CYBAGE
Delivering Value. Scientifically.

Presented and Authored by: Asfiya Khan

www.cybage.com

Agenda

Day 1
- Features
- Modules
- Global objects

Day 2
- File System
- Streams

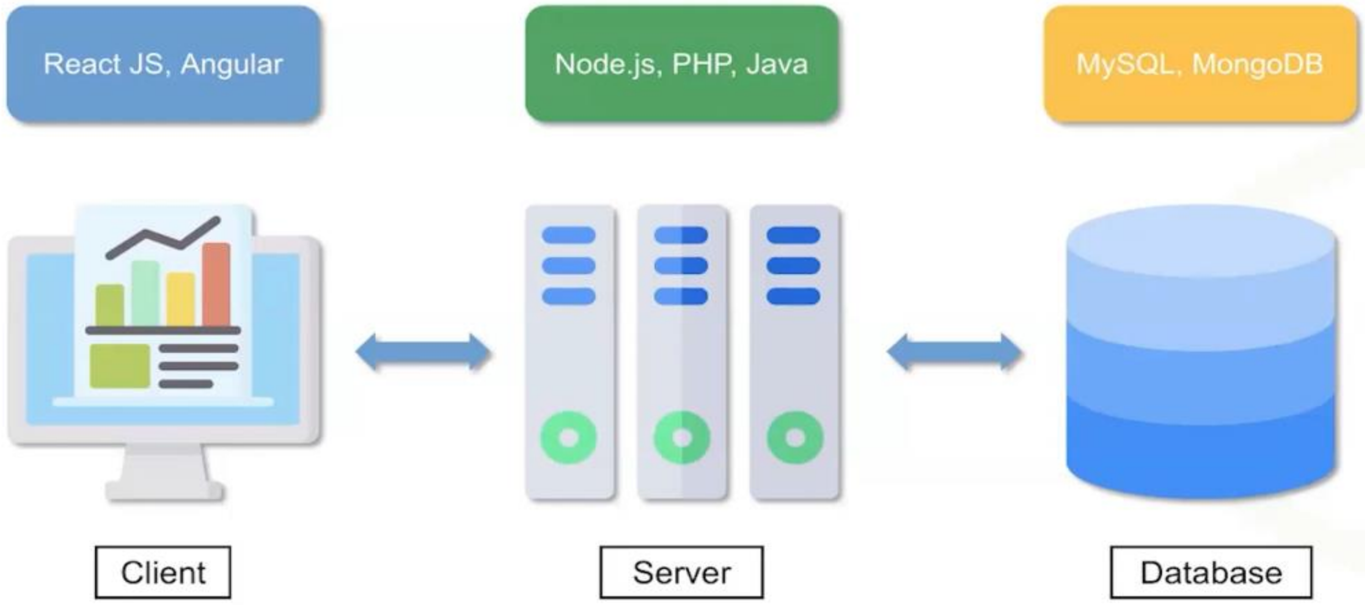
Day 3
- HTTP Module
- Express framework

Day 4
- RESTFul API
- MySQL

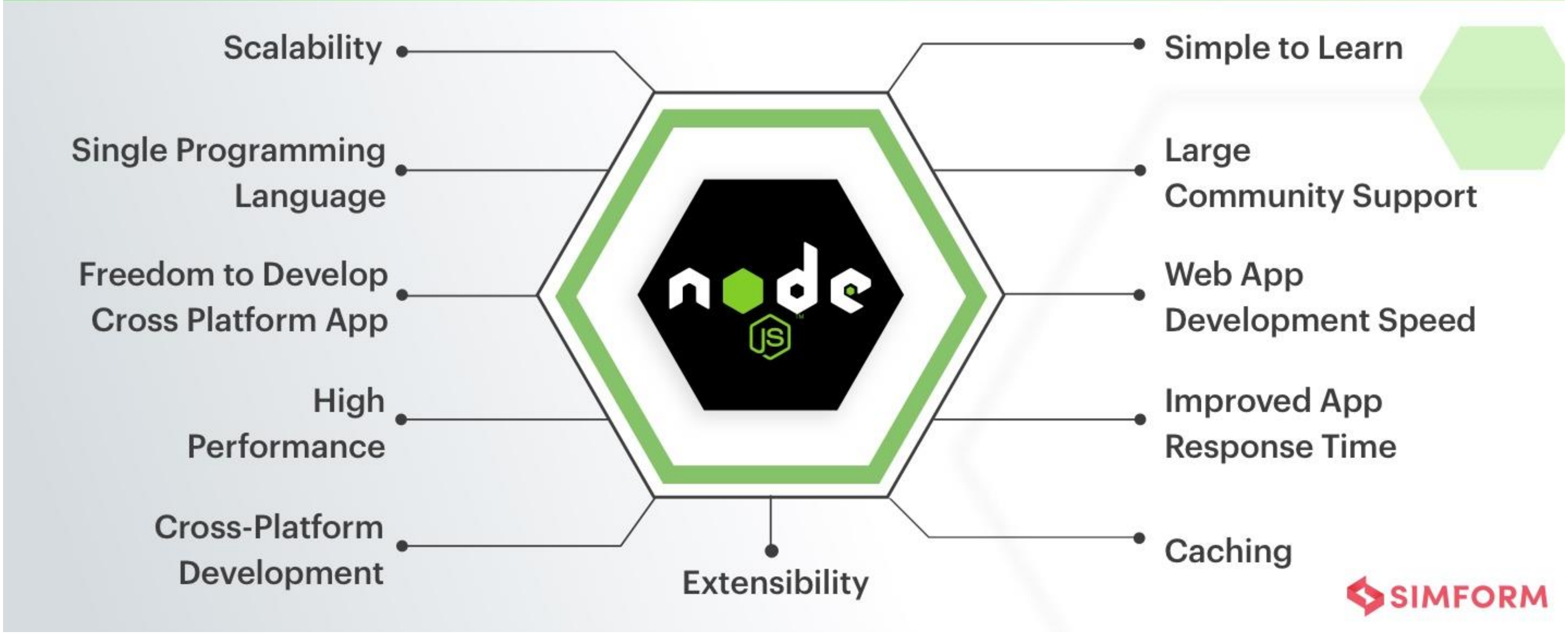
Day 5
- Error Handling
- Best practices

Introduction

- Server-side platform built on Google V8 Engine
- Build fast and scalable network applications
- Event-driven, non-blocking I/O model
- lightweight and efficient
- Open source, cross-platform for server-side applications
- Node.js = Runtime Environment + JavaScript Library


React JS, Angular
Node.js, PHP, Java
MySQL, MongoDB
Client
Server
Database

Node.js Advantages


Scalability
Single Programming Language
Freedom to Develop Cross Platform App
High Performance
Cross-Platform Development
Extensibility
Simple to Learn
Large Community Support
Web App Development Speed
Improved App Response Time
Caching
node JS
SIMFORM

Node JS Installation

https://nodejs.org/en/download/

Capabilities of Node.js

- Node.js can dynamically generate content and present it to web clients
- Node.js can do file operations like read, write, update, delete, etc
- Node.js can collect data from clients through forms
- Node.js can connect to various databases like MySQL, MongoDB, etc
- Node.js has built-in JSON module to work with JSON files

Major Implementation Areas

- I/O bound Applications
- Data Streaming Applications
- Data Intensive Real-time Applications
- JSON APIs based Applications
- Single Page Applications

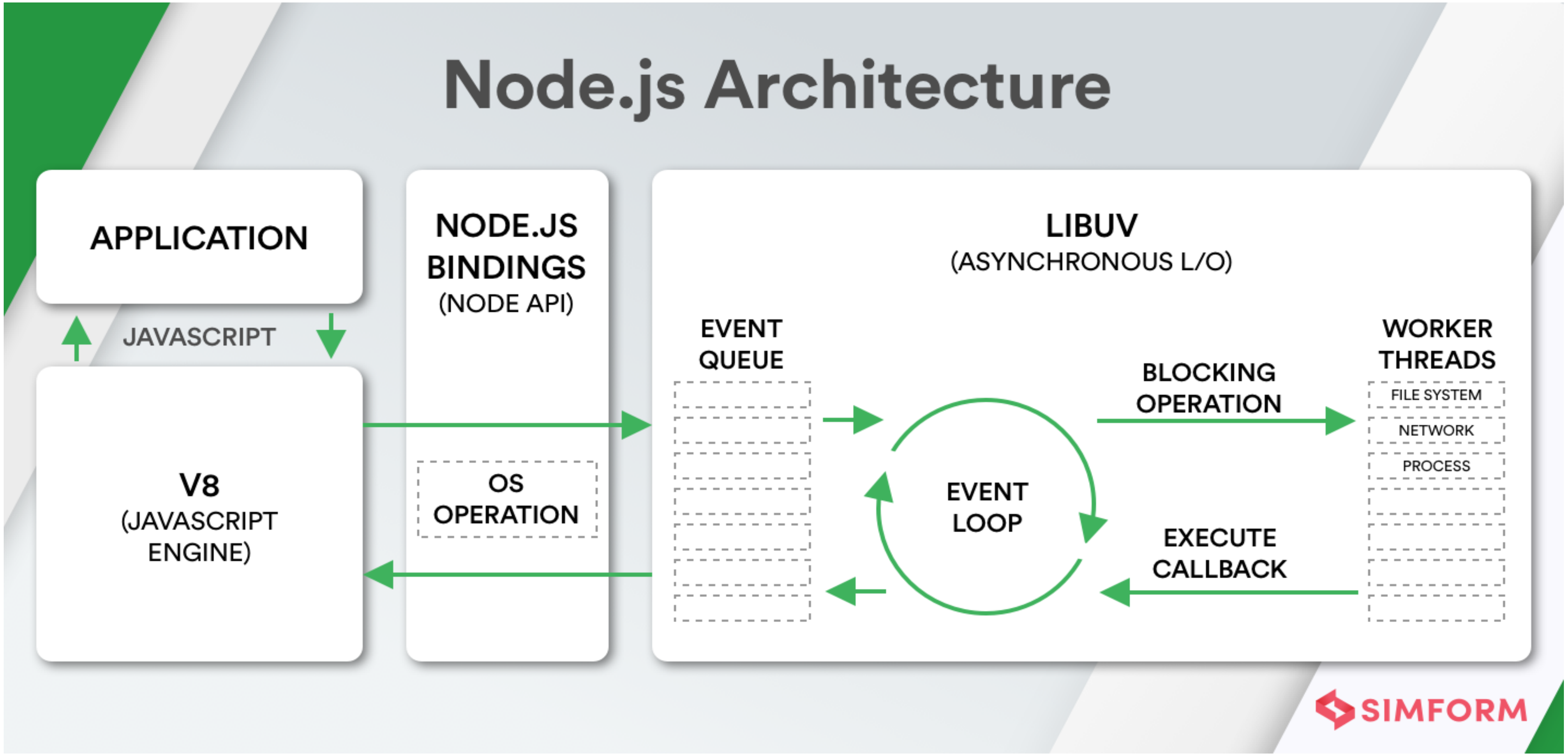
Major Clients

Walmart ebay PayPal

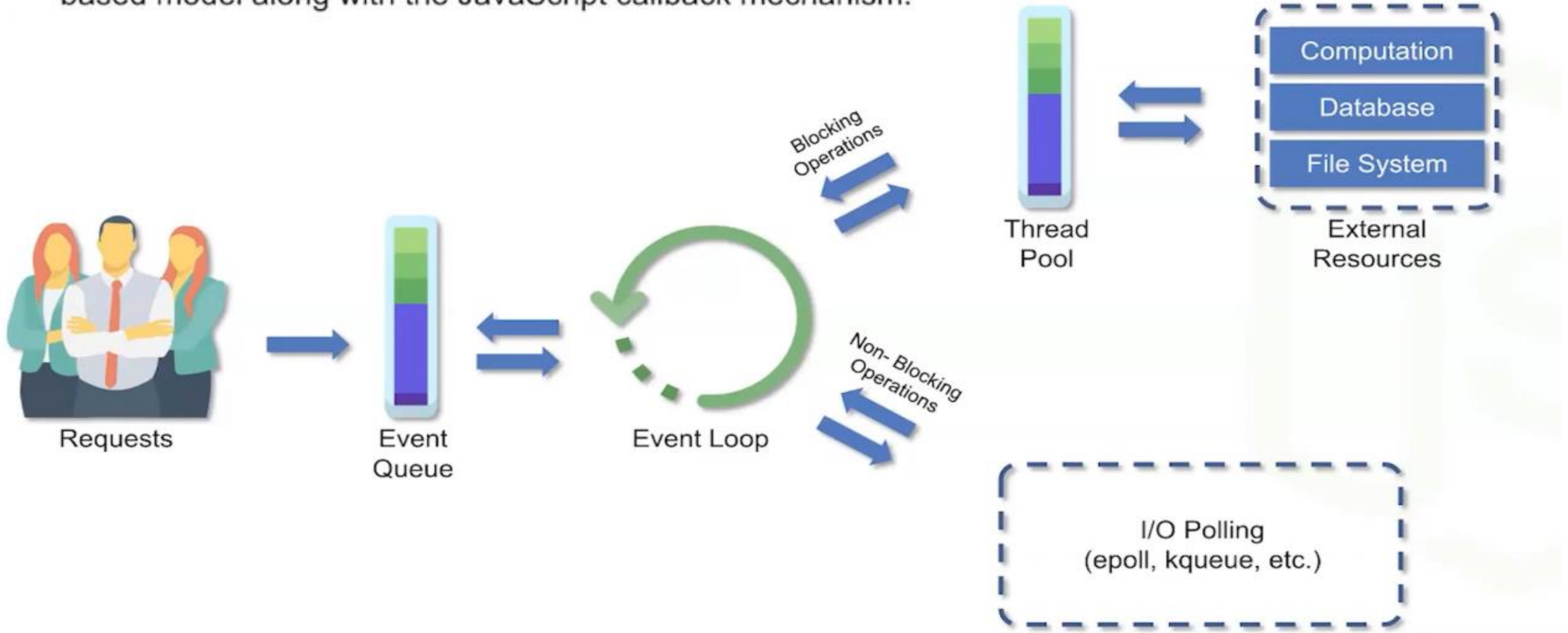
DOW JONES intuit NETFLIX

LinkedIn The New York Times Microsoft

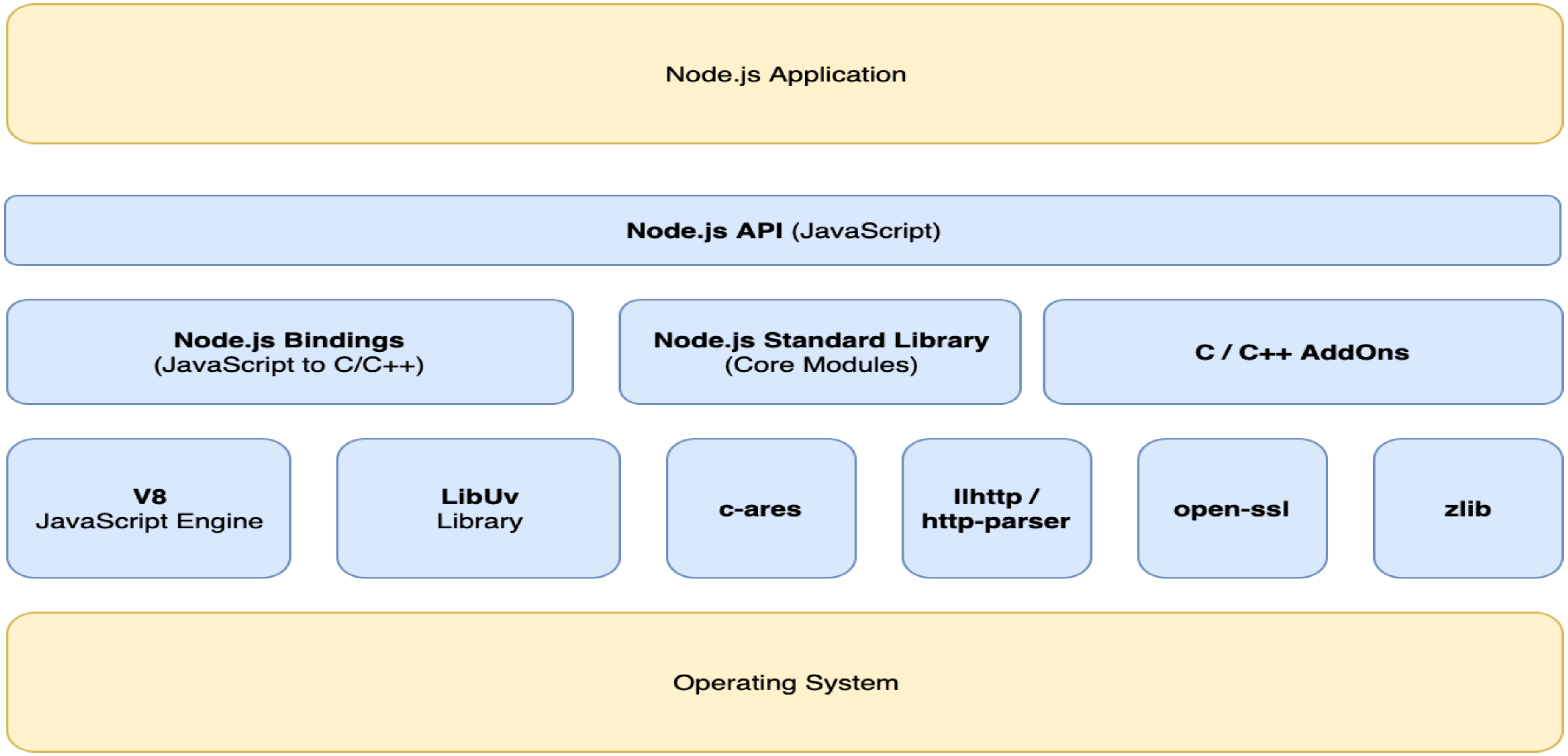
UBER YAHOO! Kingfisher


Node.js Architecture
APPLICATION
JAVASCRIPT
V8
(JAVASCRIPT ENGINE)
NODE.JS BINDINGS
(NODE API)
OS OPERATION
LIBUV
(ASYNCHRONOUS L/O)
EVENT QUEUE
EVENT LOOP
BLOCKING OPERATION
EXECUTE CALLBACK
WORKER THREADS
FILE SYSTEM
NETWORK
PROCESS
SIMFORM

- Node.js uses "Single Threaded Event Loop" architecture to handle multiple concurrent clients
- Node.js Processing model is based on the JavaScript Event based model along with the JavaScript callback mechanism.



Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine.



Npm

- npm is world’s largest software registry
- Used to share and borrow packages
- Share code with any npm user, anywhere
- Restrict code to specific developers
- Create Orgs to coordinate package maintenance, coding, and developers
- Manage multiple versions of code and code dependencies
- Update applications easily when underlying code is updated
- Find other developers who are working on similar problems and projects

Npm commands

- npm install
- Npm install package-name@version
- Npm install package-name@latest
- Npm list
- Npm config list
- npm config get prefix
- Install package in global mode: npm install package-name –global (C:\Users\<user-name>\AppData\Roaming\npm\node_modules)
- npm uninstall package-name
- npm update package-name
- Npm search package-name

Install package in local mode

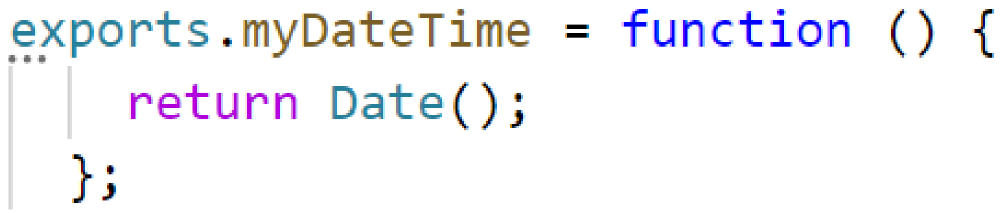
- Possible using package.json
- Create using npm init
- Install dependency using npm install package-name
- Save package as devDependency by specifying --save-dev flag
- Save package as production Dependency by specifying --save-prod flag
- devDependencies are packages used for development purposes

Modules

- Modules are reusable parts of code that usually export specific objects to be used in your Node.js programs
- Node.js provides many built-in modules
- You may also create your own module
- We can extend a built-in module, or override some of functionalities of module
- Different type of modules
- Core modules
- Local modules
- Third part modules

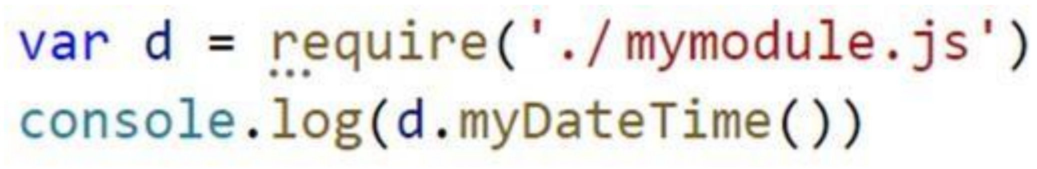
Local module

- Creating local module



```
exports.myDateTime = function () {
    return Date();
};
```

- Loading local module



```
var d = require('./mymodule.js')
console.log(d.myDateTime())
```

User defined module

- Calculator.js

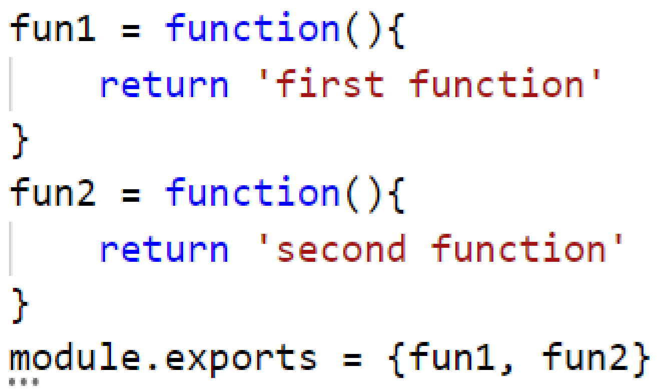
```
exports.add = function (a, b) {
  return a+b;
};

exports.subtract = function (a, b) {
  return a-b;
};

exports.multiply = function (a, b) {
  return a*b;
};
```

Cont…

- Usage

```
var calculator = require('./calculator');
var a=10, b=5;
console.log("Addition : "+calculator.add(a,b));
console.log("Subtraction : "+calculator.subtract(a,b));
console.log("Multiplication : "+calculator.multiply(a,b));
```

Multiple exports



```
fun1 = function(){
    return 'first function'
}
fun2 = function(){
    return 'second function'
}
module.exports = {fun1, fun2}
```

Extend module

- Can modify existing modules
- Steps
- include module : var newMod = require('<module_name>');
- Add function to module

```
 newMod.<newFunctionName> = function(function_parameters) {
     // function body
   };
```

- Re-export module: module.exports = newMod;

Extend module example

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

fs.printMessage = function(str) {
  console.log("Message from newly added function to the module");
  console.log(str);
}
module.exports = fs

fs.printMessage("Success");
```

Utility modules

N/w

- HTTP, NET, TLS/SSL, UDP, TTY
- URL, QueryString

Core

- STDIO, ReadLine, Stream
- FileSystem, Path
- OS, Process

Utils

- Debugger
- Utilities, ZLIB, Crypto
- Timers, Globals, Modules

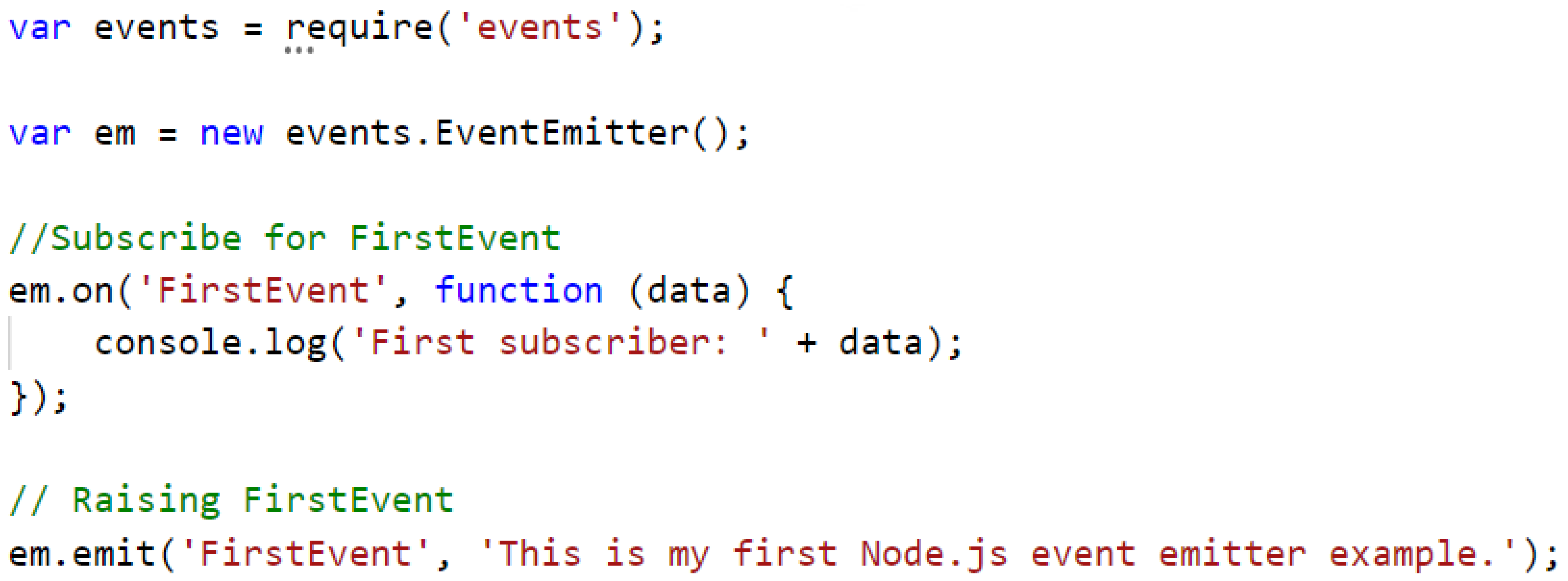
EventEmitters

- Node JS -Event driven programming model
- EventEmitters generate events
- EventHandlers(Listeners) handle events
- eventEmitters.on(nameOfEventToBind, eventHandlerFunction)
- eventEmitters.emit(eventName)

Events

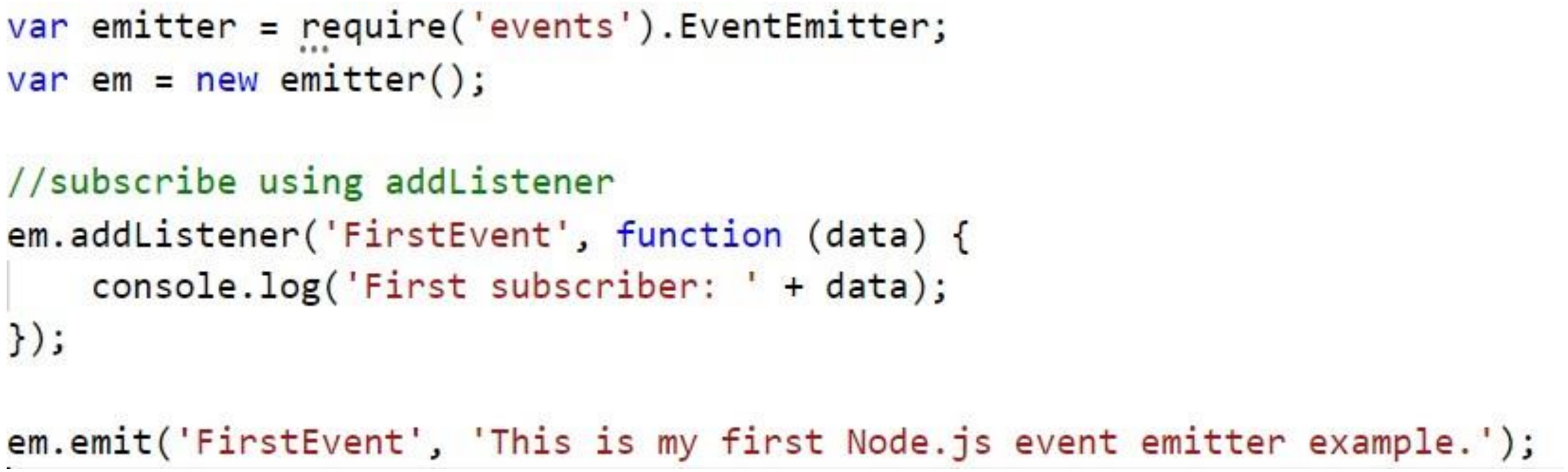
- on(event, listener) or addListener(event, listener)
- once(event, listener)
- removeListener(event, listener)
- removeAllListeners([event])
- setMaxListeners(n)
- listeners()
- emit(event, [arg1], [arg2],…)
- emitter.listenerCount(type)

EventEmitter



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

var em = new events.EventEmitter();

//Subscribe for FirstEvent
em.on('FirstEvent', function (data) {
    console.log('First subscriber: ' + data);
});

// Raising FirstEvent
em.emit('FirstEvent', 'This is my first Node.js event emitter example.');
```

EventEmitter



```
var emitter = require('events').EventEmitter;
var em = new emitter();

//subscribe using addListener
em.addListener('FirstEvent', function (data) {
    console.log('First subscriber: ' + data);
});

em.emit('FirstEvent', 'This is my first Node.js event emitter example.');
```

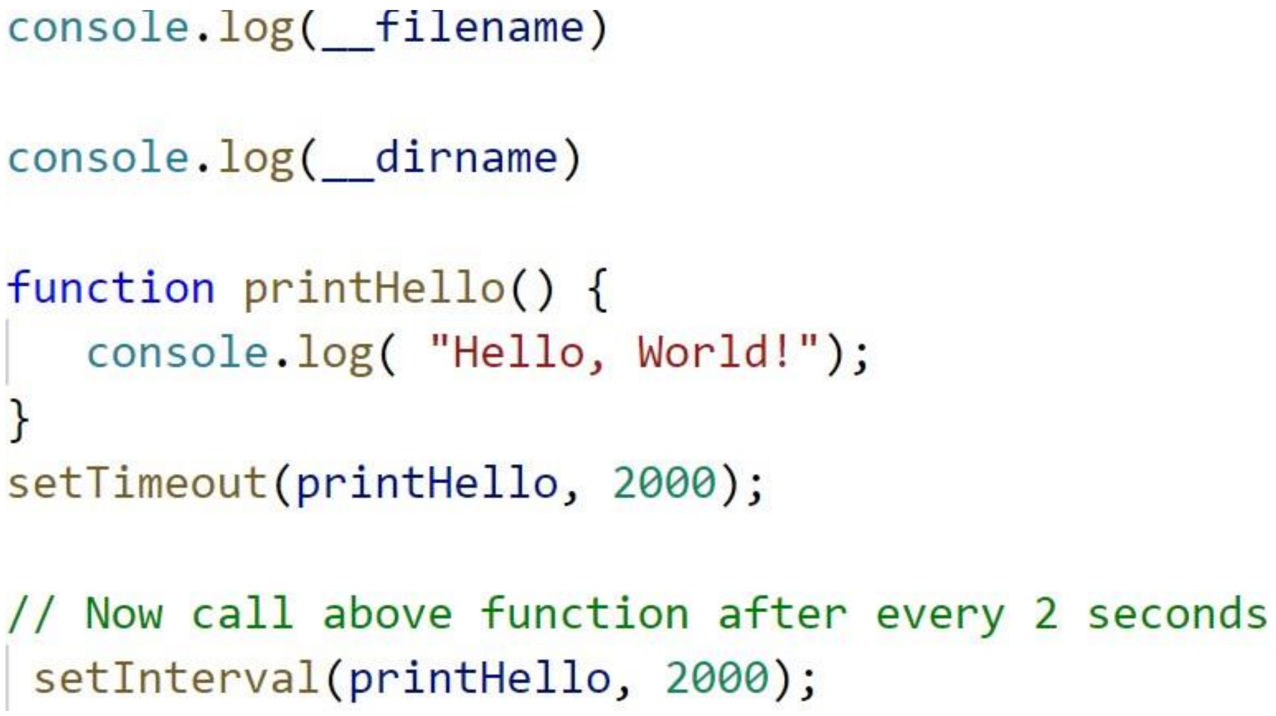
.on() vs addEventListener()

- Their functionalities are exactly the same, however, they can be used in different ways to make your code efficient
- You can use "removeListener" on addEventListener() but you cannot remove ".on(event, listener)" command

Global objects

- Global objects are available in all modules
- These objects are modules, functions, strings and object itself
- __filename
- __dirname
- setTimeout(cb, ms)
- clearTimeout(t)
- setInterval(cb, ms)

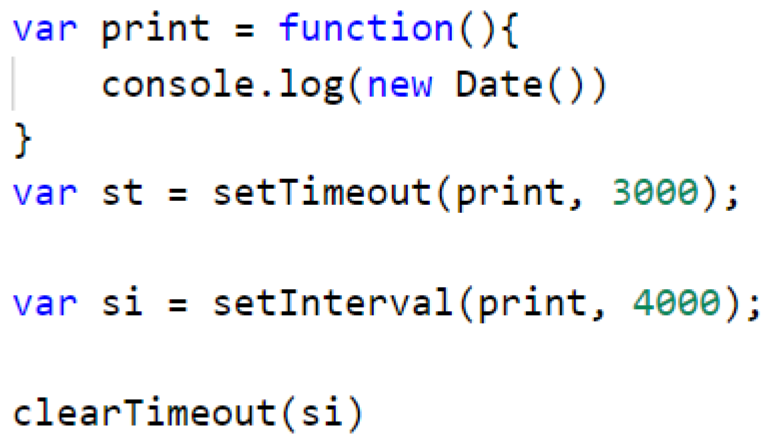
Global objects



```
console.log(__filename)

console.log(__dirname)

function printHello() {
   console.log( "Hello, World!");
}
setTimeout(printHello, 2000);

// Now call above function after every 2 seconds
 setInterval(printHello, 2000);
```

Cont...

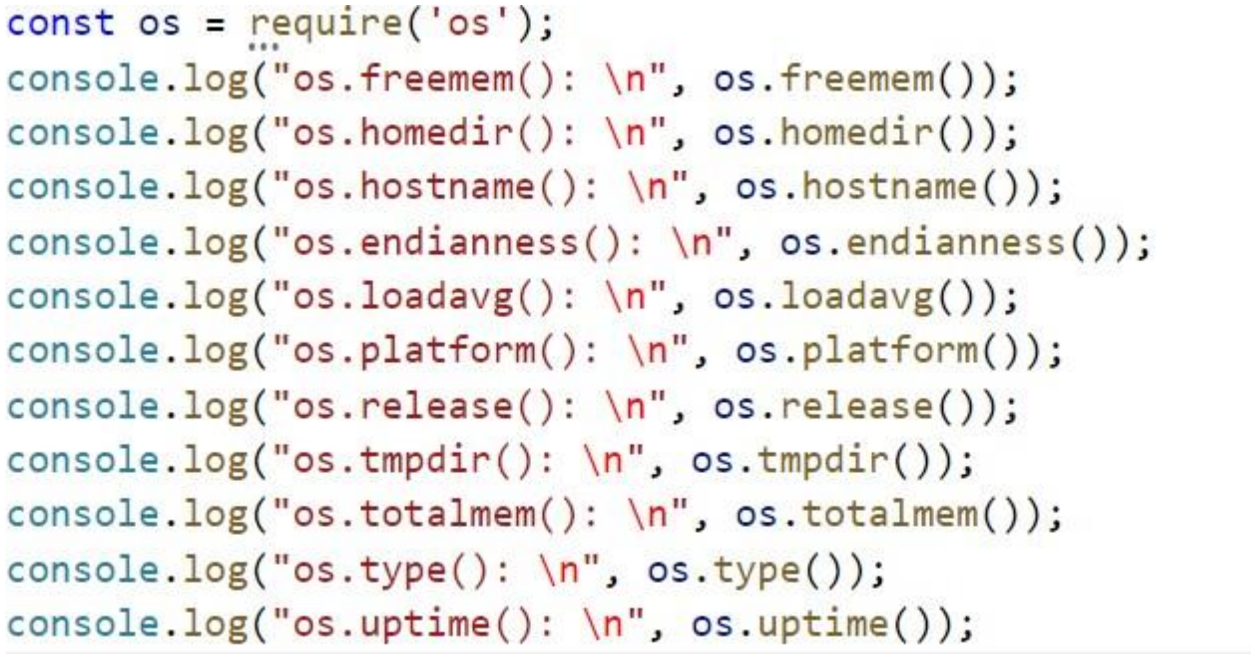
- clearTimeout



```
var print = function(){
    console.log(new Date())
}
var st = setTimeout(print, 3000);

var si = setInterval(print, 4000);

clearTimeout(si)
```

Node.js OS

- Provides basic operating-system related functions



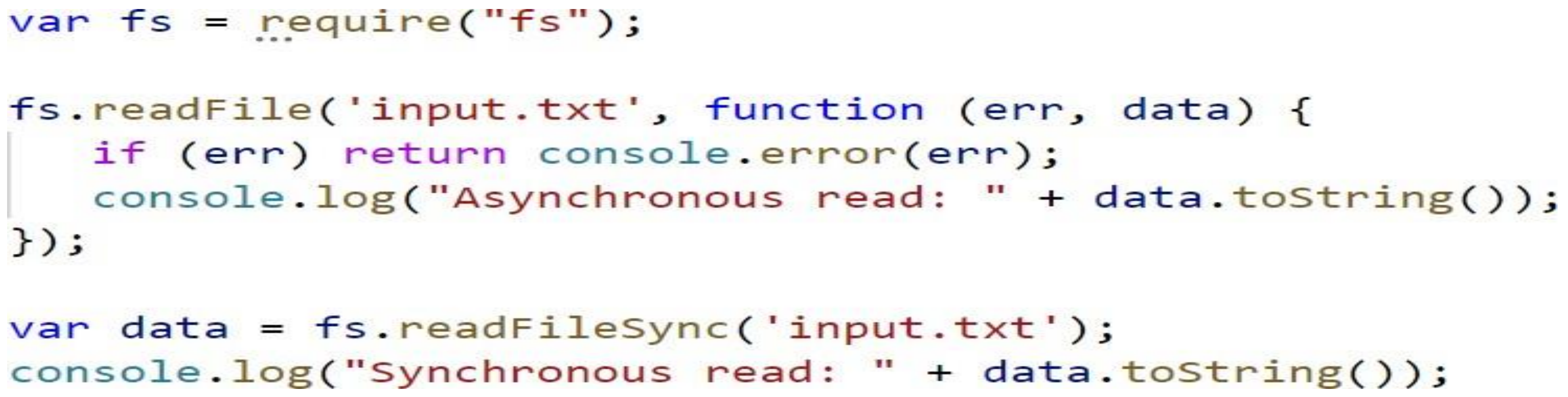
```
const os = require('os');
console.log("os.freemem(): \n", os.freemem());
console.log("os.homedir(): \n", os.homedir());
console.log("os.hostname(): \n", os.hostname());
console.log("os.endianness(): \n", os.endianness());
console.log("os.loadavg(): \n", os.loadavg());
console.log("os.platform(): \n", os.platform());
console.log("os.release(): \n", os.release());
console.log("os.tmpdir(): \n", os.tmpdir());
console.log("os.totalmem(): \n", os.totalmem());
console.log("os.type(): \n", os.type());
console.log("os.uptime(): \n", os.uptime());
```

File system

- Implements using POSIX functions
- Methods are synchronous as well as asynchronous
- Asynchronous take last parameter as completion function
- First parameter of callback function is error

CYBAGE

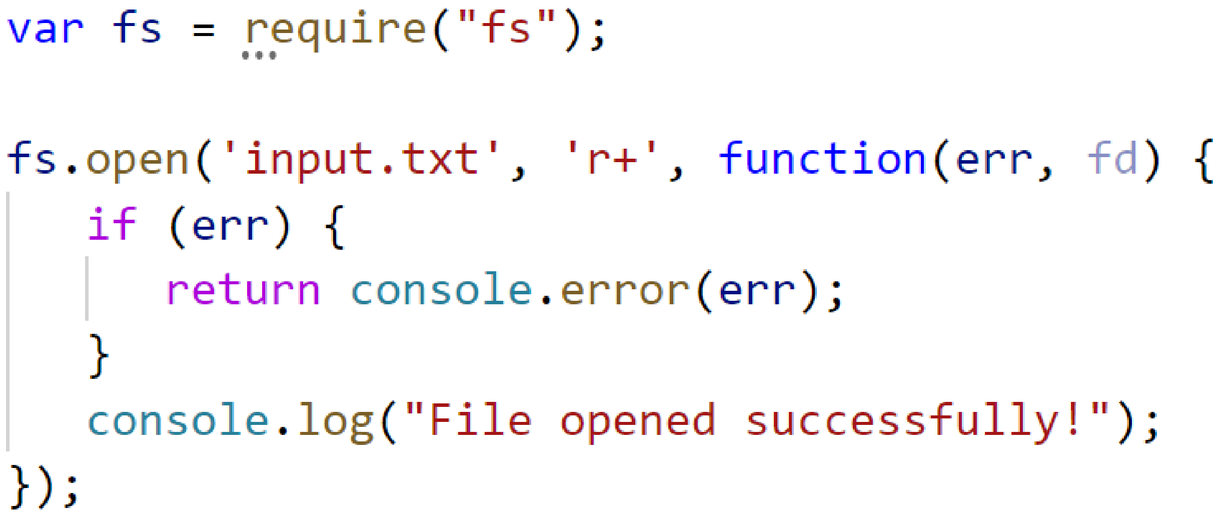
File system



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

fs.readFile('input.txt', function (err, data) {
    if (err) return console.error(err);
    console.log("Asynchronous read: " + data.toString());
});

var data = fs.readFileSync('input.txt');
console.log("Synchronous read: " + data.toString());
```

Open file

- fs.open(path, flags[, mode], callback)
- Flags includes r, r+, rs, w, wx, w+, wx+, a, a+



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

fs.open('input.txt', 'r+', function(err, fd) {
   if (err) {
      return console.error(err);
   }
   console.log("File opened successfully!");
});
```

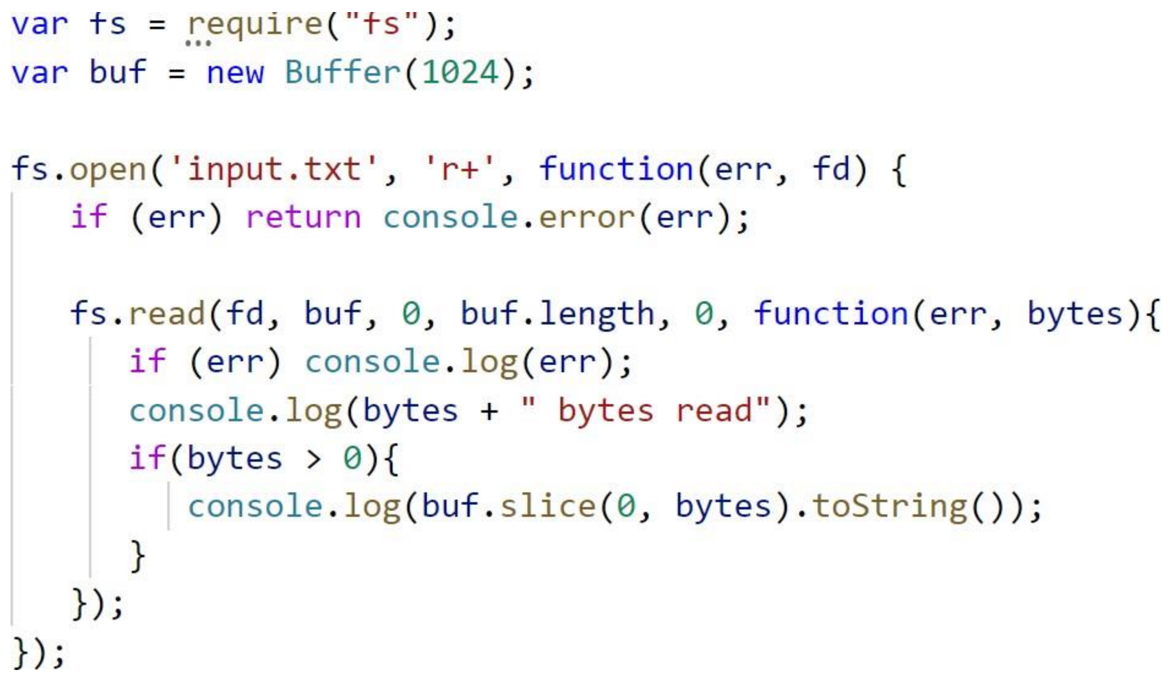
Writing File

- fs.writeFile(filename, data[, options], callback)

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

fs.writeFile('input.txt', 'simple writing example', function (err) {
   if (err) return console.error(err);
   console.log("Data written successfully!");
});
```

Reading File

- fs.read(fd, buffer, offset, length, position, callback)



```
var fs = require("fs");
var buf = new Buffer(1024);

fs.open('input.txt', 'r+', function(err, fd) {
   if (err) return console.error(err);

   fs.read(fd, buf, 0, buf.length, 0, function(err, bytes){
      if (err) console.log(err);
      console.log(bytes + " bytes read");
      if(bytes > 0){
         console.log(buf.slice(0, bytes).toString());
      }
   });
});
```

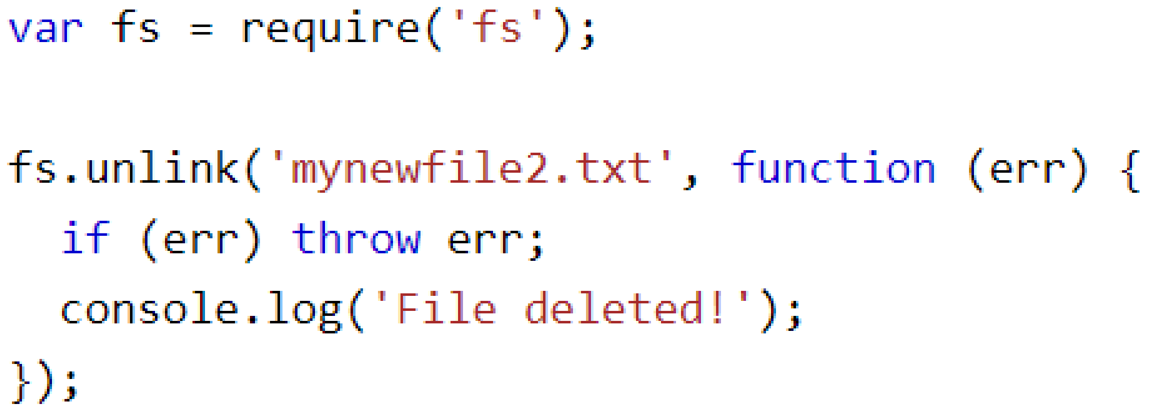
Append file

- Add contents to existing file

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

fs.appendFile('mynewfile1.txt', 'Hello content!', function (err) {
  if (err) throw err;
  console.log('Saved!');
});
```

Delete file

- Delete file from system

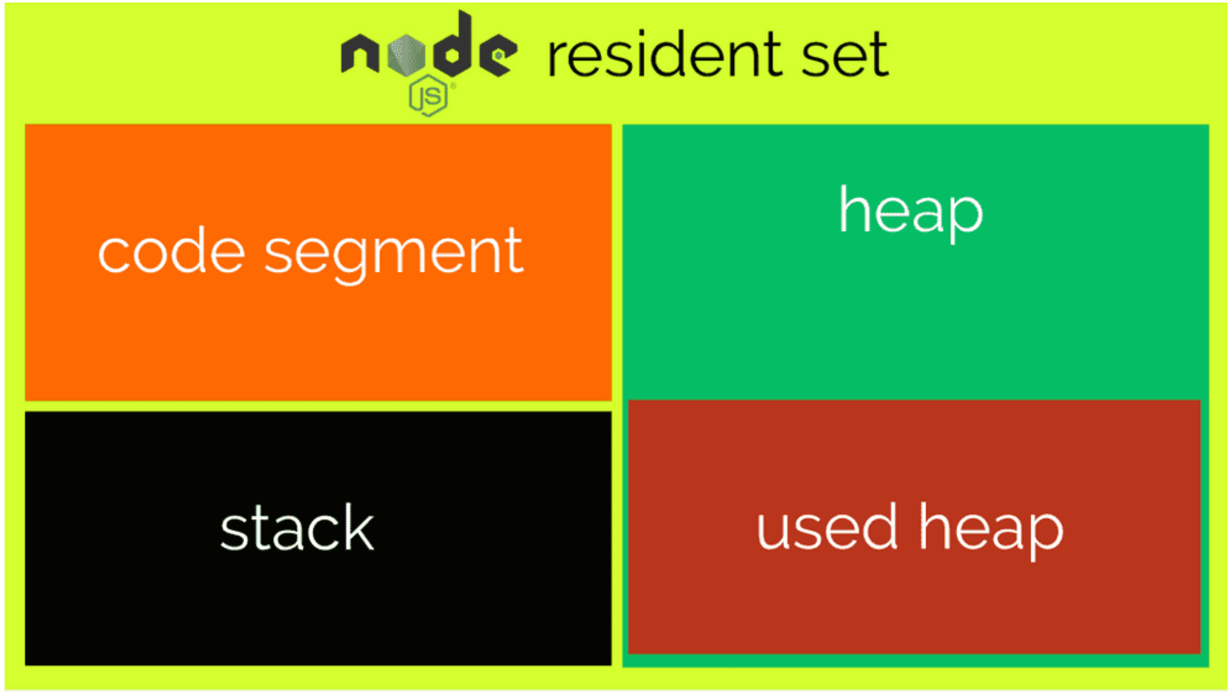


```
var fs = require('fs');

fs.unlink('mynewfile2.txt', function (err) {
  if (err) throw err;
  console.log('File deleted!');
});
```

More file operations

- Close file: fs.close(fd, callback)
- Truncate File: fs.ftruncate(fd, len, callback)
- Delete File: fs.unlink(path, callback)
- Create Directory: fs.mkdir(path[, mode], callback)
- Read Directory: fs.readdir(path, callback)
- Remove Directory: fs.rmdir(path, callback)

Nodejs memory region



Streams

- They are data-handling method and are used to read or write input into output sequentially
- Streams are a way to handle reading/writing files, network communications
- Instead of program reading file into memory **all at once** like in traditional way, streams read chunks of data piece by piece, processing its content without keeping it all in memory
- This makes streams really powerful when working with **large amounts of data**
- Example, file size can be larger than your free memory space, making it impossible to read whole file into memory in order to process it
- Using streams to process smaller chunks of data, makes it possible to read larger files
- YouTube or Netflix don't make you download video and audio feed all at once

Stream advantages

- **Memory efficiency:** you don’t need to load large amounts of data in memory before you are able to process it
- **Time efficiency:** it takes significantly less time to start processing data as soon as you have it, rather than having to wait with processing until the entire payload has been transmitted

Streams

- Read data from source or write to destination
- Readable – used for read operation
- Writable – used for write operation
- Duplex – used for both read and write operation
- Transform - duplex stream where output is computed based on input

Streams

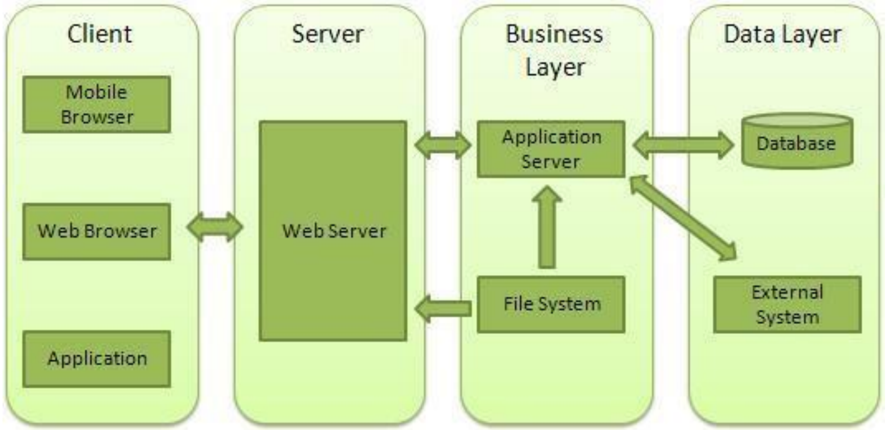
- Throws several events at different instance of times
- data – fired when data is available to read
- end – fired when no more data to read
- error – fired when any error receiving or writing data
- finish – fired when all data has been flushed

Streams-powered Node APIs

- Many Node.js core modules provide native stream handling capabilities
- process.stdin returns a stream connected to stdin
- process.stdout returns a stream connected to stdout
- process.stderr returns a stream connected to stderr
- fs.createReadStream() creates a readable stream to a file
- fs.createWriteStream() creates a writable stream to a file
- zlib.createGzip() compress data using gzip (a compression algorithm) into a stream
- zlib.createGunzip() decompress a gzip stream.
- zlib.createDeflate() compress data using deflate (a compression algorithm) into a stream
- zlib.createInflate() decompress a deflate stream

HTTP module

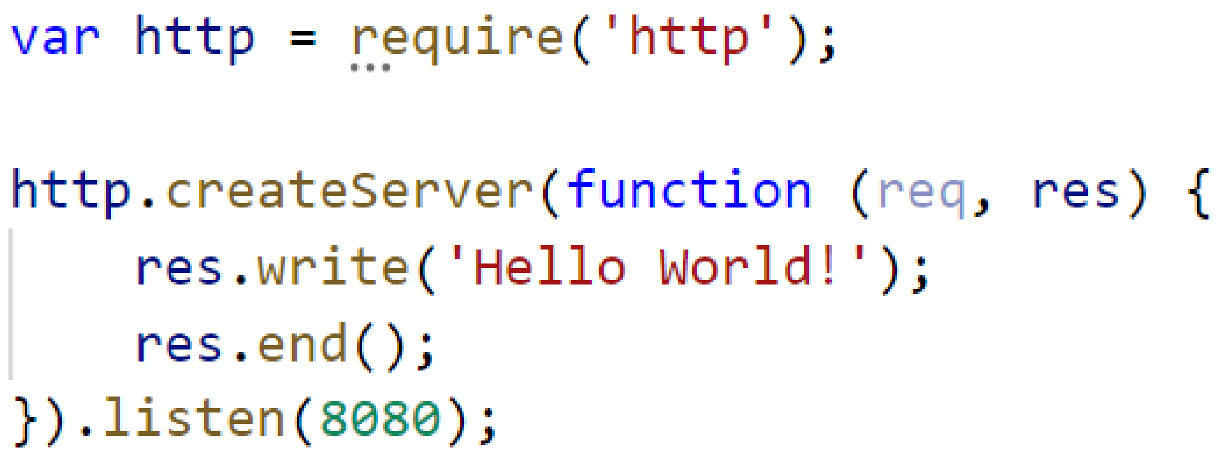
- It handles http request from client
- Web server can be created



http alternative

- net / require(‘net’): provides the foundation for creating TCP server and clients
- dgram / require(‘dgram’): provides functionality for creating UDP / Datagram sockets
- https / require(‘https’): provides an API for creating TLS / SSL clients and servers

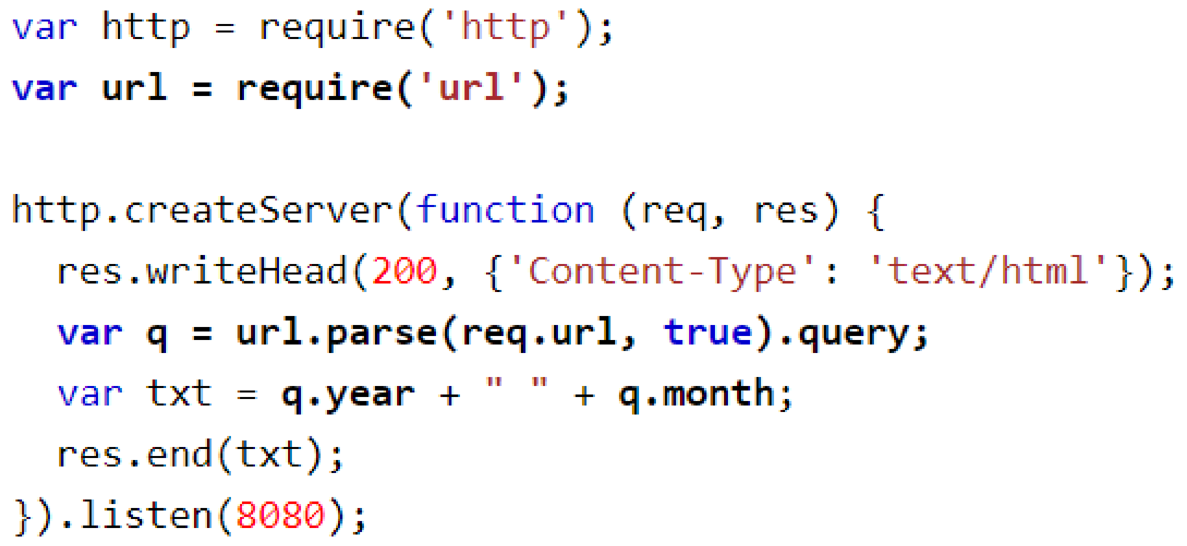
HTTP server



```javascript
var http = require('http');

http.createServer(function (req, res) {
    res.write('Hello World!');
    res.end();
}).listen(8080);
```

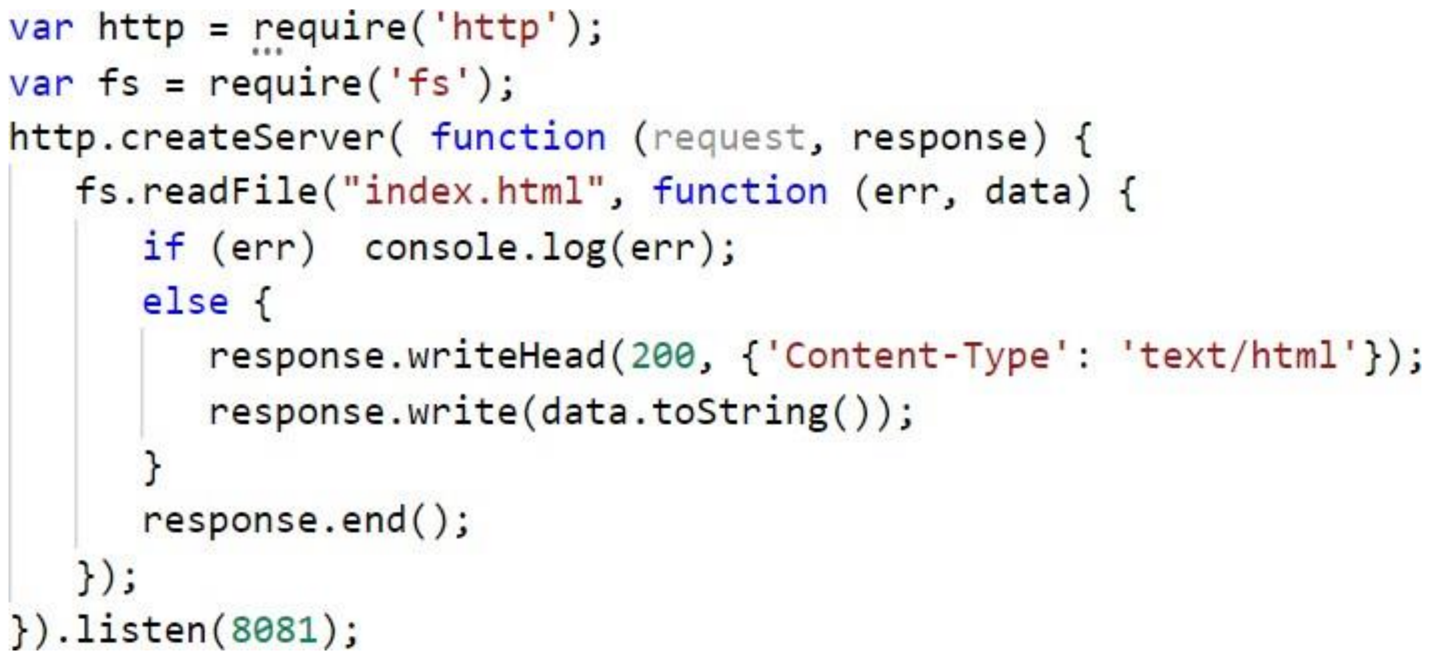
More about http module

- Adding http header: res.writeHead(200, {'Content-Type': 'text/html'})
- Or response.setHeader('content-type','text/html'); response.setHeader(200);
- Reading query string: res.write(req.url);



```
var http = require('http');
var url = require('url');

http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  var q = url.parse(req.url, true).query;
  var txt = q.year + " " + q.month;
  res.end(txt);
}).listen(8080);
```

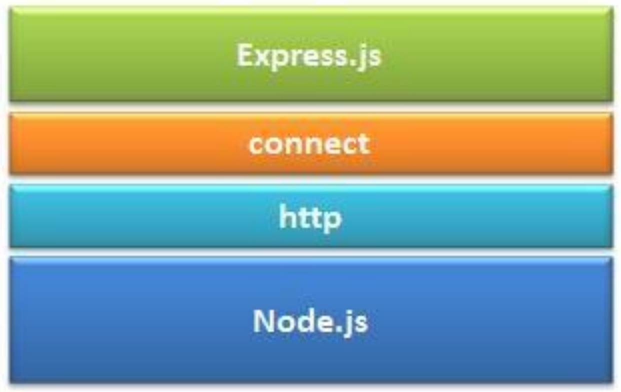
Sending html file



```
var http = require('http');
var fs = require('fs');
http.createServer( function (request, response) {
   fs.readFile("index.html", function (err, data) {
      if (err)  console.log(err);
      else {
         response.writeHead(200, {'Content-Type': 'text/html'});
         response.write(data.toString());
      }
      response.end();
   });
}).listen(8081);
```

Express framework

- Minimal and flexible Node.js framework
- Used to develop web applications
- Facilitates rapid development
- Allows to setup middleware for HTTP Requests
- Defines routing
- Allows to dynamically render HTML Pages



Express features

- Request / Response enhancements
- Routing
- Views and Templates
- Content Negotiation
- Middleware
- Session, Cookies, URL Parsing, Authentication / Authorization, Error Handling

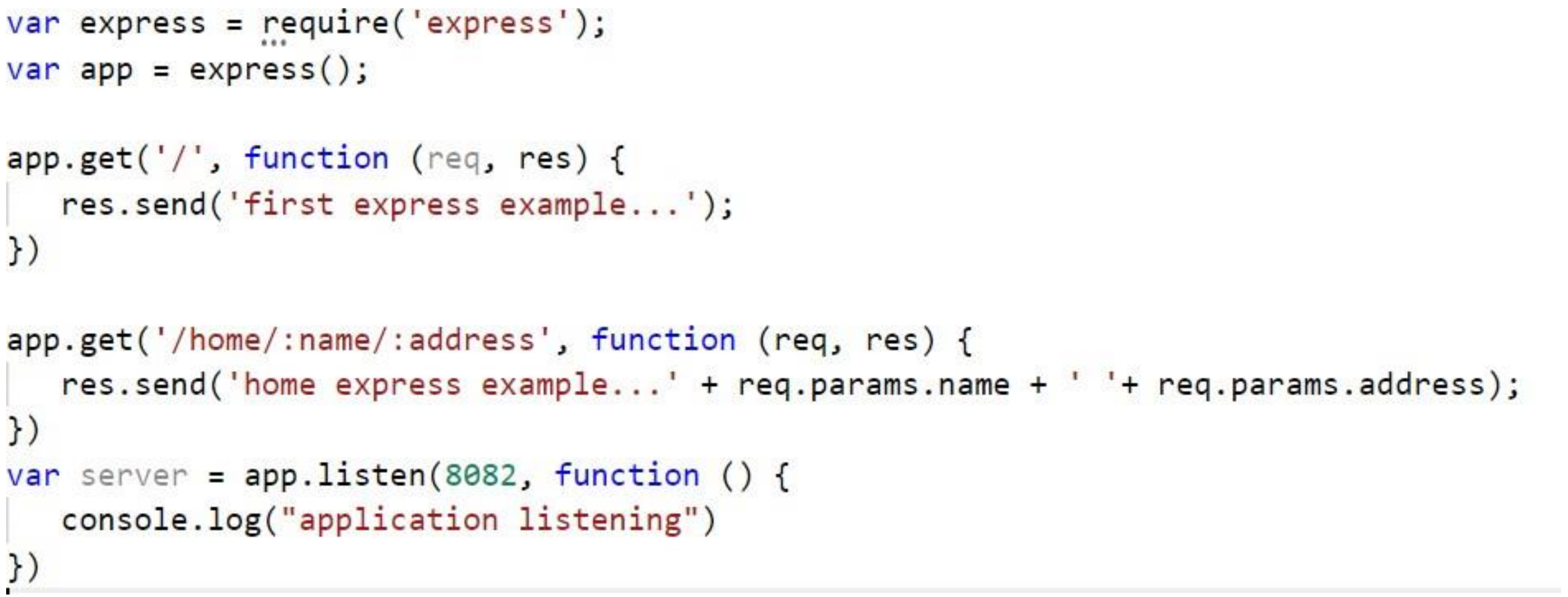
Express setup

- Install: npm install express --save
- Other installation
- npm install body-parser --save
- npm install cookie-parser --save
- npm install multer --save

Nodemon

- Monitor for any changes in your source and automatically restart your server
- Npm install -g nodemon
- Just use nodemon instead of node to run your code
- nodemon example.js

Express example



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

app.get('/', function (req, res) {
   res.send('first express example...');
})

app.get('/home/:name/:address', function (req, res) {
   res.send('home express example...' + req.params.name + ' '+ req.params.address);
})
var server = app.listen(8082, function () {
   console.log("application listening")
})
```

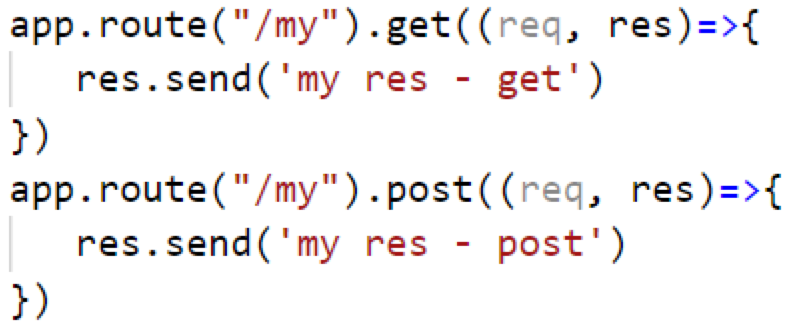
Express routing

- It is URI and HTTP request method
- Mostly used methods
- GET
- POST
- PUT
- DELETE

Routing

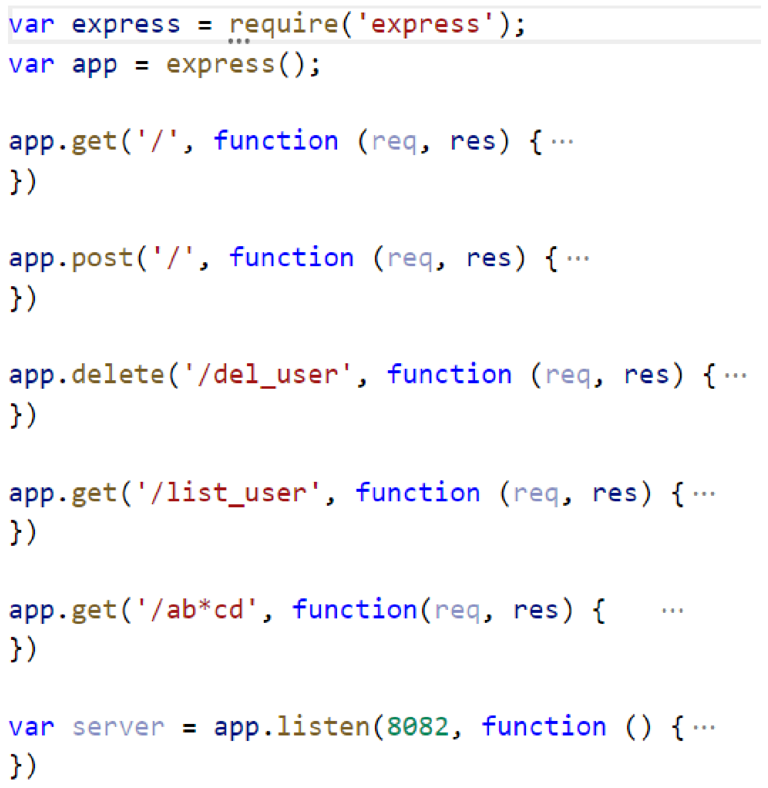
- Determining how application responds to a client request
- Each route can have one or more handlers
- To define a route: app.method(path, handler)
- app –instance of express
- method –HTTP request method
- path –path on server
- handler –function executed when route is matched

Generic route



```
app.route("/my").get((req, res)=>{
    res.send('my res - get')
})
app.route("/my").post((req, res)=>{
    res.send('my res - post')
})
```

Express routing



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

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

app.post('/', function (req, res) {⋯
})

app.delete('/del_user', function (req, res) {⋯
})

app.get('/list_user', function (req, res) {⋯
})

app.get('/ab*cd', function(req, res) {   ⋯
})

var server = app.listen(8082, function () {⋯
})
```

Request

- req object Represents HTTP request
- It has properties for query string, parameters, body, HTTP headers
- Properties of request object originalurl, params, path, protocol, query, route, secure, etc
- **req.baseUrl** – URL path on which a router instance was mounted
- **req.method** – HTTP method of the request
- **req.params** – Object containing properties mapped to named route parameters
- **req.query** – Object containing property for each query string parameter in the route
- **req.body** – Contains key-value pair of data submitted in request body
- **req.cookies** – Object containing cookies sent by the request

Response object

- Specifies response which is sent by an Express
- **res.headersSent** – Indicates if app sent HTTP headers for response
- **res.append(field [,value])** – Appends specified value to HTTP response header
- **res.attachment([filename])** – Sets HTTP response Content-Disposition header to "attachment"
- **res.location(path)** – Sets response Location HTTP header to specified path parameter
- **res.type(type)** – Sets Content-Type header to MIME type
- **res.get(field)** – Returns HTTP response header specified by field
- **res.set(field [,value])** – Sets response'dHTTP header field to value

Response object

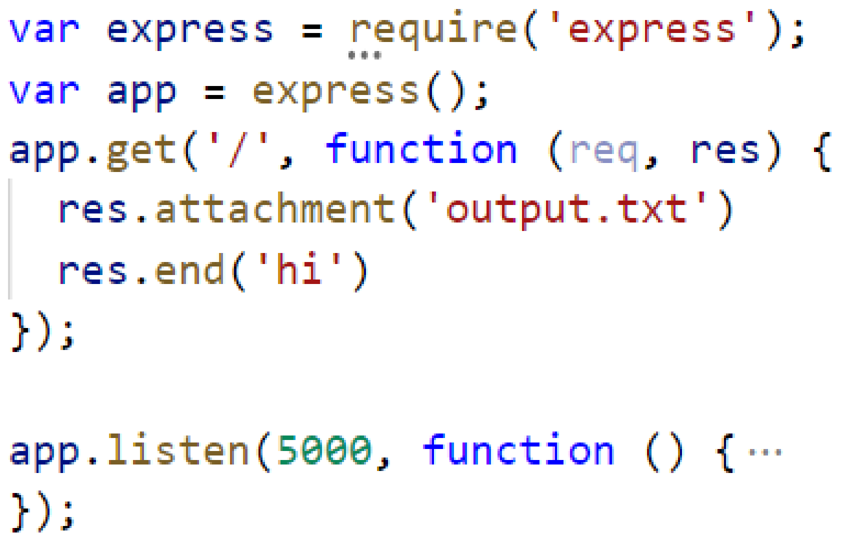
- **res.cookie(name, value [,options])** – Sets cookie name to value
- **res.clearCookie(name[,option])** – Clears the cookie specified by name
- **res.redirect([status,] path)** – Redirects to URL derived from specified path with status
- **res.json([body])** – Sends a JSON response
- **res.jsonp([body])** – Sends a JSON response with JSONP support

Response object

- **res.render(view [,locals][,callback])** – Renders a view and sends rendered HTML string
- **res.status(code)** – Sets HTTP status for response
- **res.sendStatus(statusCode)** – Sets response HTTP status code and send its string
- **res.send([body])** – Sends HTTP response
- **res.sendFile(path [,options] [,fn])** – Transfers file at given path
- **res.download(path [,filename] [,fn])** – Transfers file as an attachment
- **res.end([data] [,encoding])** – Ends response process

Response object

- Properties of response are app, headersSent, locals
- Methods are append, attachment, cookie, clearCookie, download



```
var express = require('express');
var app = express();
app.get('/', function (req, res) {
  res.attachment('output.txt')
  res.end('hi')
});

app.listen(5000, function () {⋯
});
```

Sails

- Sails.js is a Node.js framework that allows you to build enterprise-ready, custom MVC (model, view, controller) application on-the-go
- Sails.js has built-in features such as an API creator, and its socket integration in every route and database ORM makes it very useful and helps speed up development
- Sails.js has lots of features
- Auto generated API
- Database ORM
- Inbuilt task runner
- Security code
- Built-in web sockets in routes.
- Install: npm install -g sails
- Create app: sails create projectName and npm install
- Run app: sails lift

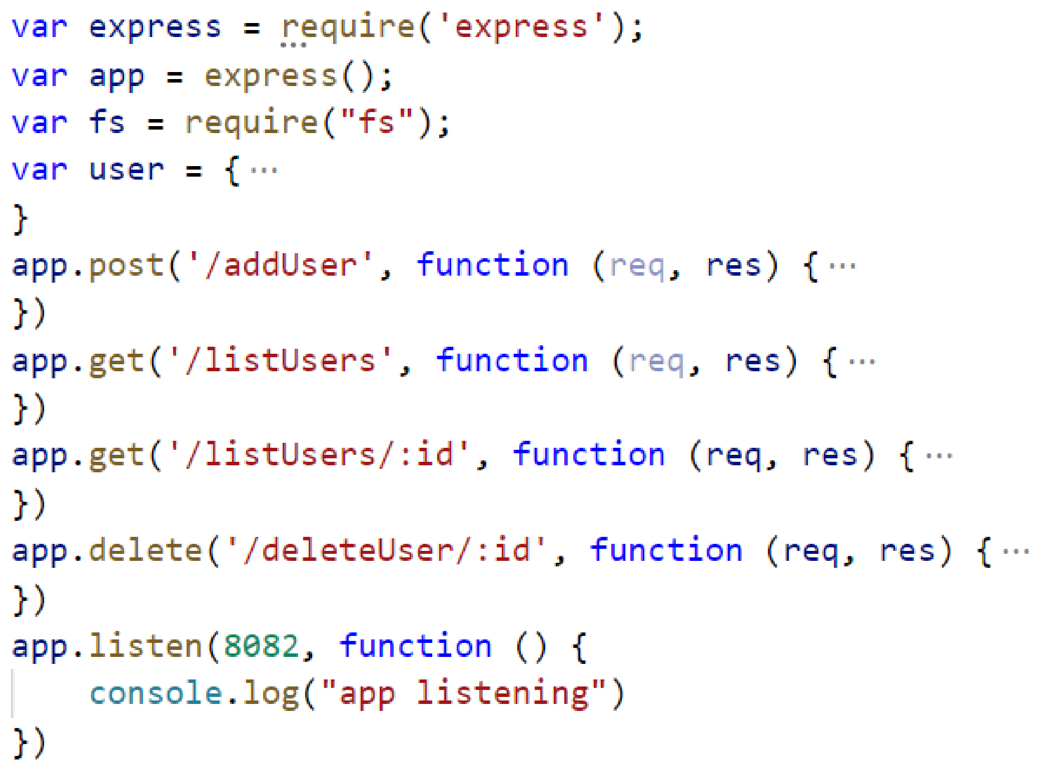
RESTful API

- It is REpresentational State Transfer
- REST server provides access to resources
- REST client accesses and modifies resources using HTTP
- Different representations are text, JSON, XML
- HTTP methods: GET, PUT, DELETE, POST

REST framework

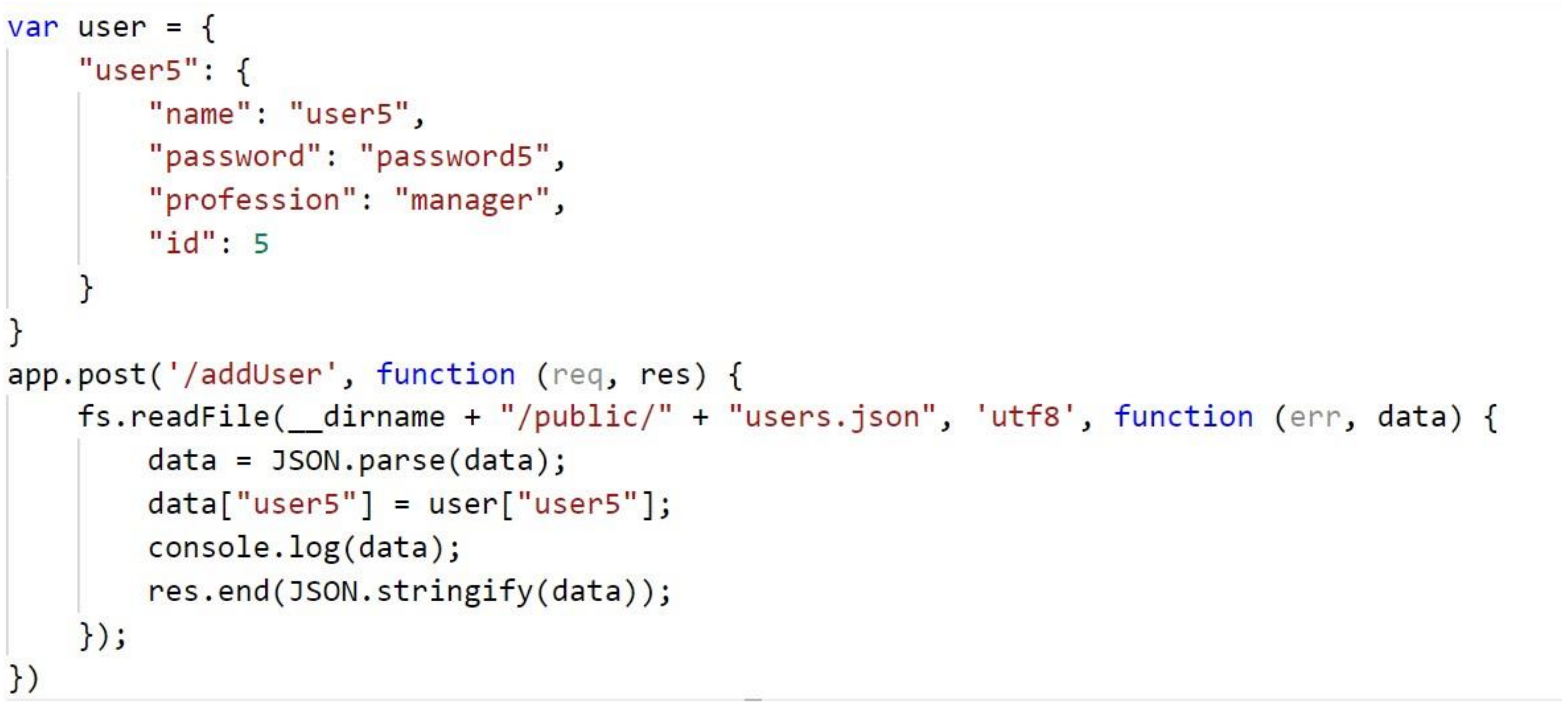
- Express.js
- Geddy
- Locomotive
- Total.js
- Restify
- Keystone
- Loopback

RESTful API



```
var express = require('express');
var app = express();
var fs = require("fs");
var user = {⋯
}
app.post('/addUser', function (req, res) {⋯
})
app.get('/listUsers', function (req, res) {⋯
})
app.get('/listUsers/:id', function (req, res) {⋯
})
app.delete('/deleteUser/:id', function (req, res) {⋯
})
app.listen(8082, function () {
    console.log("app listening")
})
```

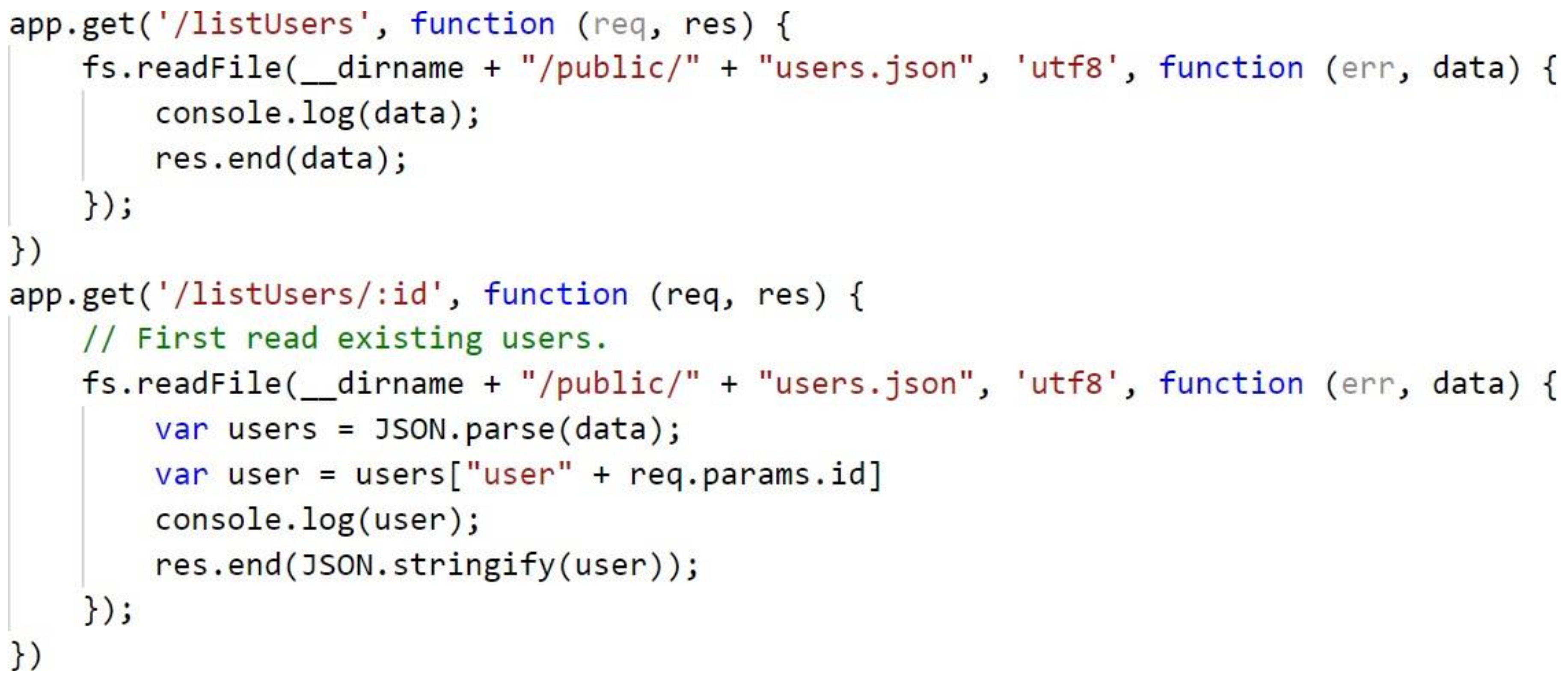
post



```
var user = {
    "user5": {
        "name": "user5",
        "password": "password5",
        "profession": "manager",
        "id": 5
    }
}
app.post('/addUser', function (req, res) {
    fs.readFile(__dirname + "/public/" + "users.json", 'utf8', function (err, data) {
        data = JSON.parse(data);
        data["user5"] = user["user5"];
        console.log(data);
        res.end(JSON.stringify(data));
    });
})
```

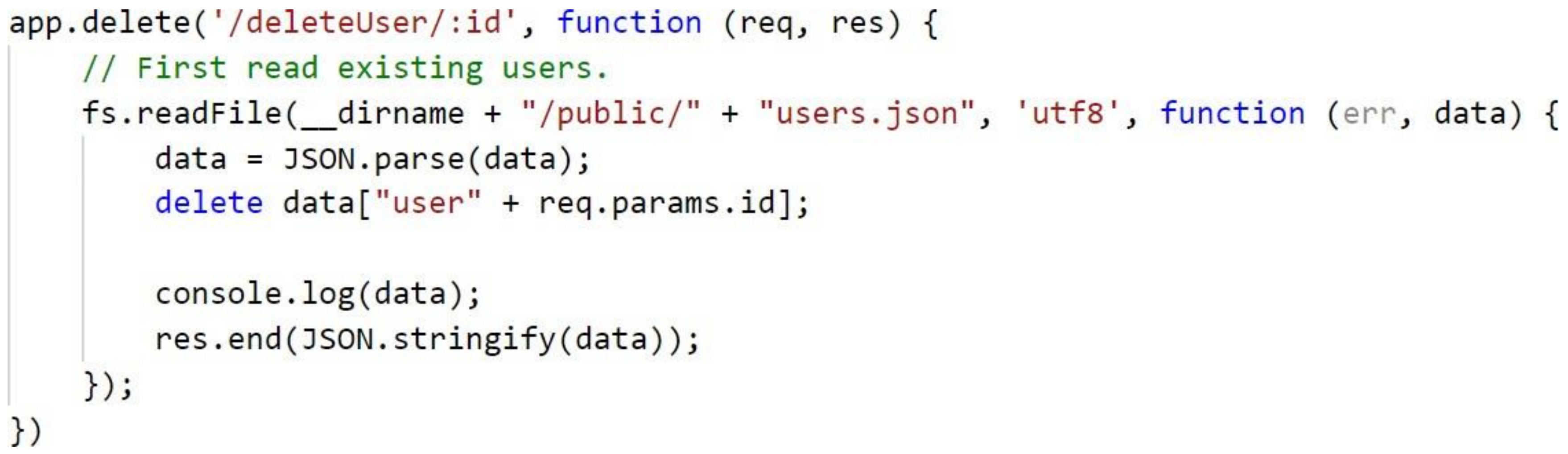
CYBAGE

get



```
app.get('/listUsers', function (req, res) {
    fs.readFile(__dirname + "/public/" + "users.json", 'utf8', function (err, data) {
        console.log(data);
        res.end(data);
    });
})
app.get('/listUsers/:id', function (req, res) {
    // First read existing users.
    fs.readFile(__dirname + "/public/" + "users.json", 'utf8', function (err, data) {
        var users = JSON.parse(data);
        var user = users["user" + req.params.id]
        console.log(user);
        res.end(JSON.stringify(user));
    });
})
```

delete



```
app.delete('/deleteUser/:id', function (req, res) {
    // First read existing users.
    fs.readFile(__dirname + "/public/" + "users.json", 'utf8', function (err, data) {
        data = JSON.parse(data);
        delete data["user" + req.params.id];

        console.log(data);
        res.end(JSON.stringify(data));
    });
})
```

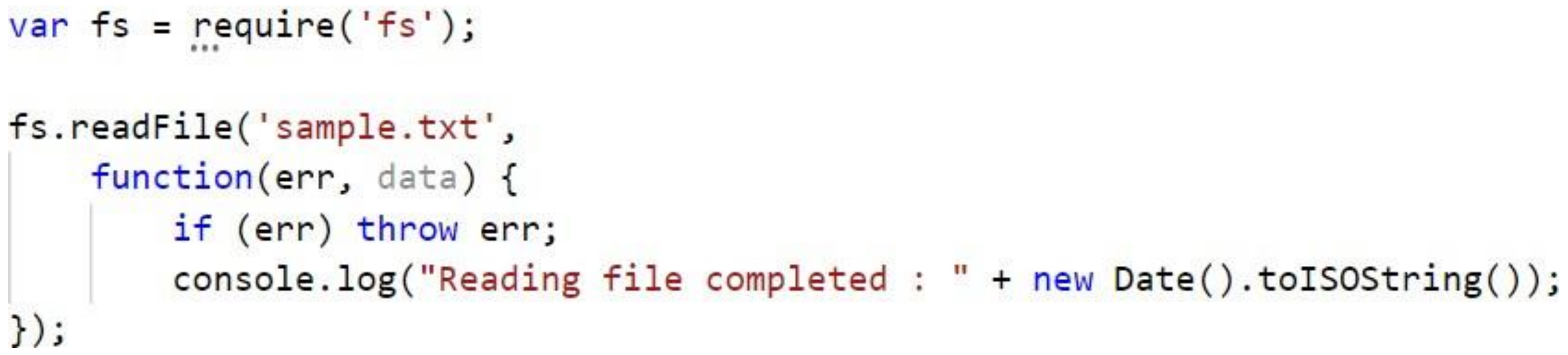
Error handling

- Using Try Catch
- When a piece of code is expected to throw an error and is surrounded with try
- Exceptions thrown in piece of code could be addressed in catch block
- If the error is not handled in any way, program terminates abruptly
- It is a good practice to use Node.js Try Catch only for synchronous operations
- If an error occurs during asynchronous operation, there is no try catch block control could know of

Error handling

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

fs.rename('sample.txt', 'sample_old.txt',
    function (err) {
        if (err) throw err;
        console.log('File Renamed.');
        fs.unlink('sample_old.txt',
            function (err) {
                if (err) throw err;
                console.log('File Deleted.');
            });
    });
```

Error handling



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

fs.readFile('sample.txt',
    function(err, data) {
        if (err) throw err;
        console.log("Reading file completed : " + new Date().toISOString());
});
```

Error handling

next(err)

```
app.get("/users", function(req, res, next){
    User.find(function(err, users){
    // an error? get it out of here!
      if (err) { return next(err); }
   // no error?
   // res.render... etc.
  });
});
```

Error handling

Error handler middleware – 4 parameters

```
app.use(function(err, req, res, next) {
    res.status(err.status || 500);
     res.render('error', {
    message: err.message,
                 error: err
    });
  });
```

Core Debugging

- Core Node.js debugger: adding debugger statement inside script
- Start program: node debug example.js

next	Stop at the next statement.
cont	Continue execute and stop at the debugger statement if any.
step	Step in function.
out	Step out of function.
watch	Add the expression or variable into watch.
watcher	See the value of all expressions and variables added into watch.
Pause	Pause running code.

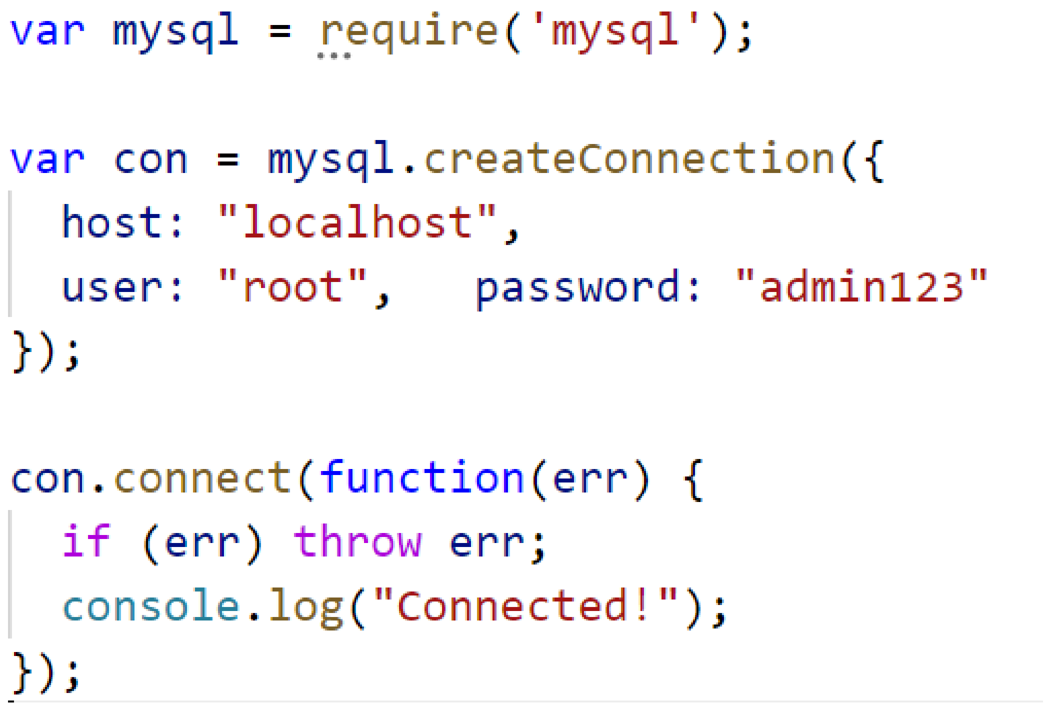
Node Inspector debugger

- Add debugger statement inside script
- Install: npm install -g node-inspector
- Start node-inspector: node-inspector --web-port=5500
- Start debugging: node --debug-brk app.js

Node.js MySQL

- Node.js can be used in database applications
- Installing MySQL: npm install mysql
- Using MySQL: var mysql = require('mysql')

Creating connection



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

var con = mysql.createConnection({
  host: "localhost",
  user: "root",   password: "admin123"
});

con.connect(function(err) {
  if (err) throw err;
  console.log("Connected!");
});
```

Demo to handle the following

- **Create Table**
- **Insert Record**
- **Find a record**
- **Delete a record**

pm2

- Daemon process manager that will help you manage and keep your application online
- Installation : npm install pm2@latest –g
- Start app: pm2 start app.js

CYBAGE

Pm2

- Some options you can pass to the CLI:
- Specify an app name: --name <app_name>
- Watch and Restart app when files change: --watch
- Set memory threshold for app reload: --max-memory-restart <200MB>
- Specify log file: --log <log_path>
- Pass extra arguments to the script: -- arg1 arg2 arg3
- Delay between automatic restarts: --restart-delay <delay in ms>
- Prefix logs with time: --time
- Do not auto restart app : --no-autorestart
- Specify cron for forced restart: --cron <cron_pattern>
- Attach to application log: --no-daemon

Environment

- Create .env file
- Add following entries

```
NODE_ENV=development
PORT=8626

NODE_ENV=production
PORT=8626
```

- Install: npm install dotenv
- Add following lines at start of nodejs app

```
const dotenv = require('dotenv');
dotenv.config();
```

- Use process.env.PORT to access port number

Sails

Sails.js has many great features:

- it’s built on Express.js
- it has real-time support with WebSockets
- it takes a “convention over configuration” approach
- it has powerful code generation, thanks to Blueprints
- it’s database agnostic thanks to its powerful Waterline ORM/ODM
- it supports multiple data stores in the same project
- it has good documentation.

Passport

- Middleware for Node.js to authenticate requests
- Can be used with any Express based web application
- Supports SSO, Oauth, token-based authentication etc

Frequently used modules

1. **Waterline**

2. **Sequilize**

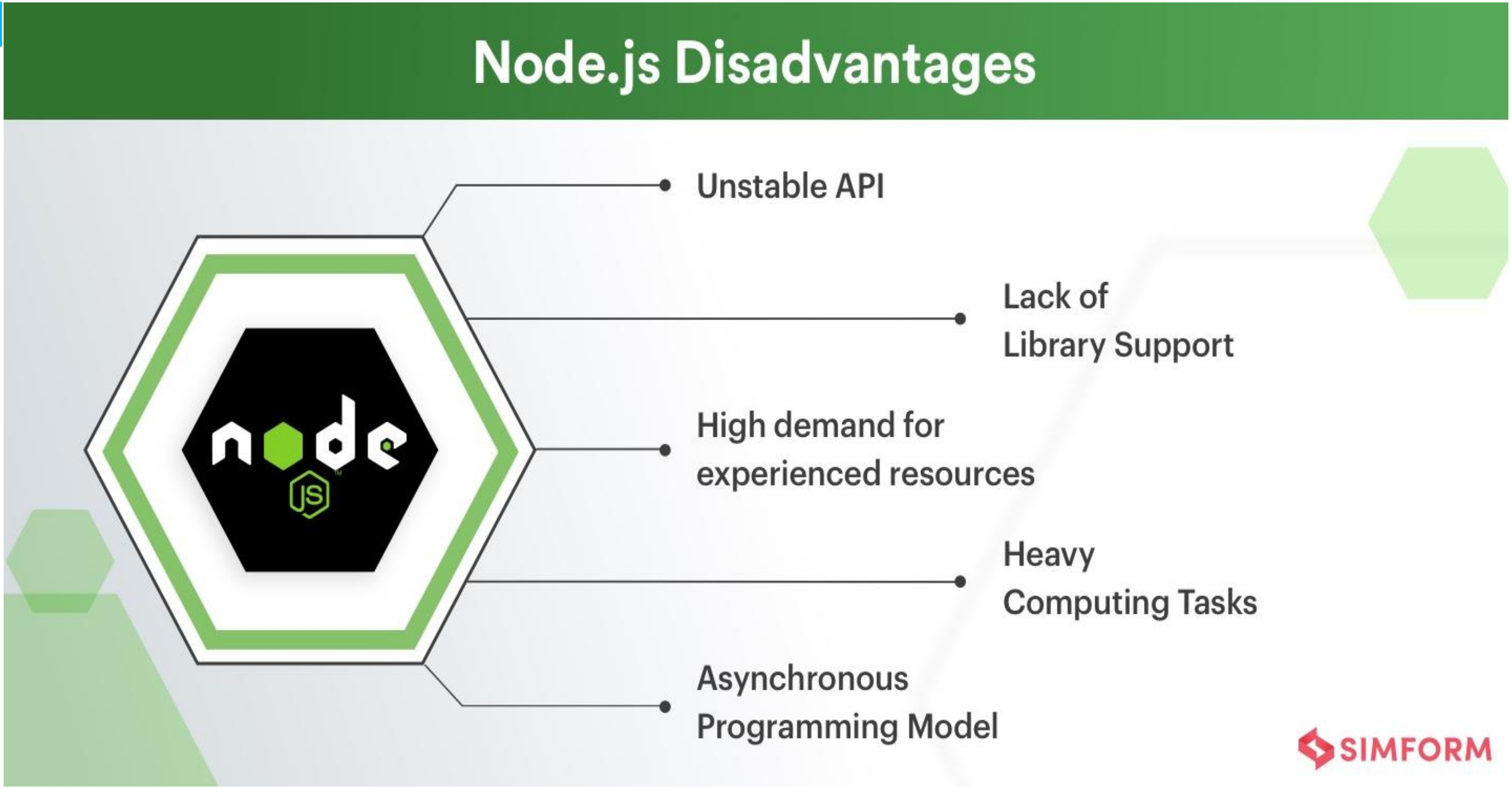
3. **Moongose**

Parameters to choose the modules from npm

- When was it last updated
- Number of issues
- Number of contributors

Cons of using Node JS

- Not efficient in handling CPU-intensive apps.
- Not mature enough.
- Simplicity.
- No client app required.
- Speed of Coding.



Node.js Disadvantages

- Unstable API
- Lack of Library Support
- High demand for experienced resources
- Heavy Computing Tasks
- Asynchronous Programming Model

Performance best practices

- Use gzip compression
  - Greatly decreases response size and hence increases speed of application
- Don’t use synchronous functions
  - Tie up executing process until they return
- Do logging correctly
  - console.log() and console.error() are synchronous
  - Use asynchronous logging libraries
- Handle exceptions properly
  - Node apps crash on encountering an uncaught exception
  - Error first callbacks
  - try-catch

CYBAGE

Thank You!

www.cybage.com

Copyright © 2019 Cybage Software Pvt. Ltd. All Rights Reserved. Cybage Confidential.

www.cybage.com