COMPUTER ENGINEERING
CHIANG MAI UNIVERSITY

Node.js and Express

CPE405 - Advanced Computer Engineering Technology

Dome Potikanond
Navadon Khunlertgit
Banana Software co., Ltd

Computer Engineering, Chiang Mai University


node
JS

Topics

- What is Node.js and how does it works?
- Installing Node.js
- NPM – Node Package Manager
- Node modules and package.json
- Basic web server

What is Node.js?

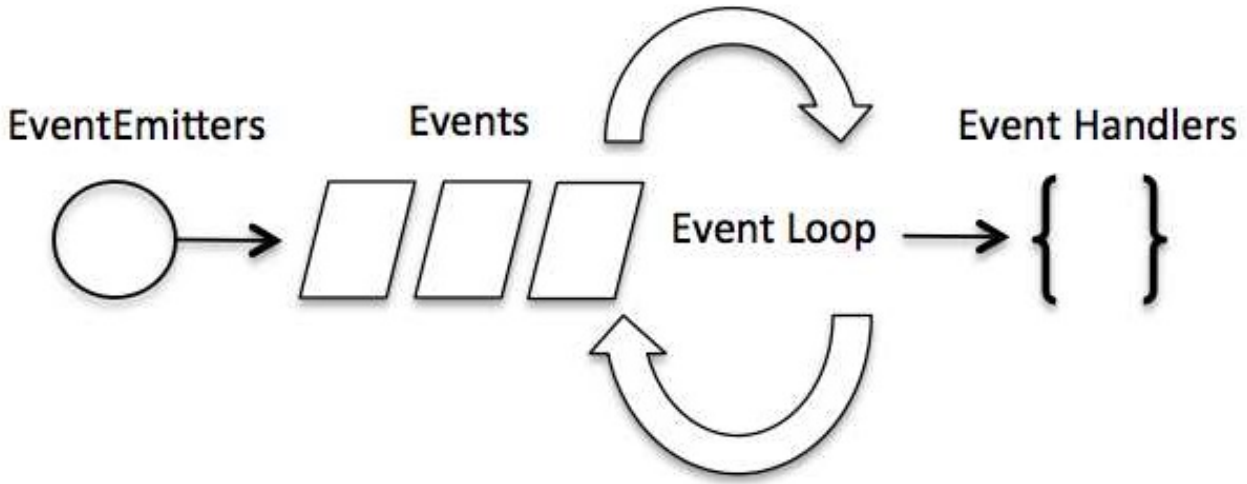
- JavaScript runtime built on Chrome's V8 JavaScript engine
- JavaScript running on the server
- Used to build powerful, fast & scalable web applications
- Uses an event-driven, non-blocking I/O model
  - Event Loop
- Download: https://nodejs.org/en/
  - ftp://10.10.182.181/cpe405/ (from within CMU)

Non-blocking I/O

- Works on a single thread using non-blocking I/O calls
- Supports tens of thousands concurrent connections
- Optimizes throughput and scalability in web applications with many I/O operations
- This makes Node.js apps extremely **fast** and **efficient**

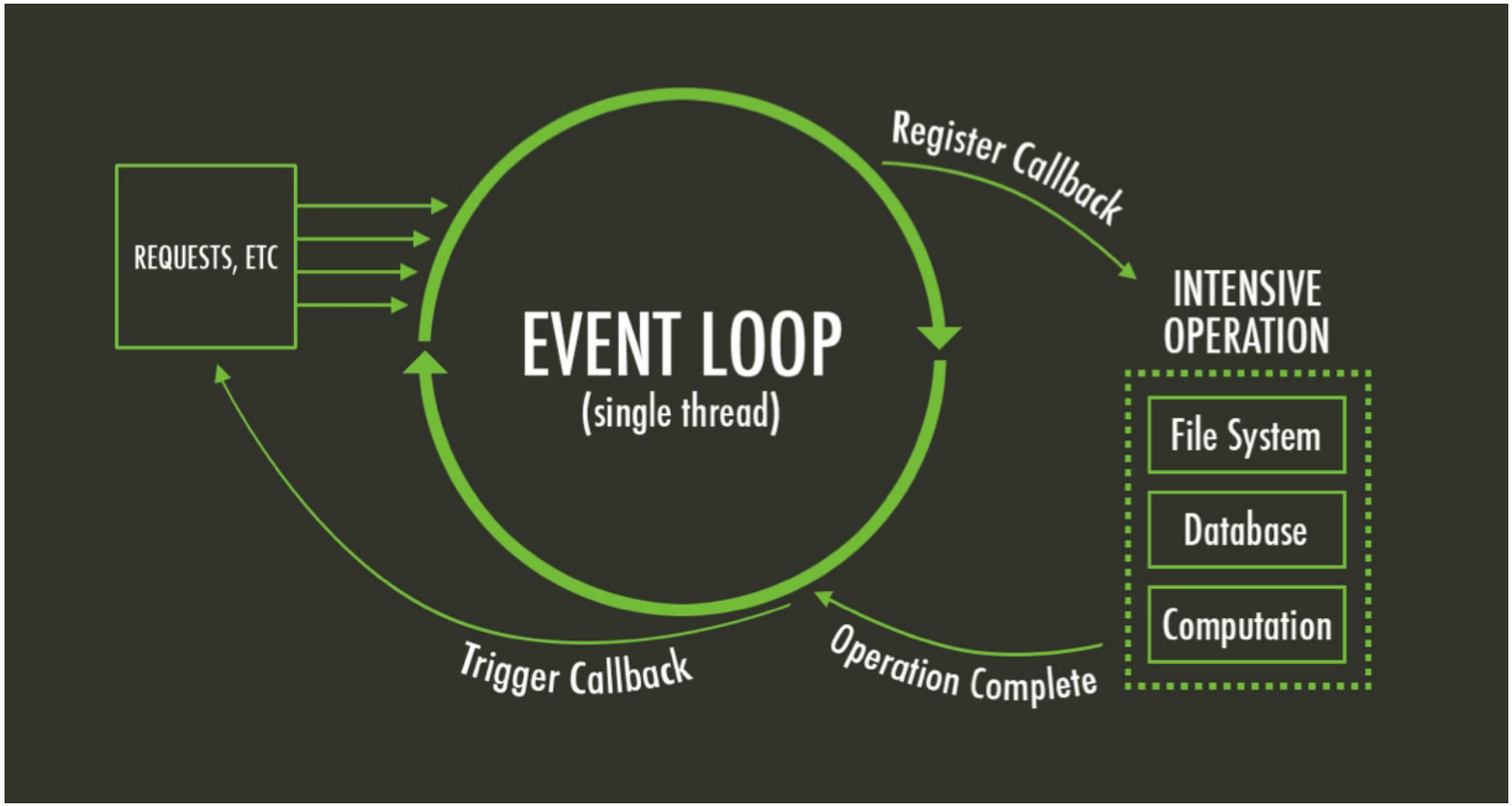
Event Loop

- Modern kernels are multi-threaded and can handle multiple operations in the background
  - When one of these operations completes, the kernel tells Node.js
  - An appropriate callback (function) may be added and executed



Event Loop Model

- Supports concurrency via **events** and **callbacks**
- **EventEmitter** class is used to bind events and event listener



What can we build with Node.js?

- REST APIs & Backend Applications
- Real-time services (Chat, Games, …)
- Blogs, CMS, Social Applications
- Utilities & Tools
- Anything that is not CPU-intensive

NPM

- Node.js Package Manager
- Used to install node programs/modules
- Modules get installed into the "***node_modules***" folder

```
# npm install  [options]  <module1  [module2]>
```

- Example:

```
# npm install express
```

```
# npm install -g express
```

Popular Modules

- **Express** – web development framework
- **Connect** – Extensible HTTP server framework
- **Socket.io** – Server side component for websocket
- **Pug/Jade** – Template engine inspired by HAML
- **Mongo/Mongoose** – Wrappers to interact with MongoDB
- **Coffee-Script** – CoffeeScript compiler
- **Redis** – Cache client library

package.json

- Located in the root of your package/application
- Tells NPM how your package is structured
  - What to do to install it

```
// inside project folder
# npm init

// start Node.js app
# node app.js
```

```
{
        "name": “mytasklist",
        “version”: “1.0.0”
        "description": "A Simple Task Manager",
        “main”: “app.js”
        "author": “John Doe <mblacky0@gmail.com>",
        "dependencies": {
                “body-parser": "^1.15.2",
                “express": "^4.14.0“,
                “mogojs”: “*”
        },
}
```

Node.js – Simple Server#1

```
// app.js - entry point for Node.js application
const http = require('http');

const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req,res) => {
    res.statusCode = 200;
    res.setHeader('Content-type', 'text/plain');
    res.end('Hello World');
});

server.listen(port, hostname, () => {
    console.log('Server started on port ' + port);
});
```

Node.js – Simple Server#2

```
const http = require('http');
const fs = require('fs');
const hostname = '127.0.0.1';
const port = 3000

fs.readFile('index.html', (err, html) => {
    if(err) { throw err; }

    const server = http.createServer((req,res) => {
     res.statusCode = 200;
     res.setHeader('Content-type', 'text/plain');
     res.write(html)
     res.end();
    });

    server.listen(port, hostname, () => {
     console.log('Server started on port ' + port);
    });
});
```

Anatomy of an HTTP Transaction

- Create the Server

```
const http = require('http');

const server = http.createServer((request, response) => {
  // magic happens here!
});
```

```
const server = http.createServer();
server.on('request', (request, response) => {
  // the same kind of magic happens here!
});
```

- When a **request** hits the server, node calls the request handler function with request and response objects

Anatomy of an HTTP Transaction (2)

- Method, URL and Headers

```
const { method, url } = request;
const { headers } = request;
const userAgent = headers['user-agent'];
```

  - Using defined property names to access request data
    - **url** – the full URL without the server, protocol or port
    - **headers** – all headers are represented in lower-case only

Anatomy of an HTTP Transaction (3)

- Request Body
  - When receiving a **POST** and **PUT** request, the **request body** might be important to your application
  - Getting the **body data** is a bit complicated
    - Grab the data out of the input stream by listening to the stream's '**data**' and '**end**' events.

```
let body = [];
request.on('data', (chunk) => {
  body.push(chunk);
}).on('end', () => {
  body = Buffer.concat(body).toString();
  // at this point, `body` has the entire request body stored in it as a string
});
```

Anatomy of an HTTP Transaction (4)

- Errors
  - An **error** in the request presents by emitting an 'error' event
  - If **no listener** for the error event, it will be thrown
    - This could crash the Node.js application

```
request.on('error', (err) => {
  // This prints the error message and stack trace to `stderr`.
  console.error(err.stack);
});
```

Anatomy of an HTTP Transaction (5)

- HTTP Status Code and Response Header
  - By default the **status code** on a response will always be 200

```
response.statusCode = 404; // Tell the client that the resource wasn't found.
```

  - **Headers** are set through a method called *setHeader*

```
response.setHeader('Content-Type', 'application/json');
response.setHeader('X-Powered-By', 'bacon');
```

    - Headers are case insensitive on their names
  - Explicitly write the header to the response stream

```
response.writeHead(200, {
  'Content-Type': 'application/json',
  'X-Powered-By': 'bacon'
});
```

Anatomy of an HTTP Transaction (6)

- Sending Response Body
  - Writing a **response body** out to the client using the *write* method

```
response.write('<html>');
response.write('<body>');
response.write('<h1>Hello, World!</h1>');
response.write('</body>');
response.write('</html>');
response.end();
```

  - The *end* function on streams can take in some optional data to send as the last bit of data on the stream

```
response.end('<html><body><h1>Hello, World!</h1></body></html>');
```

Anatomy of an HTTP Transaction (6)

- Echo Server Example

```
const http = require('http');

http.createServer((request, response) => {
  if (request.method === 'GET' && request.url === '/echo') {
    let body = [];
    request.on('data', (chunk) => {
      body.push(chunk);
    }).on('end', () => {
      body = Buffer.concat(body).toString();
      response.end(body);
    });
  } else {
    response.statusCode = 404;
    response.end();
  }
}).listen(8080);
```

Send an echo back on …
- The request **method** is **GET**
- The **URL** is **/echo**

Respond with a **404** in any other case.

Anatomy of an HTTP Transaction (6)

- Echo Server Example – piping **input** to **output** stream

```
const http = require('http');

http.createServer((request, response) => {
  request.on('error', (err) => {
    console.error(err);
    response.statusCode = 400;
    response.end();
  });
  response.on('error', (err) => {
    console.error(err);
  });
  if (request.method === 'GET' && request.url === '/echo') {
    request.pipe(response);
  } else {
    response.statusCode = 404;
    response.end();
  }
}).listen(8080);
```

Send an echo back on …
- The request **method** is **GET**
- The **URL** is **/echo**

Respond with a **404** in any other case.

Handle **error** on the request
- Log the error to **stderr**
- Send a **400** status code (Bad Request)


Express
JS

Topics

- What is ExpressJS?
- Installation and Setup
- Middleware
- Routing
- Template Engines
- Forms & Input
- Models, ORM & MongoDB

What is Express?

- A minimalistic, open source web framework for Node.js
- Used to build powerful web applications and APIs
- Most popular framework for Node.js
- Uses MVC concepts

Express Installation

- Requirement
  - Node.js
  - NPM (already comes with Node.js)

```
# cd  project_dir
```

```
# npm  init
```

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

- Optional: nodemon (similar to live-server)

```
# npm  install  nodemon  -g          // install
```

```
# nodemon                            // start
```

Express – Simple App#1

```
// app.js - entry point for Node.js application
var express = require('express');
var bodyParser = require('body-parser');

var app = express();
app.get('/', function(req, res) {
    res.send('Hello World..');
});

app.listen(3000, function() {
    console.log('Server Started on Port 3000…');
});
```

Express – Simple App#2

```
var express = require('express');
var bodyParser = require('body-parser');
var path = require('path');
var app = express();

/* Body Parser Middleware */
app.use(bodyParser.json());
app.use(bodyParser.urlencoded( {extended: true} ));

/* Path for static content: Angular, Vue.js, html, js, css */
// Create 'index.html' file inside the 'public' directory
app.use(express.static(path.join(__dirname, 'public')));

app.get('/', function(req, res) {
    res.send('Hello World..');  // index.html will overwrite this
});

app.listen(3000, function() {
    console.log('Server Started on Port 3000…');
}
```

Express – Simple API#1

```
…
var people = [
    { name: 'John Doe', age: 35 },
    { name: 'Jane Deen', age: 19 },
    { name: 'Billy bob', age: 49 }
];

app.get('/', function(req, res) {
    res.send('Hello World..');
});

app.get('/users', function(req, res) {
    res.json(people);
});

app.listen(3000, function() {
    console.log('Server Started on Port 3000…');
}
```

Express – Template Engine

- Embedded JavaScript template (EJS)
  - https://www.npmjs.com/package/ejs
- Support control flow with <% %>
- Support data interpolation <%= %>
- Complies with the Express view system
- Installation

```
# npm  install ejs --save          // install
```

Express – View engine#1

```
…
/* setup view engine */
app.set('view engine', 'ejs');
app.set('views', path.join(__dirname, 'views'));

/* handle GET request */
app.get('/', function(req, res) {
    res.render('index');    // render 'views/index.ejs'
});

…
app.listen(3000, function() {
    console.log('Server Started on Port 3000…');
}
```

Express – View engine#2

```
…
/* setup view engine and directory */
app.set('view engine', 'ejs');
app.set('views', path.join(__dirname, 'views'));

/* handle GET request */
app.get('/', function(req, res) {
    res.render('index', {              // passing params
        title: 'Customer List',
        users: people
    });
});

…
```

Express – View engine#2 (view)

```
<!- view: index.ejs -->
<html>
    <head>
        <title>My Express App</title>
    </head>
    <body>
        <h2><%= title %></h2>
        <ul>
        <% users.forEach( function(user) { %>
            <li><%= user.name %> (<%= user.age %>)</li>
        <% }) %>
        </ul>
    </body>
</html>
```

More on Array - https://www.tutorialspoint.com/javascript/javascript_arrays_object.htm

Express – View engine#2 (partials)

```
<!- view: header.ejs -->
<html>
    <head>
        <title>My Express App</title>
    </head>
    <body>
```

```
<!- view: index.ejs -->
<% include partials/header %>
    <h2><%= title %></h2>
    …
<% include partials/footer %>
```

```
<!- view: footer.ejs -->
    </body>
</html>
```

Express – Handle Form Inputs

```
<!- view: index.ejs -->
<% include partials/header %>
    <h2><%= title %></h2>

    <form method="POST" action="/users/add">
        <label>Full Name</label><br>
        <input type="text" name="name"><br>
        <label>Age</label><br>
        <input type="number" name="age"><br>
        <label>Email</label><br>
        <input type="text" name="email"><br><br>
        <input type="submit" value="Add">
    </form>
    …
<% include partials/footer %>
```

Express – Handle Form Inputs

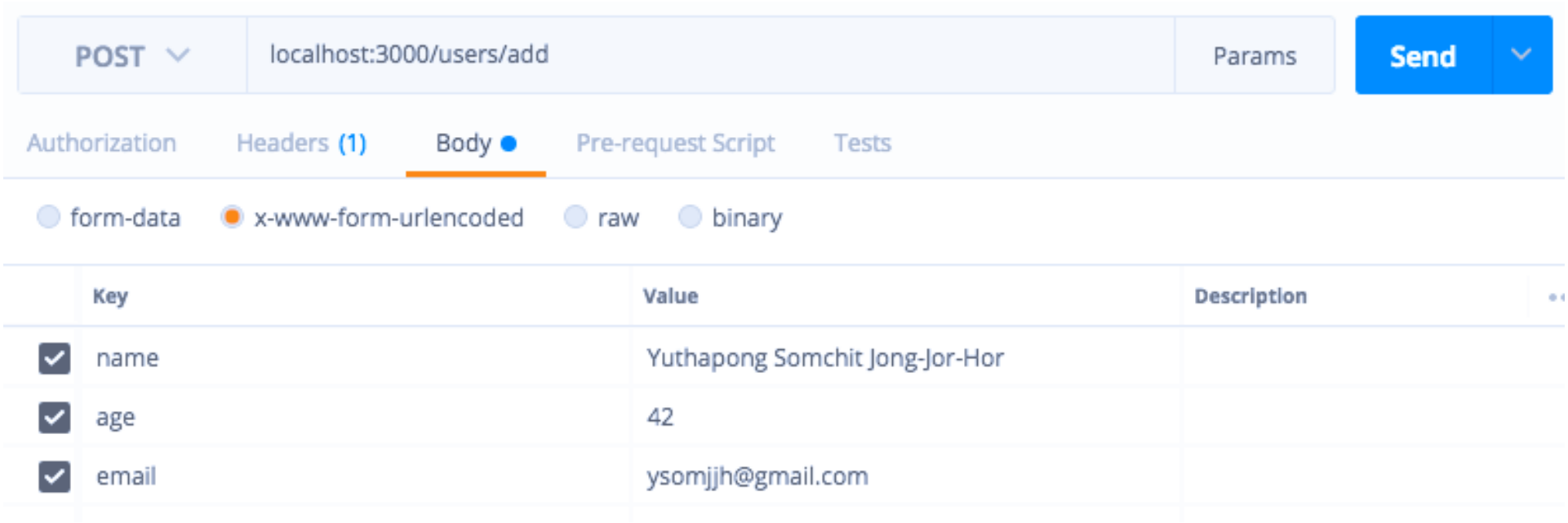
```
…
/* handle POST request */
app.post('/users/add', function(req, res) {
    var newUser = {
        name: req.body.name,
        age: parseInt(req.body.age),
        email: req.body.email
    }

    people.push(newUser);          // add new user to array

    res.render('index', {          // redirect to '/'
        title: 'Customer List',
        users: people
    });
});
…
```

Simulate Form Input

- Postman app – create POST request with URL-encoded



Simulate Form Input (2)

- CURL– create POST request with URL-encoded
  - The same as sending URL parameters
  - Using --data option

```
# curl  URL  --data <URL parameter list>
```

- Example – these two give the same result

```
# curl http://localhost:3000/users/add
       --data "name=Somchai Klaifan&age=21&
               email=sklaifan@gmail.com"


# curl http://localhost:3000/users/add
       -d "name=Somchai Klaifan"
       -d "age=21" -d "email=sklaifan@gmail.com"
```

Express – Mongo Database

- Mongojs – a Node.js module for Mongodb
  - https://www.npmjs.com/package/mongojs
- Emulates the official mongodb API as much as possible
- Installation

```
# npm  install mongojs --save       // install
```

- Import JSON file into local Mongo database

```
# mongoimport -d <db> -c <collection> --drop -u <dbuser>
              -p <dbpass> --file <JSON file> [--jsonArray]
```

Express – Mongo Database (2)

- Import and setup

```
var mongojs = require('mongojs');

/* get access to 'users' collection in 'mydb' database*/
var db = mongojs('mydb', ['users']);
```

- Find all documents and return as JSON

```
app.get('/user', (request, response) => {
    db.users.find( function (err, docs) {
        if(err) {
            console.error(err);
            response.json({ status: false });
        }
        response.json(docs);
    });
});
```

Express – Mongo Database (2)

- Find all document(s) and render in HTML template

```
app.get('/', (request,response) => {
    db.users.find( function (err, docs) {
        if(err) {
            console.error(err);
        }
        /* passing data object to render in index */
        response.render('index', {
            title: 'Customer DB list:',
            users: docs
        });
    });
});
```

Express – Mongo Database (2)

- Find just one document in the collection by '**id**' property

```
app.get('/user/:id', (req, res) => {
    var id = parseInt(req.params.id);

    db.users.findOne({id: id}, function(err, doc) {

        if (doc) {
            /* if found, return the document */
            res.json(doc);
        } else {
            /* if not, return custom error object */
            res.json({ status: false });
        }
    });
});
```

Express – Mongo Database (3)

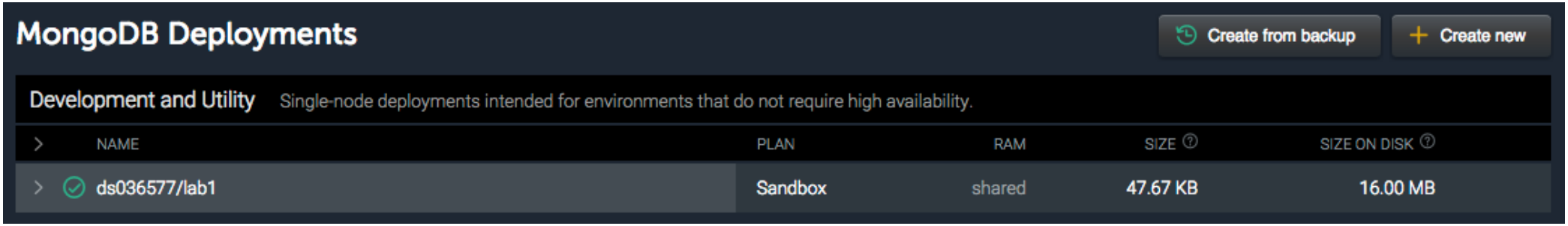
- Add document to collection

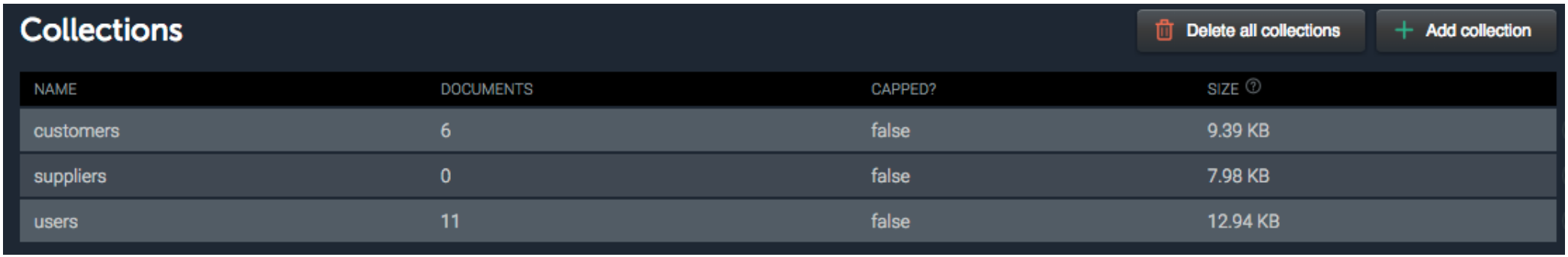
```
app.post('/user/add', (request, response) => {
    if(validateNewUser(request.body)) {
        var newUser = {
            name: request.body.name,
            age: parseInt(request.body.age),
            email: request.body.email
        }
        db.users.insert(newUser, function(err, result) {
            if(err){
                console.log(err);
            }
            response.redirect('/');
            console.log('New user has beed added.')
        });
    }
});
```

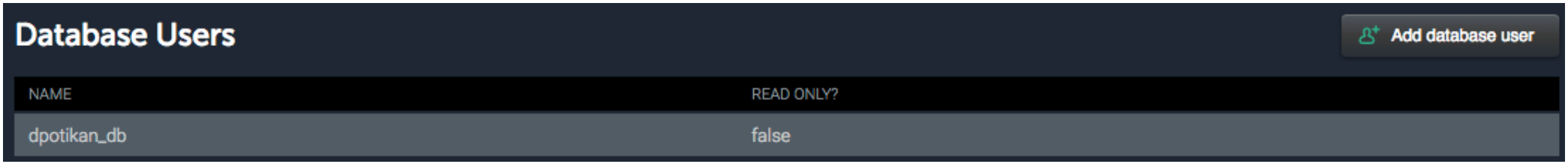

mLab
https://mlab.com/

mLab – MongoDB as a Service

- Database, collections and users

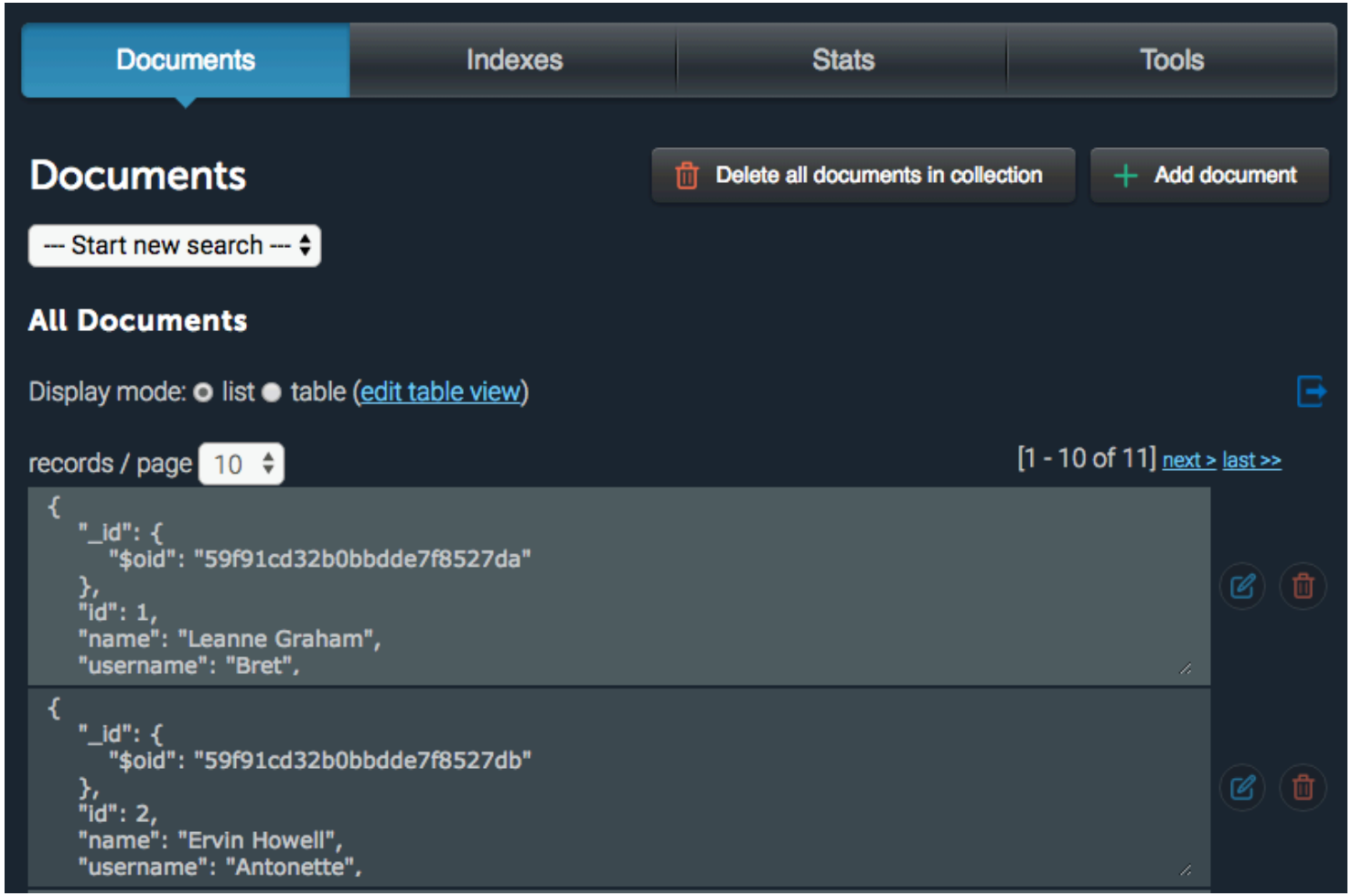






mLab – MongoDB as a Service (2)

- Documents – users (imported from users.json)



mLab – MongoDB as a Service (3)

- Setup connection for **mongojs**

```
var db = mongojs('<dbuser>:<dbpass>@<mlab-server:port>/<db>',
                 ['<collection>']);
```

- To connect using the mongo shell

```
# mongo <mlab-server:port>/<db> -u <dbuser> -p <dbpass>
```

- To import JSON file

```
# mongoimport -h <mlab-server:port> -d <db> -c <collection>
    -u <dbuser> -p <dbpass> --file <JSON file> --jsonArray
```

Note: all parameters can be found on **mLab's database dashboard**

References

- Node.js Tutorial For Absolute Beginners (Traversy Media)
  - https://www.youtube.com/watch?v=U8XF6AFGqlc&index=20&list=WL
- Node.js Event Loop and Timer
  - https://nodejs.org/en/docs/guides/event-loop-timers-and-nexttick/
- Concurrency model and Event Loop
  - https://developer.mozilla.org/en-US/docs/Web/JavaScript/EventLoop

References

- Simple API development using Node.js and Express (sitmhtml.com)
  - http://www.siamhtml.com/restful-api-with-node-js-and-express/
- RESTFul API using Node.js, Express and MongoDB (devahoy.com)
  - https://devahoy.com/posts/restful-api-with-node-js-and-mongodb/
- ExpressJs Middleware for Full Stack Dev (AlgorithmTut.com)
  - https://www.algorithmtut.com/%E0%B8%A3%E0%B8%A7%E0%B8%9A%E0%B8%A3%E0%B8%A7%E0%B8%A1-expressjs-middleware-%E0%B8%97%E0%B8%B5%E0%B9%88%E0%B8%84%E0%B8%A7%E0%B8%A3%E0%B9%83%E0%B8%8A%E0%B9%89%E0%B8%AA%E0%B8%B3%E0%B8%AB%E0%B8%A3/

References

- Postman Request
  - https://www.getpostman.com/docs/postman/sending_api_requests/requests
- Using CURL to automate HTTP jobs
  - https://curl.haxx.se/docs/httpscripting.html
- POST example with CURL
  - https://gist.github.com/joyrexus/524c7e811e4abf9afe56
- Import data to mLab
  - http://docs.mlab.com/migrating/#import