Publish and Subscribe Methods

- As already discussed in the Collections, all of our data is available on the client side. This is a security issue that can be handled with publish and subscribe methods.

Removing Autopublish

- In this example, we will use **PlayersCollection** collection with the following data in browser console (client side).

- To secure our data, we need to remove **autopublish** package that was allowing us to use the data on the client side.

C:\Users\username\Desktop\meteorApp>meteor remove autopublish

- After this step, we will not be able to get the database data from the client side. We will only be able to see it from the server side in the command prompt window. Checkout the following code –

meteorApp.js

```
var PlayersCollection = new Mongo.Collection('playersCollection');
var myLog = PlayersCollection.find().fetch(); console.log(myLog);
```

- The **command prompt** window will show the entire collection with four objects, while the **developers console** will show an empty array. **Now our app is more secure**.

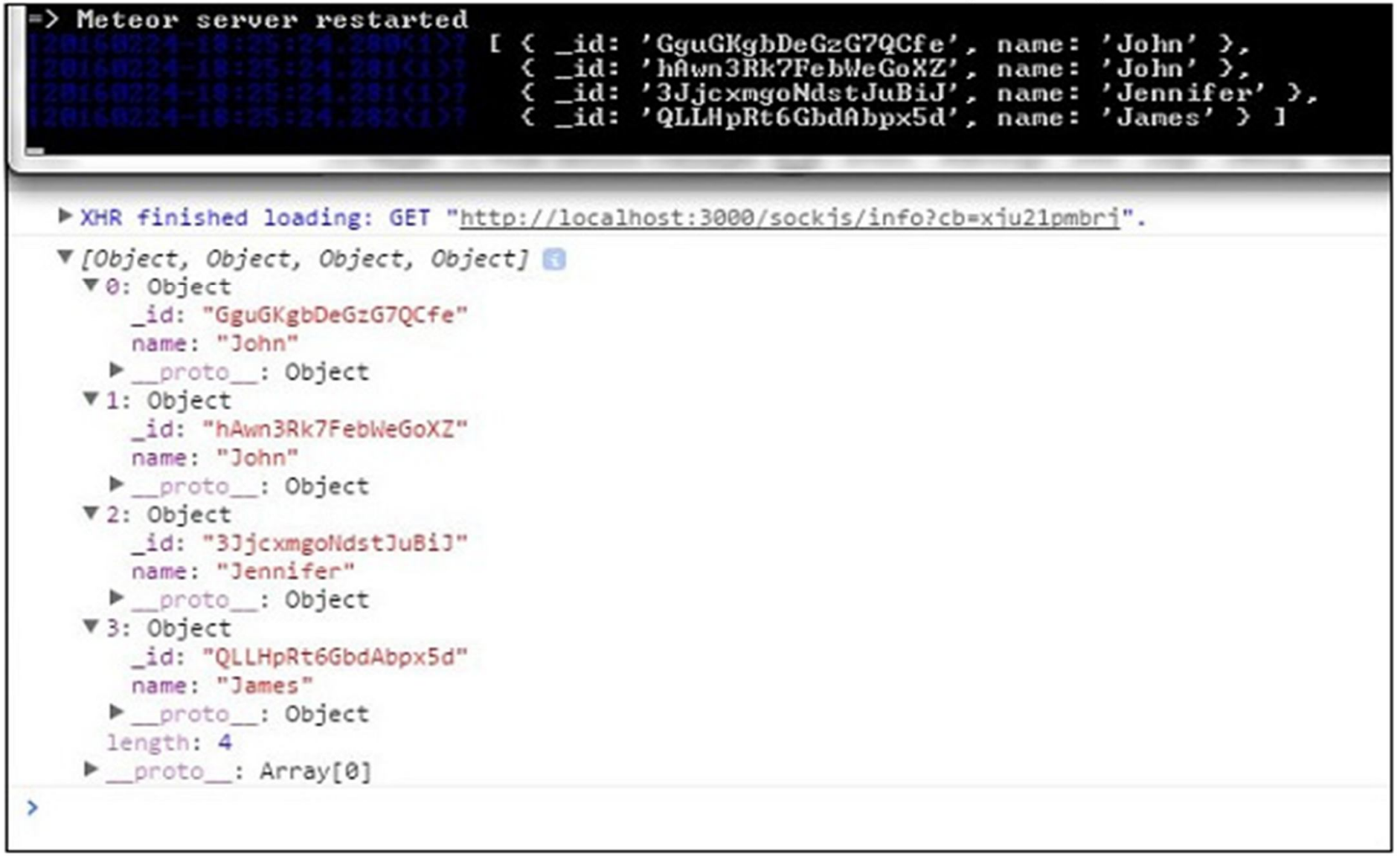
Using Publish and Subscribe

- Let's say we want to allow the clients to use our data.
- For this, we need to create **Meteor.publish()** method on the server. This method will send the data to the client.
- To be able to receive and use that data on the client side, we will create **Meteor.subscribe()** method.
- At the end of the example, we are searching the database. This code is running on both the client and the server side.

```
var PlayersCollection = new
Mongo.Collection('playersCollection');
if(Meteor.isServer) {
        Meteor.publish('allowedData', function() {
                return PlayersCollection.find(); }) }
 if (Meteor.isClient) {
        Meteor.subscribe('allowedData'); };
Meteor.setTimeout(function() {
        var myLog = PlayersCollection.find().fetch();
        console.log(myLog); }, 1000);
```

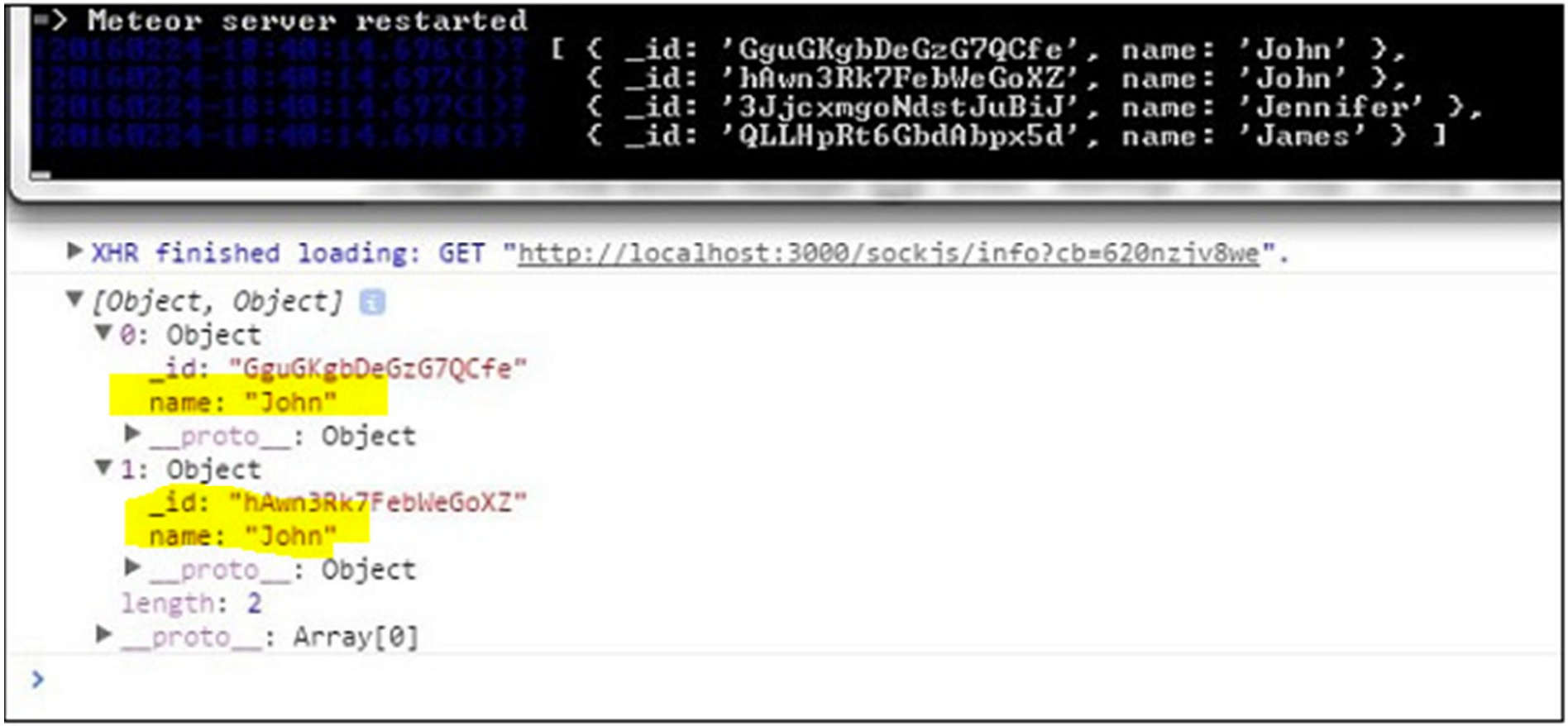
```
return PlayersCollection.find({name: "John"});
```

We can see that our data is logged in both the **developers console** and the **command prompt** window.



Filtering Client Data

- We can also publish part of the data. In this example, we are publishing data with **name = "John"**.



Accounts

- When working with most frameworks, for instance, creating a user accounts system might start by creating place to store the data of users:

```
UserAccounts = new Mongo.Collection('users');
```

- Then writing application logic for registration, logging-in, recovering lost passwords, and all those other standard features of account systems

Accounts

- This package allows complete user authentication functionality. You can add it by running the following code in the command prompt window.
- >meteor add accounts-password
- adding this "**accounts-password**" package to the project. This package creates the back-end for an accounts system that relies on an email and password for registration
- added the "accounts-password" package to our project, a collection was automatically created to store the data of registered users.
  - This collection is known as **Meteor.users**, and it works just like any collection that we might create ourselves.

Meteorjs.html

```
<head>
  <title>meteorApp</title>
</head>
 <body>
  {{#if currentUser}}
    {{> home}}
    {{else}}
    {{> register}}
    {{> login}}
  {{/if}}
</body>
<template name = "register">
  <h2>REGISTER:</h2>
  <form>
    <input type = "email" name = "registerEmail"><br>
    <input type = "password" name = "registerPassword"><br>
    <input type = "submit" value = "Register"><br>
  </form>
</template>
```

```
<template name = "login">
  <h2>LOGIN:</h2>
  <form>
    <input type = "email" name = "loginEmail"><br>
    <input type = "password" name="loginPassword"><br>
    <input type = "submit" value = "Login"><br>
  </form>
</template>

<template name = "home">
  <p>You're logged in.</p>
  <button class = "logout">Logout</button>
</template>
```

```
if (Meteor.isClient) {
 Template.register.events({
  'submit form': function(event) {
event.preventDefault();
var registerData = {
email: event.target.registerEmail.value,
password: event.target.registerPassword.value        }
Accounts.createUser(registerData, function(error) {
if (Meteor.user()) {              console.log(Meteor.userId());          } else {
console.log("ERROR: " + error.reason);          }
});     }   });
 Template.login.events({         'submit form': function(event) {         event.preventDefault();
var myEmail = event.target.loginEmail.value;         var myPassword =
event.target.loginPassword.value;
Meteor.loginWithPassword(myEmail, myPassword, function(error) {          if
(Meteor.user()) {              console.log(Meteor.userId());          } else {
console.log("ERROR: " + error.reason);          }        });     }   });
Template.home.events({      'click .logout': function(event) {         event.preventDefault();
                                    Meteor.logout(function(error) {          if(error) {
console.log("ERROR: " + error.reason);          }        });     }   });}
```

References

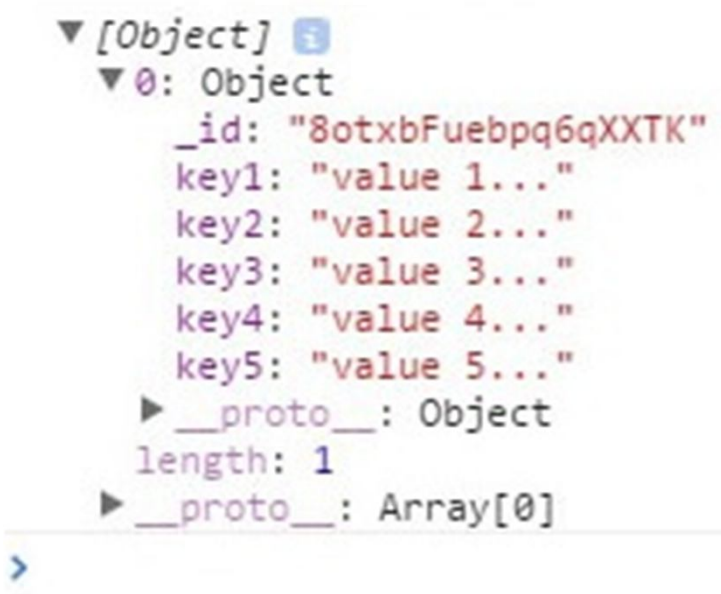
- https://scotch.io/tutorials/learn-meteor-js-from-scratch-build-a-polling-app
- https://www.tutorialspoint.com/meteor/index.htm
- https://www.meteor.com/tutorials/react/creating-an-app

Appendix

MongoDB – insert & find

meteorApp.js

```
MyCollection = new Mongo.Collection('myCollection');
var myData = { key1: "value 1…", key2: "value 2…", key3: "value 3…",
key4: "value 4…", key5: "value 5…" }
MyCollection.insert(myData);
var findCollection = MyCollection.find({key1: "value 1…"}).fetch();
console.log(findCollection);
```



MongoDB - Update

```
MyCollection = new Mongo.Collection('myCollection');
var myData = { key1: "value 1…", key2: "value 2…", key3: "value 3…", key4: "value 4…", key5: "value 5…" }
MyCollection.insert(myData);
var findCollection = MyCollection.find().fetch();
var myId = findCollection[0]._id;
var updatedData = { key1: "updated value 1…", key2: "updated value 2…", key3: "updated value 3…", key4: "updated value 4…", key5: "updated value 5…" }
MyCollection.update(myId, updatedData);
var findUpdatedCollection = MyCollection.find().fetch();
console.log(findUpdatedCollection);
```