Creating a MongoDB and MIniMongo Collection

- let's first delete delete math.js and utils.js from the imports folder
- create a folder called api inside of imports and inside of api create a file called players.js, this file will be responsible for creating the collection from the MongoDB database

1. Import MandoDB in players.js, we want this Named export. it is an object

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

- then we need this Mongo contructor

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


export const Players = new Mongo.Collection('players');
```

- Now head over to main.js and let's export Players

```
import {Players} from './../imports/api/players';
```

- we then need to to make a call to Meteor.startup because we want Meteor to be completely ready before we do anything
- so let's import it first it

```
import {Meteor} from 'meteor/meteor';
import {Players} from './../imports/api/players';


Meteor.startup(function (){


});
```

- we next are going to call a method called insert() that takes an object that takes a partial document

```
import {Meteor} from 'meteor/meteor';
import {Players} from './../imports/api/players';


Meteor.startup(function (){
    Players.insert({
        name: 'Jose',
        score: 3
```

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

we next want to fetch the players back, that is going to let us know the player data was actually inserted

```
import {Meteor} from 'meteor/meteor';
import {Players} from './../imports/api/players';

Meteor.startup(function (){
    Players.insert({
        name: 'Jose',
        score: 3
    });
    console.log(Players.find());
});
```

find() returns a cursor, which means a pointer to some documents in database, to actually get an array of documents we need to use the fetch() method as well...

```
import {Meteor} from 'meteor/meteor';
import {Players} from './../imports/api/players';

Meteor.startup(function (){
    Players.insert({
        name: 'Jose',
        score: 3
    });
    console.log(Players.find().fetch());
});
```

go over to the terminal and open up a new tab and type

```
 meteor mongo --release 1.4.2.1
```

- when I run this is going to check if a database server is actually up and if it is up is going to connect to it
- now run the next line

```
meteor:PRIMARY> db.players.find()
```

- And returns my array of objects(for some reason it return two of the same

```
{ "_id" : "SdNyMyisWGh6A3SsT", "name" : "Jose", "score" : 3 }
{ "_id" : "ycXxuEc3cnivqsRkR", "name" : "Jose", "score" : 3 }
```

Quering Data on the Client with MiniMongo

- we need to find a way to access the minimongo methods
- head over to client main.js and import Players once again

```
import React from 'react';
import ReactDOM from 'react-dom';
import {Meteor} from 'meteor/meteor';

import {Players} from './../imports/api/players';

const players = [{
  _id: '1',
    name: 'Lauren',
    score: 99
```

and also add the console log from the server main.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import {Meteor} from 'meteor/meteor';

import {Players} from './../imports/api/players';
console.log('Players list ', Players.find().fetch());

const players = [{
  _id: '1',
    name: 'Lauren',
    score: 99
```

so the above returned an empty array of objects and that is because when called find() the data hadn't reached MongoMini yet, so for now we are going to use a hacky way to prevent this but adding the setTimeout function to delay this effect

```
import React from 'react';
import ReactDOM from 'react-dom';
import {Meteor} from 'meteor/meteor';

import {Players} from './../imports/api/players';

setTimeout(function(){
  console.log('Players list ', Players.find().fetch());
}, 1000);
```

- and this will return the array of objects
- using setTimeout is not a good idea, instead we are going to use tracker by Meteor
- tracker lets you track queries and re run code when does queries change
- so let's import it in our client main.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import {Meteor} from 'meteor/meteor';
import {Tracker} from 'meteor/tracker';

import {Players} from './../imports/api/players';
```

- nows let implement tracker in our code below, so intead of using setTimeout we will be using Tracker.autorun() and this takes a function

```
import React from 'react';
import ReactDOM from 'react-dom';
import {Meteor} from 'meteor/meteor';
import {Tracker} from 'meteor/tracker';

import {Players} from './../imports/api/players';

Tracker.autorun(function(){
  console.log('Players list ', Players.find().fetch());
});
```

now we are going to render in on the screen
so first we'll make a variable that is going to hold the array of objects that Players has, put we also want to use MongoDB Tracker so we do this...

```
Tracker.autorun(function(){
    let players = Players.find().fetch();
```

and now we just add the remaining code below that we want to render on the screen

```
Meteor.startup(function(){


  Tracker.autorun(function(){
    let players = Players.find().fetch();
    let title = 'Score Keep';
    let name = 'Jose';
    let jsx = (
    <div>
      <h1>{title}</h1>
      <p>Hi my name is {name}!</p>
      <p>This is my second p</p>
      {renderPlayers(players)}
    </div>
    );
    ReactDOM.render(jsx, document.getElementById('app'))
  });


});
```

we now want to insert a new document so headover to client main.js and right below are Tracker.autorun

```
Players.insert({
    name: 'Jen',
    score: 34
  });
```

and let's check it out on the terminal by typing

```
meteor mongo --release 1.4.2.1
```

```
db.players.find()
```

Inserting Players via a Form

- now we are going to start doing our form, so go to client main.js
- when the form submits it will go to a function handleSubmit(), remember that when you put curly braces in jsx code, you can put plain vanilla js

```
      <form onSubmit = {handleSubmit}>
        <input type="text" name="playerName" placeholder="Player name"/>
        <button>Add Player</button>
      </form>
```

- so lets create that function above the Meteor.startup

```
const handleSubmit = function(e){
  e.preventDefault();
}
Meteor.startup(function(){
```

- in the function we passed e to the function and called the method prevenDefault() so the form wont submit
- we next want to have a functionality so when the user types in their name we can grab that value...
- so let's create a variable where we can store the value of the text field

```
const handleSubmit = function(e){
  let playerName = e.target.playerName.value;
  e.preventDefault();
}
```

- so in a form we can access any of its inputs by their name and then we can use the value prop
- now lets verify that the variable string is not an empty string
- once that is confirmed we don't need to show that value in our text field anymore so we do this

```
const handleSubmit = function(e){
  let playerName = e.target.playerName.value;
  e.preventDefault();
  if(playerName){
    e.target.playerName.value = '';
```

```
  }
}
```

- we then want to remove the static data that we have below

```
    ReactDOM.render(jsx, document.getElementById('app'))
  });
  Players.insert({
    name: 'Jen',
    score: 34
  });

});
```

- and we want that in our handleSubmit above

```
const handleSubmit = function(e){
  let playerName = e.target.playerName.value;
  e.preventDefault();
  if(playerName){
    e.target.playerName.value = '';
    Players.insert({
      name: playerName,
      score: 0
    });
  }
}
```

- so we set name: to our variable playerName
- let's go to the terminal so we can wipe out all of the data that we stored
- first hit control c to get out of meteor in the terminal

```
meteor reset --release 1.4.2.1
```

- now restart the application

```
meteor run --release 1.4.2.1
```

-