DevGeekWeek 2024
INSPIRED BY INNOVATION

Welcome to DevGeekWeek

We will start in a few minutes

matrix

JOHN BRYCE
תלמדו הייטק. זה עובד!
a matrix company

FullStack Development with React & Nodejs

matrix

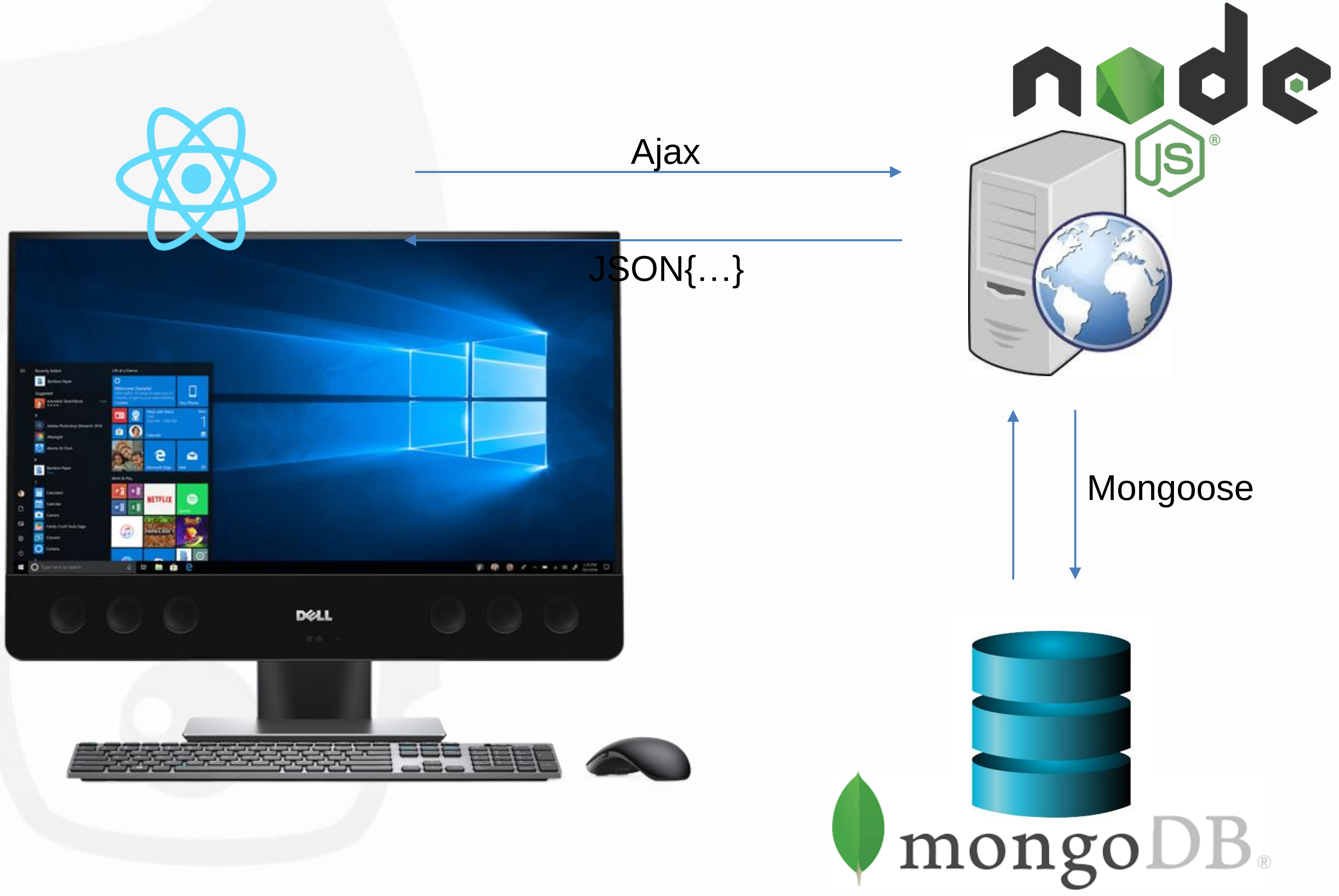
JOHN BRYCE
תלמדו הייטק. זה עובד!
a matrix company

Agenda

- Nodejs
  - JavaScript , how it works?
  - Async programming
  - Express, writing your api
  - Modular Application
  - Libs

Agenda

- React
  - Overview
  - React create app – CLI
  - React components, state and props
  - Typescript
  - Async middleware - thunk


Ajax
JSON{…}
Mongoose
mongoDB

JavaScript

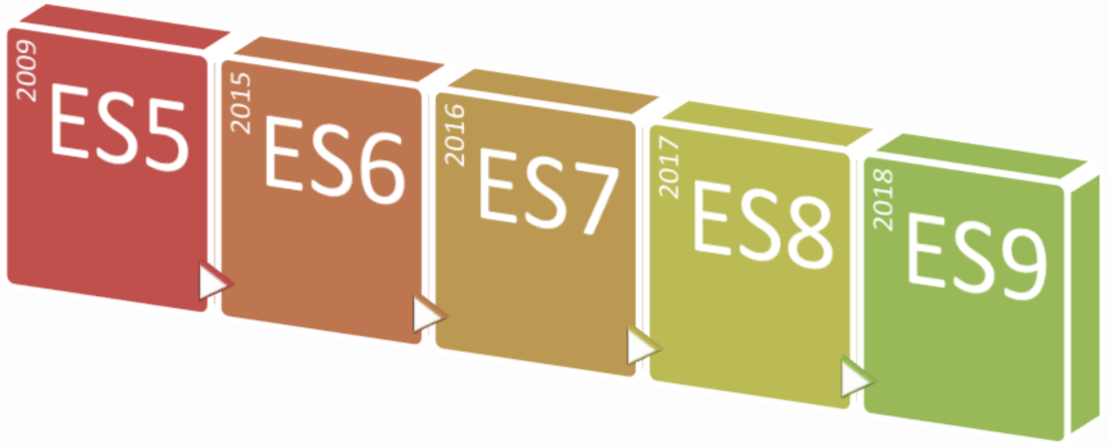
- Web pages (HTML – from static to dynamic)
- Dynamic script language
- Lightweight
- Interpreted
- Supports object-oriented language
- Prototype-based

Javascript - History

- Brendan Eich
- NetScape
- 1995
- LiveScript
- European Computer Manufacturers Association
- ECMAScript 2018

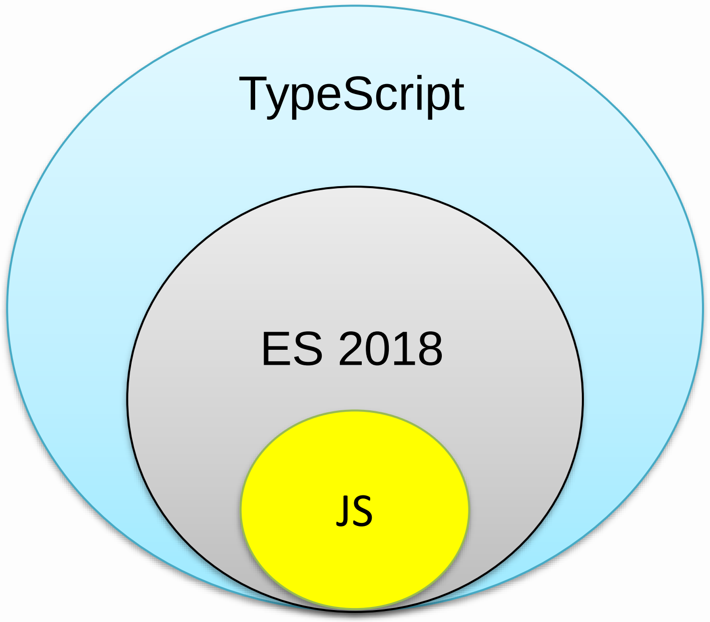
ECMA 2023 + Latest features

- Rest and spread
- Added Asynchronous iteration
- Promise.finally()
- Async await



TypeScript

- On top of JavaScript
- Strongly types
- Microsoft
- Scales JavaScript
- Transpiled to ES5\6



V8 Engine

- V8 is Google’s open source high-performance JavaScript engine, written in C++ and used in Google Chrome.
- Increased performance
- Compiling JavaScript to native machine code

JavaScript Engines & implementations

- **TraceMonkey** – Mozilla, native-code compilation
- **Rhino** – Mozilla, open-source implementation, Java
- **Chakara** – Explorer, JIT Compiler

Before we start…

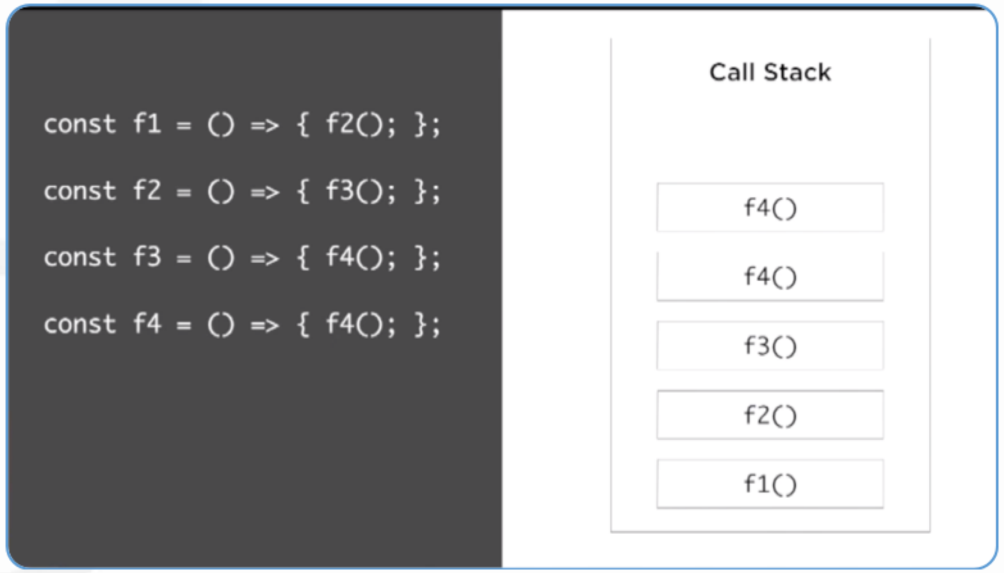
LETS PLAY A GAME...
memegenerator.net

You probably don’t know Node

What is the Call Stack
Is it part of V8?

Call Stack

- Part of V8 Engine (in case chrome & nodejs)
- Model that implemented by different browsers
- Use to keep track on function invocations



```
const f1 = () => { f2(); };

const f2 = () => { f3(); };

const f3 = () => { f4(); };

const f4 = () => { f4(); };
```

What is the Event Loop
Is it part of V8?

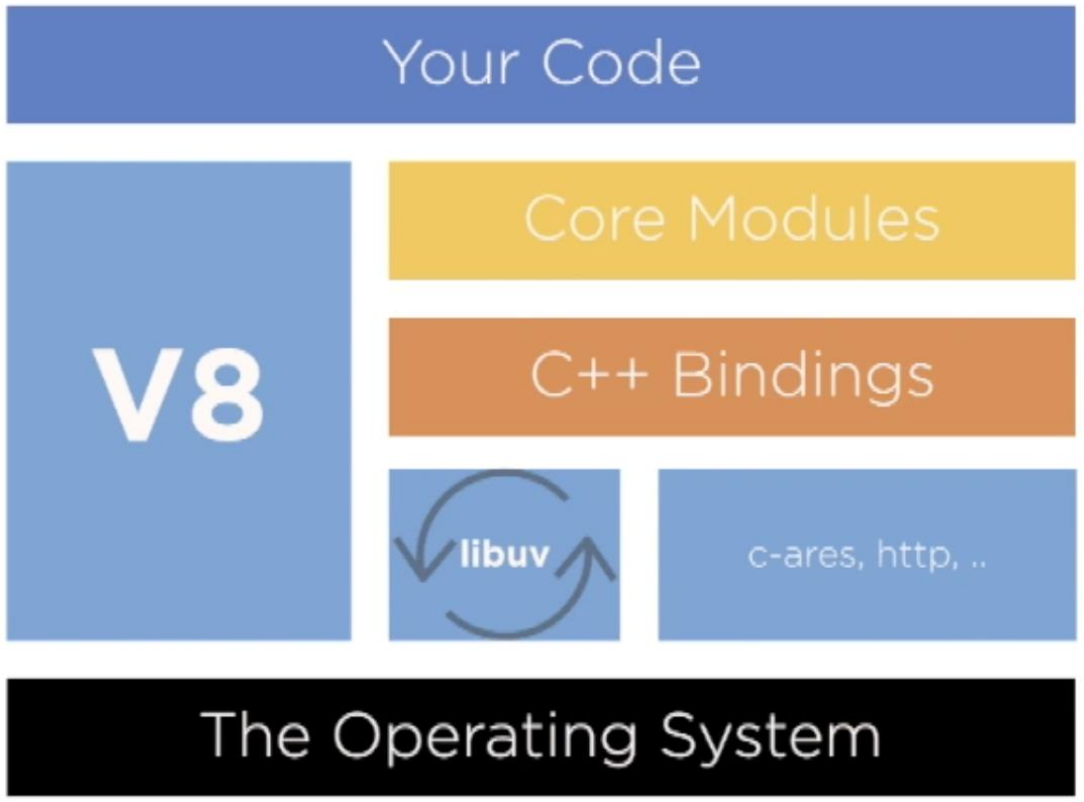
Event Loop

- Handles external events and convert them into callback invocation

Event Loop

- A loop that picks events from the event queue and pushes their callbacks to call stack

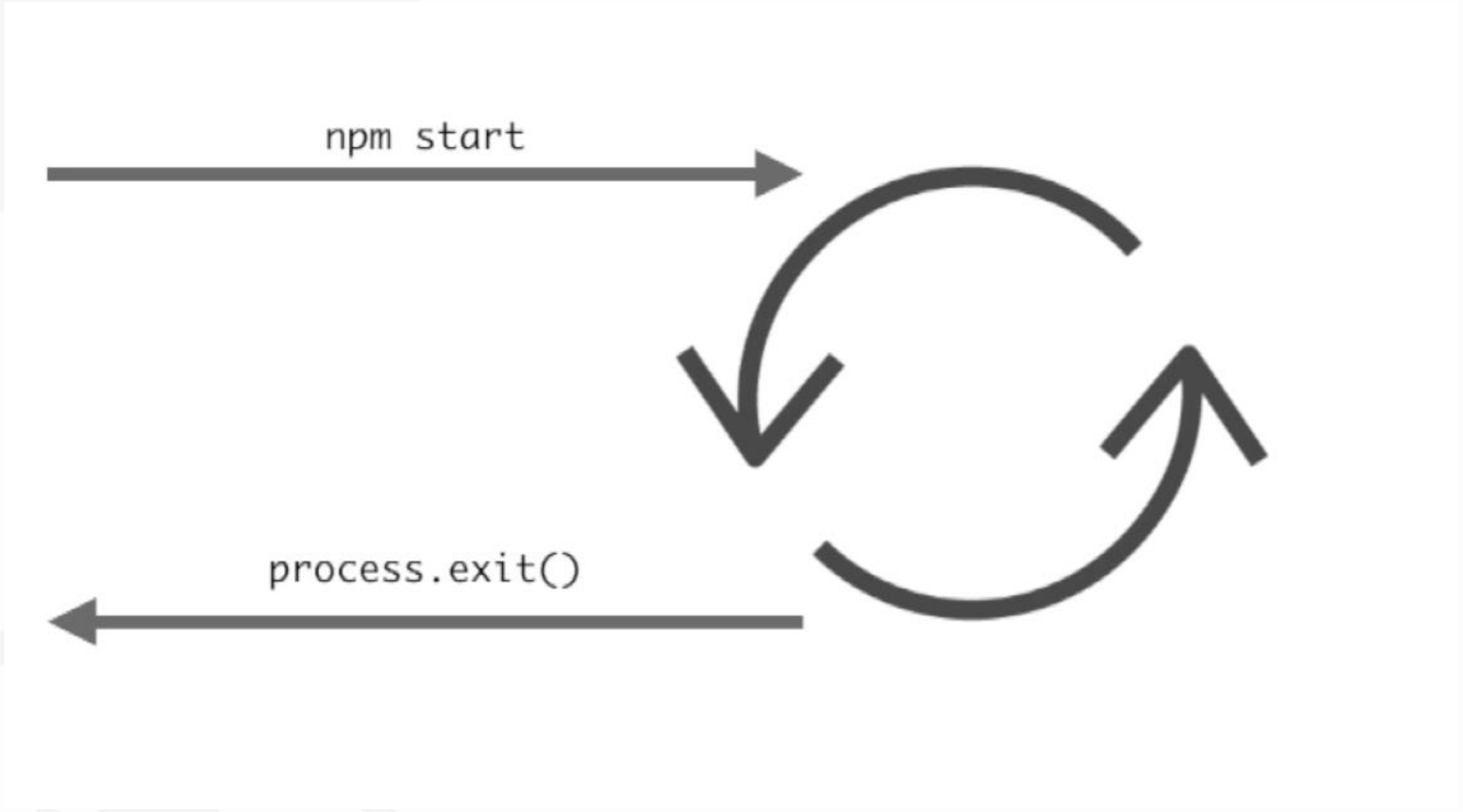
Event Loop



http://latentflip.com/loupe/?code=JC5vbignYnV0dG9uJywgJ2NsaWNrJywgZnVuY3Rpb24gb25DbGljaygpIHsKICAgIHNldFRpbWVvdXQoZnVuY3Rpb24gdGltZXIoKSB7CiAgICAgICAgY29uc29sZS5sb2coJ1lvdSBjbGlja2VkIHRoZSBidXR0b24hJyk7ICAgIAogICAgfSwgMjAwMCk7Cn0pOwoKY29uc29sZS5sb2coIkhpISIpOwp2YXIgYSA9IDU7CnZhciBiID0gYSArIDU7CgpzZXRUaW1lb3V0KGZ1bmN0aW9uIHRpbWVvdXQoKSB7CiAgICBjb25zb2xlLmxvZygiQ2xpY2sgdGhlIGJ1dHRvbiEiKTsKfSwgNTAwMCk7Cgpjb25zb2xlLmxvZygiV2VsY29tZSB0byBsb3VwZS4iKTs%3D!!!PGJ1dHRvbj5DbGljayBtZSE8L2J1dHRvbj4%3D

What would Node do when both the call stack and the event loop queue are empty ?

Exit



Can Node work Without V8 ?

Yes

v8

Chakra

What are the 5 steps the require function does?

Require(’whatEver’)

- Resolving – map the string to actual path in file system
- Loading – load the file to memory
- Wrap - wrap the file with IIFE
- Evaluating – with V8
- Cache – next time the file will be load from the cache

How can you check for the existence Of local module ?

Use Resolve

Require.resolve(“Module name”)

Nodejs

01 Fast

Transform JS into machine code

02 Single Threaded

Runs using one thread only

03 Replaceable

Other vendors aim to provide their own implementation

04 Sometimes not fast enough

GC, dynamic typing, etc – all come with a price

Nodejs

2024

8M

Servers worldwide

1st

Most popular framework (StackOverflow)

4nd

Most loved technology (StackOverlow)

Nodejs Advantages

- Platform-independent
  - Servers can run within the Node.js runtime on Windows, Mac OS X and Linux with no changes
- Open-source
- Improved performance (single-threaded)
- Same language for client and server software
- No additional web server software is needed
- Tons of libraries

Nodejs Functionality

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

Nodejs Agenda

Your Code + Community Code + Node Libraries

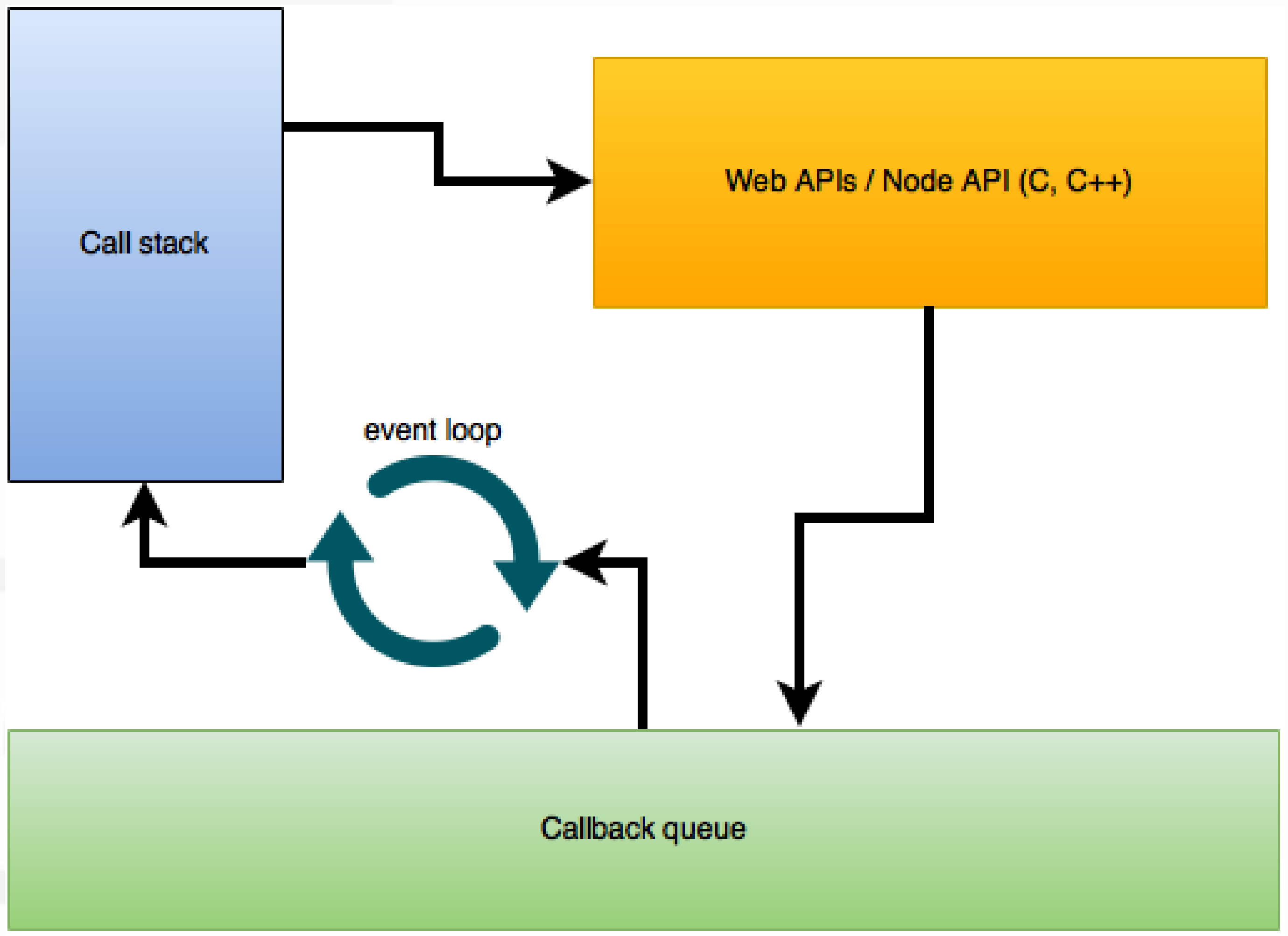
+ Service Architecture + Web Framework

+ Testing

+ Production Setup

+ Node Engine

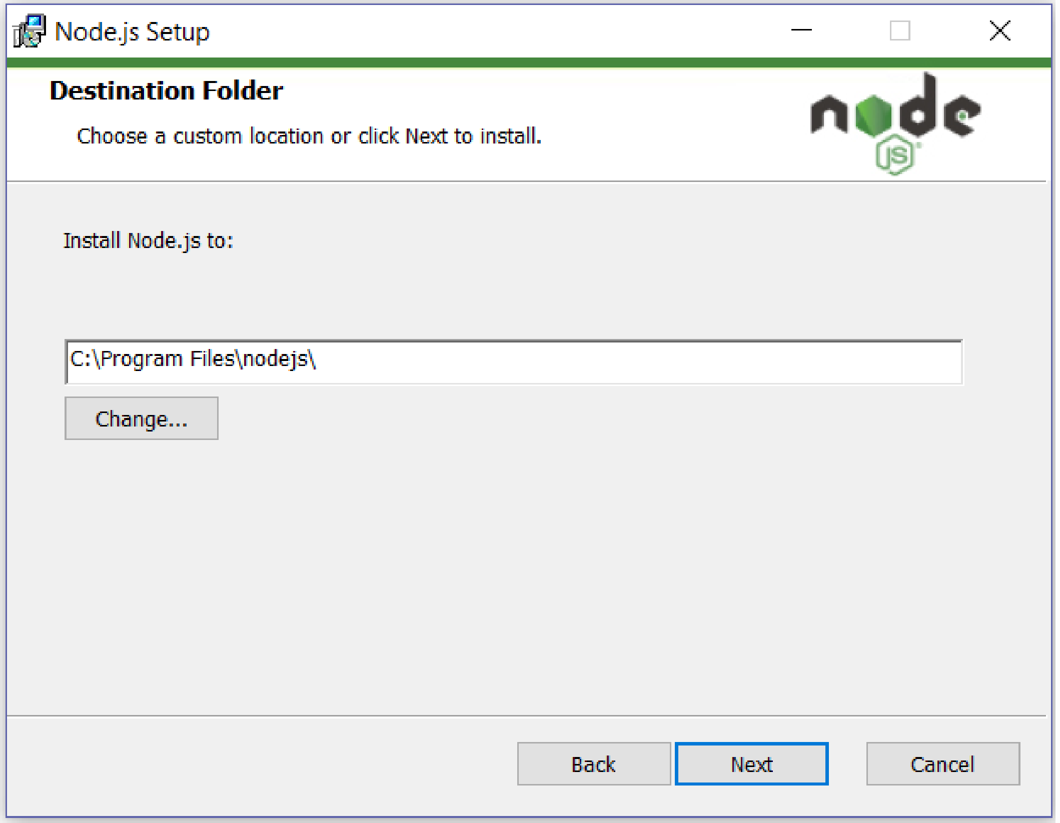
Nodejs Architecture



Getting started with Node.js

- Node.js is nothing more as node.exe file – process
- It’s very easy to install with latest installer available on nodejs.org web site.
- After downloading the MSI file, just run the setup
- By default, NodeJS will be installed in C:\Program Files\nodejs

Installation



Node.js Setup

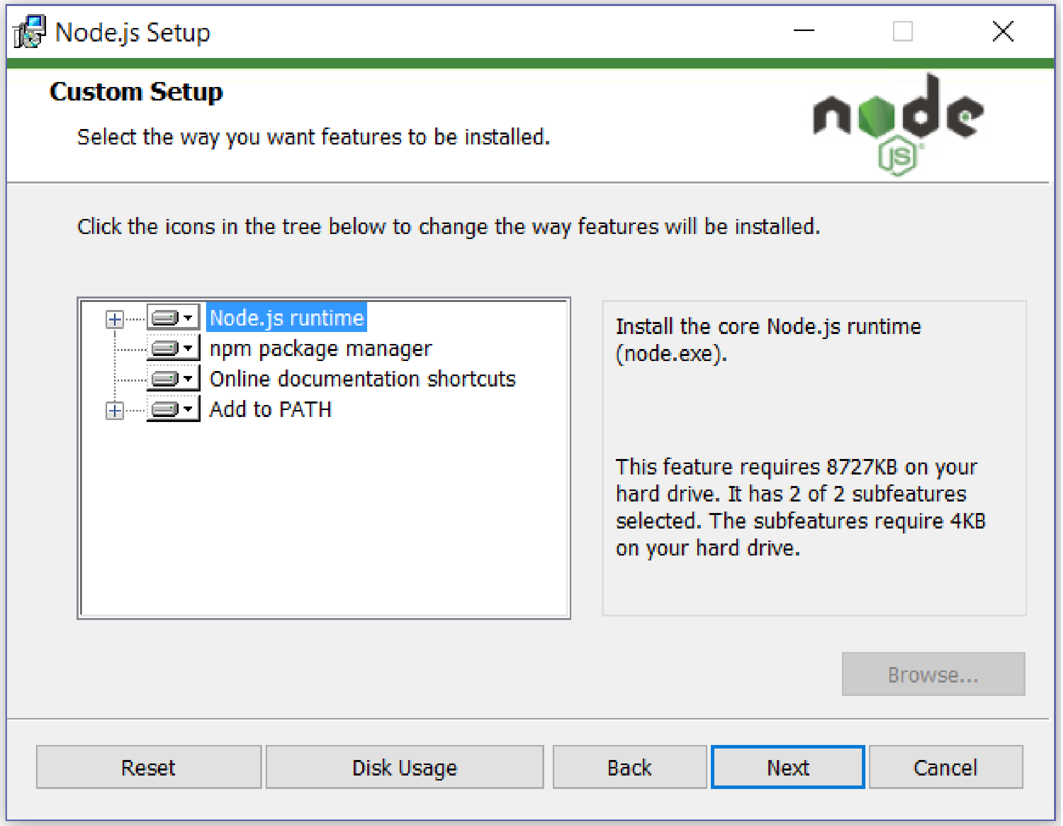
Destination Folder

Choose a custom location or click Next to install.

Install Node.js to:

C:\Program Files\nodejs\

Installation


Node.js Setup

Custom Setup

Select the way you want features to be installed.

Click the icons in the tree below to change the way features will be installed.

- Node.js runtime
- npm package manager
- Online documentation shortcuts
- Add to PATH

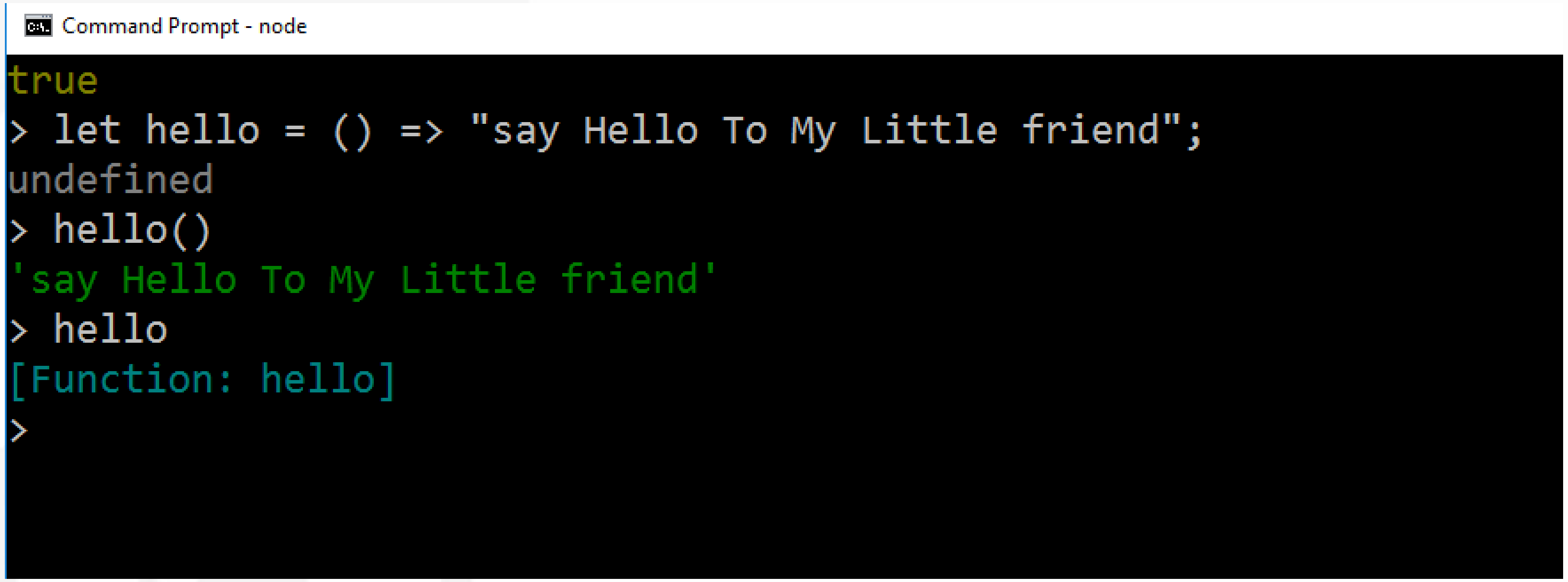
Install the core Node.js runtime (node.exe).

This feature requires 8727KB on your hard drive. It has 2 of 2 subfeatures selected. The subfeatures require 4KB on your hard drive.

Trying things out

- After installing node it should be available in your PATH.
- 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.

Example



```
true
> let hello = () => "say Hello To My Little friend";
undefined
> hello()
'say Hello To My Little friend'
> hello
[Function: hello]
>
```

Run index.js file

- Instead of typing the code into interpreter console, you can save it to a local file ‘index.js’ and run it from the command line

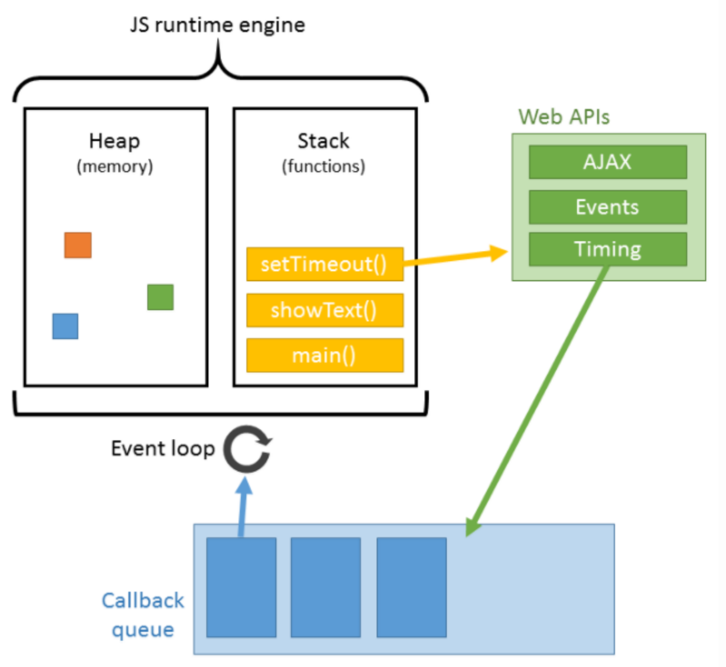
- Steps:
  - Create index.js
  - Insert some js code
  - Go to CMD and run > node index.js

Timers

- Set Interval
- Set Timeout
- Minimal Time period is 4ms.
- Timer wont start until the outer most function is finished, script ended.
- Arguments (callback, time-ms)

Timers – setTimeout VS setInterval

- Control async operations
- Ability to cancel timers



JavaScript Asynchronous programing

- Callbacks
- Promise
- Async Await

New
ES7

Callbacks

- A function supplied as a parameter to a function
- The function will execute it when completed
- Problem CallBackHell

Demo

Promise

- A placeholder object for eventual results of an asynchronies operation
- When resolved the data is available
- States: pending, fulfilled, rejected
- Chainable

Demo

Async Await

- Now we can write code that looks sync but does async operations.
- Async is a key word that comes before function.
- All the function return will be wrapped as a resolved promise

Async Await

- Using Await in async functions only
- Await is a key word for the execution of a function that returns a promise
- Function resolved successfully the result of await is the return value of the function called
- If fails return rejected value
- Try - resolve catch- reject

Demo

Modules

Good authors divide their books into chapters and sections; good programmers divide their programs into modules.

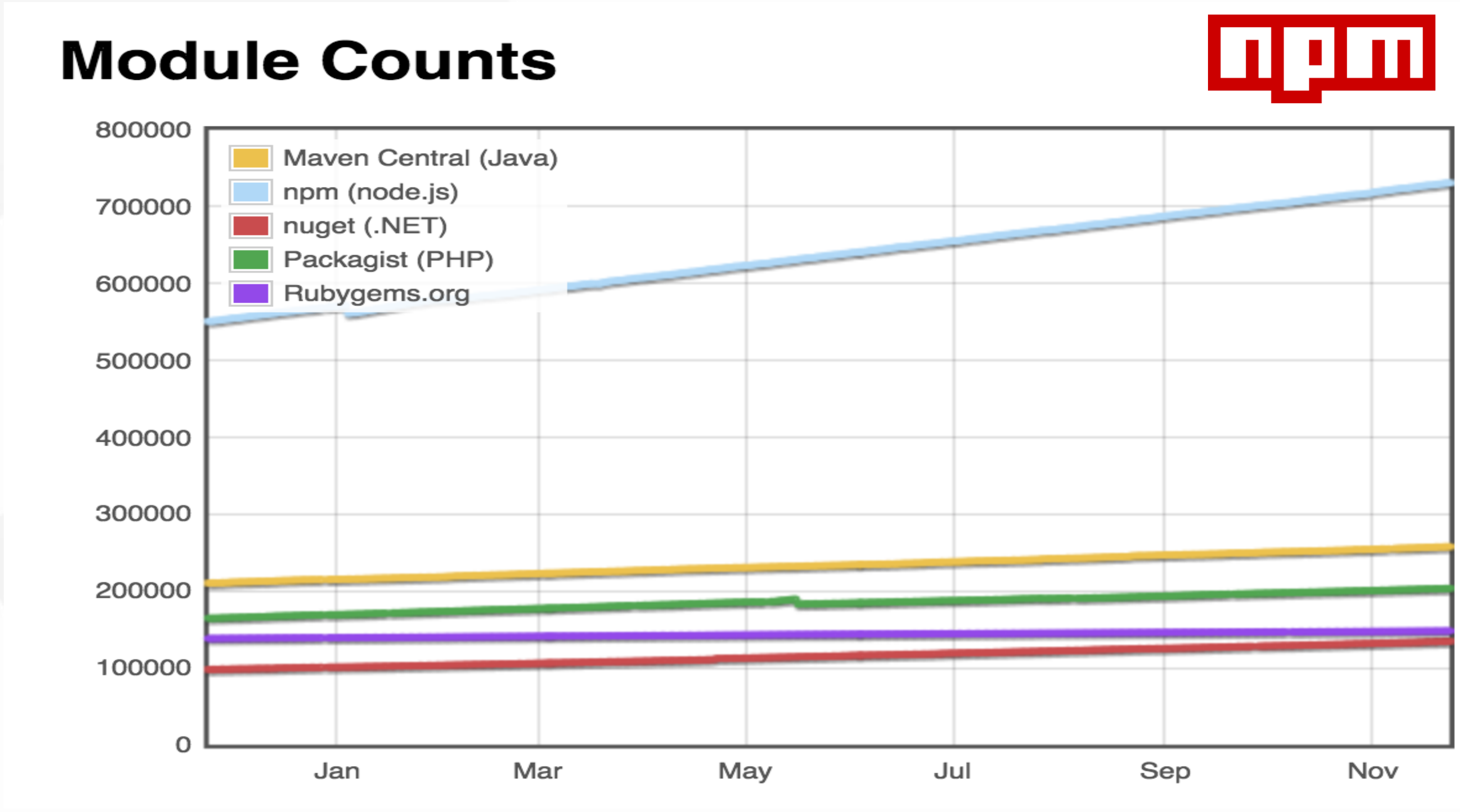
Modules

- Maintainability
- Name spacing
- Reusability

NPM

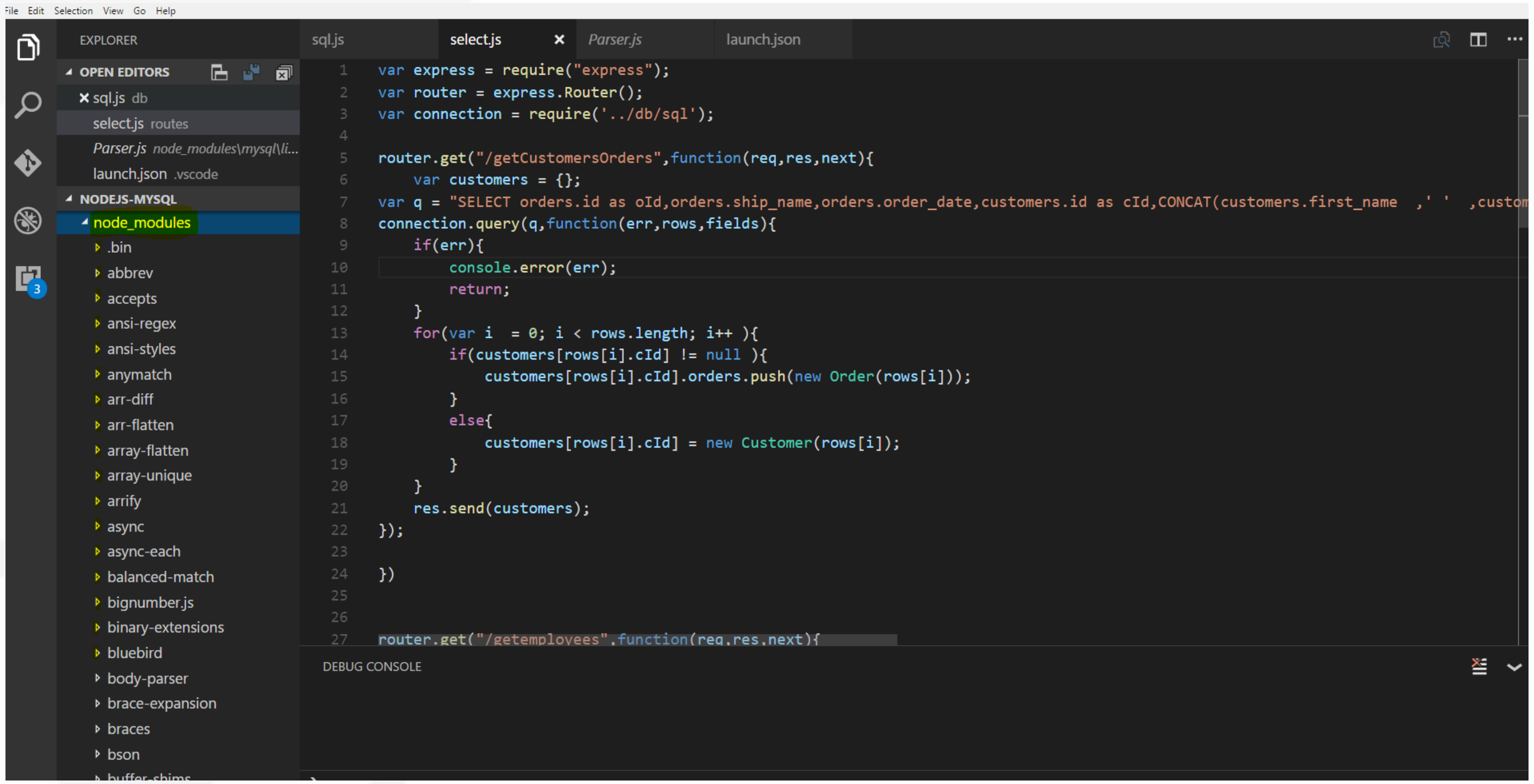
- **N**ode **P**ackaged **M**anager - a package manager for
  Node.js libraries.
- Join the modular development revolution
- Enables to install many Node.js packages.
- By default, packages are installed in %appdata%\npm
  on Windows and /usr/local on Linux

NPM



Module Counts
npm
800000
700000
600000
500000
400000
300000
200000
100000
0
Jan
Mar
May
Jul
Sep
Nov
Maven Central (Java)
npm (node.js)
nuget (.NET)
Packagist (PHP)
Rubygems.org

NPM



```
var express = require("express");
var router = express.Router();
var connection = require('../db/sql');

router.get("/getCustomersOrders",function(req,res,next){
    var customers = {};
var q = "SELECT orders.id as oId,orders.ship_name,orders.order_date,customers.id as cId,CONCAT(customers.first_name  ,' '  ,custom
connection.query(q,function(err,rows,fields){
    if(err){
        console.error(err);
        return;
    }
    for(var i  = 0; i < rows.length; i++ ){
        if(customers[rows[i].cId] != null ){
            customers[rows[i].cId].orders.push(new Order(rows[i]));
        }
        else{
            customers[rows[i].cId] = new Customer(rows[i]);
        }
    }
    res.send(customers);
});

})


router.get("/getemployees",function(req,res,next){
```

Commands

- **npm install -g <package name>** – install package globally.
  - Global packages are usually for executable commands
- **npm install <package name>** – install package locally
  - Local packages are for the use of require in the app
- **npm uninstall <package name>** - uninstall global package
- **npm uninstall -g <package name>** – uninstall global package
- **npm ls -g** – list global packages
- **npm ls** – list local packages
- **npm update -g** – update global packages
- **npm update** – update local packages
- **npm init** – creating your package.json

package.json

- With a package.json file in the root of your app dir, you don’t need to manually install every package.
- By default, **npm install** will install all modules listed as dependencies.

Creating our API

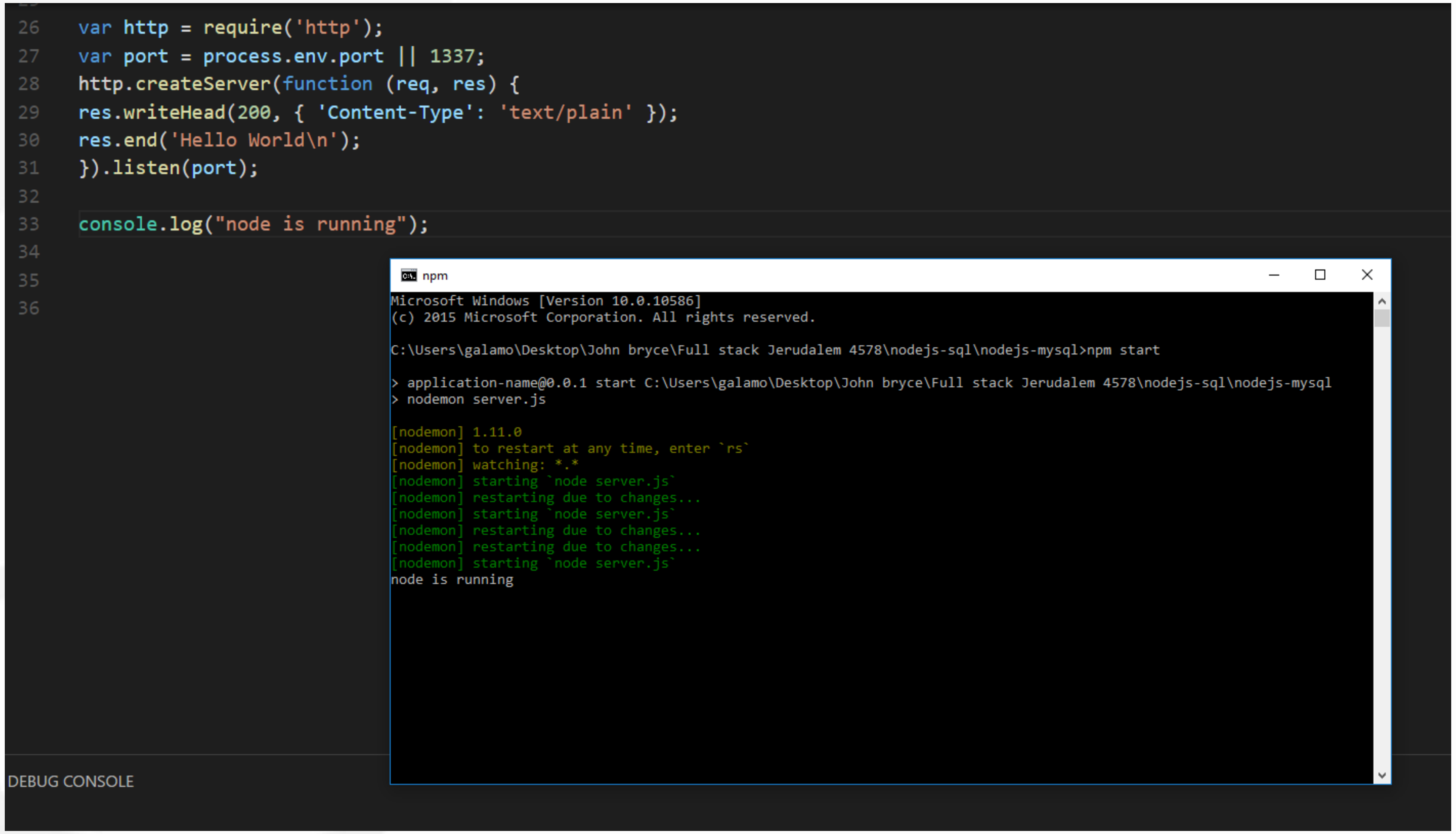
- 🤓 Express – minimal package for API
- 🤓 Nodemon
- 🤓 Zod
- 🤓 Cors

Nodemon

```
npm install -g nodemon
```

- Configure : "start": "nodemon server.js",
- Automatic restarting of application.
- Detects default file extension to monitor.
- Default support for node & coffeescript, but easy to run any executable (such as python, make, etc).
- Ignoring specific files or directories.
- Watch specific directories.
- Works with server applications or one time run utilities and REPLs.

Nodemon



```
var http = require('http');
var port = process.env.port || 1337;
http.createServer(function (req, res) {
res.writeHead(200, { 'Content-Type': 'text/plain' });
res.end('Hello World\n');
}).listen(port);

console.log("node is running");
```

DEBUG CONSOLE

npm

```
Microsoft Windows [Version 10.0.10586]
(c) 2015 Microsoft Corporation. All rights reserved.

C:\Users\galamo\Desktop\John bryce\Full stack Jerudalem 4578\nodejs-sql\nodejs-mysql>npm start

> application-name@0.0.1 start C:\Users\galamo\Desktop\John bryce\Full stack Jerudalem 4578\nodejs-sql\nodejs-mysql
> nodemon server.js

[nodemon] 1.11.0
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: *.*
[nodemon] starting `node server.js`
[nodemon] restarting due to changes...
[nodemon] starting `node server.js`
[nodemon] restarting due to changes...
[nodemon] restarting due to changes...
[nodemon] starting `node server.js`
node is running
```

Running the server from console

- Start a new cmd window
- cd to the server.js folder
- write ‘node server.js’
- The server is now up and listening to requests.

Express

- Minimal framework for web applications.
- HTTP Utility, methods and middleware
- Express provides a thin layer of fundamental web application features
- Based on middlewares

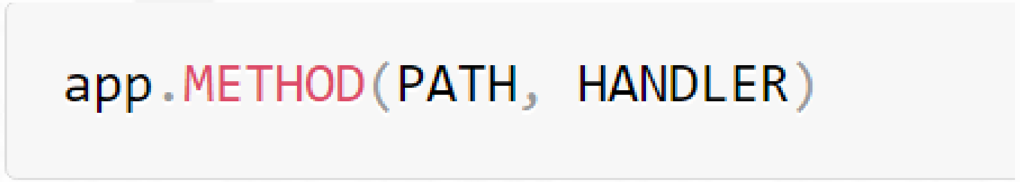
```
npm install express
```

Express

- The app object denotes the 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 (/) or route
- 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

Basic Routing

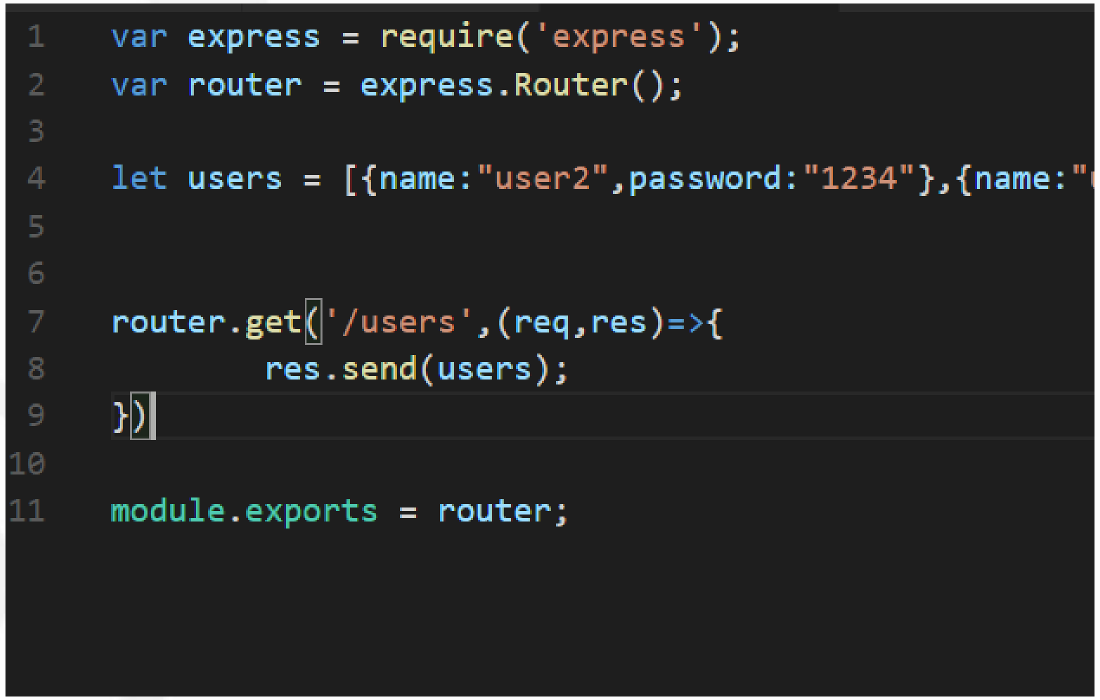
- Routing refers to determining how an application responds to a client request to a particular endpoint, which is a URI (or path) and a specific HTTP request method (GET, POST, and so on).
- Each route can have one or more handler functions, which are executed when the route is matched.
- Route definition takes the following structure:
  - Method: get , post…
  - Path: mapped URI
  - Handler: function(req,res)



```
app.METHOD(PATH, HANDLER)
```

Basic Routing

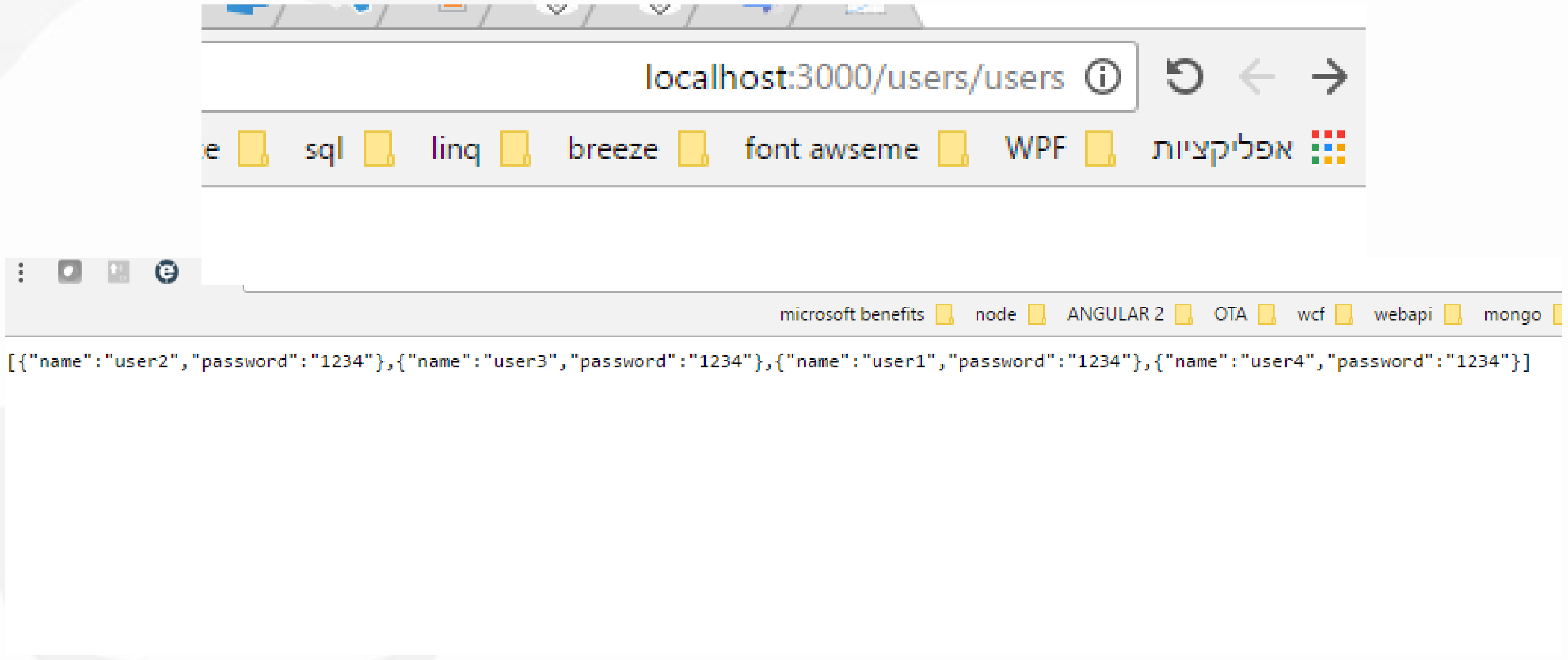
- The following examples illustrate defining simple routes.
- Respond with users array as “users” routing (get request):



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

let users = [{name:"user2",password:"1234"},{name:"


router.get('/users',(req,res)=>{
        res.send(users);
})

module.exports = router;
```

Making HTTP Request

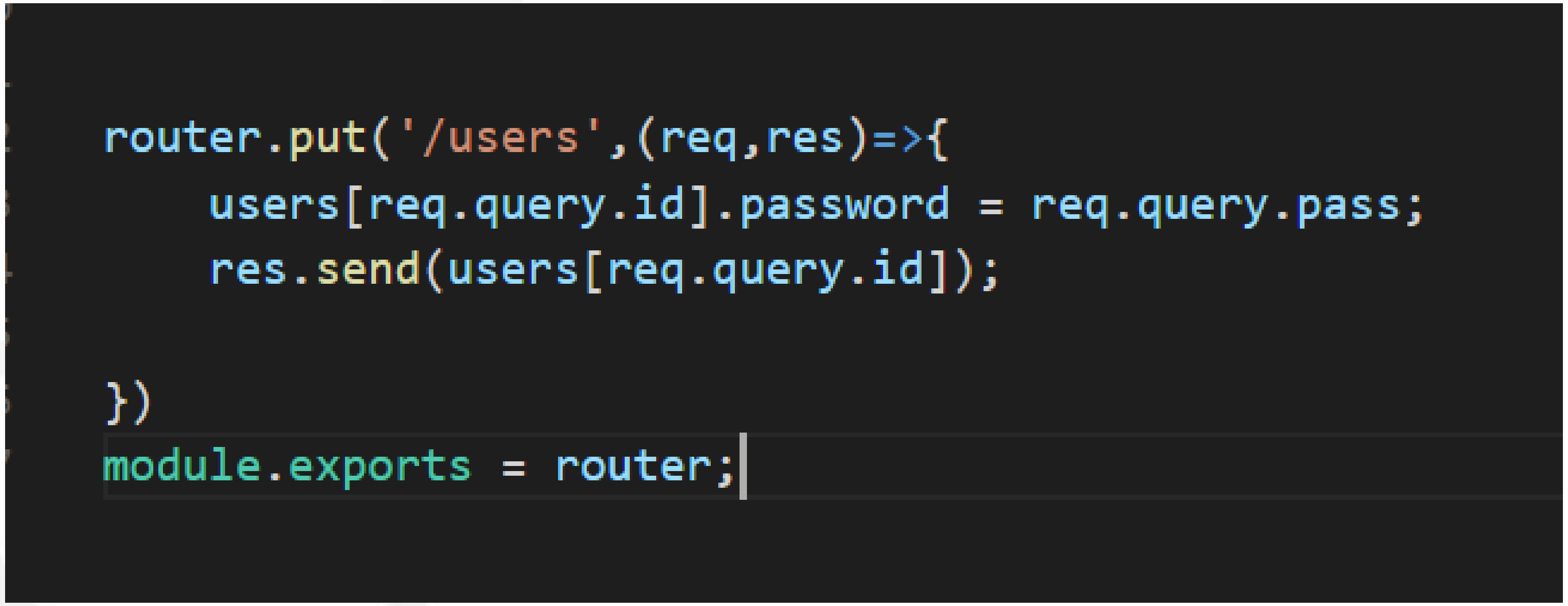
Respond with users array as “users” routing (get request):



```
[{"name":"user2","password":"1234"},{"name":"user3","password":"1234"},{"name":"user1","password":"1234"},{"name":"user4","password":"1234"}]
```

Put example

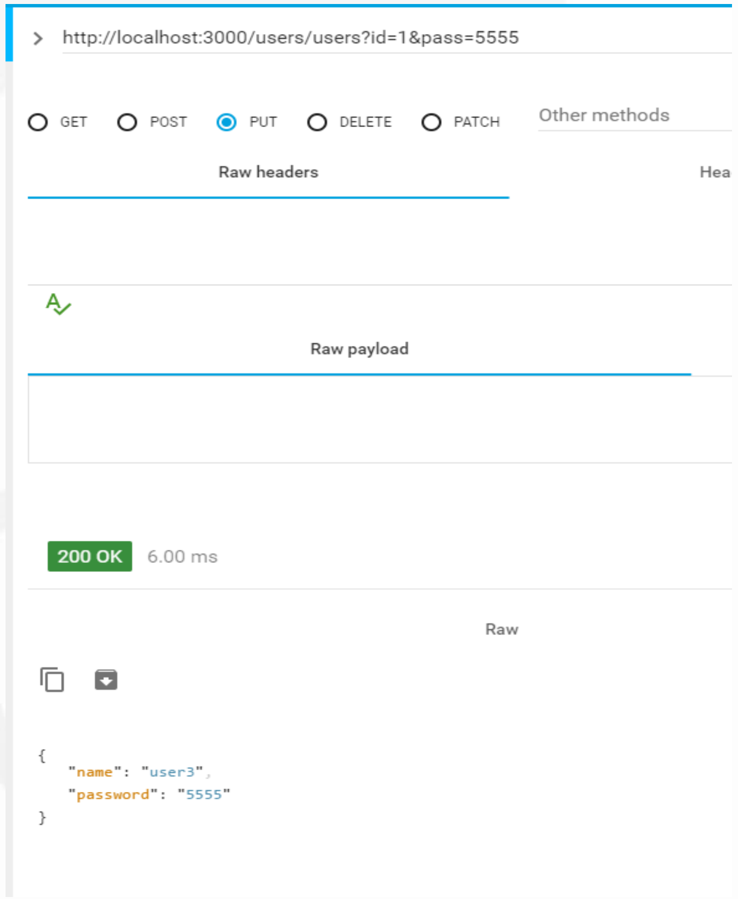
Respond to a PUT request to the /user route(update password):



```
router.put('/users',(req,res)=>{
    users[req.query.id].password = req.query.pass;
    res.send(users[req.query.id]);

})
module.exports = router;
```

Postman

Respond to a PUT request to the /user route:



http://localhost:3000/users/users?id=1&pass=5555

GET POST [✓ PUT] DELETE PATCH Other methods

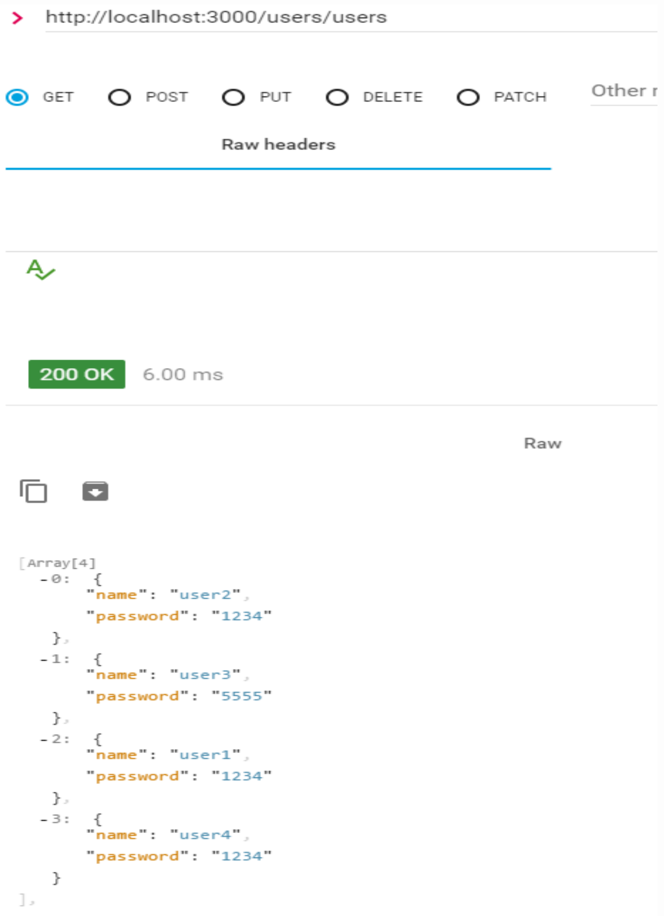
Raw headers

Raw payload

200 OK 6.00 ms

Raw

```
{
    "name": "user3",
    "password": "5555"
}
```



http://localhost:3000/users/users

[✓ GET] POST PUT DELETE PATCH Other r

Raw headers

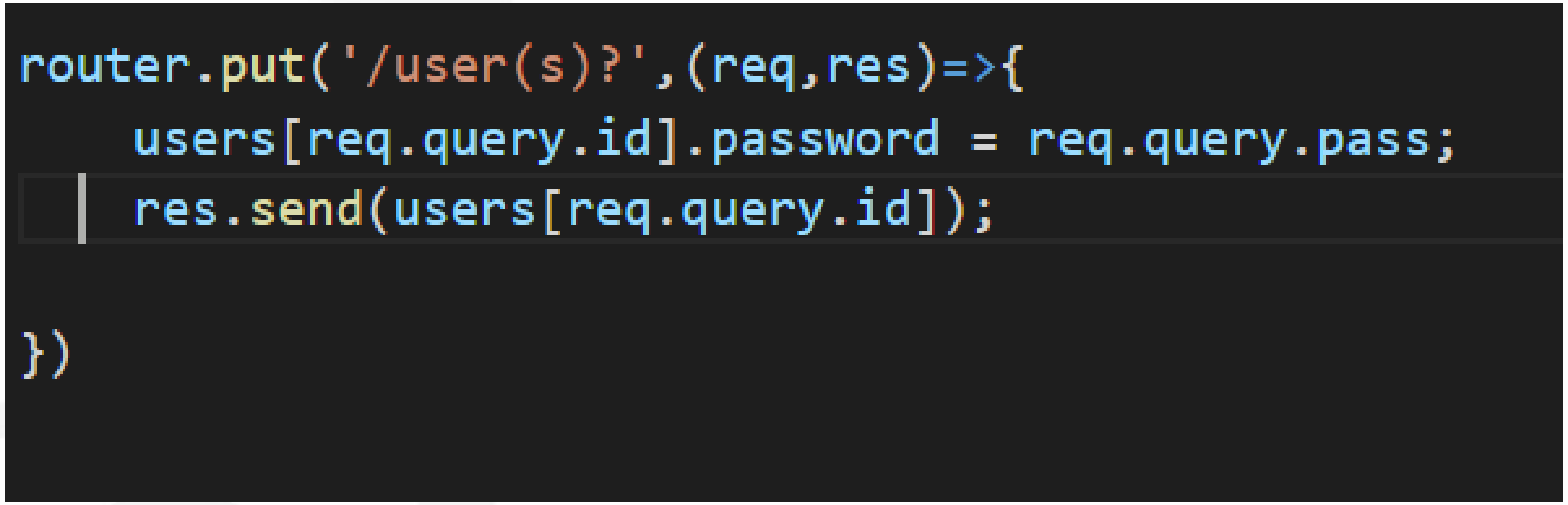
200 OK 6.00 ms

Raw

```
[Array[4]
  -0:  {
      "name": "user2",
      "password": "1234"
    },
  -1:  {
      "name": "user3",
      "password": "5555"
    },
  -2:  {
      "name": "user1",
      "password": "1234"
    },
  -3:  {
      "name": "user4",
      "password": "1234"
    }
],
```

Special route

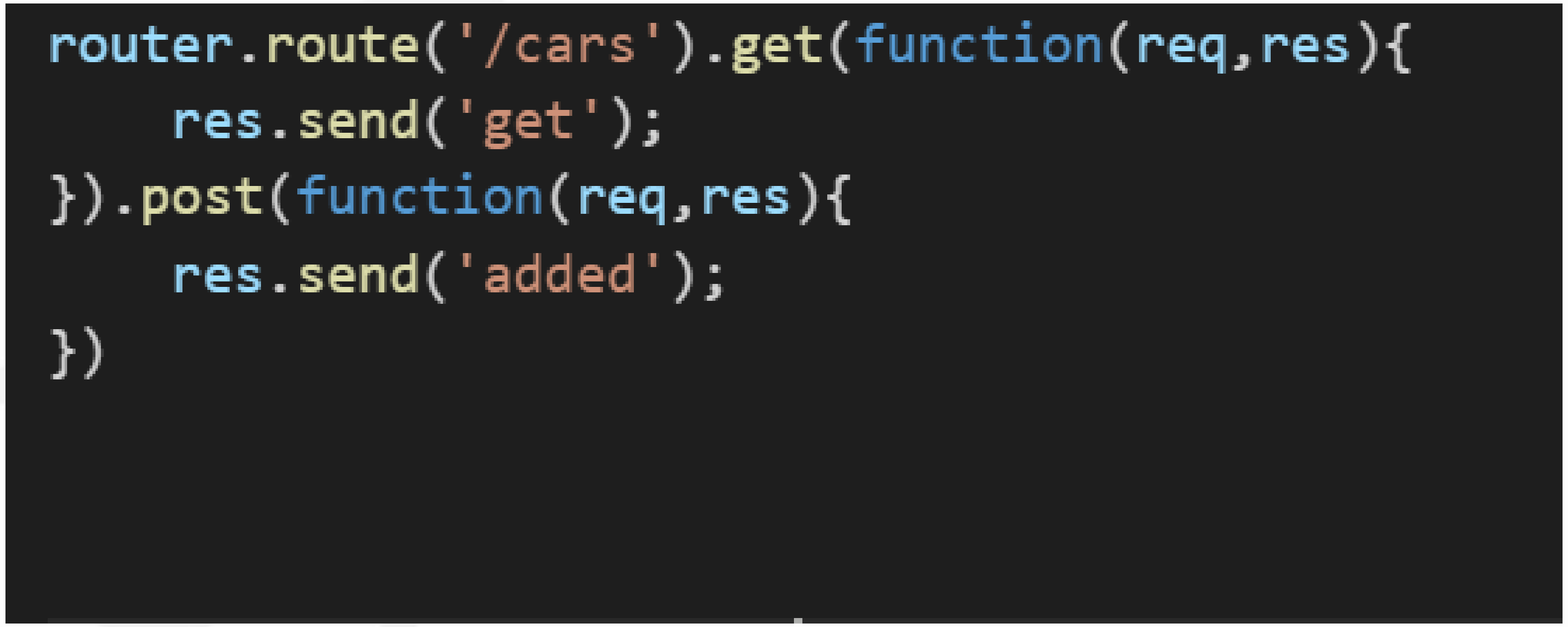
You can use regular expressions:



```
router.put('/user(s)?',(req,res)=>{
    users[req.query.id].password = req.query.pass;
    res.send(users[req.query.id]);

})
```

app.route()

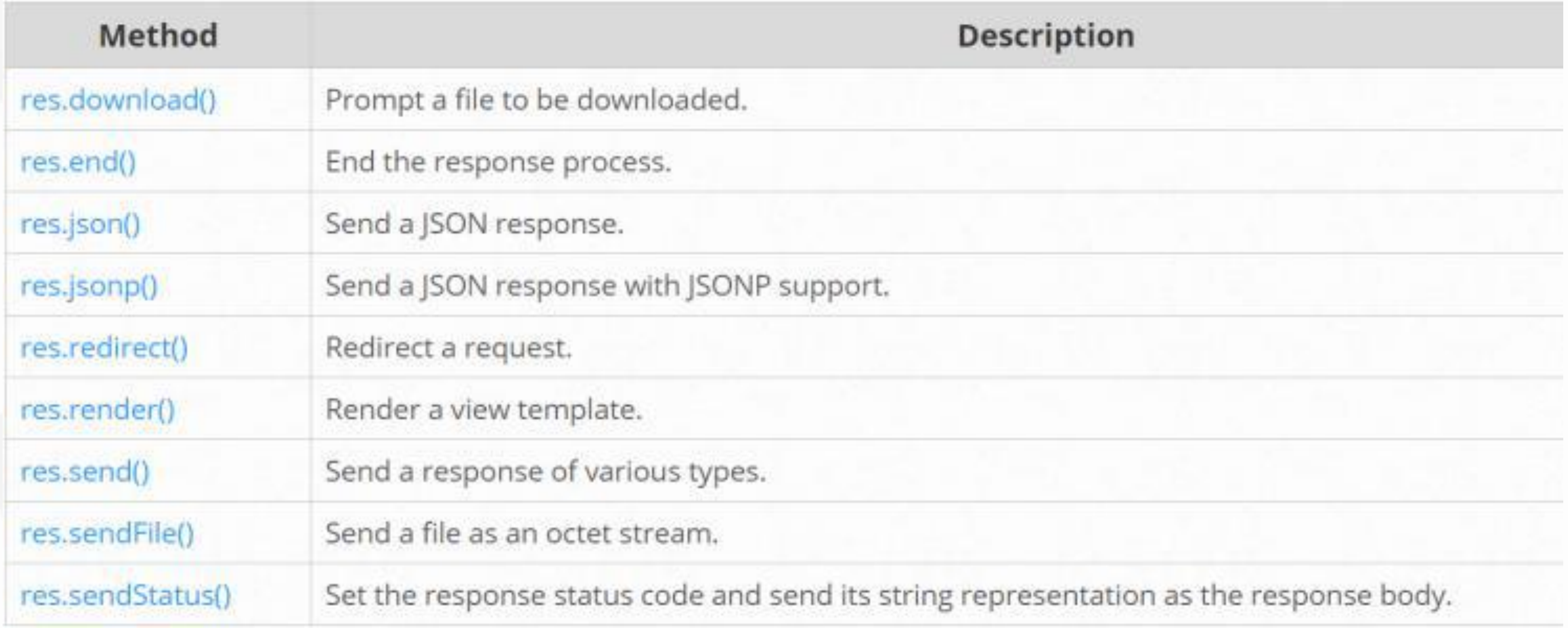
You can create chainable route handlers for a route path by using app.route().



```
router.route('/cars').get(function(req,res){
    res.send('get');
}).post(function(req,res){
    res.send('added');
})
```

Response Methods

The methods on the response object (res) send a
response to the client, and terminate the request-response cycle



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.

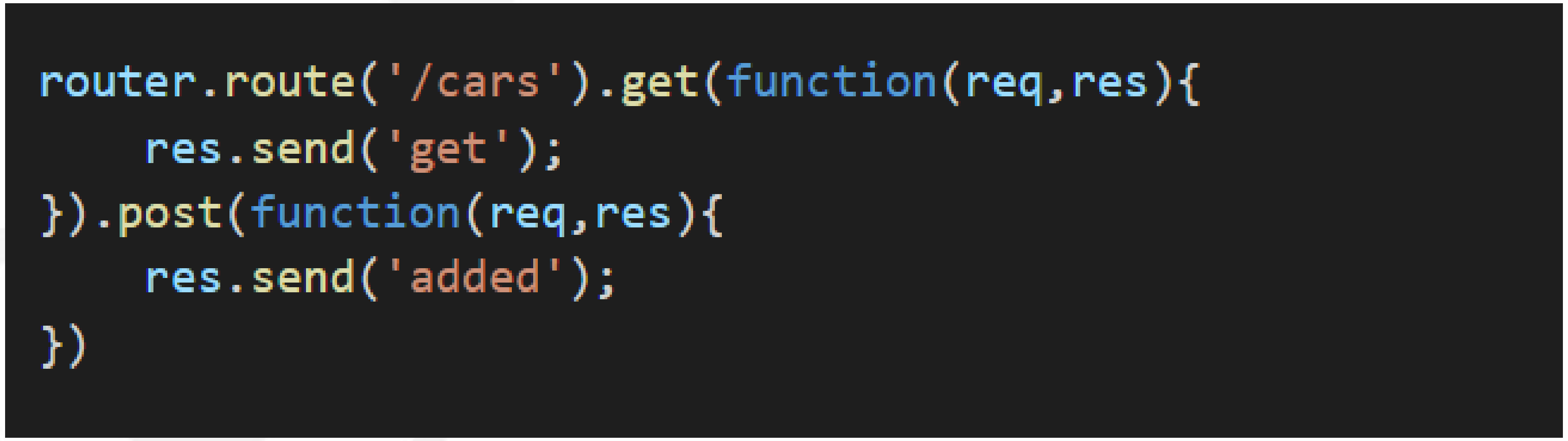
res.json()

Sends a JSON response. This method sends a response (with the correct content-type) that is the parameter converted to a JSON string using JSON.stringify().

The parameter can be any JSON type, including object, array, string, Boolean, or number

res.send()

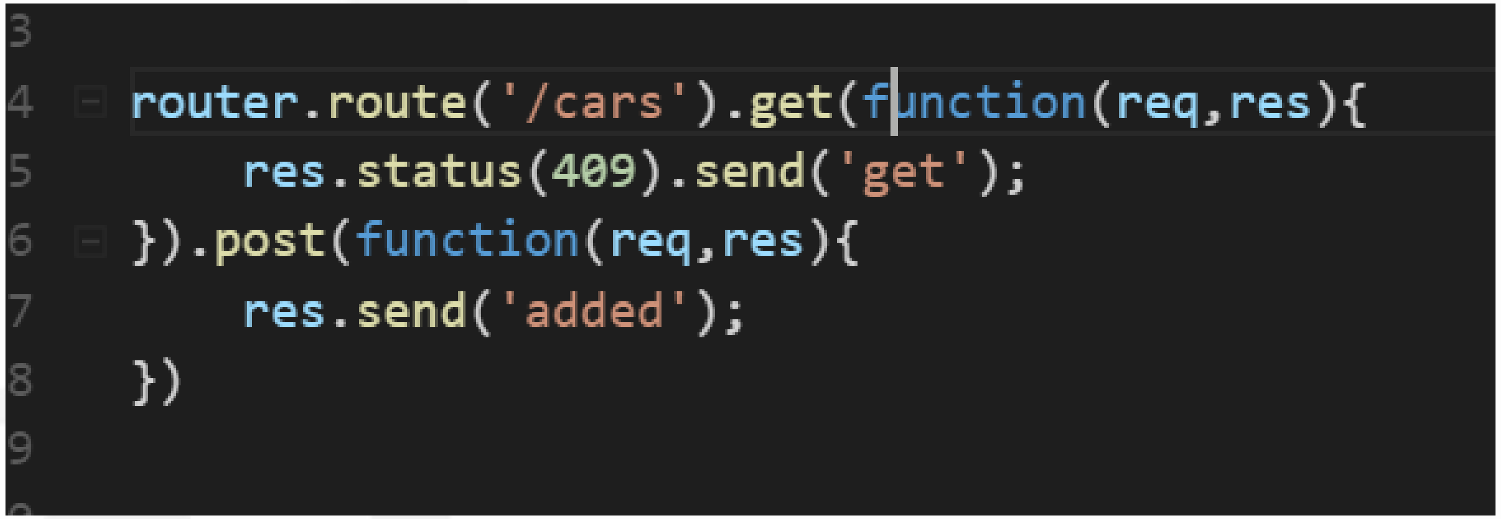
- Sends the HTTP response.
- The body parameter can be a Buffer object, a String, an object, or an Array.
- Express default representation is JSON.



```
router.route('/cars').get(function(req,res){
    res.send('get');
}).post(function(req,res){
    res.send('added');
})
```

res.send()

Chaining status code and response body.



```
router.route('/cars').get(function(req,res){
    res.status(409).send('get');
}).post(function(req,res){
    res.send('added');
})
```

res.set()

Sets the response’s HTTP header field to value.

To set multiple fields at once, pass an object as the Parameter.

```
res.set('Content-Type', 'text/plain');

res.set({
  'Content-Type': 'text/plain',
  'Content-Length': '123',
  'ETag': '12345'
});
```

res.sendFile()

- supported by Express v4.8.0 onwards.
- Transfers the file at the given path.
- Sets the Content-Type response HTTP header field based on the filename’s extension.
- Unless the root option is set in the options object, path must be an absolute path to the file.

res.sendFile()

- Two equivalent options:
  - res.sendFile(path.join(__dirname, '../public', 'index1.html'));
  - res.sendFile('index1.html', { root: path.join(__dirname, '../public');

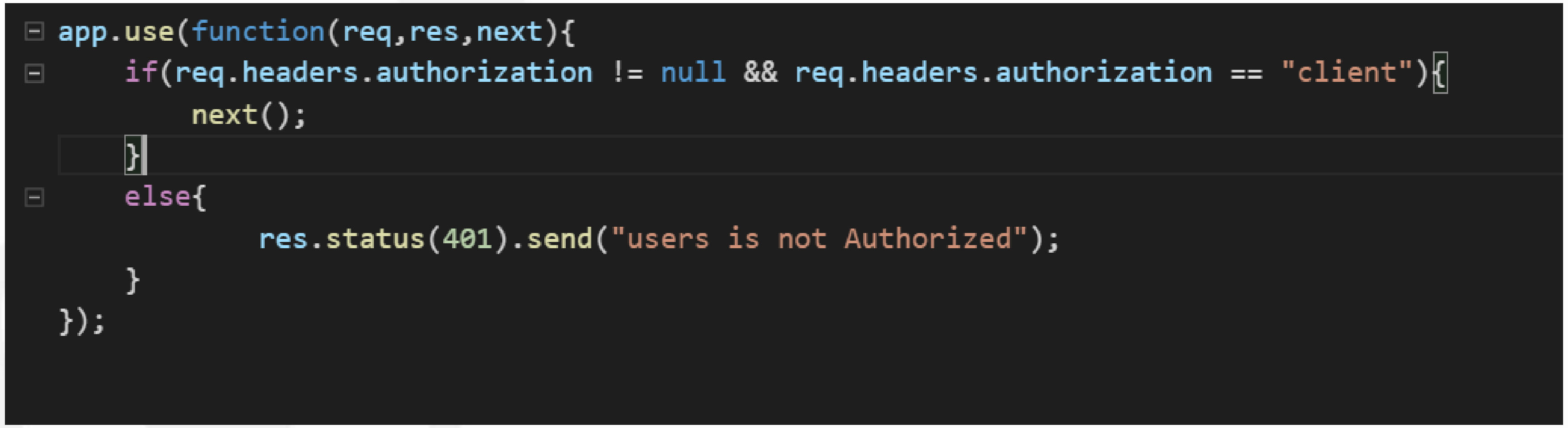
- __dirname returns the directory that the currently executing script is in.

Writing Middleware

- Middleware functions are functions that have access to the request and response objects, and the next middleware function in the request-response cycle.
- 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 in the stack.
  - Authentication and Authorization validations.

Middleware Function Example

Middleware function that checks whether the request header contains “Authorization header” and validate the value to be “client”.



```
app.use(function(req,res,next){
    if(req.headers.authorization != null && req.headers.authorization == "client"){
        next();
    }
    else{
            res.status(401).send("users is not Authorized");
    }
});
```

Router

- An isolated instance of middleware and routes.
- You can think of it as a “mini-application,” capable only of performing middleware and routing functions.
- The top-level express object has a Router() method that creates a new router object.
- You can add middleware and HTTP method routes (such as get, put, post) to it just like an application.
- New in Express 4.0

Validate your API - Joi

powerful schema description language and data validator for JavaScript

Validate your API - Joi

```
npm install @hapi/joi
```

https://hapi.dev/family/joi/

Authenticate - JsonWebToken

- Creating JSON Based access token
- Using private key or some other secret
- Sign the JSON and send it to the client
- Verify claims about the client

Authenticate - JsonWebToken

```
npm install jsonwebtoken
```

https://www.npmjs.com/package/jsonwebtoken

NodeJS Events

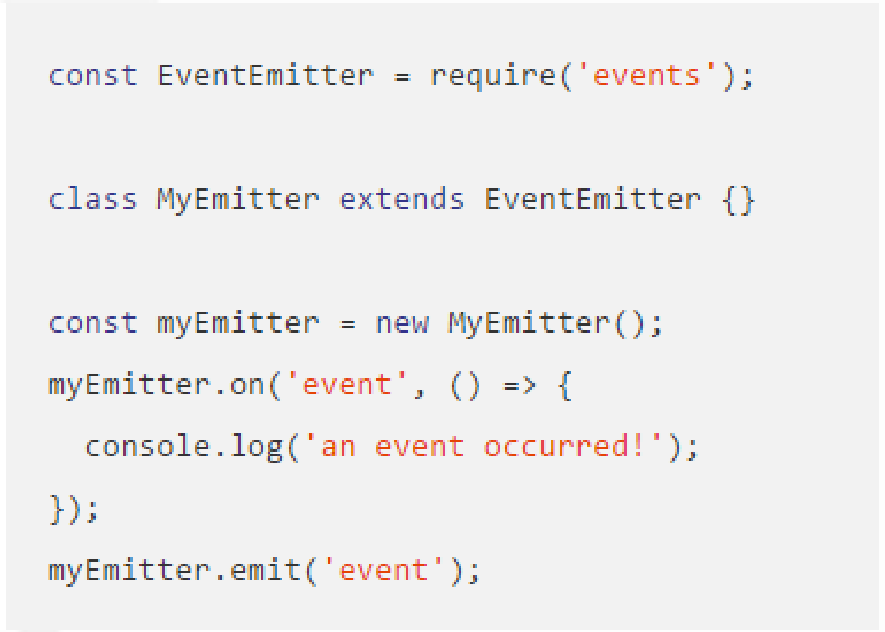
- Node.js core API is built around an asynchronous event-driven architecture in which objects called "emitters” periodically emit named events that cause function objects ("listeners") to be called.
- For instance: a net.Server object emits an event each time a peer connects to it; a stream emits an event whenever data is available to be read.

NodeJS Events

- All objects that emit events are instances of the EventEmitter class.
- These objects expose an eventEmitter.on() function that allows one or more functions to be attached to named events emitted by the object.
- When the EventEmitter object emits an event, all of the Functions attached to that specific event are called synchronously.

NodeJS Events

- The following example shows a simple EventEmitter.



```
const EventEmitter = require('events');

class MyEmitter extends EventEmitter {}

const myEmitter = new MyEmitter();
myEmitter.on('event', () => {
  console.log('an event occurred!');
});
myEmitter.emit('event');
```

React

JavaScript framework for building various Web and Mobile Applications

React

✓ **Advantages**

- High performance
- Desktop and mobile based applications
- Easy to use – write your apps faster

React

- React is a UI library developed by Facebook
- Declarative
- Creating Interactive, stateful and reusable UI components
- Support client and server side rendering

React ?

- ✓ **Fast**
  - Apps made in React can handle complex updates and still feel quick and responsive
- ✓ **Modular**
  - Instead of writing large, dense files of code, you can write many smaller, reusable files
- ✓ **Scalable**
  - Large programs that display a lot of changing data are where React performs best
- ✓ **Flexible**
  - You can use React for interesting projects that have nothing to do with making a web app

vite

Structure

```
my-app
├── README.md
├── node_modules
├── package.json
├── .gitignore
├── public
│   ├── favicon.ico
│   ├── index.html
│   └── manifest.json
└── src
    ├── App.css
    ├── App.js
    ├── App.test.js
    ├── index.css
    ├── index.js
    ├── logo.svg
    └── registerServiceWorker.js
```

Virtual DOM

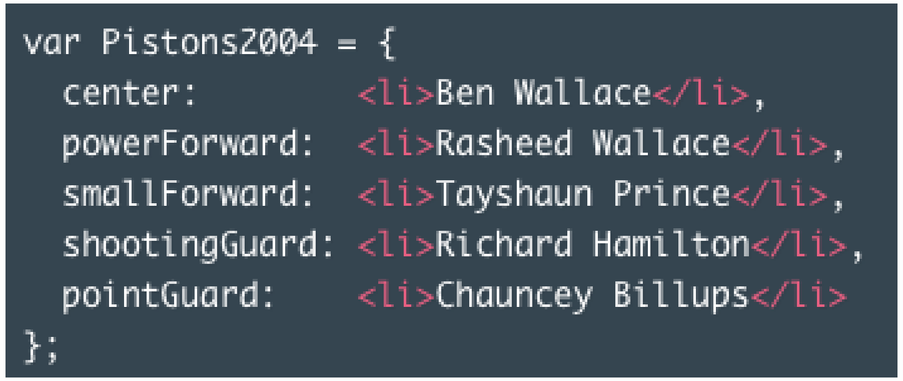
- Selectively renders subtrees of nodes based upon state changes
- It does the least amount of DOM manipulation possible in order to keep your components up to date

DOM Rendering

- ReactDOM.render makes changes by leaving the current DOM in place and simply updating the DOM elements that need to be updated.
- This smart DOM rendering is necessary for React to work in a reasonable amount of time because our application state changes a lot.
- Every time we change that state, we are going to rely on ReactDOM.render to efficiently re-render the UI.

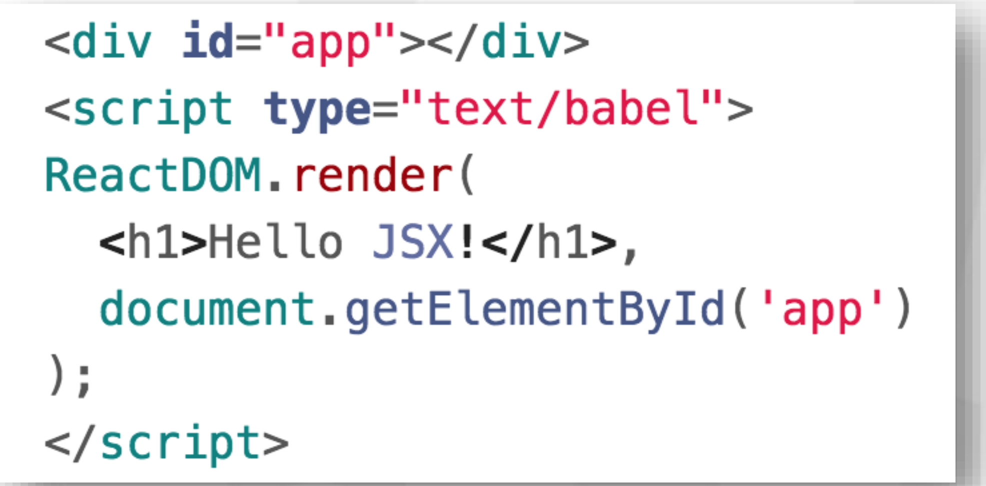
JSX

- Syntax extension for JavaScript
- It was written to be used with React
- JSX code looks a lot like HTML
  - var h1 = <h1>Hello world</h1>;
- JSX is not valid JavaScript
  - Web browsers can't read it!
  - Translation is needed



```
var Pistons2004 = {
  center:        <li>Ben Wallace</li>,
  powerForward:  <li>Rasheed Wallace</li>,
  smallForward:  <li>Tayshaun Prince</li>,
  shootingGuard: <li>Richard Hamilton</li>,
  pointGuard:    <li>Chauncey Billups</li>
};
```

Simple Example - JSX



```
<div id="app"></div>
<script type="text/babel">
ReactDOM.render(
  <h1>Hello JSX!</h1>,
  document.getElementById('app')
);
</script>
```

Simple Example - JSX

- Function
- Class

Component can accept inputs and render an HTML

```
function User(userName) {
return <h1>Hello, {props. userName}</h1>;
}
```

More about react

- Hooks
- State
- State
- Components lifecycle
- Selectors – memoization
- Redux – state management

DevGeekWeek 2024
INSPIRED BY INNOVATION

Thanks :)

See you next year

matrix

JOHN BRYCE
תלמדו הייטק. זה עובד!
a matrix company