2nd lecture

1. Publish-Subscribe Pattern (Pub-Sub-Pattern).

-design pattern

-Използва се за комуникиране на съобщения между различни системни компоненти без да има обвързване между компонентите

* Used to **communicate messages** between different   
  system components without them knowing anything about each other’s **identity**

\*проблем:

Един модул използва друг модул. Това се прави чрез импортиране на модула и правим dependency(създава зависимост: 1вия модул за да работи използва 2рия модул-depend-ва на 2рия модул.

Имаме 3ти модул който също депендва на 2рия модул. Това е нормално защото ние искаме да постигнем преизползваемост на модулите. 2ри модул се преизползва. Разделяме кода на малки парчета-компоненти(модули) и можем да ги преизползваме.

Имаме 4ти модул който също използва модул 2

+DRY-do not repeat yourself = недей да копираш код, а го преизползвай. Когато кода се повтаря на много места при edit трябва се едитва на всички места.

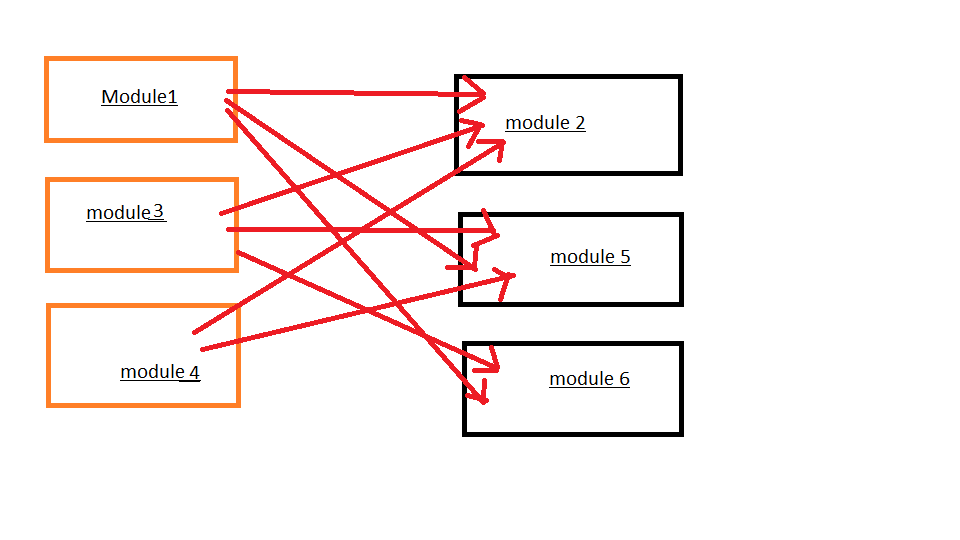
+Single Responsibility-всяка жаба да си знае гьола. Опитваме се да имаме различни модули които да имат собствена отговорност. Separation of concerns = всеки модул си има строго дефинирано предназначение.

Ако имаме 5ти модул, на който 1,3,4 също депендват. Правим депенденси.

Ако имаме 6ти модул, на който 1,3 депендват например. Правим депенденсита.

Така се получават много депенденсита и става преплитане=манджа с грозде.

Започваме да нарушаваме друг принцип: Loose coupling-да се стараем в нашата система да имаме минимален брой връзки. Да имаме отделени модули които не са толкова свързани помежду си. Гогато всичко се преплита прекалено много и става: spagetti code-системата става много трудна за подръжка, за дебъгване и т. н.



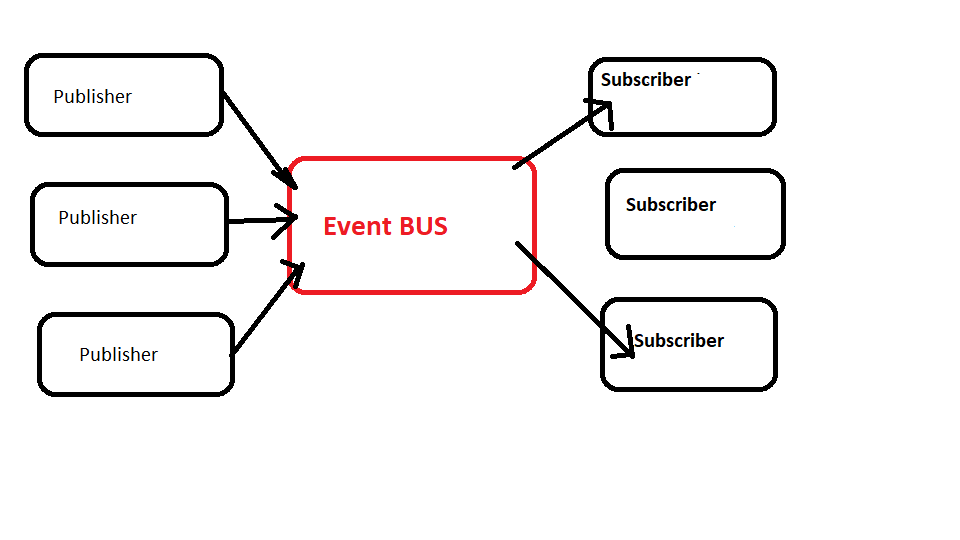
* + **Senders** (publishers), do not program the messages   
    to be sent directly to specific **receivers** (subscribers)

-разделяме системата от модули на такива които искат да изпращат(publishers) и такива които искат да получат инфо. Разделяки системата по този начин, тя се опростява. Не се налага изпращача да знае за всички получатели, за да знае какво да им прати. Изпращача изпраща, а получателите си го получават. Намесва се междинно звено „message broker“=”Event Bus”

* + Subscribers express interest in **one or more events**,   
    and only **receive messages** that are of **interest**
* Decouple and Scale Independently
* Makes software more **flexible**
* Eliminate Polling: Polling: проверява постоянно дали нещо се случило: това случи ли се? -не, това случи ли се? -не, това случи ли се? -не...

Просто когато се случи, се съобщава за това.

* Promotes **faster response time** and **reduces the delivery latency**
* Simplify Communication
* Reduces complexity by **removing** all the **point-to-point connections** with a single connection



DEMO EVENT BUS!!!

2.Streams: разбиват голям обем от данни на малки парчета (чънкове), които стават лесни за прехвърляне. Стрийм с видео: не се прехвърля целото видео наведнъж. Създават се чънкове и се изпращат като стрийм.

-използват се за обработка на данни, преьвърляне на данни.

\*Колекция от данни които не са достъпни наведнъж всичките, а са достъпни на парче (чънк). Като поток, като река от парчета.

* **Collections of data** that is not available at once
  + Data may come **continuously** in **chunks**

Видове:

-readable stream – can only be read

-writeаble stream

-duplex stream

-transform stream

3.Readable Stream:

const fs = require('fs');

const readStream =fs.createReadStream('./data.txt',{encoding: 'utf-8'});

readStream.on('data',(chunk)=>{

console.log('------New chunk----------');

console.log(chunk);

});

readStream.on('close',()=>{

    console.log('Stream closed');

});

=>

------New chunk----------

<Buffer 4c 6f 72 65 6d 20 69 70 73 75 6d 20 64 6f 6c 6f 72 20 73 69 74 20 61 6d 65 74 20 63 6f 6e 73 65 63 74 65 74 75 72 20 61 64 69 70 69 73 69 63 69 6e 67

... 65486 more bytes> ===1chunk

------New chunk----------

<Buffer 69 6c 69 73 20 62 6c 61 6e 64 69 74 69 69 73 20 65 61 71 75 65 20 69 70 73 75 6d 2c 20 76 65 6c 20 71 75 69 73 20 69 70 73 61 20 72 65 63 75 73 61 6e

... 65486 more bytes>

------New chunk----------

http.createServer((req, res)…-

req=read stream

res=write stream

* **HTTP Request** is a readable stream

const http = require('http');

http.createServer((req, res) => {

if (req.method === 'POST') {

let body = '';

req.on('data', data => { body += data });

req.on('end', () => {

console.log(body);

});

}

}).listen(5000);

4.Write stream: създава нов файл в който записва едно по едно нещата-chuk-ове които му казваме да запише.

Demo code!

“pipe.” -връзва 2 стрийма заедно:

5. Nodemon

readStream.pipe(writeStream);

Разликата между dependency and dev-dependency:

Dependency- used by developer and user

Dev-dependency-used only from developer npm i -D nodemon

Package.json:

"scripts": {

    "start": "nodemon ./index.js"

  },

-

6.Packages, updating, version:

npm outdated: намира пакети които са изтекли и неактивни;

npm install [slugify@1.0.0](mailto:slugify@1.0.0) – инсталиране на конкретна версия

npm update slugify

npm uninstall slugify

-какво означават числата на версиите???

Ex: “nodemon”: “^1.18.11”-във версия 1.18 девелъпърите откриват грешка и я поправят и update v: 1.18.1. След това откриват др бъг и издават V: 1.18.2 и т.н. Последната цифра е за **bug fixes.**

Ex: “nodemon”: “^1.18.11”-minor version: introduces some new features into the package that won`t break our code. There are not breaking changes

Ex: “nodemon”: “^1.18.11”-major version-changes that can break code> Can affect the code that is written by older version.

Ex: “nodemon”: “^1.18.11” – specifies which updates we accept for each of the pachages. ^ means that we accept patch and minor releases.

Ex: “nodemon”: “~1.18.11”: accept only minor updates and that is better

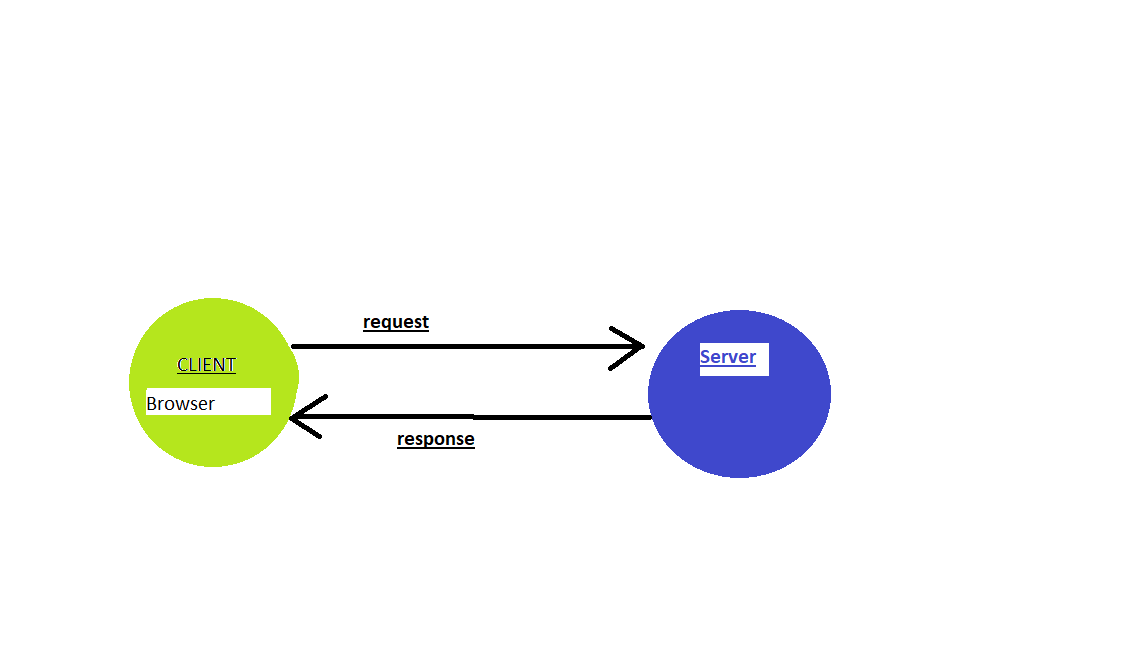
Ako ъпдейтнем версията ще получим отново 1.19... и това няма да счупи кода.

Ex: “nodemon”: “\*1.18.11”: includes all the versions

7.Node modules folder – contains all the dependencies of our project

8. How the web works:

What does happen each time when we write URL in the browser in order to open a new webpage?

Our browser which is also called client sends a request to the server where webpage is hosted. The server send back a response. This process is called request-response model == client-server architecture

If we write [www.google.com/maps](http://www.google.com/maps), or Everey url gets an HTTP or HTTPS, which is for the protocol that will be used on the connection.Then we have the domain name which is google.com in this case, and also aftet the slash, is the resource

/maps The domain name like google.com is not actually the real address of the server, but just a nice name that is easy for us to memorize. So we need a way of converting the domain name to the real address of the server and that happens through DNS.

\*DNS = Domain Name Server which are special servers that are basically like the phone-books of the internet. So the first step that happens when we open up a website is that the browser makes a request to DNS and this special server will simply match the web address that we type into the browser to the server`s real IP address. This happens through your internet service provider or ISP. NB: the domain is not the real address and that a DNS will convert it to that real IP address which a browser can then call after it being sent back to our browser.

https//2016.58.211.206.443

protocol: https

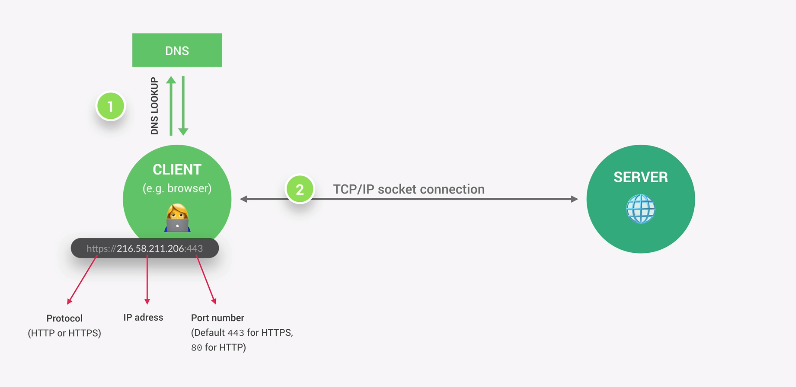
IP address: 216.58.211.206

Port number: Default 443 for https and 80 for http

So once we have the real address a TCP socket connection is established between the browser and the server and they are finally connected. And this connection is typically kept alive for the entire time it takes to transfer all the files of the website.

\*TCP is the Transmission Control Protocol

\*IP=Internet Protocol

TCP/IP together they are communicaton protocols that define exactly how data travels across the web.

After that it is time to make our request and the request that we make is an HTTP request

\*HTTP = Hyper Text Transfer Protocol

After TCP/IP, HTTP is yet another communication protocol.

Communication protocol is a system of rules that allows two or more parties to communicate.

A request message structure:

1.Srart line

GET/maps HTTP/1.1 = HTTP method + request target + HTTP version

HTTP method =GET/POST/PUT/PATCH

Request target: this is where the server is thought that we want to access the /maps resource of this example

2.HTTP request HEADERS

Host: [www.google.com](http://www.google.com)

User-Agent: mozilla/5.0

Accept-Language: en-US

…3.Request Body

<BODY>

