[NodeJS + Express]

Roi Yehoshua 2018

[DAN.IT]
EDUCATION

[Agenda]

- NodeJS
- NPM
- Express
- Templates (pug)
- Cookies
- Session
- Uploading files
- Cache management

[DAN.IT] EDUCATION

[NodeJS]

- An open-source, cross-platform JavaScript run-time environment that executes JavaScript code server-side
- Based on Google's V8 engine
- Provides core server functionality
- Asynchronous programming model using non-blocking I/O and asynchronous events
- Aims to optimize throughput and scalability in web applications with many input/output operations
- Node.js was originally written by Ryan Dahl in 2009
- http://nodejs.org

[DAN.IT] EDUCATION

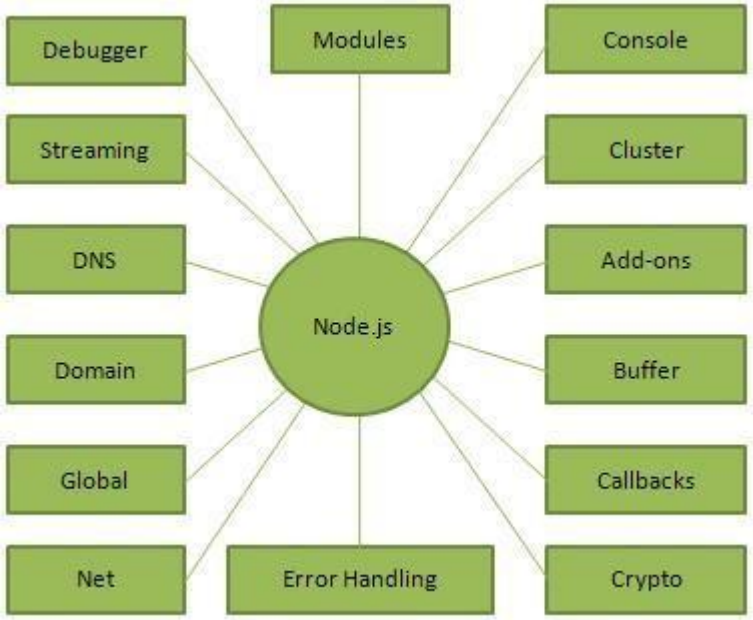
[NodeJS Releases]

Release	Code name	Release date	LTS status	Active LTS start	Maintenance start	Maintenance end
v0.10.x		2013-03-11	End-of-life	-	2015-10-01	2016-10-31
v0.12.x		2015-02-06	End-of-life	-	2016-04-01	2016-12-31
4.x	Argon	2015-09-08	Maintenance	2015-10-01	2017-04-01	2018-04-30
5.x		2015-10-29	No LTS	N/A		
6.x	Boron	2016-04-26	**Active**	2016-10-18	2018-04-30	April 2019
7.x		2016-10-25	No LTS	N/A		
8.x	Carbon[65]	2017-05-30	**Active**	2017-10-31	April 2019	December 2019
9.x		2017-10-01	No LTS	N/A		
10.x		Apr 2018	Pending	October 2018	April 2020	April 2021

[DAN.IT] EDUCATION

[NodeJS Core Functionality]

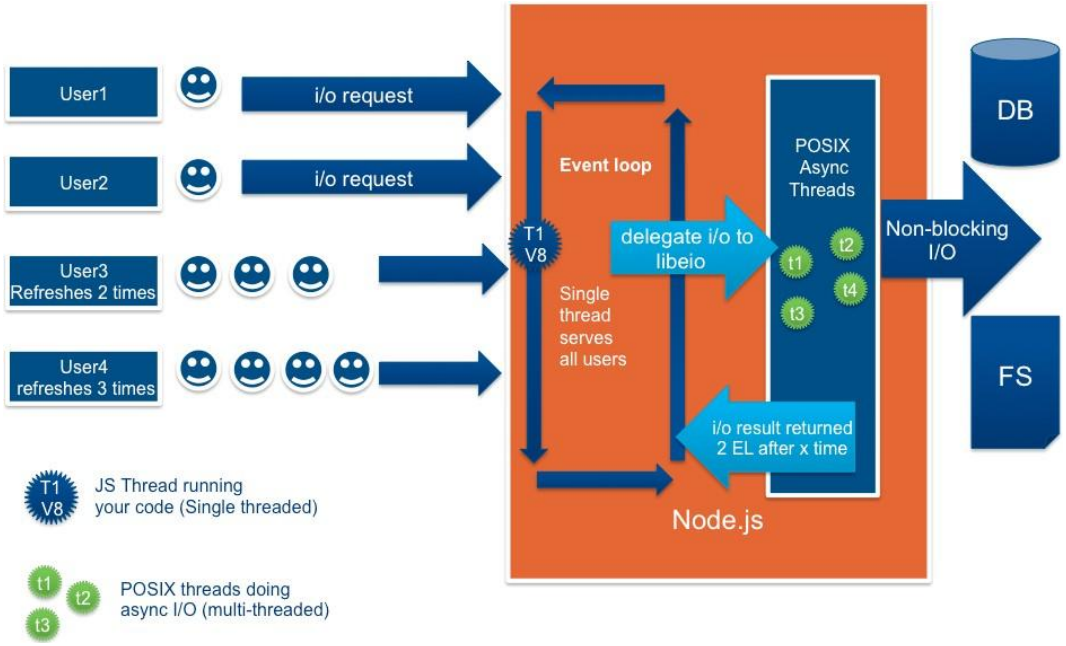
- File system I/O
- Networking (DNS, HTTP, HTTPS, TCP, TLS/SSL,UDP)
- Binary data (buffers)
- Cryptography functions
- Data streams



[DAN.IT] EDUCATION

[NodeJS Architecture]

- NodeJS is based on a **single-threaded event loop** model to handle multiple concurrent client requests
  - In contrast to other web servers, like PHP/Java/ASP.NET, in which every client request is instantiated on a new thread
  - This provides better performance and scalability under typical web loads



[DAN.IT]
EDUCATION

[NodeJS Installation]

- Download latest version from https://nodejs.org/en/download/
- Choose the appropriate type of installation according to your OS

Downloads

Latest LTS Version: **8.11.3** (includes npm 5.6.0)

Download the Node.js source code or a pre-built installer for your platform, and start developing today.

LTS Recommended For Most Users	Current Latest Features

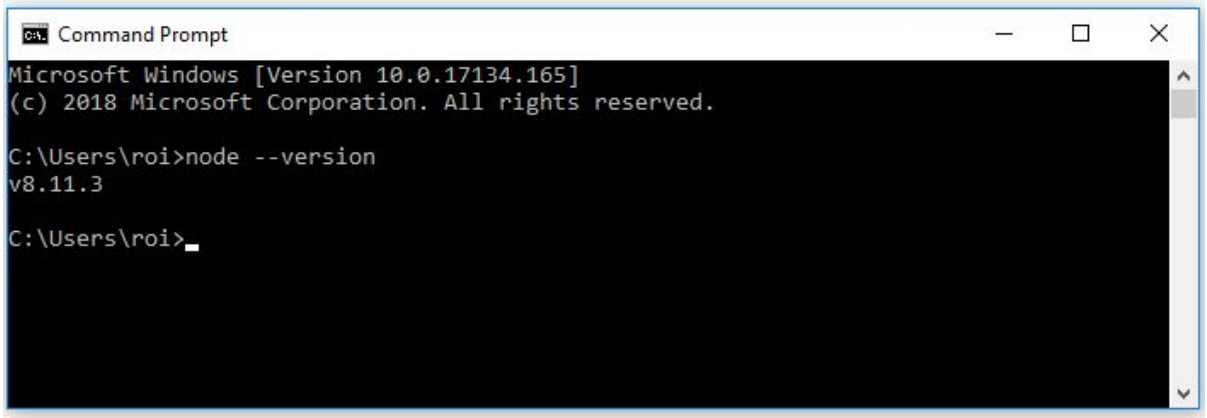
Windows Installer	macOS Installer	Source Code
node-v8.11.3-x64.msi	node-v8.11.3.pkg	node-v8.11.3.tar.gz

Windows Installer (.msi)	32-bit	64-bit	
Windows Binary (.zip)	32-bit	64-bit	
macOS Installer (.pkg)	64-bit		
macOS Binary (.tar.gz)	64-bit		
Linux Binaries (x86/x64)	32-bit	64-bit	
Linux Binaries (ARM)	ARMv6	ARMv7	ARMv8
Source Code	node-v8.11.3.tar.gz		

[DAN.IT] EDUCATION

[Verifying Installation]

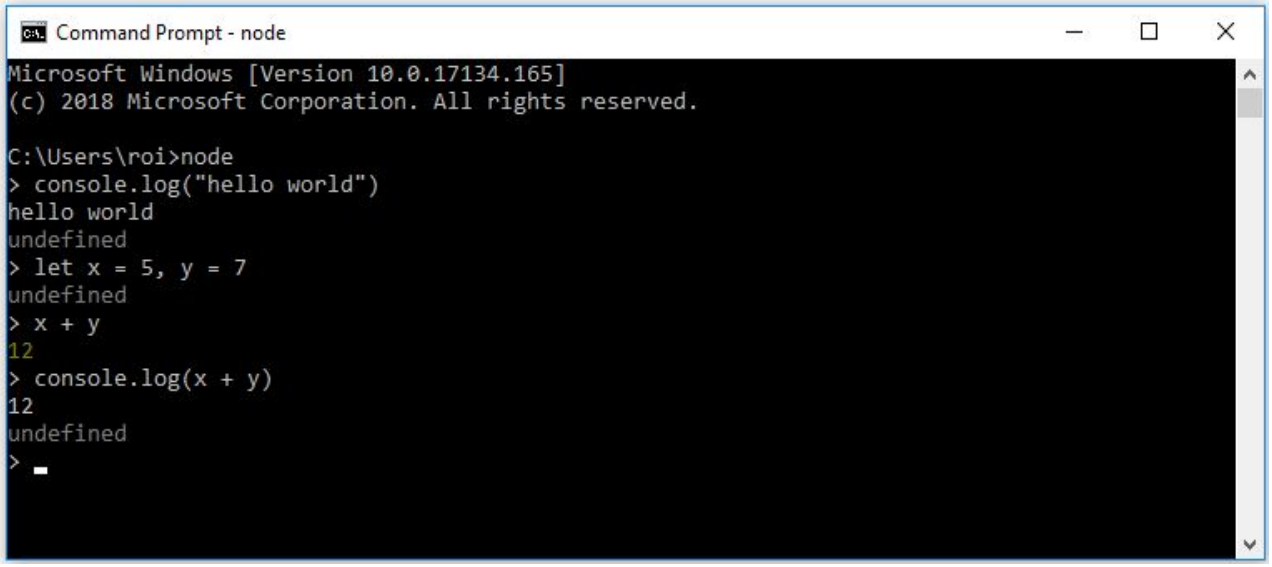
- After installing node it should be available in your PATH
- Verify that the installation was successful by running node --version from the command prompt:



[DAN.IT] EDUCATION

[Running node]

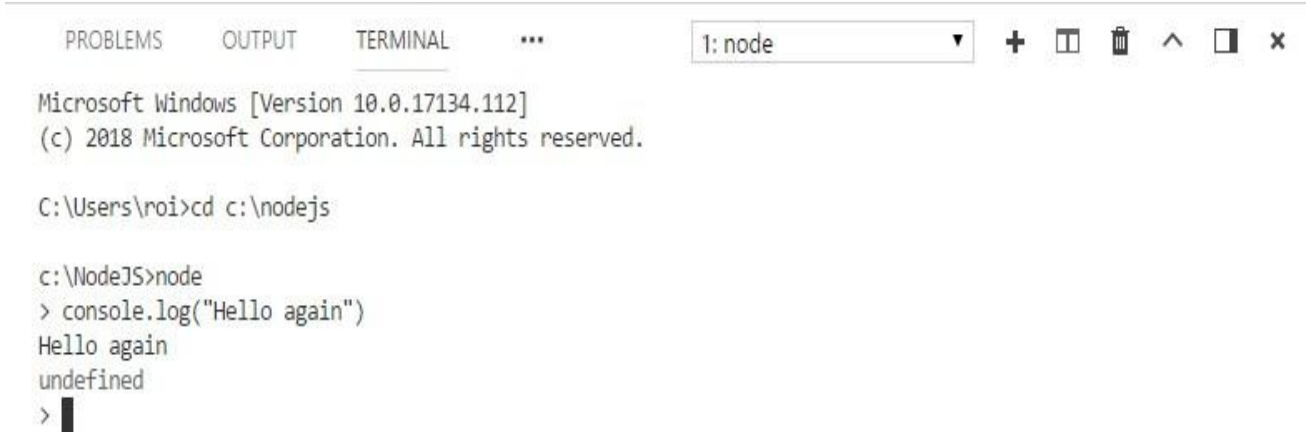
- Start a new cmd window and write ‘node’ in command line
- node.exe acts as javascript interpreter, so you can directly put some code in console



- The interpreter displays the return value of each command, and since console.log() returns undefined, you see these “undefined” messages in the output

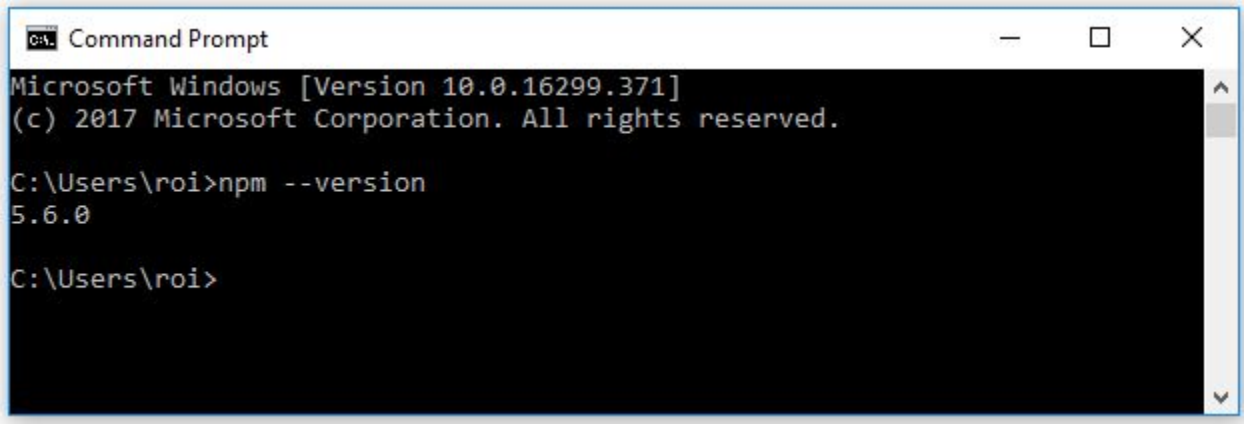
[Running node from VS Code]

- You can also run Node directly from Visual Studio Code
- Choose View -> Integrated Terminal (or Ctrl+`)
- Go to the folder where your Node server is installed (e.g., C:\NodeJS)
- Type node



[Node Package Manager (NPM)]

- NPM provides two main functionalities:
  - Online repositories for node.js packages/modules which are searchable on search.nodejs.org
  - Command line utility to install Node.js packages, do version management and dependency management of Node.js packages
- NPM comes bundled with Node.js installable
- To verify the NPM version, open console and type the following command:



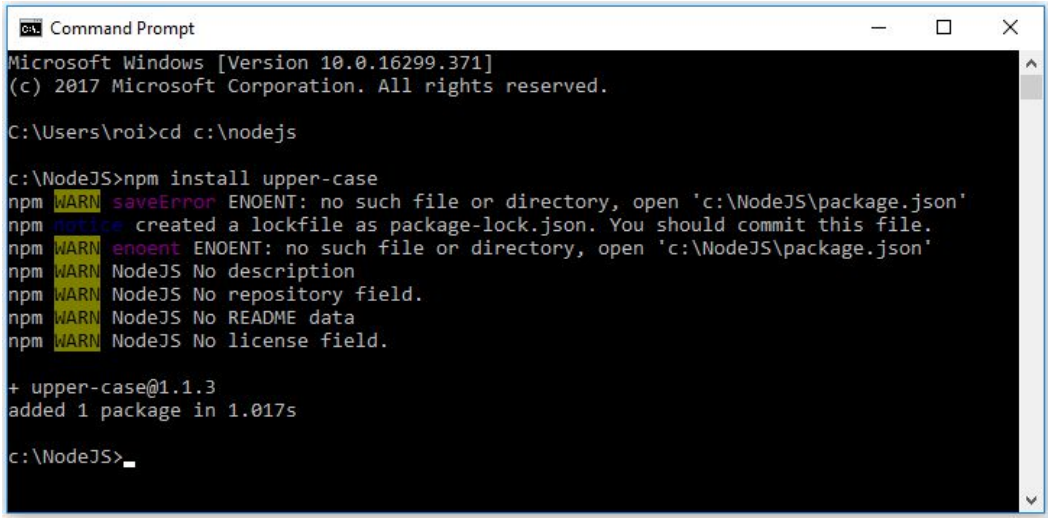
[DAN.IT] EDUCATION

[Installing Packages using NPM]

- A package in Node.js contains all the files you need for a module
- Modules are JavaScript libraries you can include in your project
- To install any Node.js package, type:

```
npm install <Package Name>
```

- For example, to install the package “upper-case” open a command line and type:



[DAN.IT] EDUCATION

[Installing Packages using NPM]

- NPM creates a folder named "node_modules“ inside the folder you are currently in, and places the new package there
- By default, npm will also install all the modules listed as dependencies of the module that you’re trying to install

View

C › Local Disk (C:) › NodeJS › node_modules › upper-case

Name	Date modified	Type	Size
LICENSE	16/12/2015 2:24 AM	File	2 KB
package.json	26/4/2018 12:58 PM	JSON File	2 KB
README.md	16/12/2015 2:25 AM	MD File	2 KB
upper-case.d.ts	16/12/2015 2:25 AM	TS File	1 KB
upper-case.js	16/12/2015 2:24 AM	JavaScript File	1 KB

[DAN.IT] EDUCATION

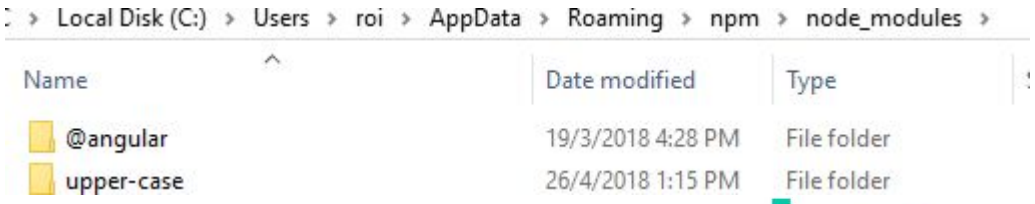
[Global vs. Local Installation]

- By default, NPM installs any dependency in local mode, i.e., in the node_modules sub folder of the folder you are currently in
- To install a package globally add the **-g** switch:

```
npm install <Package Name> -g
```

- Globally installed packages are stored in the system directory
  - In Windows: %AppData%\npm\node_modules
  - In Linux: /usr/local/lib/node_modules
- For example, to install the upper_case package globally type:

```
c:\NodeJS>npm install upper-case -g
+ upper-case@1.1.3
added 1 package in 0.853s
```



[List Packages]

- You can use **npm ls** to list down all the locally installed modules
- Or **npm ls -g** to list all the globally installed modules

Command Prompt

```
Microsoft Windows [Version 10.0.16299.371]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\roi>npm ls -g
C:\Users\roi\AppData\Roaming\npm
+-- @angular/cli@1.7.3
| +-- @angular-devkit/build-optimizer@0.3.2
| | +-- loader-utils@1.1.0 deduped
| | +-- source-map@0.5.7
| | +-- typescript@2.6.2
| | `-- webpack-sources@1.1.0 deduped
| +-- @angular-devkit/core@0.3.2
| | +-- ajv@5.5.2
| | | +-- co@4.6.0
| | | +-- fast-deep-equal@1.1.0 deduped
| | | +-- fast-json-stable-stringify@2.0.0 deduped
| | | `-- json-schema-traverse@0.3.1 deduped
| | +-- chokidar@1.7.0
| | | +-- anymatch@1.3.2
| | | | +-- micromatch@2.3.11 deduped
| | | | `-- normalize-path@2.1.1 deduped
| | | +-- async-each@1.0.1
| | | +-- UNMET OPTIONAL DEPENDENCY fsevents@^1.0.0
| | | +-- glob-parent@2.0.0
| | | | `-- is-glob@2.0.1 deduped
| | | +-- inherits@2.0.3 deduped
| | | +-- is-binary-path@1.0.1
| | | | `-- binary-extensions@1.11.0
| | | +-- is-glob@2.0.1
| | | | `-- is-extglob@1.0.0
```

[DAN.IT] EDUCATION

[NPM Commands Summary]

NPM

Command Line Interface (CLI) commands for npm

Getting Started

`npm init`
Interactively create a package.json file

`npm install`
Install packages based on package.json file in the current folder

`npm search <term>`
Search the registry for packages matching the search terms

`npm update -g <package_name>`
Updates all the packages to the latest version, respecting semver

`npm help <command>`
Show help for the specific command

`npm search <package_name>`
Search for the package

Installing Packages

`npm install <package_name>`
Install the latest version of a package

`npm install <package_name>@<version>`
Install specific version of a package

`npm install -g <package_name>`
Install a package globally, usually for command line use (On *nix requires sudo)

`npm install -S <package_name>`
Install a package and append it in the dependencies section of your package.json

`npm install -D <package_name>`
Install a package and append it in the devDependencies section of your package.json

`npm install -O <package_name>`
Install a package and append it in the optionalDependencies section of your package.json

Uninstalling Packages

`npm uninstall <package_name>`
Uninstall the latest version of a package

`npm uninstall <package_name>@<version>`
Uninstall specific version of a package

`npm uninstall -g <package_name>`
Uninstall the package globally

`npm uninstall -S <package_name>`
Uninstall the package and append it in the dependencies section of your package.json

`npm uninstall -D <package_name>`
Uninstall the package and append it in the devDependencies section of your package.json

`npm uninstall -O <package_name>`
Uninstall the package and append it in the optionalDependencies section of your package.json

Package Details

`npm docs <package_name>`
Show the docs for a package in a web browser

`npm bugs <package_name>`
Show the issues for a package in a web browser

`npm repo <package_name>`
Open package repository page in the browser

`npm ls`
Print all the versions of packages that are installed, as well as their dependencies

[DAN.IT] EDUCATION

[package.json]

- All npm packages contain a file, usually in the project root, called **package.json**
- This file is used to give information to npm that allows it to identify the project as well as handle the project's dependencies
- It must at least specify a name and version, which together form a unique identifier of your package
- Typically it also contains a list of all the packages that your project depends on

```
{
  "name" : "Todo App",
  "version"  : "1.0.0",
  "description"   : "Simple todo application.",
  "main" : "todo.js",
  "author"   : "Roi Yehoshua",  "dependencies" : {
                "express" : "~3.4.4",
              "mongoose"    : "~3.6.2"
  }
}
```

[DAN.IT]
EDUCATION

[Creating a Package]

- To create a package.json with values that you supply, run **npm init**
- This will initiate a command line questionnaire that will conclude with the creation of a package.json in the directory in which you initiated the command
- To get a default package.json, run **npm init --yes**
  - This will generate a default package.json using information extracted from the current directory
- Let's create a new package called my-first-package
  - Create a folder C:\NodeJS\my-first-package
  - Then run npm init from that folder

[DAN.IT] EDUCATION

[Creating a Package]

npm

```
Microsoft Windows [Version 10.0.17134.165]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\roi>cd c:\nodejs

c:\NodeJS>mkdir my-first-package

c:\NodeJS>cd my-first-package

c:\NodeJS\my-first-package>npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help json` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (my-first-package)
```

[Creating a Package]

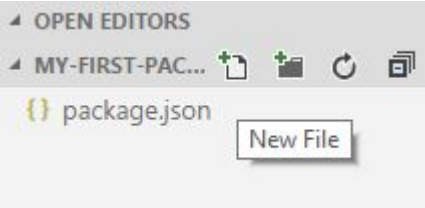
- Accepting all the defaults by pressing Enter will generate the following package.json:

```
{
  "name": "my-first-package",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}
```

[DAN.IT] EDUCATION

[Creating a Package]

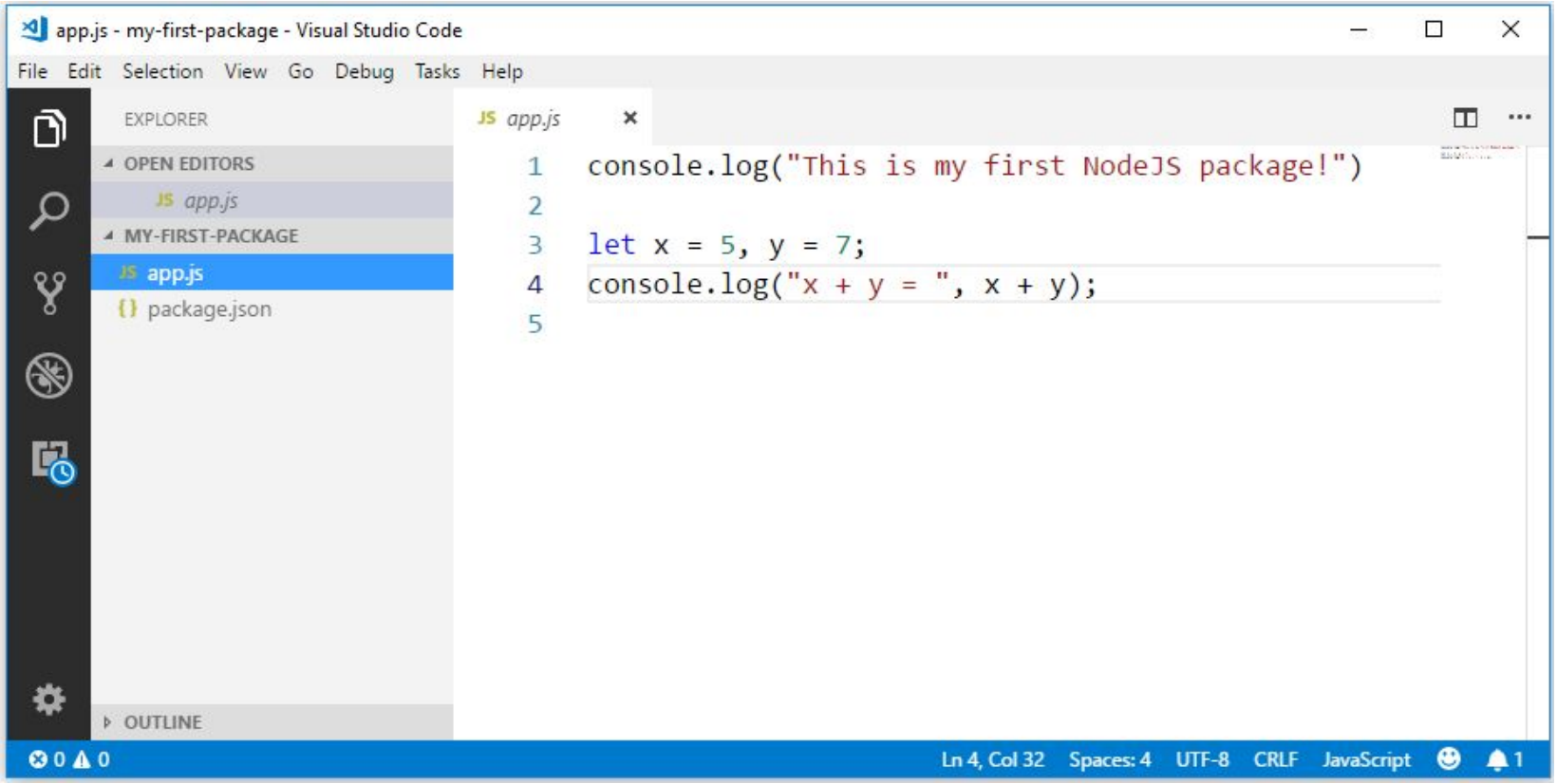
- Typically, the main entry file to your package will be called app.js
- Open the folder C:\NodeJS\my-first-package in Visual Studio Code by using the menu File > Open Folder…
- Currently your folder contains only package.json
- Create a new file app.js by clicking the New File button next to the package name
- Name your file app.js



[DAN.IT] EDUCATION

[Creating a Package]

→ Type the following code into app.js:



[DAN.IT] EDUCATION

[Running Your Server Script]

- To run node with your script, open a Console window in VS Code
  - Choose View -> Debug Console
  - Click the TERMINAL tab
- Verify that you're located at the folder of your package (C:\NodeJS\my-first-package)
- Type node app.js

[DAN.IT] EDUCATION

[Exercise (1)]

- Create a new NodeJS package named display-time
- Create a server script app.js that prints to the console the current date and time in the format dd/mm/yyyy HH:MM
- Run your script both from Visual Studio terminal and from a command prompt

[DAN.IT] EDUCATION

[NodeJS Modules]

- In the NodeJS module system, each JavaScript file is treated as a separate module
  - NodeJS modules are analogous to JavaScript libraries
- NodeJS has a set of built-in modules that you can use without any installation
- The following table shows some of the most commonly used built-in modules:

Module	Description
http	Makes Node.js act as an HTTP server
https	Makes Node.js act as an HTTPS server
fs	Handles the file system
path	Handles file paths
os	Provides information about the operation system
url	Parses url strings
stream	Handles streaming data
cluster	Splits a single Node process into multiple processes
events	Handles events

[DAN.IT] EDUCATION

[Include Modules]

- To include a module, use the require() function with the name of the module
- For example, let's another file app2.js to our package, and write the following code:

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

console.log('Platform: ' + os.platform());
console.log('Architecture: ' + os.arch());
```

- Now run this script from terminal:

```
C:\NodeJS\my-first-package>node app2.js
Platform: win32
Architecture: x64
```

[DAN.IT] EDUCATION

[Create Your Own Module]

- You can create your own modules, and easily include them in your applications
- Use the **exports** keyword to make properties and methods available outside the module file
  - **exports** is a property of an object called **module** which is included in every NodeJS file
- For example, add the file “date_formatter.js”, and add the following code to it:

```
exports.formatDate = function(date, sep) {  let day
    = date.getDate();
    let month = date.getMonth();
    let year = date.getFullYear();
    return `${day}${sep}${month}${sep}${year}`;
}

exports.formatTime = function(time, sep) {  let
    hour = time.getHours();
    let min = time.getMinutes();
    return `${hour}${sep}${min}`;
}
```

[DAN.IT] EDUCATION

[Include Your Own Module]

- Now you can include and use the module in any of your Node.js files
- Edit the file app2.js and the following code to it:

```
const dateFormatter = require('./date_formatter');

let now = new Date();
console.log('Current date: ' + dateFormatter.formatDate(now, '/'));
console.log('Current time: ' + dateFormatter.formatTime(now, ':'));
```

  - To load a module located in the same directory use ‘./’

- Now run the file:

```
C:\NodeJS\my-first-package>node app2.js
Platform: win32
Architecture: x64
Current date: 3/8/2018
Current time: 10:59
```

[DAN.IT] EDUCATION

[Exporting a Class]

- Typically, when defining a class in a module, the class is the only thing you want to export from the module
- In this case you can assign the class directly to the exports object
- For example, add a file named “circle.js” with the following code:

```
module.exports = class Circle {
    constructor(radius) {
        this.radius = radius;
    }

    getArea() {
        return Math.PI * Math.pow(this.radius, 2);
    }
}
```

  - Note that in this case you have the write the full name of the property module.exports and not only the keyword exports

[DAN.IT] EDUCATION

[Importing a Class]

→ To import the class, add the following code to app2.js:

```
const Circle = require('./circle.js');  let c1 =
new Circle(5);
console.log('Area of circle is: ' + c1.getArea());
```

```
C:\NodeJS\my-first-package>node app2.js
Platform: win32
Architecture: x64
Current date: 3/8/2018
Current time: 20:41
area of circle is: 78.53981633974483
```

[DAN.IT] EDUCATION

[Add a Dependency to a Package]

- If you need to use modules from other NodeJS packages, you first have to install them in your package's folder using **npm install**
- This will create a node_modules subfolder in your package folder containing the installed package files
- It will also add the package to the dependencies list in package.json
  - This will cause the users of your own package to install all the necessary dependencies when installing your own package
- For example, assume that our package needs to use some advanced mathematical functions from the mathjs package
- First we install the package from the command prompt:

```
C:\NodeJS\my-first-package>npm install mathjs
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN my-first-package@1.0.0 No description
npm WARN my-first-package@1.0.0 No repository field.

+ mathjs@5.0.4
added 9 packages in 3.473s
```

[Add a Dependency to a Package]

- package.json has been modified to reflect the new dependency:

```
{
  "name": "my-first-package",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",  "dependencies": {
    "mathjs": "^5.0.4"
  }
}
```

- a caret symbol matches the most recent major release, i.e., ^5.0.4 matches any 5.x.x release
- a tilde symbol matches the most recent minor release, i.e., ~5.0.4 matches any 5.0.x release

[DAN.IT] EDUCATION

[Use the New Dependency]

→ In app2.js we can now use the mathjs module:

```
const math = require('mathjs');

dx = math.derivative('x^2 + x', 'x');
console.log(dx.toString());
```

```
C:\NodeJS\my-first-package>node app2.js
Platform: win32
Architecture: x64
Current date: 3/8/2018
Current time: 21:7
Area of circle is: 78.53981633974483
2 * x + 1
```

[DAN.IT] EDUCATION

[Exercise (2)]

- Create a new NPM package called students_[your id]
- Add the module you've created in the previous exercise in the new package
- Test the package locally
- Publish your package
- Verify the installation of the published package

[DAN.IT] EDUCATION

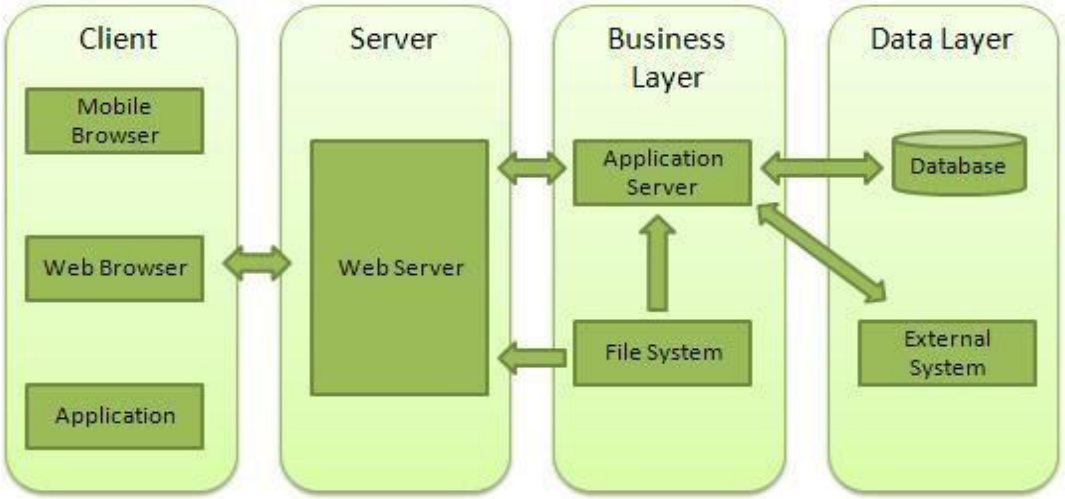
[Exercise (2)]

- Create a package named students
- Add a student.js module that exports a class named Student with the properties: id, name, age, and grades, and the following methods:
  - constructor(id, name, age)
  - addGrade(grade) - adds a grade to the student's grades list
  - computeGradesAverage() - returns the student's grades average
    - Use the package simple-statistics for computing the average
- Create another package named test_students
- Add app.js to test_students and import the student module from the students package
  - Hint: use require('../students/student') to import the module
- Create a new Student object, add a few grades and print the student's grades average
- Run app.js in node

[DAN.IT] EDUCATION

[Web Servers]

- A **Web Server** (or HTTP server) is a software application which handles HTTP requests sent by HTTP clients, like web browsers, and returns web pages/resources in response
- Most of the web servers support server-side scripts, which retrieve data from a database or perform some complex logic and sends a result to the HTTP client



[Creating a Web Server with NodeJS]

- The **http** module can create an HTTP server that listens to server ports and gives a response back to the client
- Create a new package named my-first-server, and add app.js to it
- Use the **createServer()** method to create an HTTP server:

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

http.createServer(function (req, res) {
    res.write('Hello world!'); // write a response to the client
    res.end(); // end the response
}).listen(8080); // the server listens on port 8080
```

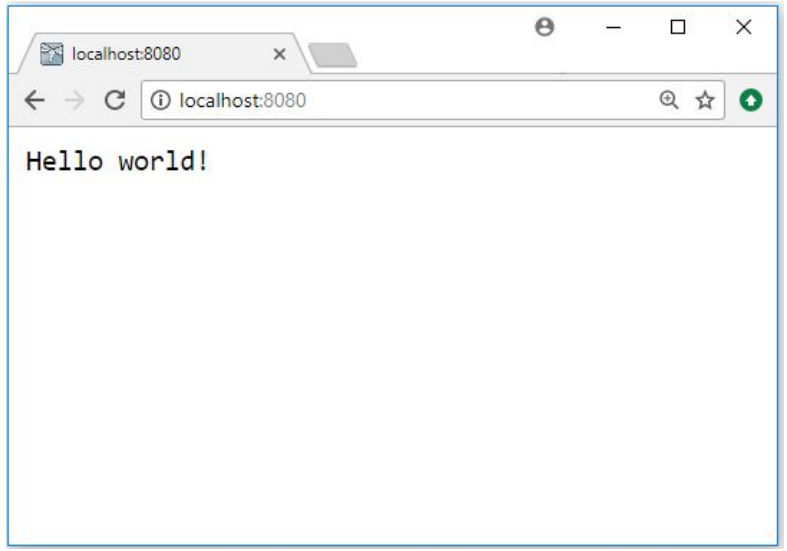
  - The method http.createServer() receives a function, that is invoked when a client sends a request to the server on the specified port
  - This function receives two parameters:
    - req – represents the request object, containing parameters sent from the client
    - res – allows you to write a response back to the client

[Creating a Web Server with NodeJS]

→ Run app.js in node:

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL                    1: node

Microsoft Windows [Version 10.0.17134.165]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\NodeJS\my-first-server>node app.js
```

→ Open http://localhost:8080 in any browser to see the result



[DAN.IT]
EDUCATION

[Exercise (3)]

- Create a NodeJS server that listens on port 9000
- Whenever it receives an HTTP request from a client, it should send back the current time on the server in the format HH:MM:SS
- Test the server using a browser

[DAN.IT] EDUCATION

[Express]

- **Express** is a minimal and flexible MVC framework for Node.js that provides a thin layer of fundamental web application features
- It is an open source framework developed and maintained by the Node.js foundation
- Many popular web frameworks are based on express
- Main features of Express:
  - Has convenient functions for parsing HTTP arguments and headers
  - Routing – can easily build RESTful APIs with Express
  - Has support for many templating languages like Jade and EJS, that reduce the amount of HTML code you have to write
  - Has support for NoSQL databases out of the box
  - Has a flexible, modular middleware pattern, where special middleware modules/functions are used to process different requests
- https://expressjs.com/

[DAN.IT] EDUCATION

[Installing Express]

- Express is installed like any other NPM package using npm install
- Create a Node package named first-express-app
- Call npm init to create a package.json file
- Now install express inside your package:

```
C:\NodeJS\first-express-app>npm install express
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN first-express-app@1.0.0 No description
npm WARN first-express-app@1.0.0 No repository field.

+ express@4.16.3
added 50 packages in 2.263s
```

[DAN.IT] EDUCATION

[Basic Express App]

- Create a new file called **app.js** and type the following code:

```
const express = require('express');  const app = express();

app.get('/', (req, res) => {  res.send("Hello world!");
});

app.listen(3000);
```

  - The **express()** function is the top-level function exported by the express module, which creates an express application
  - The app starts a server and listens on port 3000 for connections
  - The app responds with “Hello World!” for requests to the root URL (/)
  - For every other path, it will respond with a **404 Not Found**
  - The req (request) and res (response) are the exact same objects that Node provides

[DAN.IT] EDUCATION

[nodemon]

- nodemon is a utility that will monitor for any changes in your source and automatically restart your server
- Just use **nodemon** instead of node to run your code, and now your process will automatically restart when your code changes
- To install, run from the terminal:

```
npm install -g nodemon
```

```
C:\NodeJS\first-express-app>npm install -g nodemon
C:\Users\roi\AppData\Roaming\npm\nodemon -> C:\Users\roi\AppData\Roaming\npm\node_modules\nodemon\bin\
nodemon.js

> nodemon@1.18.3 postinstall C:\Users\roi\AppData\Roaming\npm\node_modules\nodemon
> node bin/postinstall || exit 0

Love nodemon? You can now support the project via the open collective:
 > https://opencollective.com/nodemon/donate

npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.4 (node_modules\nodemon\node_modules\fsev
ents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.4: wanted {"os":"d
arwin","arch":"any"} (current: {"os":"win32","arch":"x64"})

+ nodemon@1.18.3
added 232 packages in 13.704s
```

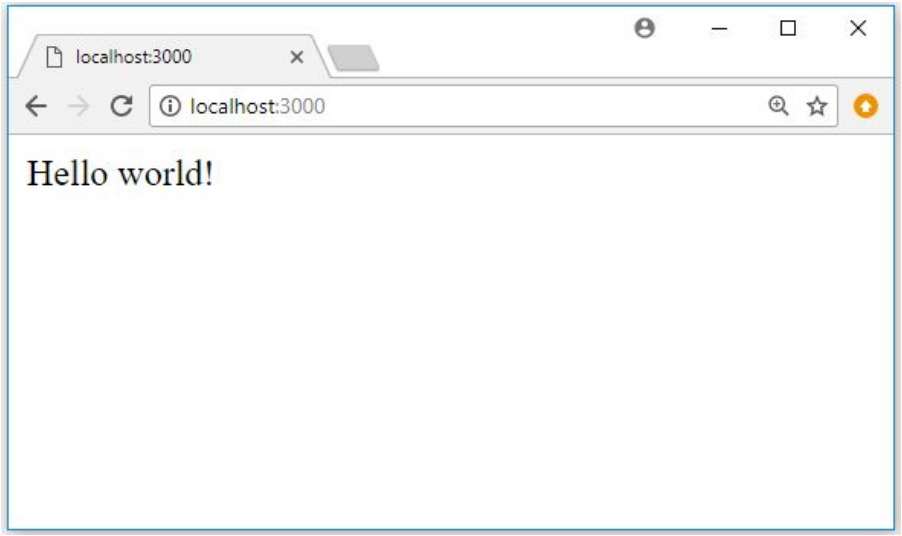
[DAN.IT] EDUCATION

[Basic Express App]

- Run the app using nodemon:

```
C:\NodeJS\first-express-app>nodemon app.js
[nodemon] 1.18.3
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: *.*
[nodemon] starting `node app.js`
```

- Then, load http://localhost:3000/ in a browser to see the output



[DAN.IT]
EDUCATION

[Basic Routing]

- Routing refers to determining how the server responds to a client request to a specific URI (or path) and a specific HTTP request method
- Each route can have one or more handler functions, which are executed when the route is matched
- Route definition takes the following structure:

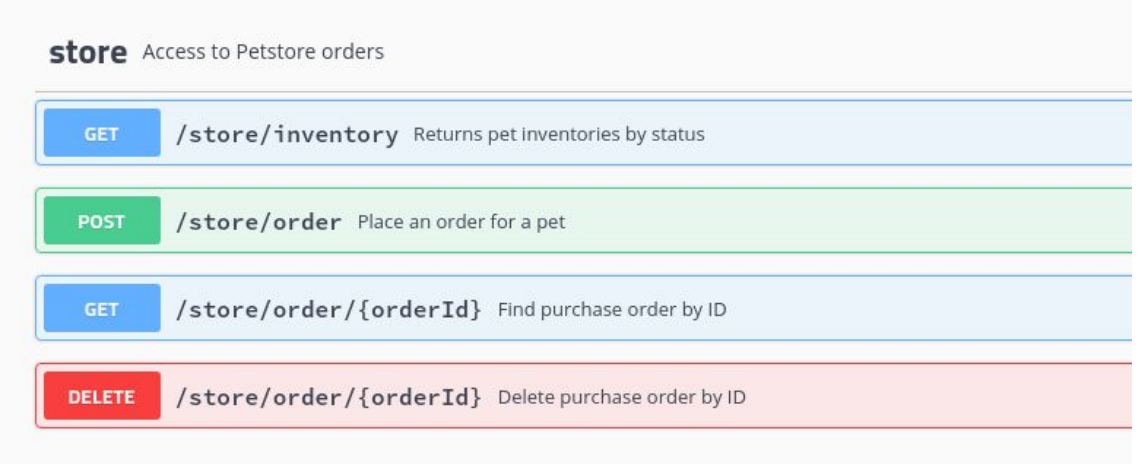
```
app.method(path, handler)
```

  - app is an instance of express application
  - *method* is an HTTP request method, in lowercase (e.g., ‘get’, ‘post’)
  - *path* is a path on the server
  - *handler* is the function executed when the route is matched

[DAN.IT] EDUCATION

[HTTP Methods]

Method	Description
GET	The GET method requests a representation of the specified resource. Requests using GET should only retrieve data.
POST	The POST method requests that the server accept the data enclosed in the request as a new object/entity of the resource identified by the URI
PUT	The PUT method replaces all current representations of the target resource with the request payload
DELETE	The DELETE method deletes the specified resource



[DAN.IT] EDUCATION

[Basic Routing]

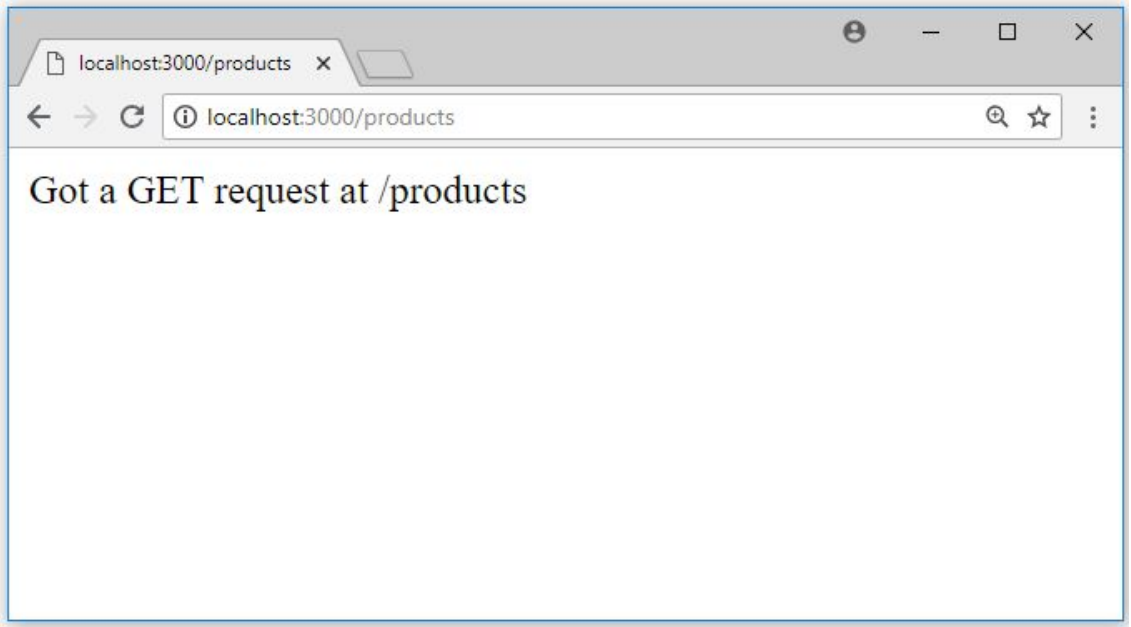
→ The following examples illustrate defining simple routes:

```
const express = require('express');
const app = express();

app.get('/', (req, res) => {  res.send("Hello world!");
});

app.get('/products', (req, res) => {  res.send("Got a
    GET request at /products");
});
app.post('/products', (req, res) => {  res.send("Got a
    POST request at /products");
});
app.put('/products', (req, res) => {  res.send("Got a
    PUT request at /products");
});
app.delete('/products', (req, res) => {  res.send("Got
    a DELETE request at /products");
});
app.listen(3000);
```

[DAN.IT] EDUCATION

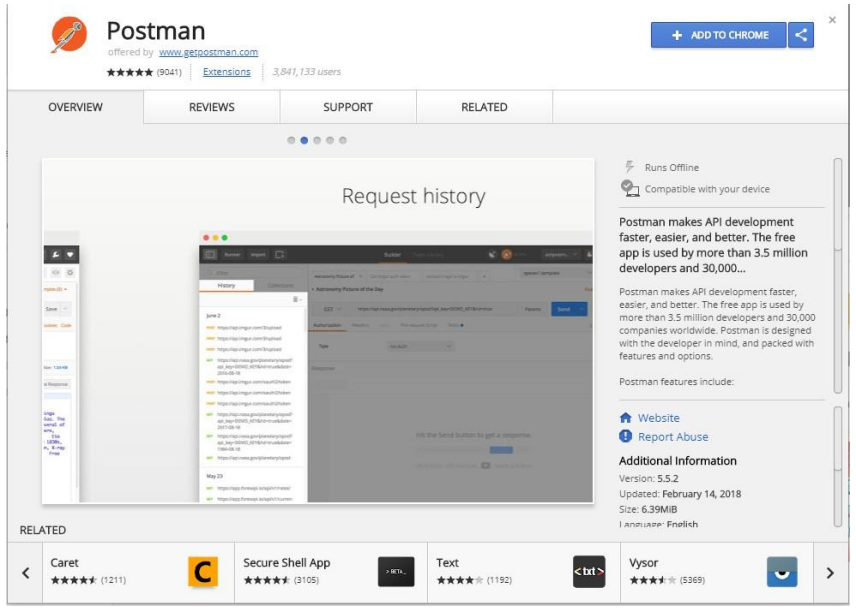
[Basic Routing]

- You can only test the GET methods in the browser
- For example, enter the URL http://localhost:3000/products to test the GET method of products



[PostMan]

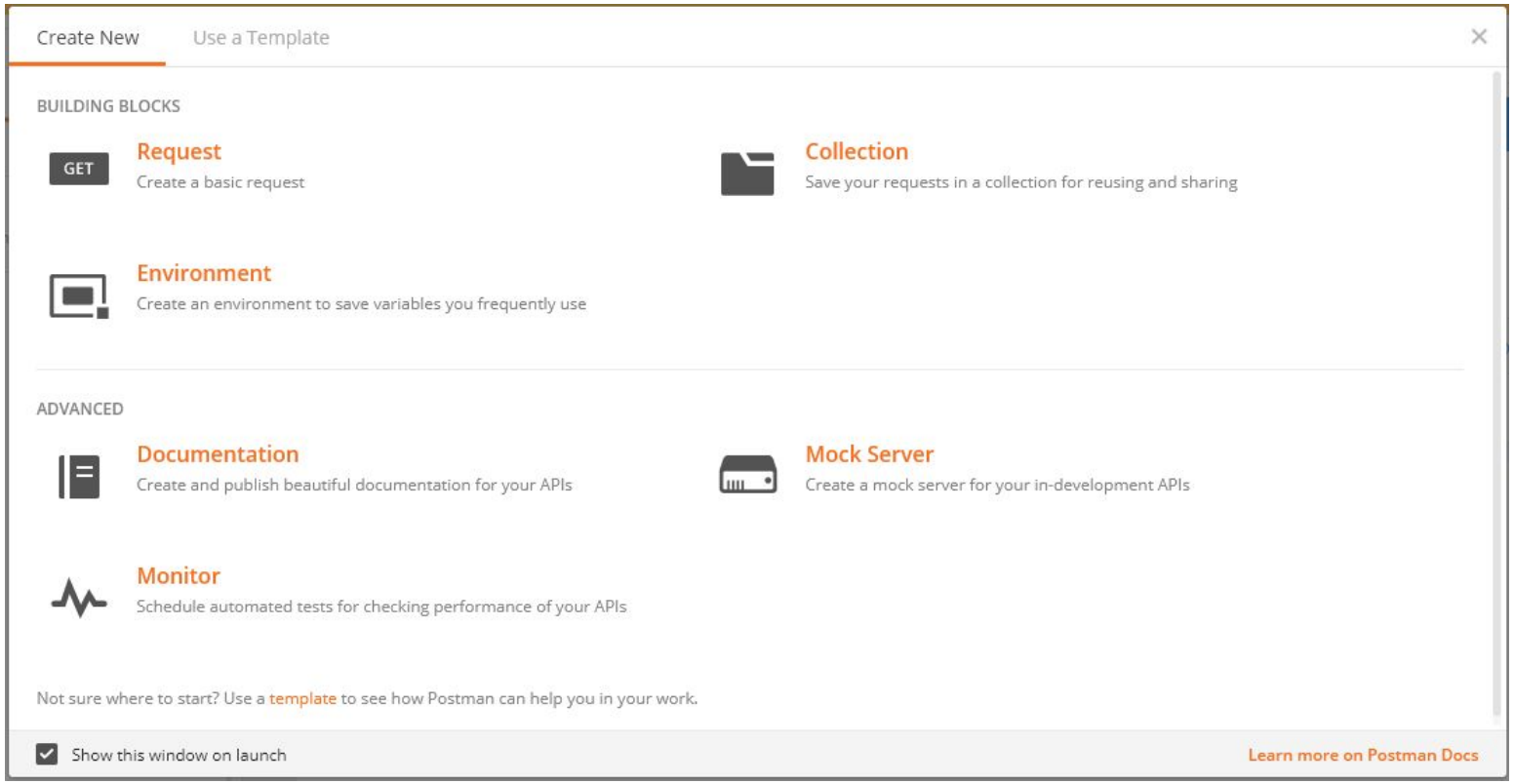
- PostMan is a web REST client that allows you to enter and monitor HTTP requests of different types and examine the responses
- Can be installed as an extension to Chrome
  - Search “PostMan – Chrome Web Store” in google, and click the first link
  - Then click the Add To Chrome button



[DAN.IT] EDUCATION

[PostMan]

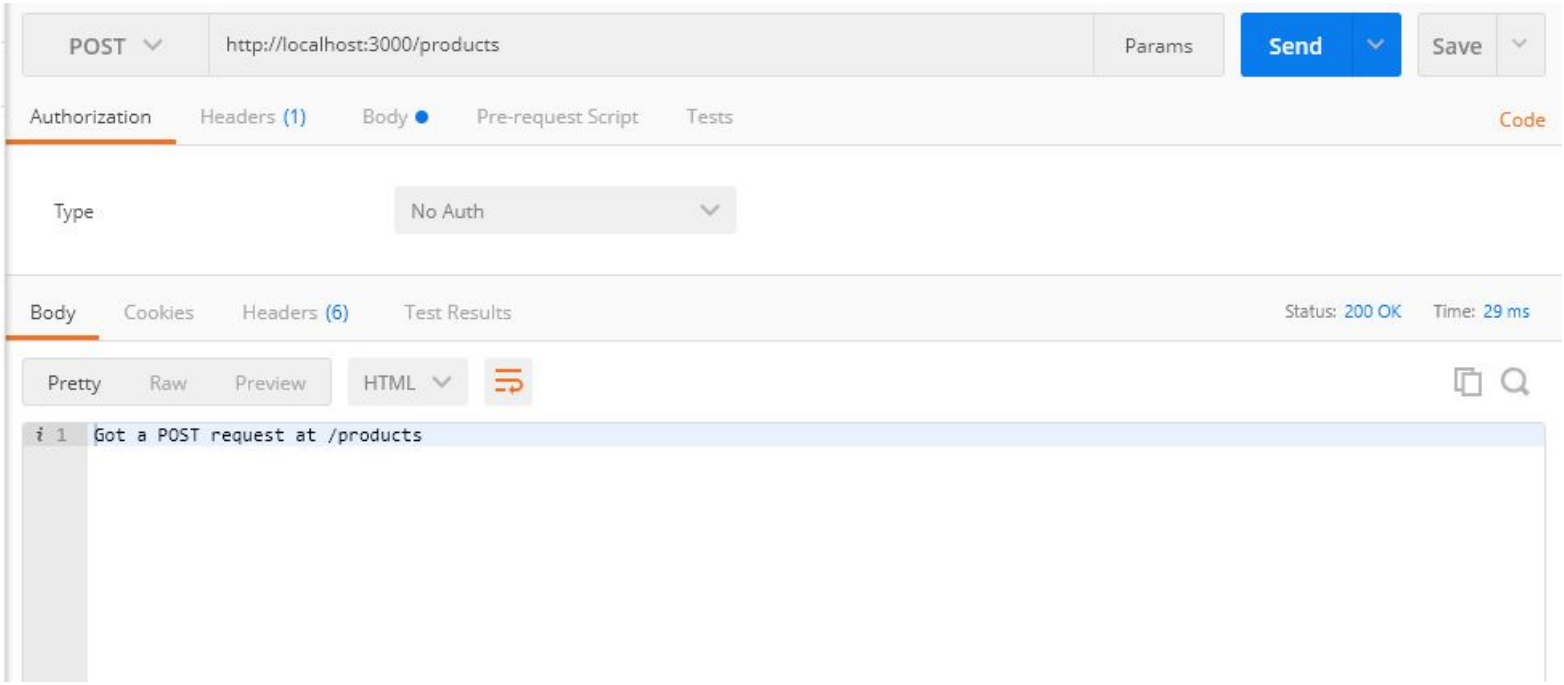
→ Create a new HTTP request



[DAN.IT] EDUCATION

[PostMan]

- Choose the HTTP method from the drop-down list
- Enter the URL of the server
- Click Send



[Route Paths]

- Route paths, in combination with a request method, define the endpoints at which requests can be made
- Route paths can be strings, string patterns, or regular expressions
- The characters $, ?, +, *, and () are subsets of their regular expression counterparts
  - The hyphen (-) and the dot (.) are interpreted literally as part of the path
- If you need to use one of the special characters in a path string, enclose it escaped within ([and])
  - For example, the path string for requests at /data/$book, would be /data/([\$])book

[DAN.IT] EDUCATION

[Route Paths Examples]

- The following route path will match acd and abcd:

```
app.get('/ab?cd', (req, res) => {
    res.send('ab?cd')
});
```

- The following route path will match abcd, abbcd, abbbcd, and so on:

```
app.get('/ab+cd', (req, res) => {
    res.send('ab+cd')
});
```

- The following route path will match abcd, abxcd, abRANDOMcd, ab123cd, and so on:

```
app.get('/ab*cd', (req, res) => {
    res.send('ab*cd')
});
```

- This following route path will match anything with an “a” in it:

```
app.get('/a/', (req, res) => {  res.send('/a/')
});
```

[DAN.IT] EDUCATION

[Route Parameters]

- Route parameters are named URL segments that are used to capture the values specified at their position in the URL
- To define routes with route parameters, simply specify the route parameters in the path of the route, preceded by a colon (:)

```
app.get('/users/:userId', (req, res) => {
    res.send("userId: " + req.params.userId);
});
```

  - The captured values are populated in the **req.params** object, with the name of the route parameter specified in the path as their respective keys
- To send a parameter from the client insert its value at the proper place in the URL:

```
Request URL: http://localhost:3000/users/34
req.params: { "userId": "34" }
```

[DAN.IT] EDUCATION

[Route Parameters]

→ Example for sending more than one parameter in the URL:

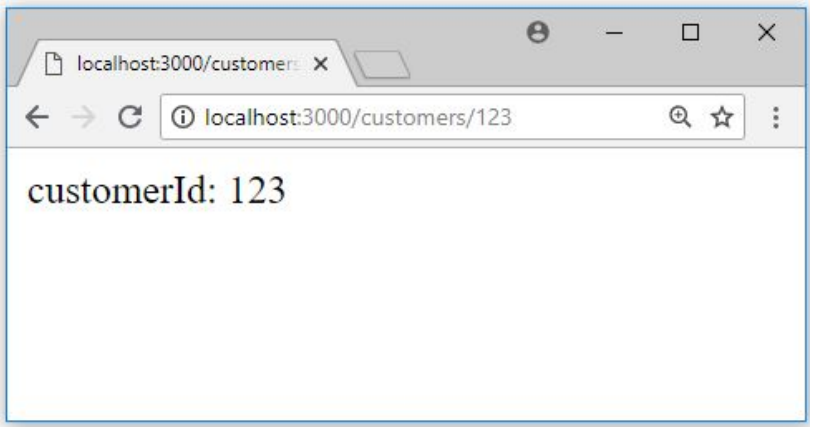
```
app.get('/users/:userId/books/:bookId', (req, res) => {
    res.send(`userId: ${req.params.userId}, bookId: ${req.params.bookId}`);
});
```

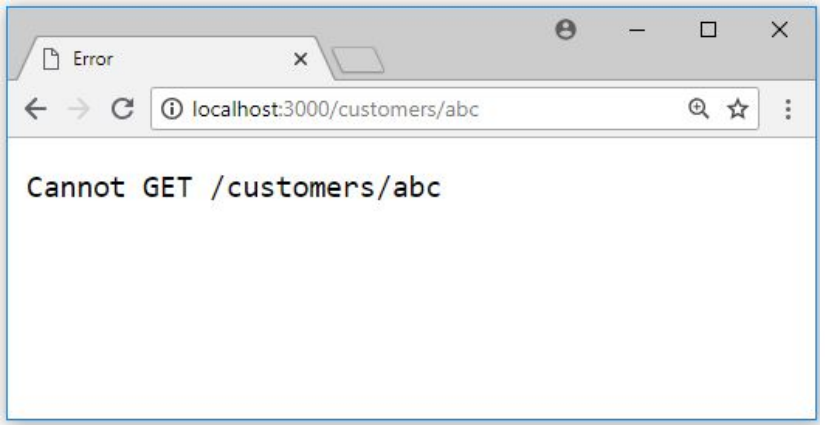
[Route Parameters]

- To have more control over the exact string that can be matched by a route parameter, you can append a regular expression in parentheses (())
- For example, the following route will only match requests that have a numeric id

```
app.get('/customers/:customerId(\\d+)', (req, res) => {
    res.send("customerId: " + req.params.customerId);
});
```

  - Because the regular expression is part of a literal string, be sure to escape any \ characters with an additional backslash, for example \\d+.





[Query String Parameters]

- Query strings are not part of the route path
- They can be appended to any URL path
- The property **req.query** is an object containing a property for each query string parameter in the route

```
app.get('/users', (req, res) => {
     res.send(`<h2>Welcome ${req.query.firstname} ${req.query.lastname}</h2>`);
});
```

[Query Params vs. Route/Path Params]

- Route params are typically used to identify a specific resource or resources, while query params are used to sort/filter those resources
- Route params are part of the URL, thus they must be specified, while query params are optional
- For example, suppose you are implementing API endpoints for an entity called Car
- You could structure your endpoints like this:

  GET /cars

  GET /cars/:id

  POST /cars

  PUT /cars/:id
  DELETE /cars/:id

- Now if you want to add the capability to filter cars by color, you could add a query parameter o your GET /cars request like this: GET /cars?color=blue

[DAN.IT] EDUCATION

[Exercise (4)]

- Create a web server that handles a repository of products
- Each product should have an id, name, and price
- The server should support the following operations:
  - Get all products – return a string with all the product names separated by a comma
  - Get product by id – get all product details (id, name and price)
  - Get product by name – get all product details (use query string param)
  - Add a new product – URL should have 3 parameters, e.g. /products/add/1/apple/2.5
  - Delete a product by id
- Test all the server methods by using PostMan

[DAN.IT] EDUCATION

[Exercise (4)]

→ Example:

POST http://localhost:3000/products/1/apple/2.5 Params Send Save

Authorization Headers (1) Body Pre-request Script Tests Code

Type No Auth

Body Cookies Headers (6) Test Results Status: 200 OK Time: 60 ms

Pretty Raw Preview HTML

```
1 Product successfully added
```

GET http://localhost:3000/products/1 Params Send Save

Authorization Headers (1) Body Pre-request Script Tests Code

Type No Auth

Body Cookies Headers (6) Test Results Status: 200 OK Time: 29 ms

Pretty Raw Preview HTML

```
1 Id: 1, Name: apple, Price: 2.5
```

[DAN.IT] EDUCATION

[Response Methods]

- The methods on the response object (res) in the following table can send a response to the client, and terminate the request-response cycle
  - If none of these methods are called, the client request will be left hanging

Method	Description
res.download()	Prompt a file to be downloaded.
res.end()	End the response process.
res.json()	Send a JSON response.
res.jsonp()	Send a JSON response with JSONP support.
res.redirect()	Redirect a request.
res.render()	Render a view template.
res.send()	Send a response of various types.
res.sendFile()	Send a file as an octet stream.
res.sendStatus()	Set the response status code and send its string representation as the response body.

[DAN.IT] EDUCATION

[res.send()]

- **res.send(body)** Sends the HTTP response
- The body parameter can be a string, an object, an array or a Buffer object
- The method automatically assigns the Content-Length response header field (unless previously defined)
- When the parameter is a string, the method sets the Content-Type to “text/html”:

```
res.send('<p>some html</p>');
```

- When the parameter is an array or object, Express responds with the JSON representation:

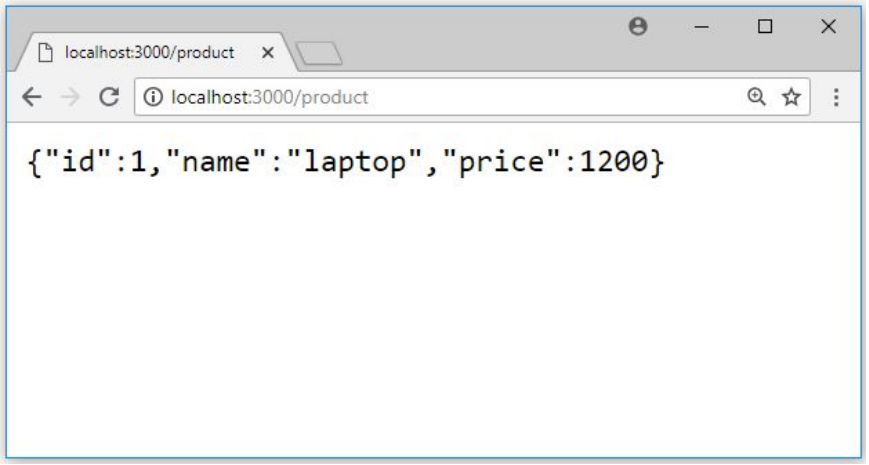
```
res.send({ user: 'tobi' });
res.send([1,2,3]);
```

[DAN.IT] EDUCATION

[res.json()]

- **res.json(body)** sends a JSON response
- This method converts its parameter to a JSON string using JSON.stringify()
- The parameter can be any JSON type, including object, array, string, Boolean, or number
- Example:

```
app.get('/product', (req, res) => {
    let product = {  id: 1,
        name: 'laptop',  price: 1200
    }

    res.json(product);
});
```

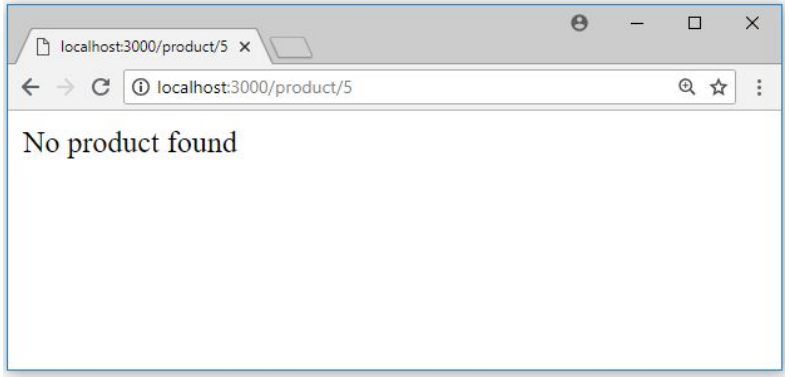


[DAN.IT]
EDUCATION

[res.status()]

- Sets the HTTP status for the response
- For example, if the product id was not found on the server, we can set the status code to HTTP 404 (Not Found):

```
let products = [
    { id: 1, name: 'laptop', price: 1200 },
    { id: 2, name: 'chair', price: 200 },
    { id: 3, name: 'printer', price: 250 }
];

app.get('/product/:id', (req, res) => {  let productId =
    req.params.id;

    let product = products.find(p => p.id == productId);
    if (!product) {
        return res.status(404).send("No product found");
    }
    else {
        res.json(product);
    }
});
```



[HTTP Status Codes]

1XX Informational	
100	Continue
101	Switching Protocols
102	Processing

2XX Success	
200	OK
201	Created
202	Accepted
203	Non-authoritative Information
204	No Content
205	Reset Content
206	Partial Content
207	Multi-Status
208	Already Reported
226	IM Used

3XX Redirectional	
300	Multiple Choices
301	Moved Permanently
302	Found
303	See Other
304	Not Modified
305	Use Proxy
307	Temporary Redirect
308	Permanent Redirect

4XX Client Error	
400	Bad Request
401	Unauthorized
402	Payment Required
403	Forbidden
404	Not Found
405	Method Not Allowed
406	Not Acceptable
407	Proxy Authentication Required
408	Request Timeout

4XX Client Error Continued	
409	Conflict
410	Gone
411	Length Required
412	Precondition Failed
413	Payload Too Large
414	Request-URI Too Long
415	Unsupported Media Type
416	Requested Range Not Satisfiable
417	Expectation Failed
418	I'm a teapot
421	Misdirected Request
422	Unprocessable Entity
423	Locked
424	Failed Dependency
426	Upgrade Required
428	Precondition Required
429	Too Many Requests
431	Request Header Fields Too Large
444	Connection Closed Without Response
451	Unavailable For Legal Reasons
499	Client Closed Request

5XX Server Error	
500	Internal Server Error
501	Not Implemented
502	Bad Gateway
503	Service Unavailable
504	Gateway Timeout
505	HTTP Version Not Supported
506	Variant Also Negotiates
507	Insufficient Storage
508	Loop Detected
510	Not Extended
511	Network Authentication Required
599	Network Connect Timeout Error

HTTP STATUS CODES

When a browser requests a service from a web server, an error may occur.
This is a list of HTTP status messages that might be returned.

[DAN.IT] EDUCATION

[res.sendStatus()]

- **res.sendStatus(statusCode)** sets the response HTTP status code to statusCode and sends its string representation as the response body

```
res.sendStatus(200); // equivalent to res.status(200).send('OK')  res.sendStatus(403);
// equivalent to res.status(403).send('Forbidden')  res.sendStatus(404); // equivalent
to res.status(404).send('Not Found')  res.sendStatus(500); // equivalent to
res.status(500).send('Internal Server Error')
```

[DAN.IT] EDUCATION

[res.sendFile()]

- **res.redirect([status,] path)** redirects to the URL derived from the specified path, with the specified HTTP status code
  - If not specified, status defaults to 302 “Found”

[DAN.IT] EDUCATION

[res.redirect()]

- **res.redirect([status,] path)** redirects to the URL derived from the specified path, with the specified HTTP status code
  - If not specified, status defaults to 302 “Found”
- Redirects can be relative to the root of the host name
  - For example, if the application is on http://example.com/admin/post/new, the following would redirect to the URL http://example.com/admin:

```
res.redirect('/admin');
```

- Redirects can be relative to the current URL
  - For example, from http://example.com/blog/admin/ (notice the trailing slash), the following would redirect to the URL http://example.com/blog/admin/post/new

```
res.redirect('post/new');
```

- Redirects can be a fully-qualified URL for redirecting to a different site:

```
res.redirect('http://google.com');
```

[DAN.IT] EDUCATION

[Exercise (5)]

- Change the products server from previous exercise to return JSON instead of strings
- The server should support the following operations:
  - Get all products – return a JSON with all the products
  - Get product by id – return a JSON with the product's details
  - Get product by name – return a JSON with the product's details
  - Add a new product – unchanged
  - Delete a product by id – unchanged
- In case of an error (e.g., product id not found) send the appropriate HTTP status code
- Test the methods by using PostMan

[DAN.IT] EDUCATION

[Middleware]

- An Express application is essentially a series of middleware function calls
- **Middleware** functions are functions that have access to the request object (req), the response object (res), and the next middleware function in the application's request-response cycle (next)
- Middleware functions can perform the following tasks:
  - Execute any code
  - Make changes to the request and the response objects
  - End the request-response cycle
  - Call the next middleware function in the stack
- If the current middleware function doesn't end the request-response cycle, it must call next() to pass control to the next middleware function
  - Otherwise, the request will be left hanging

[DAN.IT] EDUCATION

[Middleware Function]

```
var express = require('express');
var app = express();

app.get('/', function(req, res, next) {
  next();
})

app.listen(3000);
```

HTTP method for which the middleware function applies.

Path (route) for which the middleware function applies.

The middleware function.

Callback argument to the middleware function, called "next" by convention.

HTTP response argument to the middleware function, called "res" by convention.

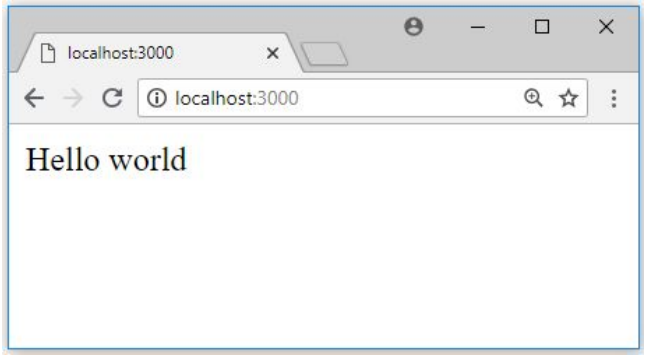
HTTP request argument to the middleware function, called "req" by convention.

[DAN.IT] EDUCATION

[Application-Level Middleware]

- Application-level middlewares are bound to the **app** object by using the app.use() and app.METHOD() functions, where METHOD is the HTTP method of the request that the middleware function handles (such as GET, PUT, or POST)
- The following example shows a middleware function that is executed every time the app receives a request:

```
const express = require('express');
const app = express()

let myLogger = function(req, res, next) {
    console.log('Logging');
    next();
}

app.use(myLogger);
app.get('/', function(req, res) {
    res.send('Hello world');
});
app.listen(3000);
```

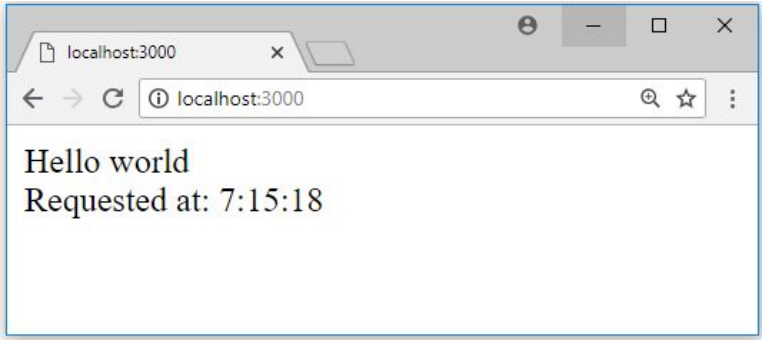



```
C:\NodeJS\middlewares>nodemon app.js
[nodemon] 1.18.3
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: *.*
[nodemon] starting `node app.js`
Logging
```

[Application-Level Middleware]

- Next, we'll create a middleware function that adds a property called requestTime to the request object:

```
let requestTime = function (req, res, next) {  time = new
    Date();
    req.requestTime = time.getHours() + ':' +
time.getMinutes() + ':' + time.getSeconds();
    next();
}
app.use(requestTime)

app.get('/', function(req, res) {
    let responseText = 'Hello world<br/>';
    responseText += 'Requested at: ' + req.requestTime;
    res.send(responseText);
});
```



[Configurable Middleware]

- If you need your middleware to be configurable, export a function which accepts an options object or other parameters, which, then returns the middleware implementation based on the input parameters:

```
// my-middleware.js
module.exports = function(options) {  return function(req, res, next) {
        // Implement the middleware function based on the options object
        next();
    }
}
```

- The middleware can now be used as shown below:

```
let mw = require('./my-middleware.js');  app.use(mw({ option1: '1',
option2: '2' }));
```

[DAN.IT] EDUCATION

[Error Handling Middleware]

- Define error-handling middleware functions in the same way as other middleware functions, except with four arguments instead of three:

```
app.use((err, req, res, next) => {
    console.error(err.stack)
    res.status(500).send('Something broke!')
});
```

- You should define the error-handling middleware last, after other app.use() and routes calls

[Built-in Middleware]

- Express has the following built-in middleware functions:
- express.static serves static assets such as HTML files, images, and so on
- express.json parses incoming requests with JSON payloads
  - NOTE: Available with Express 4.16.0+
- express.urlencoded parses incoming requests with URL-encoded payloads
  - NOTE: Available with Express 4.16.0+

[DAN.IT] EDUCATION

[Static Files]

- To serve static files such as images, CSS files, and JavaScript files, use the **express.static()** built-in middleware function:

```
express.static(root)
```

  - The root argument specifies the root directory from which to serve static assets

- For example, use the following code to serve static files from a folder named public:

```
app.use(express.static('public'));
```

- Now, you can load the files that are in the public directory, e.g.
  - http://localhost:3000/welcome.html
  - http://localhost:3000/js/app.js
  - http://localhost:3000/images/kitten.jpg

[DAN.IT] EDUCATION

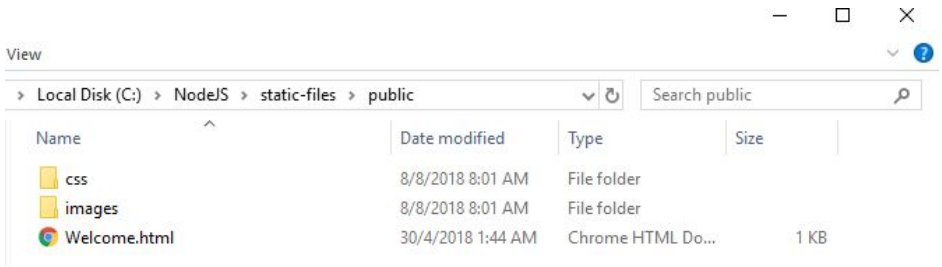
[Static Files - Using Absolute Path]

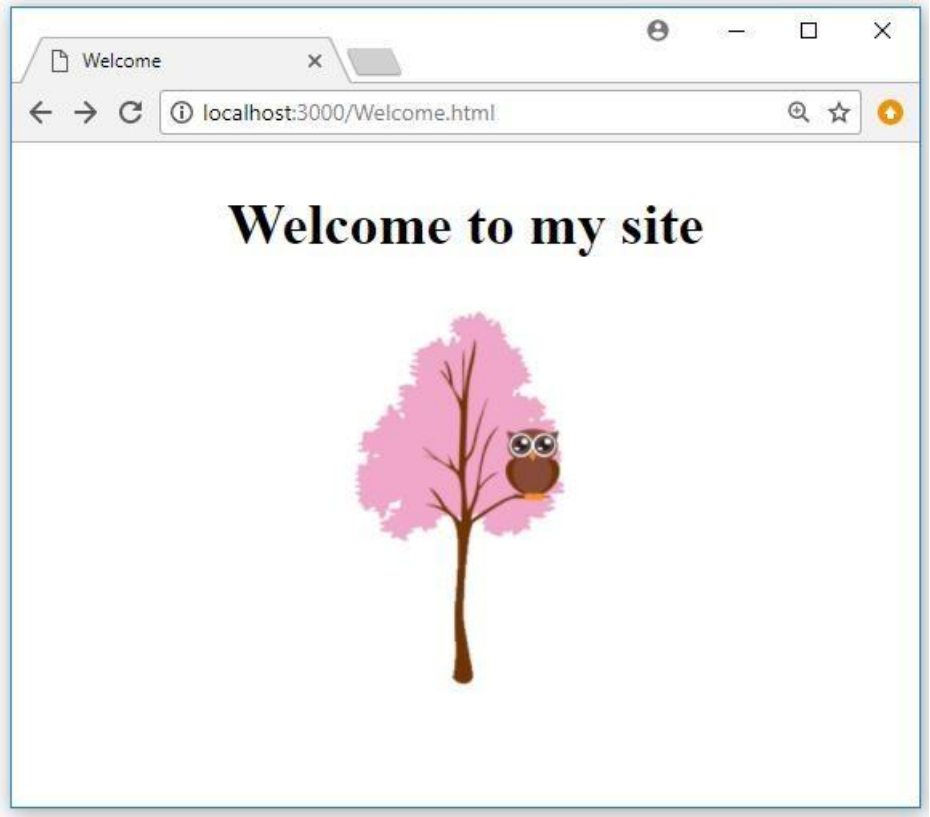
- The path that you provide to the express.static() is relative to the directory from where you launch your node process
- If you run the express app from another directory, it’s safer to use the absolute path of the directory that you want to serve:

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

app.use(express.static(path.join(_dirname, 'public')));
```

[DAN.IT] EDUCATION

[Static Files]





[Virtual Path Prefix]

- To create a virtual path prefix (where the path doesn't actually exist in the file system) for files that are served by the express.static() function, specify a mount path for the static directory:

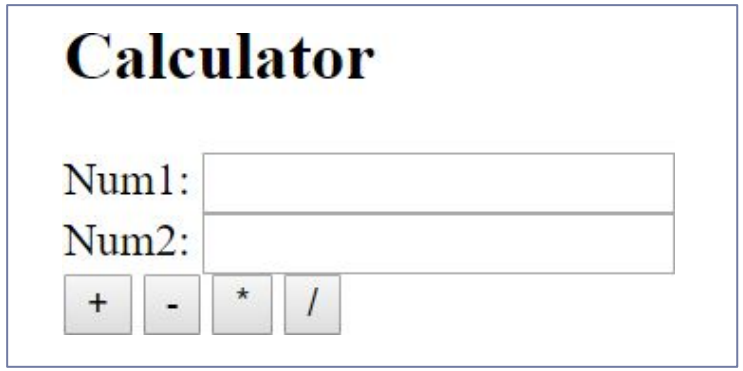
```
app.use('/static', express.static('public'));
```

- Now, you can load the files that are in the public directory from the /static path prefix, e.g.
  - http://localhost:3000/static/welcome.html
  - http://localhost:3000/static/js/app.js
  - http://localhost:3000/static/images/kitten.jpg

[DAN.IT] EDUCATION

[Exercise (6)]

- Build an HTML page that displays a simple calculator, such as the following:



- The calculator should submit the exercise to your web server, passing the following params:
  - num1 – the first operand
  - num2 – the second operand
  - op – the operator
- The server should send back an HTML with the result of the computation

[express.json()]

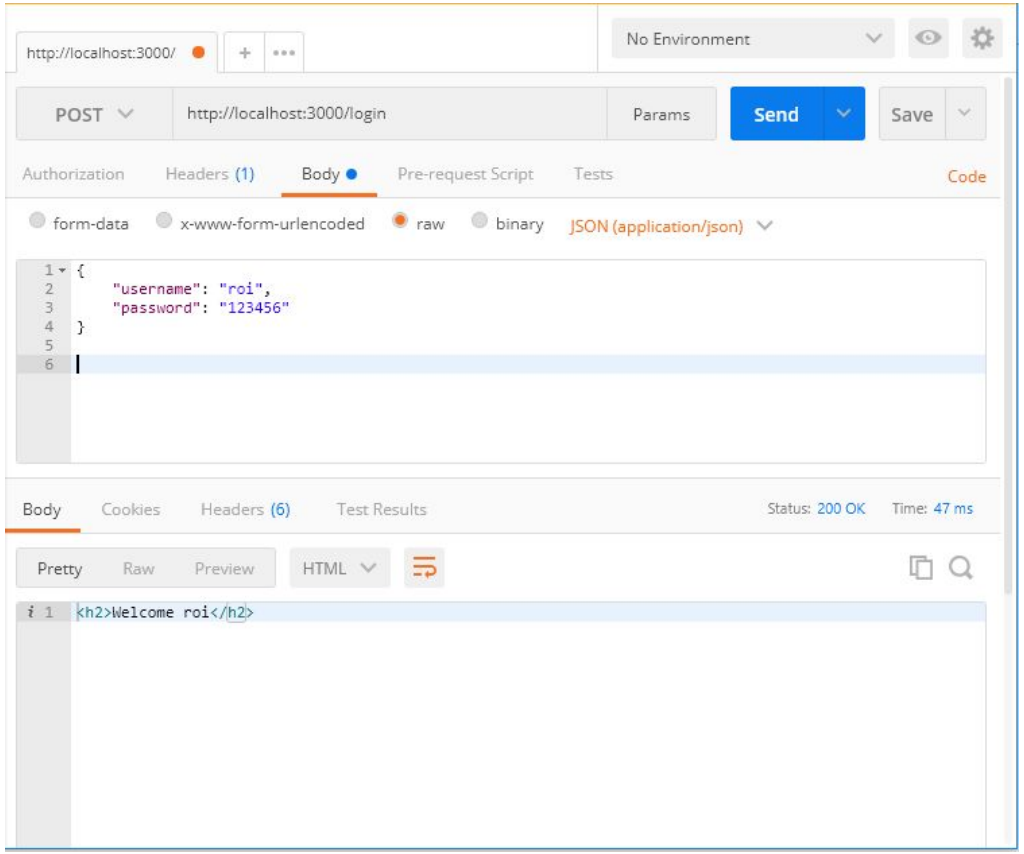
- This is a built-in middleware function, which parses requests with JSON payloads
- A new body object containing the parsed data is populated on the request object after the middleware (i.e., req.body)
  - or an empty object ({}) if there was no body to parse, or an error occurred

```
const express = require('express');  const app =
express();

app.use(express.json());

// POST /login gets JSON bodies  app.post('/login',
(req, res) => {
    if (!req.body)
        return res.sendStatus(400);  let user =
    req.body;
    res.send(`<h2>Welcome ${user.username}</h2>`);
});

app.listen(3000);
```

[DAN.IT] EDUCATION

[Sending JSON in PostMan]

- Under Body choose raw option to send URL encoded form data
- Define all the parameters inside a JSON object



[express.urlencoded()]

- This is a middleware function, which parses requests with urlencoded payloads
- Urlencoded payloads use the same encoding as the one used in query string parameters (key-value pairs)
- When you submit a HTML form with method="POST", the Content-Type of the request is application/x-www-form-urlencoded by default, and it looks like this:

```
POST /some-path HTTP/1.1
Content-Type: application/x-www-form-urlencoded

foo=bar&name=John
```

  - Whereas a request with a JSON payload is typically submitted via AJAX call, and looks like this:

```
POST /some-path HTTP/1.1
Content-Type: application/json

{ "foo" : "bar", "name" : "John" }
```

[DAN.IT] EDUCATION

[express.urlencoded()]

- Example for using the urlenocded body parser:

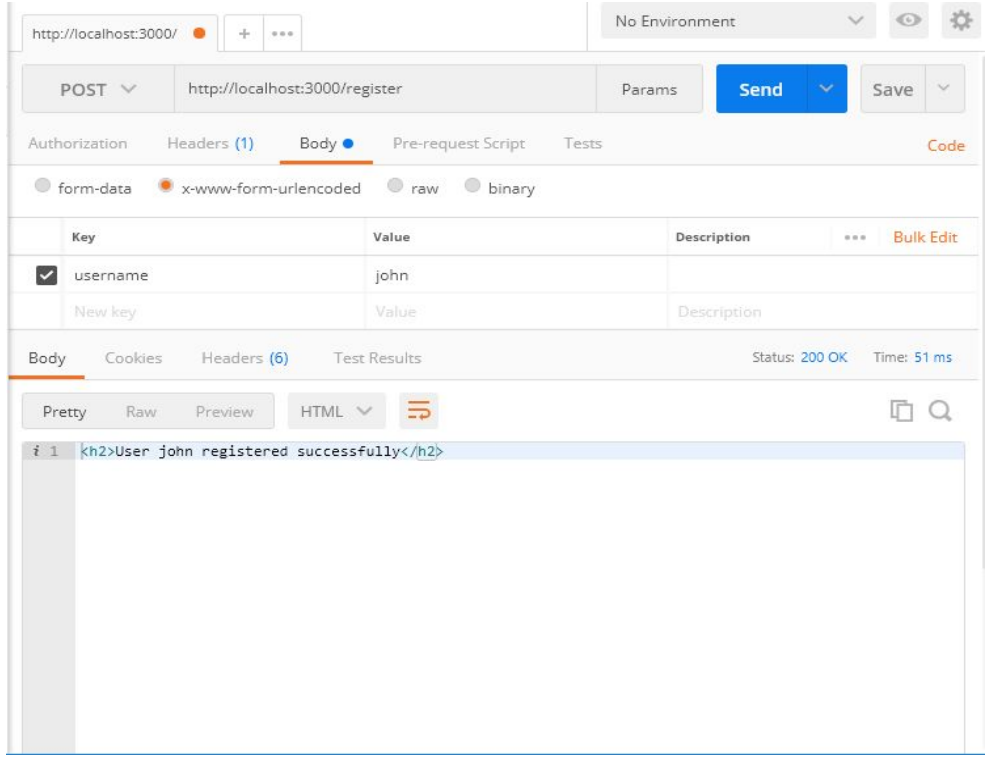
```
app.use(express.urlencoded({ extended: false }));

// POST /register gets urlencoded bodies  app.post('/register', (req,
res) => {
    if (!req.body)
        return res.sendStatus(400);

    let user = req.body;
    res.send(`User ${user.username} registered successfully`);
});
```

- The option extended allows to choose between parsing the URL-encoded data with the querystring library (when false) or the qs library (when true)
- The “extended” syntax allows for rich objects and arrays to be encoded into the URL-encoded format, allowing for a JSON-like experience with URL-encoded

[DAN.IT] EDUCATION

[Sending Form Data in PostMan]

- Under Body choose x-www-form-urlencoded option to send URL encoded form data
- Then enter the parameters as key/value pairs



[DAN.IT]
EDUCATION

[Exercise (7)]

- Create a site for managing the products list of a store
- Each product has an id, name and price
- The site should contain two pages:
  - The first page displays a form for entering the details of a new product to be added to the store
  - Clicking the Add Product button sends the product details to the server using HTTP POST, and lets the user enter another product
  - The second page shows the list of products in the store (saved in the server's memory)

Product Details Form

Id: 3

Name: Melon

Price: 5.7

Add Product

Show products list

Id	Name	Price
1	Apple	2.31
2	Banana	3.13
3	Melon	5.7

[DAN.IT] EDUCATION

[express.Router]

- Use the **express.Router** class to create modular, mountable route handlers
- A Router instance is a complete middleware and routing system
  - For this reason, it is often referred to as a “mini-app”
- Router allows you to separate the route definitions from the main app.js file

- The following example creates a router as a module, loads a middleware function in it, defines some routes, and mounts the router module on a path in the main app
- Create a new package named express-router
- Run npm init
- Create a routes sub-folder inside your package folder

[express.Router]

- Create a file named users.js in the routes sub-folder and add the following code to it:

```
const express = require('express');  const router =
express.Router();

// middleware that is specific to this router
router.use(function (req, res, next) {
    console.log('Time: ', Date.now());  next();
});

// define the login route  router.get('/login',
function(req, res) {
    res.send('User login');
});

// define the register route
router.get('/register', function(req, res) {
    res.send('User registration');
});

module.exports = router;
```

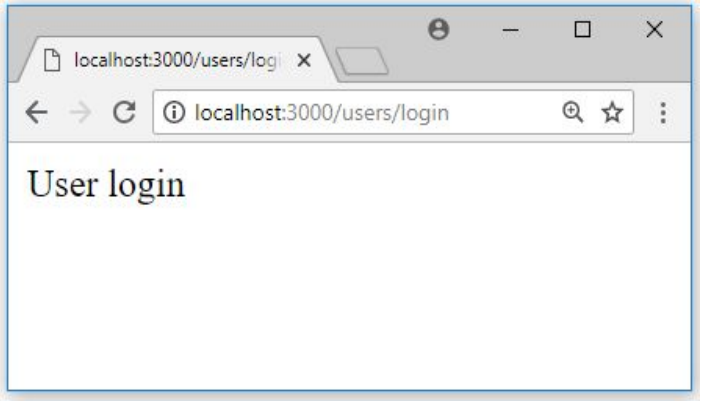
[DAN.IT] EDUCATION

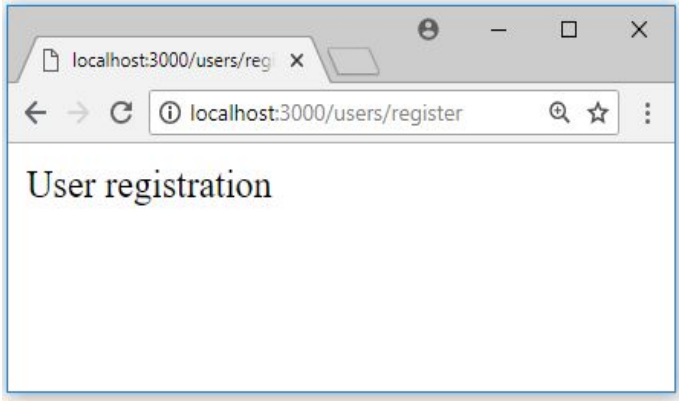
[express.Router]

- Then, load the router module in the app:

```
const express = require('express');  const app = express();

const users = require('./routes/users');  app.use('/users', users);

app.listen(3000);
```

- The app will now be able to handle requests to /users/login and /users/register, as well as call the middleware function that is specific to the route





[Third-Party Middleware]

- Use third-party middleware to add functionality to your Express apps
- Install the Node.js module for the required functionality, then load it in your app at the application level or at the router level
- For exmaple, to work with cookies, you can install and load the cookie-parser middleware

```
$ npm install cookie-parser
```

```
const express = require('express');
const app = express();
const cookieParser = require('cookie-parser');

// load the cookie-parsing middleware
app.use(cookieParser());
```

[DAN.IT] EDUCATION

[Third-Party Middleware]

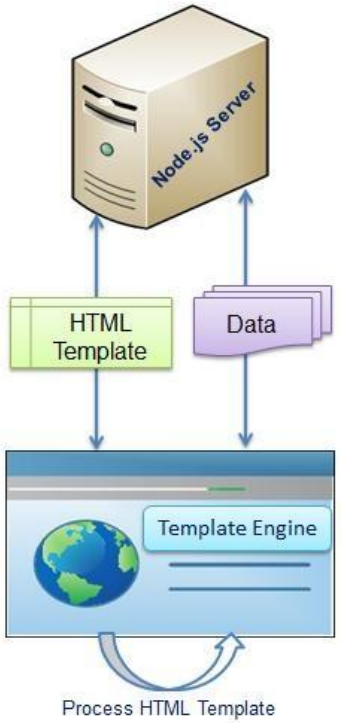
→ A partial list of third-party middleware functions that are commonly used with Express:

Middleware module	Description	Replaces built-in function (Express 3)
body-parser	Parse HTTP request body. See also: body, co-body, and raw-body.	express.bodyParser
compression	Compress HTTP responses.	express.compress
connect-rid	Generate unique request ID.	NA
cookie-parser	Parse cookie header and populate `req.cookies`. See also cookies and keygrip.	express.cookieParser
cookie-session	Establish cookie-based sessions.	express.cookieSession
cors	Enable cross-origin resource sharing (CORS) with various options.	NA
csurf	Protect from CSRF exploits.	express.csrf
errorhandler	Development error-handling/debugging.	express.errorHandler
method-override	Override HTTP methods using header.	express.methodOverride
morgan	HTTP request logger.	express.logger
multer	Handle multi-part form data.	express.bodyParser
response-time	Record HTTP response time.	express.responseTime
serve-favicon	Serve a favicon.	express.favicon
serve-index	Serve directory listing for a given path.	express.directory
serve-static	Serve static files.	express.static
session	Establish server-based sessions (development only).	express.session
timeout	Set a timeout period for HTTP request processing.	express.timeout
vhost	Create virtual domains.	express.vhost

[DAN.IT] EDUCATION

[Template Engines]

- Template engine helps us create an HTML template with minimal code
- At runtime, the template engine replaces variables in a template file with actual values, and transforms the template into an HTML file sent to the client



[DAN.IT] EDUCATION

[Pug]

- There are plenty of template engines to use with Node.js
- Some popular template engines that work with Express are Pug (formerly known as Jade), Mustache, and EJS
- To install a template engine, you need to install the corresponding npm package
- For example, to install Pug:

```
C:\NodeJS\template-engine>npm install pug
npm WARN template-engine@1.0.0 No description
npm WARN template-engine@1.0.0 No repository field.

+ pug@2.0.3
added 63 packages in 4.19s
```

[DAN.IT]
EDUCATION

[Template Page]

- Create a directory views, the directory where the template files are located
- Create a Pug template file named index.pug in the views directory, with the following content:

```
html
    head
        title= title  body
        h1= message
```

- The equals sign (=) is used to evaluate JavaScript expressions and output the result in the HTML code

[Using Template Engines with Express]

- To render template files, first set the following application setting properties:

```
app.set('views', './views');
app.set('view engine', 'pug');
```

  - **views** is the directory where the template files are located
    - This defaults to the views directory in the application root directory
  - **view engine** is the name of the template engine to use
- Then create a route to render index.pug
- Use **res.render(view, [locals])** to return the rendered HTML of the view
  - It accepts an optional parameter that is an object containing local variables for the view

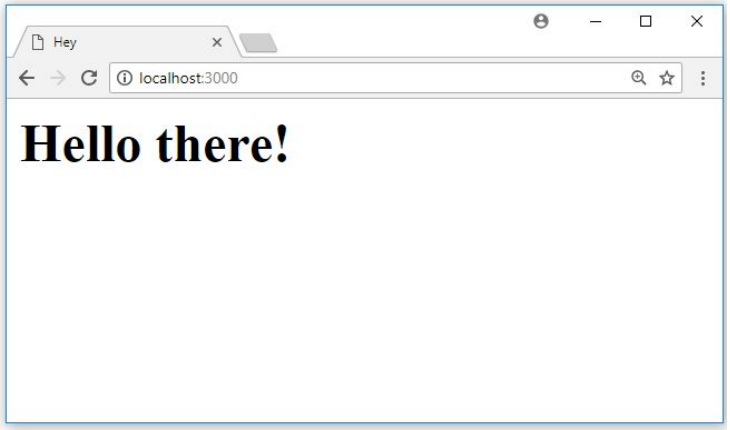
```
app.get('/', function (req, res) {
    res.render('index', { title: 'Hey', message: 'Hello there!' });
});
```

[DAN.IT] EDUCATION

[Using Template Engines with Express]

→ The final app.js looks like this:

```
const express = require('express');  const app = express();
app.set('view engine', 'pug');  app.get('/', function (req, res) {

    res.render('index', { title: 'Hey', message: 'Hello there!' });
});

app.listen(3000);
```



[HTML Tags]

- Text at the start of a line (or after only white space) represents an HTML tag
- Everything after the tag and one space will be the text contents of that tag
- Indented tags are nested, creating the tree structure of HTML

```
ul
    li Item A  li Item B  li Item C
```

```
<ul>
    <li>Item A</li>
    <li>Item B</li>
    <li>Item C</li>
</ul>
```

- Pug also knows which elements are self-closing:

```
img
```

```
<img/>
```

- To save space, Pug provides an inline syntax for nested tags:

```
a: img
```

```
<a><img/></a>
```

[Tags with Blocks]

- Often you might want large blocks of text within a tag
- A good example is writing JavaScript and CSS code in the script and style tags
- To do this, just add a . right after the tag name and indent the text contents of the tag one level:

```
script.
    let usingPug = true;
    if (usingPug)
        console.log('you are awesome');  else
        console.log('use pug');
```

```
<script>
    let usingPug = true;  if (usingPug)
        console.log('you are awesome');
    else
        console.log('use pug');
</script>
```

[JavaScript Code]

- Pug allows you to write inline JavaScript code in your templates
- Lines that start with - contain JavaScript code, which is not rendered to the output

```
- for (let i = 0; i < 3; i++)  li item
```

```
<li>item</li>
<li>item</li>
<li>item</li>
```

- Code between #{ and } is evaluated, escaped, and the result rendered into the output

```
- let title = "I, Robot";
- let author = "Issac Asimov";

p The book #{title} was written by
#{author}.
```

```
<p>The book I, Robot was written by Issac
Asimov.</p>
```

```
- let num1   =  5;
- let num2   =  8;

p num1 * num2 = #{num1 * num2}
```

```
<p>num1 * num2 = 40</p>
```

[DAN.IT] EDUCATION

[Tag Attributes]

- Tag attributes look similar to HTML (with optional commas), but their values are just regular JavaScript

```
a(href="http://www.google.com") Google
```

```
<a href="http://www.google.com">Google</a>
```

```
a(class="button",  href="http://www.google.com")
Google
```

```
<a class="button"
href="http://www.google.com">Google</a>
```

- Normal JavaScript expressions work fine, too:

```
- let url = "http://www.example.com";
a(href=url) Example
```

```
<a href="http://www.example.com">Example</a>
```

```
- let authenticated = true
div(class=authenticated ? 'authed' : 'anon')
```

```
<div class="authed"></div>
```

[DAN.IT] EDUCATION

[Style, Class and Id Attributes]

- The style attribute can be a string, or an object, which is handy when styles are generated by JavaScript:

```
a(style={color: 'red', background: 'green'})
```

```
<a style="color:red;background:green;"></a>
```

- Classes may also be defined using a .classname syntax:

```
a.button
```

```
<a class="button"></a>
```

- IDs may be defined using a #idname syntax:

```
a#main-link
```

```
<a id="main-link"></a>
```

[Conditions]

- Like in JavaScript, you can use if statements for checking conditions
  - The parentheses around the logical expression are optional
  - You may also omit the leading -

```
- let num = 15

if num > 10
    h2.green num is big
else
    h2.red num is small
```

```
<h2 class="green">num is big</h2>
```

[Iteration]

- Pug supports two primary methods of iteration: each and while

```
ul
    each val in [1, 2, 3, 4, 5]  li= val
```

```
<ul>
    <li>1</li>
    <li>2</li>
    <li>3</li>
    <li>4</li>
    <li>5</li>
</ul>
```

```
- let n = 0;
ul
    while  n < 4
        li= n++
```

```
<ul>
    <li>0</li>
    <li>1</li>
    <li>2</li>
    <li>3</li>
</ul>
```

- You can also use for as an alias for each

[DAN.IT] EDUCATION

[Comments]

- JavaScript comments produce HTML comments in the rendered page

```
// just some paragraphs  p First
paragraph
p Second paragraph
```

```
<!-- just some paragraphs-->
<p>First paragraph</p>
<p>Second paragraph</p>
```

- Comments that start with a hyphen (-) are only for commenting on the Pug code itself, and *do not* appear in the rendered HTML

```
//- this comment will not appear in the output
p First paragraph
p Second paragraph
```

```
<p>First paragraph</p>
<p>Second paragraph</p>
```

- Block comments work too

```
body
  //-
    Comments for your template writers.  Use as
    much text as you want.
```

[Includes]

- Includes allow you to insert the contents of one Pug file into another
- This is useful for sharing some HTML code between different pages

```
//- home.pug  doctype html  html
    include includes/head.pug  body
        h1 My Site
        p Welcome to my amazing site.
        include includes/footer.pug
```

```
//- includes/head.pug  head
    title My Site
    script(src='/scripts/jquery.js')
    script(src='/scripts/app.js')
```

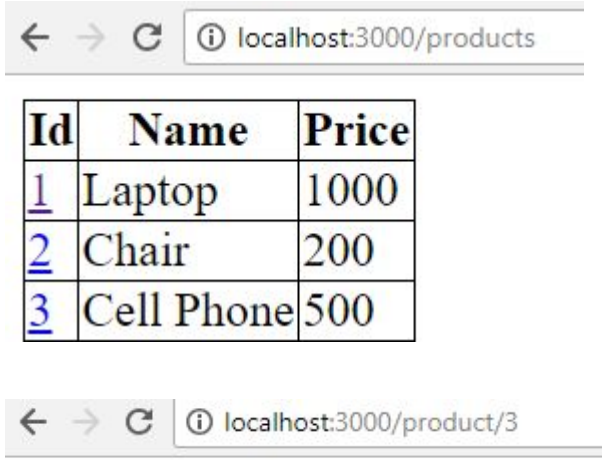
```
//- includes/foot.pug  footer#footer
    p Copyright (c) Roi Yehoshua
```

```
<!DOCTYPE html>
<html>
<head>
    <title>My Site</title>
    <script src="/scripts/jquery.js"></script>
    <script src="/scripts/app.js"></script>
</head>
<body>
    <h1>My Site</h1>
    <p>Welcome to my amazing site.</p>
    <footer id="footer">
        <p>Copyright (c) Roi Yehoshua</p>
    </footer>
</body>
</html>
```

[DAN.IT] EDUCATION

[Exercise (8)]

- Continue from the previous exercise
- Convert the products list into a template page (instead of building its HTML in the code)
- Add another template page that displays the details of a selected product
- In the products table, add a link for each product id, that will lead to the product's details page



[State Management]

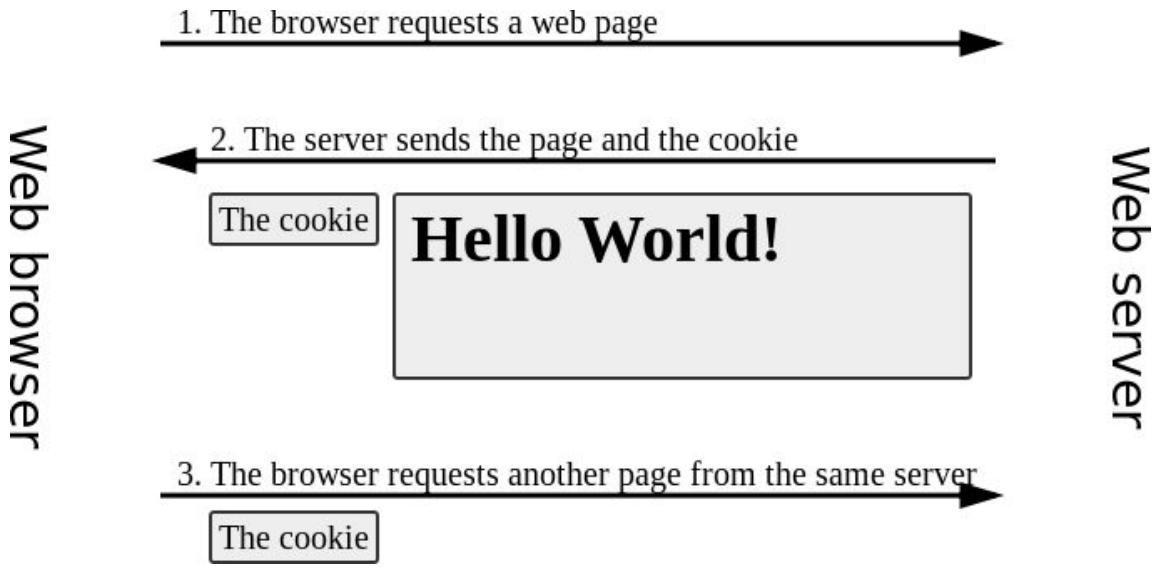
- HTTP is stateless
- In order to associate a request to any other request, you need a way to store user data between HTTP requests
- Cookies are used to transport data between the client and the server
- Sessions allow you to store information associated with the client on the server

Cookies	Session
Stored on the client side	Stored on the server side
Can only store strings	Can store objects
Can be set to a long lifespan	When users close their browser, they also lose the session

[DAN.IT] EDUCATION

[Cookies]

- Cookies are simple, small files/data that are stored on the client side
- Every time the user loads the website back, this cookie is sent with the request
- This helps us keep track of the user's actions



[cookie-parser]

- To use cookies with Express, you can use the cookie-parser middleware
- To install it, type the following command:

```
C:\NodeJS\cookies-demo>npm install cookie-parser
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN cookies-demo@1.0.0 No description
npm WARN cookies-demo@1.0.0 No repository field.

+ cookie-parser@1.4.3
added 3 packages in 1.227s
```

- The cookie-parser is used the same way as other middlewares:

```
const express = require('express');
const cookieParser = require('cookie-parser');

const app = express();  app.use(cookieParser());
```

[DAN.IT] EDUCATION

[Setting New Cookies]

- To set a new cookie, use **res.cookie(name, value [, options])**
- The value parameter may be a string or object converted to JSON
- Example:

```
res.cookie('name', 'Roi');
```

- You can pass an object as the value parameter - it is then serialized as JSON and parsed by bodyParser() when received in the request

```
res.cookie('cart', { items: [1,2,3] });
```

[DAN.IT] EDUCATION

[Cookies with Expiration Time]

- The options object allows you to set expiration time to the cookie:

```
// Expires after 600000 ms (10 min) from the time it is set
res.cookie('rememberme', 1, {expire: Date.now() + 600000});
```

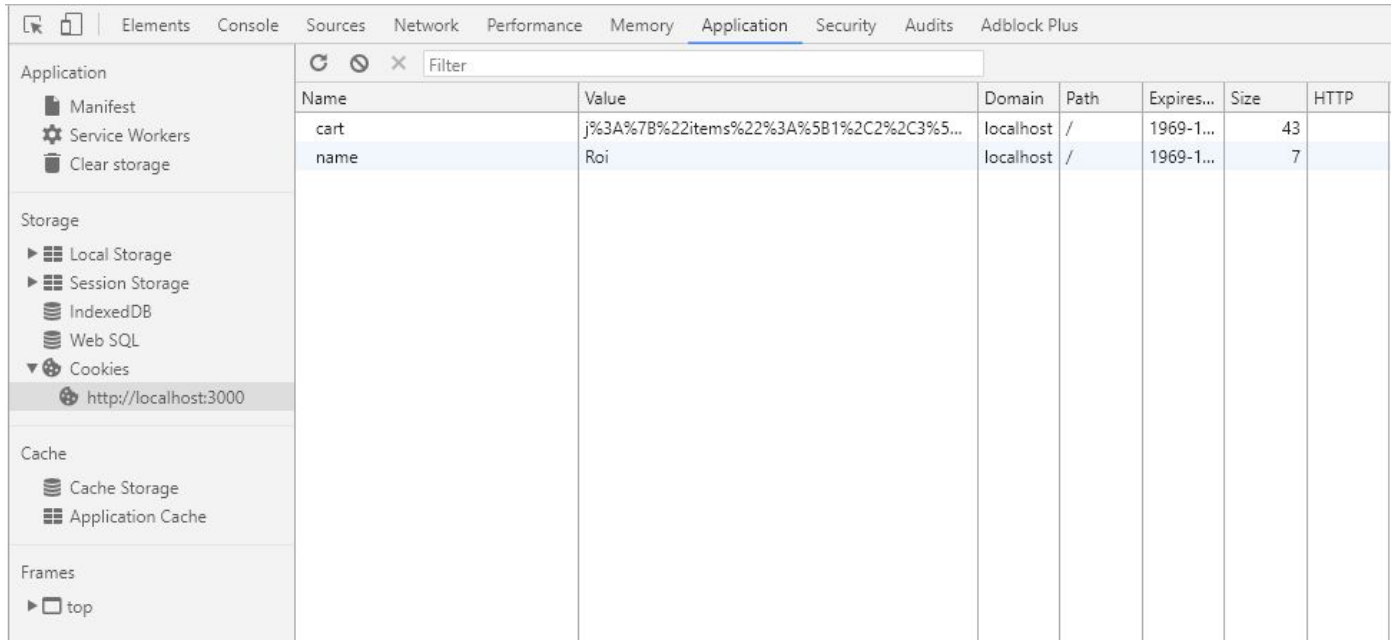
- If expiration time is not specified or set to 0, then it creates a session cookie
  - i.e., a cookie that is erased when the user closes the browser
- Another way to set expiration time is using the maxAge property
  - Using this property, we can specify expiration time which is relative to the current time (in milliseconds) instead of absolute time
- The following is equivalent to the example above:

```
res.cookie('rememberme', 1, {maxAge: 600000});
```

[DAN.IT] EDUCATION

[Inspecting Cookies in Chrome Developer Tools]

→ In the Google Chrome developer tools, you can view the cookies sent to the browser under the Application tab:



[DAN.IT]
EDUCATION

[Reading Cookies]

- The browser sends back the cookies every time it queries the server
- cookie-parser parses the Cookie header and populates req.cookies with an object keyed by the cookie names
- If the request contains no cookies, it defaults to {}

```
app.get('/show_cookies', (req, res) => {  res.write('name: ' +
    req.cookies.name + '<br/>');  res.write('remember me: ' +
    req.cookies.rememberme);  res.end();
});
```

- To delete a cookie, use the clearCookie() function

```
res.cookie('name', 'Roi');  res.clearCookie('name');
```

[DAN.IT] EDUCATION

[Signed Cookies]

- You can sign your cookies, so it can be detected if the client modified the cookie
- When the cookie gets read, it recalculates the signature and makes sure that it matches the signature attached to it
- If it does not match, then it will give an error
- To create a signed cookie you would use

```
res.cookie('name', 'value', {signed: true})
```

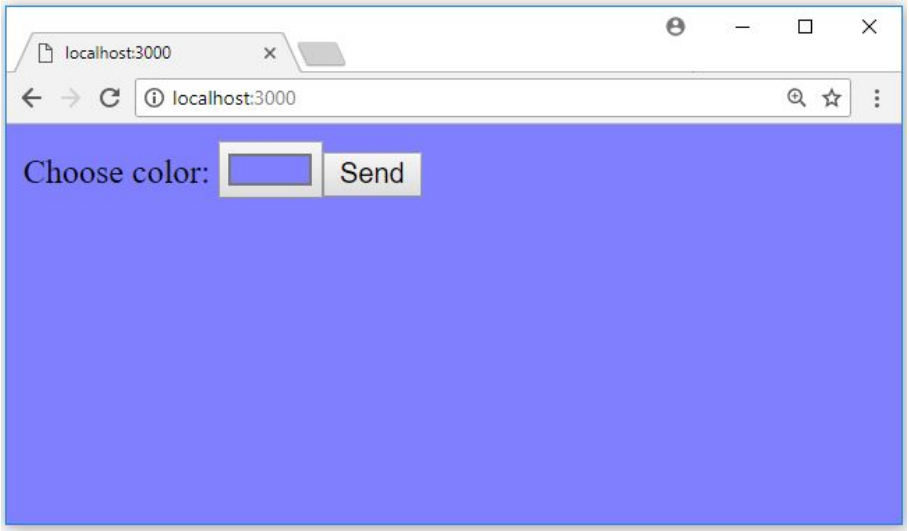
- And to access a signed cookie use the signedCookies object of req:

```
req.signedCookies['name']
```

[DAN.IT] EDUCATION

[Exercise (9)]

- Create a form that will enable the user to choose his favorite color
- After choosing a color, the background color of the page should change to that color
- The next time the user visits the page, his last chosen color should be displayed as the background color of the page



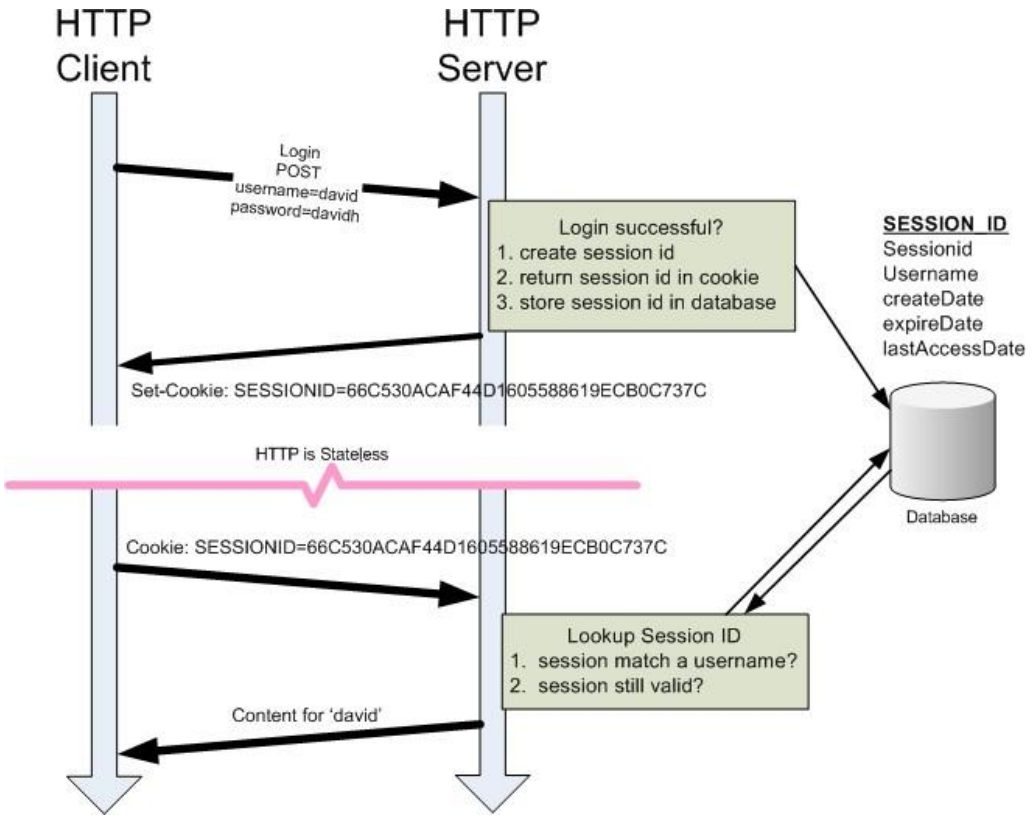
[Session]

- Provides a way to identify a user across more than one page request or visit to the web site and to store information about that user
- Session allows the application to store state
  - Based on what action a user took on Page A, we can show a different Page B

[DAN.IT] EDUCATION

[How Does Session Work?]

→ A **session ID or token** is a unique number which is used to identify a user that has logged into a website



[express-session]

- The package express-session can be used as a simple session middleware for Express

```
C:\NodeJS\session-demo>npm install express
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN session-demo@1.0.0 No description
npm WARN session-demo@1.0.0 No repository field.

+ express@4.16.3
added 50 packages in 3.236s

C:\NodeJS\session-demo>npm install cookie-parser
npm WARN session-demo@1.0.0 No description
npm WARN session-demo@1.0.0 No repository field.

+ cookie-parser@1.4.3
added 1 package in 1.045s

C:\NodeJS\session-demo>npm install express-session
npm WARN session-demo@1.0.0 No description
npm WARN session-demo@1.0.0 No repository field.

+ express-session@1.15.6
added 5 packages in 1.755s
```

[DAN.IT] EDUCATION

[Setting a Session]

- Here's how you can set up a simple session in Express:

```
const express = require('express');
const cookieParser = require('cookie-parser');
const session = require('express-session');

const app = express();

app.use(cookieParser());
app.use(session({ secret: 'keyboard cat'}));
```

  - The secret is used to sign the session ID cookie

- To store or access session data, simply use the request property req.session

[DAN.IT] EDUCATION

[Session Storage]

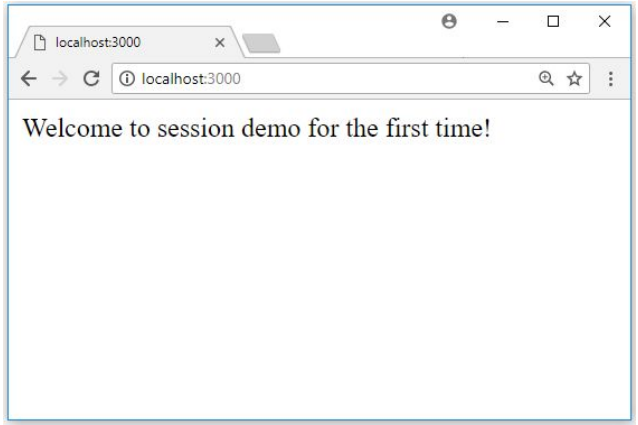
- The default server-side session storage is MemoryStore
- MemoryStore uses the application RAM for storing session data
- It is not recommended for use in a production environment because:
  - Memory consumption will keep growing with each new session
  - In case the app is restarted for any reason all session data will be lost
  - Session data cannot be shared by other instances of the app in a cluster
- Other ways to store session data:
  - In a cookie (use the session-cookie package)
  - In a memory cache server (use connect-memcached package)
  - In a database (e.g., for working with MongoDb use connect-mongo)

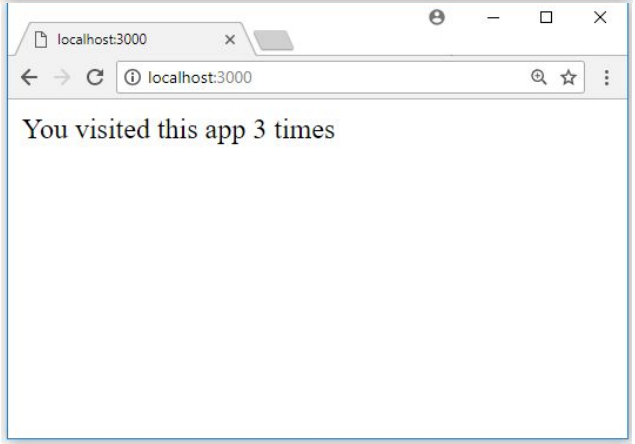
[DAN.IT] EDUCATION

[Session Usage Example]

→ To store or access session data, simply use the request property req.session

```
const express = require('express');
const cookieParser = require('cookie-parser');
const session = require('express-session');
const app = express();  app.use(cookieParser());

app.use(session({ secret: 'keyboard cat' }));

app.get('/', (req, res) => {
    if (req.session.pageViews) {  req.session.pageViews++;
        res.send('You visited this app ' + req.session.pageViews + ' times');
    } else {
        req.session.pageViews = 1;
        res.send('Welcome to session demo for the first time!');
    }
});

app.listen(3000);
```





[Session Timeout]

- **cookie.maxAge** specifies the expiration time of the session relative to the current server time (in milliseconds)
- By default, no maximum age is set

```
app.use(session({ secret: 'keyboard cat', cookie: { maxAge: 600000 }}))
// session expires after 10 minute
```

[DAN.IT] EDUCATION

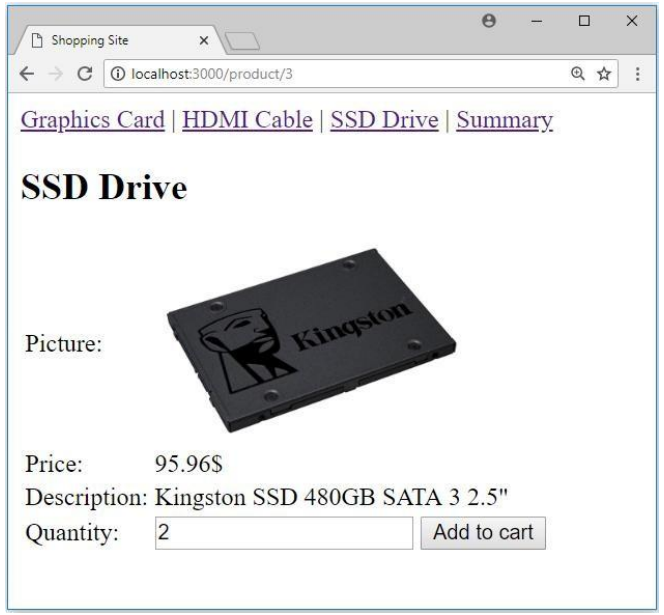
[Deleting a Session]

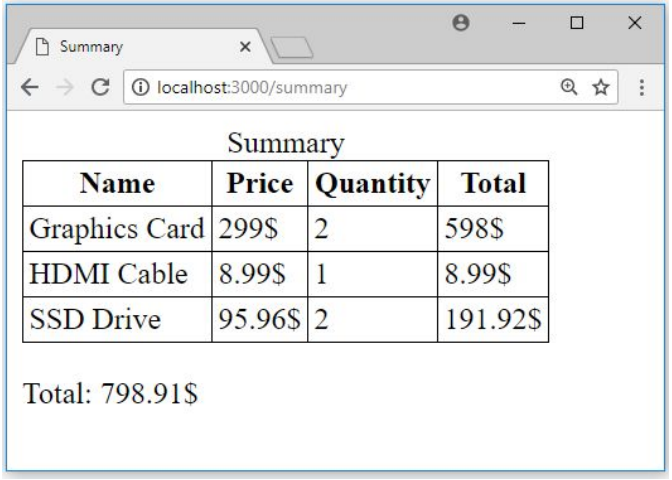
- The method **req.session.destroy()** is used to logout from the current session
- It destroys the session, deletes the session cookie, and unsets req.session
- The destroy() method accepts an optional callback function to be executed after the session is cleared from the store

```
req.session.destroy(function() {  res.send('Session
    deleted');
});
```

[Exercise (10)]

- Create a shopping site that will let the user choose one of 3 products
- Save the products the user has picked and their amounts in a shopping cart
  - Note: the user can choose the same product more than once
- The summary page should display the shopping cart and its total price





[Uploading Files]

- Multer is a node.js middleware for handling multipart/form-data, which is primarily used for uploading files
- Multer adds a body object and a file or files object to the request object
  - The **body** object contains the values of the text fields of the form
  - The **file** or files object contains the files uploaded via the form
- Create a new Node package named file-upload
- Install express and multer into your Node project:

```
C:\NodeJS\upload-file>npm install express multer
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN upload-file@1.0.0 No description
npm WARN upload-file@1.0.0 No repository field.

+ express@4.16.3
+ multer@1.3.0
added 71 packages in 6.347s
```

[DAN.IT] EDUCATION

[Uploading Files]

→ Copy the following code into app.js:

```
const express = require('express');  const path = require('path');
const multer = require('multer');  const app = express();

app.use(express.static(path.join(_dirname, 'public')));
app.set('view engine', 'pug');
const upload = multer({
    dest: 'public/pictures' // uploaded files will be saved in this
    folder
});




app.post('/profile', upload.single('profile_pic'), (req, res) =>
{
    // req.file is the photo file
    // req.body holds the text fields, if there are any  let name
    = req.body.name;
    let pictureUrl = path.join('pictures', req.file.filename);

    res.render('index', { name, pictureUrl
    });
});
app.listen(3000);
```

Check that the file name matches the name attribute in your html

[File API]

→ Each file contains the following information:

Key	Description
originalname	Name of the file on the user's computer
mimetype	Mime type of the file
size	Size of the file in bytes
destination	The folder to which the file has been saved
filename	The name of the file within the destination
buffer	A buffer of the entire file

[Uploading Files]

→ Create register.html in the /public folder and copy the following code into it:

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Register</title>
</head>
<body>
    <form action="/profile" enctype="multipart/form-data" method="post">
        <table>
            <tr>
                <td>Display name:</td>
                <td><input type="text" id="name" name="name" /></td>
            </tr>
            <tr>
                <td>Profile picture:</td>
                <td><input type="file" id="profile_pic" name="profile_pic"
                     accept="image/png, image/jpeg" /></td>
            </tr>
        </table>
        <input type="submit" value="Register" />
    </form>
</body>
</html>
```

Note the encoding type of the form

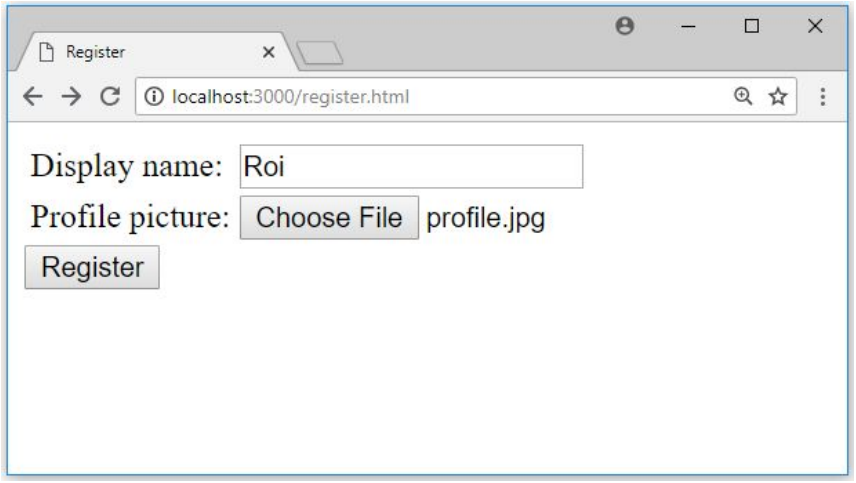
The accept attribute specifies the types of files that the server accepts

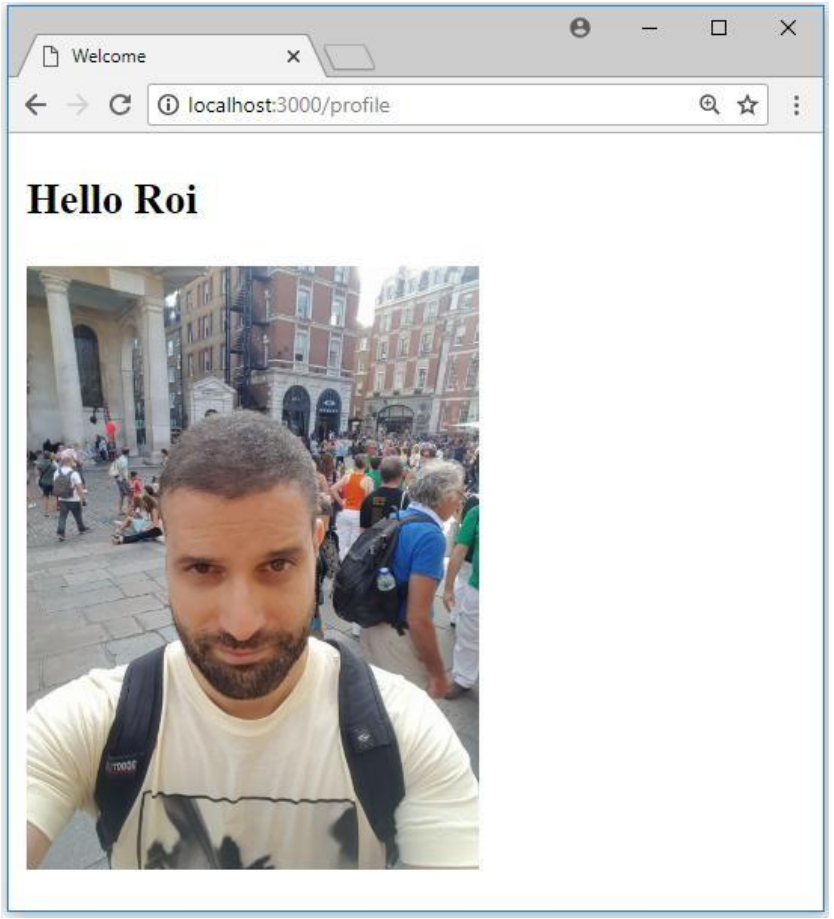
[Uploading Files]

→ Create index.pug in the /views folder and copy the following code into it:

```
<!DOCTYPE html>
html(lang="en")  head
     meta(charset="UTF-8")
     meta(name="viewport", content="width=device-width, initial-scale=1.0")
     meta(http-equiv="X-UA-Compatible", content="ie=edge")  title Welcome
body
     h3 Hello #{name}  img(src=pictureUrl, width="200")
```

[Demo]





[DAN.IT]
EDUCATION

[Cache Management]

- Caching is a commonly used technique to improve the performance of any application, be it desktop, mobile or web
- For example, what if we have a complex and heavy page that takes 2 second to generate the HTML output?
- Even if we enable client-side cache for this page, the web server will still need to render the page for each different user accessing our web application
- The goal of **server-side cache** is to avoid rendering the same page over and over again by saving the result of client requests
- In our example above, the first request that reaches our server would still take 2 seconds to generate the HTML, but the following requests would hit the cache instead and the server would be able to send the response in a few milliseconds

[DAN.IT] EDUCATION

[Cache Management]

- The memory-cache npm module is a simple in-memory cache for NodeJS
- Create a new package named cache-demo
- Install express, pug and memory-cache:

```
C:\NodeJS\cache-demo>npm install express pug memory-cache
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN cache-demo@1.0.0 No description
npm WARN cache-demo@1.0.0 No repository field.

+ pug@2.0.3
+ memory-cache@0.2.0
+ express@4.16.3
added 114 packages in 13.044s
```

[Cache Management]

- require('cache') returns the default instance of Cache
- Cache provides two main methods:
  - **put(key, value, time, timeoutCallback)**
    - Simply stores a value
    - If time isn't passed in, it is stored forever
    - Will actually remove the value in the specified time in ms (via setTimeout)
    - timeoutCallback is optional function fired after entry has expired with key and value passed (function(key, value) {})
  - **get(key)**
    - Retrieves a value for a given key
    - If value isn't cached, returns null

[DAN.IT] EDUCATION

[Cache Management]

→ Simple example for using the cache:

```
const  mcache  =  require('memory-cache');

mcache.put('houdini', 'disappear', 100, function(key, value) {
    console.log(key + ' did ' + value);
});
console.log('Houdini will now ' + mcache.get('houdini'));

setTimeout(() => {

    console.log('Houdini is ' + mcache.get('houdini'));
}, 200);
```

```
C:\NodeJS\cache-demo>nodemon app.js
[nodemon] 1.18.3
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: *.*
[nodemon] starting `node app.js`
Houdini will now disappear
houdini did disappear
Houdini is null
```

[DAN.IT] EDUCATION

[Cache Management]

→ Now define the following cache middleware:

```
let cache = (duration) => {
    return (req, res, next) => {

        // Look for the cached output using the request's URL as the key  let key = '_express_' +
        req.originalUrl || req.url;
        let cachedBody = mcache.get(key);

        if (cachedBody) {
            // If the cache is found, it sent directly as the response  res.send(cachedBody);
        } else {
            // If the URL is not cached yet, we wrap Express's send() function to cache the response
            // before actually sending it to the client and then calling the next middleware.
            res.sendResponse = res.send;
            res.send = (body) => {
                mcache.put(key, body, duration * 1000);  res.sendResponse(body);
            };
            next();
        }
    }
};
```

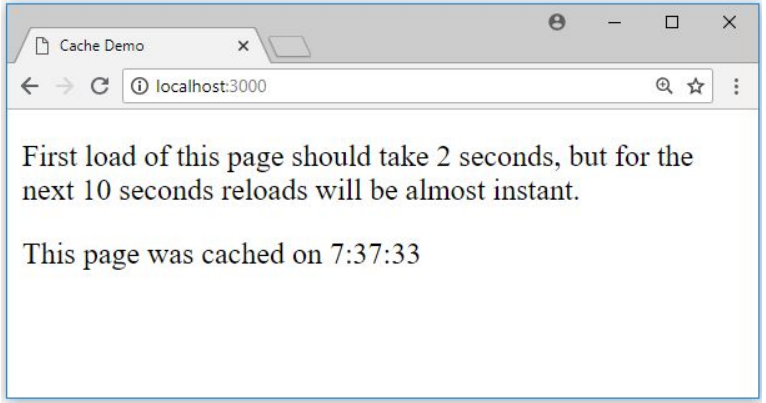
[Cache Management]

- You can easily plug the middleware into any existing Express web application by simple adding the cache middleware for each route you may want to cache:

```
app.get('/', cache(10), (req, res) => {  setTimeout(() => {
        let date = new Date();
        let time = date.getHours() + ':' + date.getMinutes() + ':' + date.getSeconds();
        res.render('index', { time });
    }, 2000) // setTimeout is used to simulate a slow processing request
});
```

- Index.pug:

```
<!DOCTYPE  html>
html(lang="en")
head
    title Cache Demo
body
    p First load of this page should take 2 seconds, but
for the next 10 seconds reloads will be almost instant.
    p This page was cached on #{time}
```



[DAN.IT] EDUCATION

[Cache Management]

- The caching mechanism works for routes that respond with HTML, JSON, XML or any other content-type
- PUT, DELETE and POST methods should never be cached
  - Since the cache will prevent the route handler from running
- In-memory cache is the fastest option available, but we'll lose the cached content if the server goes down, and it won't be shared between multiple node.js process
- Alternatively, you can use a distributed cache service like Redis, using the npm module express-redis-cache

[DAN.IT] EDUCATION