



BLG 411E  
Software  
Engineering

Recitation 3

Introduction  
Features  
Environment Setup

REPL  
Terminal

NPM

Concepts  
Callback  
EventEmitter  
Buffer  
Stream  
File Operations  
Timeout  
Child Process  
WebServer  
Express  
REST Architecture  
Database connection

References

# BLG 411E – Software Engineering

## Recitation Session 3

### Node.js

Bilge Süheyla Akkoca, Müge Erel Özçevik, Beyza Eken

24.10.2017



# Outline

BLG 411E  
Software  
Engineering

Recitation 3

Introduction  
Features  
Environment Setup

REPL  
Terminal

NPM

Concepts  
Callback  
EventEmitter  
Buffer  
Stream  
File Operations  
Timeout  
Child Process  
WebServer  
Express  
REST Architecture  
Database connection

References

## 1 Introduction

- Features
- Environment Setup

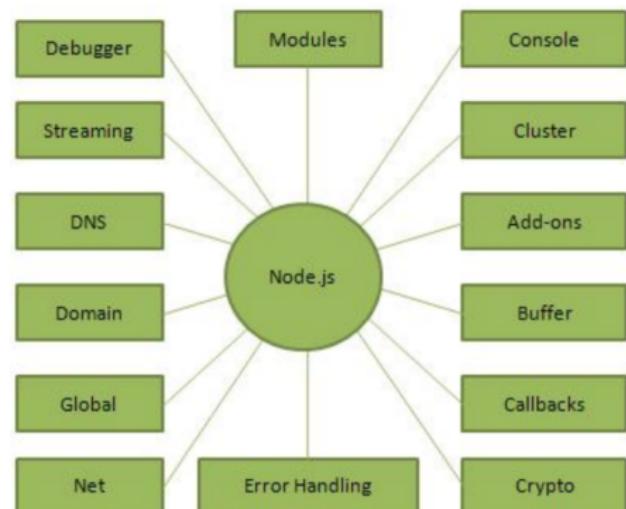
## 2 REPL Terminal

## 3 NPM

## 4 Concepts

- Callback
- EventEmitter
- Buffer
- Stream
- File Operations
- Timeout
- Child Process
- WebServer
- Express
- REST Architecture
- Database connection

- JavaScript-based framework/platform
- Used for web applications
- Asynchronous, Event-driven, non-blocking I/O
- Single threaded
- Open source
- **Prerequisites:**
  - JavaScript, HTML, CSS, AJAX



- Use a text editor
  - Windows Notepad, OS Edit command, vim or vi etc.
  - example.js
- Download Node.js archive
  - Windows: node-v6.3.1-x64.msi
  - Linux: node-v6.3.1-linux-x86.tar.gz
  - Mac: node-v6.3.1-darwin-x86.tar.gz

### Installation on UNIX/Linux/Mac OS X

- cd /tmp
- wget <http://nodejs.org/dist/v6.3.1/node-v6.3.1-linux-x64.tar.gz>
- tar xvfz node-v6.3.1-linux-x64.tar.gz
- mkdir -p /usr/local/nodejs
- mv node-v6.3.1-linux-x64/\* /usr/local/nodejs

- Add PATH

- export PATH=\$PATH:/usr/local/nodejs/bin

## REPL Examples

**node****>1+3**

4

**>x=10**

10

**>var y=10**

undefined

**>x+y**

20

**> var sum = \_**

undefined

**> console.log(sum)**

20

## Example commands:

- **ctrl + c**
- **ctrl + c** twice
- **ctrl + d**
- **Up/Down Keys**
- **tab Keys**
- **.help**
- **.break**
- **.save filename**
- **.load filename**

- Node.js packages/modules
- Enabling to install packages from command line

### npm installation

**npm - -version**

2.7.1

**sudo npm install npm -g**

### Module installation, uninstallation

**npm install express -g**

**npm ls -g** to check installed modules

**npm uninstall express**

## ■ Completion of I/O operations

### Blocking code example

The screenshot shows a Geany code editor window titled "callback.js - /Users/mugeerel/Desktop/nodejs/Callback - Geany". The code in "callback.js" reads the contents of "input.txt" and logs them to the console. The terminal window below shows the output: "HELLO WORLD!!!" followed by "Program Ended".

```
1 var fs = require("fs");
2
3 var data = fs.readFileSync('input.txt');
4
5 console.log(data.toString());
6 console.log("Program Ended");
```

Muge-MacBook-Pro:Callback mugeerel\$ node callback.js  
HELLO WORLD!!!  
Program Ended

## Non-Blocking code example

The screenshot shows the Geany IDE interface with a file named 'callback\_non\_blocking.js' open. The code reads 'input.txt' and logs its contents to the console. It also logs a message when the program ends.

```
callback_non_blocking.js - /Users/mugeerel/Desktop/nodejs/Callback - Geany
input.txt x callback_non_blocking.js x
1 var fs = require("fs");
2
3 fs.readFile('input.txt', function (err, data) {
4     if (err) return console.error(err);
5     console.log(data.toString());
6 });
7
8 console.log("Program Ended");
9
```

Below the code editor, a terminal window shows the output of running the script:

```
Muge-MacBook-Pro:Callback mugeerel$ node callback_non_blocking.js
Program Ended
HELLO WORLD!!!
```

# EventEmitter Class

## Introduction

Features

Environment Setup

## REPL

Terminal

## NPM

## Concepts

Callback

EventEmitter

Buffer

Stream

File Operations

Timeout

Child Process

WebServer

Express

REST Architecture

Database connection

## References

```
EventEmitter.js x
1 // Import events module
2 var events = require('events');
3
4 // Create an eventEmitter object
5 var eventEmitter = new events.EventEmitter();
6
7 // Create an event handler as follows
8 var connectHandler = function connected() {
9   console.log('connection succesful.');
10
11   // Fire the data_received event
12   eventEmitter.emit('data_received');
13 }
14
15 // Bind the connection event with the handler
16 eventEmitter.on('connection', connectHandler);
17
18 // Bind the data_received event with the anonymous
19 var dataReceivedHandler = function(){
20   console.log('data received succesfully.');
21 };
22
23 // Fire the connection event
24 eventEmitter.emit('connection');
25
26 console.log("Program Ended.");
```

```
EventEmitter — bash — 61x18
~/Desktop/nodejs/EventEmitter — bash
Muge-MacBook-Pro: EventEmitter mugeerel$ node EventEmitter.js
connection successful.
data received succesfully.
Program Ended.
Muge-MacBook-Pro: EventEmitter mugeerel$
```

## Introduction

Features

Environment Setup

REPL

Terminal

NPM

Concepts

Callback

EventEmitter

Buffer

Stream

File Operations

Timeout

Child Process

WebServer

Express

REST Architecture

Database connection

References

```
buffer.js x
1  buf = new Buffer(26);
2  for (var i = 0 ; i < 26 ; i++) {
3    buf[i] = i + 97;
4  }
5
6  console.log( buf.toString('ascii')); // outputs:
7  console.log( buf.toString('ascii',0,5)); // outputs:
8  console.log( buf.toString('utf8',0,5)); // outputs:
9  console.log( buf.toString(undefined,0,5)); // encoding
10
11 var buffer1 = new Buffer('HELLO WORLD!!!!');
12 var buffer2 = new Buffer('HELLO BLG411E');
13 var buffer3 = Buffer.concat([buffer1,buffer2]);
14 console.log("buffer3 content: " + buffer3.toString());
15
16 var result = buffer1.compare(buffer2);
17
18 if(result < 0) {
19   console.log(buffer1 +" comes before " + buffer2);
20 }else if(result == 0){
21   console.log(buffer1 +" is same as " + buffer2);
22 }else {
23   console.log(buffer1 +" comes after " + buffer2);
24 }
```

```
Muge-MacBook-Pro:Buffers mugeerel$ node buffer.js
abcdefghijklmnopqrstuvwxyz
abcde
abcde
abcde
abcde
buffer3 content: HELLO WORLD!!!!HELLO BLG411E
HELLO WORLD!!! comes after HELLO BLG411E
Muge-MacBook-Pro:Buffers mugeerel$
```

## Read/Write example

```
master_exec.js x master_fork.js x master_spawn.js x support.js x stream.js x
1 var fs = require("fs");
2 var data = '';
3
4 /*Reading a file*/
5
6 // Create a readable stream
7 var readerStream = fs.createReadStream('input.txt');
8
9 // Set the encoding to be utf8.
10 readerStream.setEncoding('UTF8');
11
12 // Handle stream events --> data, end, and error
13 readerStream.on('data', function(chunk) {
14     data += chunk;
15 });
16
17 readerStream.on('end',function(){
18     console.log(data);
19 });
20
21 readerStream.on('error', function(err){
22     console.log(err.stack);
23 });
24
25 /*Writing to file*/
26
27 var data = 'FALL 2017';
28
29 // Create a writable stream
30 var writerStream = fs.createWriteStream('output.txt');
31
32 // Write the data to stream with encoding to be utf8
33 writerStream.write(data,'UTF8');
34
35 // Mark the end of file
36 writerStream.end();
37
38 // Handle stream events --> finish, and error
39 writerStream.on('finish', function() {
40     console.log("Writing to file completed.");
41 });
42
43 writerStream.on('error', function(err){
44     console.log(err.stack);
45 });
46
47 console.log("Program Ended");
```

```
input.txt
HELLO BLG411E!!!
output.txt
FALL 2017,
File&Streams — bash — 80x24
~/Desktop/nodejs/File&Streams — bash
cdLast login: Tue Oct 17 13:27:14 on console
Muge-MacBook-Pro:~ mugeerel$ cd Desktop
Muge-MacBook-Pro:Desktop mugeerel$ cd nodejs/File&Streams/
Muge-MacBook-Pro:File&Streams mugeerel$ node stream.js
Program Ended
Writing to file completed.
FALL 2017,HELLO BLG411E!!!
Muge-MacBook-Pro:File&Streams mugeerel$
```

## Introduction

Features

Environment Setup

## REPL

Terminal

## NPM

## Concepts

Callback

EventEmitter

Buffer

Stream

File Operations

Timeout

Child Process

WebServer

Express

REST Architecture

Database connection

## References

```
fileoperations.js x
1  var fs = require("fs");
2
3  // Asynchronous read
4  fs.readFile('input.txt', function (err, data) {
5    if (err) {
6      return console.error(err);
7    }
8    console.log("Asynchronous read: " + data.toString());
9  });
10
11
12 // Synchronous read
13 var data = fs.readFileSync('input.txt');
14 console.log("Synchronous read: " + data.toString());
15
16 fs.stat('input.txt', function (err, stats) {
17   if (err) {
18     return console.error(err);
19   }
20   console.log(stats);
21   console.log("Got file info successfully!");
22
23   // Check file type
24   console.log("isFile ? " + stats.isFile());
25   console.log("isDirectory ? " + stats.isDirectory());
26 });
27
28 console.log("Program Ended");
```

```
File&Streams -- bash -- b5x26
~/Desktop/nodes/File&Streams -- bash
Muge-MacBook-Pro:File&Streams mugeerel$ node fileoperations.js
Synchronous read: HELLO BLG411E!!!
Program Ended
Stats {
  dev: 16777220,
  mode: 33188,
  nlink: 1,
  uid: 501,
  gid: 20,
  rdev: 0,
  blksize: 4096,
  ino: 24068339,
  size: 17,
  blocks: 8,
  atime: 2017-10-15T10:38:26.000Z,
  mtime: 2017-10-15T10:34:14.000Z,
  ctime: 2017-10-15T10:34:14.000Z,
  birthtime: 2017-10-15T10:34:14.000Z
}
Got file info successfully!
.isFile ? true
.isDirectory ? false
Asynchronous read: HELLO BLG411E!!!
Muge-MacBook-Pro:File&Streams mugeerel$
```

## Timeout and Interval example

A screenshot of a terminal window titled "node timeout.js". The window shows the command being run and its output. The code in the terminal is:

```
timeout.js x
1 function printHello(){
2   console.log( "Hello, World!" );
3 }
4 // Now call above function after 2 seconds
5 setTimeout(printHello, 2000);
6
7 setInterval(printHello, 2000); //periodic
8
```

The output of the code is:

```
Muge-MacBook-Pro:Timeout mugeerel$ node timeout.js
Hello, World!
```

## Introduction

Features

Environment Setup

## REPL

Terminal

## NPM

## Concepts

Callback

EventEmitter

Buffer

Stream

File Operations

Timeout

## Child Process

WebServer

Express

REST Architecture

Database connection

## References

The screenshot shows a terminal window titled "ChildProcess — bash — 80x24" running on a MacBook Pro. The command executed is `node master\_exec.js`. The terminal output shows three child processes being spawned and their exit codes:

```
Muge-MacBook-Pro:ChildProcess mugeerel$ node master_exec.js
Child process exited with exit code 0
Child process exited with exit code 0
stdout: Child Process 1 executed.

stderr:
Child process exited with exit code 0
stdout: Child Process 2 executed.

stderr:
stdout: Child Process 0 executed.

stderr:
Muge-MacBook-Pro:ChildProcess mugeerel$
```

Below the terminal, the code for `master_exec.js` is displayed in a code editor. The code uses the `child_process` module to spawn three child processes, each executing `support.js` with a different argument (`i`). The `support.js` file contains a `console.log` statement that outputs the process ID and the word "executed".

```
const fs = require('fs');
const child_process = require('child_process');

for(var i=0; i<3; i++) {
  var workerProcess = child_process.exec('node support.js '+i,function
    (error, stdout, stderr) {

      if (error) {
        console.log(error.stack);
        console.log('Error code: '+error.code);
        console.log('Signal received: '+error.signal);
      }
      console.log('stdout: ' + stdout);
      console.log('stderr: ' + stderr);
    });
}

workerProcess.on('exit', function (code) {
  console.log('Child process exited with exit code ' +code);
});
```

## Creating a new process

The screenshot shows a code editor with four tabs: master\_exec.js, master\_fork.js, master\_spawn.js, and support.js. The master\_spawn.js tab is active, displaying the following code:

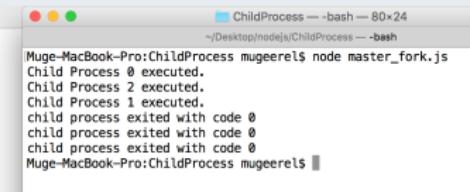
```
1 const fs = require('fs');
2 const child_process = require('child_process');
3
4 for(var i = 0; i<3; i++) {
5     var workerProcess = child_process.spawn('node', ['support.js', i]);
6
7     workerProcess.stdout.on('data', function (data) {
8         console.log('stdout: ' + data);
9     });
10
11    workerProcess.stderr.on('data', function (data) {
12        console.log('stderr: ' + data);
13    });
14
15    workerProcess.on('close', function (code) {
16        console.log('child process exited with code ' + code);
17    });
18}
19
```

To the right of the code editor is a terminal window titled "ChildProcess — bash — 80x24". The terminal shows the command "node master\_spawn.js" being run, followed by three lines of output:

```
Muge-MacBook-Pro:ChildProcess mugeerel$ node master_spawn.js
stdout: Child Process 2 executed.
stdout: Child Process 1 executed.
child process exited with code 0
stdout: Child Process 0 executed.
child process exited with code 0
child process exited with code 0
Muge-MacBook-Pro:ChildProcess mugeerel$
```

## Creating a child process

```
master_exec.js x master_fork.js x master_spawn.js x support.js x
1  const fs = require('fs');
2  const child_process = require('child_process');
3
4  for(var i=0; i<3; i++) {
5    var worker_process = child_process.fork("support.js", [i]);
6
7    worker_process.on('close', function (code) {
8      console.log('child process exited with code ' + code);
9    });
10 }
11
```



```
Muge-MacBook-Pro:ChildProcess mugeerel$ node master_fork.js
Child Process 0 executed.
Child Process 2 executed.
Child Process 1 executed.
child process exited with code 0
child process exited with code 0
child process exited with code 0
Muge-MacBook-Pro:ChildProcess mugeerel$
```

## Introduction

Features

Environment Setup

## REPL

## Terminal

## NPM

## Concepts

Callback

EventEmitter

Buffer

Stream

File Operations

Timeout

Child Process

## WebServer

Express

REST Architecture

Database connection

## References

## Hello World

```
WebServer.js x index.htm x
1  var http = require('http');
2  var fs = require('fs');
3  var url = require('url');
4
5  // Create a server
6  http.createServer( function (request, response) {
7      // Parse the request containing file name
8      var pathname = url.parse(request.url).pathname;
9
10     // Print the name of the file for which request is made.
11     console.log("Request for " + pathname + " received.");
12
13     // Read the requested file content from file system
14     fs.readFile(pathname.substr(1), function (err, data) {
15         if (err) {
16             //console.log(err);
17             // HTTP Status: 404 : NOT FOUND
18             // Content Type: text/plain
19             response.writeHead(404, {'Content-Type': 'text/html'});
20         } else {
21             //Page found
22             // HTTP Status: 200 : OK
23             // Content Type: text/plain
24             response.writeHead(200, {'Content-Type': 'text/html'});
25
26             // Write the content of the file to response body
27             response.write(data.toString());
28         }
29         // Send the response body
30         response.end();
31     });
32 }).listen(8081);
33
34 // Console will print the message
35 //console.log('Server running at http://127.0.0.1:8081/');
36
```

127.0.0.1:8081/index.htm

Hello World!

```
WebServer&json — node WebServer.js — 59×14
~/Desktop/nodejs/WebServer&json — node WebServer.js
+Last login: Sun Oct 15 13:47:45 on ttys002
Muge-MacBook-Pro:~ mugeerel$ cd Desktop
Muge-MacBook-Pro:Desktop mugeerel$ cd nodejs/WebServer\&json\
Muge-MacBook-Pro:WebServer&json mugeerel$ node WebServer.js

Server running at http://127.0.0.1:8081/
Request for /index.htm received.
```

- A framework that enables robust features for web and mobile applications
- Based on HTTP method and URL:
  - **GET:** read only access to resource
  - **PUT:** create a new resource
  - **DELETE:** remove a resource
  - **POST:** update existing resource or create a new resource
- Offers dynamic HTML pages that take arguments

## Installation

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

```
npm install body-parser - -save
```

```
npm install cookie-parser - -save
```

```
npm install multer - -save
```

## Name &amp; Password: index.htm

```
WebServer2.js x index.htm x
1 1<!DOCTYPE html>
2 2<html lang="eng">
3 3  <head>
4 4    <meta charset="iso-8859-9">
5 5    <title>Web Server</title>
6 6    <meta name="viewport" content="width=device-width, initial-scale=1, maximum-scale=1, user-scalable=no"/>
7 7
8 8    <meta name="apple-mobile-web-app-capable" content="yes">
9 9    <meta name="apple-mobile-web-app-status-bar-style" content="black-translucent">
10 10   </head>
11 11   <body>
12 12     <form action = "http://127.0.0.1:8081/process_get" method = "GET">
13 13       User Name : <input type = "text" name = "name"> <br>
14 14       Password : <input type = "password" name = "password"> <br>
15 15       <input type = "submit" value = "Submit">
16 16     </form>
17 17   </body>
18 18 </html>
19 19
20 20
```

## Introduction

Features

Environment Setup

## REPL

Terminal

## NPM

## Concepts

Callback

EventEmitter

Buffer

Stream

File Operations

Timeout

Child Process

WebServer

## Express

REST Architecture

Database connection

## References

## Name &amp; Password: / index.htm

```
WebServer.js  x  index.htm  x
1  var express = require('express');
2  var app = express();
3  var bodyParser = require('body-parser');
4
5  // Create application/x-www-form-urlencoded parser
6  var urlencodedParser = bodyParser.urlencoded({ extended: false
7
8  app.use(express.static('public'));
9  app.get('/index.htm', function (req, res) {
10    res.sendFile(__dirname + "/" + "index.htm" );
11  })
12
13 app.get('/process_get', function (req, res) {
14   // Prepare output in JSON format
15   response = {
16     name:req.query.name,
17     password:req.query.password
18   };
19   console.log(response);
20   res.end(JSON.stringify(response));
21 })
22
23 app.post('/process_post', urlencodedParser, function (req, res) {
24   // Prepare output in JSON format
25   response = {
26     name:req.body.name,
27     password:req.body.password
28   };
29   console.log(response);
30   res.end(JSON.stringify(response));
31 })
32
33 var server = app.listen(8081, function () {
34   var host = server.address().address
35   var port = server.address().port
36   console.log("Example app listening at http://%s:%s", host, port)
37 })
```

User Name : BLG411E  
Password : \*\*\*\*\*  
Submit

```
Last login: Mon Oct 16 11:14:41 on ttys001
Muge-MacBook-Pro:~ mugeerel$ cd Desktop
Muge-MacBook-Pro:Desktop mugeerel$ cd nodejs/WebServe
Muge-MacBook-Pro:WebServer&json mugeerel$ node WebSer
Example app listening at http://:::8081
```

## Introduction

Features

Environment Setup

## REPL

## Terminal

## NPM

## Concepts

Callback

EventEmitter

Buffer

Stream

File Operations

Timeout

Child Process

WebServer

Express

REST Architecture

Database connection

## References

Name & Password: /  
process\_get?name=BLG411E&password=12345

The screenshot shows a terminal window with the following text:

```
Last login: Mon Oct 16 11:14:41 on ttys001
Muge-MacBook-Pro:~ mugeerel$ cd Desktop
Muge-MacBook-Pro:Desktop mugeerel$ cd nodejs/WebServer
Muge-MacBook-Pro:WebServer&json mugeerel$ node WebSer
Example app listening at http://:8081
{ name: 'BLG411E', password: '12345' }
```

Below the terminal is a code editor window showing a file named WebServer2.js. The code defines an Express application that handles two routes: '/process\_get' and '/process\_post'. Both routes use a middleware function to parse URL-encoded bodies. The '/process\_get' route returns a JSON object with 'name' and 'password' fields. The '/process\_post' route also returns a similar JSON object. Finally, the server is listened on port 8081.

```
WebServer2.js | index.htm |
1 var express = require('express');
2 var app = express();
3 var bodyParser = require('body-parser');
4
5 // Create application/x-www-form-urlencoded parser
6 var urlencodedParser = bodyParser.urlencoded({ extended: false });
7
8 app.use(express.static('public'));
9 app.get('/index.htm', function (req, res) {
10   res.sendFile(__dirname + "/" + "index.htm");
11 })
12
13 app.get('/process_get', function (req, res) {
14   // Prepare output in JSON format
15   response = {
16     name: req.query.name,
17     password: req.query.password
18   };
19   console.log(response);
20   res.end(JSON.stringify(response));
21 })
22
23 app.post('/process_post', urlencodedParser, function (req, res) {
24   // Prepare output in JSON format
25   response = {
26     name: req.body.name,
27     password: req.body.password
28   };
29   console.log(response);
30   res.end(JSON.stringify(response));
31 })
32
33 var server = app.listen(8081, function () {
34   var host = server.address().address
35   var port = server.address().port
36   console.log("Example app listening at http://%s:%s", host, port)
37 })
```

- REpresentational State Transfer
- Web based architecture
- REST Server and REST Client
- HTTP methods (GET, PUT, DELETE, POST)
- Resource is identified with global ids
- JSON format is most popular one

## Introduction

Features

Environment Setup

## REPL

Terminal

## NPM

## Concepts

Callback

EventEmitter

Buffer

Stream

File Operations

Timeout

Child Process

WebServer

Express

REST Architecture

Database connection

## References

## REST Example /listUsers

The screenshot shows a browser window with the URL `127.0.0.1:8081/listUsers`. The page displays a JSON response from a REST API. The JSON object contains three user entries: "user1", "user2", and "user3". Each user has properties: name, password, degree, and id.

```
1 var express = require('express');
2 var app = express();
3 var fs = require("fs");
4
5 app.get('/listUsers', function (req, res) {
6   fs.readFile(__dirname + "/" + "users.json", 'utf8', function (err, data) {
7     console.log( data );
8     res.end( data );
9   });
10 }
11
12 app.get('/:id', function (req, res) {
13   // First read existing users.
14   fs.readFile(__dirname + "/" + "users.json", 'utf8', function (err, data) {
15     var users = JSON.parse(data);
16     var user = users["user" + req.params.id];
17     console.log(user);
18     res.end(JSON.stringify(user));
19   });
20 }
21
22 app.delete('/deleteUser', function (req, res) {
23   // First read existing users.
24   fs.readFile(__dirname + "/" + "users.json", 'utf8', function (err, data) {
25     data = JSON.parse(data);
26     delete data["user" + 2];
27
28     console.log( data );
29     res.end( JSON.stringify(data));
30   });
31 }
32 }
```

```
{
  "user1" : {
    "name" : "muge",
    "password" : "12345",
    "degree" : "PhD Candidate",
    "id": 1
  },
  "user2" : {
    "name" : "beyza",
    "password" : "54321",
    "degree" : "PhD Candidate",
    "id": 2
  },
  "user3" : {
    "name" : "bilge",
    "password" : "345",
    "degree" : "PhD Candidate",
    "id": 3
  }
}
```

Introduction

NPM

## Concepts

REST Architecture

The screenshot shows a browser window at `127.0.0.1:8081/` displaying the JSON object:

```
{"name": "muge", "password": "12345", "degree": "PhD Candidate", "id": 1}
```

Below the browser is a code editor with a file named `WebServer3.js`. The code defines a Node.js application using Express. It includes three routes: `/listUsers` (GET), `/:id` (GET), and `/deleteUser` (DELETE). The `/listUsers` route reads the `users.json` file. The `/:id` route reads the `users.json` file, parses it into an array, and returns the user with the specified ID. The `/deleteUser` route reads the `users.json` file, removes the user with the specified ID from the array, and returns the modified array.

```
1 var express = require('express');
2 var app = express();
3 var fs = require("fs");
4
5 app.get('/listUsers', function (req, res) {
6   fs.readFile(__dirname + "/" + "users.json", 'utf8'
7     , console.log( data );
8   res.end( data );
9 });
10
11
12 app.get('/:id', function (req, res) {
13   // First read existing users.
14   fs.readFile(__dirname + "/" + "users.json", 'utf8'
15     , var users = JSON.parse( data );
16     var user = users["user" + req.params.id]
17     console.log( user );
18   res.end( JSON.stringify(user));
19 });
20
21
22 app.delete('/deleteUser', function (req, res) {
23
24   // First read existing users.
25   fs.readFile(__dirname + "/" + "users.json", 'utf8'
26     , data = JSON.parse( data );
27     delete data["user" + 2];
28
29   console.log( data );
30   res.end( JSON.stringify(data));
31 });
32 })
```

## Requirements:

- 1 Nodejs (npm i knex mysql express body-parser - -save)
- 2 MySQL etc.
- 3 HTTP API to write database
- 4 HTML and JS files to POST to the API
- 5 Knex migration to create user table in database
- 6 Use migration to add name and password to user table in database

# Database connection

## 3. HTTP API to write database

BLG 411E  
Software  
Engineering

Recitation 3

Introduction

Features

Environment Setup

REPL

Terminal

NPM

Concepts

Callback

EventEmitter

Buffer

Stream

File Operations

Timeout

Child Process

WebServer

Express

REST Architecture

Database connection

References

## index.htm, app.js under public folder

The screenshot shows a code editor with two files open: index.html and app.js. Both files are located in a directory named 'public'.

**index.html:**

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title>Node database tutorial</title>
5   </head>
6   <body>
7     <form class="CreateUser">
8       <h1>Create a new user</h1>
9       <input type="text" class="username" placeholder="username">
10      <input type="password" class="password" placeholder="password">
11      <input type="submit" value="Create user">
12    </form>
13    <script src="/app.js"></script>
14  </body>
15 </html>
```

**app.js:**

```
1 const CreateUser = document.querySelector('.CreateUser')
2 CreateUser.addEventListener('submit', (e) => {
3   e.preventDefault()
4   const username = CreateUser.querySelector('.username').value
5   const password = CreateUser.querySelector('.password').value
6   post('/createUser', { username, password })
7 })
8
9 function post (path, data) {
10   return window.fetch(path, {
11     method: 'POST',
12     headers: {
13       'Accept': 'application/json',
14       'Content-Type': 'application/json'
15     },
16     body: JSON.stringify(data)
17   })
18 }
```

# Database connection

## 4. HTML and JS files to POST to the API

BLG 411E  
Software  
Engineering

Recitation 3

Introduction  
Features  
Environment Setup

REPL  
Terminal

NPM

Concepts  
Callback

EventEmitter

Buffer

Stream

File Operations

Timeout

Child Process

WebServer

Express

REST Architecture

Database connection

References

### index.js, store.js, knexfile.js

```
app.js x index.html x index.js x knexfile.js x store.js x
1 const express = require('express')
2 const bodyParser = require('body-parser')
3 const store = require('./store')
4 const app = express()
5 app.use(express.static('public'))
6 app.use(bodyParser.json())
7
8 app.post('/createUser', (req, res) => {
9   store
10    .createUser({
11      username: req.body.username,
12      password: req.body.password
13    })
14    .then(() => res.sendStatus(200))
15  })
16 app.listen(8081, () => {
17   console.log('Server running on http://localhost:8081')
18 })
19
```

```
knexfile.js
1 module.exports = {
2   client: 'mysql',
3   connection: {
4     user: 'root',
5     password: 'password',
6     database: 'tutorial_database'
7   }
8 }
```

```
store.js
1 const knex = require('knex')(require('./knexfile'))
2
3 module.exports = {
4   createUser ({ username, password }) {
5     console.log(`Add user ${username} with password ${password}`)
6     return knex('user').insert({
7       username,
8       password
9     })
10   }
11 }
```

### Creating a user table

```
knex migrate:make create_user_table
```

- migrations/20171015114925\_create\_user\_table.js
  - new migration file to add, modify, delete a table or column
  - checking schema changes instead of using MySQL directly
  - using js file instead of SQL
  - keep database schema up to date for all developers

# Database connection

## 5. Knex migration to create user table in database

BLG 411E  
Software  
Engineering

Recitation 3

Introduction

Features  
Environment Setup

REPL  
Terminal

NPM

Concepts

Callback

EventEmitter

Buffer

Stream

File Operations

Timeout

Child Process

WebServer

Express

REST Architecture

Database connection

References

- up method: create table
- down method: delete function for roll back

### modify migrations/20171015114925\_create\_user\_table.js

```
app.js x index.html x index.js x knexfile.js x store.js x 20171015114925...e_user_table.js x
1
2 exports.up = function(knex, Promise) {
3   return knex.schema.createTable('user', function (t) {
4     t.increments('id').primary()
5     t.string('username').notNullable()
6     t.string('password').notNullable()
7     t.timestamps(false, true)
8   })
9 };
10
11 exports.down = function(knex, Promise) {
12   return knex.schema.dropTableIfExists('user')
13 };
14
```

## Run migration

knex migrate:latest

(MySQL 5.7.19) Local/tutorial\_database/user

Tables:

- knex\_migrations
- knex\_migrations\_lock
- user**

Search:

	id	username	password	created_at	updated_at
1					

TABLE INFORMATION

- created: 15/10/17
- updated: 15/10/17
- engine: InnoDB
- rows: 3
- size: 16,0 Kib
- encoding: utf8
- auto\_increment: 4

# Database connection

## 6. User migration to add name and password to user table

BLG 411E  
Software  
Engineering

Recitation 3

Introduction

Features

Environment Setup

REPL

Terminal

NPM

Concepts

Callback

EventEmitter

Buffer

Stream

File Operations

Timeout

Child Process

WebServer

Express

REST Architecture

Database connection

References

## Add user to database

The screenshot shows a dual-pane interface. On the left, a web browser window displays a form titled "Create a new user" with fields for "username" (muge) and "password" (\*\*\*\*\*). A "Create user" button is visible. On the right, a MySQL Workbench window shows the "tutorial\_database" schema with a "user" table selected. The table has columns: id, username, password, created\_at, and updated\_at. Below the table, a "TABLE INFORMATION" panel provides details like rows: 3, size: 16,0 ... etc. At the bottom, a terminal window shows the command "node" being run and the message "Server running on http://localhost:8081".

# Database connection

## 6. User migration to add name and password to user table

BLG 411E  
Software  
Engineering

Recitation 3

Introduction

Features

Environment Setup

REPL

Terminal

NPM

Concepts

Callback

EventEmitter

Buffer

Stream

File Operations

Timeout

Child Process

WebServer

Express

REST Architecture

Database connection

References

## Add user to database

The screenshot displays a dual-monitor setup. On the left monitor, a browser window titled 'Google Çeviri' shows a Node.js application at 127.0.0.1:8081. The page has a title 'Create a new user' and two input fields: 'username' containing 'muge' and 'password' containing '12345'. A 'Create user' button is visible. On the right monitor, a MySQL Workbench window titled '(MySQL 5.7.19) Local/tutorial\_database/user' shows the 'user' table. The table has columns: id, username, password, created\_at, and updated\_at. One row is present with id=4, username='muge', password='12345', created\_at='2017-10-16 13:46:03', and updated\_at='2017-10-16 13:46:03'. Below the table, a 'TABLE INFORMATION' panel lists various statistics. At the bottom of the screen, a terminal window on a MacBook Pro shows the command 'Add user muge with password 12345' being run.

	id	username	password	created_at	updated_at
4	muge	12345	2017-10-16 13:46:03	2017-10-16 13:46:03	

Muge-MacBook-Pro:Database mugeerel\$ r...  
Server running on http://localhost:8081  
Add user muge with password 12345

# Database connection

## 6. User migration to add name and password to user table

BLG 411E  
Software  
Engineering

Recitation 3

Introduction

Features

Environment Setup

REPL

Terminal

NPM

Concepts

Callback

EventEmitter

Buffer

Stream

File Operations

Timeout

Child Process

WebServer

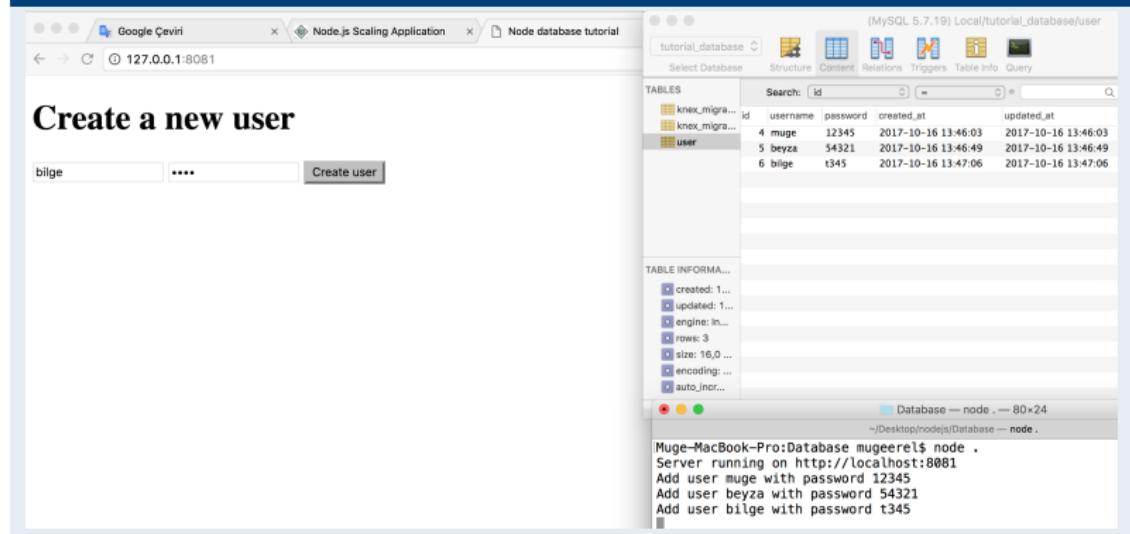
Express

REST Architecture

Database connection

References

## Add user to database



The screenshot illustrates the process of adding a new user to a MySQL database and its subsequent addition to a Node.js application.

**MySQL Database Screenshot:**

- The MySQL interface shows the `tutorial_database` selected.
- The `user` table is displayed with the following data:

id	username	password	created_at	updated_at
4	muge	12345	2017-10-16 13:46:03	2017-10-16 13:46:03
5	beyza	54321	2017-10-16 13:46:49	2017-10-16 13:46:49
6	bilge	t345	2017-10-16 13:47:06	2017-10-16 13:47:06

- The `TABLE INFORMATION` section shows details about the `user` table.

**Node.js Application Screenshot:**

- A terminal window shows the command `node ./app` running on port 8081.
- The output of the application shows the users added:

```
Muge-MacBook-Pro:Database mugeerel$ node .
Server running on http://localhost:8081
Add user muge with password 12345
Add user beyza with password 54321
Add user bilge with password t345
```

# References and Further Reading

BLG 411E  
Software  
Engineering

Recitation 3

Introduction  
Features  
Environment Setup

REPL  
Terminal

NPM

Concepts  
Callback  
EventEmitter  
Buffer  
Stream  
File Operations  
Timeout  
Child Process  
WebServer  
Express  
REST Architecture  
Database connection

References

- `https://www.tutorialspoint.com/nodejs/nodejs_discussion.htm`
- S. Tilkov and S. Vinoski, "Node.js: Using JavaScript to Build High-Performance Network Programs," in IEEE Internet Computing, vol. 14, no. 6, pp. 80-83, Nov.-Dec. 2010. doi: 10.1109/MIC.2010.145, <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5617064&isnumber=5617049>
- T. Bosak and K. Zakova, "Node.js based remote control of thermo-optical plant," Proceedings of 2015 12th International Conference on Remote Engineering and Virtual Instrumentation (REV), Bangkok, 2015, pp. 209-213. doi: 10.1109/REV.2015.7087293, <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7087293&isnumber=7087248>