121-Testing Note Updating video

- go to notes.js and we are going to be creating a new method called notes.update

```
    'notes.remove'(_id){
        if(!this.userId){
            throw new Meteor.Error('not-authorized');
        }

        new SimpleSchema({
            _id: {
                type: String,
                min: 1
            }
        }).validate({ _id });

        Notes.remove({ _id, userId: this.userId });
    },

    'notes.update'() {

    }

});
```

- this method is going to take two arguments, first one is _id, what id are we trying to update , and the second one is going to be an object updates

```
    'notes.update'(_id, updates) {

    }

});
```

- the first thing we need to do in this method is to check if the user has a userId, so we'll do the samething as notes.remove

```
    'notes.update'(_id, updates) {
        if(!this.userId){
            throw new Meteor.Error('not-authorized');
        }
    }

});
```

- and also SimpleSchema for some validation

```
    'notes.update'(_id, updates) {
        if(!this.userId){
            throw new Meteor.Error('not-authorized');
        }

        new SimpleSchema({

        })
    }

});
```

- next we are also going to define the three things we need to validate, in our schema the title and body are optional

```
    'notes.update'(_id, updates) {
        if(!this.userId){
            throw new Meteor.Error('not-authorized');
        }

        new SimpleSchema({
            _id:{
                type: String,
                min: 1
            },
            title: {
                type: String,
```

```
                optional: true
            },
            body: {
                type: String,
                optional: true
            }
        })
    }

});
```

- and we are going to validate _id and also validate, not too sure why, didn't understand very, well , but we are going to be using the ... spread operator, so apperantly if someone decides to add something more than _id, title and body, this will throw an error, and that's what we want

```
    'notes.update'(_id, updates) {
        if(!this.userId){
            throw new Meteor.Error('not-authorized');
        }

        new SimpleSchema({
            _id:{
                type: String,
                min: 1
            },
            title: {
                type: String,
                optional: true
            },
            body: {
                type: String,
                optional: true
            }
        }).validate({
            _id,
            ...updates
        });
```

```
    }

});
```

- now lets update the note, we are going to be using the spread operator to allow us to customize our updates, and we also want to update our time stamp

```
                optional: true
            },
            body: {
                type: String,
                optional: true
            }
        }).validate({
            _id,
            ...updates
        });

        Notes.update(_id, {

        })
    }
```

- after this we are going to be using the $set operator, and inside $set we want to provide the stuff we want to change, we first want to update the time stamp updatedAt, and we'll use moment again

```
        }).validate({
            _id,
            ...updates
        });

        Notes.update(_id, {
            $set: {
                updatedAt : moment().valueOf(),
            }
        })
    }
```

```
});
```

- next we want to spread out all the updates properties

```
            title: {
                type: String,
                optional: true
            },
            body: {
                type: String,
                optional: true
            }
        }).validate({
            _id,
            ...updates
        });

        Notes.update(_id, {
            $set: {
                updatedAt : moment().valueOf(),
                ...updates
            }
        })
    }

});
```

- now lets start writing some test cases for our method
- we are going to do some restructering over at notes.test.js
- first cut out the object from Notes.insert

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

if(Meteor.isServer){
    describe('notes', function(){
```

```
        beforeEach(function(){
            Notes.remove({});
            Notes.insert({
                _id: 'testNoteId1',
                title: 'My Title',
                body: 'My body for note',
                updateAt: 0,
                userId: 'testUserId1'
            });
        })
```

- and create it up above with a variable

```
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'
       }

        beforeEach(function(){
            Notes.remove({});
            Notes.insert(noteOne);
        })
```

- now we can reference that new variable to our code below

```
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'
       }

        beforeEach(function(){
            Notes.remove({});
            Notes.insert(noteOne);
        })

        it('should insert new note', function(){
            const userId = 'testid';
            const _id = Meteor.server.method_handlers['notes.insert'].apply({ userId });

            expect(Notes.findOne({ _id, userId })).toExist();

        });

        it('should not insert note if not authenticated', function(){
            expect(()=>{
                Meteor.server.method_handlers['notes.insert']();
            }).toThrow();
        });

        it('should remove note', function(){
            Meteor.server.method_handlers['notes.remove'].apply({userId: noteOne.userId}, [noteO
ne._id]);

            expect(Notes.findOne({_id: noteOne._id})).toNotExist();
        });
```

```
        it('should not remove note if unauthenticated', function(){
            expect(()=>{
                Meteor.server.method_handlers['meteor.remove'].apply({},[noteOne._id]);
            }).toThrow();
        });

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

- now lets do some test cases below to our notes.update method that we defined over in notes.js
- so lets go to notes.test.js
- lets start with the success case
- first lets do it function , and as an argument it takes a string and a function, in the function we are calling our notes.update

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

        it('should update note', function(){
            Meteor.server.method_handlers['notes.update'].apply();
        })
    });
}
```

- in the apply methods we are going to pass an object to setup the userId

```
        it('should not remove note if unauthenticated', function(){
            expect(()=>{
                Meteor.server.method_handlers['meteor.remove'].apply({},[noteOne._id]);
```

```
            }).toThrow();
        });

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

        it('should update note', function(){
            Meteor.server.method_handlers['notes.update'].apply({
                userId: noteOne.userId
            });
        })
    });
}
```

- now are going to provide a second argument to apply, an array where each item in the array is a new argument, as you notice we also created a title const

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

        it('should update note', function(){
            const title = "This is an updated title"
            Meteor.server.method_handlers['notes.update'].apply({
                userId: noteOne.userId
            }, [
                noteOne._id,
                { title }
            ]);
        })
```

```
    });
}
```

- next want to fetch the note

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

        it('should update note', function(){
            const title = "This is an updated title"
            Meteor.server.method_handlers['notes.update'].apply({
                userId: noteOne.userId
            }, [
                noteOne._id,
                { title }
            ]);

            const note = Notes.findOne(noteOne._id);
        })
    });
}
```

- we then are going to be a using an expect method, to check if our timestamp has been updated, we are also going to be using toInclude(), this works with arrays and objects, it allows you to check that something is inside of something else

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

        it('should update note', function(){
```

```
            const title = "This is an updated title"
            Meteor.server.method_handlers['notes.update'].apply({
                userId: noteOne.userId
            }, [
                noteOne._id,
                { title }
            ]);

            const note = Notes.findOne(noteOne._id);

            expect(note.updatedAt).toBeGreaterThan(0);
        })
    });
}
```

- the next expect will have a note to include something

```
        it('should update note', function(){
            const title = "This is an updated title"
            Meteor.server.method_handlers['notes.update'].apply({
                userId: noteOne.userId
            }, [
                noteOne._id,
                { title }
            ]);

            const note = Notes.findOne(noteOne._id);

            expect(note.updatedAt).toBeGreaterThan(0);
            expect(note).toInclude({
                title,
                body: noteOne.body
            });
        })
    });
}
```

- back in your browser it should have passed all cases

- now we are going to create another it case but this for errors

```
            ]);

            const note = Notes.findOne(noteOne._id);

            expect(note.updatedAt).toBeGreaterThan(0);
            expect(note).toInclude({
                title,
                body: noteOne.body
            });
        });

        it('should throw error if extra updates provided', function(){



        })
    });
}
```

- next up we are calling the expect method with toThrow so it does throw an error if there is one

```
            const note = Notes.findOne(noteOne._id);

            expect(note.updatedAt).toBeGreaterThan(0);
            expect(note).toInclude({
                title,
                body: noteOne.body
            });
        });

        it('should throw error if extra updates provided', function(){

            expect(()=>{

            }).toThrow();
```

```
        })
    });
}
```

- and now let call the notes.update but with added stuff to it

```
            const note = Notes.findOne(noteOne._id);

            expect(note.updatedAt).toBeGreaterThan(0);
            expect(note).toInclude({
                title,
                body: noteOne.body
            });
        });

        it('should throw error if extra updates provided', function(){
            const name = 'jose'
            expect(()=>{
                Meteor.server.method_handlers['notes.update'].apply({
                    userId: noteOne.userId
                }, [
                    noteOne._id,
                    { title: 'new title', name: 'jose' }
                ]);
            }).toThrow();
        })
    });
}
```

-