145- Testing the Editor part II video

- go to Editor.test.js
- we are going to be testing for handleBodyChange and handleTitleChange

```
        it('Should remove note', function(){
            const wrapper = mount(<Editor browserHistory={browserHistory} call={call} selectedNo
teId={notes[0]._id} note={notes[0]}/>);

            wrapper.find('button').simulate('click');
            // expect(Session.set).toHaveBeenCalledWith('selectedNoteId',notes[0]._id);
            expect(call).toHaveBeenCalledWith('notes.remove',notes[0]._id );
            expect(browserHistory.push).toHaveBeenCalledWith('/dashboard');

        });

        it('Should update the note body on textarea change', function(){

        });
```

- **we are again going to define wrapper with all three props**

```
        it('Should update the note body on textarea change', function(){
            const wrapper = mount(<Editor browserHistory={browserHistory} call={call} selectedNo
teId={notes[0]._id} note={notes[0]}/>);

        });
```

- now we want to simulate a change on that text area

```
        it('Should update the note body on textarea change', function(){
            const wrapper = mount(<Editor browserHistory={browserHistory} call={call} selectedNo
teId={notes[0]._id} note={notes[0]}/>);

            wrapper.find('textarea').simulate('change', {
```

```
        });
```

- and for the first time we passed in a second argument after change, and this is going to be an object and we want to target the 'e' as an event, so we are trying to replicate e.target.value, and this case our value is the string we gave the variable newBody

```
        it('Should update the note body on textarea change', function(){
            const newBody = 'This is my new body text'
            const wrapper = mount(<Editor browserHistory={browserHistory} call={call} selectedNo
teId={notes[0]._id} note={notes[0]}/>);

            wrapper.find('textarea').simulate('change', {
                target: {
                    value: newBody
                }
            });
        });
```

- now we are ready to make our assertions
- first we are expecting the state body to be our newBody

```
        it('Should update the note body on textarea change', function(){
            const newBody = 'This is my new body text'
            const wrapper = mount(<Editor browserHistory={browserHistory} call={call} selectedNo
teId={notes[0]._id} note={notes[0]}/>);

            wrapper.find('textarea').simulate('change', {
                target: {
                    value: newBody
                }
            });

            expect(wrapper.state('body')).toBe(newBody);
        });
```

- then we are going to expect call to have been called with some arguments - first one is notes.update, then the note._id, and the last one is an object with the body property

```
        it('Should update the note body on textarea change', function(){
            const newBody = 'This is my new body text'
            const wrapper = mount(<Editor browserHistory={browserHistory} call={call} selectedNo
teId={notes[0]._id} note={notes[0]}/>);

            wrapper.find('textarea').simulate('change', {
                target: {
                    value: newBody
                }
            });

            expect(wrapper.state('body')).toBe(newBody);
            expect(call).toHaveBeenCalledWith('notes.update', notes[0]._id, {body: newBody});
        });
```

- this should have worked but there was a typo that kept the test from passing, in Editor.js in componentDidUpdate, I had prevProps.note_id, it should have been prevProps.note._id

```
 componentDidUpdate(prevProps, prevState){
        const currentNoteId = this.props.note ? this.props.note._id : undefined;
        const prevNoteId = prevProps.note ? prevProps.note._id : undefined;

        if(currentNoteId && currentNoteId !== prevNoteId){
            this.setState({
                title: this.props.note.title,
                body: this.props.note.body
            });
        }
    }
```

- we then are going to be doing to the same test case but now for title, and it is basically the same steps with a few minor details

```
        it('Should update the note body on textarea change', function(){
            const newBody = 'This is my new body text';
            const wrapper = mount(<Editor browserHistory={browserHistory} call={call} selectedNo
teId={notes[0]._id} note={notes[0]}/>);

           wrapper.find('textarea').simulate('change',{
               target: {
                   value: newBody
               }
           });

            expect(wrapper.state('body')).toBe(newBody);
            expect(call).toHaveBeenCalledWith('notes.update', notes[0]._id, { body: newBody });
        });

        it('Should update the note title on input change', function(){
            const newTitle= 'This is my new title text';
            const wrapper = mount(<Editor browserHistory={browserHistory} call={call} selectedNo
teId={notes[0]._id} note={notes[0]}/>);

           wrapper.find('input').simulate('change',{
               target: {
                   value: newTitle
               }
           });

            expect(wrapper.state('title')).toBe(newTitle);
            expect(call).toHaveBeenCalledWith('notes.update', notes[0]._id, { title: newTitle
});

        });
```

- **our next test case is going to make sure when our props change componentDidUpdate handle things correctly**

```
        it('Should update the note title on input change', function(){
            const newTitle= 'This is my new title text';
            const wrapper = mount(<Editor browserHistory={browserHistory} call={call} selectedNo
teId={notes[0]._id} note={notes[0]}/>);

           wrapper.find('input').simulate('change',{
               target: {
                   value: newTitle
               }
           });

            expect(wrapper.state('title')).toBe(newTitle);
            expect(call).toHaveBeenCalledWith('notes.update', notes[0]._id, { title: newTitle
});
        });

        it('Should set state for new note', function(){

        })
```

- **then we are going to an instance of Editor component without selectedNoteId, and note props**

```
        it('Should set state for new note', function(){
            const wrapper = mount(<Editor browserHistory={browserHistory} call={call}/>);
        })
```

- and then we are going to set our missing props by using and enzyme method called setProps(), and that takes an object, and that is where we are going to define our props

```
        it('Should set state for new note', function(){
            const wrapper = mount(<Editor browserHistory={browserHistory} call={call}/>);

            wrapper.setProps({
                selectedNoteId: notes[0]._id,
```

```
            note: notes[0]
        });
    });
```

- now we are ready to make the assertions for the title and body

```
    it('Should set state for new note', function(){
        const wrapper = mount(<Editor browserHistory={browserHistory} call={call}/>);

        wrapper.setProps({
            selectedNoteId: notes[0]._id,
            note: notes[0]
        });

        expect(wrapper.state('title')).toBe(notes[0].title);
        expect(wrapper.state('body')).toBe(notes[0].body);
    });
```

- let do another test case by copyin the code above and making some minor changes, we are going to be removing one prop from setProps, and are expecting title to be empty strings

```
    it('Should not set state if note prop not provided', function(){
        const wrapper = mount(<Editor browserHistory={browserHistory} call={call}/>);

        wrapper.setProps({
            selectedNoteId: notes[0]._id
        });

        expect(wrapper.state('title')).toBe('');
        expect(wrapper.state('body')).toBe('');
    });
```

- now lets push our code to github

```
joses-MacBook-Pro:notes mendoza$ git add .
```

```
joses-MacBook-Pro:notes mendoza$ git commit -m "Add test cases for editor component"

joses-MacBook-Pro:notes mendoza$ git push
```

-