1st Lecture: Node JS

Node js=сървърен js.

JS е създаден да живее и да се изпълнява в браузъра на клиента, в който има специален енджин който парсва js кода. Популярността на js нараства във времето. Всички устройства имат браузър => Изпълняват js код.

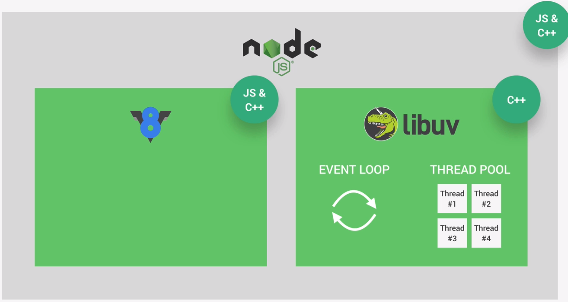
Създаването на node JS: вземане на парсера к. е в браузъра. В браъзъра има хромиум енджин – голям пакет с множество програми. V8 engine е част от този пакет и той е парсера на js code. Взема се този енджин и се слага в програма коят да изпълнява JS code извън браузъра, в самата машина. Така JS започва да работи като сървърен език. От client side code, JS become server side code. Node JS e този рънтайм и позволява JS да се изпълнява на машината.

\*Node.js is a JS runtime built on google`s opensource v8 JS Engine.

\*javascript runtime: Browser is JS runtime. Every B natively understans HTML, CSS and JS. What if we take JS out of the B???. Out of the B Node.js allowes us to use JS. Node JS is just another JS Runtime. As a container where JS can be executed.

\* v8 JS Engine developed by google comes to play, where JS code will be parsed and run in node.js.

1.NodeJS is single threaded and this makes node js very lightweight and efficient. Node is perfect for building super fast and scalable apps

**NODE ARCHITECTURE**

there are a couple of libraries that Node depends on. The Node runtime has a several dependencies, and the most important are V8 JS engine and libuv.

V8

Without V8 for Node there is no way for understanding JS code. So V8 is a fundamental part of Node anrchitecture. V8 is the engine that converts JS code into machine code that a computer can understand.

Libuv is an open source library with a stong focus on asynchronous IO=input/output. Libuv implements two extremely important features in node.js: Event Loop and Thread Pool

Event Loop: is responsible for handling task like executing call backs and network IO while the thread pool is for more heavy work. Libuv is written in C++ and JS. V8 also use C++ code. SO Node is program written in C++ and JS. Node Js ties all these libs together, no matter if written in C++ or JS and then gives us developers access to their functions in pure JS.

1.NB: EVENT LOOP; Stack,Heap???

2.MODULES = allow larger apps to be split and organized. Всеки модул си има собствен модулен контекст и са изолирани от глобалния контекст. Можем да пишем в модула, но не замърсяваме глобалния контекст. Можем да си изолираме кода в модули и той не може да се достъпва отвън свободно. Кода е енкапсулиран в рамките на модула.

Node JS има 3 типа модули!!!

2.1. Core Modules: те идват от стандартната библиотека на ноде js

2.2. Local Modules: custom modules, които ние можем да си създаваме и да ги реферираме помежду си

2.3. Third-Party Modules: може да се изтеглят

2.1 Core Modules: не се инсталират а те идват от стандартната библиотека на ноде js.

Node.js api => ще видим вградените библиотеки и имат документация

--Ex: Path-модул

1st step import: const path = require(‘path’)

log(path.resolve(‘./’))

-Ex: URL – modul: split url на съставните му части:

1st step import: const url = require(‘node : url’)

Const url= new URL(‘URL’)

Съставни части на URL: <http://localhost:3000/catalog?page=5>

- **href**: <http://localhost:3000/catalog?page=5>

- **http**: protocol

-**Host**: localhost:3000

-**Port** :3000

-**Pathname**: всичко до ?: <http://localhost:3000/catalog>

-**Query**/search: всичко след ?:page=5

-**Slug**: the last part of URL , that contains a unique string that identifies the resource, that the website is displaying: ex fresh-avocado instead of id: number, that is not meaningful

#fragment-работи само в браузъра

-Ex: query string – тази част от url след ?-декодира url string и го парсва

2.2. Local Modules-two types модулни системи; могат да се създават локално в node.js application

-ES6 modules: export default Name; export const Name;

Появяват се по късно 6г след commons JS modules. 2015g

-commonJS modules:

\*export default:

module.exports = calc 2009, те са част от node.js

Node.js работи с commons modules. Те си вървят дефолтни в node js. Node js може да работи и с ES6 modules; В Package.json{“type”: commonjs} – default option, но може да го променим на {“type”: module } =>ES6

\*import default:

const calc = require(“./calculator”) .js Не е задължително

\*named export: exports.calc=calc

\*named import: const math(or some other name)=require(‘./calculator’)

Log(math.calc(1,2)

Или destruct: const {calc}=require(‘./calculator’)

NB: не може да се прави circular dependency. Модулте трябва да са в йерархична структура/дърво и да намя завъртане без значение през колко модула минава

Run f. in vs code : има задатен дефолтен старт ctr+f5 който зарежда от package.son {“main”: name.js} и колкото и Js files да сме създали, ще се рънва майн фаила. Това може да се промени с команда:

node .\calculator.js или node .\name.js

2.3. Third-Party Modules: npmjs.com =>search “is”

-is: популярна библиотека

npm insall is/npm I is

-за да се използва библиотеката:

1.Инстал: npm insall is/npm i is: създава папка node modules и вътре е всичко инсталирано и там е кода на библиотеката

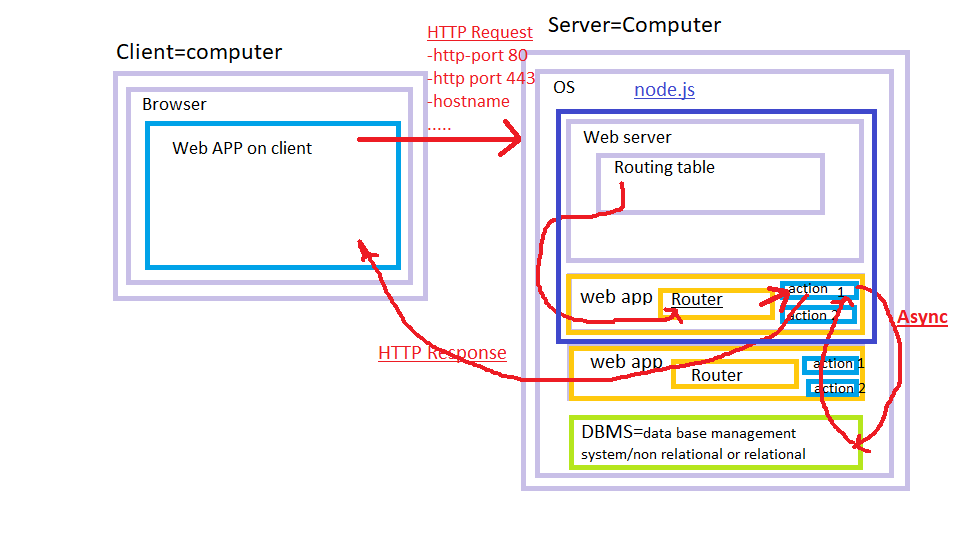
2.импорт: const is=requiere(“is”)

3. Node.js Web Server

Web Servers – are software products that use operating system to handle web requests. Web servers serve Web content.

Node.js може да създадем сървър койото да слуша за заявки и да връща отговор.

HTTP request:



Module:http

Const http=require(“http”);

Const server=http.createServer((reqest,response)=>{

});

Server.listen(3000)

Create unique id:    'asdf'+('0000'+(Math.random()\*9999 | 0)).slice(-4)

“|0”-обръща го в цяло число. Това е трик. Побитово орване с 0. Побитовите операции в JS са с 32битови числени данни, докато стандартните дании в JS са 64 битови флоутове. Това автоматично ще превърне числото в цяло число.

const sth=require(‘../sth.json’)

Когато Require види json file, го чете като обикновех JS -object и не е небходимо json.stringify