Using MongoDB as realtime DB with nodeJS.

Sharath Vignesh Follow
Oct 3, 2018 · 3 min read

Have you ever run into a scenario where you wanted to update your UI as soon as there is some change to your Mongo database ? For example, A new user gets added and you wanted that change to reflect in you UI without having to make an API call or constantly polling for changes. If so, then this post is for you.

While the go to would be to use real time databases like firebase or RethinkDB, you actually can achieve this using your MongoDB too and it is pretty simple. MongoDB has something called `change streams` that allows you to listen to your database.

The method I'm going to show uses `replicaSets` , you can also achieve the same using `sharded clusters.` (click those links to read more about replicasets and sharded clusters)

- So, the first step would be to convert your stand alone MongoDB to replica sets. This is because `change streams` is not available with standalone MongoDB (Honestly I was about to give up on seeing that, but trust me it's simple).

```
$ mongod --port 27017 --replSet rs0
```

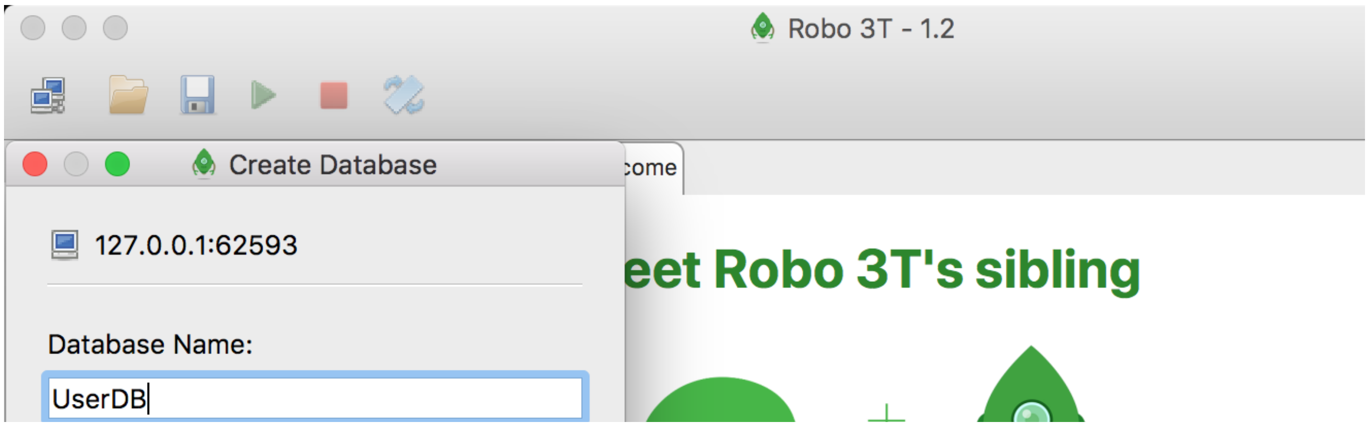
PS: Stop your current standalone mongo and then run the above command.

The above command starts your mongo instance as a `replicaSet` named as rs0.

- Now with your mongo running as `replicaSet,` before creating a new database we must initiate our replication set. To do so, open a new window and do the following

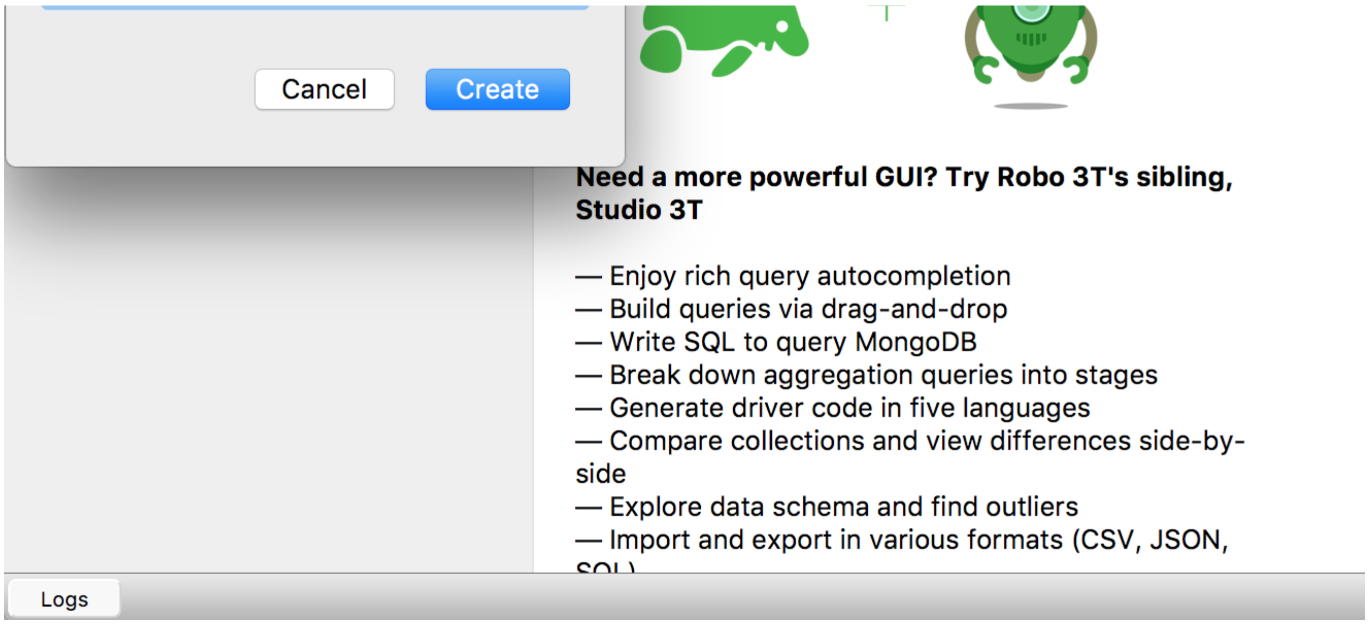
```
$ mongo
$ rs.initiate()
```

- Now let's go and create our database. I use Robo 3T to manage my databases, you can create it in terminal using `$use <Database_name>` . I'm creating a db called UserDB.



127.0.0.1:62593

Database Name: UserDB



Need a more powerful GUI? Try Robo 3T's sibling, Studio 3T

— Enjoy rich query autocompletion
— Build queries via drag-and-drop
— Write SQL to query MongoDB
— Break down aggregation queries into stages
— Generate driver code in five languages
— Compare collections and view differences side-by-side
— Explore data schema and find outliers
— Import and export in various formats (CSV, JSON, SQL)

Create a new Database

- Now our DB part is set. Scaffold an Express application and install Mongoose. My `User` model looks like this,

```
var mongoose = require("mongoose");
var Schema = mongoose.Schema;

var UserSchema = new Schema({
  name: {
    type: String,
    required: [true, 'Need User name']
  }
});

var User = mongoose.model('User', UserSchema);

module.exports = User;
```

User.js hosted with ❤ by **GitHub** **view raw**

User model

- Now assume I have two users inside my `User collection` of `UserDB`. Now I want to subscribe to my `UserDB` for any changes on `User collection`. In your `app.js` code, the mongoDB connection string should look like this,

```
require('./models/connection').connectMongoDB('mongodb://localhost/Us
erDB?replicaSet=rs0');
```

I’m passing my dbURL to my `connection` module. (we need to explicitly mention the replicaSet).

- Now you can subscribe to you userDB using `changeStreams` so as to send those changes immediately on to the front end. Something like below,
  P.S: Do not put the change stream code inside io.onConnection, the CPU usage will shoot as high as 95%.

```
const socket_io = require('socket.io');
var io = socket_io();
const User = require('../models/User');

const changeStream = User.watch();

changeStream.on('change', (change) => {
    console.log(change); // You could parse out the needed info and send only that data.
    io.emit('changeData', change);
});

io.on('connection', function () {
    console.log('connected');
});

var socket = io;
module.exports = socket;
```

socket.js hosted with ❤ by GitHub view raw

I listen to the changes in `User` collection using the `watch()` method, and whenever there is any change, the `change` event listener provided by `changeStreams` gets fired. It accepts a callback function that receives the changed data as parameter.

We can go ahead and test it to see if it works. Run an update operation on `User` collection and at the same time in another tab watch for the console.log messages of your app. You can see the changes real time. At this place, We can now forward this to the front end using `socket.io`.

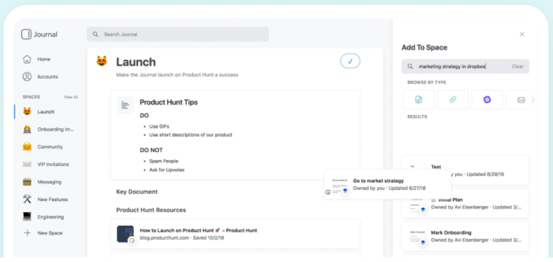
You’re mongoDB is now acting as a real time DB 😉.

. . .

Thank you for reading. If you find something wrong or better ways to do it, let me know in the comments below.

If you like the post, hit the 👏 button below so that others may find it useful. You can follow me on Twitter.

. . .



Mongodb Nodejs Database Realtime Database JavaScript