135-Rendering Empty List Item video

- lets clear our notes by going to the terminal
- first connect to meteor mongo

```
joses-MacBook-Pro:notes mendoza$ meteor mongo
```

- **and we can remove() our database by passing an empty object**

```
meteor:PRIMARY> db.notes.remove({})

WriteResult({ "nRemoved" : 8 })
```

- now in the browser you should see no notes
- go to NoteList.js and import our new component called NoteListEmptyItem and also lets make that new component in imports/ui
- go to NoteListEmptyItem.js and import React because we do have some jsx
- then we want define our stateless functional component - this will not take any props
- and we will just return simple text

```
import React from 'react';


export default ()=>{
    return (
        <div>
            <h5>You have no notes</h5>
            <p>Create a note to get started</p>
        </div>
    );
}
```

- now go to NoteList.js and we are going to check if {notes.length === 0 } ? then we will show are component <NoteListEmptyItem/> if we do have notes : undefined

```
import React from 'react';
import { Meteor } from 'meteor/meteor';
import { createContainer } from 'meteor/react-meteor-data';
import PropTypes from 'prop-types';
```

```
import NoteListItem from './NoteListItem';


import { Notes } from '../api/notes'

import NoteListHeader from './NoteListHeader';

import NoteListEmptyItem from './NoteListEmptyItem';



export const NoteList = (props)=>{

    return (

        <div>

            <NoteListHeader/>

            {props.notes.length === 0 ? <NoteListEmptyItem /> : undefined }

            { props.notes.map((note)=>{

                    return <NoteListItem key={note._id} note={note}/>

            })}

            NoteList { props.notes.length }

        </div>

    );

};
```

- we are now going to write test cases for the NoteList.js component
- lets remove our notes once again on the terminal

```
meteor:PRIMARY> db.notes.remove({})

WriteResult({ "nRemoved" : 4 })
```

- now we are gong to make a quick tweak to NoteListEmptyItem.js for dom tree structural reasons

```
import React from 'react';


const NoteListEmptyItem = ()=>{

    return (

        <div>

            <h5>You have no notes</h5>
```

```
            <p>Create a note to get started</p>
        </div>
    );
}

export default NoteListEmptyItem;
```

- **and now if we check this in our chrome tools we are going to be able to see our component, instead of a stateless component**
- **now we are ready for our tests, create a new file in imports/ui called NoteList.test.js**
- **lets import our test modules and also our NoteList component**

```
import React from 'react';
import expect from 'expect';
import  { mount } from 'enzyme';
import { Meteor } from 'meteor/meteor';

import { NoteList } from './NoteList';
```

- next up is to make sure we are on the client side

```
import React from 'react';
import expect from 'expect';
import  { mount } from 'enzyme';
import { Meteor } from 'meteor/meteor';

import { NoteList } from './NoteList';

if(Meteor.isClient){

}
```

- **now we can have a describe block**

```
import React from 'react';
import expect from 'expect';
```

```
import  { mount } from 'enzyme';
import { Meteor } from 'meteor/meteor';

import { NoteList } from './NoteList';

if(Meteor.isClient){
    describe('NoteList', function(){

    });
}
```

- now we can set up our two test cases
- lets first create dummy data to compare it to our real data
- lets create an array first so it can mimic the query of objects in our database
- and this will have everything our database will have

```
import React from 'react';
import expect from 'expect';
import  { mount } from 'enzyme';
import { Meteor } from 'meteor/meteor';

import { NoteList } from './NoteList';

const notes = [
    {
        _id: 'noteId1',
        title: 'Test title',
        body: '',
        updatedAt: 0,
        userId: 'userId1'
    }, {
        _id: 'noteId2',
        title: '',
        body: 'Something in here',
```

```
        updatedAt: 0,
        userId: 'userId2'
    }
];

if(Meteor.isClient){
    describe('NoteList', function(){

    });
}
```

- lets do our first test

```
if(Meteor.isClient){
    describe('NoteList', function(){
        it('shoud render NoteListItem for each note', function(){

        })
    });
}
```

- now to the second test

```
if(Meteor.isClient){
    describe('NoteList', function(){
        it('shoud render NoteListItem for each note', function(){

        });

        it('should render NoteListEmptyItem if zero notes', function(){

        })
    });
}
```

- now in the first test lets create a const wrapper and we are going to mount an instance of <NoteList/> and we are going to pass our dummy data we created

```
if(Meteor.isClient){

    describe('NoteList', function(){

        it('shoud render NoteListItem for each note', function(){

            const wrapper = mount(<NoteList notes={notes}/>)

        });


        it('should render NoteListEmptyItem if zero notes', function(){


        })

    });

}
```

- we can now do our assertions and this time we'll look for a react component rather than an h1 or a button
- and we are going to check the length of NoteListItem to find out how many instances of NoteListItem are rendered, and we are expecting toBe 2 instances

```
if(Meteor.isClient){

    describe('NoteList', function(){

        it('shoud render NoteListItem for each note', function(){

            const wrapper = mount(<NoteList notes={notes}/>)


            expect(wrapper.find('NoteListItem').length).toBe(2);

        });


        it('should render NoteListEmptyItem if zero notes', function(){


        })

    });

}
```

- and we are expecting NoteListEmptyItem toBe 0

```
if(Meteor.isClient){

    describe('NoteList', function(){

        it('shoud render NoteListItem for each note', function(){

            const wrapper = mount(<NoteList notes={notes}/>)


            expect(wrapper.find('NoteListItem').length).toBe(2);

            expect(wrapper.find('NoteListEmptyItem').length).toBe(0);

        });


        it('should render NoteListEmptyItem if zero notes', function(){


        })

    });

}
```

- go to the terminal and shutdown meteor mongo and start the test suite

```
joses-MacBook-Pro:notes mendoza$ npm test
```

- **now lets do the second test over in NoteList.test.js**
- **we are going to be copying the first test but with some tweaks**
- **we are going to be checking an empty array**

```
if(Meteor.isClient){

    describe('NoteList', function(){

        it('shoud render NoteListItem for each note', function(){

            const wrapper = mount(<NoteList notes={notes}/>);


            expect(wrapper.find('NoteListItem').length).toBe(2);

            expect(wrapper.find('NoteListEmptyItem').length).toBe(0);

        });


        it('should render NoteListEmptyItem if zero notes', function(){

            const wrapper = mount(<NoteList notes={[]}/>);
```

```
            expect(wrapper.find('NoteListItem').length).toBe(0);
            expect(wrapper.find('NoteListEmptyItem').length).toBe(1);
        });
    });
}
```

- lets now quit out of npm test
- and run git

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

- lets now add all of our files

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

- **lets commit**

```
joses-MacBook-Pro:notes mendoza$ git commit -m "render empty item if no notes"
```

- **now lets push it up to our github repo**

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