aerrity / **client.js**

Last active last month

Star

Code Revisions 16 Stars 129 Forks 24

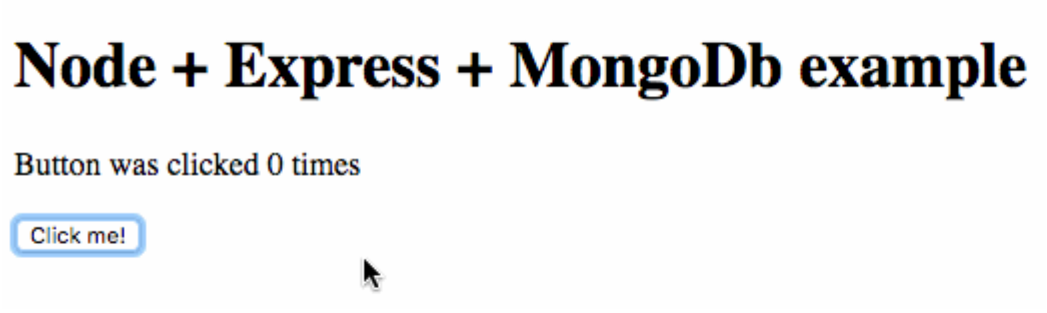
Node, Express & MongoDB: Button click example

readme.md

Node, Express & MongoDB: Button click example

This example assumes you already have node.js installed and are comfortable with JavaScript ES6.

In this example you will build a web page with a single button on it. When this button is clicked this will be recorded in a MongoDB collection. The web page will display an updated count of the number of times the button has been clicked by anyone, anywhere on the web.



Node + Express + MongoDb example

Button was clicked 0 times

Refer to the lecture notes and in-class demonstrations for further information on the technologies and concepts covered here.

Part 1 - Setting up your node project

Create a folder named `node-express-mongo` :

```
mkdir node-express-mongo
```

Change directory to this folder:

```
cd node-express-mongo
```

Run `npm init` to use the node package manager (NPM) to setup a new node project. You can press enter to accept the default settings, but when you get to `entry point:` type in `server.js` .

The node project has now been initialised and you should see a `package.json` file in your folder. This file stores the project settings and will be updated by npm as you add/remove project dependencies.

Now we need to install express:

```
npm install express --save
```

and the MongoDB driver:

```
npm install mongodb@2.2.33 --save
```

Note: the above installs a specific version of the mongodb driver. *If* you wish to use the latest version of the MongoDB driver you will need to modify the code below as described here.

Create an empty file named `server.js` in your folder. Also, create a folder named `public` and put empty files named `index.html` and `client.js` in that folder. The `public` folder will store our client-side code. `server.js` will contain our server-side code.

Project structure:

```
node-express-mongo/package.json
node-express-mongo/server.js
node-express-mongo/public/index.html
node-express-mongo/public/client.js
```

Part 2 - Handling button clicks and serving files

Edit your files to contain the following:

`index.html`

```
<!DOCTYPE html>
<html>
```

```
    <head>
      <meta charset="utf-8">
      <title>Node + Express + MongoDb example</title>
    </head>
    <body>
      <h1>Node + Express + MongoDb example</h1>
      <p id="counter">Loading button click data.</p>
      <button id="myButton">Click me!</button>
    </body>
    <script src="client.js"></script>
  </html>
```

client.js

```
console.log('Client-side code running');

const button = document.getElementById('myButton');
button.addEventListener('click', function(e) {
  console.log('button was clicked');
});
```

server.js

```
console.log('Server-side code running');

const express = require('express');

const app = express();

// serve files from the public directory
app.use(express.static('public'));

// start the express web server listening on 8080
app.listen(8080, () => {
  console.log('listening on 8080');
});

// serve the homepage
app.get('/', (req, res) => {
  res.sendFile(__dirname + '/index.html');
});
```

You can now run your server code using `node server.js`.

You will need to close (CTRL+C) the server and restart it each time you make changes to `server.js`. During development this can become a pain. To make this process easier install nodemon, a tool that will monitor your code for changes and automatically restart the server when necessary. To install nodemon:

```
npm install -g nodemon
```

You can then run `nodemon server.js` to run your server and have it reloaded when you make changes.

Try it out by going to http://localhost:8080. When you click the button it should log to the console.

Part 3 - Connecting to MongoDB

This assumes you have already have access to a server running MongoDB. There are a few options you could use for this:

1. Install MongoDB locally on your machine following these instructions. Note: you are required to create a folder `/data/db` on Mac or `C:\data\db` on Windows to act as a data store for MongoDB. Once installed you need to have `mongod` running on your machine and listening for connections. You can use Compass to manage the data in MongoDB. You should create database that contains a collection named `clicks`.
2. Use the MongoDB database on the IADT server named daneel. First, use Compass to connect to daneel and create a collection named `clicks` under your database. You should use your student number as username and password (lowercase `n`). Then use the connection URL `mongodb://n000xyz:n000xyz@daneel` from your JavaScript code (where `n000xyz` is your student number).
3. Create a free account at mLab. Once you have created an account you should created a database containing a collection named `clicks` and add a new user with access to this collection. *Note:* you will be unable to access mLab from within IADT as traffic is blocked by the firewall 😞. Again, you can test that this is working by connecting using Compass.

Once you have got one of the above setup and verified that you can connect using Compass, you can proceed to connecting from your server-side JavaScript code.

Require the mongodb module and add the MongoDB connection code to `server.js`:

```
console.log('Server-side code running');
```

```
const express = require('express');
const MongoClient = require('mongodb').MongoClient;
const app = express();

// serve files from the public directory
app.use(express.static('public'));

// connect to the db and start the express server
let db;

// ***Replace the URL below with the URL for your database***
const url =  'mongodb://user:password@mongo_address:mongo_port/databaseN
// E.g. for option 2) above this will be:
// const url =  'mongodb://localhost:21017/databaseName';

MongoClient.connect(url, (err, database) => {
  if(err) {
    return console.log(err);
  }
  db = database;
  // start the express web server listening on 8080
  app.listen(8080, () => {
    console.log('listening on 8080');
  });
});

// serve the homepage
app.get('/', (req, res) => {
  res.sendFile(__dirname + '/index.html');
});
```

If you have done this correctly there will be no errors from Node.

Part 4 - Recording button clicks in the DB

Let's modify `client.js` so that it sends a HTTP POST request to the http://localhost:8080/clicked API endpoint on our server. Note: we are using `fetch()` here as covered in a recent lecture, we could also use XHR, jQuery, await + async, or a similar means of making an asynchronous request.

`client.js`

```
console.log('Client-side code running');

const button = document.getElementById('myButton');
button.addEventListener('click', function(e) {
  console.log('button was clicked');
```

```
    fetch('/clicked', {method: 'POST'})
      .then(function(response) {
        if(response.ok) {
          console.log('Click was recorded');
          return;
        }
        throw new Error('Request failed.');
      })
      .catch(function(error) {
        console.log(error);
      });
  });
```

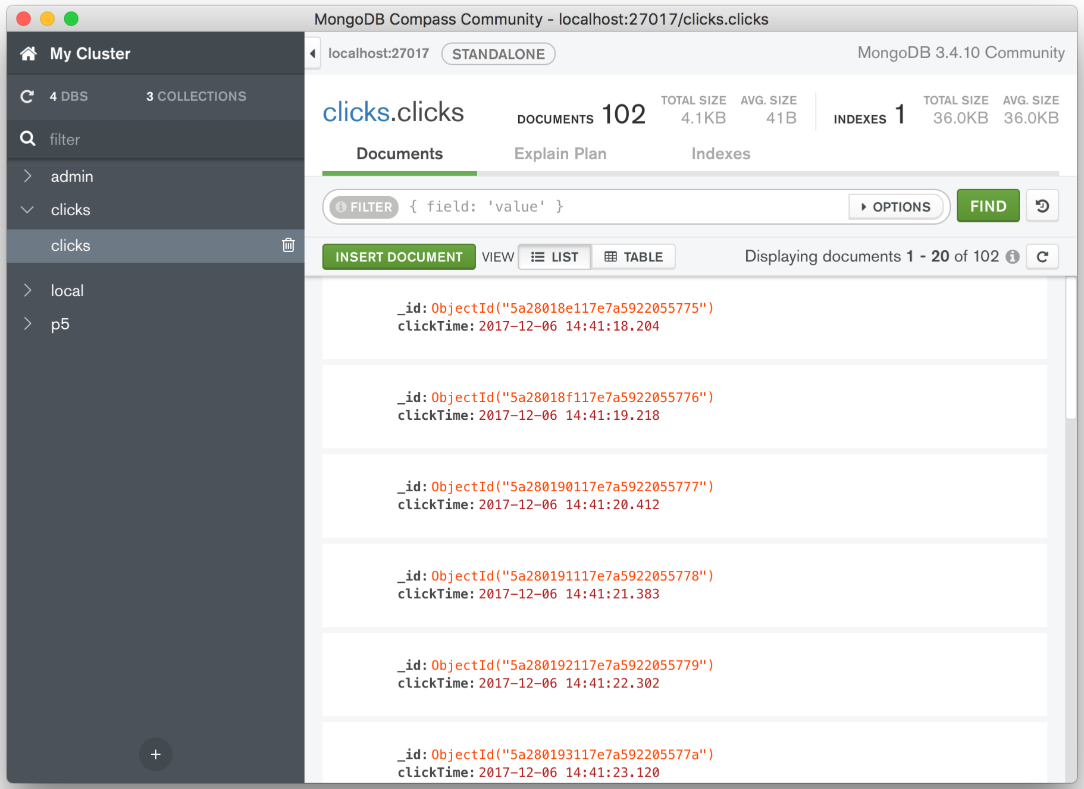
But... the http://localhost:8080/clicked API endpoint does not exist yet on our server. We can set up this endpoint (or route) by adding the following to the *end* of `server.js`.

```
// add a document to the DB collection recording the click event
app.post('/clicked', (req, res) => {
  const click = {clickTime: new Date()};
  console.log(click);
  console.log(db);

  db.collection('clicks').save(click, (err, result) => {
    if (err) {
      return console.log(err);
    }
    console.log('click added to db');
    res.sendStatus(201);
  });
});
```

Note that this has added a POST route at http://localhost:8080/clicked. The anonymous callback supplied uses Mongo's `save()` method to add a new document to the collection containing the current date & time.

If you try this out and view the content of your database (via a tool like Compass) you will see these new documents appearing in your `clicks` collection.



```
_id: ObjectId("5a28018e117e7a5922055775")
clickTime: 2017-12-06 14:41:18.204

_id: ObjectId("5a28018f117e7a5922055776")
clickTime: 2017-12-06 14:41:19.218

_id: ObjectId("5a280190117e7a5922055777")
clickTime: 2017-12-06 14:41:20.412

_id: ObjectId("5a280191117e7a5922055778")
clickTime: 2017-12-06 14:41:21.383

_id: ObjectId("5a280192117e7a5922055779")
clickTime: 2017-12-06 14:41:22.302

_id: ObjectId("5a280193117e7a592205577a")
clickTime: 2017-12-06 14:41:23.120
```

Part 5 - Getting data from the DB

Add a new route to the *end* of `server.js` :

```
// get the click data from the database
app.get('/clicks', (req, res) => {

  db.collection('clicks').find().toArray((err, result) => {
    if (err) return console.log(err);
    res.send(result);
  });
});
```

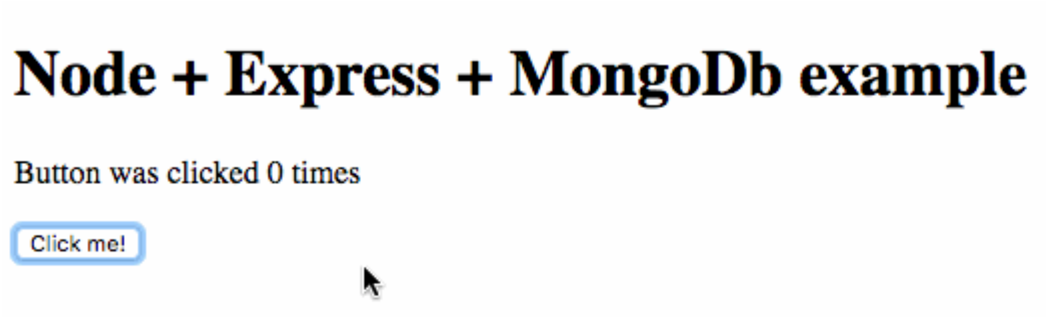
This adds a GET endpoint at http://localhost:8080/clicks which will return an array containing all the documents in the database, i.e. records of every time the button was clicked. To test this you can just visit http://localhost:8080/clicks in a browser (make sure you have `server.js` running in node).

Next we will consume (use) this API endpoint from our client-side code. We will add a `setInterval()` function that will poll the server every second, retrieve all the click data and then display the number of clicks in the UI.

Add this to the end of `client.js` :

```
  setInterval(function() {
    fetch('/clicks', {method: 'GET'})
      .then(function(response) {
        if(response.ok) return response.json();
        throw new Error('Request failed.');
      })
      .then(function(data) {
        document.getElementById('counter').innerHTML = `Button was clickec
      })
      .catch(function(error) {
        console.log(error);
      });
  }, 1000);
```

Done 🙌 ! The webpage should now update every second based on the number of clicks recorded in the database. This will record total clicks from all users, across multiple sessions!



But... there is some lag between the button press and the webpage updating. Why? Can you improve this?

Final versions of the three key files are below.

client.js

```
console.log('Client-side code running');

const button = document.getElementById('myButton');
button.addEventListener('click', function(e) {
  console.log('button was clicked');

  fetch('/clicked', {method: 'POST'})
    .then(function(response) {
      if(response.ok) {
        console.log('click was recorded');
        return;
      }
      throw new Error('Request failed.');
    })
    .catch(function(error) {
      console.log(error);
    });
});

setInterval(function() {
  fetch('/clicks', {method: 'GET'})
    .then(function(response) {
      if(response.ok) return response.json();
      throw new Error('Request failed.');
    })
    .then(function(data) {
      document.getElementById('counter').innerHTML = `Button was clicked
${data.length} times`;
    })
    .catch(function(error) {
      console.log(error);
    });
}, 1000);
```

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Node + Express + MongoDb example</title>
  </head>
  <body>
    <h1>Node + Express + MongoDb example</h1>
    <p id="counter">Loading button click data.</p>
    <button id="myButton">Click me!</button>
  </body>
  <script src="client.js"></script>
</html>
```

server.js

```
console.log('Server-side code running');

const express = require('express');
const MongoClient = require('mongodb').MongoClient;

const app = express();

// serve files from the public directory
app.use(express.static('public'));

// connect to the db and start the express server
let db;

// Replace the URL below with the URL for your database
```

```javascript
const url =  'mongodb://user:password@mongo_address:mongo_port/clicks';

MongoClient.connect(url, (err, database) => {
  if(err) {
    return console.log(err);
  }
  db = database;
  // start the express web server listening on 8080
  app.listen(8080, () => {
    console.log('listening on 8080');
  });
});

// serve the homepage
app.get('/', (req, res) => {
  res.sendFile(__dirname + '/index.html');
});

// add a document to the DB collection recording the click event
app.post('/clicked', (req, res) => {
  const click = {clickTime: new Date()};
  console.log(click);
  console.log(db);

  db.collection('clicks').save(click, (err, result) => {
    if (err) {
      return console.log(err);
    }
    console.log('click added to db');
    res.sendStatus(201);
  });
});

// get the click data from the database
app.get('/clicks', (req, res) => {
  db.collection('clicks').find().toArray((err, result) => {
    if (err) return console.log(err);
    res.send(result);
  });
});
```

avinashbhutekar commented on Apr 9, 2019

thank you

BenjaminBrodwolf commented on Aug 2, 2019

Super short helpful example. Now I understand much more how the communication between client and nodejs works. I looked for that since 2 days.
Thank you very much.

Victoryerz commented on Aug 4, 2019

Thanks a lot.

springfugue commented on Aug 27, 2019

Very helpful. Thanks!

rakeshyadavitan commented on Oct 16, 2019

thanks a lot

liubomyrgavryliv commented on Dec 2, 2019

Brilliant! Thanks!

umarmuha commented on Feb 24, 2020 • edited

This is **awesome**!! this was such an amazing exercise to follow along to understand the basic concept of these technologies. Thanks so much!

In case anyone decided to go with the latest MongoDB driver version like me, the following might be helpful. I had to search around StackOverflow and the latest documentation for the MongoDB node-js driver to figure out how to use the MongoClient.connect constructor. Node the callback function has the err and client parameters and database is selected by client.db('database name')

```
// connect to the db and start express server
let db;

const url = "mongodb://localhost:27017";

MongoClient.connect(url, { useUnifiedTopology: true }, (err, client) => {
  if (err) {
    return console.log(err);
  }
  db = client.db("testDatabase");
  // start the express web server listening on 8080
  app.listen(8080, () => {
    console.log("listening on 8080");
  });
});
```

Blertan commented on Feb 24, 2020

Hi, I don't have clear this part about clicked! I am getting this error:
POST http://localhost:8080/clicked 500 (Internal Server Error)
Error: Request failed.
at client.js:13 (fetch('/clicked', {method: 'POST'})).
So how can I get rid of this error?
I have created on Mongo also DB(clicks) and collection (clicks).

aerrity commented on Feb 24, 2020 Author

See umarmuha's comment above. You may be using a more recent version of MongoDB.

kalyanveenam commented on Apr 19, 2020

Thank you so much for the example. It makes things clear for me, Could you please explain how do we pass query params from UI and the values to particular route in server.

aerrity commented on Apr 22, 2020 Author

You could use query strings.

Or take a look at this example. It includes a PUT request in the client and matching route in the server. This shows how to send data from the UI to the server.

shreyanshnayak... commented on Jun 25, 2020

Thank you so much for this.It made my concept clear.

diegoavellaneda... commented on Jul 9, 2020

Thanks !! Great Information

Riolaurenti commented on Jul 19, 2020 • edited

Hi, my client.js file doesn't log anything to console at Part 2.. Ideas?

iamphoenix310 commented on Sep 15, 2020

Very hellpful - thanks

smartens21 commented on Oct 4, 2020

> Hi, my client.js file doesn't log anything to console at Part 2.. Ideas?

here the same

marcin2x4 commented on Nov 9, 2020 • edited

Great tutorial! I'm trying to replicate it but with different db (Snowflake). On click event I get however this issue:
`Content blocked „http://localhost:8080/client.js” due incombatibility (X-Content-Type-Options: nosniff) type MIME („text/html”).`

I switched `app.use(express.static('public'));` to `app.use('/static', express.static(__dirname + '/static'));` but no luck. My browser is Firefox.

Another issue I get is that `script` element has not been loaded `http://localhost:8080/client.js`

Sohumkalia3939 commented on Mar 8, 2021

Thanks

xXpsyXx commented on May 3, 2021 • edited

I m unable to connect this project to a cluster
I m getting invalid schema expected mongodb
and i want to do it via mongoose

rohansingh20 commented on Jul 15, 2021

Im getting

```
TypeError: db.collection(...).save is not a function
```

any thoughts

ShaakhDev commented on Sep 6, 2021

> Im getting
>
> ```
> TypeError: db.collection(...).save is not a function
> ```
>
> any thoughts

I have the same problem

projektorius96 commented on Sep 13, 2021

@rohansingh20 , **@ShaakhDev** and anyone else that may have a problem with deprecated .save() method

Try to use **insertOne()** for one document (record) insertion or **insertMany()** for multiple documents (records) insertion instead of *deprecated **.save()*** method .

References

Insert method || **Note** : **insert()** is deprecated, use **bulkWrite()** instead !
Node.js Driver v4.1 API – current version
Node.js Driver vX.Y API – if lower version used

ShaakhDev commented on Sep 13, 2021

Thank you so much!

projektorius96 commented on Sep 14, 2021

@Aerity idea with interactive button on client side for Express server instead of `<a href="/some-endpoint">` or statically typed into address bar within Browser – I was surfing nearly whole week for this kind of solution ! All I can say "*it's briliant*", just tested out and can say it worked like a charm . Thank you so much ! ❤️

p.s. as mention per comment above **.save()** is deprecated , I would suggest the source code to be revised and updated ! 👍

wwha2121 commented on Jan 12, 2022

thank you so much!