# Node.js Basics

Summarized by S. Saeid Hosseini


Udemy

# What is Node.js

- Open source, cross platform runtime env for developing server-side application
- Built on the Google Chrome V8 JavaScript engine
- Event-driven, non-blocking I/O model
- Simple answer: ***"JavaScript on the Server"***
- Check it out https://nodejs.org

# What is npm

- **N**ode **P**ackage **M**anager is the world's largest Software Registry
- The registry contains over 1,000,000 code packages by 2019
- Open-source developers use npm to share software
- npm can be installed with *node.js* and is **free** to use
- All npm packages are defined in files called ***package.json***
- npm can manage and install **dependencies**
- Check it out https://www.npmjs.com/

# Getting Started

- Make sure to have ***node.js*** & ***npm*** installed
- Install an editor, e.g. ***Sublime Text***, ***VS Code***, etc.
- Note that we need a *package.json* file
- Use *npm init* to create the file
- Check out the *about* page on the node.js website
- Create a *Server.js* file and paste the code in it
- Use *npm start* or *node server* to run the code
- Check out the result in a browser

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

const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello World');
});

server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

# Simple Web Server

- We'll try to learn by doing
- Our first project will be to create a simple **static** Web Server
- We want to serve files with ***html***, ***css***, ***javascript***, ***jpeg***, ***jpg*** and ***png*** extension
- We want to generate **proper response** to every request
- All requests need to be **logged with details** in the console and a file
- We don't want to use any framework/middleware

# Simple Web Server

- Let's create a HTTP server using the ***createServer*** method of ***http*** module on port 80
- To get the requested file, we need to parse *req.url* using the ***parse*** method of ***url*** module
- In order to check if the file exists, we need to create a string with the complete address of the file requested.
  - Use the ***cwd*** method of the ***process*** module to get the address of current dir
  - Use the ***join*** method of the ***path*** module to create the complete address of the file (Why not just concatenate?)

# Simple Web Server

- Now we need to check if the file really exists on the server
- To do that, we'll use the ***lstatSync*** method of the ***fs*** module
  - It will either give you an status of the address you called upon (is it a file, directory, …) or *throw an exception* indicating that the file does **not** exist
- So we'll need a ***try/catch*** block to generate a ***404 Not Found*** response on the occasion of the second case
  - Call ***setHeader*** and ***end*** methods of the ***res*** object to create proper response

# Simple Web Server

- Now if we haven't responded yet, it means that the address actually exists
- Here there are 3 possible scenarios in which we'd like to generate a response:

    1. Request for a **file**
    2. Request for a **directory**
    3. **Neither** of the above cases

# Simple Web Server

- In the occasion of the first case, we'll need to *stream* the file
- Since we need to tell what kind of file we'll be streaming in the **HTTP** request (via a ***Content-type*** header), we need to extract the file extension and interpret that to the appropriate header value (a *mimeType key value pair* could help)
  - If the extension is not supported, we'll generate a ***415 Content Not Supported*** response
  - Else we'll create a read stream using the ***createReadStream*** method of the ***fs*** module and ***pipe*** it through the response

# Simple Web Server

- In the event of a request on a directory, we'll need to redirect it to the ***index.html*** file of that directory; This is what common Web Servers do
- We'll send a ***301 Moved Permanently*** with a ***Location*** header key set to ***/index.html***
- This would result in a second request for the ***index.html*** file of the directory

# Simple Web Server

- For the occurrence of the neither of the previous scenario, we'll send a ***500 Internal Server Error*** because we cannot process the request!

# Simple Web Server

- For logging to the terminal, use the ***log*** method of the ***console*** module
- For logging to a file, use ***appendFile*** method of the ***fs*** module
- In order to specify the exact occurrence time of the request, we can create a ***Date*** object and use ***toISOString*** method to get a proper format

# Learn Node.js by building 12 projects

By **Brad Traversy**

*Simple Web Server* project explained and summarized by


SayidHosseini

SayidHosseini