# CS 207: Applied Database Practicum Week 9

Varun Dutt

School of Computing and Electrical Engineering
School of Humanities and Social Sciences
Indian Institute of Technology Mandi, India

Scaling the Heights

## HTTP server in Node.js

- The Node.js framework is mostly used to create server based applications. The framework can easily be used to create web servers which can serve content to users.
- There are a variety of modules such as the "http" and "request" module, which helps in processing server related requests in the web server space.

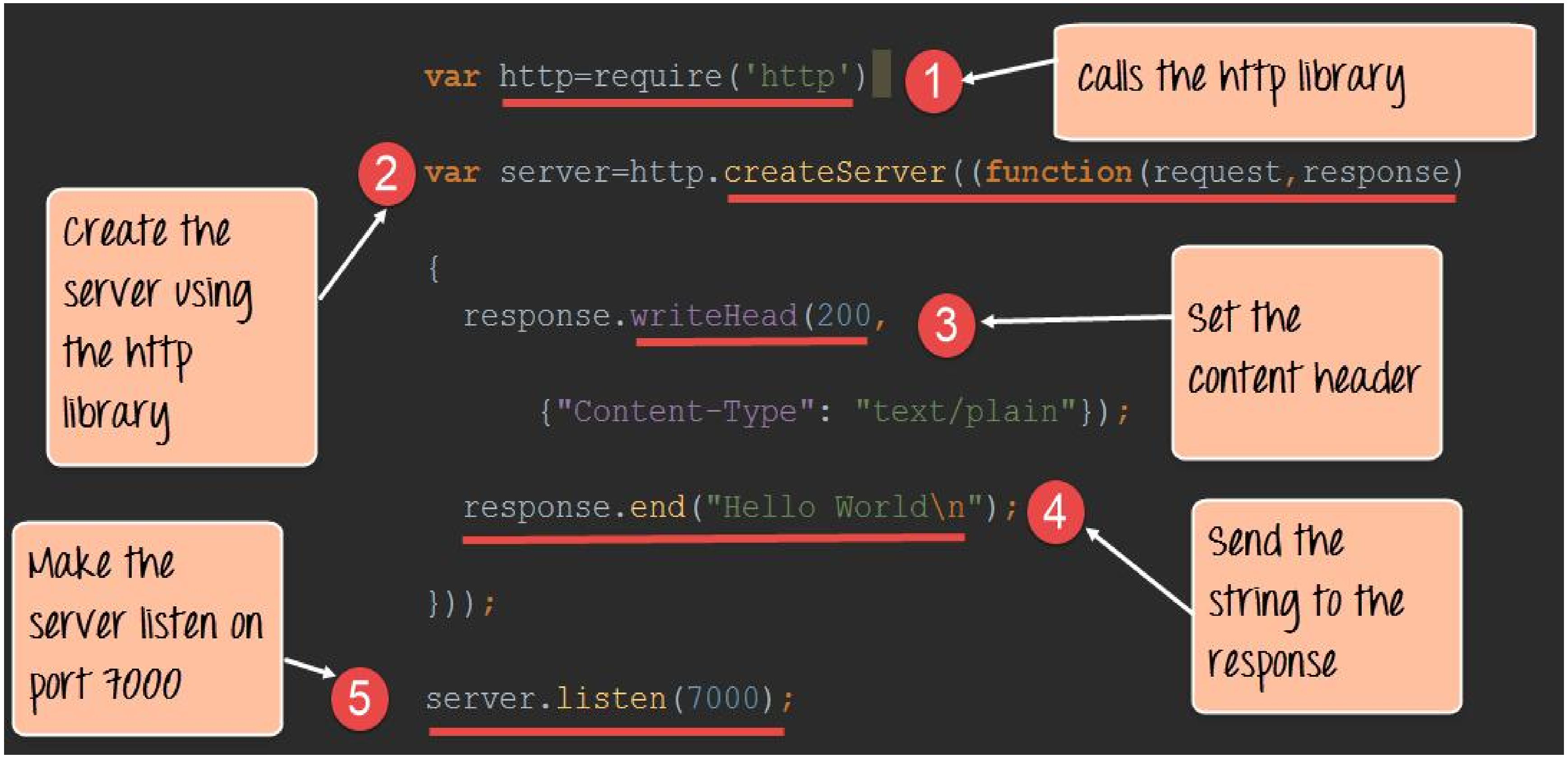
## Node.js as web server using HTTP

So we’ll starting with example-

Our application is going to create a simple server module which will listen on port no 7000. If a request is made through the browser on this port no, then server application will send a 'Hello' World' response to the client.

# Node as web server

## Code -



```
var http=require('http')

var server=http.createServer((function(request,response)

{
  response.writeHead(200,

      {"Content-Type": "text/plain"});

  response.end("Hello World\n");

}));

server.listen(7000);
```

1
calls the http library
2
Create the server using the http library
3
Set the content header
4
Send the string to the response
5
Make the server listen on port 7000

## Node as web server

Above code explanation -

1. We use the require function to get the required functions from the http module so that it can be used in our application.
2. We are creating a server application which is based on a simple function. This function is called whenever a request is made to our server application.

## Node as web server

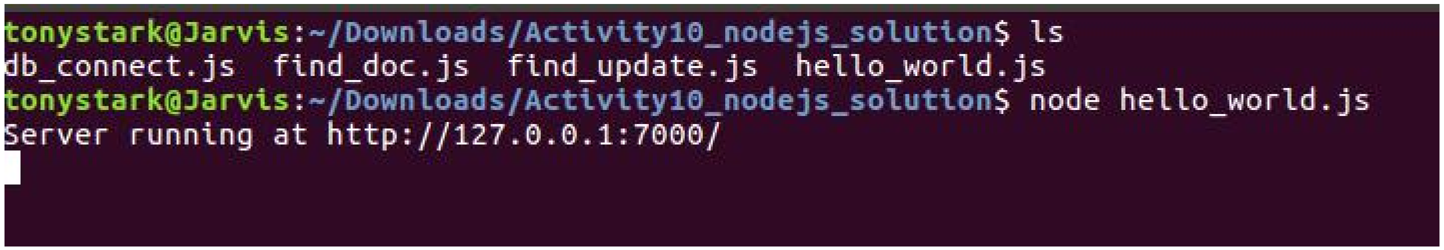
3. When a request is received, we are saying to send a response with a header type of '200.' This number is the normal response which is sent in an http header when a successful response is sent to the client.

4. In the response itself, we are sending the string 'Hello World.'

## Node as web server

5. We are then using the server.listen function to make our server application listen to client requests on port no 7000. You can specify any available port over here.

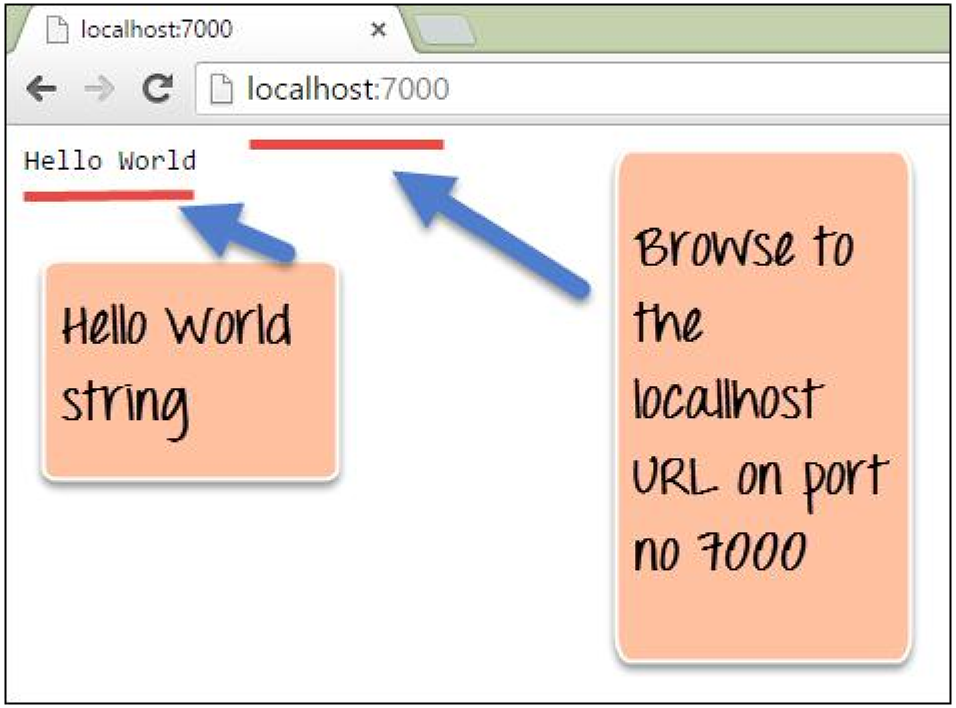
## Running a nodejs application



```
tonystark@Jarvis:~/Downloads/Activity10_nodejs_solution$ ls
db_connect.js  find_doc.js  find_update.js  hello_world.js
tonystark@Jarvis:~/Downloads/Activity10_nodejs_solution$ node hello_world.js
Server running at http://127.0.0.1:7000/
```

## Node as web server

Output -



```
Hello World
```

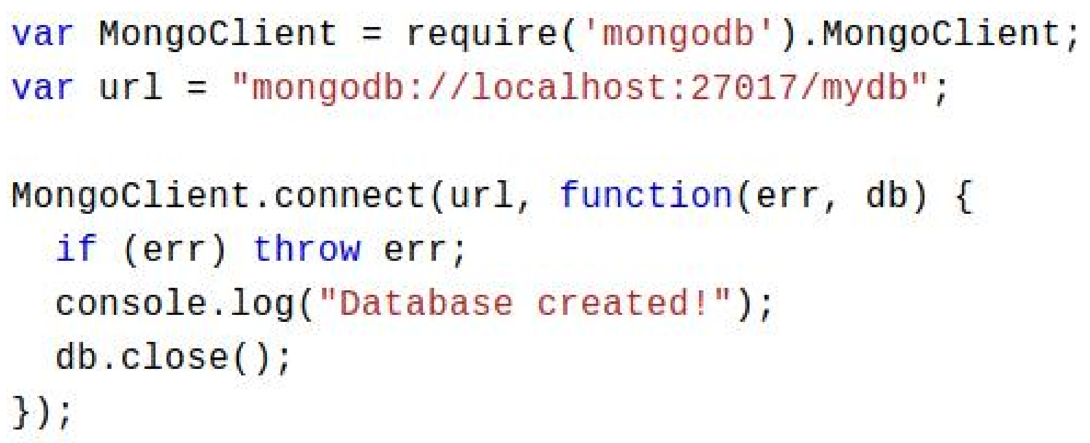
## Node.js MongoDB Create Database

**Creating a database -**
To create a database in MongoDB, start by creating a MongoClient object, then specify a connection URL with the correct ip address and the name of the database you want to create.
MongoDB will create the database if it does not exist, and make a connection to it.

## Node.js MongoDB Create Database

**Example -**

Code -



```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/mydb";

MongoClient.connect(url, function(err, db) {
  if (err) throw err;
  console.log("Database created!");
  db.close();
});
```

## Node.js MongoDB Create Database

**Run the above code -**

```
node <file_name>.js
```

**Output -**

```
Database created!
```

**In MongoDB, a database is not created until it gets content!**

MongoDB waits until you have created a collection (table), with at least one document (record) before it actually creates the database (and collection).

## Node.js MongoDB Create Collection

To create a collection in MongoDB, use the **createCollection()** method.

**Example -**

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";

MongoClient.connect(url, function(err, db) {
  if (err) throw err;
  var dbo = db.db("mydb");
  dbo.createCollection("customers", function(err, res) {
    if (err) throw err;
    console.log("Collection created!");
    db.close();
  });
});
```

## Node.js MongoDB Create Collection

On running the above code -

**Output -**

```
Collection created!
```

**In MongoDB, a collection is not created until it gets content!**

MongoDB waits until you have inserted a document before it actually creates the collection.

## Node.js MongoDB Insert in Collection

Example using insertOne() method-

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";

MongoClient.connect(url, function(err, db) {
  if (err) throw err;
  var dbo = db.db("mydb");
  var myobj = { name: "Company Inc", address: "Highway 37" };
  dbo.collection("customers").insertOne(myobj, function(err, res) {
    if (err) throw err;
    console.log("1 document inserted");
    db.close();
  });
});
```

## Node.js MongoDB Insert in Collection

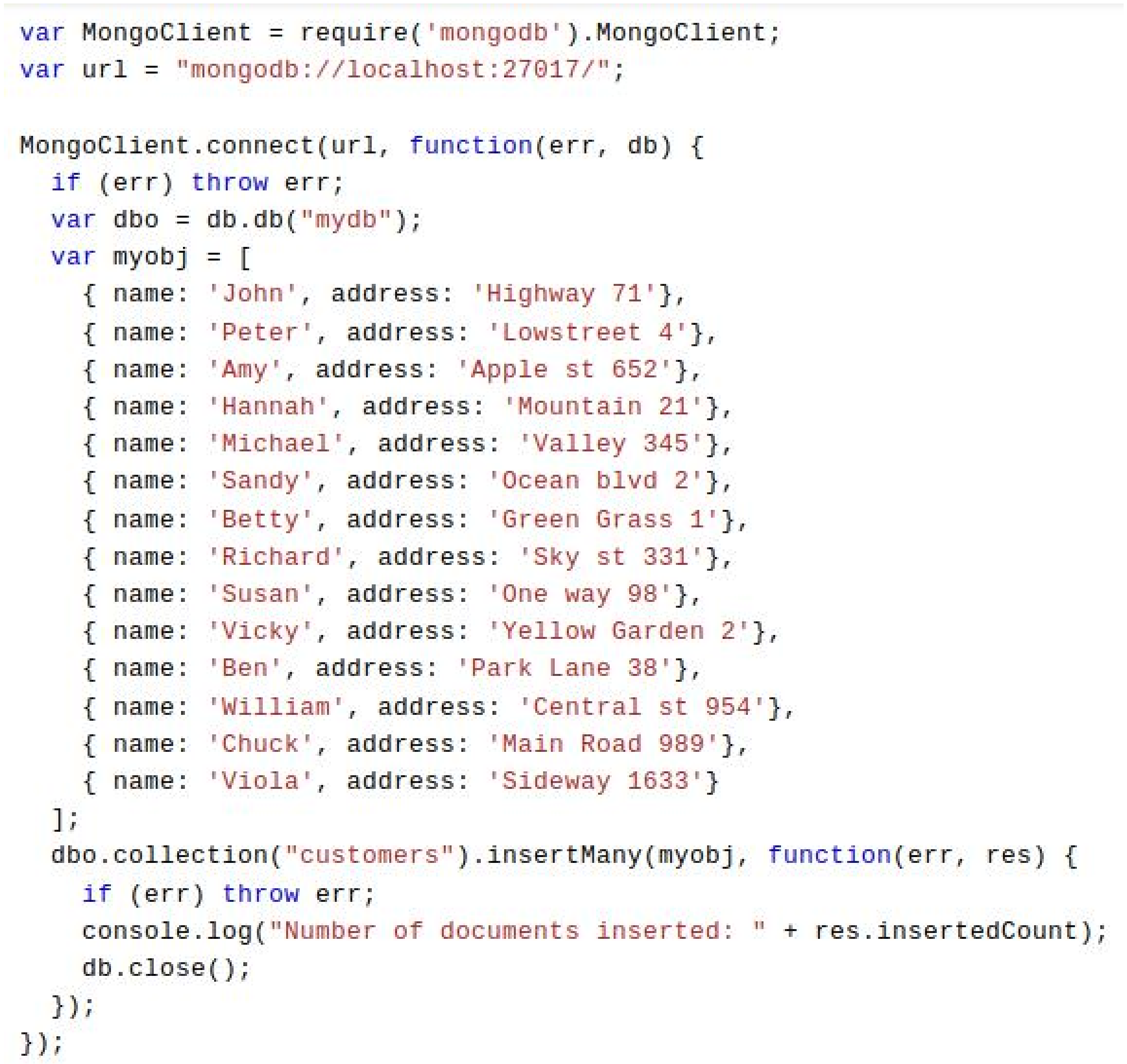
After running the above code-
Output -
1 document inserted

**If you try to insert documents in a collection that do not exist, MongoDB will create the collection automatically.**

## Node.js MongoDB Insert in Collection

Example on inserting multiple documents using insertMany() method.
With array of objects, insertMany also takes a callback function where you can work with any errors, or the result of the insertion.

# Node.js MongoDB Insert in Collection



```javascript
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";

MongoClient.connect(url, function(err, db) {
  if (err) throw err;
  var dbo = db.db("mydb");
  var myobj = [
    { name: 'John', address: 'Highway 71'},
    { name: 'Peter', address: 'Lowstreet 4'},
    { name: 'Amy', address: 'Apple st 652'},
    { name: 'Hannah', address: 'Mountain 21'},
    { name: 'Michael', address: 'Valley 345'},
    { name: 'Sandy', address: 'Ocean blvd 2'},
    { name: 'Betty', address: 'Green Grass 1'},
    { name: 'Richard', address: 'Sky st 331'},
    { name: 'Susan', address: 'One way 98'},
    { name: 'Vicky', address: 'Yellow Garden 2'},
    { name: 'Ben', address: 'Park Lane 38'},
    { name: 'William', address: 'Central st 954'},
    { name: 'Chuck', address: 'Main Road 989'},
    { name: 'Viola', address: 'Sideway 1633'}
  ];
  dbo.collection("customers").insertMany(myobj, function(err, res) {
    if (err) throw err;
    console.log("Number of documents inserted: " + res.insertedCount);
    db.close();
  });
});
```

## Node.js MongoDB Insert in Collection

Output -

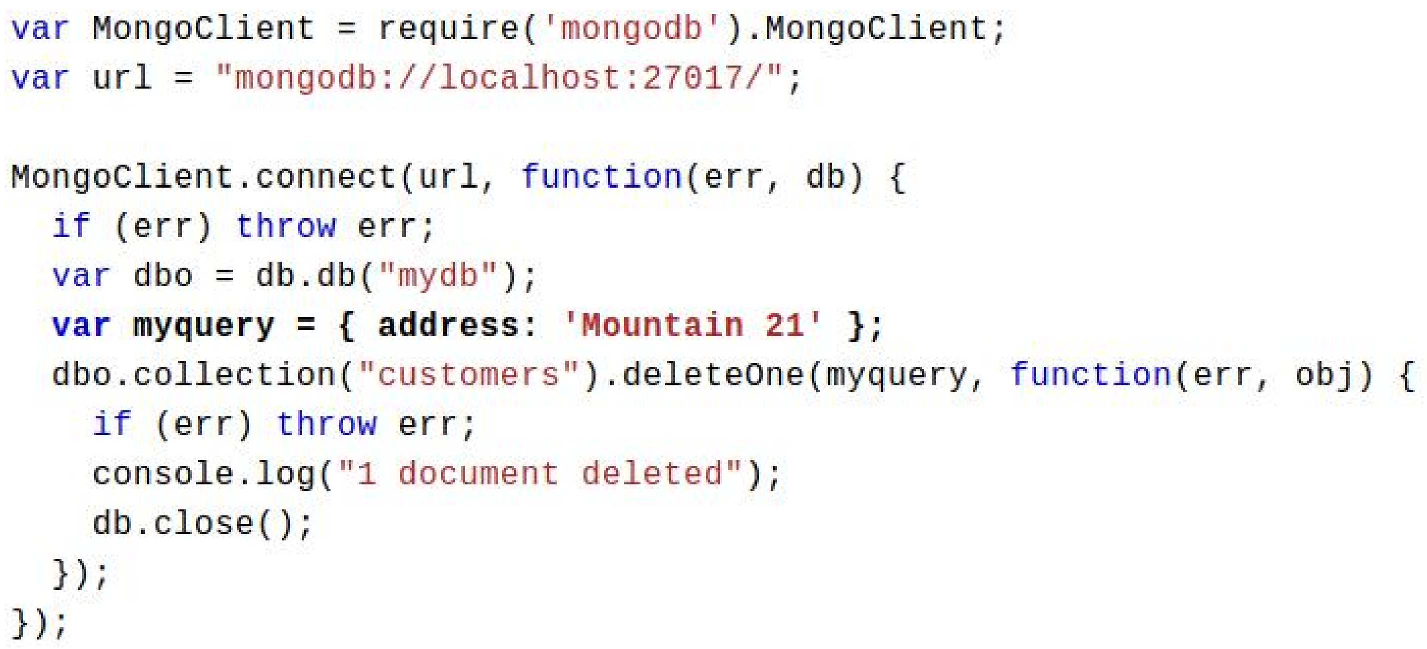
```
Number of documents inserted: 14
```

## Node.js MongoDB Delete in Collection

Example using deleteOne() method for deleting one record or document.

**If the query finds more than one document, only the first occurrence is deleted.**

# Node.js MongoDB Delete in Collection



```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";

MongoClient.connect(url, function(err, db) {
  if (err) throw err;
  var dbo = db.db("mydb");
  var myquery = { address: 'Mountain 21' };
  dbo.collection("customers").deleteOne(myquery, function(err, obj) {
    if (err) throw err;
    console.log("1 document deleted");
    db.close();
  });
});
```

## Node.js MongoDB Delete in Collection

Output -
1 record deleted

Similar to insertion case, in delete case too we have **deleteMany()** method to delete more than one document.

## Node.js MongoDB Update in Collection

Example on update document -
Update a record or document using updateOne() method.

**If the query finds more than one record, only the first occurrence is updated.**

# Node.js MongoDB Update in Collection

## Example -

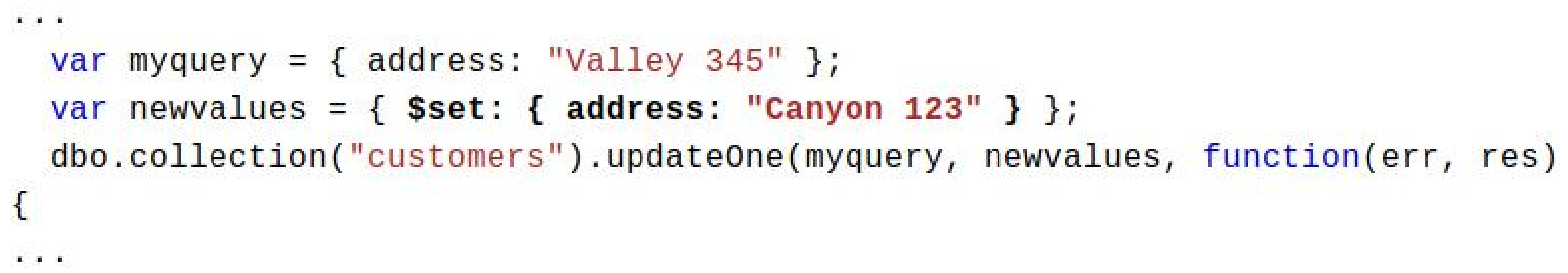
```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://127.0.0.1:27017/";

MongoClient.connect(url, function(err, db) {
  if (err) throw err;
  var dbo = db.db("mydb");
  var myquery = { address: "Valley 345" };
  var newvalues = { $set: {name: "Mickey", address: "Canyon 123" } };
  dbo.collection("customers").updateOne(myquery, newvalues, function(err, res)
{
    if (err) throw err;
    console.log("1 document updated");
    db.close();
  });
});
```

## Node.js MongoDB Update in Collection

Output -
1 document updated

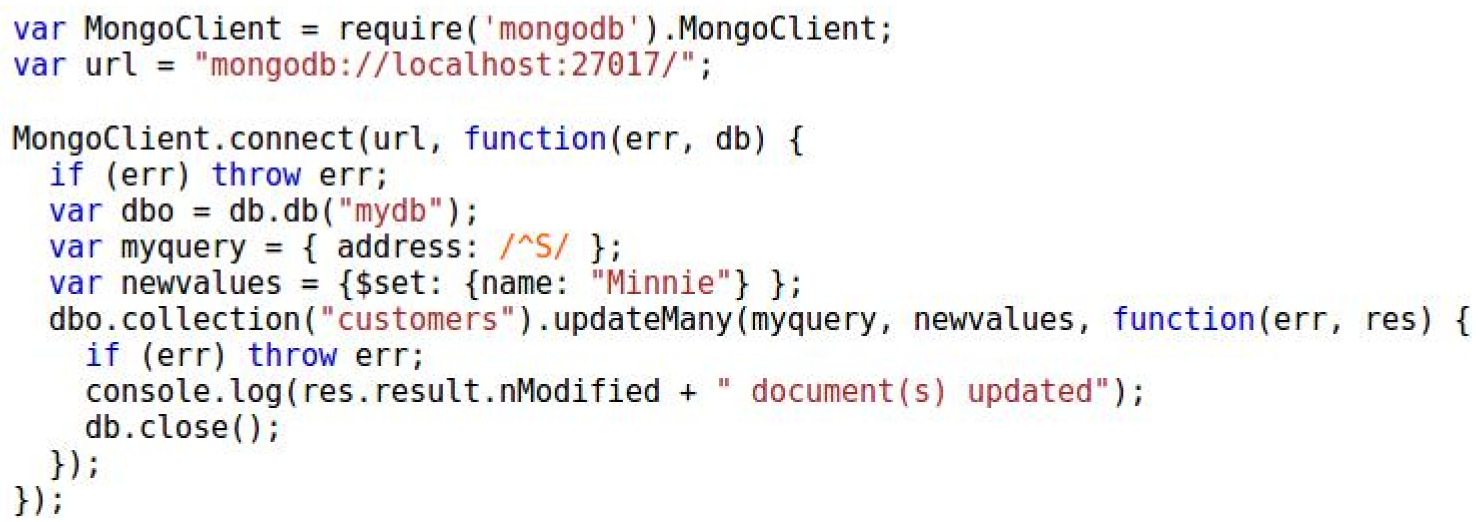
**Update only specific fields -**
When using the **$set operator**, only the specified fields are updated:



```
...
  var myquery = { address: "Valley 345" };
  var newvalues = { $set: { address: "Canyon 123" } };
  dbo.collection("customers").updateOne(myquery, newvalues, function(err, res)
{
...
```

# Node.js MongoDB Update in Collection

Likewise in insertion, update too has **updateMany()** method to update multiple documents.



```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";

MongoClient.connect(url, function(err, db) {
  if (err) throw err;
  var dbo = db.db("mydb");
  var myquery = { address: /^S/ };
  var newvalues = {$set: {name: "Minnie"} };
  dbo.collection("customers").updateMany(myquery, newvalues, function(err, res) {
    if (err) throw err;
    console.log(res.result.nModified + " document(s) updated");
    db.close();
  });
});
```

## What is AJAX?

- AJAX stands for **A**synchronous **J**avascript **A**nd **X**ML.
- AJAX is not a programming language.
- AJAX is a way of using existing standards (JavaScript and XML) to make more interactive web applications.
- AJAX was popularized in 2005 by Google (with Google suggest)

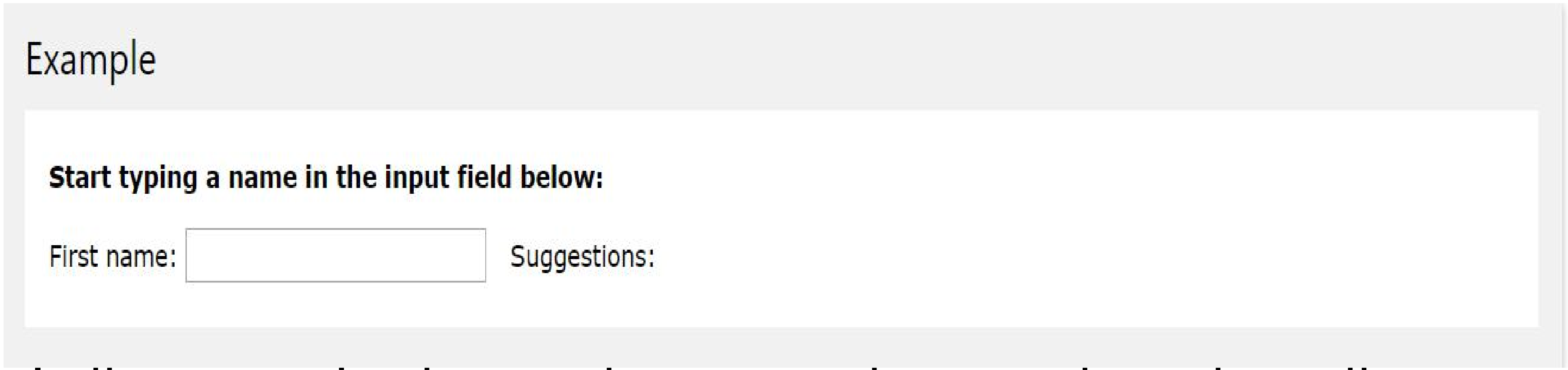
## How AJAX Works?

1. An event occurs in a web page (the page is loaded, a button is clicked).
2. An XMLHttpRequest object is created by JavaScript.
3. The XMLHttpRequest object sends a request to a Web Server.
4. The server processes the request.
5. The server sends a response back to the web page.
6. The response is read by JavaScript.
7. Proper action (like page update) is performed by JavaScript.

## AJAX PHP

AJAX is used with PHP to create more interactive applications.

The following example demonstrates how a web page can communicate with a web server while a user types characters in an input field:

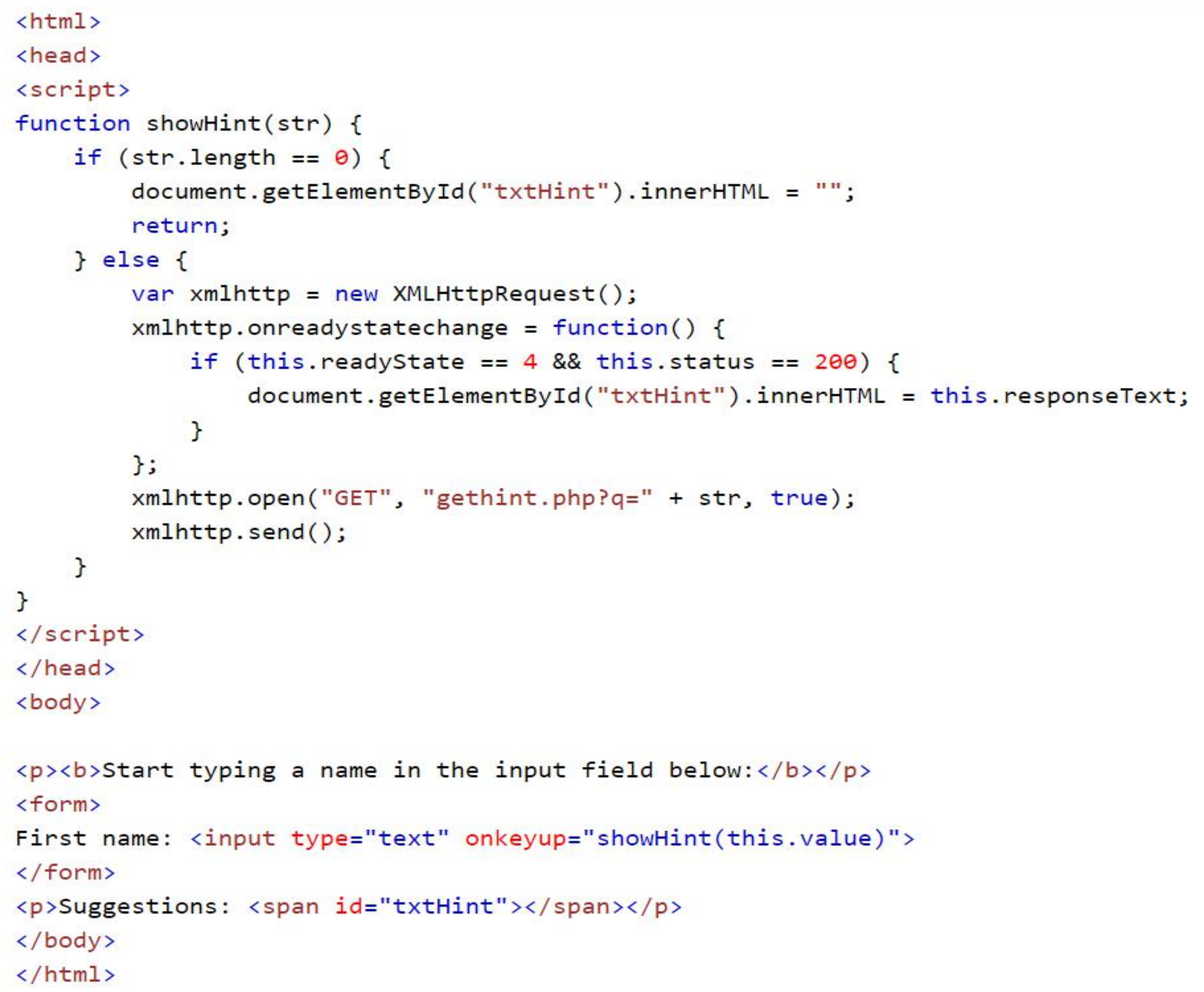

Example

**Start typing a name in the input field below:**

First name: Suggestions:

In the example above, when a user types a character in the input field, a function called "showHint()" is executed.
The function is triggered by the onkeyup event.

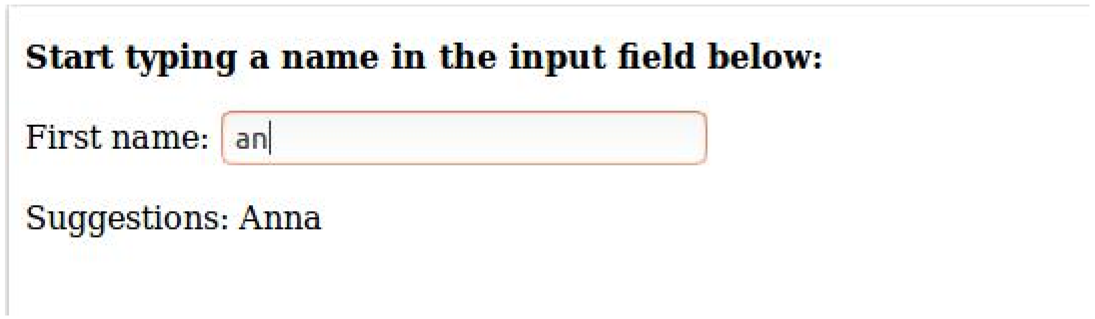
# AJAX PHP code



```
<html>
<head>
<script>
function showHint(str) {
    if (str.length == 0) {
        document.getElementById("txtHint").innerHTML = "";
        return;
    } else {
        var xmlhttp = new XMLHttpRequest();
        xmlhttp.onreadystatechange = function() {
            if (this.readyState == 4 && this.status == 200) {
                document.getElementById("txtHint").innerHTML = this.responseText;
            }
        };
        xmlhttp.open("GET", "gethint.php?q=" + str, true);
        xmlhttp.send();
    }
}
</script>
</head>
<body>

<p><b>Start typing a name in the input field below:</b></p>
<form>
First name: <input type="text" onkeyup="showHint(this.value)">
</form>
<p>Suggestions: <span id="txtHint"></span></p>
</body>
</html>
```

## AJAX PHP output



**Start typing a name in the input field below:**

First name: an

Suggestions: Anna

## Summary

- An AJAX transaction involves the client sending an asynchronous HTTP request and the server responding with XML
  - The client processes the resulting XML document tree
- AJAX applications run entirely on the client except when they need to access data on the server – Can treat the server as a database/file system.
- Well-written AJAX applications, running with a fast Internet connection, can be as nice to use as traditional applications (or nicer).

## REFERENCES

- http://cglab.ca/~morin/teaching/2405/notes/ajax1.pdf
- https://www.w3schools.com