Page-03-Score Keeper App

ES6 Aside - Arrow Function

- head over server main.js
- we are going to compare the two types of functions
- so lets first do the ES5 version

```
Meteor.startup(function (){
    // Players.insert({
    //     name: 'carla',
    //     score: 33
    // });
    // console.log(Players.find().fetch());
    let square = function  (x) {
        return x * x;
    }
    console.log(square(10));
});
```

we can also do a ES5 named function

```
 function square (x) {
        return x * x;
    }
    console.log(square(10));
```

- these two will give you the same result 100,
- arrow functions on the other hand don't support named functions, they all are annonymous

```
    let square = (x) => {
        return x * x;
    }
    console.log(square(10));
});
```

- this will give you the same result 100

- one of the reasons we want to use arrow functions is of their simplicity, when you have one statement in you function you can use like so..

```
let square = (x) =>  x * x;
```

in the above notice that we dont need the curly braces or the return keyword
we are going to take a look where arrow functions don't work

```
 let user = {
        name: 'Jose',
        sayHi: ()=>{
            console.log(this.name);
        }
    };
    user.sayHi();
```

in the above if we try to use an arrow function instead of the named function we will see 'undefined' because 'this' doesn't bind, and this also happens when you want to check for arguments

```
    let user = {
        name: 'Jose',
        sayHi: () =>{
            console.log(arguments);
        }
    };
    user.sayHi(1, 5);
});
```

the above will return an empty object {}
but there is a shortcut with ES6 to avoid this issue, with the following code this will work

```
    let user = {
        name: 'Jose',
        sayHi () {
            console.log(arguments);
        }
    };
    user.sayHi(1, 5);
});
```

we removed the colon : and the arrow =>
and the following will also work and return the name value

```
    let user = {
        name: 'Jose',
        sayHi () {
            console.log(this.name);
        }
    };
    user.sayHi();
});
```

so the above is an ES5 function and that is why the this works
but if you want to do a setTimeout function we can use this keyword because the this key word references to the parent, for example

```
    let user = {
        name: 'Jose',
        sayHi () {
            setTimeout(()=>{
                console.log(this.name);//this right is referencing to the user.name not to the s
ayHi function
            }, 1000);
        }
    };
    user.sayHi();
});
```

Arrow Function Refactor

delete everything from the server main.js, just leave this, and also we are going to be changing some functions to arrow functions so let's start with this one

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


Meteor.startup(()=>{
});
```

now let's convert the functions in the client main.js to arrow functions. so now the client main.js should look like this

```
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';


const renderPlayers = (playerList) => {

  return playerList.map((player) => {

    return <p key={player._id}>{player.name} has {player.score} point(s)</p>

  });

};

const handleSubmit = (e) => {

  let playerName = e.target.playerName.value;

  e.preventDefault();

  if(playerName){

    e.target.playerName.value = '';

    Players.insert({

      name: playerName,

      score: 0

    });

  }

}

Meteor.startup( () => {


  Tracker.autorun(() => {

    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)}
        <form onSubmit = {handleSubmit}>
          <input type="text" name="playerName" placeholder="Player name"/>
          <button>Add Player</button>
        </form>
      </div>
      );
      ReactDOM.render(jsx, document.getElementById('app'))
    });
  });
```

Removing Player Document

go over to your terminal and open up a new tab, and we are going to run mongo to connect

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

to check what you have in mongo players databse type this

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

to find a specific document you use this, for example if you want to find some one with name of Jen

```
meteor:PRIMARY> db.players.find({name:'jen'})
{ "_id" : "kWL7pC5FX2ETbSSsB", "name" : "jen", "score" : 0 }
```

- and that will return that specific document
- we can also do that we with score

```
db.players.find({score: 0});
```

and that will return all scores with 0s
to remove documents you use the remove method lke so... (remove method can not be called with no arguments)

```
db.players.remove({_id:'mapjiNntSFiBQXLzZ'})
```

to remove all document in the players database you would just use an empty object like so..

```
db.players.remove({})
```

so now are going to create a button in our app to remove the target clicked
go client main.js
first lets modify the renderPlayers function by adding () because we are going to be adding more code there

```
const renderPlayers = (playerList) => {
  return playerList.map((player) => {
    return (
    <p key={player._id}>
      {player.name} has {player.score} point(s)
    </p>
  );
  });
};
```

let's add a button

```
const renderPlayers = (playerList) => {
  return playerList.map((player) => {
    return (
    <p key={player._id}>
      {player.name} has {player.score} point(s).
      <button>X</button>
    </p>
  );
```

now lets add an onClick event to that button, we can use an arrow function right there in jsx to test it out

```
const renderPlayers = (playerList) => {
  return playerList.map((player) => {
    return (
    <p key={player._id}>
      {player.name} has {player.score} point(s).
      <button onClick={()=> alert('button pushed')}>X</button>
    </p>
    );
  });
};
```

now let's try to remove the document from the Players db

```
const renderPlayers = (playerList) => {
  return playerList.map((player) => {
    return (
    <p key={player._id}>
      {player.name} has {player.score} point(s).
      <button onClick={()=> Players.remove({_id: player._id})}>X</button>
    </p>
  );
  });
};
```

Updating Player Documents

go ahead and start meteor mongo in a different tab

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

and just to check what I have in my database

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

so what we want to do is update the score every time we click a button, mongo has an update method this method takes two arguments, the first one {} we want to target the documents we want to to update and the second one we specify the update, so in this case we are going to target the _id and in the second argument we are going to change/update the name

```
db.players.update({_id:'hP5NcsWWrDk7q88Fq'},{name: 'Mike'})
```

the result of that is that it will replace the whoe doc with just name: "Mike" and will get rid of everything else

```
{ "_id" : "hP5NcsWWrDk7q88Fq", "name" : "Mike" }
{ "_id" : "jT5H6ArgqNyXe93fw", "name" : "josh", "score" : 0 }
```

to learn more about this google mongodb update operators
scroll own to Fields operators, we are going to be using the $set operator takes an argument and we specify a key value pairs

```
db.players.update({_id:'hP5NcsWWrDk7q88Fq'},{$set:{score: 0}})
```

and the result of this, is what we are expecting

```
meteor:PRIMARY> db.players.find()
{ "_id" : "hP5NcsWWrDk7q88Fq", "name" : "Mike", "score" : 0 }
{ "_id" : "jT5H6ArgqNyXe93fw", "name" : "josh", "score" : 0 }
```

now we are going to use increment mongo operator $inc

```
db.players.update({_id:'hP5NcsWWrDk7q88Fq'},{$inc:{score: 1}})
```

notice that we are not setting the score to 1 we are incrementing by 1
so now lets go back to our client main.js and add a button

```
const renderPlayers = (playerList) => {
  return playerList.map((player) => {
    return (
    <p key={player._id}>
      {player.name} has {player.score} point(s).
      <button>+1</button>
      <button onClick={()=> Players.remove({_id: player._id})}>X</button>
    </p>
  );
  });
};
```

the we add an onclick event with an arrow function

```
const renderPlayers = (playerList) => {
  return playerList.map((player) => {
    return (
    <p key={player._id}>
      {player.name} has {player.score} point(s).
      <button onClick={()=>Players.update({_id: player._id}, {$inc:{score:1}})}>+1</button>
      <button onClick={()=> Players.remove({_id: player._id})}>X</button>
    </p>
  );
  });
};
```

so now we add another button to decrement by 1so we just do a -1 like so...

```
  const renderPlayers = (playerList) => {
    return playerList.map((player) => {
      return (
      <p key={player._id}>
        {player.name} has {player.score} point(s).
        <button onClick={()=>Players.update({_id: player._id}, {$inc:{score:1}})}>+1</button>
        <button onClick={()=>Players.update({_id: player._id}, {$inc:{score:-1}})}>-1</button>
        <button onClick={()=> Players.remove({_id: player._id})}>X</button>
      </p>
    );
    });
  };
```

there is actually a shortcut to the code above, you don't need to specify the _id: so we can use remove that and it'll work just as well

```
  const renderPlayers = (playerList) => {
    return playerList.map((player) => {
      return (
      <p key={player._id}>
        {player.name} has {player.score} point(s).
        <button onClick={()=>Players.update(player._id, {$inc:{score:1}})}>+1</button>
        <button onClick={()=>Players.update(player._id, {$inc:{score:-1}})}>-1</button>
        <button onClick={()=> Players.remove(player._id)}>X</button>
      </p>
    );
    });
  };
```