Developing Restful Services

Chapter3:

NodeJS

Chapter Objectives

In this chapter, we will discuss:

- How to use node.js to become your web server
- What are modules?
- How to access databases from JavaScript

Chapter Concepts

- **Features and First Steps**
- The package.json File
- Basic Database Access
- JSON data-mock File
- Chapter Summary

What Is Node.js?

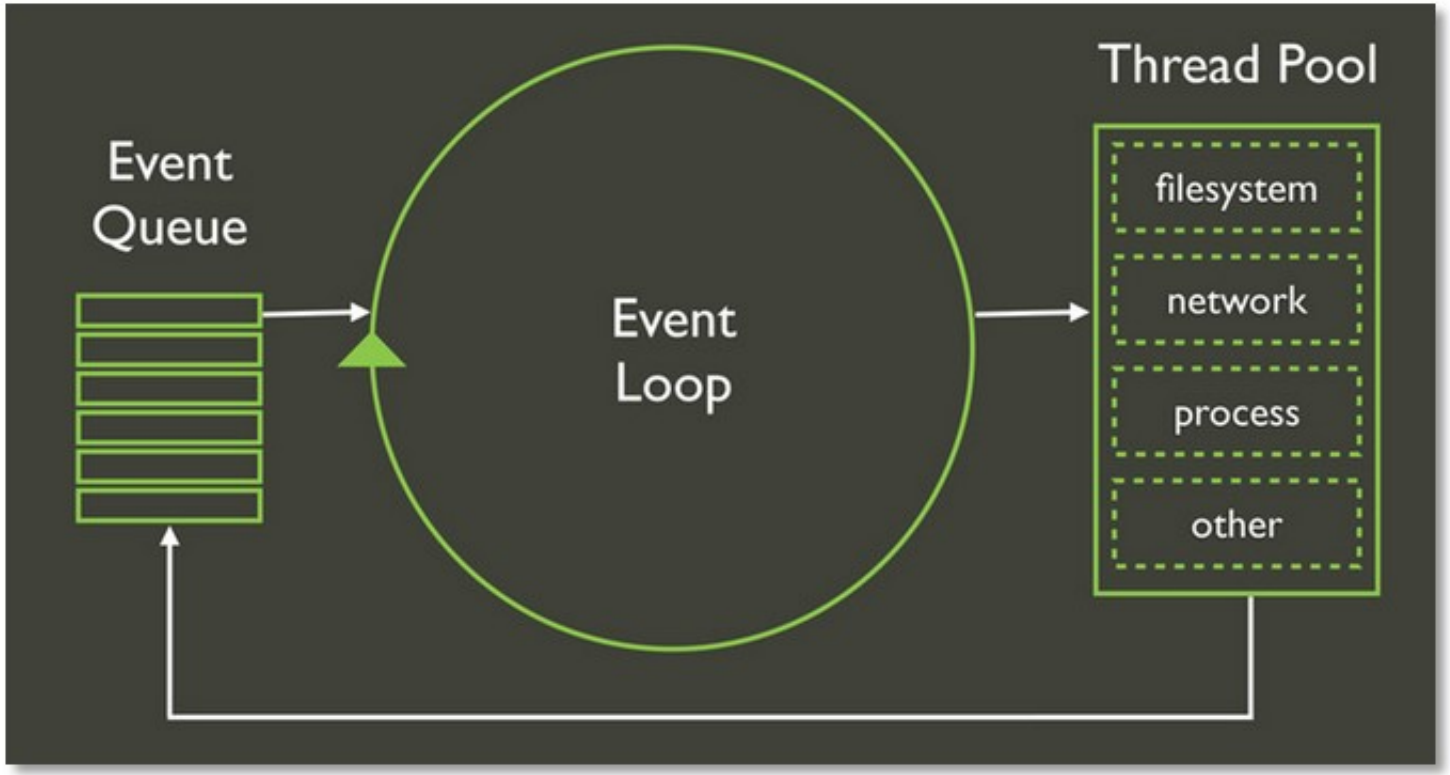
- Node.js is a platform built on Google's V8 JavaScript runtime for easily building fast, scalable network applications
- Available for download from https://nodejs.org/en/
- Open Source, cross platform
- Huge community
- Ryan Dahl, Joyent
- Developed in 2009
- Published in 2011 (Linux + Windows)
- 40% JS and 60% C++

Node.js Features

- Uses JavaScript so allowing client- and server-side development with single language
- Event-driven, lightweight, and efficient with high scalability
- Non-blocking I/O model
  - Asynchronous execution
- High performance—very fast request processing cycle
  - V8 introduced compiled JavaScript
- Supports re-useable modules
- Provides a package manager called “Node Package Manager” (npm)

The Event Loop

- Node.js is an event-driven, single-threaded, non-blocking I/O framework



Blocking Code

- Blocking code forces other JavaScript code in the Node.js process to wait
  - Until the blocking code completes
  - The event loop must wait while the blocking code runs
- The i/o methods in Node.js that are blocking have names that end with `Sync`

```
const fs = require('fs');

// blocks on this line until file is read
const data = fs.readFileSync('/file.md');
console.log(data);
moreWork(); //will run after console.log
```

Non-Blocking Code

- Non-blocking code executes asynchronously
  - All the i/o methods in Node.js provide asynchronous versions
- The asynchronous calls allow higher throughput
  - The callback function executes after the non-blocking code completes

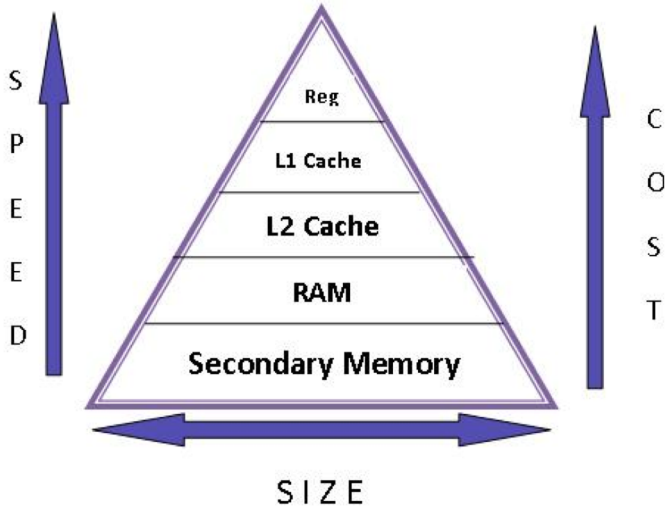
```
const fs = require('fs');
fs.readFile('/file.md', (err, data) => {
  if (err) throw err;
  console.log(data);
});
moreWork(); //will run before console.log
```

Success stories

- **Netflix**: The team decided to use Node.js to achieve lightweight, modular and fast application. As a result, the startup time of their new app has been reduced by 70%
- **Linkedin**: When compared with the previous Ruby on Rails based version, the new mobile app is up to 20 times faster
- **Walmart**: %55 of all traffic on Black Friday went to Node servers. Not a single Node server went down
- **PayPal**: Build a Node version of a Java app in half the time with fewer developers. Node version of the app doubled the number of requests/second. Request time dropped 35%
- **Groupon**: In comparison with earlier Ruby on Rails platform, web pages got faster by about 50% and are able to serve much higher traffic
- **GoDaddy**: 10x fewer servers to host our customer websites and we reduced the Time To First Byte (TTFB) considerably from ~60ms to something around ~12ms

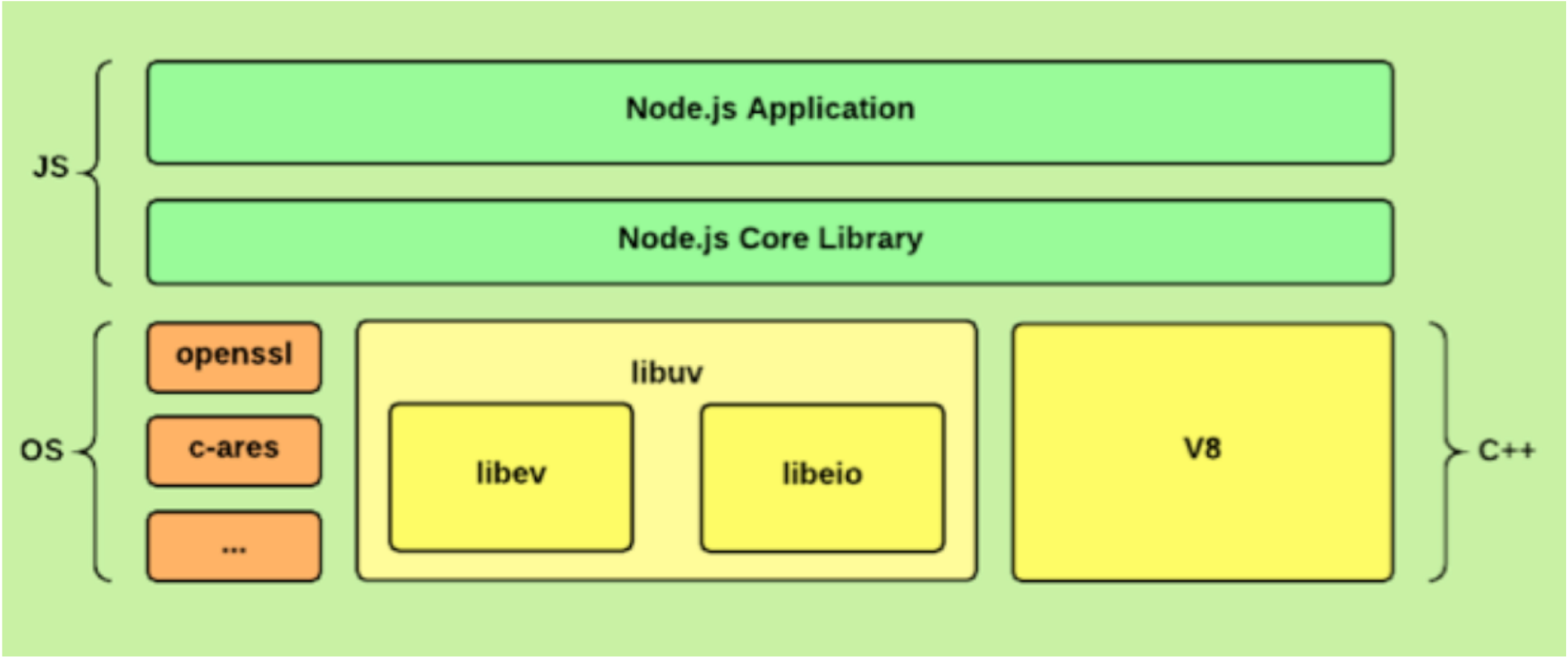
IO is expensive

Event	Latency	Scaled
1 CPU cycle	0.3 ns	1 s
Level 1 cache access	0.9 ns	3 s
Level 2 cache access	2.8 ns	9 s
Level 3 cache access	12.9 ns	43 s
Main memory access (DRAM, from CPU)	120 ns	6 min
Solid-state disk I/O (flash memory)	50–150 μs	2–6 days
Rotational disk I/O	1–10 ms	1–12 months
Internet: San Francisco to New York	40 ms	4 years
Internet: San Francisco to United Kingdom	81 ms	8 years
Internet: San Francisco to Australia	183 ms	19 years
TCP packet retransmit	1–3 s	105–317 years
OS virtualization system reboot	4 s	423 years
SCSI command time-out	30 s	3 millennia
Hardware (HW) virtualization system reboot	40 s	4 millennia
Physical system reboot	5 m	32 millennia



NodeJS Architecture

- Relies on Google's V8 runtime engine
- Node Bindings allow for server operations (IO, network)
- Libev: Event Loop
- LibEio: Async I/O
- Libuv responsible for both asynchronous I/O & event loop



When to use NodeJS?

- Highly Event driven & Heavily I/O bound
- Chat application
- Online game
- Collaboration tool
- Monitoring Dashboard
- Trader's Dashboard

When not to use NodeJS?

- Heavy CPU intensive calculations on server-side
- Concurrent Task based applications

npm

- Package manager for Node
  - Bundled and installed automatically with the environment
- Frequently Usage
  - npm install --save package_name
  - npm update
- How Does it works?
  - Read package.json
  - Installs the dependencies in the local node_modules folder
  - In global modes, it make a node module accessible to all

Install npm modules

- global modules:
  - NPM installs global packages into */<User>/local/lib/node_modules* folder.
  - Apply -g in the install command to install package globally.
- local modules:
  - Modules installed in the node_modules folder in the current project
- core modules:
  - Modules that are installed with NodeJS Installation

Getting Started with Node.js

- Steps to getting started with Node.js:
  1. Create a `.js` file such as `index.js` in a new directory
  2. Identify which Node.js “modules” are needed in the code at the top of the file
  3. Add your code and save the file
  4. Run the following command to run your code:

```
node index.js
```

Exercise 3.1: Hello World

15 **min**

- Follow the instructions in your Exercise Manual for this exercise

The global context

- In the browser this is: `window`
- In Node, this is: `global`

- `global.require()` is same as `require()`
- `global.process === process`
- `global.console === console`

What Are Node.js Modules?

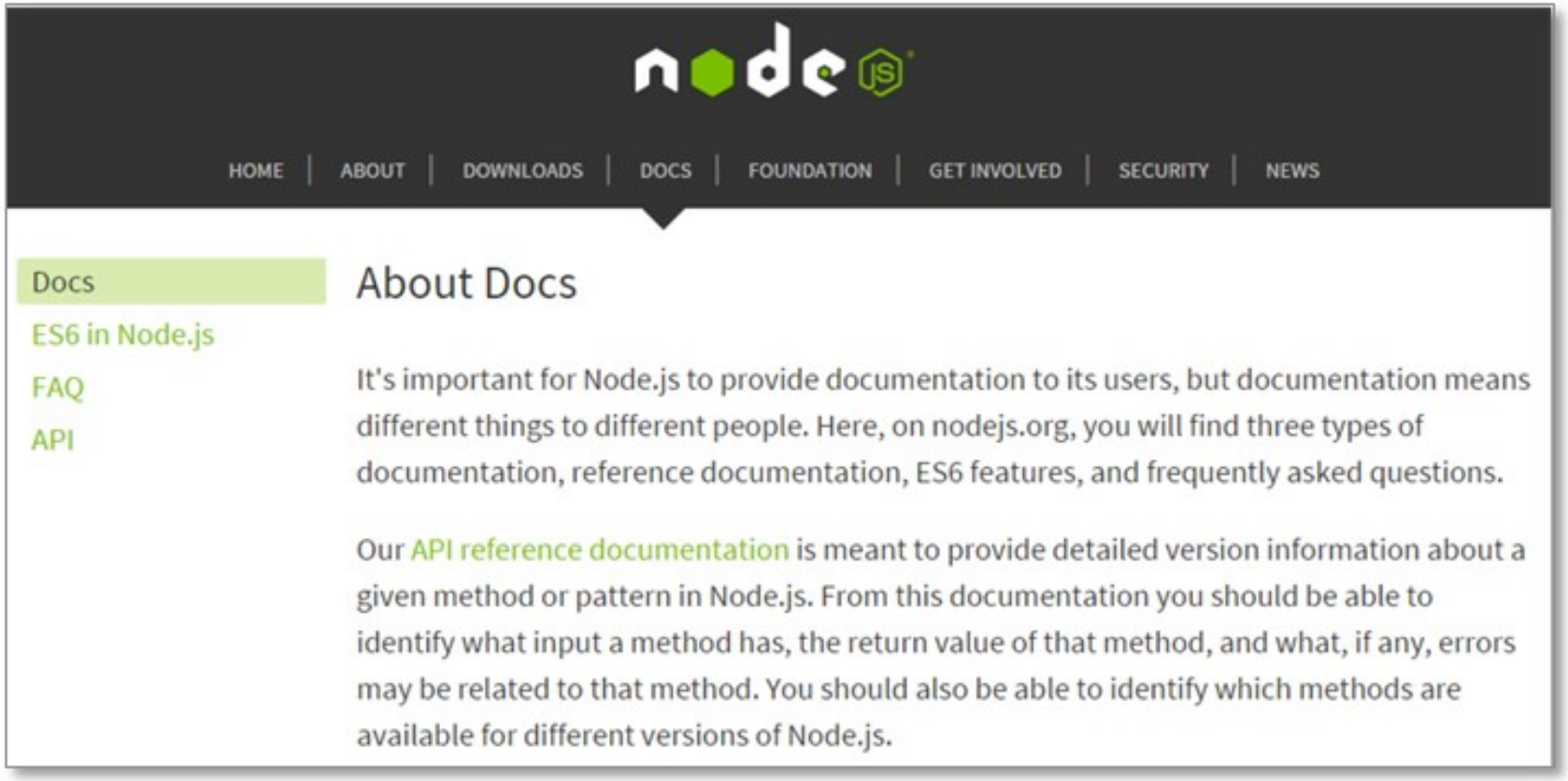
- Node.js provides a minimalist core library composed of modules
- Examples of a few built-in “core” modules:

Module	Description	Usage
fs	Provides file I/O	var fs = require('fs');
http	HTTP server and client functionality	var http = require('http');
net	Asynchronous network wrapper	var net = require('net');
path	Utilities for handling and transforming file paths	var path = require('path');
util	Various utility functions used by Node.js and custom applications	var utils = require('util');

- Custom modules can be installed using a tool called “npm” (more on this later)

Node.js Documentation

- Get documentation on Node.js and the core modules at: https://nodejs.org/api



Creating a Node Server

1. The following code creates and runs a server listening on port 8080

```
const http = require('http');

const server = http.createServer( (req, res) => {
    res.end('Hello World from the Server!');
}).listen(8080);
```

2. To launch the server, run **`node myFile.js`** at the command line
3. Navigate to http://localhost:8080 in your browser to communicate with the server

Exercise 3.2: Creating a Server

20 **min**

- Follow the instructions in your Exercise Manual for this exercise

Creating an HTTP Server to Return HTML

- Node.js can be used to create an HTTP server that returns HTML (or other) content types

```
const http = require('http');

const server = http.createServer( (req, res) => {

    res.writeHead(200, { 'Content-Type': 'text/html' });
    res.write('<h1>Hello World</h1>');
    res.end();

}).listen(8080);
```

Exercise 3.3: Returning HTML

20 **min**

- Follow the instructions in your Exercise Manual for this exercise

Node.js Modules

- Modules are re-useable/self-contained pieces of software
- Node.js provides a module system for loading application resources:
  - Core Modules – Native modules built-in to Node.js such as http, networking, file system, and more
  - File Modules – Used to load custom modules from .js files
- Packages/Modules can be loaded using **npm** – Node Package Manager

Loading Modules

- Modules are loaded by using `require()`
- Core modules are defined using a shortcut name:

```
const http = require('http');
const net = require('fs');
```

- File modules can be loaded by defining a path:

```
const parser = require('./stringParser');
```

Using a Core Node.js Module

- File System module (fs) can be used to load files

Async callback function

```
const fs = require('fs');

fs.readFile('myfile.html', (err, fileData) => {
    if (err) {
        console.log(err);
        return;
    }
    else {
        res.write(fileData);
        res.end();
    }
});
```

https://nodejs.org/api/fs.html

Exercise 3.4: Using a Core Module

20 **min**

- Follow the directions in your Exercise Manual for this exercise

Creating and Loading a Custom Module

- A custom module "exports" functionality using *module.exports* or the *exports* alias

hello.js

```
module.exports = function() {
    return 'Hello!';
}
```

index.js

```
const hello = require('./hello');


let text = hello();

console.log(text);
```

Installing Modules with npm

- npm can be used to access packages from http://npmjs.org
  - Access thousands of packaged modules
  - Store modules globally or locally in a project
  - Handles dependencies automatically

```
npm help install
npm install socket.io -g
npm install underscore
npm ls
```

Store module in global location

Store module in local node_modules folder

List local modules

Chapter Concepts

Features and First Steps

The package.json File

Basic Database Access

JSON data-mock File

Chapter Summary

The package.json File

- Node.js projects normally have a **package.json** file at their root that defines information such as:
  - Project author
  - Source control
  - Startup scripts
  - License information
  - Project dependencies
  - Project dev dependencies
  - More
- Create a **package.json** file by running **npm init**

package.json File Example

```
{
    "name": "myApp",
    "version": "0.1.0",
    "description": "My super cool node app!",
    "main": "server.js",
    "author": "John \"Guru\" Doe",
    "license": "ISC"
    "dependencies": {
        "socket.io": "^1.3.5"
    },
    "devDependencies": {
        "gulp": "^3.8.11"
    }
}
```

Installing Modules into package.json

- The npm install command can be used to install packages and update package.json:

  `--save`

  `--save-dev`

```
npm install socket.io --save
npm install gulp --save-dev
```

Install module and update dependencies property

Install module and update devDependencies property

- ***Note:***
  - devDependencies are for the development-related scripts, e.g., unit testing, packaging scripts, documentation generation, etc.
  - dependencies are required for production use, and assumed required for dev as well

Exercise 3.5: Using npm

20 **min**

- Follow the directions in your Exercise Manual for this exercise

Chapter Concepts

Features and First Steps

The package.json File

Basic Database Access

JSON data-mock File

Chapter Summary

Database and node.js

- Most database have easy to install processes (e.g., npm install, etc.), installing the necessary drivers and modules
  - E.g., for MS SQL Server its: `npm install mssql`
- Oracle and node.js
  - Oracle is an exception in the way the driver is installed
  - Fairly elaborate, involves compilation of python code and C++ code, ~ 2hours
    - Knowledge exists within Fidelity and can be obtained if required
  - See the installation instructions here:

    https://oracle.github.io/node-oracledb/INSTALL.html

Simple node-database Example

- MongoDB is a commonly used No-SQL database, particular for Big-Data scenarios
- Example: Create database called “mydb”
- Query for all customers with address: Park Lane 38

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/mydb";

MongoClient.connect(url, function(err, db) {
  if (err) throw err;
  var query = { address: "Park Lane 38" };
  db.collection("customers").find(query).toArray(
          function(err, result) {
            if (err) throw err;
            console.log(result);
            db.close();
          });
});
```

Custom Approach

- Each database (MySQL, MSSQL, Oracle, etc.) has its own approach
- Research the Internet and sufficient code examples will be available
- ***Note:*** Don't forget to run the `npm install` from within your express project or the modules won't be found!

Chapter Concepts

Features and First Steps

The package.json File

Basic Database Access

JSON data-mock File

Chapter Summary

JSON as Data File Format

- JSON-like XML can be used to store data
- When databases are not available at the time of development, mock-data source can be used to emulate a data depository
- Check out: `./data/contact.json`
- The code shown will load the entire file and exports.list will parse it to a JSON object, making it easy to access in JavaScript

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

function read_json_file() {
  var file = './data/contact.json’;
  return fs.readFileSync(file);
}

exports.list = function() {
  return JSON.parse(read_json_file());
};
```

Chapter Concepts

Features and First Steps

The package.json File

Basic Database Access

JSON data-mock File

Chapter Summary

Chapter Summary

In this chapter, we have discussed:

- How to use node.js to become your web server
- What are modules?
- How to access databases from JavaScript