

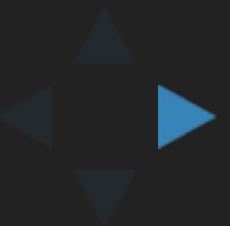
MIDDLEWARES

ASYNCHRONOUS SERVER TECHNOLOGIES

César Berezowski

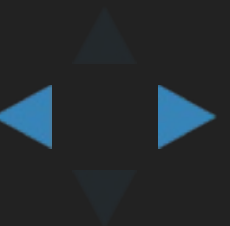
Big Data Consultant @ Adaltas

cesar@adaltas.com



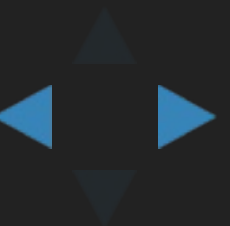
RECAP

- Developer tools: terminal, editor, github, stack overflow, travis-ci...
- Best practices on a node project :
 - `scripts`: don't repeat long and complicated commands
 - `examples`: tell people how to use your code
 - `npm`: external libraries
 - `modules`: split your code intelligently
 - `unit testing`: check that your code does what it is supposed to do
 - `transpilers`: write cleaner code faster



LAST CLASS

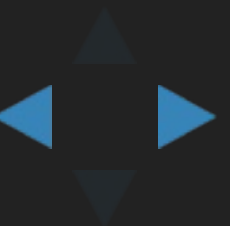
- Tools: Nodemon & Postman
- Framework: ExpressJS
- Database: LevelDB



YOUR PROJECT

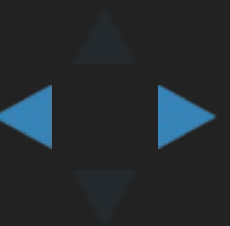
- Project on github linked to travis CI

```
myproject/  
|-- .gitignore  
|-- .travis.yml  
|-- package.json  
|-- readme.md  
|-- db/      -> levelDB files, NOT IN GIT REPO  
|-- bin/     -> scripts  
|-- src/     -> coffee code  
|-- lib/     -> compiled JS from Coffee, NOT IN GIT REPO  
|-- public/  -> static files (css/js/imges)  
|-- views/   -> pug views  
+-- test/    -> unit tests
```

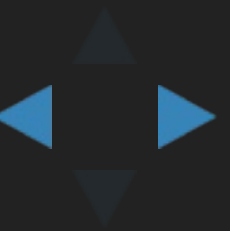


FINAL PROJECT

- Based on code from class
- Simple dashboard app :
 - User login
 - A user can insert metrics
 - A user can retrieve his metrics in a graph
 - A user can only access his own metrics



QUESTIONS ?



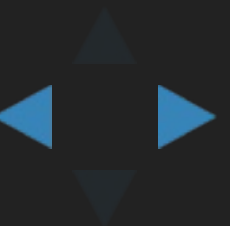
MIDDLEWARE



WHAT IS IT ?

- Very vague term, multiple definition
- In our case :

"Middleware are functions that handle requests"



EXAMPLE

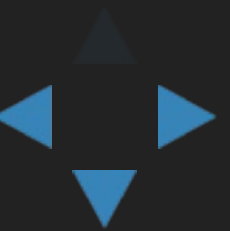
```
express = require 'express'
app = express()

myMiddleware = (req, res, next) ->
  console.log "#{req.method} on #{req.url}"
  next()

app.use myMiddleware

app.get '/', (req, res) ->
  res.status(200).send "Hello world !"

app.listen 1337, -> console.log 'listening on port 1337'
```



2ND EXAMPLE

Install **morgan** middleware with npm

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

app.use morgan 'dev'

app.get '/', (req, res) ->
  res.status(200).send "Welcome to the api"

app.get '/hello/:name', (req, res) ->
  res.status(200).send "Hello #{req.params.name}"

app.listen 1337, -> console.log 'listening on port 1337'
```



HOW TO USE IT

Global middleware

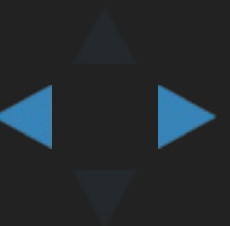
```
app.use middleware
```

Route specific middleware

```
app.get '/myroute', middleware, (req, res) ->  
  # route logic
```

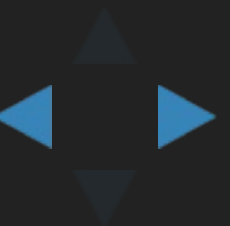
Router specific middleware

```
router = express.Router()  
  
router.use middleware  
router.get '/myroute', (req, res) ->  
  # route logic  
app.use router
```



WHAT CAN WE USE IT FOR ?

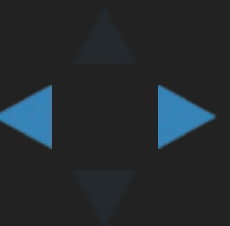
- Anything !
- Content validation / parsing
- Data completion
- User authentication / authorization
- Logging
- ...



SOME MIDDLEWARES

- body-parser
- errorhandler
- cookie-parser
- morgan
- ...

Exhaustive list, use the ones you find useful ! [Express doc on middlewares](#)



LET'S SETUP AUTHENTICATION

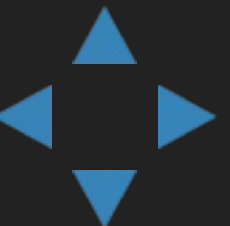
- We'll need :
 - User CRUD (Create Read Update Delete)
 - DB persistance
 - User sessions
 - User auth
 - Authorization middleware
 - Login pages
- We could also use [PassportJS](#)



USER CRUD

We need a user module !

```
module.exports =  
  get: (username, callback) ->  
    # TODO: get a user by username  
  
  save: (username, password, name, email, callback) ->  
    # TODO: save a user with it's info  
  
  remove: (username, callback) ->  
    # TODO: delete a user by username  
  
  # We won't do update
```



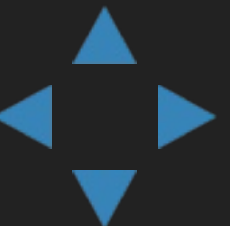
DB PERSISTENCE

get

```
db = require('./db') "#{__dirname}/../db/user"

module.exports =
  get: (username, callback) ->
    user = {}
    rs = db.createReadStream
      gte: "user:#{username}"
    rs.on 'data', (data) ->
      # parsing logic
    rs.on 'error', callback
    rs.on 'close', ->
      callback null, user
```

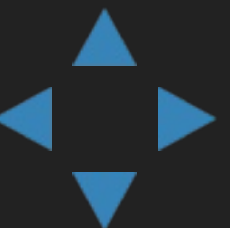
Do the save and remove by yourself



USER SESSIONS

- Install `level-session-store` middleware with npm
- Install `express-session` middleware with npm
- In our `app.coffee`:

```
session = require 'express-session'  
LevelStore = require('level-session-store')(session)  
  
app.use session  
  secret: 'MyAppSecret'  
  store: new LevelStore './db/sessions'  
  resave: true  
  saveUninitialized: true
```



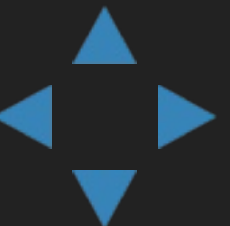
USER AUTHENTICATION

In our `app.coffee`

```
app.get '/login', (req, res) ->
  res.render 'login'

app.post 'login', (req, res) ->
  user.get req.body.username, (err, data) ->
    return next err if err
    unless # user login validation
      res.redirect '/login'
    else
      req.session.loggedIn = true
      req.session.username = data.username
      res.redirect '/'

app.get '/logout', (req, res) ->
  delete req.session.loggedIn
  delete req.session.username
```

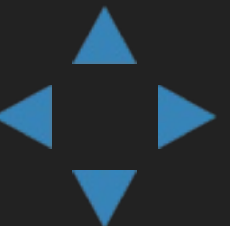


USER AUTHORIZATION MIDDLEWARE

In our `app.coffee`

```
authCheck = (req, res, next) ->
  unless req.session.loggedIn == true
    res.redirect '/login'
  else
    next()

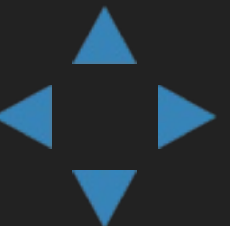
app.get '/', authCheck, (req, res) ->
  res.render 'index', name: req.session.username
```



LOGIN PAGE LAYOUT

In a `views/login.jade`

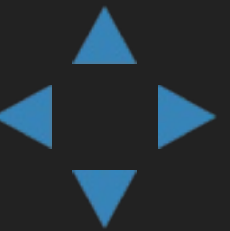
```
block content
  #form
    p Please login to your account
    hr
    form#login(action='/login', method="post")
      .form-group
        label Username
        input(type='text', name="username")
      .form-group
        label Password
        input(type='password', name="password")
    button#login_submit.btn.btn-primary.btn-block(type='submit')
      i.icon-ok.icon-white
      | Connect
    hr
    button.btn.btn-success.btn-block(type='button' href='/signup')
```



INDEX PAGE LAYOUT

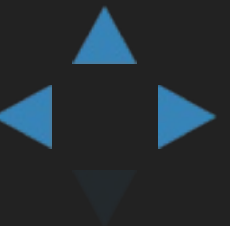
In your `index.jade`:

```
button.btn.btn-danger(href='/logout' onClick='document.location.href:')
```



YOUR TURN

- Do the `save` and `remove` functions for a user
- Do the `/signup` routes and form



QUESTIONS ?



YOUR WORK

- Fully implement `user` authentication
- Using the `metrics` module implemented for this week:
 - Create a `user-metric` relation module
 - Write the CRUD functions for this module
 - Bind them to the corresponding routes
 - Implement the mechanisms for a user to add metrics and retrieve them (only it's own !)
- On the front-end:
 - Display data accordingly on the connected user
 - Allow a user to display each of his metrics group

