Node.js

Complete course

Douglas Collioni
@dcollioni

agenda

1. overview
2. npm
3. express
4. mongodb
5. tests
6. deploy

1. overview

1. overview / intro

What’s Node.js?

- Node.js is an open source server framework
- Node.js is free
- Node.js runs on various platforms (Windows, Linux, Unix, Mac OS X, etc.)
- Node.js uses JavaScript on the server

1. overview / intro

Why Node.js?

- **Node.js uses asynchronous programming**

  Scenario: server must open a file and return the content to the client

1. overview / intro

Why Node.js?

Scenario: server must open a file and return the content to the client

PHP, Java, C# execution:

1. Sends the task to the computer's file system.
2. Waits while the file system opens and reads the file.
3. Returns the content to the client.
4. Ready to handle the next request.

1. overview / intro

Why Node.js?

Scenario: server must open a file and return the content to the client

Node.js execution:

1. Sends the task to the computer's file system.
2. Ready to handle the next request.
3. When the file system has opened and read the file, the server returns the content to the client.

1. overview / intro

What Can Node.js Do?

- Node.js can generate dynamic page content
- Node.js can create, open, read, write, delete, and close files on the server
- Node.js can collect form data
- Node.js can add, delete, modify data in your database

1. overview / intro

What is a Node.js File?

- Node.js files contain tasks that will be executed on certain events
- A typical event is someone trying to access a port on the server
- Node.js files must be initiated on the server before having any effect
- Node.js files have extension ".js"

1. overview / installation

- The official Node.js website has installation instructions for Node.js:

  https://nodejs.org

1. overview / running

- Node.js files must be initiated in a"Command Line Interface" program:
    - Command Prompt (cmd)
    - Git Bash
    - PowerShell
    - Terminal

1. overview / running

- Create a folder to your code:

  ```
  mkdir /c/code/nodejs
  ```

- Navigate to the folder:

  ```
  cd /c/code/nodejs
  ```

- Initiate the Visual Code in this folder:

  ```
  code .
  ```

1. overview / running

- Create a new file named “hello.js”

  ```
  touch hello.js
  ```

- Write some JavaScript code into it:

  ```
  console.log('Hello, World!')
  ```

- Run the code using node:

  ```
  node hello.js
  ```

1. overview / running server

- Create a new file named “server.js”
- Write some JavaScript code into it:

```
const http = require('http')
http.createServer(function ( req, res ) {
   res.writeHead( 200, { 'Content-Type': 'text/plain' } )
   res.end('Hello World!')
}).listen(8080)
```

- Run the server:

```
node server.js
```

1. overview / modules

What is a Module in Node.js?

- Consider modules to be the same as JavaScript libraries.
- A set of functions you want to include in your application.

Built-in Modules

- Node.js has a set of built-in modules which you can use without any further installation. Look at our Built-in Modules Reference for a complete list of modules.

1. overview / modules

Include Modules

- To include a module, use the `require()` function with the name of the module:

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

- Now your application has access to the HTTP module:

```
http.createServer(function ( req, res ) { // ...
```

1. overview / modules

Create your own modules

- You can create your own modules, and easily include them in your applications.
- Create a module that returns the current date and time:

```
exports.myDateTime = () => {
 return Date()
}
```

- Save the code above in a file called "myDateTime.js"

1. overview / modules

Include your own modules

- Now you can include and use the module in any of your Node.js files.

```
const dt = require('./myDateTime')
console.log('myDateTime:', dt.myDateTime())
```

- Notice that we use `./` to locate the module, that means that the module is located in the same folder as the Node.js file.
- Save the code above in a file called "demo_module.js" and run it:

```
node demo_modules.js
```

1. overview / modules

Exercício

- Crie um módulo separado chamado “meuNome.js” com uma função que retorne o seu nome. Depois utilize esse módulo no arquivo “demo_modules.js” para mostrar seu nome do console.

1. overview / http

- Node.js has a built-in module called HTTP, which allows Node.js to transfer data over the Hyper Text Transfer Protocol (HTTP).

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


// create a server object:
http.createServer(function (req, res) {
 res.write('Hello World!') // write a response to the client
 res.end() // end the response
}).listen(8080) // the server object listens on port 8080
```

1. overview / http

- If the response from the HTTP server is supposed to be displayed as HTML, you should include an HTTP header with the correct content type:

```
const http = require('http');
http.createServer(function (req, res) {
 res.writeHead(200, { 'Content-Type': 'text/html' })
 res.write('Hello World!')
 res.end()
}).listen(8080)
```

1. overview / http

- The function passed into the `http.createServer()` has a `req` argument that represents the request from the client.
- This object has a property called `url` which holds the part of the url that comes after the domain name:

```
const http = require('http');
http.createServer(function (req, res) {
 res.writeHead(200, {'Content-Type': 'text/html'})
 res.write(req.url)
 res.end()
}).listen(8080)
```

1. overview / http

- There are built-in modules to easily split the query string into readable parts, such as the URL module:

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

http.createServer(function (req, res) {
 res.writeHead(200, {'Content-Type': 'text/html'})
 const q = url.parse(req.url, true).query
 const txt = q.year + " " + q.month
 res.end(txt)
}).listen(8080)
```

1. overview / fs

- The Node.js file system (fs) module allow you to work with the file system on your computer.

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

- Common use for the File System module:
  - Read files
  - Create files
  - Update files
  - Delete files
  - Rename files

1. overview / fs

- The `fs.readFile()` method is used to read files on your computer.

```
const http = require('http')
const fs = require('fs')
http.createServer((req, res) => {
 fs.readFile('demo.html', (err, data) => {
   res.writeHead(200, { 'Content-Type': 'text/html' })
   res.write(data)
   res.end()
 })
}).listen(8080)
```

1. overview / fs

Exercício

- Crie um servidor HTTP que leia o `pathname` da url e leia um arquivo com esse nome para retornar ao cliente. Exemplo:

  localhost:8080/demo.html → devolve o arquivo demo.html

  localhost:8080/home.html → devolve o arquivo home.html

2. npm

2. npm

What is NPM?

- NPM is a package manager for Node.js packages, or modules if you like.
- npmjs.com hosts thousands of free packages to download and use.
- The NPM program is installed on your computer when you install Node.js
- NPM is already ready to run on your computer

2. npm

What is a package?

- A package in Node.js contains all the files you need for a module.
- Modules are JavaScript libraries you can include in your project. Examples:
    - http
    - fs
    - url
    - express
    - mongodb
    - mocha

2. npm

Download a package

- Downloading a package is very easy.
- Open the command line interface and tell NPM to download the package.
- I want to download a package called "upper-case":

```
npm install upper-case
```

- NPM creates a folder named "**node_modules**", where the package will be placed. All packages you install in the future will be placed in this folder.

2. npm

Using a package

- Once the package is installed, it is ready to use.
- Include the "upper-case" package the same way you include any other module:

```
var uc = require('upper-case')
console.log(uc("Hello, World!"))
```

2. npm

- You can initialize a new project using:

```
npm init
```

- This command will generate a `package.json` file for your project.
- There you can place some important information about your project.

```
mkdir demo
```

```
cd demo
```

```
npm init
```

2. npm

- Now you can save modules installed via npm:

```
npm install upper-case --save
```

- The modules installed using the option `--save` will be saved at your `package.json` file in the section `dependencies`.

```
"dependencies": {
   "upper-case": "^1.1.3"
}
```

2. npm

- You can also update and remove the previously installed modules:

```
npm update upper-case --save
```

```
npm remove upper-case --save
```

2. npm

- In the package.json file you can declare scripts that can be executed for your project. Example:

```
"scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "node index.js"
}
```

- You can call these scripts using npm:

```
npm run start
```

3. express

3. express

- Fast and minimalist web framework for Node.js

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

- With a myriad of HTTP utility methods and middleware at your disposal, creating a robust API is quick and easy.

  http://expressjs.com/

3. express

Hello world

```
mkdir express-demo

cd express-demo

npm init

npm install express --save

touch index.js
```

3. express

Hello world

```
const express = require('express')
const app = express()

app.get('/', (req, res) => res.send('Hello World!'))

app.listen(3000, () =>
    console.log('App listening on port 3000!'))
```

3. express

Express application generator

- Use the application generator tool, express-generator, to quickly create an application skeleton. http://expressjs.com/en/starter/generator.html

```
npm install express-generator -g
express --view=pug myapp
cd myapp
npm install
npm start
```

3. express / basic routing

- Routing refers to determining how an application responds to a client request to a particular endpoint, which is a URI (or path) and a specific HTTP request method (GET, POST, and so on).
- Each route can have one or more handler functions, which are executed when the route is matched.

```
app.METHOD(PATH, HANDLER)
```

3. express / basic routing

```
app.get('/', (req, res) => {
 res.send('Hello World!')
})
app.post('/', (req, res) => {
 res.send('Got a POST request')
})
app.put('/user', (req, res) => {
 res.send('Got a PUT request at /user')
})
app.delete('/user', (req, res) => {
 res.send('Got a DELETE request at /user')
})
```

3. express / static files

- To serve static files such as images, CSS files, and JavaScript files, use the `express.static` built-in middleware function in Express.
- For example, use the following code to serve images, CSS files, and JavaScript files in a directory named public:

```
app.use(express.static('public'))
```

- Now, you can load the files that are in the public directory:
  - http://localhost:3000/images/kitten.jpg
  - http://localhost:3000/css/style.css
  - http://localhost:3000/js/app.js

3. express / advanced routing

- There is a special routing method, `app.all()`, used to load middleware functions at a path for all HTTP request methods.

```
app.all('/secret', (req, res, next) => {
  console.log('Accessing the secret section ...')
  next() // pass control to the next handler
})
```

3. express / advanced routing

- Route parameters are named URL segments that are used to capture the values specified at their position in the URL. The captured values are populated in the `req.params` object, with the name of the route parameter specified in the path as their respective keys.

```
app.get('/users/:userId/books/:bookId', (req, res) => {
 res.send(req.params)
})
```

http://localhost:3000/users/10/books/20

3. express / advanced routing

- More than one callback function can handle a route (make sure you specify the next object). For example:

```
app.get('/example/b', (req, res, next) => {
  console.log('response will be sent by the next function')
  next()
}, (req, res) => {
  res.send('Hello from B!')
})
```

3. express / advanced routing

- Response methods:
  - `res.download()` Prompt a file to be downloaded.
  - `res.end()` End the response process.
  - `res.json()` Send a JSON response.
  - `res.jsonp()` Send a JSON response with JSONP support.
  - `res.redirect()` Redirect a request.
  - `res.render()` Render a view template.
  - `res.send()` Send a response of various types.
  - `res.sendFile()` Send a file as an octet stream.
  - `res.sendStatus()` Set the response status code and send its string as body.

3. express / advanced routing

- You can create chainable route handlers for a route path by using `app.route()`:

```
app.route('/book')
  .get((req, res) => { res.send('Get a random book') })
  .post((req, res) => { res.send('Add a book') })
  .put((req, res) => { res.send('Update the book') })
  .delete((req, res) => { res.send('Delete the book') })
```

3. express / advanced routing

- Use the `express.Router` class to create modular, mountable route handlers:

```
const express = require('express')
const router = express.Router()

router.get('/', (req, res) => {
 res.send('Birds home page')
})
router.get('/about', (req, res) => {
 res.send('About birds')
})
module.exports = router
```

3. express / advanced routing

- Then, load the router module in the app:

```
const birds = require('./birds')
// ...
app.use('/birds', birds)
```

http://localhost:3000/birds/

http://localhost:3000/birds/about

3. express / middleware

- Middleware functions are functions that have access to the request object (req), the response object (res), and the next function in the application’s request-response cycle.
- Middleware functions can perform the following tasks:
  - Execute any code.
  - Make changes to the request and the response objects.
  - End the request-response cycle.
  - Call the next middleware in the stack.

3. express / middleware

```
const requestTime = (req, res, next) => {
  req.requestTime = Date()
  next()
}

app.use(requestTime)

app.get('/', (req, res) => {
  res.send('Hello, World! ' + req.requestTime)
})
```

3. express / body-parser

- Parse incoming request bodies in a middleware before your handlers, available under the `req.body` property.

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

```
const bodyParser = require('body-parser')
app.use(bodyParser.json())
```

3. express / body-parser

- Adding a generic JSON parser as a top-level middleware, which will parse the bodies of all incoming requests:

```
app.post('/login', (req, res) => {
  const { user, password } = req.body
  if (user !== 'root' || password !== '1234') {
    res.sendStatus(403)
  } else {
    res.sendStatus(200)
  }
})
```

3. express

Exercício

- Crie um servidor HTTP com um middleware que valide os parâmetros user e password enviados via querystring e bloqueie o acesso à rota /admin caso eles não sejam válidos.

4. mongodb

4. mongodb

- The official MongoDB Node.js driver provides both callback-based and Promise-based interaction with MongoDB.
- Given that you have created your own project using `npm init` we install the mongodb driver and it's dependencies by executing:

```
npm install mongodb --save
```

4. mongodb / connection

```
const { MongoClient } = require('mongodb')
const url = 'mongodb://localhost:27017';
const dbName = 'myproject';

MongoClient.connect(url, (err, client) => {
  console.log("Connected successfully to server");
  const db = client.db(dbName);
  client.close();
});
```

4. mongodb / insert a document

```
const insert = async (db) => {
  const collection = db.collection('users')

  const { result } = await collection.insertOne({
    name: 'Pedro Machado',
    email: 'pedro.machado@gmail.com'
  })

  console.log(result);
}
```

4. mongodb / insert many documents

```
const insertMany = async (db) => {
  const collection = db.collection('countries');

  const { result } = await collection.insertMany([
    { name: 'Brasil', code: 'BR' },
    { name: 'Itália', code: 'IT' }
  ])

  console.log(result);
}
```

4. mongodb / find all

```
const findUsers = async (db) => {
  const collection = db.collection('users')

  const users = await collection.find({}).toArray()

  console.log(users)
}
```

4. mongodb / find documents with a query

```
const findCountries = async (db) => {
  const collection = db.collection('countries')

  const query = { code: 'BR' }
  const countries = await collection.find(query).toArray()

  console.log(countries)
}
```

4. mongodb / update a document

```
const updateCountry = async (db) => {
  const collection = db.collection('countries')

  const query = { code: 'IT' }
  const { result } = await collection.updateOne(query, {
    $set: {
      name: 'Italy'
    }
  })

  console.log(result)
}
```

4. mongodb / remove a document

```
const removeUser = async (db) => {
  const collection = db.collection('users')

  const query = { email: 'pedro.machado@gmail.com' }
  const { result } = await collection.deleteOne(query)

  console.log(result)
}
```

4. mongodb / map collections

```
const { MongoClient } = require('mongodb')
const url = 'mongodb://localhost:27017';
const dbName = 'myproject'

const db = async () => (await MongoClient.connect(url)).db(dbName)

const users = async () => (await db()).collection('users')
const countries = async () => (await db()).collection('countries')

module.exports = { users, countries }
```

4. mongodb / using collections

```
const { users } = require('./mongo')

const run = async () => {
  const usersCollection = await users()
  const usersDocs = await usersCollection.find({}).toArray()
  console.log(usersDocs)
}

run()
```

4. mongodb / mongoose

- Elegant mongodb object modeling for node.js
- Mongoose provides a straight-forward, schema-based solution to model your application data. It includes built-in type casting, validation, query building, business logic hooks and more, out of the box.

```
npm install mongoose --save
```

4. mongodb / mongoose connection

```
const mongoose = require('mongoose')
const url = 'mongodb://localhost:27017/myproject';
const connection = mongoose.connect(url);
```

4. mongodb / schema and model

```
const countrySchema = mongoose.Schema({
  name: String,
  code: String
});

const Country = mongoose.model('Country', countrySchema)
const france = new Country({ name: 'França', code: 'FR' })
```

4. mongodb / insert a document

```
const insert = async () => {
  const france = new Country({ name: 'França', code: 'FR' })
  const result = await france.save()
  console.log(result)
}

insert()
```

4. mongodb / find documents

```
const find = async () => {
  const countries = await Country.find({})
  console.log(countries)

  const brazil = await Country.findOne({ code: 'BR' })
  console.log(brazil)
}

find()
```

4. mongodb / update a document

```
const update = async () => {
  const brazil = await Country.findOne({ code: 'BR' })
  brazil.name = 'Brazil'
  await brazil.save()

  console.log(brazil)
}

update()
```

4. mongodb / remove a document

```
const remove = async () => {
  const italy = await Country.findOne({ code: 'IT' })
  const result = await italy.remove()
  console.log(result)
}

remove()
```

4. mongodb / schema validators

```
const breakfastSchema = mongoose.Schema({
  eggs: {
    type: Number, min: [6, 'Too few eggs'], max: 12
  },
  bacon: {
    type: Number, required: [true, 'Why no bacon?']
  },
  drink: {
    type: String, enum: ['Coffee', 'Tea'],
    required: function() { return this.bacon > 3; }
  }
});
```

4. mongodb / schema validators

```
const Breakfast = mongoose.model('Breakfast', breakfastSchema);

const run = async () => {
  const bf = new Breakfast({ eggs: 4, bacon: 2 })
  try {
    await bf.save()
  } catch (err) {
    console.log(err)
  }
  console.log(bf)
}
run()
```

4. mongodb / schema validators

Exercício

- Crie um servidor HTTP utilizando express com rotas para inserir, buscar, atualizar e remover países de um banco de dados MongoDB.
- Teste todas as rotas utilizando o Postman.
- Complemente esse exercício criando páginas HTML para acessar essas rotas e poder fazer as operações de CRUD nos países.

5. tests

5. tests

- Mocha is a feature-rich JavaScript test framework running on Node.js and in the browser, making asynchronous testing simple and fun.
- Mocha tests run serially, allowing for flexible and accurate reporting, while mapping uncaught exceptions to the correct test cases.

```
npm install --save-dev mocha
```

```
mkdir test
```

```
cd test
```

```
touch test.js
```

5. tests

```
const assert = require('assert')

describe('Array', () => {
 describe('#indexOf()', () => {
   it('should return -1 when the value is not present', () => {
     assert.equal([1,2,3].indexOf(4), -1)
   })
 })
})
```

5. tests

- To run the tests, we just run:

```
./node_modules/mocha/bin/mocha
```

```
Array
    #indexOf()
      √ should return -1 when the value is not present
1 passing (7ms)
```

5. tests

- Set up a test script in package.json:

```
"scripts": {
   "test": "mocha"
}
```

- To call mocha like in the script above, install mocha globally:

```
npm install mocha -g
```

5. tests

- Then you can run:

```
npm test
```

```
Array
    #indexOf()
      √ should return -1 when the value is not present
1 passing (7ms)
```

5. tests

```
before(async () => {
 await db.clear()
 await db.save([tobi, loki, jane])
})

describe('#findUsers()', () => {
 it('responds with matching records', async () => {
   const users = await db.find({ type: 'User' })
   users.should.have.length(3)
 })
})
```

5. tests

Exercício

- Crie um teste para garantir que a função que busca países no banco sempre retorne todos os países cadastrados.

6. deploy

6. deploy

- Deploy is the process to publish your application to be used in a production or tests environment. In other words, it is to make them available to the public.
- There are many tools to make our Node.js applications available. PM2 is one of them.
- PM2 is straightforward, it is offered as a simple and intuitive CLI, installable via NPM.

```
npm install pm2 -g
```

6. deploy / pm2

- To start running an application, run:

```
pm2 start index.js --name my-api
```

- Other useful commands:

```
pm2 list
pm2 logs
pm2 stop
pm2 restart
pm2 delete
```

references

- https://www.w3schools.com/nodejs/
- https://npmjs.com
- http://expressjs.com
- http://expressjs.com/en/resources/middleware/body-parser.html
- https://mongodb.github.io/node-mongodb-native/
- http://mongoosejs.com/
- https://mochajs.org/
- http://pm2.keymetrics.io/