123-Testing Meteor Publications video

- go to notes.js and check if we are on the server
- and we are going to call publish and call it notes

```
import { Mongo } from 'meteor/mongo';

import { Meteor } from 'meteor/meteor';

import SimpleSchema from 'simpl-schema';


import moment from 'moment';


export const Notes = new Mongo.Collection('notes');


if(Meteor.isServer){

    Meteor.publish('notes')

}


Meteor.methods({

    'notes.insert'(){

        if(!this.userId){

            throw new Meteor.Error('not-authorized');

        }
```

- and it is going to require a function, and we will use ES5 function because we are going to be using the this keywork

```
export const Notes = new Mongo.Collection('notes');


if(Meteor.isServer){

    Meteor.publish('notes', function(){


    })

}
```

- and this function will return the userId

```
if(Meteor.isServer){
    Meteor.publish('notes', function(){
        return Notes.find({userId: this.userId})
    });
}
```

- now over to notes.test.js we are going to be making another seed note, so lets copy the object from the noteOne and paste it below it

```
import { Meteor } from 'meteor/meteor';
import expect from 'expect';
import { Notes } from './notes'

if(Meteor.isServer){
    describe('notes', function(){
       const noteOne = {
            _id: 'testNoteId1',
            title: 'My Title',
            body: 'My body for note',
            updateAt: 0,
            userId: 'testUserId1'
       },
       const noteOne = {
        _id: 'testNoteId1',
        title: 'My Title',
        body: 'My body for note',
        updateAt: 0,
        userId: 'testUserId1'
   }
```

- lets change the name to noteTwo, lets change the value of each property too

```
       const noteOne = {
            _id: 'testNoteId1',
            title: 'My Title',
            body: 'My body for note',
            updateAt: 0,
            userId: 'testUserId1'
        };
        const noteTwo = {
         _id: 'testNoteId2',
         title: 'Things to buy',
         body: 'Couch',
         updateAt: 0,
         userId: 'testUserId2'
    };
```

- **now** lets add this to the database down below

```
     describe('notes', function(){
        const noteOne = {
            _id: 'testNoteId1',
            title: 'My Title',
            body: 'My body for note',
            updateAt: 0,
            userId: 'testUserId1'
        };
        const noteTwo = {
         _id: 'testNoteId2',
         title: 'Things to buy',
         body: 'Couch',
         updateAt: 0,
         userId: 'testUserId2'
    };
         beforeEach(function(){
            Notes.remove({});
            Notes.insert(noteOne);
            Notes.insert(noteTwo);
         })
```

- now lets test out the publication, so lets do a test case below

```
         it('should not update note if invalid _id', function() {
            expect(()=>{
```

```
            Meteor.server.method_handlers['notes.update'].apply({_id: noteOne.userId});
        }).toThrow();
    });

    it('should return a users notes', function(){

    });
```

- **first we are going to access the Notes publication**

```
    it('should return a users notes', function(){
        Meteor.server.publish_handlers['notes']
    });


});
```

- right here since notes doesnt have notes.#### then just using Meteor.server.publish_handlers.notes is allowed

```
    it('should return a users notes', function(){
        Meteor.server.publish_handlers.notes
    });
```

- in order to test this we are going to call it with a userId using apply() like we did before, and we are going to be use the userId from noteOne

```
    it('should not update note if invalid _id', function() {
        expect(()=>{
            Meteor.server.method_handlers['notes.update'].apply({_id: noteOne.userId});
        }).toThrow();
    });

    it('should return a users notes', function(){
        Meteor.server.publish_handlers.notes.apply({userId: noteOne.userId});
    });
    });
}
```

- now lets store this in a variable res for result

```
        it('should return a users notes', function(){
            const res = Meteor.server.publish_handlers.notes.apply({userId: noteOne.userId});
         });
```

- and this is cursor, so were are going to fetch in order to get the actually array of objects

```
        it('should return a users notes', function(){
            const res = Meteor.server.publish_handlers.notes.apply({userId: noteOne.userId});
            const notes = res.fetch();
        });
```

- we are going to expect one note

```
        it('should not update note if invalid _id', function() {
            expect(()=>{
                Meteor.server.method_handlers['notes.update'].apply({_id: noteOne.userId});
            }).toThrow();
        });

        it('should return a users notes', function(){
            const res = Meteor.server.publish_handlers.notes.apply({userId: noteOne.userId});
            const notes = res.fetch();

            expect(notes.length).toBe(1);
        });

    });
}
```

- we are now going to compare two objects, the first one from notes[0] and isEqual to noteOne

```
        it('should return a users notes', function(){
            const res = Meteor.server.publish_handlers.notes.apply({userId: noteOne.userId});
            const notes = res.fetch();


            expect(notes.length).toBe(1);
            expect(notes[0]).toEqual(noteOne);
        });


    });
}
```

- now if you check in the browser i should pass
- lets now make another test case, where a user has no notes, and return zero notes

```


        it('should return zero notes for user that has none', function(){
            const res = Meteor.server.publish_handlers.notes.apply({userId: 'some id'});
            const notes = res.fetch();


            expect(notes.length).toBe(0);
        });


    });
}
```

-
-