ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

ΚΕΝΤΡΟ ΕΠΙΜΟΡΦΩΣΗΣ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗΣ

Node.js και Express

Μ. Καραμπάτσης

Node.js

- Το NodeJS δεν είναι γλώσσα προγραμματισμού ούτε framework.
- Είναι ένα περιβάλλον που μας επιτρέπει να γράψουμε JavaScript, παρέχοντας μας τη δυνατότητα αλληλεπίδρασης με τον "έξω κόσμο", πέρα από το πρόγραμμα περιήγησης.
- Δεν παρέχει δυνατότητες για την υλοποίηση μόνο Web API (setTimeout, ή event handlers όπως click, mouse over, scroll κα), παρέχει API για αλληλεπίδραση με στοιχεία όπως το σύστημα αρχείων, το HTTP και το λειτουργικό σύστημα.

Η χρήση του NodeJS προϋποθέτει την εγκαταστάση τους είτε μέσω του NodeJS.org (https://nodejs.org/en) είτε με NVM (https://github.com/coreybutler/nvm-windows)

Node.js

Έλεγχος εγκατάστασης: σε ένα terminal πληκτρολογούμε την εντολή node -v

```
>node -v
v20.11.1
```

Για να κάνουμε χρήση του Node περιβάλλοντος πληκτρολογούμε την εντολή node.

```
C:\Users\user>node
Welcome to Node.js v20.11.1.
Type ".help" for more information.
>
```

Εντολή **console.log**:

```
> console.log(1+1)
2
```

NodeJS και JavaScript αρχεία

Μπορούμε να τρέξουμε JavaScript αρχεία κάνοντας χρήση της εντολής **node**

- Δημιουργήστε αρχείο με όνομα app1.js
- Στο αρχείο προσθέστε τις παρακάτω εντολές:
- Ανοίγουμε ένα terminal (στο φάκελο που βρίσκεται το αρχείο app1.js) και πληκτρολογούμε την εντολή

```
// Αρχείο app1.js
const message = "Hello Word";
console.log(message);
```

```
C:\Users\user>node app1.js
> Hello Word
```

Node.js και βιβλιοθήκες

Με το Node.js μπορούμε να δημιουργήσουμε εφαρμογές κάνοντας χρήση βιβλιοθηκών.

- Υπάρχουν ενσωματωμένες βιβλιοθήκες, είτε άλλες που έχουν δημιουργηθεί από την κοινότητα.

Βιβλιοθήκες FS, OS και HTTP

- Δημιουργήστε αρχείο με όνομα app2.js
- Στο αρχείο προσθέστε τις παρακάτω εντολές:

```
// Αρχείο app2.js
const fs = require('fs');
const http = require('http');
const os = require('os');

// Επιστρέφει το λειτουργικό απο το συστήμα μας
osType = os.type();

// Δημιουργία string με HTML περιεχόμενο
htmlContent = `<html><h3>Hello, World! Your OS type is ${osType}</h3></html>`
```

Node.js και βιβλιοθήκες

Το module writeFile της fs βιβλιοθήκης χρησιμοποιείται για τη δημιουργία και την επεξεργασία αρχείων.

Κάνοντας χρήση ασύγχρονων κλήσεων με callback συναρτήσεις:

1. Θα δημιουργήσουμε index.html αρχείο με περιεχόμενα την μεταβλητή htmlContent.
2. Με χρήση callback συναρτήσεων θα δημιουργήσουμε αρχικά το άρχειο index.html.

```
const server = http.createServer((req, res) => {

  console.log("stage1");
  fs.writeFile('./index.html', htmlContent, (err) => {
    if (err) {
      console.log("Problem in writing file")
    } else {
      console.log("stage3");
      fs.readFile('index.html', (err, content) => {
        console.log("stage4");
        res.setHeader('Content-Type', 'text/html');
        res.end(content);
        if (err) {
          console.log(err)
        }
      });
    }
  })
});
```

Node.js και βιβλιοθήκες

3. Στη συνέχεια θα δημιουργήσουμε έναν web server με χρήση της μεθόδου createServer
4. Μετά τη δημιουργία του server θα διαβάσουμε το αρχείο index.html
5. Στο τέλος θα ξεκινήσουμε τον web server κάνοντας χρήση της μεθόδου listen της createServer.
6. Προκειμένου να έχουμε μια ενημέρωση ότι ο server ξεκίνησε, δημιουργούμε callback κλήση με ένα console.log
7. Στο τέλος από ένα terminal τρέχουμε την εντολή

```
C:\Users\user>node app2.js
> Listening on port 3000!
```

8. Ανοίγουμε έναν browser και πληκτρολογούμε http://localhost:3000/

Node.js και synchronous βιβλιοθήκες

Το module **writeFileSync** και **readFileSync** της fs βιβλιοθήκης χρησιμοποιούνται αντίστοιχα για τη δημιουργία/τροποποίηση αρχείων και για το διάβασμα αρχείων.

1. Με χρήση callback συναρτήσεων θα δημιουργήσουμε έναν web server με χρήση της μεθόδου createServer
2. Στη συνέχεια με την writeFileSync θα δημιουργήσουμε το index.html αρχείο και με την readFileSync θα διαβάσουμε τα περιεχόμενα του.

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

osType = os.type();

htmlContent = `<!DOCTYPE html>
  <html>
    <h3>
      Hello, World! Your OS type is ${osType}, new Index File
    </h3>
  </html>`

const server = http.createServer((req, res) => {
  res.setHeader('Content-Type', 'text/html');
  fs.writeFileSync('./index2-1.html', htmlContent);
  let readFile = fs.readFileSync('index2-1.html', 'utf8');
  if (readFile) {
    res.end(readFile);
  }
});

server.listen(3000, () => {
  console.log('Listening on port 3000!');
});
```

Δημιουργία εφαρμογής στο NodeJS

Δημιουργία νέας εφαρμογής

1. Δημιουργούμε φάκελο,
2. Από command line πληκτρολογούμε την εντολή `npm init`,
3. Απαντάμε στις σχετικές ερωτήσεις,
4. Δημιουργείται το αρχείο **package.json**

```
Press ^C at any time to quit.
package name: (usersapp)
version: (1.0.0)
git repository:
keywords:
license: (ISC)
```

```
{
  "name": "usersapp",
  "version": "1.0.0",
  "description": "this is a crud app for users",
  "main": "app.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "coding factory",
  "license": "ISC"
}
```

package.json

Το αρχείο **package.json** περιέχει σημαντικές πληροφορίες που έχουν σχέση με το project.

- Περιέχει **μεταδεδομένα σχετικά με το project** (όπως το όνομα και την περιγραφή του έργου),
- **Λειτουργικά μεταδεδομένα** όπως την έκδοση του πακέτου και μια λίστα εξαρτήσεων που απαιτούνται από την εφαρμογή.
- Χρησιμοποιείται από το npm CLI (ή yarn) προκειμένου να διαχειριστεί και να εγκαταστήσει αρχεία εξαρτήσεων καθώς και για να ξεκινήσει την εφαρμογή.

package.json - Metadata

Metadata στο package.json

- **name**: ορίζει το όνομα του πακέτου(lowercase letters, no spaces). Χρήσιμο όταν θέλουμε να δημοσιεύσουμε το πακέτο στο npm, σε αυτή τη περίπτωση θα πρέπει να είναι μοναδικό.
- **version**: Απαραίτητο όταν θέλουμε να δημοσιεύσουμε ένα package. Ακολουθεί το semantic versioning, (New product 1.0.0, Patch release 1.0.1, Minor release 1.1.0, Major release 2.0.0)
- **license**: ορίζει τον τύπο της άδειας χρήσης που ισχύει για τον κώδικα που περιγράφει το package.json (MIT, ISC, UNLICENSED)
- **author and contributors**: συγγραφείς του πακέτου
- **description**: χρησιμοποιείται από npm για τα δημοσιευμένα πακέτα, για να περιγράψει το πακέτο στα αποτελέσματα αναζήτησης στον ιστότοπο npmjs.com.

package.json - Metadata

Metadata στο package.json

- **keywords**: είναι μια σειρά από συμβολοσειρές και όπως το description χρησιμοποιείται από το npm στην εύρεση πακέτων όταν κάποιος τα αναζητά.
- **main**: ορίζει το αρχείο που χρησιμοποιείται για την έναρξη της εφαρμογής.
- **scripts**: είναι ένα object με τα κλειδιά του να είναι scripts που μπορούμε να εκτελέσουμε με την εντολή εκτέλεσης `npm <scriptName>` και η τιμή είναι η πραγματική εντολή που εκτελείται.

```
"scripts": {
        "start": "node index.js",
        "dev": "nodemon"
}
```

nodemon: επιτρέπει την αυτόματη επανεκκίνηση της εφαρμογής όταν εντοπίζονται αλλαγές στα αρχεία της εφαρμογής (https://www.npmjs.com/package/nodemon)

package.json - Metadata

Metadata στο package.json

- **repository**: είναι ένα object που ορίζει το url στο οποίο ο πηγαίος κώδικας βρίσκεται. Επίσης ορίζει το version control system που χρησιμοποιείται όπως, GitHub, Bitbucket κλπ.

```
"repository": {
        "type": "git",
        "url": "https://github.com/osiolabs/example.git"
}

"dependencies": {
        "express": "^4.16.4",
}
```

- **dependencies**: ένα από τα πιο σημαντικά πεδία στο package.json, εμφανίζει όλα τα packages που χρησιμοποιεί η εφαρμογή. Όταν ένα package εγκαθίσταται με `npm install` τοποθετείται στο φάκελο node_modules και προστίθεται μια καταχώρηση στο πεδίο dependencies με key το όνομα του πακέτου και value την έκδοση

  `npm install --production` : εγκαθιστά μόνο αυτά που είναι στο dependencies

package.json - Metadata

Metadata στο package.json

- **devDependencies**: Παρόμοιο με το πεδίο dependencies, αλλά για πακέτα που χρειάζονται μόνο κατά το development και όχι στο production

  `npm install --save-dev <package>`

```
"devDependencies": {
        "nodemon": "^1.18.11"
}
```

Περισότερα στοιχεία για το package.json στην ιστοσελίδα: https://docs.npmjs.com/cli/v9/configuring-npm/package-json

Τα dependencies διαχειρίζονται με τις εντολές
npm install, **npm uninstall**, **npm update**

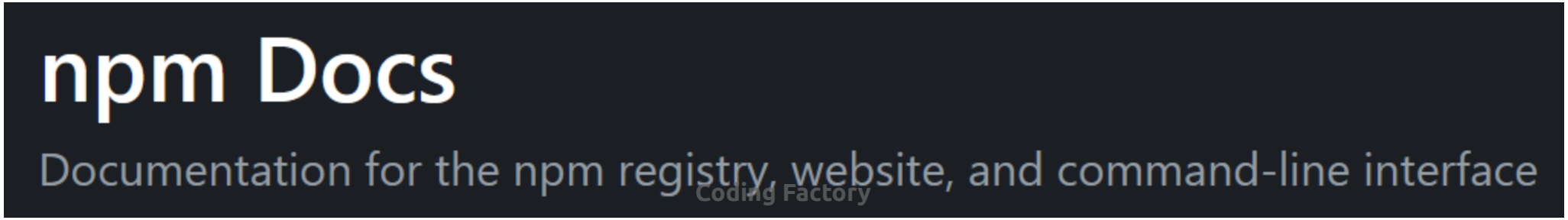
package-lock.json: Περιέχει τα packages που έχουν εγκατασταθεί με τις ακριβής εκδόσεις αυτών, έτσι ώστε οι επόμενες εγκαταστάσεις να μπορούν να δημιουργήσουν πανομοιότυπες εγκαταστάσεις

npm (node package manager)

Το npm είναι το μεγαλύτερο μητρώο λογισμικού στον κόσμο. (https://docs.npmjs.com/)

Το npm αποτελείται από τρία διακριτά στοιχεία:

- το website: για να αναζήτηση πακέτων (https://www.npmjs.com/)
- το Command Line Interface (CLI): το CLI τρέχει από command line και μέσω αυτού οι developers αλληλεπιδρούν με το npm.
- την registry: είναι μια μεγάλη δημόσια βάση δεδομένων λογισμικού JavaScript και των μεταπληροφοριών που το περιβάλλουν.



Φάκελος node_modules

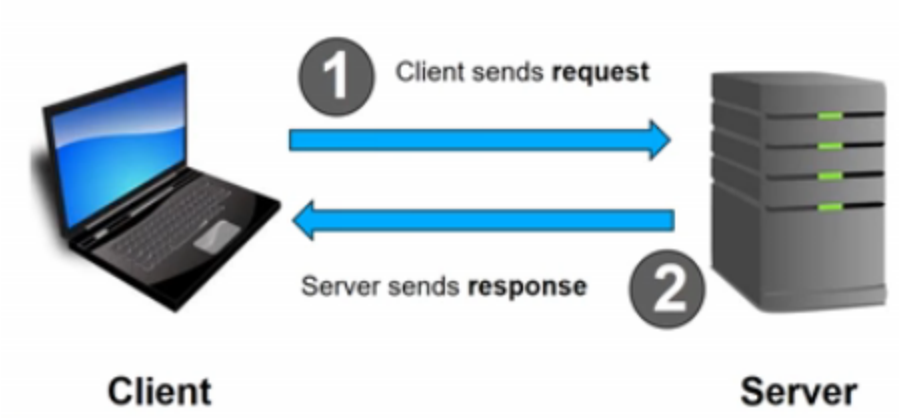
Ο φάκελος **node_modules** χρησιμοποιείται για την αποθήκευση όλων των πακέτων που εγκαθίστανται στον υπολογιστή μας μετά την εκτέλεση της εντολής

```
npm install <package_name>
```

Ο φάκελος επίσης δημιουργείται όταν σε μια Node.js εφαρμογή τρέξουμε `npm install` και εγκατασταθούν τα πακέτα που βρίσκονται στα πεδία dependencies και devdependencies του package.json αρχείου.

ΠΡΟΣΟΧΗ: στην περίπτωση που ανεβάσουμε την εφαρμογή μας στο gitHub ο φάκελος node_modules δεν χρειάζεται να ανέβει. Η πληροφόρια αυτή πρέπει να μπει στο .gitignore αρχείο.

Ποια είναι η δουλειά του web server;



- Ακούει και δέχεται εισερχόμενες αιτήσεις (request) HTTP.
- Αναλύει το αίτημα και το επεξεργάζεται.
- Στέλνει πίσω μια σωστά διαμορφωμένη HTTP απόκριση (response).

HTTP requests

GET: χρησιμοποιείται για την ανάγνωση/ανάκτηση δεδομένων από έναν web server. Το GET επιστρέφει έναν κωδικό κατάστασης HTTP 200 (OK) εάν τα δεδομένα ανακτηθούν με επιτυχία από τον web server.

Μέθοδος GET με query string parameters

- Ζεύγη κλειδιών/τιμών που αποτελούν μέρος της διεύθυνσης URL,

```
https://www.example.com/index.html?name1=value1&name2=value2
```

Μέθοδος GET με query path parameters

- Οι τιμές είναι μέρος του url path

```
https://www.example.com/index.html/value1/value2
```

HTTP requests

POST: χρησιμοποιείται για την αποστολή δεδομένων (αρχείο, δεδομένα φόρμας κ.λπ.) στον web server. Με την επιτυχή δημιουργία, επιστρέφει έναν κωδικό κατάστασης HTTP 201

PUT: χρησιμοποιείται για την τροποποίηση των δεδομένων στον web server. Αντικαθιστά ολόκληρο το περιεχόμενο σε μια συγκεκριμένη τοποθεσία με δεδομένα που διαβιβάζονται στο body του request. Εάν δεν υπάρχουν πόροι που να ταιριάζουν με το αίτημα, θα δημιουργήσει έναν.

PATCH: είναι παρόμοιο με το αίτημα PUT, αλλά η μόνη διαφορά είναι ότι τροποποιεί ένα μέρος των δεδομένων. Θα αντικαταστήσει μόνο το περιεχόμενο που θέλετε να ενημερώσετε

DELETE: χρησιμοποιείται για τη διαγραφή των δεδομένων στον web server.

Express

Το **Express** είναι framework του Node.js και παρέχει ένα σύνολο χαρακτηριστικών/ δυνατοτήτων για την υλοποιήση web και mobile εφαρμογών.

Με το Express μπορούμε να οργανώσουμε μια εφαρμογή με πολλούς τρόπους:

1. Να ορίσουμε **ξεχωριστές ενότητες** που έχουν **διαφορετικές ευθύνες**.
2. Να διαχειριστούμε **διαφορετικές κλήσεις** (requests) μέσω των **routes** και των **routers**.
3. Να χωρίσουμε κάθε βήμα της επεξεργασίας των κλήσεων σε **ενδιάμεσες συναρτήσεις** (middleware functions).

Χρήση του Express στην εφαρμογή μας

- Για να χρησιμοποιήσουμε το express στην εφαρμογή μας από command line τρέχουμε την εντολή: (https://www.npmjs.com/package/express),(https://expressjs.com/)

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

- Η εντολή θα κατεβάσει το πακέτο Express και θα το εγκαταστήσει (δείτε το φακελο node_modules)
- Επίσης θα τροποποιήσει το αρχείο package.json

```
"dependencies": {
        "express": "^4.18.2"
}
```

Express - Hello World

- Στο φάκελο που βρίσκεται η εφαρμογή μας δημιουργούμε ένα αρχείο **app.js**.
- Στο αρχείο app.js πληκτρολογήστε το παρακάτω κώδικα:

```
const express = require('express')
const app = express()
const port = 3000

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

app.listen(port, () => {
  console.log(`Example app listening on port ${port}`)
})
```

- Για να τρέξει το πρόγραμμα από command line

  `node app.js`

- Στο command line εμφανίζεται:

  `Example app listening on port 3000`

- Ανοίξτε σε ένα web browser τη σελίδα

  `http://localhost:3000`

Express - Hello World

app.get(path, callback [, callback ...])

Routes HTTP GET requests σε συγκεκριμένο μονοπάτι (στην περίπτωση μας στο "/" δηλαδή στο root) και με συγκεκριμένη callback function.

(req, res)

Το object **req** αναπαριστά το HTTP request και έχει σαν properties το request query string, τα parameters, το HTTP headers κλπ.

Το object **res** αναπαριστά το HTTP response που η Express εφαρμογή στέλνει όταν δέχεται ένα HTTP request.

app.listen([port[, host[, backlog]]][, callback])

Δημιουργεί έναν http server, δεσμεύοντας και ακούγοντας στην πόρτα και στον host που του δηλώνουμε

Express - Hello World

res.send([body])

Επιστρέφει το HTTP response.

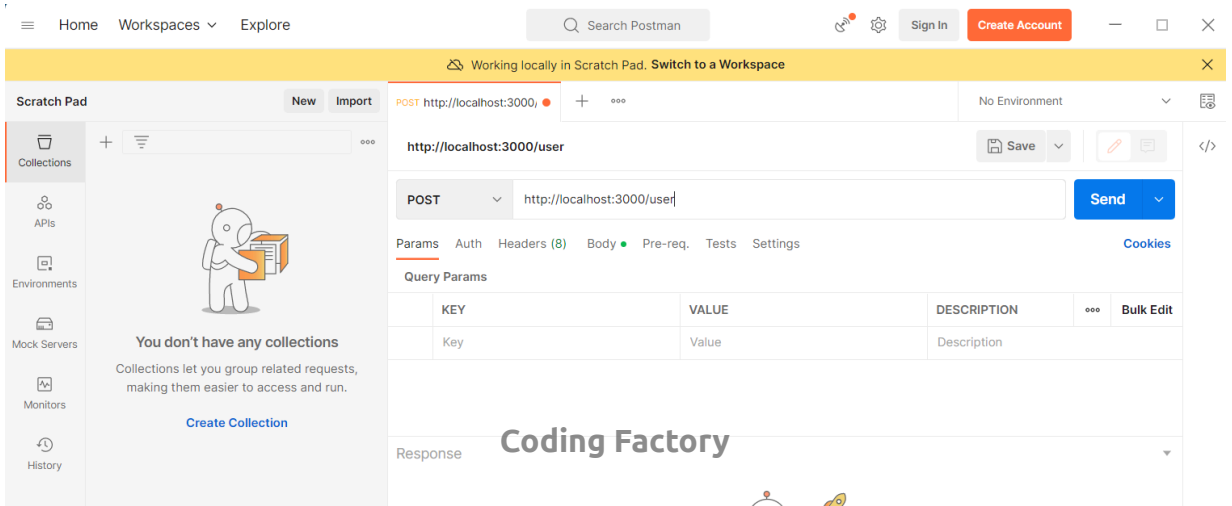
Η παράμετρος body μπορεί να είναι: String, Object, Boolean, ή Array.

```
res.send({ some: 'json' })
res.send('<p>some html</p>')
res.status(404).send('Sorry, we cannot find that!')
res.status(500).send({ error: 'something blew up' })
```

Express - Hello World

Για να ελέγξουμε μια HTTP κλήση έχουμε δύο τρόπους:

1. Σε περίπτωση get κλήσεων, ανοίγουμε browser και πληκτρολογούμε την διεύθυνση `http://localhost:3000/`
2. Για όλους τους τύπους κλήσεων, χρησιμοποιούμε το λογισμικό Postman (https://www.postman.com/). Επιτρέπει να κάνουμε HTTP κλήσεις προς οποιοδήποτε API στέλνοντας οποιοδήποτε τύπο παραμέτρων και λαμβάνοντας τα σχετικά response



Example 2 - Πολλαπλά Routings

```
const express = require('express');
const app = express();
const port = 3000;

app.get('/about', (req, res) => {
        console.log('about page');
        res.send('This is the about page');
});

app.get('/login', (req, res) => {
        console.log('login page');
        res.send('This is the login page');
});
app.listen(port, () => {
  console.log(`Example app listening on port ${port}`)
})
```

Example 3 - GET Query parameters

```
const express = require('express');
const app = express();
const port = 3000;

app.get('/user', (req, res) => {
  // ?name=Bob&surname=Dylan&age=81
  let query = req.query;
  console.log(query);

  let name = query.name;
  let surname = query.surname;
  let age = query.age;

  let length = Object.keys(query).length;
  console.log("Lenght: ", length);

  res.send('Name: ' + name + ' Surname: ' + surname + ' Age: ' + age);
});

app.listen(port, () => {
  console.log(`Example app listening on port ${port}`)
})
```

Προκειμένου να διαβάσουμε τα query parameters χρησιμοποούμε την εντολή:

let name = **req.query**.Ονομα_Πεδίου;

Example 4 - GET Query path parameters

```
const express = require('express');
const app = express();
const port = 3000;

app.get('/user/:name/:surname/:age', (req, res) => {
  let params = req.params;
  console.log(params);

  let name = params.name;
  let surname = params.surname;
  let age = params.age;

  let length = Object.keys(params).length;
  console.log("Lenght: ", length);

  res.send('Name: ' + name + ' Surname: ' + surname + ' Age: ' + age);
});

app.listen(port, () => {
  console.log(`Example app listening on port ${port}`)
})
```

Προκειμένου να διαβάσουμε τα query path parameters χρησιμοποούμε την εντολή:

let name = **req.params**.Ονομα_Πεδίου;

Example 5 - Post method

```
const express = require('express');
const app = express();
const port = 3000;
// const bodyParser = require('body-parser');
// app.use(bodyParser.json())
// app.use(bodyParser.urlencoded({ extended: false }))
app.use(express.json())
app.use(express.urlencoded({ extended: false }))


app.use('/user', express.static('files'));

app.post('/user', (req, res) => {
  console.log(req.body);

  let firstname = req.body.firstname;
  let lastname = req.body.lastname;
  let email = req.body.email;
  let sex = req.body.sex;

  res.send('Name: ' + firstname + ' Surname: ' + lastname + ' Email: ' + email + ' Sex:' + sex);
  console.log(JSON.stringify(req.headers));
  console.log("Content-Type>>",res.get('Content-type'));
});

app.post('/userForm', (req, res) => {
  console.log(req.body);

  let firstname = req.body.firstname;
  let lastname = req.body.lastname;
  let email = req.body.email;
  let sex = req.body.sex;

  res.send('Name: ' + firstname + ' Surname: ' + lastname + ' Email: ' + email + ' Sex:' + sex);
  console.log(JSON.stringify(req.headers));
});

app.listen(port, () => {
  console.log(`Example app listening on port ${port}`)
})
```

1. Προκειμένου να διαβάσουμε τα post data χρησιμοποούμε την εντολή:
   let firstname = **req.body**.Ονομα_Πεδίου;
2. Package **body-parser**: Κάνει parse τα incoming request bodies προκειμένου να διαβάσει τα δεδομένα που υπάρχουν.
3. **app.use(bodyParser.json()) ή app.use(express.json())**: Επιστρέφει middleware function που κάνει parse μόνο json και εξετάζει requests που το Content-Type header είναι json.

Example 5 - Post method

4. **app.use(bodyParser.urlencoded({ extended: false })) ή app.use(express.urlencoded({ extended: false }))**:
    - Εξετάζει στοιχεία που η φόρμα είναι **application/x-www-form-urlencoded**. Είναι ο default τύπος φόρμας στην HTML.

    Άλλοι τύποι είναι:

    - multipart/form-data: για ανέβασμα αρχείων
    - text/plain: νέος τύπος που εμφανίστηκε στην HTML5, στέλνει στοιχεία χωρίς κάποιο encoding
5. **extended: false**: ορίζει ότι το parsing θα γίνει με την querystring library (false) ή την qs library (true). Η δεύτερη επιλογή έχει γίνει deprecated.

Example 5 - Post method

4. **app.use([path,] callback [, callback...])**

Καλεί μια middleware function οι συναρτήσεις στο συγκεκριμένο path.

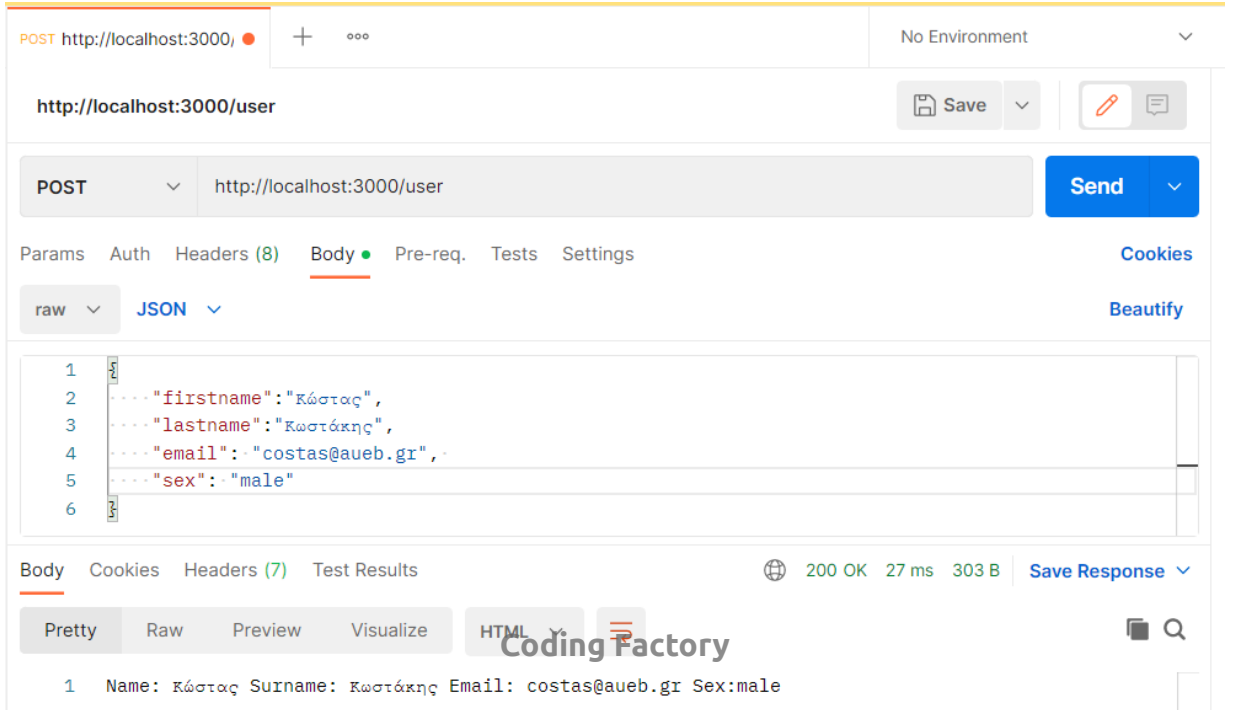
Η **next()** επιτρέπει να περάσει ο έλεγχος στην επόμενη συναρτηση που ικανοποιεί το path

```
app.use('/user', function (req, res, next) {
  console.log('Time: %d', Date.now());
  next();
})

app.post('/user', (req, res) => {
        console.log(req.body);

        let firstname = req.body.firstname;
        let lastname = req.body.lastname;

        console.log(firstname, lastname);
});
```

Example 5 - Post method

Για να ελέγξουμε μια HTTP κλήση με post, put, patch, delete:

- χρησιμοποιούμε το λογισμικό Postman (https://www.postman.com/). Επιτρέπει να κάνουμε HTTP κλήσεις προς οποιοδήποτε API στέλνοντας οποιοδήποτε τύπο παραμέτρων και λαμβάνοντας τα σχετικά response



Middleware functions

- **Middleware** συνάρτηση καλούμε μια συνάρτηση που χρησιμοποιείται στο χειρισμό των HTTP κλήσεων.
- Χρησιμοποιείται ως ο **"ενδιάμεσος"** στη διαδικασία της αποδοχής ενός **HTTP request** και της αποστολής ενός **HTTP response**.
- Σε ένα **HTTP request** μπορούν να χρησιμποποιηθούν πολλά **middleware**.
- Μια **middleware** συνάρτηση μπορεί να περίεχει όσο κώδικα Javascript επιθυμούμε με οποιαδήποτε λειτουργικότητα.
- Οι **middleware** συνάρτησεις παίρνουν τρεις παραμέτρους **req**, **res** και **next**.
- Η συνάρτηση **next()** πρέπει να καλείται στο τέλος κάθε middeware συνάρτησης έτσι ώστε να καλείται η επόμενη **middleware** συνάρτηση ή το τελικό response.

Example 6 - Middleware functions

```
const express = require('express');
const app = express();
const port = 3000;

const logger = (req, res, next) => {

  let url = req.baseUrl;
  let time = new Date();
  console.log('Received request for ' + url + ' at' + time);

  next();
};

app.get('/public', logger, (req, res) => {
  console.log('public page');
  res.send('This is the public');
});

app.listen(port, () => {
  console.log(`Example app listening on port ${port}`)
})
```

req.baseUrl: το path από το οποίο έγινε η κλήση

Για μια GET κλήση

```
'http://www.example.com/admin/new?sort=desc'
req.originalUrl // '/admin/new?sort=desc'
req.baseUrl // '/admin'
req.path // '/new'
```

Middleware functions

Παράδειγμα με πολλά middleware

```
const logger1 = (req, res, next) => {...};
const logger2 = (req, res, next) => {...};
const logger3 = (req, res, next) => {...};

app.use('/public', logger1, logger2 (req, res) => {
        console.log('public page');
        res.send('This is the public');
});

app.use('/about', logger1, logger3 (req, res) => {
        console.log('public page');
        res.send('This is the public');
});
```

EJS

Για να διαχωρίσουμε τη JavaScript από την HTML χρησιμοποιούμε εφαρμογές (πακέτα) που είναι γνωστά ως **view engine **, όπως το EJS (https://ejs.co/)

- Είναι **templating** γλώσσες που μας επιτρέπουν να δημιουργούμε HTML με χρήση JavaScript.
- Παρέχουν τη δυνατότητα σε μια HTML σελίδα να **ενσωματώνουμε** *δεδομένα, στατικό περιεχόμενο και JavaScript*.
- Η HTML μπορεί να γίνει **δυναμικά** στο διακομιστή και στη συνέχεια να **σταλεί** ως περιεχόμενο στο πρόγραμμα περιήγησης.
- Το EJS είναι ένα πακέτο που πρέπει να εγκατασταθεί με την εντολή `npm install ejs` .
- Άλλα view engines: Puq (https://pugjs.org/api/getting-started.html), mustache (https://www.npmjs.com/package/mustache)

Express και EJS

- Ορίζουμε ως **view engine** το **ejs**
- Αντί των συναρτήσεων **response**, **write** ή **send** χρησιμοποιούμε την **render**.
- Η συνάρτηση render παίρνει ως παράμετρο το **όνομα του EJS αρχείου** και Javascript Object που περίεχει τις *τιμές των παράμετρων* που θέλουμε να περάσουμε.
- Εξ' ορισμού όλα τα EJS αρχεία θα πρέπει να δημιουργιούνται στον **υποφάκελο views**.

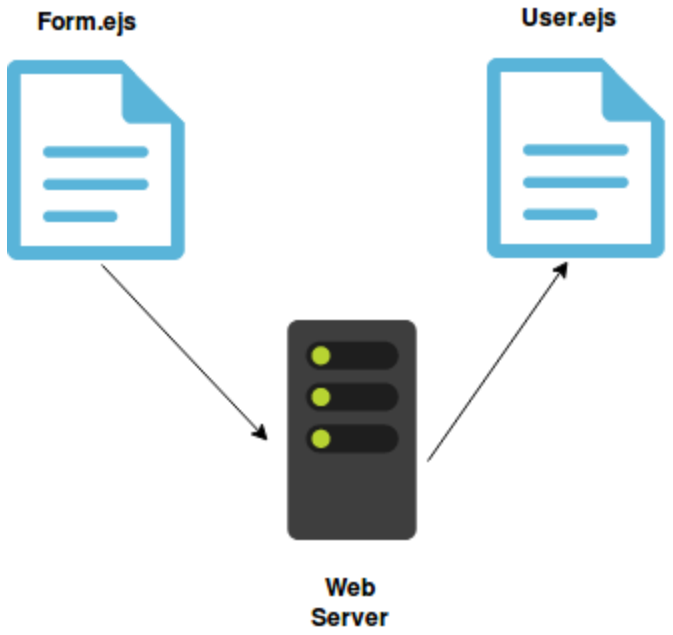
```
var express = require('express');
var app = express()
app.set('view engine', 'ejs');

app.get('/' (req,res) =>{
        res.render('welcome', {firstname: 'Κώστας'});
});
```

```
├── app.js
├── node_modules
│   ...
├── package.json
├── package-lock.json
└── views
    └── form.ejs
```

Example 7 - EJS app

Υλοποιήστε τη παρακάτω εφαρμογή με Express και EJS

- Ο χρήστης ανοιγει τη σελίδα με διεύθυνση `http://localohost:3000/create`
- Συμπληρώνει τη φόρμα που περιέχει τα απαραίτητα πεδία για τη δημιούργια ενός νέου πελάτη.
- Τα δεδομενα της φόρμας επιστρέφουν στο route /user με χρήση της μεθόδου post.
- Τα στοιχεία του νέου πελάτη εμφανίζονται στη σελίδα user.ejs.



Example 7 - EJS app

app.js

```
const express = require('express');
const app = express();
const port = 3000;

app.set('view engine', 'ejs');

// const bodyParser = require('body-parser')
// app.use(bodyParser.urlencoded({ extended: true}));
app.use(express.urlencoded({ extended: true}));

app.get('/create', (req, res) => {
   res.render('form', {});
});
```

```
app.post('/user', (req, res) => {

  let firstname = req.body.firstname;
  let lastname = req.body.lastname;
  let email = req.body.email;
  let sex = req.body.sex;

  res.render('user',
  {
    name:firstname,
    surname:lastname,
    mail:email,
    sex:sex
  });
});

app.listen(port, () => {
  console.log(`Example app listening on port ${port}`)
})
```

Example 7 - EJS app

form.ejs

```
<form method="post" action="/user">
  <ul>
    <li>
      <label for="firstname">Όνομα:</label>
      <input type="text" name="firstname" id="firstname">
    </li>
    <li>
      <label for="lastname">Επίθετο:</label>
      <input type="text" name="firstname" id="lastname">
    </li>
    <li>
      <label for="email">Email:</label>
      <input type="email" name="email" id="email">
    </li>
    <li>
      <input type="radio" name="sex" value="male">Άνδρα
      <input type="radio" name="sex" value="female">Γυναίκα
    </li>
    <li>
      <input type="submit" value="Υποβολή">
      <input type="reset" value="Καθάρισμα">
    </li>
  </ul>
</form>
```

user.ejs

```
<body>
  <p>Εμφάνιση χρήστη</p>

  Όνομα: <%= name %> <br>
  Επίθετο: <%= surname %><br>
  Email: <%= mail %><br>
  Φύλο: <%= sex %><br>

  <p><a href="/create">Επιστροφή</a></p>
</body>
```