# Fullstack JavaScript

**Tamas Piros**
Full Stack Training Ltd
tamas@fullstacktraining.com
http://fullstacktraining.com
http://me.tamas.io

## Agenda

- ES2015
- TypeScript
- Angular 4
- Node.js
- NoSQL
- Three tiered architectures
- Deployment

## ES2015

## Ecma

- Standards organisation
- Develops standard for JavaScript (as well as JSON and other standards)
- ES2015 demarcates the latest addition to JavaScript
- ES2016 (or “ES7”) is in the works
- http://kangax.github.io/compat-table/esnext/

## ES2015

- const, let
- Object destructuring
- Arrow functions
- Rest, spread operator
- Template literals
- Object literals
- Generators, iterators
- Promises
- Classes

If the environment can’t understand ES2015 “transpilation” is required.

Tools such as ‘Babel’ and ‘Traceur’ can help

## `const` & `let`

- Still do hoisting like **`var`** but they are both block scoped
- General rule of thumb: use either but do not use **`var`**  (forget that it even existed)
- Use **`const`** when declaring static, non-changing variables values (warning: using **`const`** doesn’t mean creating immutable objects!)

## Object destructuring

- Object destructuring assignment
- Works on objects (and of course arrays)
- Allows for assigning variables for items in an object/array, as well as to create aliases

## Arrow functions

- New way of declaring functions in JavaScript, it’s not only more terse but has some benefits
- Bound to their lexical scope (**`this`**, anyone?)

## Rest parameters, spread operator + default params

- Easier way to add arguments to functions (rest parameters)
- Can add any number of arguments
- Spread allows for invoking dynamically generated functions (without **`.apply()`**)
- Finally default params can be specified for functions

## Template literals

- Uses backticks -- `` -- and the **${}** syntax
- Easier way to utilise variables within the codebase (interpolation!)
- Also allows for calling functions

## Object literals

- Allows for using property shorthands
- Capable of utilising computed property names

## Generators & Iterators

- Iterator and iterable protocol define how to iterate over any object
- Iterable is a method that returns an iterator object, which has a next() method
- The next method returns objects with two properties, value and done
  - value: current value of sequence
  - done: indicates whether more items are available in the iteration
- Loop through it using **`for...of`**

## Generators & Iterators

- A generator function is a special kind of iterator that uses **`function* ()`** and **`yield`** / **`yield*`**
- A generator function execution is suspended, and it also remembers the last position
    - Four potential options: **`yield`**, **`return`**, **`throw`** and **`{ done: true }`**

## Promises

- Make synchronous code asynchronous
- A promise can either resolve (success, fulfilled) or reject (error, rejected)
- Utilises `.then()` and `.catch()` method
- Allows for chaining!

## Classes

- Syntactic sugar over ES5’s prototyping
  - Simpler syntax
- JavaScript is *still* a prototype based language

## TypeScript

## TypeScript

- Superset of JavaScript
- Supports typings! (big thing!)
  - Boolean, Number,  String, Array, Tuple, Enum, Any, Void, Null & Undefined, Never
  - Type assertion
- Needs transpiling
- Classes, Interfaces, Modules, Namespaces are all native to TS

## npm

## npm

- npmjs.com
- Package manager for Node.js
- Node.js installation also installs npm
    - 5.0.3 is the latest version
    - **`npm -v`** to get version number
- Always try to get the latest version
    - **`npm install npm@latest -g`**

## npm

Some important commands / options for npm

- **`npm install <package> [-g] --production`** (or `npm i <package> [-g]`)
    - Install package, globally (for example gulp)
- **`npm uninstall <package> [-g]`**
- **`npm init`**
- **`npm update`**
- **`npm outdated [-g] [--depth=0]`**

## npm - Semantic Versioning

Semantic Versioning ("semver") - 1.0.0

| major | . | minor | . | patch |
|---|---|---|---|---|
| 1 | | 3 | | 5 |

Bug fix and minor change

- patch release: increment last number (1.3.**6**)

New features which are non breaking

- Minor release: increment middle number (1.**4**.5)

Changes that break backward compatibility

- Major release: increment first number (**2**.3.5)

**package@2.0.0** - install package @ version 2.0.0
**package >= 1.2.7** - install package 1.2.7 or 1.2.8 or even 2.5.5 but not 1.2.6
**package ~1.2.3** - install patch level changes (1.2.3, 1.2.4, but not 1.3.0)
**package ^1.2.3** - install patch and minor updates (1.2.3, 1.2.4, 1.5.6, but not 2.0.0)

## package.json

File containing all the packages for a given project, including version numbers

Ideal for distributing project amongst team, or on GitHub

To create a new package.json file: **`$ npm init`**

**`$ npm i(nstall)`** - will install all packages

All installed modules go to **`node_modules/`**

```json
{
  "name": "01-npm",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Tamas Piros",
  "license": "MIT",
  "dependencies": {
    "lodash": "^4.17.4"
  },
  "devDependencies": {
    "winston": "^2.3.1"
  }
}
```

## package.json - scripts

`npm start` - executes 'scripts/start'

`npm run [x]` - executes 'scripts/x'

```
{
  "name": "01-npm",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "start": "echo starting",
    "runMe": "node -e 'console.log(\"hello from Node.js\")'"
  },
  "author": "Tamas Piros",
  "license": "MIT",
  "dependencies": {
    "lodash": "^4.17.4"
  },
  "devDependencies": {
    "winston": "^2.3.1"
  }
}
```

## npm - changing version numbers

Version numbers can change frequently - i.e. within days

Best option is to lock version numbers

Before v5.3.0: npm-shrinkwrap (https://docs.npmjs.com/files/shrinkwrap.json)

After v5.3.0: package-lock.json is created automatically

## Angular 4

## Angular 4

- Popular frontend framework by Google
- Written in TypeScript - can be used with TypeScript or ES2015
- Create applications for any platform (web or mobile)
- Angular CLI
- Components, Services, Modules, Pipes

## Angular CLI

- Install it via npm (npm i -g @angular/cli)
- ng [command] { options }
  - new
  - lint
  - generate
- .angular.json - config file for the application

## Node.js LTS vs Current

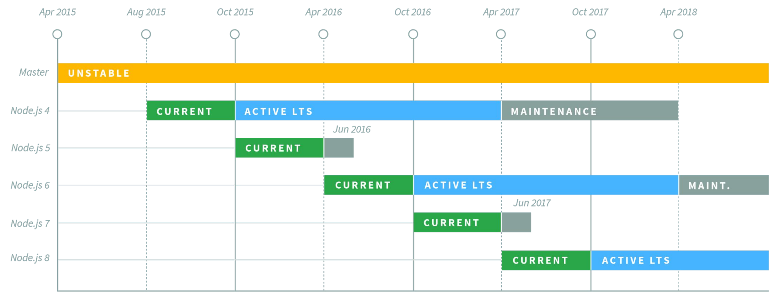
## Node.js LTS vs Current

**LTS** - Long Term Support for each major version

Always guaranteed to be stable

Still allows for security updates

Node.js Long Term Support (LTS) Release Schedule



**Current** version has more features, including more ‘experimental’ features

Adds more API support

One recent notable example: experimental HTTP2 support

## Basics of Node.js

## Node.js basics

Event-driven

- Flow control is determined by events or by changes in state
- Listens for events, calls a callback once and event has been detected

Non-blocking

- Asynchronous approach

## require() and module.exports

You can not only require packages installed via npm but also custom packages

use module.exports to define what to export

use **`require(‘/path/to/file’);`** to import package

## Node.js - HTTP

One of the built-in modules (others include `fs, utils`)

Allows for the creation of basics HTTP apps, for more advanced/complex HTTP applications use an npm package:

- Express, HAPI, KOA, Restify

## Full stack development

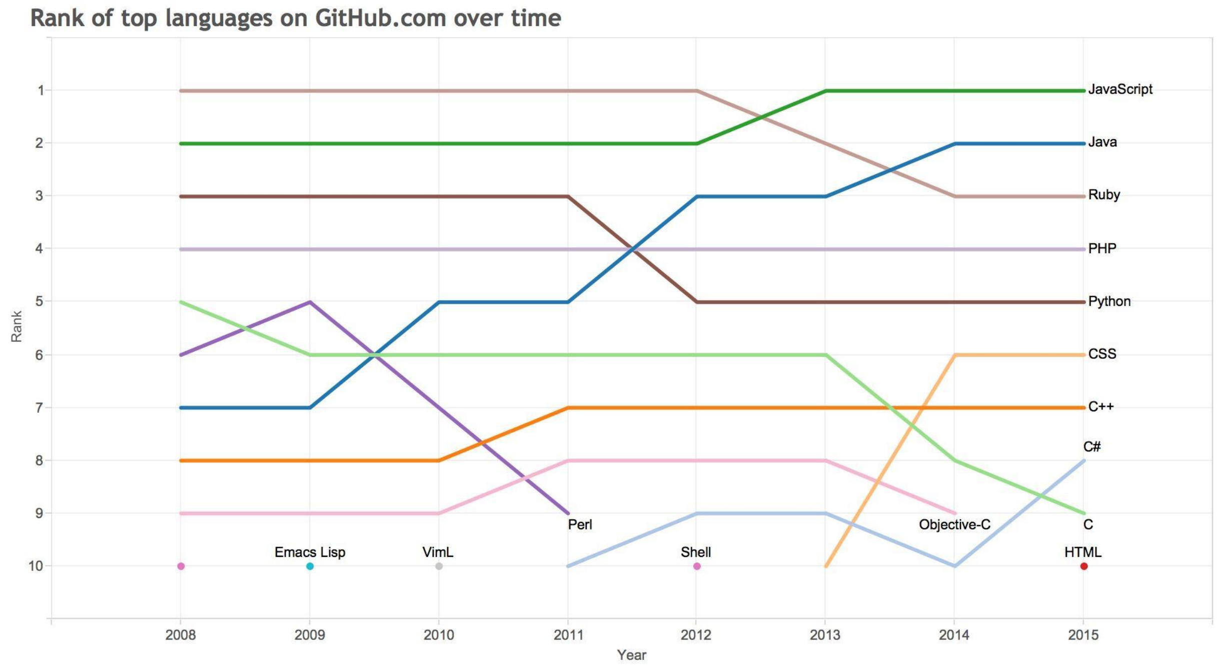
## Full stack development

- A full stack developer is able to - at a basic level - understand:
  - Architecture components
  - Backend languages
  - Frontend languages and design
  - Basic algorithms
  - Deployments
- The power of full stack development using JavaScript is the language itself

## JavaScript everywhere

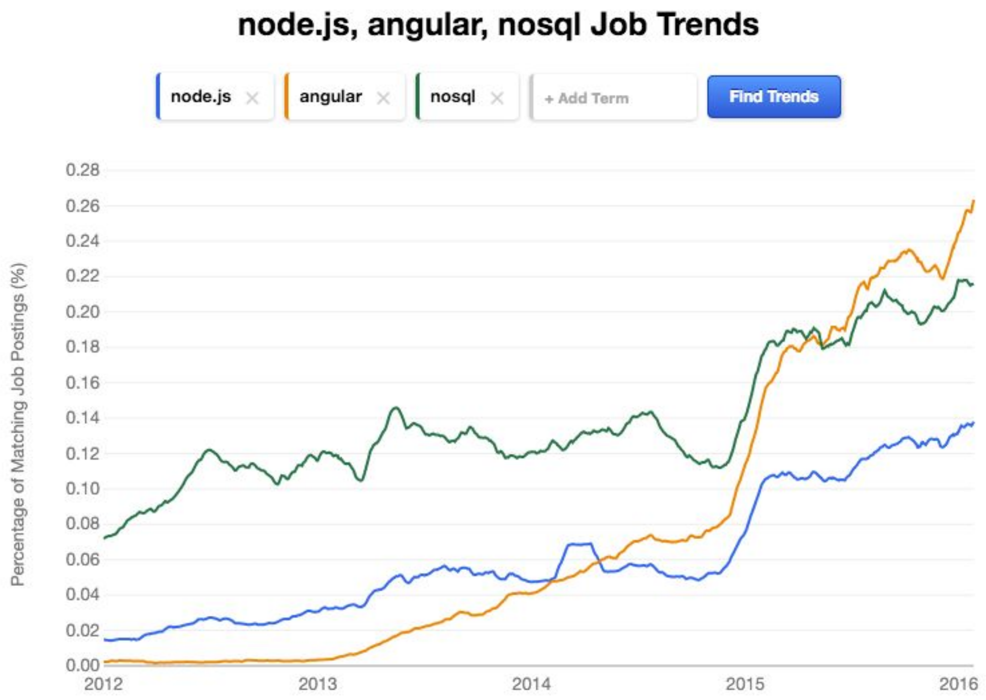
- JavaScript at the backend and frontend means:
  - Same data-structures, data-types, same “functionality”
  - No conversion required
  - No need to learn multiple languages

# JavaScript is eating the world



Source: GitHub.com

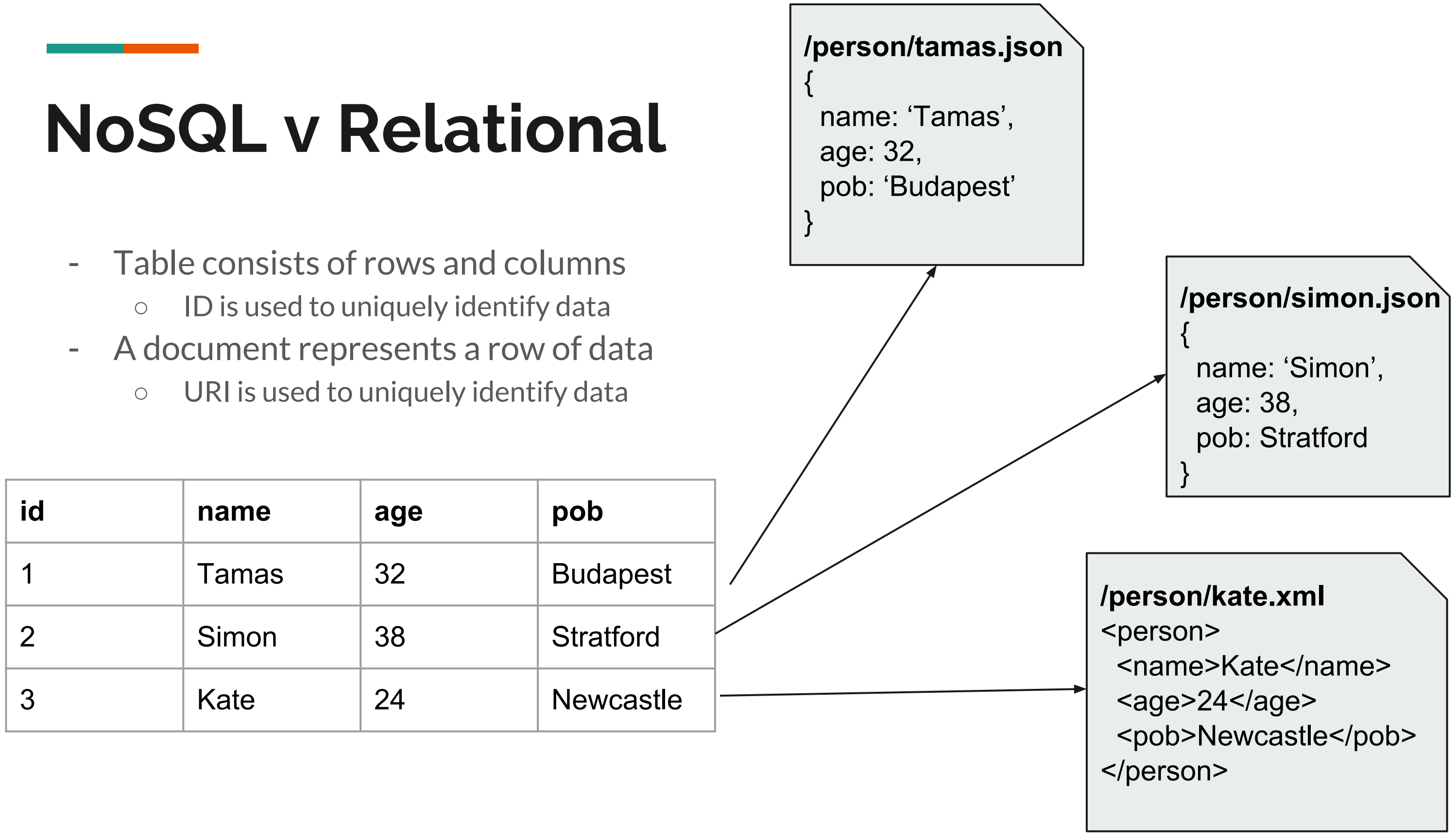
## And if you need more convincing



## NoSQL 101

## NoSQL v Relational

- Table consists of rows and columns
  - ID is used to uniquely identify data
- A document represents a row of data
  - URI is used to uniquely identify data



| id | name | age | pob |
|---|---|---|---|
| 1 | Tamas | 32 | Budapest |
| 2 | Simon | 38 | Stratford |
| 3 | Kate | 24 | Newcastle |

**/person/tamas.json**

```
{
  name: ‘Tamas’,
  age: 32,
  pob: ‘Budapest’
}
```

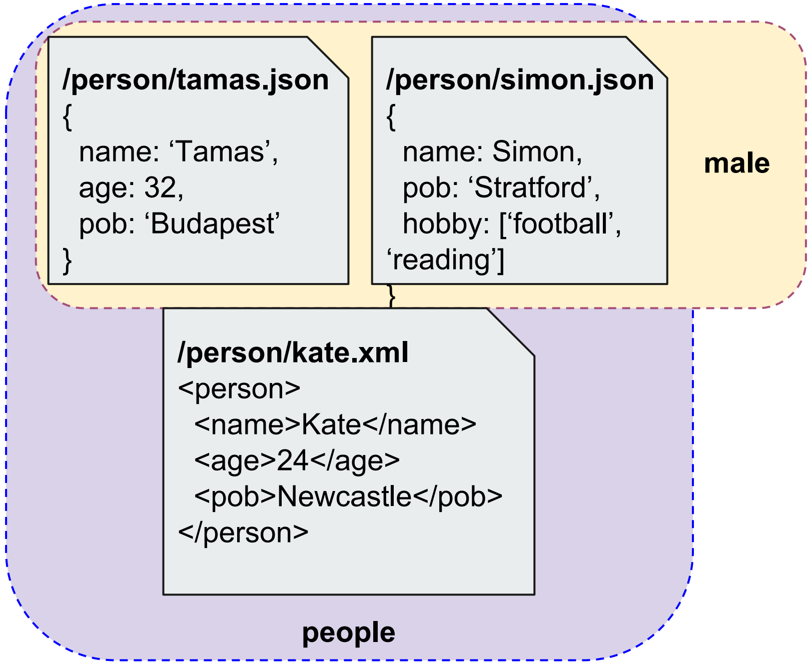
**/person/simon.json**

```
{
  name: ‘Simon’,
  age: 38,
  pob: Stratford
}
```

**/person/kate.xml**

```
<person>
  <name>Kate</name>
  <age>24</age>
  <pob>Newcastle</pob>
</person>
```

## NoSQL v Relational



```
/person/tamas.json
{
  name: 'Tamas',
  age: 32,
  pob: 'Budapest'
}
```

```
/person/simon.json
{
  name: Simon,
  pob: 'Stratford',
  hobby: ['football',
'reading']
}
```

```
/person/kate.xml
<person>
  <name>Kate</name>
  <age>24</age>
  <pob>Newcastle</pob>
</person>
```

- Tables are like collections (labels) but…
- Documents can belong to zero, one or multiple collections
- Document with a different structure and format can be in the same collection
- No need to account for 'NULL' values

## Putting all this together

## Application Architecture

User Interface

- Data views
- User workflow
- Browser

JSON over HTTP

Middle-tier

- Business rules
- Application logic

**Pros**

- Same language throughout the stack
- Lightweight data format
- Data format ‘natively’ understood by JavaScript

**Con(s)**

- Missing persistent data storage

## Application Architecture

### User Interface

- Data views
- User workflow
- Browser

JSON over HTTP

### Middle-tier

- Business rules
- Application logic

JSON/XML over HTTP

### Database-tier

- Persistent storage

**MarkLogic** can:

- store JSON documents natively (along with XML, binary and RDF)
- allow you to construct queries using JavaScript
- have ACID properties instead of eventual consistency
- Give you all the indexes you need and allow you to execute search out of the box
- Apply role based, document level security
- Execute SPARQL queries
- Manage the database via REST API calls

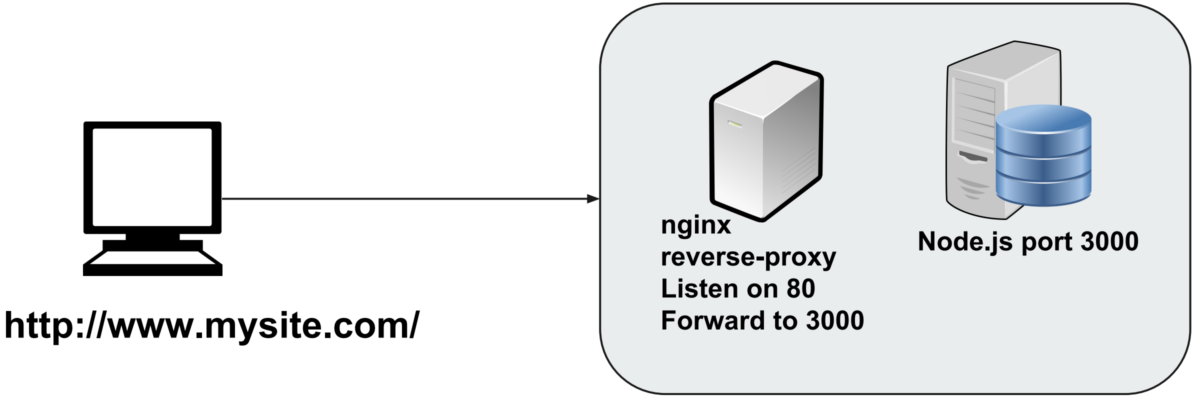
## Deployment

## For development

- Use **`nodemon`** - a tool that keeps the process running and restarts upon detecting changes ([https://www.npmjs.com/package/nodemon](https://www.npmjs.com/package/nodemon))
- Can configure it via **`nodemon.json`**

## For production

- Use **forever** to keep a Node.js process running (https://www.npmjs.com/package/forever)
- Use nginx and reverse-proxy



## LiveReload & BrowserSync

- Both allows to reload browser when code changes
- Faster development
- Integrates with gulp (and grunt)
- Customisation via config files
- Synchronised cross-device testing

## gulp

- Gulp is a task automation tool (get it from npm, install it globally)
- Allows for enhanced workflows
- Utilises pipes (similar to *nix pipes)
- Plugin based
  - Linting
  - Concatenation
  - Minification (JS, CSS, HTML) / compilation / transpilation
  - etc