

#### INTERNET SYSTEMS

# NODE.JS

TOMASZ PAWLAK, PHD MARCIN SZUBERT, PHD INSTITUTE OF COMPUTING SCIENCE, POZNAN UNIVERSITY OF TECHNOLOGY

### PRESENTATION OUTLINE

- What is Node.js?
- Design of Node.js
- Modules
- Web frameworks for Node.js
- Node tools for front-end developers

### MOTIVATION

- Node.js might be the most exciting single piece of software in the current JavaScript universe — used by LinkedIn, Groupon, PayPal, Walmart, etc.
- Node.js is one of the most watched projects on GitHub; it has more than million modules in npm package manager.
- Node.js combined with a client-side MV\* framework, a NoSQL database (such as MongoDB or CouchDB) and JSON offers a unified JavaScript development stack.

### WHAT IS NODE.JS?

- Node.js is a platform built on Chrome's JavaScript V8 runtime for building scalable network applications.
- Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices. <u>www.nodejs.org</u>
- Node is a way of running JavaScript outside the browser.

### WHEN TO USE NODE.JS?

- Node.js allows the creation of web servers and networking tools, using JavaScript and a collection of modules that handle various core functionality.
- Modules handle file system I/O, networking, binary data (buffers), cryptography functions, data streams, etc.
- Frameworks can be used to accelerate the development of web applications; common frameworks are Express.js, Koa, Sails.js, Total.js.
- Node.js is not limited to the server-side it provides a number of tools for frontend development workflows.

### NODE.JS AND JAVASCRIPT

- JavaScript is a compilation target, and there are a number of languages that compile to JavaScript already.
- JSON is a very popular data interchange format today and is native to JavaScript.
- JavaScript is the language used in various NoSQL databases so interfacing with them is a natural fit.
- V8 gives Node.js a huge boost in performance because it prefers straight compilation into native machine code over executing bytecode or using an interpreter.

### PRESENTATION OUTLINE

- What is Node.js?
- Design of Node.js
- Modules
- Web frameworks for Node.js
- Node tools for front-end developers

# THE BIRTH OF NODE.JS

 Ryan Dahl started the Node.js project in 2009 out of frustration with the current state of web servers in the industry.



- The core premise behind Node's design: most web applications are Input/Output (I/O) bound.
- I/O-bound programs are constrained by data access. These are programs where adding more processing power or RAM often makes little difference — the bottleneck of their performance is the latency of I/O.

### I/O LATENCY

Operation	CPU Cycles
L1	<b>3</b> cycles
L2	<b>14</b> cycles
RAM	<b>250</b> cycles
DISK	<b>41 000 000</b> cycles
NETWORK	<b>240 000 000</b> cyles

TABLE FROM RYAN DAHL'S 2009.11.08 PRESENTATION ON NODE.JS <u>HTTPS://NODEJS.ORG/JSCONF.PDF</u>

### BLOCKING I/O

 In many programming languages I/O operations are blocking — they block the progress of the program while waiting on an I/O task such as reading from the hard drive or making a network request.



### SCALING WITH THREADS

- If a program blocks on I/O, what does the server do when there are more requests to handle?
- A typical approach is to use multithreading employ one thread per connection and set up a thread pool for those connections.



#### Handles up to 4 concurrent requests

### SCALING WITH THREADS

- While this approach allows us to **scale by adding more threads**, each thread still spends most of its time waiting for I/O, not processing your application logic.
- Unfortunately, continuing to add threads introduces context switching overhead and uses considerable memory to maintain execution stacks.



#### Handles up to 4 concurrent requests

### SCALING WITH NODE.JS

 Node.js employs a single thread, using non-blocking I/O
 — any function performing I/O is handled asynchronously
 and then triggers a callback.

```
db.query("select..", function (result) {
    doSomething(result);
});
nextTask();
```



### SCALING WITH NODE.JS

- By using a single thread with an event loop, instead of threads, Node.js supports tens of thousands of concurrent connections without incurring the cost of thread contextswitching.
- However, tasks in the event loop must execute quickly to avoid blocking the queue — be careful with CPU intensive tasks.



### EVENT-DRIVEN PROGRAMMING

- Event-driven programming a programming paradigm in which the flow of the program is determined by events such as user actions, sensor outputs, or messages from other programs.
- The dominant paradigm used in **graphical user interfaces** and applications (e.g. **JavaScript web apps**) that are centered on **performing certain actions in response to user input**.
- Writing event-driven programs is easy if the programming language provides high-level abstractions, such as closures.
- JavaScript is an event-driven language it has always dealt with user interaction, employed an event loop to listen for events, and callback functions to handle them.

# NODE.JS IS EVENT-DRIVEN AND ASYNCHRONOUS

- Node.js brings event-driven programming to the server.
- The fundamental design behind Node is that an application is executed on a **single thread**, and all **events** are handled **asynchronously**.
- Node.js uses the event loop architecture to make programming highly scalable servers both easy and safe.
- Programming concurrency is hard Node sidesteps this challenge while still offering impressive performance.
- To support the event-loop approach, Node supplies a set of nonblocking I/O modules — these are interfaces to things such as the filesystem or databases, which operate in an event-driven way.

### PRESENTATION OUTLINE

- What is Node.js?
- Design of Node.js
- Modules
- Web frameworks for Node.js
- Node tools for front-end developers

### MODULES

- Modules are reusable software components that form the building blocks of applications.
- Modular programming is a software design technique that emphasizes separating the functionality of a program into independent, interchangeable modules such that each covers only one aspect of the desired functionality.

### • Modules should be **FIRST**:

- Focused.
- Independent.
- **R**eusable.
- Small.
- **T**estable.

### MODULES IN JAVASCRIPT

### • Benefits of modular programming include:

- easier understanding of your large system
- simplified debugging
- separation of concerns
- an increase in maintainability
- code reuse
- Although ECMAScript 5 does not have built-in support for modules, there are ways to define modules in JavaScript:
  - the module pattern,
  - **CommonJS** modules (the inspiration for Node modules)
  - AMD (Asynchronous Module Definition)

### MODULE PATTERN IN ECMASCRIPT5

```
var testModule = (function () {
  var counter = 0;
  return {
    incrementCounter: function () {
      return counter++;
   },
    resetCounter: function () {
      console.log( "counter value prior to reset: " + counter );
      counter = 0;
})();
```

```
// Increment our counter
testModule.incrementCounter();
```

// Check the counter value and reset
testModule.resetCounter();

### COMMON.JS MODULES

- **CommonJS** is a project with the goal of specifying an **ecosystem** for **JavaScript** outside the browser (for example, on the server)
- CommonJS provides specification for JavaScript environments that attempts to make engine implementations more compatible.
- CommonJS describes a simple syntax for JavaScript programs to require (or import) other JavaScript programs into their context.
- The Node module system is an implementation of the CommonJS specification.
- A Node module is a JavaScript library that can be modularly included in Node applications using the require() function.

### COMMON.JS MODULES

```
//math.js
exports_add = function() {
    var sum = 0, i = 0, args = arguments, l = args.length;
    while (i < l) {
        sum += args[i++];
    return sum;
};
//increment.js
var add = require('math').add;
exports.increment = function(val) {
    return add(val, 1);
};
//program.js
var inc = require('increment').increment;
var a = 1;
console.log(inc(a)); // 2
```

### MODULES IN NODE.JS

 Node core is made up of about two dozen modules, some lower level ones like events and stream, some higher level ones like http.

// Load the http module to create an http server.
var http = require('http');

// Configure our HTTP server to respond with Hello World
var server = http.createServer(function (request, response) {
 response.writeHead(200, {"Content-Type": "text/plain"});
 response.end("Hello World\n");
});

// Listen on port 8000, IP defaults to 127.0.0.1
server.listen(8000);

// Put a friendly message on the terminal
console.log("Server running at <u>http://127.0.0.1:8000/</u>");

### MODULES IN NODE.JS

- Node core is supposed to be small, and the modules in core should be focused on providing tools for working with common protocols and formats in a way that is cross-platform.
- For everything else there is npm; anyone can create a node module that adds some functionality and publish it to npm.
- The idea of **many small programs** working together is one of the **foundations of Node.js**.
- This helps us steer clear of large monolithic libraries such as jQuery — libraries of that size would be split up into smaller modules, allowing the user to use only what they require.





- **npm** makes it easy for JavaScript developers to share and reuse code in the form of modules.
- **npm** comes preinstalled with Node distributions.
- npm runs through the command line and allows to retrieve modules from the public package registry maintained on <u>http://npmjs.org</u>
- It is the fastest growing package manager: <u>http://www.modulecounts.com</u>

### PACKAGE.JSON



- A package is a folder containing a program described by a package.json file — a package descriptor.
- A package descriptor is used to store all metadata about the module, such as name, version, description, author etc.
- This file is a manifest of your Node project and should be placed at your project root to allow:
  - reinstalling your dependencies by defining a dependencies field;
  - publishing your module to npm by defining the name and version fields,
  - storing common scripts related to the package by defining the scripts object.

### PRESENTATION OUTLINE

- What is Node.js?
- Design of Node.js
- Modules
- Web frameworks for Node.js
- Node tools for front-end developers

### EXPRESS.JS

- Express.js is the **most popular** Node.js web application framework used today.
- Express.js is a minimal yet flexible and powerful web development framework inspired by Sinatra.
- Features of Express.js include:
  - Robust routing
  - Focus on high performance
  - View system supporting several **template engines**
  - Content negotiation
  - Executable for **generating applications** quickly

### EXPRESS.JS

```
var express = require('express');
var app = express();
app.get('/', function (req, res) {
  res.send('Hello World!');
});
var server = app.listen(3000, function () {
  var host = server.address().address;
  var port = server.address().port;
  console.log('Listening at http://%s:%s', host, port);
```

});

### RESTIFY

- Restify is a Node.js module built specifically to enable you to build correct / strict REST web services that are maintanable and observable.
- It intentionally borrows heavily from express as that is more or less the de facto API for writing web applications on top of node.js.
- Express' use case is targeted at **browser applications** and contains a lot of functionality, such as templating and rendering, to support that. **Restify does not**.
- Restify gives control over interactions with HTTP and full observability into the latency and the other characteristics of the applications — automatic DTrace support for all your handlers.





- Sails is the most popular MVC framework for Node.js built on top of Express and inspired by Ruby on Rails.
- Sails.js API scaffolding (blueprints) allows to automatically build RESTful JSON API for your models.
- Sails bundles a powerful **ORM**, Waterline, which provides a simple data access layer that just works, no matter what database you're using.
- Sails supports WebSockets with no additional code Sails translates incoming socket messages for you, they're automatically compatible with every route in your Sails app.

### KOA.JS

- Koa is a next-generation web framework designed by the team behind Express, which aims to be a smaller, more expressive, and more robust foundation for web applications and APIs.
- Koa employs **generators**, a feature that's a part of the **ECMAScript 6** specification,
- Koa aims to save developers from the spaghetti of callbacks, making it less errorprone and thus more manageable.

### GENERATORS IN ECMASCRIPT 6

The function\* declaration defines a generator function:

function\* name([param[, param[, ... param]]]) {
 statements
}

- Generators are functions which can be exited and later re-entered. Their context (variable bindings) will be saved across re-entrances.
- Calling a generator function does not execute its body immediately; an iterator object for the function is returned instead.

### GENERATORS IN ECMASCRIPT 6

- When the iterator's next() method is called, the generator function's body is executed until the first yield expression.
- The next() method returns an object with a value property containing the yielded value and a done property which indicates whether the generator has yielded its last value.

```
function* idMaker(){
  var index = 0;
  while(index < 3)
    yield index++;
}</pre>
```

```
var gen = idMaker();
```

```
console.log(gen.next().value); // 0
console.log(gen.next().value); // 1
console.log(gen.next().value); // 2
console.log(gen.next().value); // undefined
```

### PRESENTATION OUTLINE

- What is Node.js?
- Design of Node.js
- Modules
- Web frameworks for Node.js
- Node tools for front-end developers

## NPM ON THE CLIENT-SIDE

- A common misconception about Node and npm is that they can only be used for server side JavaScript modules.
- Node.js is not limited to the server-side the npm modules themselves can be whatever you want.
- The cutting-edge tools for the latest versions of HTML and JavaScript are developed in the Node universe:
  - package managers
  - task runners
  - generators
- **npm** is the gateway to other tools.

### BROWSERIFY



- Browserify is a build tool that lets you use Node's CommonJS module system for frontend JavaScript development.
- Browserify tries to convert any Node module into code that can be run in browsers — it brings modularity to the browser.
- It integrates seamlessly with npm, and you can use the same npm workflow for installing and publish modules.
- Browserify also opens up the possibility of code reuse between the server and the client.

### BROWSERIFY



- Browserify starts at the entry point files that you give it and searches for any require() calls it finds using static analysis of the source code's abstract syntax tree.
- For every require(), browserify resolves those modules to file paths and then searches those paths for require() calls recursively until the entire dependency graph is visited.
- Each file is concatenated into a **single javascript bundle file**.
- This means that the bundle you generate is completely selfcontained and has everything your application needs to work.





- Bower is to the web browser what npm is to Node.js.
- It is a package manager built by Twitter for your front-end development libraries like jQuery, Bootstrap etc.
- Bower is optimized for the front-end. Bower uses a flat dependency tree, requiring only one copy for each package, reducing page load to a minimum.
- It works similarly to npm, using bower.json file and providing command line interface with bower install, bower init and bower search commands.





- Grunt is the JavaScript Task Runner which allows to automate repetitive tasks like minification, compilation, unit testing, end-to-end testing, linting, etc.
- Grunt and Grunt plugins are installed and managed via npm.
- Once installed, we can execute grunt on the command line. This tells Grunt to look for a Gruntfile.js file which typically resides next to package.json in the root directory of the project.
- Gruntfile is the entry point to our build, which can define tasks inline, load tasks from external files and configure these tasks.

```
// Every Gruntfile defines the "wrapper" function
module.exports = function(grunt) {
```

```
// Project and tasks configuration
grunt.initConfig({
    pkg: grunt.file.readJSON('package.json'),
    jshint: {
      files: ['Gruntfile.js', 'src/**/*.js', 'test/**/*.js']
    },
    watch: {
      files: ['<%= jshint.files %>'],
      tasks: ['jshint']
    }
});
```

// Loading Grunt plugins and tasks.
grunt.loadNpmTasks('grunt-contrib-jshint');
grunt.loadNpmTasks('grunt-contrib-watch');

```
// A very basic default custom task.
grunt.registerTask('default', 'Log some stuff.', function() {
   grunt.log.write('Logging some stuff...').ok();
});
```



### YEOMAN



- Yeoman has become the de-facto standard scaffolding toolkit for creating modern JavaScript applications.
- Yeoman is build around **generators** for particular types of projects and provides the infrastructure for running them.
- Yeoman helps you kickstart new projects, prescribing best practices and tools to help you stay productive.
- Yeoman as well as its generators are distributed as a Node modules.

### YEOMAN WORKFLOW

- The Yeoman workflow comprises three types of tools for improving your productivity when building a web app:
  - the **scaffolding** tool (yo),
  - the build tool (Grunt, Gulp, etc)
  - the **package manager** (like Bower and npm).



### CONCLUSIONS

- The fundamental design behind Node is that an application is executed on a single thread, and all events are handled asynchronously.
- Like any technology, Node is not a silver bullet. It just helps you tackle certain problems and opens new possibilities.
- Node is extremely extensible, with a large volume of community modules that have been built in the relatively short time since the project's release.
- Among modules there are many **tools** that can improve also **client-side web development**.

### REFERENCES

- Node: Up and Running Tom Hughes-Croucher, Mike Wilson O'Reilly Media, Inc., 2012, available online at: <u>http://chimera.labs.oreilly.com/books/1234000001808</u>
- Speaking JavaScript Axel Rauschmayer, O'Reilly Media, Inc., 2014 <u>http://speakingjs.com</u>
- Node.js in Action Alex Young, Bradley Meck, and Mike Cantelon with Tim Oxley, Marc Harter, T.J. Holowaychuk, and Nathan Rajlich, Manning Publications, 2017
- JavaScript Guide at Mozilla Developer Network <u>https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide</u>
- <u>http://jsbooks.revolunet.com</u>
- <u>http://javascript.crockford.com</u>