ADALTAS

EXPRESSJS

ASYNCHRONOUS SERVER TECHNOLOGIES

César Berezowski

Big Data Consultant @ Adaltas

cesar@adaltas.com

ADALTAS

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

ADALTAS

YOUR PROJECT

- Project on github linked to travis CI

```
myproject/
|-- .gitignore
|-- .travis.yml
|-- package.json
|-- readme.md
|-- bin/ -> scripts
|-- src/ -> coffee code
|-- lib/ -> compiled JS from Coffee
+-- test/
```

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

ADALTAS

QUESTIONS ?

ADALTAS

TERMINAL

- Nodemon (tool)
- ExpressJS (framework)
- Postman (tool)
- LevelDB (database)

ADALTAS

NODEMON

ADALTAS

WHAT IS IT ?

- A simple utility
- Watches your development files
- Restarts the server on saving

ADALTAS

HOW TO USE IT ?

```
npm i --save nodemon
./nodmodules/.bin/nodemon src/app.coffee
```

ADALTAS

EXPRESSJS

WHAT IS IT ?

- Minimalist framework for NodeJS apps
- Provides features for web app development
- Create robust APIs
- Functions to expose a front end

ADALTAS

WHAT'S AN API ?

- Application Programming Interface
- In web: REST
  - Expose a set of HTTP routes
  - Use HTTP verbs (GET / POST / PUT / DELETE)
  - Client connects to communicate
  - Usually communicating in JSON

ADALTAS

HOW TO USE AN API ?

- Combination of two sides:
  - Back-end: rest api
  - Front-end: web pages w/ JS, mobile app, …
- Express brings both for the web !

ADALTAS

CREATE A BASIC SERVER

- Manually: use `node-http`
- With express:

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

app.set 'port', 1337

app.listen app.get('port'), () ->
  console.log "server listening on #{app.get 'port'}"
```

ADALTAS

API’S ROUTING

- Manually: parse the url and apply corresponding logic
- With Express:

```
app.get '/', (req, res) ->
  # GET

app.post '/', (req, res) ->
  # POST

app.put '/', (req, res) ->
  # PUT

app.delete '/', (req, res) ->
  # DELETE
```

ADALTAS

API'S ROUTING

You can add parameters in the routes :

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

PREPARE A FRONT END

```
npm i --save pug jstransformer-coffee-script
```

- Create a `view/` directory
- Create a `layout.pug` file in it:

```
doctype html
html(lang='en')
  head
    title My Web Page
    block head
  body
    block content
```

PREPARE A FRONT END

- Create an index.pug file :

```
extends layout

block head
  # Here will go our css/js links

block content
  p Hello world !
```

PREPARE A FRONT END

Tell express to use our pug views

```
app.set 'views', "#{__dirname}/../views"
app.set 'view engine', 'pug'
```

Render our index on /

```
app.get '/', (req, res) ->
  res.render 'index', {}
```

MAKE IT SEXY !

- Expose static content (JS, CSS, Images, ...)
- Download bootstrap getbootstrap.com/getting-started/#download
- Download JQuery code.jquery.com/jquery-2.1.4.min.js
- Add the css in public/css and the js in public/js

ADALTAS

MAKE IT SEXY !

In our `app.coffee`

```
app.use '/', express.static "#{__dirname}/../public"
```

In our index.pug

```
block head
  script(type="text/javascript" src="js/jquery-2.1.4.min.js" charset=
  script(type="text/javascript" src="js/bootstrap.min.js" charset="u
  link(rel='stylesheet', href='/css/bootstrap.min.css')
```

Notice how the font changed ?

LET’S BRING SOME AJAX

- Technologies used to dynamically update static pages
- Use JS embedded in HTML
- Get data from a server
- Update page without reloading

CREATE DUMMY DATA

- Prepare the data on the back-end
- Let’s create a new module called `metrics`:

```
module.exports =
 ###
  `get(callback)`
  --------
  returns some hard-coded metrics

  `callback`: callback function
 ###

 get: (callback) ->
  callback null, [
   timestamp:(new Date '2013-11-04 14:00 UTC').getTime(), value:1
  ,
    timestamp:(new Date '2013-11-04 14:30 UTC').getTime(), value
  ]
```

ADALTAS

CREATE DUMMY DATA

- Expose the metrics on the back-end

```
app.get '/metrics.json', (req, res) ->
  metrics.get (err, data) ->
    throw next err if err
    res.status(200).json data
```

ADALTAS

AND GET IT ON THE FRONT-END !

- In our `index.pug`

```
block content
  div.container
    div.col-md-6.col-md-offset-3
      p hello world !
      button(type="button" class="btn btn-success" id="show-metrics"
      #metrics
```

ADALTAS

AND GET IT ON THE FRONT-END !

- In our `index.pug`

```
block content
  script
    :coffee-script
      $('#show-metrics').click (e) ->
        e.preventDefault()
        $.getJSON "/metrics.json", {}, (data) ->
          content = ""
          for d in data

content += "timestamp: #{d.timestamp}, value: #{d.value}"
          $('#metrics').append content
```

ADALTAS

POSTMAN

WHAT IS IT ?

- Dashboard to test your API
- Simulate HTTP request
- Specify custom body & headers
- getpostman.com

ADALTAS

HOW ABOUT STORING ?

DATABASES

- RDBMS -> MySQL, PostGreSQL, Hive
- NoSQL
  - Column families: HBase, Cassandra
  - Document Store: MongoDB, ElasticSearch
  - Key Value: LevelDB
  - Graph DBs: Titan, Neo4J

LEVELDB

- In-memory key-value store embedded in Node
- OpenSource
- NoSQL DB, Key Value store
- Originally written by Google
- leveldb.org

WHY LEVELDB FOR OUR PROJECT ?

- It’s blazing fast
- In memory & backed by the file system
- Keys are ordered : suitable for metrics
- Data compression with Snappy
- Embedded in the app, nothing else to setup / manage

SOME LIMITATIONS

- Not an SQL database
- Only a single process at a time

LET'S SETUP

```
npm install --save level level-ws
```

- Create a `db/` directory at root

ADALTAS

USE THE DB

To open the db

```
levelup = require 'levelup'
levelws = require 'level-ws'
db = levelws levelup "path/to/db_file"
```

To write

```
db.put key, value, (err) ->
  if err then …
```

To read

```
db.get key, (err, value) ->
  if err then …
```

THE METRICS

- Key: metrics:#{id}:#{timestamp}
- Value: an integer

READ/WRITE METRICS

- One by one ? Too heavy !
- Use streaming :

```
stream = db.createReadStream(...)
stream = db.createWriteStream()
```

ADALTAS

LET’S POST SOME METRICS

In our `metrics.coffee`, add a save function

```
save: (id, metrics, callback) ->
  ws = db.createWriteStream()
  ws.on 'error', callback
  ws.on 'close', callback
  for metric in metrics
    {timestamp, value} = metric
    ws.write key: "metric:#{id}:#{timestamp}", value: value
  ws.end()
```

LET’S POST SOME METRICS

Install body-parser to parse the request’s body

```
npm i --save body-parser
```

Configure Express to use it

```
app.use bodyparser.json()
app.use bodyparser.urlencoded()
```

LET’S POST SOME METRICS

- Using Postman :
  - Set up a POST request on /metrics
  - Set the header Content-Type:application/json
  - Add an array of metrics as RAW body :

```
[
  { "timestamp":"1384686660000", "value":"10" }
]
```

ADALTAS

OR USE A SCRIPT ?

```
#!/usr/bin/env coffee

metric = require '../src/metrics'

met = [
  timestamp:(new Date '2013-11-04 14:00 UTC').getTime(), value:12
,
  timestamp:(new Date '2013-11-04 14:10 UTC').getTime(), value:13
]

metric.save 0, met, (err) ->
  throw err if err
  console.log 'Metrics saved'
```

ADALTAS

QUESTIONS ?

ADALTAS

YOUR WORK

- Front:
  - Work on the front’s layout with CSS
  - Display the metrics in a graph with `d3.js`
- Back:
  - Add get and remove to the metrics module
  - Use Postman to test the API
  - Enhance the populatedb script to add multiple metric batches