Udemy NodeJs 55- 97

Udemy:
https://www.udemy.com/understand-nodejs/learn/v4/overview

Command Line Commands
https://www.computerhope.com/issues/chusedos.html
https://www.lifewire.com/list-of-command-prompt-commands-4092302

NodeJs Command Line
https://nodejs.org/api/cli.html

Node,js Native Modules:
https://nodejs.org/api/

Testing it with postman
https://www.getpostman.com/

Section 7 HTTP/Web Server
Lesson 55 TCP/IP
Lesson 56 Adresses and Ports
Lesson 57 HTTP
Lesson 58 HTTP Parser
Lesson 59 Build a webserver in Node
Lesson 60 Outputting HTML and Templates
Lesson 61 Streams and Performance
Lesson 62 API and Endpoints
Lesson_63_Outputting-JSON
Lesson 64 Routing
Section 8 Node Package Manager
Lesson 67 Semantic Versioning
Lesson 69 init, nodemon, and package.json
Section 9 Express
Lesson 73 Installing express
Lesson 74 Routing (variable, parameter)
Lesson 75 Static Files, MiddleWare
Lesson 76 Templates
Lesson 77 Querystring and Post Parameters
Lesson 78 RestFul API Data
Lesson 79 Structure App Data
Section 10 Databases
Lesson 80 Relational databases and SQl
Lesson 81 NodeJs + MySQL
Lesson 82 No SQL Database
Lesson 83 Mongo DB
Section 11 MEAN Stack
Lesson 85 MEAN Stack
Lesson 86 Managing Client
Lesson 87 Managing Client Part(2)

Lesson 88 Managing Client Part(2)
Section 12 Build an App
Lesson 92 inital Setup
Lesson 93 Setup MongoDB Mongoose
Lesson 94 Adding Seed Data
Lesson 94 Creating our API
Lesson 96 Testing Our API
Lesson 97 Adding Front End

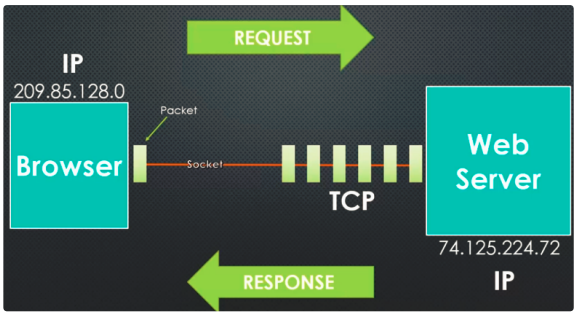
Lesson 55 TCP/IP

Protocoll:
Set of rules two sides agree on to use when communicating.

IP: Internet Protocoll
Socket: Line in which information is actually flowing.

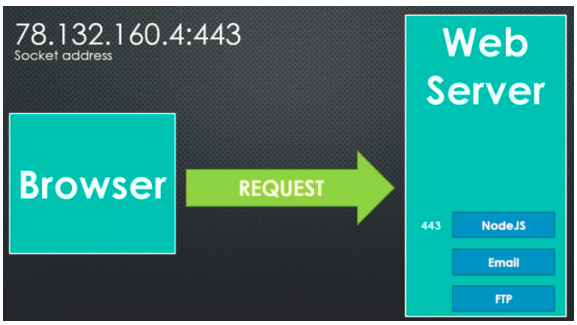
TCP: Transmissopn control protocoll.

HTTPS, FTPS: The structure of the recieved package.



Lesson 56 Adresses and Ports

Address: **IP:Port, so the computer will know where to send the data,**
Port: Once a computer recieved a packet, how it knows what program to send it to.
So bascially, when it recieved data on the port 8080, it will know to send it to a specfic program.



Lesson 57 HTTP

Hyper **T**ext **T**ransfer **P**rotocol,
A set of rules (formats) for data being sent over the WEB.

Headers: Namve value pairs,

HTTP Request:

HTTP Response:

MIME Type:
(**M**ultipurpose**I**nternet**M**ail**E**xentesion)
A standard of specific the type of data being sent.

Lesson 58 HTTP Parser

This is a built-in program, which will make request, parses response in HTTP format.

Lesson 59 Build a webserver in Node

```
// In order to create http request/ parse responses
var http = require('http');


//When that object is emitting an event, it will trigger a listener, a callback
function.
// req --> request
// res --> stream, which we can write.
http.createServer(function (req, res) {


// Determining, the Head, to infrom, it will send text/plain data
  res.writeHead(200, {'Content-Type': 'text/plain'});
//  I am done sending, this will be the last thing to be sent.
```

```
  res.end('Hello World\n');


}).listen(1337, '127.0.0.1');  //Standard Local IP address
//localhost:1137


// 1. Create a response with the built in HTTP module
// 2. Tell node, on which port should it listen for the response.
```

Lesson 60 Outputting HTML and Templates

```
// 1. We create the httprequest
// 2. We determine the content-type
// 3. We read the file, which will read it in little chunkgs, buffers.
// 4. We create are using a simple, string replace to put custom string into the html


var http = require('http');
var fs = require('fs');


http.createServer(function(req, res) {


    res.writeHead(200, { 'Content-Type': 'text/html' });
//Header
    var html = fs.readFileSync(__dirname + '/index.html', 'utf8');
//Reading in 8 bits encoding
    var message = 'Hello world...';
//Custom message string
    html = html.replace('{Message}', message);
//Replacing string
    res.end(html);
//returning the html
```

```
}).listen(1337, '127.0.0.1');
//listen to the port
```

Lesson 61 Streams and Performance

```
//In order use streams, for better performance
//We are reading the html, and creating a writeable response stream
var http = require('http');
var fs = require('fs');

http.createServer(function(req, res) {

    res.writeHead(200, { 'Content-Type': 'text/html' });
    fs.createReadStream(__dirname + '/index.htm').pipe(res);

}).listen(1337, '127.0.0.1');
```

Lesson 62 API and Endpoints

API: ApplicationProgramming Interface
A set of tools for building a software application.

Set of URLS, which recieve and sends Data

Endpoint: 1 Particular URL in a web API.

Lesson_63_Outputting-JSON

```
// 1. We notify the reciever that we are returnin json in the header
// 2. Create an object literal,
// 3. Responding the transformed object literal to JSON.
```

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

http.createServer(function(req, res) {

    res.writeHead(200, { 'Content-Type': 'application/json' });
    var obj = {
        firstname: 'John',
        lastname: 'Doe'
    };
    res.end(JSON.stringify(obj));

}).listen(1337, '127.0.0.1');
```

Lesson 64 Routing

Routing:
Mapping http request to content.

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

//Listening to the req.url string,
http.createServer(function(req, res) {
//Simply, read the file in a readable stream and return the response in a
readable/writeable stream
    if (req.url === '/') {
        fs.createReadStream(__dirname + '/index.htm').pipe(res);
    }
//Retruning JSON
    else if (req.url === '/api') {
```

```
        res.writeHead(200, { 'Content-Type': 'application/json' });
        var obj = {
            firstname: 'John',
            lastname: 'Doe'
        };
        res.end(JSON.stringify(obj));
    }
//Sending the not found response.
    else {
        res.writeHead(404);
        res.end();
    }

}).listen(1337, '127.0.0.1');
```

Lesson 67 Semantic Versioning

Versioning: Specifying what version of a set code this is.
Semantic: sth conveys meaning

Major.Minor.Patch
1.7.2
Patch: Bugs were fixed, code works fine
Minor: Added new features, code works fine.
Major: Big changes, your code might be break.

Lesson 69 init, nodemon, and package.json

```
npm init
```

name: name of the app
description:
entry point: what is the javascript what the node will run
etc..

it will create the **package.json** file which will tell node.js about what dependecies should be installed, what are the configurations.

```
{
  "name": "nodejs-test-app",
  "version": "1.0.0",
  "description": "NodeJS Test App",
  "main": "app.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "moment": "^2.10.6"
  }
}
```

nodemon:

For use during development of a node.js based application.
nodemon will watch the files in the directory in which nodemon was started, and if any files change, nodemon will automatically restart your node application.

https://www.npmjs.com/package/nodemon

Every dependecies are stored in **node_modules.**
So a lot of time, a dependency has different different dependecies and soo on.

Lesson 73 Installing express

Enviromental variable:
Variable defined on the server, where our code lives. (pl Heroku)

```
// Building a simple web server


var express = require('express');
var app = express();                          //This will be our webserver object
```

```
// process.env.PORT --->                    //If there is an Evniromental variable, it
will be used.
var port = process.env.PORT || 3000;


//Respond with html
app.get('/', function(req, res) {
    res.send('<html><head></head><body><h1>Hello world!</h1></body></html>');
});
//respond with json
app.get('/api', function(req, res) {
    res.json({ firstname: 'John', lastname: 'Doe' });
});


app.listen(port);
```

Lesson 74 Routing (variable, parameter)

```
//You can listen to parameters in the url
app.get('/person/:id', function(req, res) {
    res.send('<html><head></head><body><h1>Person: ' + req.params.id + '</h1></body>
</html>');
});
```

Lesson 75 Static Files, MiddleWare

Enable static files to be accessed

```
app.use('/assets', express.static(__dirname + '/public'));
```

Middleware: Code executed between two layers

```
app.use('/', function (req, res, next) {
    console.log('Request Url:' + req.url);
    next();
});
```

Lesson 76 Templates

You can install **ejs** which is a template rendering, understood by express

1. You have to notify express which template engine should be used
2. You have to specify which template should be "**rendered**" on each route.
3. You can pass paremeters from the request, to the template

app.js

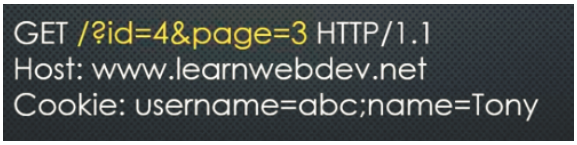
```
//*****1*****//
app.set('view engine', 'ejs');
//*****2*****//
app.get('/', function(req, res) {
    res.render('index');
});
//*****3*****//
app.get('/person/:id', function(req, res) {
    res.render('person', { ID: req.params.id });
});
```

views/index.ejs

```
<body>
        <h1>Person: <%= ID %></h1>
</body>
```

Lesson 77 Querystring and Post Parameters

Data recieved on GET Request

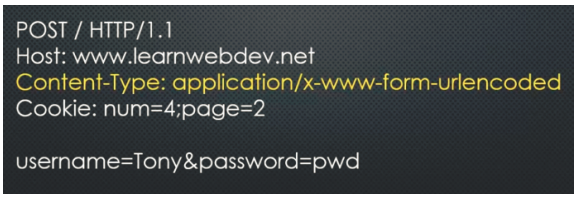


QueryParemeters are part of the URL.

So we can just retrieve it with the built in express urlEncoder

```
app.get('/person/:id', function(req, res) {
    res.render('person', { ID: req.params.id, Qstr: req.query.qstr });
});
```

Data recieved on POST Request (queryParameter)



QueryParameters are stored in the body of the POST request.

For POST request, we will need a **middlware** which will **get out the information** from the **POST Request's body**

required 3rd party module

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

inserting the middleware to extraxt the data

```
app.post('/person', urlencodedParser, function(req, res) {
    res.send('Thank you!');
    console.log(req.body.firstname);
    console.log(req.body.lastname);
```

```
});
```

Submitting a form, making a POST request.
index.ejs

```
        <form method="POST" action="/person">
            First name: <input type="text" id="firstname" name="firstname" /><br />
            Last name: <input type="text" id="lastname" name="lastname" /><br />
            <input type="submit" value="Submit" />
        </form>
```

Data recieved on POST Request (JSON)

Making a Post request with AJAX with JSON format
JSON.stringify() is needed since we are transferint object literals, and it needs to be converted to JSON

```
        <form method="POST" action="/person">
            First name: <input type="text" id="firstname" name="firstname" /><br />
            Last name: <input type="text" id="lastname" name="lastname" /><br />
            <input type="submit" value="Submit" />
        </form>
        <script>
            $.ajax({
                type: "POST",
                url: "/personjson",
                data: JSON.stringify({ firstname: 'Jane', lastname: 'Doe' }),
                dataType: 'json',
                contentType: 'application/json'
            });
        </script>
```

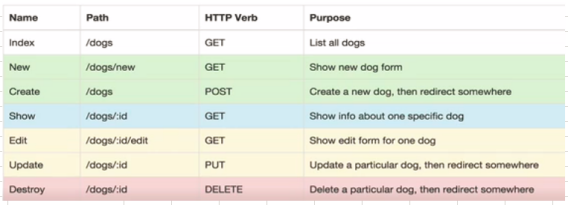
Importing the json extracting middleware from "**body-parser**"

```
var jsonParser = bodyParser.json();
```

Using the **jsonParser** as a middleware to extract information from the JSON body, and add it to `req.body` object

```
app.post('/personjson', jsonParser, function(req, res) {
    res.send('Thank you for the JSON data!');
    console.log(req.body.firstname);
    console.log(req.body.lastname);
});
```

Lesson 78 RestFul API Data



Name	Path	HTTP Verb	Purpose
Index	/dogs	GET	List all dogs
New	/dogs/new	GET	Show new dog form
Create	/dogs	POST	Create a new dog, then redirect somewhere
Show	/dogs/:id	GET	Show info about one specific dog
Edit	/dogs/:id/edit	GET	Show edit form for one dog
Update	/dogs/:id	PUT	Update a particular dog, then redirect somewhere
Destroy	/dogs/:id	DELETE	Delete a particular dog, then redirect somewhere

Lesson 79 Structure App Data

Express Generator let's us create a webserver really easliy.

```
npm install express-generator -g
```

Create folder

```
express my app

cd my app

npm install // to install the packages
```

You can delegate the routes in the following ways

app.js

```
var route = require("./routes/routes"),

app.use("/", routes);
```

routes/routes.js

```
var express     = require("express");

var router      = express.Router();
```

```
router.get("/", function(req, res) {
    res.render("landing");
});
```

or invoking a function passing the express object,
app.js

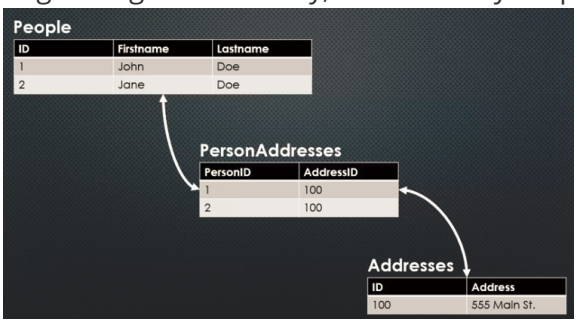
```
var apiController = require('./controllers/apiController');
htmlController(app);
```

htmlController.js

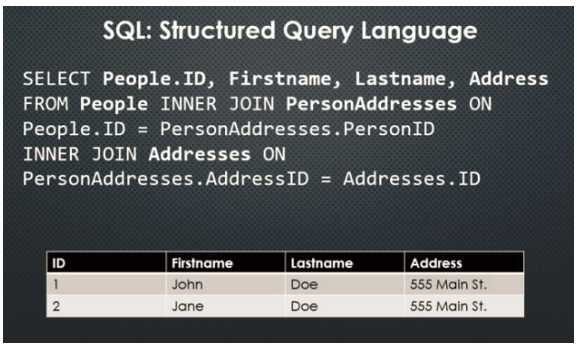
```
var bodyParser = require('body-parser');
var urlencodedParser = bodyParser.urlencoded({ extended: false });
module.exports = function(app) {

    app.get('/', function(req, res) {
        res.render('index');
    });
}
```

Lesson 80 Relational databases and SQl

Organizing data in a way, that it is easy to query



Make a query, for fast data retrival



THis is the javascript way to dea with data

Lesson 81 NodeJs + MySQL

1. Install MySQL program, create tables
2. Install npm mysql package
3. Create connection
4. create a query

2.

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

3.

```
//Confiquring the mySql data connection
    var con = mysql.createConnection({
        host: "localhost",
        user: "test",
        password: "test",
        database: "addressbook"
    });
```

4.

```
//Making the SQL query and returning the response
    con.query('SELECT People.ID, Firstname, Lastname, Address FROM People INNER JOIN PersonAddresses ON People.ID = PersonAddresses.PersonID INNER JOIN Addresses ON PersonAddresses.AddressID = Addresses.ID',
        function(err, rows) {
            if(err) throw err;
            console.log(rows[0].Firstname);
        }
    );
```

Lesson 82 No SQL Database

Data can be stored in a really flexible way.

Hardware space is sacrified for it flexibility.

Lesson 83 Mongo DB

See example program in lesson 83 folder

Lesson 85 MEAN Stack

Lesson 86 Managing Client

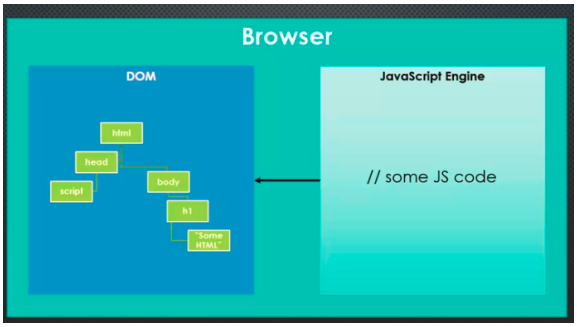
DOM: "Document Object Model"

Browser uses the DOM to render the webPage, and the browser add accessibility to javascript to change the DOM.

When it is changed, the page will be rerendered.

Browser, is actually a program, which sits on the client's device.

HTML structure in a tree structure so it sits well with the DOM.



Lesson 87 Managing Client Part(2)

You can add the **AngularJS** to the index.ejs file, to the script tag.

```
<html>
    <head>
        <title>The MEAN stack</title>
        <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.5/angular.js">
</script>
    </head>
    <body>
    </body>
</html>
```

Lesson 88 Managing Client Part(2)

1. **TestApp** is the identifier, which will let angularjs know what to control.
2. **MainController as vm** is the reference to the controller, **vm** will be the reference to that scope.
3. **ng-repeat** is used to display a set of names.

index.ejs

```
<html ng-app="TestApp">
    <head>
        <title>The MEAN stack</title>
        <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.5/angular.js">
</script>
    </head>
    <body ng-controller="MainController as vm">

        <input type="text" ng-model="vm.message" />
        <br />
        {{ vm.message }}
        <br />
        <ul>
            <li ng-repeat="person in vm.people">
                {{ person.name }}
            </li>
        </ul>

        <script src="assets/js/app.js"></script>
    </body>
</html>
```

app.js

```
angular.module('TestApp', []);

angular.module('TestApp')
    .controller('MainController', ctrlFunc);

function ctrlFunc() {
    this.message = "Hello";
```

```
    this.people = [
        {
            name: 'John Doe'
        },
        {
            name: 'Jane Doe'
        },
        {
            name: 'Jim Doe'
        }
    ]
}
```

Lesson 92 inital Setup

We just create the basic, simple express app.

Lesson 93 Setup MongoDB Mongoose

1. npm install them, include to the app.js
2. create seperate module, for the configuration to store as seperate object (safe, debuggable version)
3. Create a seperate model

******1*****
app.js

```
var mongoose = require('mongoose');                                    //In order
to work with mongoDB
var config = require('./config');


mongoose.connect(config.getDbConnectionString());                      //Good
Practice, to store them in a seperated Modul
```

****2***
config.json

```
{
    "uname": "YOUR_USERNAME_HERE",
    "pwd": "YOUR_PASSWORD_HERE"
}
```

****3****
index.js

```
var configValues = require('./config');            // A simple object, with username,
password properties, which can be used
module.exports = {

    getDbConnectionString: function() {            //Only this function will be
exported
        return 'YOUR_MONGO_URL';
    }

}

var mongoose = require('mongoose');                          //In order to use
schema
var Schema = mongoose.Schema;                                //Creating schemas
var todoSchema = new Schema({                                //Basic Schema object
    username: String,
    todo: String,
    isDone: Boolean,
    hasAttachment: Boolean
});

var Todos = mongoose.model('Todos', todoSchema);             //Creating a model from
the schema
```

```
module.exports = Todos;                                    //Exporting the created
model
```

Lesson 94 Adding Seed Data

1. Calling the imported, module to setup database
2. Creating a route, + array of todos + adding them to the database with mongoose.

app.js

```
setupController(app);
```

//1.Importing mongoose schema,
//2. exporting a function, which is basically a route, in which
// 2. an array is created inside the routing,
//3. array is added to the Todos model

setupController.js

```
var Todos = require('../models/todoModel');


module.exports = function(app) {


   app.get('/api/setupTodos', function(req, res) {


       // seed database
       var starterTodos = [
           {
               username: 'test',
               todo: 'Buy milk',
               isDone: false,
               hasAttachment: false
           },
           {
               username: 'test',
```

```
                todo: 'Feed dog',
                isDone: false,
                hasAttachment: false
            },
            {
                username: 'test',
                todo: 'Learn Node',
                isDone: false,
                hasAttachment: false
            }
        ];
        Todos.create(starterTodos, function(err, results) {
            res.send(results);
        });
    });

}
```

Lesson 94 Creating our API

Importing the Api Controller into the main **app.js**

```
apiController(app);
```

apiController..js

```
var Todos = require('../models/todoModel');                    //Using the object model
var bodyParser = require('body-parser');                       //Encoding recieved
information


module.exports = function(app) {
```

```
    app.use(bodyParser.json());                         //To exctract data from http
request body
    app.use(bodyParser.urlencoded({ extended: true }));   // To exctact data from the
encoded URL of GET request

//It is to fetch the whole todo  list of the specific user
    app.get('/api/todos/:uname', function(req, res) {

        Todos.find({ username: req.params.uname }, function(err, todos) {
            if (err) throw err;

            res.send(todos);
        });

    });
// It is to fetch a specific todo, with the to do id,
    app.get('/api/todo/:id', function(req, res) {

       Todos.findById({ _id: req.params.id }, function(err, todo) {
           if (err) throw err;

           res.send(todo);
       });

    });
// it is to post a tod id, find in the database and post if it can be found
    app.post('/api/todo', function(req, res) {

        if (req.body.id) {
            Todos.findByIdAndUpdate(req.body.id, { todo: req.body.todo, isDone:
req.body.isDone, hasAttachment: req.body.hasAttachment }, function(err, todo) {
                if (err) throw err;
```

```
                res.send('Success');
            });
        }

        else {
//If not, you can just create a new one with the Todos Object Model
            var newTodo = Todos({
                username: 'test',
                todo: req.body.todo,
                isDone: req.body.isDone,
                hasAttachment: req.body.hasAttachment
            });
            newTodo.save(function(err) {
                if (err) throw err;
                res.send('Success');
            });

        }

    });
//Delete with when it can find it.
    app.delete('/api/todo', function(req, res) {

        Todos.findByIdAndRemove(req.body.id, function(err) {
            if (err) throw err;
            res.send('Success');
        })

    });
}
```

Lesson 96 Testing Our API

Postman

Lesson 97 Adding Front End

Download seed project from
https://angular.io/

We are keeping the angular2 dependecies " our nodejs seperately.

http://reactivex.io/

Basically, we are creating an index html function, and our whol express app is just serving some API data to our angular app.