NODE PROGRAM

2014

PRESENTER

Azat Mardan:

- US Federal government
- Startups (Storify)
- Corporations (DocuSign)
- 9 books on Node

@azat_co
webapplog.com

AGENDA DAY 1

1. Introductions
2. JavaScript Fundamentals
3. Break
4. Node.js Fundamentals
5. Debugging and other tools
6. NPM: utils, fs, buffer, url, path
7. Learn You Node Workshop
8. Lunch
9. Hello World Server
10. Heroku
11. Stream Adventure Workshop
12. Functional JavaScript Workshop

AGENDA DAY 2

1. NoSQL and MongoDB
2. REST API
3. Break
4. Express.js Fundamentals
5. Jade and Stylus
6. Express REST API
7. ExpressWorks Workshop
8. Lunch
9. Production
10. Elective personal projects

JAVASCRIPT FUNDAMENTALS

EXPRESSIVNESS

LOOSE TYPING

- string
- boolean
- number
- regexp
- object

OBJECT LITERAL NOTATION

```
var obj = {
  a: 4,
  b: 9,
  x: "node.js"
}
```

FUNCTIONS

```
var f = function() {
  ...
}

function f2 () {
  ...
}

function f3() {
  ...
  return "ok"
}
```

ARRAYS

```
var arr = [1, 2, 3]
arr[0] // == 1
```

PROTOTYPAL NATURE

- Pseudo-classical (new, Object, prototype)
- Functional inheritance pattern

FUNCTIONAL INHERITANCE

```
var vehicle = function(name) {
  return {
    name: name,
    speed: 0,
    accelerate: function(speedIncrease) {
      this.speed += speedIncrease
      console.log(this)
    }
  }
}

var car1 = vehicle('toyota')
car1.accelerate(10)
console.log(car1.name + ' is going at ' + car1.speed)
```

CONVENTIONS

- Names
- Indentation
- Semi-colons
- Comma-first style

NO MODULES

CLOSURES

GLOBAL AND PRIVATE VARS

IMMEDIATELY-INVOKED FUNCTION EXPRESSIONS

```
;(function(){
    ...
}())
```

KEYWORD 'THIS'

PITFALLS

- Leaking vars into global space
- Using wrong "this" context
- Using wrong == or ===
- Missing break in a switch case
- Not using "return" when needed
- Dealing with pseudo-classical inheritance

EXERCISE

Write a "class" book with functional inheritance pattern:

- Takes the name option
- Has a method read that takes number of pages
- Remembers the number of the last page read

ANSWER

```
var book = function(name) {
  return {
    name: name,
    lastPageRead: 1,
    read: function(pages) {
      this.lastPageRead += pages
    }
  }
}

var practicalNode = book('Practical Node.js')
practicalNode.read(101)
console.log('last page read for ' + practicalNode.name + ' is ' + practicalNod
```

NODE.JS FUNDAMENTALS

EVENT LOOP

```
console.log('a')
setTimeout(function(){
  console.log('b')
}, 200)
console.log('c')
```

NON BLOCKING I/O

- Google Chrome V8 with C++ and JS/ECMA
- Super efficient!
- Can be blocked by sync code
- Can be blocked by lots of computations

READ-EVAL-PRINT LOOP

(a.k.a. Console) in Node.js

```
$ node -v
$ node
>
```

LAUNCHING NODE.JS SCRIPTS

```
$ node program.js
$ node script.js
$ node -e "console.log('hello world')"
```

NODE.JS PROCESS INFORMATION

```
$ node -e "console.log(process)"
```

```
process.argv
process.env
process.pid
```

ACCESSING GLOBAL SCOPE IN NODE.JS

```
global
```

IMPORTING AND EXPORTING MODULES

Import:

```
var fs = require('fs')
var express = require('express')
```

Export:

```
module.exports = {
...
}
module.exports = function() {
...
}
exports.a = function() {
}
```

Never:

```
 exports = ...
```

BUFFER

Buffer is a Node.js binary data type

__DIRNAME VS. PROCESS.CWD

```
var path = __dirname + fileName
console.log(process.cwd)
```

EXERCISE

Write two files:

1. a module
2. a program that executes a method from that module

ANSWER

Module (module.js):

```
exports.f = function(arr, num){
  return arr.slice(num)
}
```

Program:

```
var module = require('./module')
var arr = [1, 2, 3]
console.log(module.f(arr,2)) // [3]
```

CORE MODULES

no need to install with NPM

NET

HTTP

PATH

FS

URL

STRINGDECODER

CRYPTO

DEBUGGING AND OTHER TOOLS

- Node Inspector
- WebStorm
- nodemon/supervisor/forever/node-dev
- Foreman
- http-server/node-static(static)
- Sublime Text 2 and its plugins

NPM

NPM INSTALL

```
$ npm install express
$ npm install express@4.2.0
$ npm install express --save
$ npm install express --save-dev
```

PACKAGE.JSON

```
 {
  "name": "stream-adventure",
  "version": "2.5.0",
  "description": "an educational stream adventure",
  "bin": {
    "stream-adventure": "bin/cmd.js"
  },
  "dependencies": {
    "hyperquest": "~0.1.6",
    "ws": "~0.4.25"
  },
  "devDependencies": {
    "tape": "~2.3.0"
  }
}
```

NODE_MODULES

NPM INSTALL -G

YOULEARNNODE

```
$ sudo npm install -g learnyounode
```

WORKSHOP TIME

HELLO WORLD SERVER

(local setup)

TUTORIAL & CODE

6.1 Node.js in Rapid Prototyping with JS on page 139 in print and 153 in PDF.

HELLO WORLD SERVER

(Heroku deployment)

NECESSARY COMPONENTS

- https://toolbelt.heroku.com
- SSH keys
- SSH to Heroku
- Procfile (web: node server.js)
- package.json (npm init)

STEPS BRIEFLY

1. Make it work locally
2. Add Procfile, package.json
3. $ git init / add / git commit
4. $ heroku create
5. $ git push ...

TUTORIAL & CODE

6.1.4 Deploying "Hello World" to PaaS in Rapid Prototyping with JS on page 139 in print and 158 in PDF.

STREAM

STREAM ADVENTURE

```
$ sudo npm install -g  stream-adventure
$ stream-adventure
```

WORKSHOP TIME

FUNCTIONAL JAVASCRIPT

```
$ sudo npm install -g functional-javascript-workshop
$ functional-javascript-workshop
```

WORKSHOP TIME!

CHAT

(run-time memory store)

TUTORIALS & CODE

6.2 Chat: Run-Time Memory Version in Rapid Prototyping with JS on pages 146 in print, and 159 in PDF.

Source code: http://bit.ly/1usviBi

ENDPOINTS

POST /messages.json
GET /messages.json

FILES

- test.js: unit tests
- mb-server.js: server

MB-SERVER.JS

POST /messages/create.json
GET /messages/list.json

util.inspect
querystring.parse
exports.getMessages
exports.addMessage

CURL

```
$ curl http://127.0.0.1:1337
$ curl -X POST -d 'name=azat&message=hi' http://127.0.0.1:1337
```

NODE SCHOOL

http://nodeschool.io

DAY 2

AGENDA DAY 2

- Front-end and back-end overview
- NoSQL and MongoDB
- REST API
- Break
- Express.js Fundamentals
- Jade and Stylus
- Express REST API
- ExpressWorks Workshop
- Lunch
- Production
- Elective personal projects

FRONT-END AND BACK-END

- Traditional web
- Thick client / XHR web

MONGODB BASICS

- No relational data
- Fast, scalable, and easily distributed
- Uses JavaScript and BSON (~JSON)!!!

Commands:

```
$ mongod
$ mongo
```

Tools:

webapplog.com/mongoui

MONGO SHELL

```
> use dbname
> show collections
> db.local.find()
> db.local.insert({a:1})
```

Main methods:

```
> db.local.find({...})
> db.local.insert({...})
> db.local.save({...})
> db.local.remove({_id: ...})
> db.local.update({_id: ...})
```

EXERCISE

1. Start MongoDB server
2. Create an object in a collection using shell
3. Download mongoui and find your object
4. Update your object, check the changes

NOSQL AND MONGODB

Page 168 in Rapid Prototyping with JS (PDF)
Page 154 in print
6.4 MongoDB

CHAT REST API

(db store)

MONGOHQ

```
var uri = process.env.MONGOHQ_URL || 'mongodb://@127.0.0.1:27017'
```

```
$ heroku addons:add mongohq:sandbox
```

MONGODB LIBRARY

https://github.com/mongodb/node-mongodb-native

Alternatives:
Mongoskin
Mongoose
Monk
Magnolia

TUTORIALS & CODE

6.5 Chat: MongoDB Version page 176 in PDF and 158 in print

http://bit.ly/1AkobeN

EXPRESS.JS FUNDAMENTALS

APP STRUCTURE

1. Includes
2. Instantiations
3. Configurations
4. Middleware
5. Routes
6. Boot-up

MIDDLEWARE

Almost always use:

- static
- body-parser
- express-session
- compression
- all from cheatsheet*

* http://bit.ly/Us2qbP

EXPRESS.JS STACK

- Jade
- Stylus
- Mongoose or Mongoskin

GENERATOR

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

OTHER FRAMEWORKS

- Hapi
- Sails
- Derby

many others at
http://nodeframeworks.com

EXPRESS.JS HELLO WORLD

CHAT REST API SERVER

(Express.js)

INSPIRATION

Code: https://github.com/azat-co/rest-api-express

Description: http://bit.ly/1jy30tn

JADE AND STYLUS

EXPRESSWORKS WORKSHOP

PRODUCTION

NODE.JS STACK

- Heroku
- AWS with Nginx + Varnish Cache + Upstart scripts
- MongoDB or MongoHQ/MongoLab: Mongoskin, Mongoose
- Logging with Winston, Elastic Kibana or Papartrailapp

TIPS

- Deployment scripts with Salt (saltstack.com)
- Increase MaxSockets
- Lock versions
- Don't trust user input
- Have good error handling in place
- Use upstart or forever

ELECTIVE PERSONAL PROJECTS

- Full-stack*
- Integration
- Deployment

* Come up with an idea, e.g., todo app

FULL-STACK

Write an front-end application for Chat (or download Backbone.js version from https://github.com/azat-co/rpjs/tree/master/board).

Make it work with your REST API Chat server.

INTEGRATION

Write an OAuth 1.0 server that can sign in with Twitter (or another provider).

You can use OAuth 1.0 Sign in with Everyauth from Introduction to OAuth with Node.js

DEPLOYMENT

Deploy Node.js application (Hello World) to AWS

END

If it's not fun, it's not JavaScript!

@azat_co