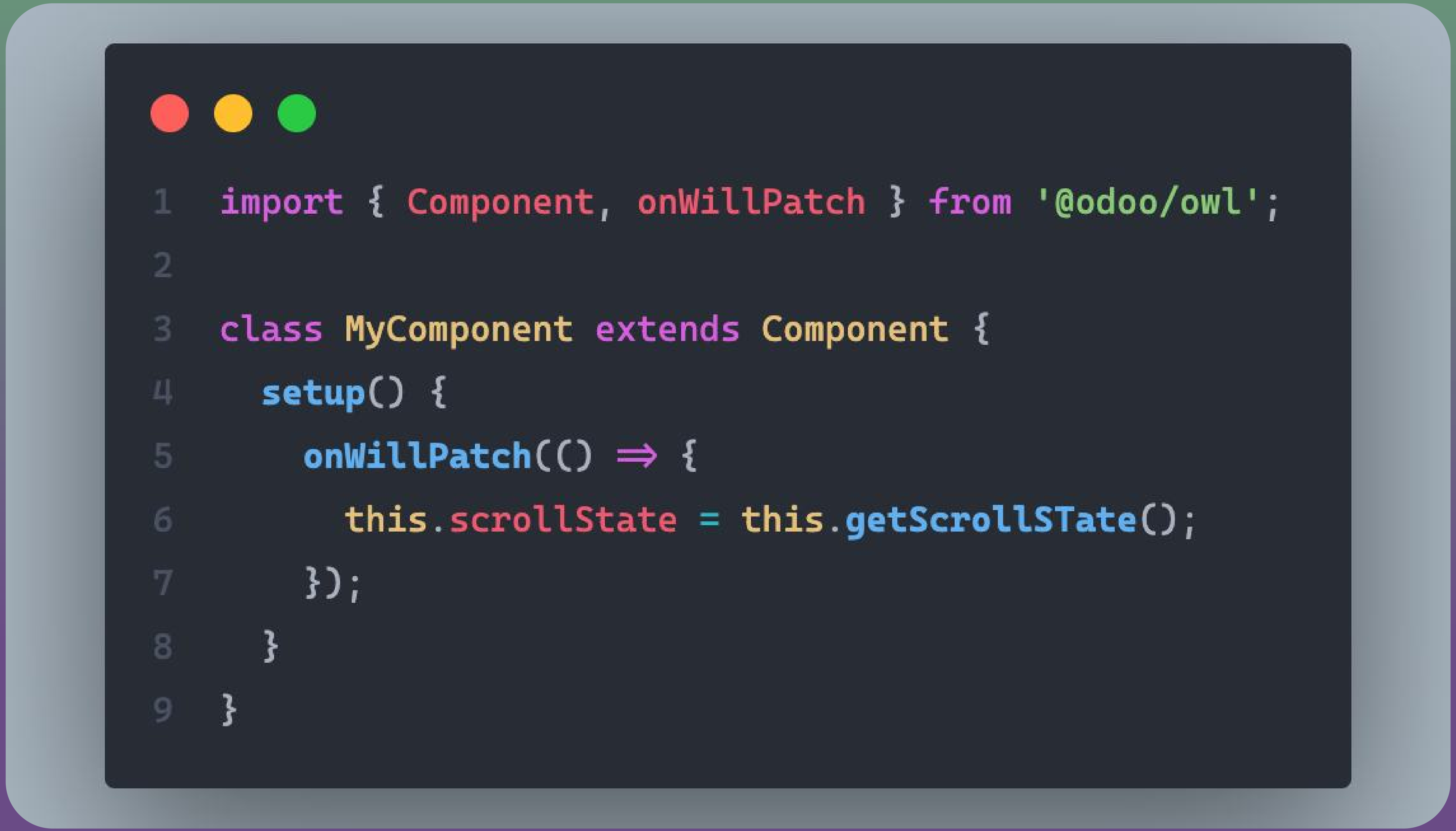
```
import { Component, onWillUpdateProps } from '@odoo/owl';

class MyComponent extends Component {
  setup() {
    onWillUpdateProps(nextProps => {
      return this.loadData({id: nextProps.id});
    });
  }
}
```

willPatch

hook: onWillPatch(callback)

Called just before the DOM is patched with the new data. You can use this method to prepare any changes to the DOM that need to be handled just before OWL applies the differences to the DOM.



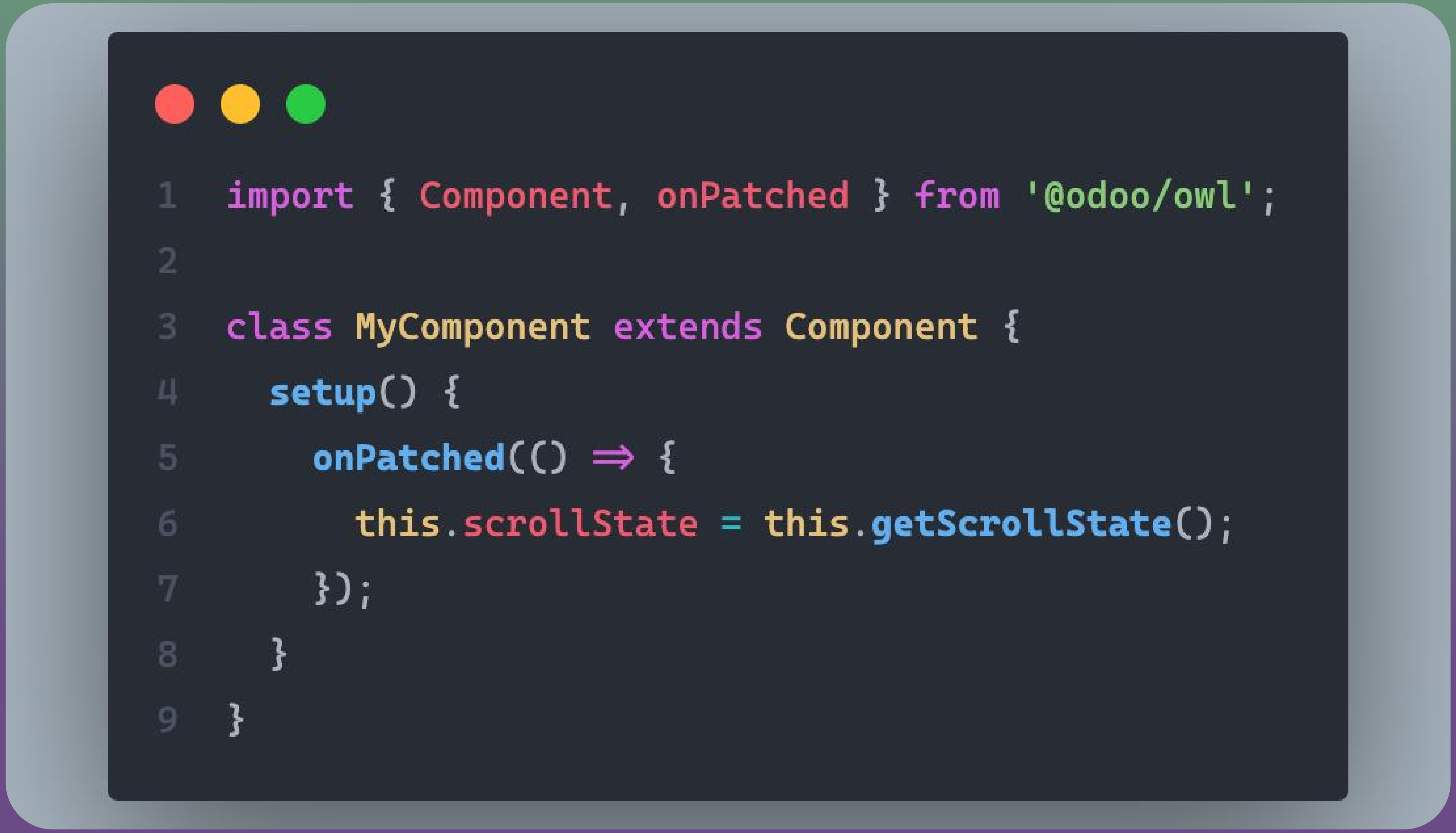
```javascript
import { Component, onWillPatch } from '@odoo/owl';

class MyComponent extends Component {
  setup() {
    onWillPatch(() => {
      this.scrollState = this.getScrollSTate();
    });
  }
}
```

patched

hook: onPatched(callback)

It is invoked right after the DOM has been patched. It is useful for making final adjustments or reactions to the DOM update.



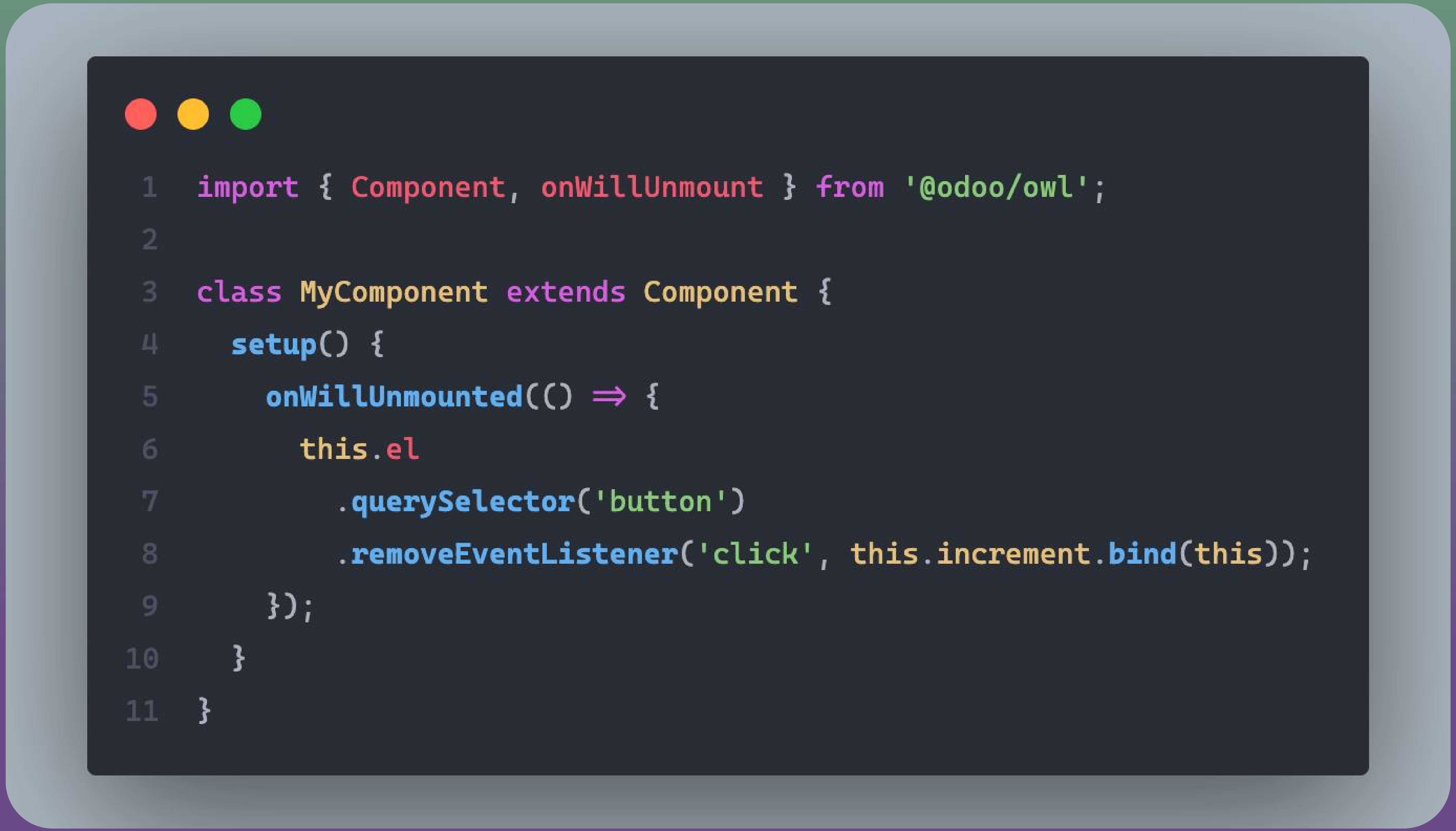
```javascript
import { Component, onPatched } from '@odoo/owl';

class MyComponent extends Component {
  setup() {
    onPatched(() => {
      this.scrollState = this.getScrollState();
    });
  }
}
```

willUnmount

hook: onWillUnmount(callback)

It is used just before the component is removed from the DOM. It is ideal for cleaning up resources or event listeners that were configured during the component's lifecycle.



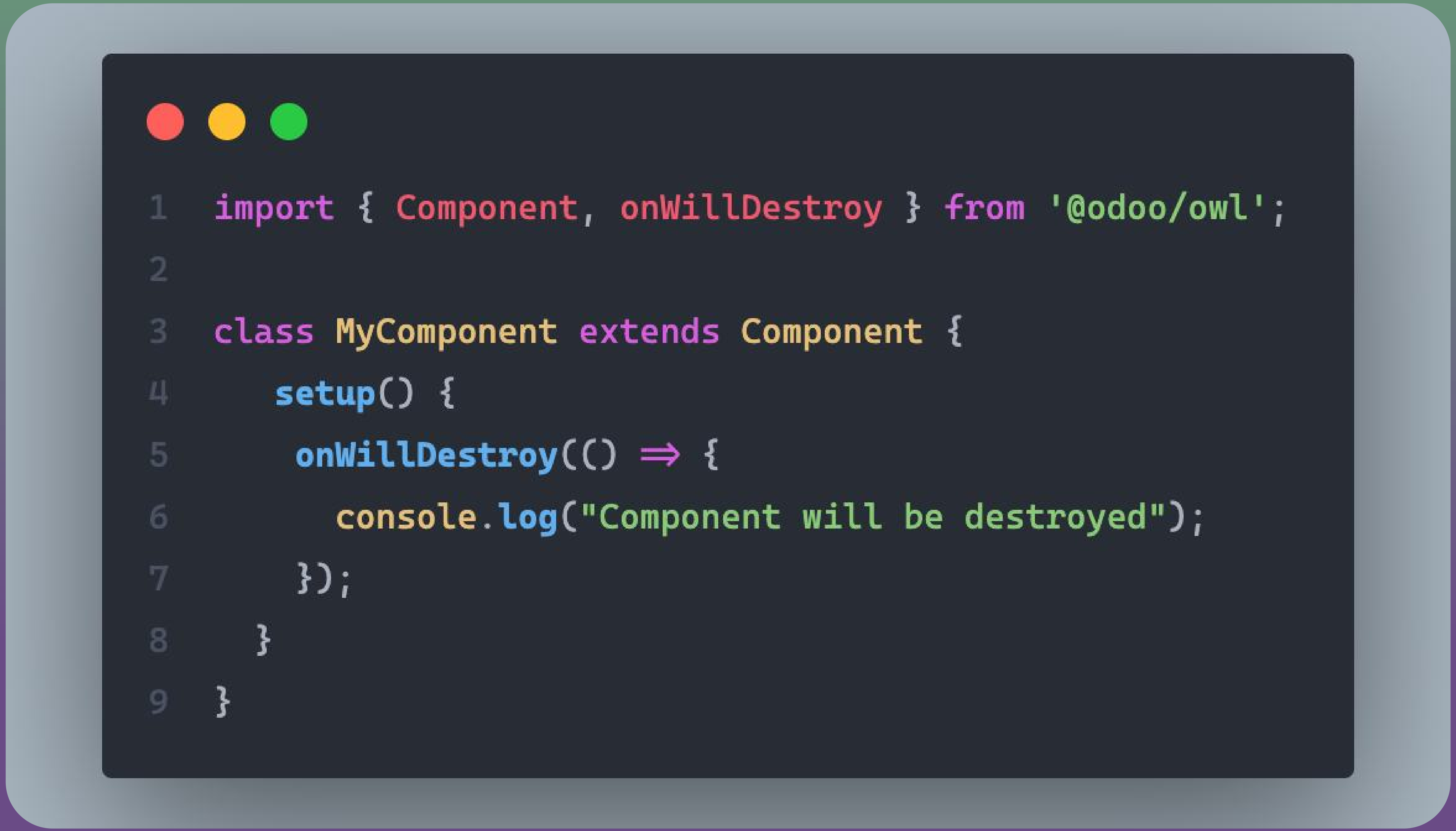
```
import { Component, onWillUnmount } from '@odoo/owl';

class MyComponent extends Component {
  setup() {
    onWillUnmounted(() => {
      this.el
        .querySelector('button')
        .removeEventListener('click', this.increment.bind(this));
    });
  }
}
```

willDestroy

hook: onWillDestroy(callback)

Called just before the component is destroyed. Here you can clean up any resources that the component has created or managed, such as database connections, subscriptions, etc.



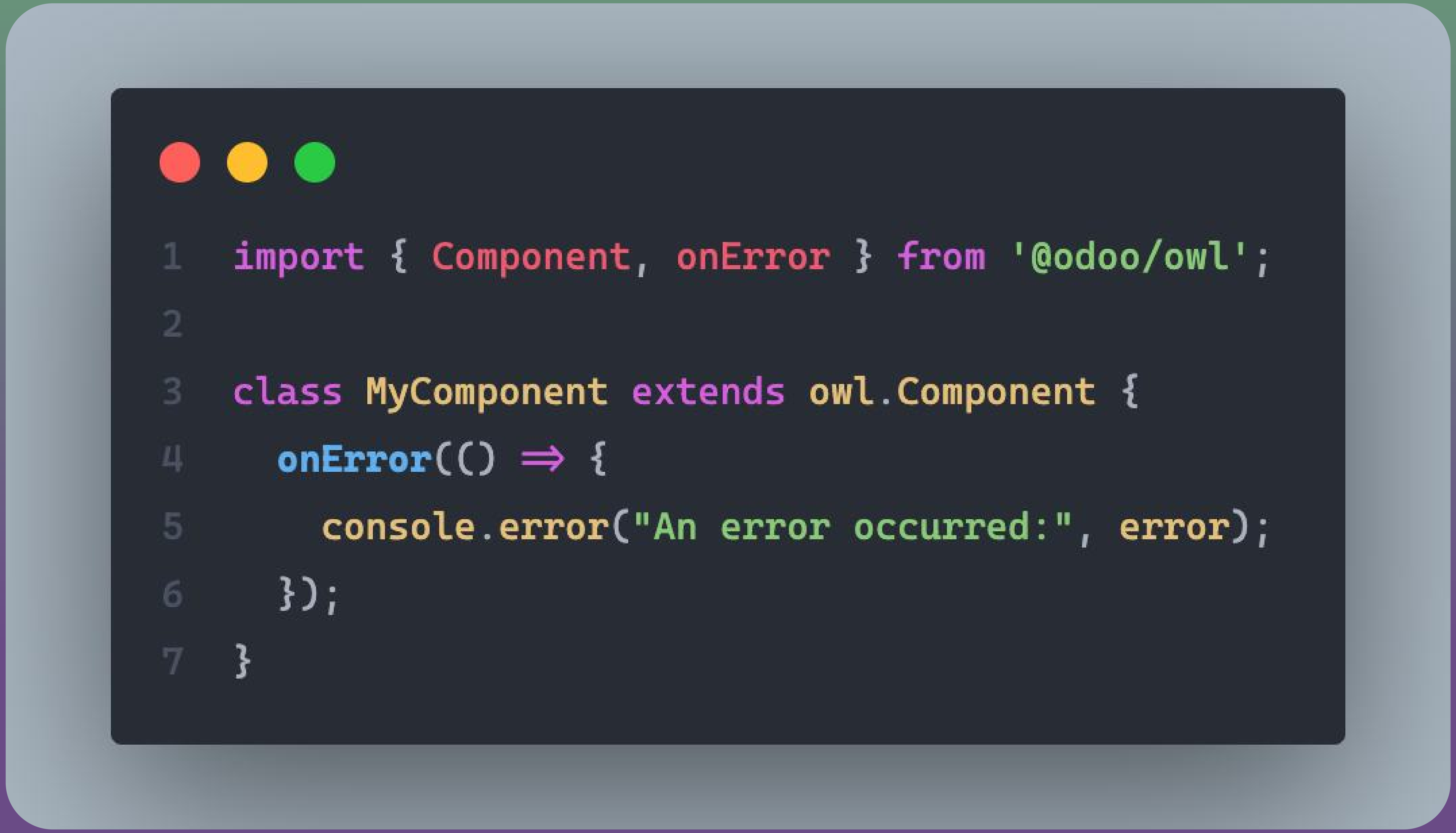
```javascript
import { Component, onWillDestroy } from '@odoo/owl';

class MyComponent extends Component {
   setup() {
    onWillDestroy(() => {
      console.log("Component will be destroyed");
    });
  }
}
```

error

hook: onError(callback)

Handles errors that occur during the component lifecycle. You can use this method to capture and handle errors globally in the component.



```javascript
import { Component, onError } from '@odoo/owl';

class MyComponent extends owl.Component {
  onError(() ⇒ {
    console.error("An error occurred:", error);
  });
}
```