126-Mocking functions with Spies video

- we are going to be using Spies to trigger some of the functions in our code
- the docs for spies is in https://github.com/mjackson/expect
- Go to PrivateHeader.test.js and make a new test

```
if(Meteor.isClient) {

    describe('PrivateHeader', function(){

        it('should set button text to logout', function(){

           const wrapper = mount(<PrivateHeader title="test title"/>);

           const buttonText = wrapper.find('button').text();


           expect(buttonText).toBe('Logout');

        });


        it('should use title prop as h1 text', function(){

            const title = 'test title';

            const wrapper = mount(<PrivateHeader title={title}/>)

            const h1 = wrapper.find('h1').text();


            expect(h1).toBe(title);

        });


        it('should call the function', function(){


        });

    });

}
```

- lets create a new spy

```
        it('should call the function', function(){
```

```
            const spy = expect.createSpy();

        });

    });

}
```

- now we are expect for spy to be called

```
        it('should call the function', function(){


            const spy = expect.createSpy();

            expect(spy).toHaveBeenCalled()

        });

    });

}
```

- in the browser the test didn't pass because nothing was called
- now lets call spy with any arguement we want because it is just a mock function

```
        it('should call the function', function(){


            const spy = expect.createSpy();

            spy(2, 34, true);

            expect(spy).toHaveBeenCalled()

        });

    });

}
```

- in a real worl app you wouldn't just call spy, you would injected somewhere in your code component, but we are going this to get familiar with spy
- now lets check if it didn't get called

```
        it('should call the function', function(){


            const spy = expect.createSpy();

            spy(2, 34, true);

            expect(spy).toNotHaveBeenCalled()
```

```
    });
```

- and this fails because we did call it up above
- now we are going to check if spy was called with a particular set of arguments
- now we are going to be using the method toHaveBeenCalledWith, it lets you specify the arguments you are passing in spy

```
  it('should call the function', function(){


      const spy = expect.createSpy();

      spy(2, 34, true);

      spy('jose');

      expect(spy).toHaveBeenCalledWith(2, 34, true);

    });
```

- this will pass on the browser
- now if I do this

```
    it('should call the function', function(){


      const spy = expect.createSpy();

      spy(2, 34, true);

      spy('jose');

      expect(spy).toHaveBeenCalledWith(2);

    });

});
```

- it does fail on the browser
- now if do this

```
    it('should call the function', function(){


      const spy = expect.createSpy();

      spy(2, 34, true);

      spy('jose');

      expect(spy).toHaveBeenCalledWith('jose');
```

- it passes as well
- lets throw debugger to our code to see, what is going on with spy

```
        it('should call the function', function(){


            const spy = expect.createSpy();

            spy(2, 34, true);

            spy('jose');

            debugger;

            expect(spy).toHaveBeenCalledWith('jose');
```

- now over on chrome tools in the console lets expore
- this is what happens when we call spy.calls
- it returns an array of objects,

```
spy.calls

(2) [{…}, {…}]

0

:

{context: undefined, arguments: Array(3)}

1

:

{context: undefined, arguments: Array(1)}

length

:

2
```

- and if you open up those objects in the console you can clearly see the arguments that getting passed on

```
(2) [{…}, {…}]

0

:

arguments

:

(3) [2, 34, true]
```

```
context
:
undefined
__proto__
:
Object
1
:
arguments
:
["jose"]
context
:
undefined
__proto__
:
Object
length
:
2
__proto__
:
Array(0)
```

- in the console to check the length of a spy (how many time spy got called), in our case it got called twiece, so check if it did

```
expect(spy.calls.length).toBe(2)
```

- **and this returns**

```
Expectation {actual: 2}
```

- if you do 3, you will get an error

```
expect(spy.calls.length).toBe(3)

Uncaught Error: Expected 2 to be 3
```

- **now lets** remove the debugger keyword
- now we need to figure how to get spy() into PrivareHeader.js, we are going to pass in our logout function into a prop
- so head over to PrivateHeader.js and create a new PropType

```
import React from 'react';
import { Accounts } from 'meteor/accounts-base';

const PrivateHeader = (props)=>{
    return (
        <div className="header">
            <div className="header__content">
                <h1 className="header__title">{props.title}</h1>
                <button className="button button--link-text " onClick={()=> Accounts.logout()}>L
ogout</button>
            </div>
        </div>
    );
}

PrivateHeader.propTypes = {
    title: React.PropTypes.string.isRequired,
    handleLogout: React.PropTypes.func.isRequired
};

export default PrivateHeader;
```

- now swap out the Accounts.logout() with this new prop

```
const PrivateHeader = (props)=>{
    return (
        <div className="header">
```

```
            <div className="header__content">
                <h1 className="header__title">{props.title}</h1>
                <button className="button button--link-text " onClick={()=>
props.handleLogout()}>Logout</button>
            </div>
        </div>
    );
}

PrivateHeader.propTypes = {
    title: React.PropTypes.string.isRequired,
    handleLogout: React.PropTypes.func.isRequired
};

export default PrivateHeader;
```

- now go to PrivateHeader.test.js
- create a new test case

```
        it('should call the function', function(){

            const spy = expect.createSpy();
            spy(2, 34, true);
            spy('jose');
            expect(spy).toHaveBeenCalledWith('jose');
        });

        it('should call handleLogout on click', function(){

        });
```

- **lets pass our new handleLogout to our component**
- **and for now is just an empty arrow function, and that is fine**

```
f(Meteor.isClient) {
```

```
    describe('PrivateHeader', function(){
        it('should set button text to logout', function(){
            const wrapper = mount(<PrivateHeader title="test title" handleLogout={()=>{}}/>);
            const buttonText = wrapper.find('button').text();

            expect(buttonText).toBe('Logout');
        });
```

- lets add the same prop to our second PrivateHeader instance

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

import PrivateHeader from './PrivateHeader';

if(Meteor.isClient) {
    describe('PrivateHeader', function(){
        it('should set button text to logout', function(){
            const wrapper = mount(<PrivateHeader title="test title" handleLogout={()=>{}}/>);
            const buttonText = wrapper.find('button').text();

            expect(buttonText).toBe('Logout');
        });

        it('should use title prop as h1 text', function(){
            const title = 'test title';
            const wrapper = mount(<PrivateHeader title={title} handleLogout={()=>{}}/>)
            const h1 = wrapper.find('h1').text();

            expect(h1).toBe(title);
        });
```

- now back in our test case below lets create another spy

```
    it('should call the function', function(){

        const spy = expect.createSpy();
        spy(2, 34, true);
        spy('jose');
        expect(spy).toHaveBeenCalledWith('jose');
    });

    it('should call handleLogout on click', function(){
        const spy = expect.createSpy();
    });
```

- next up we are going to render header

```
    it('should call handleLogout on click', function(){
        const spy = expect.createSpy();

        const wrapper = mount( <PrivateHeader title="title" handleLogout={spy}/> )
    });
```

- next we need to simulate a button click, we are goign to be using the simulate event,it is an enzyme method

```
    it('should call handleLogout on click', function(){
        const spy = expect.createSpy();

        const wrapper = mount( <PrivateHeader title="title" handleLogout={spy}/> );

        wrapper.find('button').simulate('click');
```

- now lets make assertion

```
        it('should call handleLogout on click', function(){
            const spy = expect.createSpy();

            const wrapper = mount( <PrivateHeader title="title" handleLogout={spy}/> );

            wrapper.find('button').simulate('click');
            expect(spy).toHaveBeenCalled();
        });
```

- lets comment out the test case above it

```
        it('should use title prop as h1 text', function(){
            const title = 'test title';
            const wrapper = mount(<PrivateHeader title={title} handleLogout={()=>{}}/>)
            const h1 = wrapper.find('h1').text();

            expect(h1).toBe(title);
        });

        // it('should call the function', function(){

        //     const spy = expect.createSpy();
        //     spy(2, 34, true);
        //     spy('jose');
        //     expect(spy).toHaveBeenCalledWith('jose');
        // });

        it('should call handleLogout on click', function(){
            const spy = expect.createSpy();

            const wrapper = mount( <PrivateHeader title="title" handleLogout={spy}/> );

            wrapper.find('button').simulate('click');
```

```
        expect(spy).toHaveBeenCalled();
    });
```

-