# Pregled ogrodij za NodeJs

Da lahko na zaledju pri delu z omrežjem oz. izdelovanju funkcij spletnega strežnika v NodeJs pišemo krajšo, urejeno kodo ter pohitrimo razvoj naše aplikacije, je dobro uporabiti eno od ogrodij ustvarjenih prav v ta namen. Med funkcije, ki jih potrebujemo za razvoj in jih ta programska oprema navadno zagotavlja, spadajo olajšano delo s HTTP (vzpostavitev povezave, obdelava zahtev ter odgovorov), usmerjanje (procesiranje URL), podpora na vmesnemu nivoju (middleware) in številne druge.

## So ogrodja obveznost ali izbira?

Aplikacijo je mogoče razviti brez uporabe ogrodja, vendar moramo vedeti, da Node neposredno ne vsebuje podpore za spletni razvoj, predlog za dinamično ustvarjanje odgovorov itd.. V primeru, da se odločimo za razvoj brez ogrodja, moramo vso podporo za to načrtovati sami.

Ob primerjavi kode preprostih strežnikov z enakimi funkcijami, prvega s uporabo Expressa in drugega brez, so očitno vidne razlike v sami sintaksi oz. kompleksnosti le te. Z rastjo naše aplikacije v kode razsežnosti več sto ali tisoč vrstic, bi bilo zelo težko slediti poteku programa. Z uporabo ogrodja v kombinaciji s MVC načrtovalskim vzorcem, torej razbijanjem aplikacije na več manjših, preglednih delov, si lahko močno olajšamo razvoj (še posebej kadar na projektu dela več razvijalcev, kot to tudi poteka v našem primeru). Vseeno je izvajanje programa, napisanega v »nativnem« nodejs okolju, prilagojenega reševanju našega specifičnega problema, pogosto občutno hitrejše, kot če vmes stoji še neko ogrodje.

## Izbira ogrodja

Z porastom popularnosti okolja NodeJs za izdelovanje programskih rešitev na zalednem delu se je na trgu pojavilo in se še pojavlja veliko število različnih ogrodij. Med bolj priljubljene štejemo Express, Hapi, Meteor, Sails, Koa, Feathers, Adonis, LoopBack in še. Ko se odločamo za uporabo specifičnega ogrodja, moramo pomisliti na tako tehnične, kot tudi poslovne potrebe naše aplikacije. Upoštevati moramo faktorje, kot so hitrosti izvajanja in učnega procesa razvijalcev, struktura izvorne kode, fleksibilnost, razširljivost (knjižnice/moduli), razvojna skupnost, dostopna dokumentacija,…

## Express

Ta trenutek najbolj priljubljeno spletno aplikacijsko ogrodje je zastonjska programska oprema, ki je bila izdana pod MIT licenco leta 2010. Med največje prednosti ogrodja zagotovo spada prav njegova razširjenost. Najbolj uporabljano ogrodje seveda uporablja velika množica ljudi, to posledično zagotavlja več dokumentacije, rednega vzdrževanja, podpore in sprotne odprave napak. Ob razvoju hitreje naletimo na ljudi s težavami podobnimi našim in s tem tudi njihovih rešitev. Največkrat ga najdemo v kombinaciji s MongoDB in AngularJs, kot sestavni del MEAN sklada pri full stack razvoju. Razvijalci lahko razvijajo tako zaledni, kot tudi čelni del, ker pa je učni proces uporabe Expressa relativno kratek (dokler razumemo javascript, ter osnovne značilnosti protokolov aplikacijske plasti) se iz zalednega dela lažje znajdejo tudi razvijalci iz drugih razvojnih področij (čelnega dela). Preprost strežnik lahko postavimo v le nekaj minutah.

Zahteve lahko speljemo na različne poti (usmerjanje), odvisno od metode http (get, post, put, delete, all) ali po potrebi, glede na parametre v URL-ju. Zahtevo predstavlja objekt req (http request), odgovor pa objekt res (http response).

Ko govorimo o middlewareu mislimo na sloje procesiranja zahteve (obdelovanje na vmesnem nivoju) oz. skozi katere korake mora zahteva potovati, od prejetja, pa do poslanega odgovora. Za uporabo v Expressu imamo na voljo veliko število raznih modulov (javascript knjižnic) različnih funkcionalnosti, od dela s piškotki in sejami (cookie-parser, session), avtentikacijo in avtorizacijo uporabnika (express-stormpath), prejemanje zahtev iz različnih domen (cors), varnostjo (helmet), delo z zahtevami, ki v telesu ne vsebujejo le besedila (multer), itd.. Torej vmesna programska oprema lahko vrši kakršno koli programsko kodo (večinoma spreminja objekte zahteve ali odgovora). Na samem koncu se cikel request-response ali zaključi (send(),end(),render(),..), ali pa se kliče naslednja vmesna programska oprema v verigi (kot je določena v usmerniku) s klicem funkcije next(). Standardne module koristimo s namestitvijo (npm) in vključitvijo, nekateri, kot so npr. static, json, urlencode, pa so v express že vgrajeni.

Ni nujno, da so naši odgovori vedno preprosti json, namreč odgovarjamo lahko tudi s stranmi dinamične vsebine. Na strežniški napravi hranimo šablone spletnih strani, katere nato ob zagonu predprocesorja napolnimo s podatki, pridobljenimi iz podatkovne baze (view engines/template engines). Med bolj poznane spadajo hbs, ejs, pug, ...

## Hapi (http API)

Med Hapi in Express je več podobnosti, kot razlik. Je nekoliko novejše ogrodje, napram starejšemu, ustaljenemu, skoraj standariziranemu expressu.Express uporablja minimalističen (lightweight) pristop, je tanek sloj na vrhu osnovne funkcijonalosti NodeJs in se predvsem zanaša na middleware. Pri Hapiju pojma middleware ne poznamo, večina funkcij oz. programov, ki jih pri obdelovanju zahtev in odgovorov potrebujemo, je integriranih in zato navadno dodatna vljučevanja niso potrebna. Če potrebujemo funkcijonalosti, ki jih hapi v osnovi ne nudi, pa vključimo plugine ali razširitve (ang. pluggins, extensions).

Plugini (moduli) so drug drugega neodvisni, zato ni nameščanja dodatnih, za delovanje potrebnih programov (dependencies), vse je namreč zapakirano v en paket. Nekateri izmed mnogih so: joi , boom, glue, qs, vision,.. Sproti jih dodajamo s klicem funkcije set().

Razširitve oz. razširitvene točke (Extension Points) so členi verige funkcij, ki tvorijo življensko cikel zahteve na našem strežniku. Privzeto obstaja 7 točk, po vrsti glede na izvajanje razvrščene so: onRequest, OnPreAuth, onCredentials, onPostAuth, onPreHandler, onPostHandler, onPreResponse. Funkcionalnosti razširjamo s dodajanjem svoje kode v te točke oz. funkcije (podobno, kot pri razvoju aplikacj za android). To storimo s klicem funkcije ext(), kjer kot parameter podamo ime razširitvene točke.

Usmerjanje poteka s klicem funkcije route nad objektov server (glavni objekt, okoli katerega gradimo aplikacijo, kot app pri expressu), nato navedemo vse dodatne specifikacije: metodo, pot, katera funkcija naj se izvede. Sintaksa je bolj »izrazita«, zaradi tega tudi daljša.

Ob primerjavi hitrosti izvajanja express in hapi strežnikov ugotovimo, da hapi deluje nekoliko počasneje.

## Koa

Za razvojem ogrodja Koa stojijo razvijalci Expressa. Razlog, zakaj koa ni le nova verzija Expressa leži v tem, da so razlike med ogrodjema preprosto prevelike. Neko Aplikacijo, napisano s ogrodjem Express, bi bilo potrebno, za prenos v Koa ogrodje, napisati praktično na novo, zato so to programsko opremo raje izdali kot knjižnico zase.

Razlika je že v samem cilju ogrodij: cilj Expressa je nadgradnja oz. prilagoditev obstoječih Node funkcionalnosti, na voljo imamo dodatne metode za manipulacijo nad req in res objektoma in za usmerjanje. Cilj Koe je med tem zamenjati Node funkcionalnosti, Koa namreč privzeta req in res objekta ne nadgradi, pač pa ju nadomesti z njuno abstrakcijo, objektom ctx (kontekst) in njegovima lastnostima ctx.request in ctx.response. Tradicionalne funkcije vmesnega nivoja niso podprte (funkcije s argumenti tipa res,req,next), ob razvoju se moramo tudi izogniti uporabi Node lastnosti, kot so statusCode, write, writeHead, end in drugih.

Koa je izredno minimalistično ogrodje, sestavlja ga manj kot tisoč vrstic kode. Ne vključuje podpore za usmerjanje, za prenos datotek itd.. Pri izdelovanju potrebnih programov za upravljanje z zahtevami in odgovori nam daje prosto pot, zato je ogrodje bolj primerno za že izkušene razvijalce, ki Node nekoliko bolje poznajo (četudi razvoj aplikacij poteka bistveno drugače, kot z Expressom ali brez ogrodja).

Uporaba ogrodja Koa se priporoča, kadar izdelujemo aplikacijo, ki ne temelji na uporabi brskalnika (zaradi neobstoja pripravljene podpore za usmerjanje) in kadar je hitro izvajanje ključnega pomena, saj Koa omogoča nekoliko hitrejše delovanje.

## Sails

MVC ogrodje SailsJs je dodatna plast, zgrajena nad ExpressJs, zato se, če si za ogrodje izberemo Sails, priporoča predhodnje znanje uporabe Expressa. Na njegovo uporabo bi se nahitreje navadili razvijalci, ki imajo izkušnje s Ruby on Rails, katerega MVC zasnovo (poleg dodane podpore za porebe modernih podatkovno usmerjenih APIjev) poskuša emulirati.

Kjer Express za ORM, torej interakcijo s PB MongoDB uporablja Mongoose, Sails uporablja Waterline, katerega velika prednost je fleksibilnost: koncept adapterjev. S uporabo teh lahko uporabljamo širok nabor različnih PB hkrati, ali brez težav preklapljamo med njimi (MySql, PostgreSql, MongoDB, Redis, …). Pri tem kodo, ki se tiče uporabe PB pišemo samo enkrat, z nekaj manjšimi popravki.

Sails se za uporabo priporoča, kadar razvijamo realnočasovne aplikacije, obdelujemo in hranimo velike količine podatkov (big data).

## Zakaj Express in ne njegove alternative?

Express je od vseh naštetih ogrodij najstarejši in najbolj uveljavljen. Kot najbolj prepoznavno, skoraj standarizirano ogrodje ima trenutno tudi največ uporabnikov, njegove osnove smo spoznali tudi pri predmetu Razvoj Aplikacij Za Internet, kjer smo ga uporabili pri dveh vajah. Na spletu ne manjka člankov, nasvetov, opisov ter razne druge dokumentacije kar se tiče uporabe tega ogrodja. Seveda obstaja verjetnost, da ga katera od alternativ v prihodosti nadomesti, kot »go to« NodeJs ogrodje, toda večina teh se sklicuje na Express: ali ga nadgrajuje, ali pa poskuša odpraviti njegove napake, dobro ga je poznati v vsakem primeru.

## Viri

Primerjava express, hapi in koa:

<https://medium.com/@theomalaper.cognez/express-vs-koa-and-hapi-a2c65f949b78>

Express template engines: <https://github.com/expressjs/express/wiki#template-engines>

Hapi plugins: <https://hapi.dev/plugins/#universe>

Opis uporabe funkcionalnosti Expressa:

<https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction>

Primerjava Hapi in Express ogrodij: <https://www.konstantinfo.com/blog/hapi-vs-express/>