# GLOBAL OBJECT

Jadi kemarin, kita menggunakan "console.log ()" untuk mencatat sesuatu di konsol. Object console ini, kita sebut, global object. Ini adalah bagian dari global scope, yang berarti kita dapat mengaksesnya di mana saja dalam file apa pun.

setTimeout()

clearTimeout()

setInterval()

clearInterval()

Nah, function-function diatas adalah global object di javascript.

Nah, di web browser, kita memiliki window object, yang mewakili global scope. Jadi semua variabel dan function, didefinisikan secara global, sehingga kita dapat mengaksesnya melalui object window ini. Jadi kita bisa ketik window.console.log, atau cukup hanya console.log. Javascript engine akan otomatis memberikan prefix window.console.log. Demikian pula semua fungsi di atas, bisa diakses di window object, seperti window.setTimeout ().

Sama halnya apabila kita define sebuah varilable. Var message = ‘test message’; varilable ini juga dapat diakses melalui window object.

Kemarin, saya katakan bahwa di nodejs kita tidak memiliki object window ini, karena itu dimiliki oleh web browser, sedangkan nodejs adalah runtime environment yang berjalan diluar browser. Sebaliknya, kita memiliki objek lain yang disebut global. Jadi kita bisa melakukan global.console.log, atau global.console.setTimeout.

# CREATE MODULE

Kemarin kita sudah mencoba untuk membuat sebuah modul dalam aplikasi ini. Kita buat 1 file, dengan nama logger.js . Bayangkan kita akan membuat module untuk logging service. Di modul ini, bayangkan kita akan menggunakan service remote logging services untuk record system log kita. Jadi ada website diluar sana yang menyediakan jasa logging, mereka berikan kita 1 URL dan kita bisa kirim HTTP request ke URL tersebut untuk logging di cloud.

Jadi disini, kita buat 1 varilable var url = ‘http://mylogger.io/log’; Pada contoh ini, bayangkan kita akan kirimkan HTTP request ke URL ini. Kita juga membutuhkan function dengan nama log dengan input parameter message.

Di function ini kita akan kirimkan HTTP request

A picture containing drawing, player, ball

Description automatically generated

Supaya sederhana, karena kita hanya ingin memahami konsep modular dari node js, kita ga perlu pusing dengan prosedur HTTP request. Jadi sekarang, kita hanya akan log message di console.

A screenshot of a cell phone

Description automatically generated

Varilable url dan function log, keduanya hanya bisa diakses di modul ini saja. Mereka tidak exist, ataupun diakses diluar dari file ini. Tapi di app.js, which is modul utama kita, kita ingin gunakan function yang ada di logger.js. Untuk itu, kita harus membuat function log ini, public.

Kita akan set module.exports.log, jadi kita tambahkan function log ke exports object, dan kita set ke function log yang sudah kita define di atas. Dengan kata lain, object yang kita export disini punya single method yang kita sebut, log.

A screenshot of a cell phone

Description automatically generated

Hal yang sama, klo kita mau export varilable url,

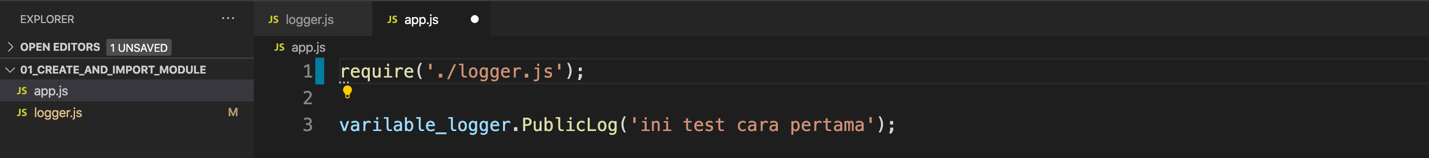
A screenshot of a cell phone

Description automatically generated

Tentu kita bisa mengubah nama yang mau kita export keluar. Contoh, internally, kita sebut varilabel url sebagai url, tapi saat kita export, kita mau menyebutnya endPoint.

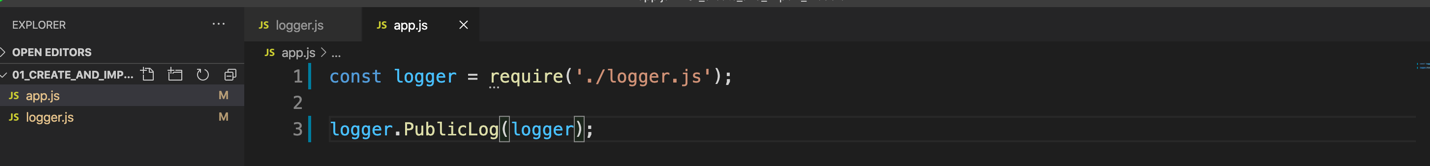
# LOADING A MODULE

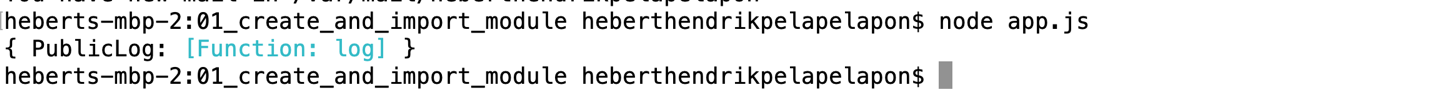
Untuk load sebuah modul, kita gunakan require function.



Function ini membutuhkan 1 parameter, yaitu nama atau path dari modul yang mau kita load. Klo kita perhatikan, app module dan logger module ada di folder yang sama. Jadi kita gunakan ./logger.js, atau secara singkat kita bisa tuliskan ./logger, karena node akan mengasumsikan ini adalah javascript file dan akan secara otomatis diberikan extensi .js di belakang. Kalau ada di subfolder maka kita bisa gunakan ./subfolder/logger, kalua dia ada di parent folder kita bisa gunakan ../logger.

Nah require function akan return object yang di export dari modul target. Sebelumnya kita melihat klo export object nya kosong, karena kita belum load modul apapun di main module app.js kita. Tapi karena sekarang kita sudah import atau log modul logger, maka object export akan terlihat. Mari kita buktikan dengan run app dibawah ini.





Kita punya object, dengan single function PublicLog. Dengan begitu, kita bisa panggil function ini di app.js.

A screenshot of a cell phone

Description automatically generated

Begini cara kita untuk bekerja dengan modul di nodejs. Saat kita sudah membuat modul, kita export, 1 atau lebih function, lalu load modul dengan function require.

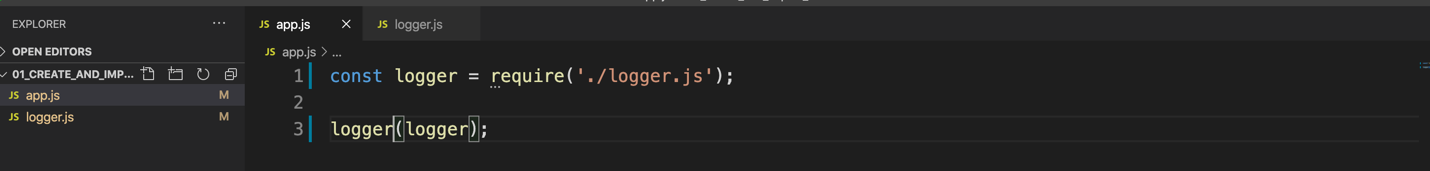
Nah di javascript versi terbaru, kita bisa menggunakan constant untuk membuat sebuah variable. Nah klo kita coba untuk load module dengan require function, akan lebih baik apabila kita simpan result nya menggunakan const. Alasannya adalah karena kita tidak mau secara tidak sengaja override variable logger, seperti ini.

Ada kalanya, kita tidak butuh export object dari modul, karena hanya ada 1 function saja disana. Misalnya, di logger module, kita ga butuh object karena kita cuma punya 1 function. Object akan berguna, kalau kita punya lebih dari 1 function. Nah dalam kasus ini, daripada kita export object, kita bisa export single function.

A screenshot of a cell phone

Description automatically generated

Di app.js, logger sudah bukan lagi object, tapi jadi sebuah function yang bisa kita panggil langsung.



Jadi di nodejs, kita bisa export single function, atau object.

# PATH MODULE

DI pelajaran kemarin, sempat disebutkan bahwa di nodejs, kita punya modul-modul yang berguna di core nodejs. Kita bisa buka nodejs.org. dan klik docs, dan di sebelah kiri, klik versi LTS.

Ada banyak list yang bisa kalian pakai disini, tapi saya coba highlight beberapa yang penting. Seperti, kita punya File system untuk bekerja dengan file dan directory,

kita punya HTTP untuk membuat web server yang akan listening HTTP request

Kita punya OS untuk bekerja dengan operating system

Kita punya path.

Kita punya process yang bisa menjabarkan informasi tentang proses yang terjadi di server

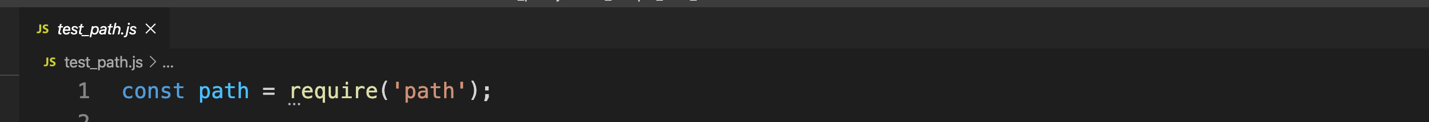
Dan modul-modul lainnya.

Nah di pelajaran kali ini, kita akan coba gunakan path. Di dokumentasi pada website nodejs, dijabarkan apa saja yang tergabung di dalam path. Kali ini, kita akan coba gunakan function parse. Nah kalau kita scroll kebawah, kita sudah pernah pakai require function untuk load module.

A screenshot of a social media post

Description automatically generated

Jadi mari kita coba pakai path ini di dalam nodejs. Kita buat 1 file dengan nama test\_path.js, dan ketik require(‘path’); Dalam kasus ini, nodejs akan mengasumsikan kalua ini adalah sebuah built in module. Dan kita simpan di dalam constant path.



Method yang akan kita coba gunakan adalah parse method.

A picture containing drawing

Description automatically generated

Coba run app ini.

A screenshot of a cell phone

Description automatically generated

# OS MODULE

Sekarang kita coba gunakan OS module. Kalau kita kembali lagi ke nodejs.org, pilih OS, kita bisa lihat function-function yang available di modul ini.

Contohnya kita punya freemem untuk melihat freememory di server,

Kita punya totalmem untuk melihat total memory

Kita punya userInfo untuk melihat informasi terkait current user

Kita punya uptime untuk melihat uptime dari mesin

Dan lain sebagainya

Mari kita coba pakai beberapa function disini. Kalau kita scroll kebawah, sama seperti path sebelumnya, kita gunakan require.

A picture containing food

Description automatically generated

\*\* run app \*\*

Nah disini bisa kita lihat, sebelum adanya nodejs, dan hanya menggunakan javascript di web browser, akses ke operating system seperti ini, tidak mudah. Dan rata-rata, di web browser, kita hanya bisa gunakan object window / document.

# FILE SYSTEM MODULE

Kali ini kita akan belajar bagaimana cara bekerja dengan file di nodejs. Kembali ke nodjs.org, kita pilih file system, dan bisa kita lihat kalua kita punya banyak sekali function yang bisa kita gunakan terkait file system.

Jadi kembali ke codingan, dan coba ketik fs.

A screenshot of a cell phone

Description automatically generated

Kalau kita lihat disini, ingat waktu introduction kemarin, kita mengenal ada 2 model kerja dari node. Ada synchronous (atau blocking), ada yang asynchronous (atau non blocking). Disini ada access (which is asynchronous method), dan ada accesssync (which is synchronouse method). Di real world application, kita harus menghindari penggunaan synchronous. Sebaiknya menggunakan asinkronus karena non-blocking.

Sebagai contoh, kita coba gunakan function readdir.

Pertama, kita coba gunakan sync function. Readdirsync. Parameter pertama, kita tentukan path Ini akan return, semua file dan folder di path ini.

A picture containing light

Description automatically generated

Sekarang kita akan coba menggunakan async function. Sama seperti sebelumnya, parameter pertama adalah path. Nah tapi semua async function membutuhkan function lain sebagai parameter selanjutnya. Node akan memanggil function tersebut saat async operation nya selesai. Kita sebut function ini sebagai, callback. Callback membutuhkan 2 paramter, yaitu error, dan result. Which is di case ini adalah string array.

A screenshot of a cell phone screen with text

Description automatically generated

\*\* run app\*\*

Sekarang kita coba untuk bikin error. Ganti path jadi path yang ga exist. \*\* run app \*\*

Kali ini kita akan mendapatkan error.

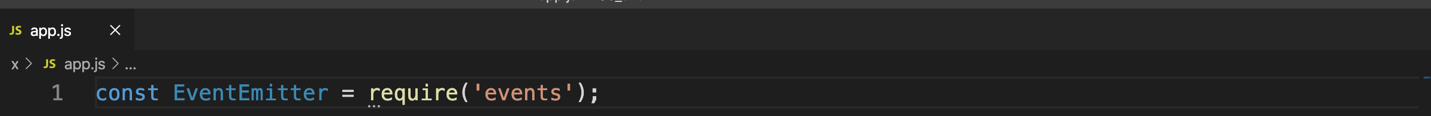
# EVENT

Salah satu fitur utama dalam node adalah konsep event. Banyak functionality di node menggunakan event.

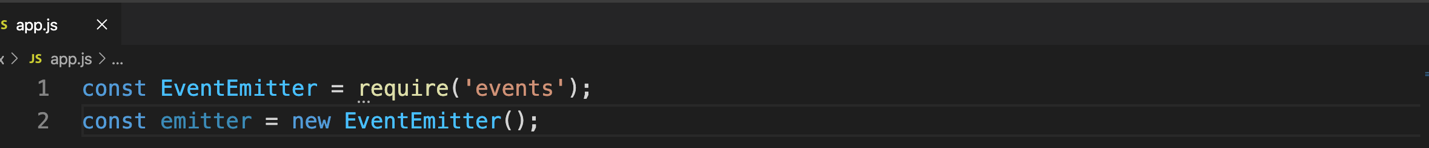
Event pada dasarnya adalah signal yang mengindikasikan sesuatu telah terjadi di aplikasi kalian. Saya sempat sounding sebelumnya, nodejs punya modul HTTP yang bisa kita gunakan untuk membuat web server. Nah pada saat web server ini active, server akan listening ke port specific yang kita tentukan. Begitu ada request masuk, maka node akan membuat 1 event, eksekusi, dan return respond.

Kembali ke nodejs documentation, events. Di dalam modul ini, kita punya 1 class yang Namanya eventemitter. Ini adalah salah satu modul inti, dan banyak class di nodejs yang berbasiskan eventemitter ini.

Back to code editor, kita load event module.



Untuk dapat menggunakan EventEmitter kita harus membuat instance dari Class EventEmitter.



EventEmiiter punya banyak method, tapi kebanyakan, kita sering pakai 2 method saja, yaitu emit dan on. Emit digunakan untuk memberikan signal, atau membuat event. Nah function emit, butuh 1 parameter, yaitu nama dari sebuah event. Misalnya kita buat nama event nya messageLogged.

A close up of a logo

Description automatically generated

Nah kalau kita run app ini, ga ada yang muncul. \*\* run app \*\*

Ini karena, kita memberikan signal klo event ini terjadi, tapi ga ada di source code kita, kita set listener dari event itu. Basically listener adalah function yang akan dipanggil klo event terjadi.

A screenshot of a cell phone

Description automatically generated

Untuk membuat listener atau respond dari event, kita buat dengan method **on.** On memiliki 2 parameter. Parameter pertama adalah nama dari event, yang kedua adalah callback.

A screenshot of a cell phone

Description automatically generated

Untuk saat ini, kita hanya akan coba log tulisan di console.

A screenshot of a cell phone

Description automatically generated

\*\* run app \*\*

Urutan menjadi sangat penting disini. Pendaftaran event, harus dibuat, sebelum pemanggilan event dibuat. Kalau terbalik, maka tidak akan terjadi apa-apa.

# EVENT ARGUMENTS

Cukup sering saat kita memanggil sebuah event, kita ingin send data terkait event tersebut. Contoh, di logger modul kita saat kita mau log message, remote logging service kita akan generate ID.

Jadi di argument pada pemanggilan event, kita bisa masukan parameter data. Contoh, id, dan url.

A screenshot of a cell phone

Description automatically generated

Tapi cara ini tidaklah bagus. Daripada kirim berbagai data terpisah, best practice nya adalah kirim data dalam bentuk json object.

A screenshot of a cell phone

Description automatically generated

Nah kita refer object tersebut, sebagai event arugement.

Sebelumnya diatas, kita tidak memberikan argument apa2. Jadi disini kita bisa tambahkan parameter arg . Nama tidaklah penting, tapi di naming convention, rata2 orang akan menggunakan e untuk refer ke event, atau eventArg. Penamaan apapun yang kalian gunakan, bebas.

A screenshot of a cell phone

Description automatically generated

\*\* run app \*\*

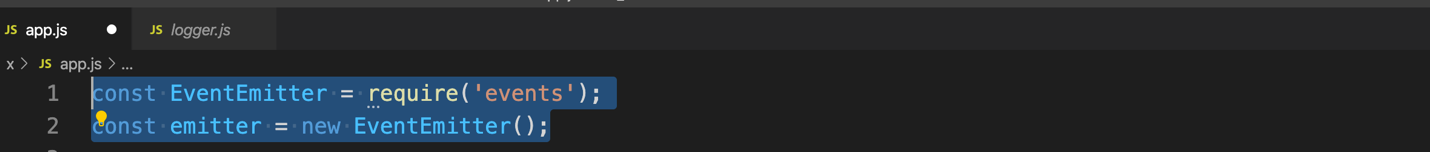
# EXTENDING EVENT EMITTER

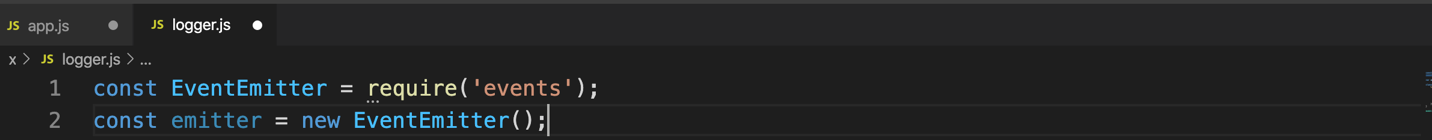
Di dunia nyata, bukan best practice kita selalu import eventemitter di setiap module. Jadi kita hanya akan pasang event emitter di class yang kita butuhkan.

Mari kita kembali ke logger.js. Di modul ini kita sudah export simple function log, dan kita display message nya di console.

Tapi, setelah ini, Kita mau memanggil sebuah event. Dan nanti, di app module, kita akan listen terhadap event tersebut dan lakukan sesuatu.

Kita kembali ke app module. Copy 2 baris loader event emitter ke logger.





Kembali ke app module. Kita juga akan pindahkan bagian register listener, ke logger. Karena perintah ini tidak seharusnya ada disini. Logger module lah yang seharusnya memanggil perintah ini dan kirim signal event. \*\* cut, pidahkan ke logger module di bawah console.log \*\*

A screenshot of a cell phone

Description automatically generated

Sekarang, kalau kita kembali ke app module, kita akan load module logger, supaya kita bisa panggil function nya dari app js.

Pada contoh ini, saat kita jalankan, hanya akan muncul tulisan message. Pada bagian event listener, tidak akan terpanggil. Nanti akan saya jelaskan alasannya.

A screenshot of a cell phone

Description automatically generated

Alasan kenapa event listener ini tidak dipanggil karena disini kita bekerja dengan 2 eventemitter berbeda. Di appjs, kita punya event emitter. Di logger module, kita juga punya event emitter lagi.

Di logger, kita menggunakan emitter untuk signal event, di app js kita menggunakan emitter untuk handle event. 2 hal ini berbeda. Saat kita gunakan buat listener di appjs, listener tersebut hanya di register dengan event emitter appjs, which is berbeda dengan logger event emitter.

Untuk itu, kita akan membuat class yang punya semua fungsi dari event emitter tapi juga punya additional capability, dalam kasus ini, logging data.

Sekarang kita akan buat sebuah class.

A screenshot of a cell phone

Description automatically generated

Dan kita akan pindahkan semua function log, ke dalam class tersebut.

A screenshot of a cell phone

Description automatically generated

Kalau kita perhatikan, kata akan mendapatkan error di tulisan function ini. Kenapa? Karena saat kita buat function di dalam class, kita tidak perlu tulisan function. Dan kita akan sebut function di dalam class, sebagai method.

Pada bagian bawah, instead of kita export function, kita akan export class.

A screenshot of a cell phone

Description automatically generated

Nah, kita ingin Logger class, punya semua capability dari event emitter. Cara untuk melakukan itu adalah dengan memberikan keyword extends keyword.

A screenshot of a cell phone

Description automatically generated

Dengan perubahan ini, sekarang logger class jadi sudah punya function log, yang akan kita panggil untuk logging. Dan fungsi2 event emitter.

Berhubung kita sudah extends eventemitter object, instead of pakai emitter, kita bisa pakai this.

Kita tidak perlu lagi pakai const emitter, karena kita sudah punya capability dari EventEmitter.

Kita sudah selesai dari logger module. Sekarang kita kembali ke app module.

Karena skarang kita import Logger class dan bukan function lagi, maka akan mengganti varilable log, jadi Logger. Perhatikan, untuk naming convention, sebaiknya kalian menggunakan Huruf besar untuk varilable atau const yang merupakan Class. Dengan demikian, akan lebih mudah bagi kalian untuk tau klo itu adalah Class. Lalu kita buat sebuah object, logger.

A screenshot of a cell phone screen with text

Description automatically generated

Lalu untuk log message kita panggil, logger.log. Sama dengan perubahan yang kita lakukan di logger module, kita tidak perlu lagi pakai emiiter. Kita akan gunakan emitter dari logger object. Jadi, kita akan register listener di app module ini, di logger object.

Sekarang, kita akan pindahkan code listener.

A screenshot of a cell phone

Description automatically generated

\*\* ubah emitter.on jadi logger.on \*\* dengan begini, kurang lebih jadinya saat event messageLogged terjadi, I want to execute this code console.log(‘listener called’). Kita tidak perlu const emitter object, dan kita bisa hapus baris tersebut.

A screenshot of a cell phone

Description automatically generated

Now, saat kita jalankan aplikasi ini, kita akan melihat message di console, tapi juga karena kita menggunakan object logger yang sama untuk register event, kita juga akan lihat message “listener called”, di console.

Jadi kesimpulannya, apabila kalian ingin membuat event di aplikasi kalian untuk memberikan signal bahwa sesuatu telah terjadi , kalian perlu membuat class yang extends EventEmitter. Dengan demikian, class tersebut akan punya semua functionality yang dimiliki oleh event emitter tapi kalian juga bisa menambahkan additional function sesuai kebutuhan kalian. Dalam kasus ini, kita punya ability untuk log message. Dan di dalam class ini, kapanpun kita butuh memanggil sebuah event, kita tinggal pakai this.emmit. karena “this” mereferensikan Logger class, yang sudah extends EventEmitter.

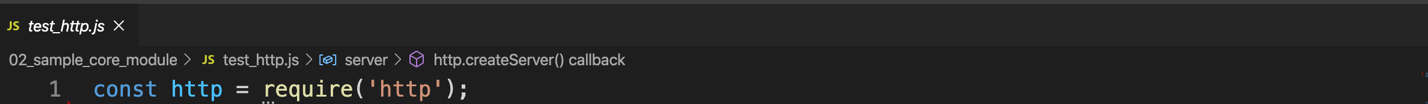
# HTTP MODULE

Salah satu module utama juga di dalam node adalah HTTP module, yang kita gunakan untuk membuat networking application. Contoh, kita membuat web server, yang listen HTTP request di port yang kita siapkan. Dengan demikian, kita bisa dengan mudah membuat backend service untuk client application, seperti aplikasi yang dibuat menggunakan react, angular js, dll.

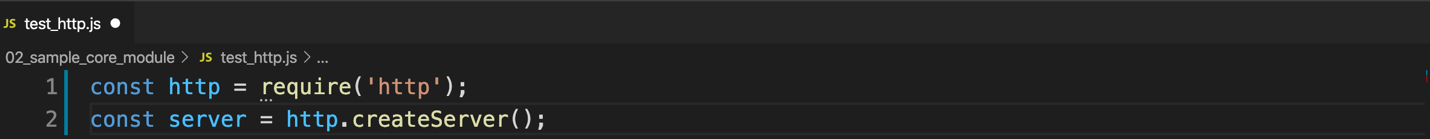
Jadi sekali lagi, kita kembali ke nodejsorg. Di daftar module, kita bisa click di HTTP.

Kembali ke code editor.

Mari kita load http module di test\_http.js.



Untuk kali ini, kita coba pakai salah satu function yang ada di object http. Yaitu http.createServer. Dan kita akan simpan ini di dalam 1 const



Seperti yang sempat saya jelaskan di awal tentang event, bahwa event adalah salah satu modul penting. Banyak modul-modul node yang akan menggunakan atau terintegrasi dengan event. Salah satunya adalah http ini. Kalau kita perhatikan, kita bisa menggunakan method on, add, emit, dll.

A picture containing screenshot, sitting, player

Description automatically generated

Sekarang kita bisa gunakan method listen, dan tentukan angka port nya, let’s say 3000. Dan terakhir, kita log, dengan console.log(‘Listening on port 3000…’);

A picture containing screenshot, drawing

Description automatically generated

Saat kita jalankan ini, maka server akan listen di port 3000. Dan seperti yang saya sampaikan sebelumnya, setiap kali kita membuat sebuah connection atau request, server ini akan memanggil sebuah event. Jadi kita bisa gunakan “on” method, untuk handle event tersebut. Jadi sebelum listening, kita akan buat handler nya terlebih dahulu.

A screenshot of a cell phone

Description automatically generated

Nama event yang kita butuhkan adalah “connection”. By the way, kita ga perlu menghafal semua nama-nama ini, karena pada akhirnya, kalian akan refer ke dokumentasi. Parameter kedua adalah callback function socket.

A screenshot of a cell phone

Description automatically generated

\*\* run app \*\* \*\*access web browser http://localhost:3000\*\*

Di aplikasi pada umumnya, kita tidak akan respond terhadap connection event untuk membuat HTTP request. Ini hanya untuk menjelaskan bagaimana http module bekerja. Jadi kita hapus 3 haris ini \*\* hapus 3 baris function new connection \*\*.

A screenshot of a cell phone

Description automatically generated

Yang akan kita lakukan adalah, memasukan callback function pada method createServer. Function ini akan memiliki 2 parameter, req (request) dan res (respond)

A screenshot of a cell phone

Description automatically generated

Nah disini, ketimbang kita bekerja dengan socket, kita bisa bekerja dengan request atau respond object. Disini mari kita coba klo request dot url == ‘/’, Maka kita akan kirimkan sesuatu ke client. Contohnya, kita akan print “Hello World”. Dan kita akhiri respond nya.

A screenshot of a cell phone

Description automatically generated

\*\* run app \*\* \*\*run web browser \*\*

Nah, kalau kita mau membuat backend services, seperti API, kita harus handle berbagai variasi route disini. Contoh, kita bisa punya if block lainnya.

A screenshot of a cell phone

Description automatically generated