VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
INFORMATIKOS KATEDRA

Baigiamasis bakalauro darbas

Javascript kalbos potencialas kuriant pilnas aplikacijas

(Javascript potential for creating full-stack applications)

Atliko: 4 kurso, 4 grupės studentas:
Rokas Tulaba

Darbo vadovas:
Liutauras Ričkus
Recenzentė:
Lina Povilavičiūtė

Vilnius

2020

Santrauka

Šio bakalaurinio darbo „Javascript kalbos potencialas kuriant pilnas aplikacijas“ autorius yra Rokas Tulaba. Darbe yra atliekamas tyrimas ir eksperimentai, kuriais siekama nustatyti JavaScript kalbos potencialą kuriant pilnas aplikacijas.

Darbas tiria visas pilnų aplikacijų aktualias sritis – vartotojo sąsajos, loginės sisteminės dalies kūrimą ir bendrus technologinius pranašumus. Tai yra atliekama tiriant 6 pagrindinius aspektus: vieno puslapio aplikacijas, kaip JavaScript kalba padeda kurti mikroservisus, kodo rašymo ir aplikacijos efektyvumą, JavaScript kalbos privalumus, kurie yra būdingi tik šiai kalbai, programinės kalbos efektyvumą ir technologinį populiarumą. Tiriant programinės kalbos efektyvumą yra atliekami eksperimentai.

Darbo rezultatas: Atlikus tyrimus ir eksperimentus nustatyti ar JavaScript iš tikrųjų turi didelį potencialą kuriant pilnas aplikacijas.

ABSTRACT

This bachelor thesis „Javascript potential for creating full-stack applications“ author is Rokas Tulaba. Research and experiments will be performed in the thesis, which purpose is to determine the potential of JavaScript programming language potential for creating full stack applications.

The thesis investigates all areas important for full stack applications – front-end, back-end development and common technological advantages. That will be done by examining 6 main aspects: One page applications, how JavaScript helps to create microservices, code writing and application effectiveness, advantages specific only to JavaScript language, programming language effectiveness experiments and technological popularity in the IT community.

The thesis result: Determination whether JavaScript actually has a big potential for creating full-stack applications.

Turinys

Įvadas

Įvairių aplikacijų - WEB, mobiliųjų telefonų, kompiuterių - kūrimo procesai smarkiai patobulėjo per paskutinį dešimtmetį. Anksčiau, mobiliosios ir darbalaukio programos turėdavo būti kuriamos kiekvienai operacinei sistemai atskirai, o jeigu buvo norima turėti ir tinklapį, tuomet prisidėdavo dar viena papildoma, atskirai programuojama aplikacija. Todėl naujai atsiradusios technologijos, padedančios išspręsti šią problemą buvo didelis šuolis į priekį aplikacijų pasaulyje. Viena iš tokių technologijų - JavaScript kalbos evoliucija iš naršyklės scenarijavimo (angl. scripting) kalbos į pilną, funkcionalią programavimo kalbą, kuri gali būti vykdoma bet kokioje aplinkoje, pavyzdžiui Linux, Windows, Android ar iOS/MacOS operacinėse sistemose. Ši evoliucija, dar kitaip žinoma, kaip Node.js technologijos atsiradimas. Ši technologija į pilnų aplikacijų programavimo pasaulį atnešė naują paradigmą, angliškai vadinamą “JavaScript everywhere”. Iš pradžių, tai buvo aktualu tik WEB aplikacijoms, kadangi WEB aplikacija galėjo būti pilnai - tiek loginė, tiek vartotojo sąsajos dalys - sukurta naudojantis tik viena programavimo kalba - JavaScript. Bėgant metams ir didėjant mobiliųjų programėlių paklausai buvo sukurtos tokios karkasinės sistemos kaip “React Native” ir “NativeScript”, kurios leido, pasitelkiant JavaScript kalbą, kurti mobilias aplikacijas visoms operacinėms sistemoms, su ta pačia, viena, kodo baze naudojant tas pačias praktikas, standartus ir įrankius, kaip ir kuriant WEB aplikacijas. Svarbu paminėti ir kitas technologijas, tokias kaip “Electron” - kuri leidžia kurti kompiuterines aplikacijas su JavaScript ir “Ionic” - išskirtinė karkasinė sistema, leidžianti kurti mobilias, kompiuterines ir WEB aplikacijas su viena kodo baze. Ir visa tai, tik JavaScript kalbos dėka. Būtent Node.js ir anksčiau išvardintų vartotojo sąsajos karkasinių sistemų kombinaciją leidžia kurti efektyvias, kokybiškas ir universalias aplikacijas naudojantis tik viena kalba. O technologijos kūrimas, viena kalba yra didelis privalumas, kadangi tuomet tos aplikacijos palaikymui ir kūrimui yra reikalingi tik vienos kalbos specialistai, reikia spręsti tik vieno tipo problemas ir yra sutaupoma daug resursų - pinigų ir laiko. Vis dėl to, kurti aplikaciją pilnai tik su viena kalba nėra būtinai geriau, nei tos pačios aplikacijos kūrimas skirtingomis programavimo kalbomis, jeigu kitų kalbų techniniai, populiarumo, dokumentaciniai ar kitokie aspektai pranoksta JavaScript kalbą.

Taigi, šio darbo tikslas yra ištirti JavaScript kalbos potencialą kuriant pilnas aplikacijas atsižvelgiant į pilnoms aplikacijoms svarbias sritis – vartotojo sąsajos, loginės sistemos dalies kūrimą ir bendrus technologinius pranašumus. Tai bus atliekama tiriant šiuos aspektus:

1. Vieno puslapio aplikacijas.
2. Kaip JavaScript kalba padeda kurti mikroservisus.
3. Kodo rašymo ir aplikacijos efektyvumą.
4. JavaScript kalbos privalumus, kurie yra būdingi tik šiai kalbai.
5. Programinės kalbos efektyvumą.
6. Technologinį populiarumą.

1 JavaScript kalba

JavaScript programavimo kalba, dažnai žymima tiesiog trumpiniu JS, yra aukšto lygio, paleidimo laiku kompiliuojama (angl. just-in-time compilation) kalba, kuri palaiko skirtingas programavimo paradigmas - procedūrinį, funkcinį, ir objektinį stilius. Ji buvo sukurta pagal ECMAScript (ES) standartą. Yra ir daugiau kalbų sukurtų pagal šį standartą, tačiau JavaScript yra labiausiai žinoma. JavaScript 1995m., sukūrė Brendan Eich, dirbdamas kompanijoje Netscape, tam kad ji atliktų scenarijavimo rolę Java kalbai, tuo metu ji buvo pavadinta “LiveScript”. Vėliau, JavaScript pagrinde buvo naudojama tik programavimui vartotojo sąsajos dalyje naršyklėse, kaip scenarijavimo (angl. scripting) kalba. Ji buvo skirta interaktyvumo kūrimui statiniuose tinklapiuose. Tačiau, per paskutinius du dešimtmečius ji smarkiai patobulėjo ir yra naudojama programų sistemų, duomenų bazių kūrimui, mašinų mokymuisi, robotikoje ir pilnų aplikacijų kūrimui.

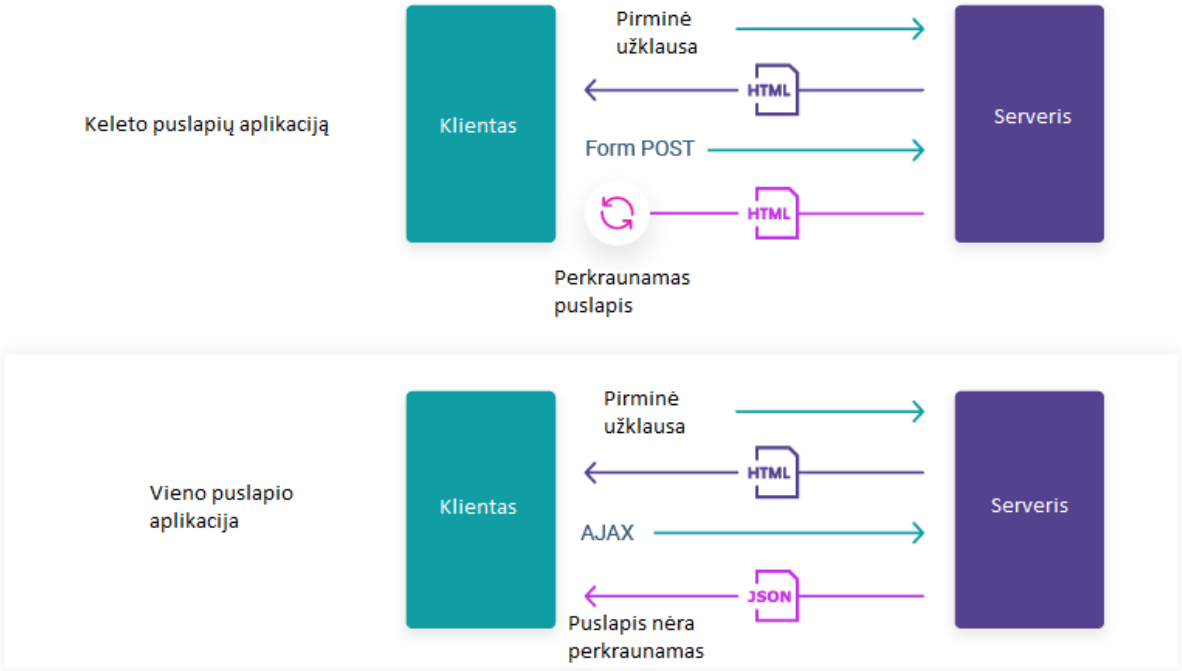
Iki kol buvo išleista Node.js ir įgijo savo populiarumą, įprasta buvo naudoti technologijų rinkinius, tokius kaip LAMP (Linux, Apache, MySQL, PHP), HOPE (Haskell, OpenBSD, PostgreSQL, Enginx), OPEN (OpenBSD, PostgreSQL, Enginx, Nim) kombinacijoje su JavaScript. Pasitelkiant JavaScript kalbą, taip pat yra sukurta daugybė karkasinių sistemų (angl. frameworks), skirtų vartotojo sąsajos kūrimui, kurios leido kurti sudėtingesnes, kokybiškesnes ir efektyvesnes vartotojo sąsajas skirtingiems įrenginiams, įvairiose aplinkose, tokias kaip:

- React, Angular, Vue.js, Knockout, Pug, EJS - WEB aplikacijoms.
- Electron, QT, GTK - darbalaukio aplikacijoms.
- React Native, NativeScript - mobilioms aplikacijoms.
- Ionic - Visoms platformoms, WEB, mobilioms ir kompiuterių, aplikacijoms.

Šių technologijų kombinacija, kaip pavyzdžiui, React + Node.js ar NativeScript + Vue.js + Node.js ir yra ta vieta, kur slypi šios kalbos potencialas. Įvairios JavaScript kalbos technologijų kombinacijos yra naudojamos tokių technologinių gigantų kaip PayPal, LinkedIn, Trello, Uber, Medium, Pinterest, Twitter, Reddit. Tokių kompanijų naudojimasis šiomis technologijomis yra neblogas rodiklis, jog JavaScript kalba puikiai tinka pilnų aplikacijų kūrimui ir turi didelį potencialą. Taip pat jų naudojimasis šia kalba skatina JavaScript kalbos ir su ja sukurtų technologijų ir profesionalų atsiradimą.

2 Vieno puslapio aplikacijos

Vieno puslapio aplikacija (toliau VPA) (angl. single page application) yra modernaus tipo WEB aplikacija kurią reikia užkrauti tik vieną kartą. Ji sąveikaudama su naršykle dinamiškai perrašinėja puslapio turinį - HTML, CSS kodą - vietoj standartinio, senojo tipo aplikacijos, kuomet norint patekti į naują puslapį, turi būti daroma kiekvieną kartą nauja užklausa (paveiksliukas 1). Dažniausiai tokio tipo aplikacijos yra kuriamos su tokiomis karkasinėmis sistemomis kaip: React, AngularJS, Vue.js, Ember.js, Knockout.js ir t.t.



Paveiksliukas 1. Keleto puslapių aplikacijų ir vienos puslapio aplikacijos veikimo principai

Tam tikrais atvejais, yra įmanoma sukurti netgi tokią VPA, jog visam tinklapiui apeiti ir visam jo funkcionalumui išnaudoti užtenka padaryti tik pirminę užklausą. Tokiais atvejais, visi reikalingi naršyti duomenys yra atsiunčiami kartu su pirmine užklausa ir tuomet visas puslapio turinys yra generuojamas iš tų pradinių atsiųstų duomenų ir nėra būtina atlikti nei vienos papildomos užklausos per visą naršymo sesiją, norint pamatyti kažkokį tinklapio turinį. Kitais atvejais, kai to nėra įmanoma padaryti, yra daroma atskira AJAX/WebSocket užklausa su kuria yra paprašoma tik specifinė reikalinga informaciją turiniui užkrauti.

Vienintelis būdas kurti VPA - kurti jas su JavaScript kalba. Vienintelė išimtis šiam teiginiui yra Flutter technologija, kuri kuriama su Dart kalba, tačiau turi kitų trūkumų tokių kaip:

1. Vis dar yra labai nauja, todėl:
    a. Turi žymiai daugiau įvairių klaidų/problemų.
    b. Gali sulaukti didesnių architektūrinių pakeitimų, kurių pasekoje jau sukurtos aplikacijos programinis kodas turėtų būti papildomai atnaujinamas.
2. Yra labai mažai įrankių kuriuos galėtų naudoti programuotojai, palyginus su kitomis kalbomis.
3. Didelis aplikacijos dydis.
4. Iš pradžių Flutter technologija buvo labiausiai orientuota tik į mobilias aplikacijas, vėliau į kompiuterines ir tik vėliau į WEB, todėl WEB aplikacijos kolkas yra silpniausia šios technologijos vieta.

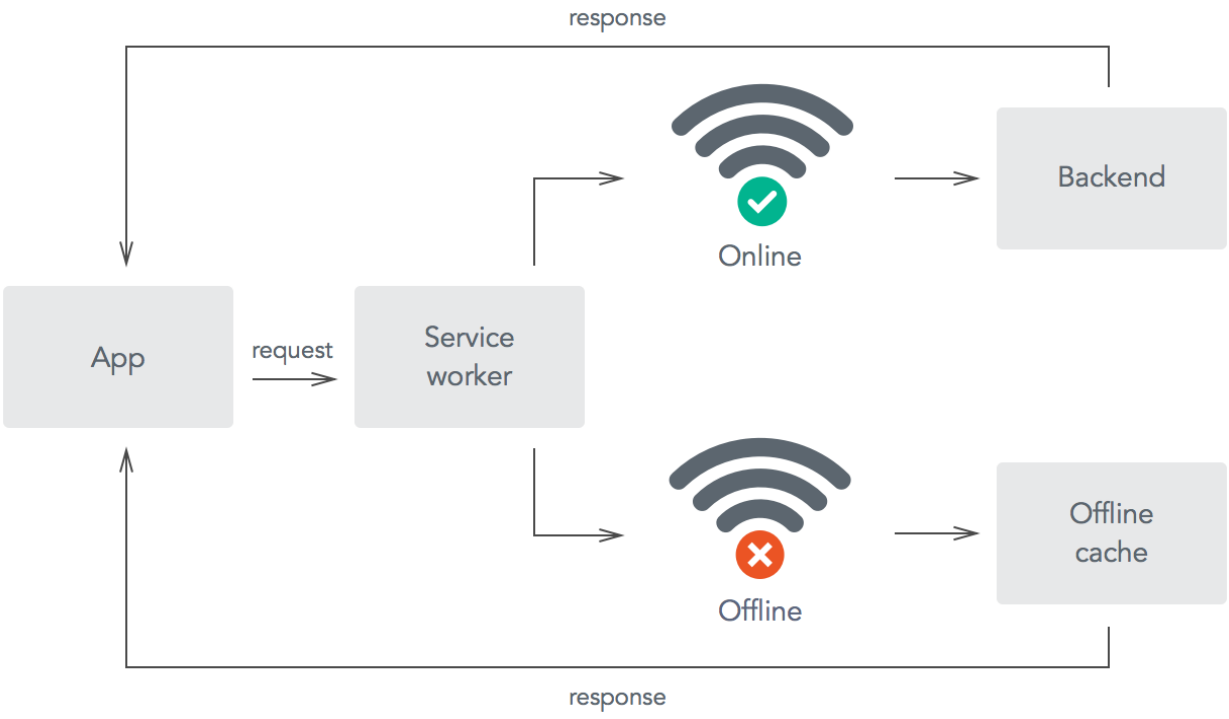
Apart Flutter technologijos (Dart kalbos) ir JavaScript kalbos, jokio kito būdo nėra. Tai yra labai svarbus faktas, nes vieno puslapio aplikacijos:

1. Gali būti lengvai paverčiamos į progresyvias WEB aplikacijas (toliau PWA) (angl. Progressive Web Application).
2. Yra greitesnės už kelių puslapių aplikacijas, nes reikia užkrauti tik vieną kartą ir toliau visas turinys keičiasi dinamiškai.
3. Klientas išnaudoja mažiau interneto duomenų.
4. Dėl savo greitumo, dinamiškumo ir mažesnio interneto duomenų išnaudojimo yra sukuriama geresnė vartotojo patirtis.
5. Serveris patiria mažesnę apkrova, todėl gali palaikyti daugiau vartotojų.
6. Technologiškai, dėl naudojamų karkasinių sistemų jas kurti, yra labai gerai pritaikytos sudėtingiems, realaus laiko atnaujinimų reikalaujančių tinklapių kūrimui, kaip pavyzdžiui: socialinės medijos, susirašinėjimo, žaidimų, kazino, aukcionų ir t.t. tinklapiams.

2.1 Progresyvios WEB aplikacijos

Tolimesnė galima optimizacija vieno puslapio aplikacijoms yra minėtosios progresyvios WEB aplikacijos. Tačiau kas tai yra? PWA - yra tiesiog WEB aplikacija, kuri gali pilnai funkcionuoti neegzistuojant interneto ryšiui ir suteikti stumiančių

pranešimų (angl. push notifications) ir aparatinės įrangos (kaip pavyzdžiui, kameros, akselerometro, giroskopo ir t.t sensoriai esantys mob. įrenginiuose) funkcionalumą. PWA yra kuriamos naudojant, vadinamą, aptarnaujančio darbuotojo (angl. service worker) programinį kodą. Šis kodas veikia kaip tarpininkas tarp WEB aplikacijos ir serverio, kuris nusprendžia kaip apdoroti užklausas kurias vartotojas nori atlikti, priklausomai nuo to, ar vartotojas yra prisijungęs prie interneto ar ne. Dažniausiai, jei vartotojas yra prisijungęs, tuomet aplikacija tiesiog praleis užklausą, jog ji galėtų pasiekti serverį ir tuo pat metu išsaugos tos užklausos atsakymą įrenginio atmintyje ateičiai, o jei vartotojas bus atsijungęs, tuomet tikrins ar turi atmintyje jau išsaugotą atsakymą tai užklausai ir jį atvaizduos. Tais atvejais, kai išsaugoto atsakymo nėra, tuomet atsakymas priklausys nuo tolimesnės aptarnaujančio darbuotojo programinio kodo implementacijos. Visą veikimo modelį galima matyti paveiksliuke 2.



Paveiksliukas 2. [Pja20]. Aptarnaujančio darbuotojo programinio kodo veikimo principas

Pagrindiniai šios technologijos privalumai:

a. Veikia nepriklausomai nuo interneto ryšio.
b. Greitesnė ir efektyvesnė aplikacija. [Api17]
c. Nereikia instaliuoti.

d. Gali veikti pilnai kaip mobili aplikacija, tačiau užima mažiau vietos, nereikia siųsti savo vartotojų per programėlių parduotuvę, programėles nereikia kelti į programėlių parduotuvę ir atlikinėti varginančias ir ilgai užtrunkančias patvirtinimo procedūras.
e. Padidina konversijas, vartotojų pasitenkinimą. [Api17]

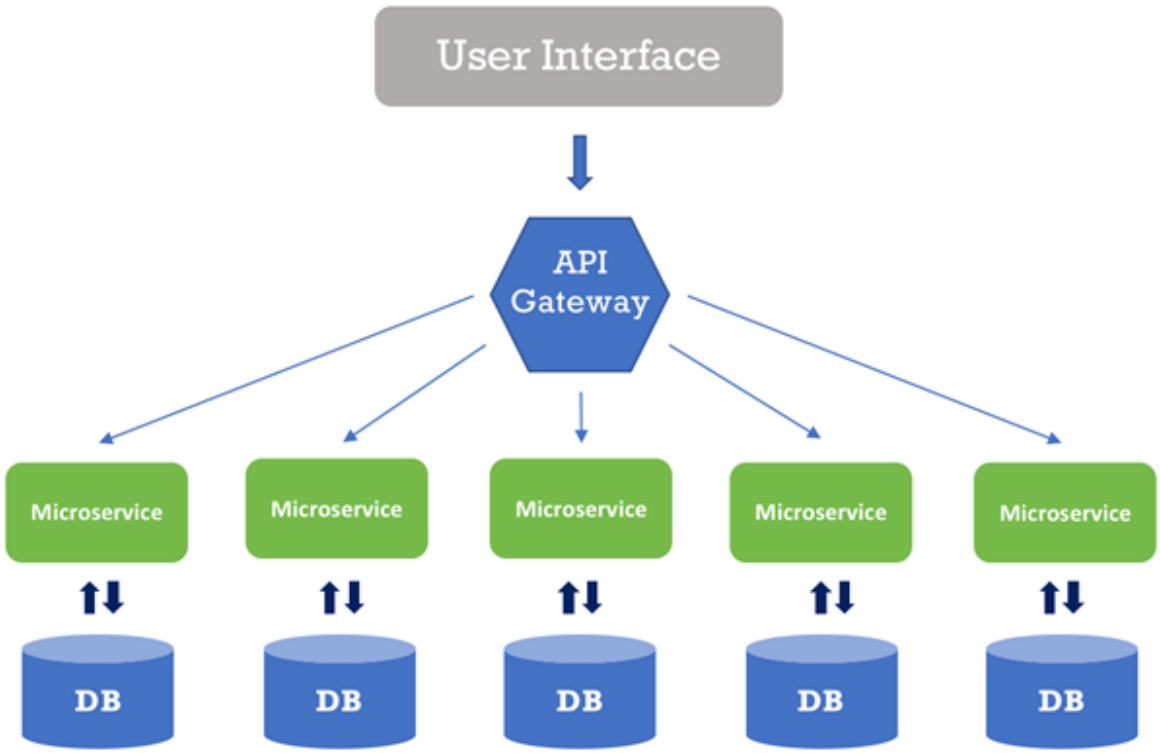
Svarbu paminėti, jog kelių puslapių tinklapiai gali taip pat būti paverčiami į PWĄ, tačiau tai atlikti su VPA yra paprasčiau ir efektyviau, kadangi VPA:

1. Visuomet natūraliai jau turės integruotą vartotojo sąsajos gilųjį maršrutizatorių, kurį dažnais atvejais, kelių puslapių aplikacijose reiktų kurti nuo nulio. Pavyzdžiui, parduotuvės tinklapis, kuris turi dinamiškai generuojamus prekių URL (kas yra dažna praktika dėl geresnės paieškos sistemų optimizacijos) ir dėl to nėra įmanoma iš anksto žinoti visų egzistuojančių tinklapių ir išsaugoti tų tinklapių turinį naršyklės atmintyje, neaplankius visų prekių.
2. Kodo struktūra ir naudojamos technologijos padės sukurti kokybiškesnį ir suprantamesnį vartotojui PWA per mažesnį laiko tarpą ir parašius mažiau kodo eilučių. Dėl to, nes kelių puslapių aplikacijos paprastai būna statinės su jau sugeneruotu turiniu ir parašytas programinis kodas natūraliai nebūna pritaikytas atlikti dinamiškiems pakitimams priklausomai nuo būsenos - šiuo atveju, nuo būsenos ar įrenginys yra prisijungęs prie interneto ar ne.

Taigi, kitas didelis JavaScript kalbos privalumas yra VPA ir PWA aplikacijos. Kadangi VPA turi daugybę privalumų palyginus su kelių puslapių aplikacijomis ir jos gali būti kuriamos tik su JavaScript kalba. Nors ir PWA gali būti kuriamos ne tik su VPA, tačiau kuriant PWA su VPA reikia parašyti mažiau kodo ir galima lengviau sukurti geresnę vartotojo patirtį, todėl tai taip pat yra JavaScript kalbos sukuriama nauda.

3 Mikroservisai ir Node.js

Mikroservisai yra architektūrinis aplikacijos stilius, kai aplikacija yra išskirstoma į daug mažų, funkcionalių dalių, kurios gali veikti individualiai, nepriklausomai nuo jokio kito programinio kodo, palyginimui su tradiciniu, monolitiniu modeliu, kai visos aplikacijos dalys yra kuriamos kaip vienas bendras vienetas. Mikroservisų veikimo modelį galima pamatyti 3 paveiksliuke.



Paveiksliukas 3. [Kum19]. Mikroservisų veikimo principas.

Mikroservisų architektūrinis stilius turi daug privalumų, tokių kaip:

1. Lengva prižiūrėti**.** Dėl savo fundamentalaus paprastumo, jog mikroserviso esmė yra atlikti vieną specifinę rolę ir nieko daugiau, yra žymiai paprasčiau suvokti ir įvertinti kaip veikia mikroservisas ir kokie pakeitimai turi būti atlikti, norint pasiekti norimą rezultatą. Taip pat yra žymiai lengviau įvertinti atliekamų programinių pakeitimų galimas pasekmes ir jeigu jų būtų - jas išspręsti, kadangi jie įtakotų tik to mikroserviso sritį.
2. Paprasta testuoti**.** Kadangi mikroservisas yra nepriklausomas nuo kitų aplikacijos dalių, tai reiškia, jog jį galimą testuoti kaip pavienį komponentą. Tokiu atveju, jeigu sistema yra didelė ir sudėtinga, užtenka įjungti tik vieną mažą mikroservisą, o ne visą sistemą.
3. Paprastas ir nepriklausomas dislokavimas produkcijai. Tai reiškia, jog dalis aplikacijos, atsakingą už specifinę rolę gali būti atnaujinama nepaliečiant kitų aplikacijos dalių.
4. Autonomiškumas. Mikroservisų kūrimas gali būti paskiriamas atskiroms, autonominėms komandoms. Tai suteikia programuotojams daugiau savarankiškumo, kas leidžia atlikti techninius sprendimus greičiau, mažesnėse

grupėse. Tai yra didelis privalumas, jeigu kuriama aplikacija bus labai didelė ar su ja dirbs daug žmonių.

5. Plečiamumas. Kadangi mikroservisai yra pavieniai, tai leidžia reikiamus sistemos komponentus, pavyzdžiui tuos, kurie susiduria su didesne apkrova ar naudoja daugiau resursų, plėsti nepriklausomai nuo kitų sistemos dalių, o tai sutaupo daug resursų - tiek įrangos, tiek laiko atžvilgiu.

Stackoverflow.com 2019m. statistiką [Sta19] puikiai atspindi mikroservisų architektūrinio stiliaus populiarumo kilimą. Docker ir Kubernetes - dvi su mikroservisais labiausiai susijusios technologijos - buvo įvertintos kaip labiausiai patinkančios platformos apklaustiesiems po Linux. Docker taip pat buvo 1-oje vietoje, o Kubernetes 4-oje vietoje kaip labiausiai norimos technologijos.

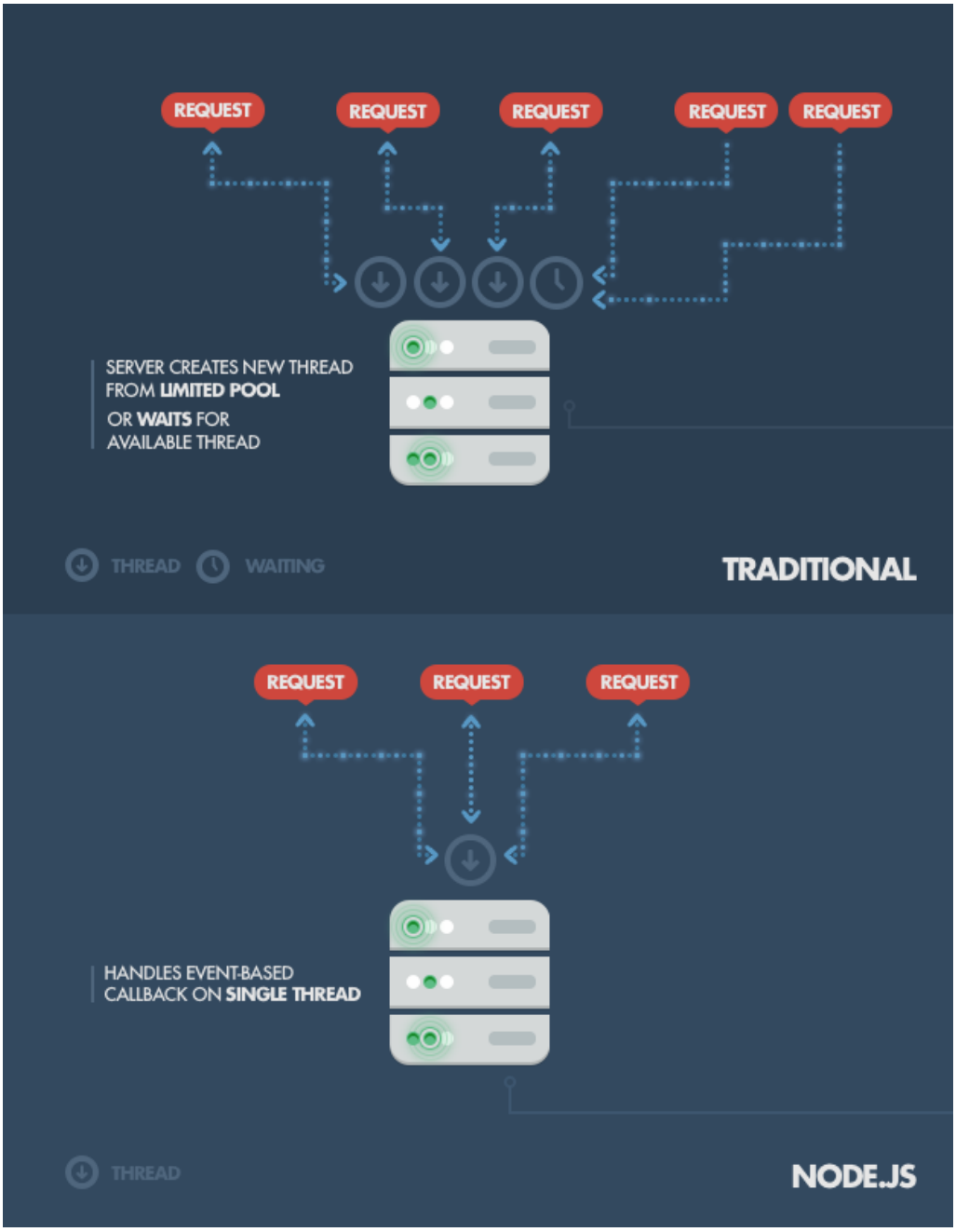
Tačiau, kaip mikroservisai susiję su Node.js? Node.js, dėl savo architektūros ir veikimo modelio yra viena iš dažniausiai pasirenkamų kalbų mikroservisų architektūros aplikacijoms kurti. Oficialus Node.js technologijos išsakytas tikslas, dėl ko ji buvo suprojektuota yra: “Node.js yra suprojektuotas kurti besiplečiančias internetines aplikacijas.” (angl. “is designed to build scalable network applications”) [Nod20]. Tai yra atliekama pasitelkiant šias technologijas:

1. Įvykiais varoma architektūrą.
2. Neblokuojantis įvesties/išvesties (angl. non-blocking input/output) modelis.
3. Asinchroniškas programavimas.

Šie visi technologiniai niuansai kombinacijoje leidžia efektyviai kurti mikroservisus, nes:

1. Node.js programa gali apdoroti užklausas neblokuojant pagrindinės darbinės gijos ir iš jos vienos išgauti didelį efektyvumą, kai reikia apdoroti daug paralelinių operacijų vienu metu. (Tai bus įrodoma eksperimento metu.) Pagrindinis skirtumas, kaip Node.js užklausų apdorojimas skiriasi nuo kitų populiarių standartinių kalbų kaip PHP, Java, Python yra tas, jog visos šios kalbos kiekvienai naujai užklausai sukuria naują giją, su kuria bus apdorojama užklausa, o Node.js naudoja tik vieną giją, kuri yra valdoma vidinio įvykių ciklo algoritmo (angl. event loop) (paveiksliukas 4). Efektyvumas apdorojant daug paralelinių operacijų šiame modelyje palyginus su gijų modeliu atsiranda iš to, jog kiekvienai gijai sukurti, turi būti rezervuojami resursai, kurie didžiąją laiko

dalį praleidžia tiesiog laukdami atsakymo iš duomenų bazės / HTTP užklausos / failo nuskaitymo, o ne iš tikrųjų naudodami jiems paskirtus resursus. Taip pat ir pačios gijos sukūrimo operacijos, nors ir nėra labai brangios, tačiau turi būti atliekamos papildomai, palyginus su Node.js. Žinoma, bėgant laikui pamačius šio modelio privalumus, atsiranda bandymų implementuoti toki patį modelį į kitas kalbas, kaip pavyzdžiui, python kalbos WEB aplikacijų karkasinė sistema “Tornado Web Server”, tačiau tuomet iškarto yra susiduriama su architektūrine problema - jog visas kodas tokiam WEB serveriui turi būti rašomas tik asinchroninis, norint neužblokuoti viso WEB serverio darbo, tačiau kadangi Python kalba nėra taip gerai architektūriškai pritaikyta dirbti su asinchroninėmis operacijomis kaip Node.js, dėl to dažnai prireikia naudoti papildomas pagalbines bibliotekas, funkcijas ir rašyti papildomą apkraunantį kodą, kuris atliktų norimą darbą.



Paveiksliukas 4. [Kap19] Tradicinis užklausų apdorojimo modelis viršuje, Node.js užklausų apdorojimo modelis apačioje.

2. Taip pat mikroserviso tipo kodą yra paprasčiau parašyti nei tokį patį mikroservisą kita populiaria kalba kaip pavyzdžiui PHP, Java ar Python ar panašiai. Tai natūraliai susiklosto dėl to, kadangi mikroservisams reikia atlikti daugiau:

a. Komunikacinio tipo darbo, kurį padeda išspręsti egzistuojanti įvykiais varoma architektūra, nenaudojant jokių papildomų bibliotekų, o tiesiog pasitelkiant pačios kalbos funkcionalumą.
b. Asinchroninio darbo, susidariusio dėl didelio kiekio komunikacijos, kurį padeda išspręsti natūraliai Node.js kalboje egzistuojantis asinchroninis programavimas ir neblokuojantis išvesties/įvesties modelis. Pavyzdžiui, Node.js automatiškai iškviečia nematomus “vaikinius” procesus, kurie rūpinasi tokiomis operacijomis kaip failų nuskaitymas/rašymas, tinklo/duomenų bazių operacijų apdorojimas. Šie procesai programuotojui paprastai būna nematomi ir atliekami už jį, kas smarkiai padidina kodo rašymo greitį ir programavimo patogumą. Taip pat, pati kalba duoda pasirinkimą programuotojui, ar jis nori operaciją atlikti sinchroniškai ar asinchroniškai, pridedant vieną papildomą žodį prieš operaciją “await” arba pridedant iškviečiamos funkcijos gale žodį “Async” (jeigu tai yra gimtoji Node.js biblioteka). Pavyzdžiui gimtosios “fs” bibliotekos, skirtos failų skaitymui ir rašymui, funkcija “fs.readFile” arba “fs.readFileAsync” ir panašiai.

Taigi, apibendrinant, JavaScript kalba, specifiškai Node.js technologija, turi daug privalumų kuriant mikroservisus, palyginus su standartinėmis populiariomis kalbomis. O tai yra svarbu, nes daugelis naujai kuriamų programų sistemų šiais laikais yra kuriami pasitelkiant mikroservisų architektūrą. Taip pat, gali išgauti didelį efektyvumą iš vienos darbinės gijos ir yra puikiai pritaikyta dirbti su asinchroninėmis operacijomis.

4 Aplikacijos programinio kodo rašymo efektyvumas ir aplikacijos efektyvumas

JavaScript padidina aplikacijų greitį ir efektyvumą. Puikus pavyzdys yra gerai žinomos internetinių mokėjimų įmonės PayPal loginės sistemos dalies parašytos su JAVA kalba ir Spring karkasine sistema migracija į Node.js. Visą savo migracijos procesą nuo vienos programavimo kalbos iki kitos jie aprašė, išanalizavo ir išleido ataskaitą [Pay13] apie tai, kaip ši migracija padėjo jiems pakelti sistemos efektyvumą ir smarkiai sumažino parašyto programinio kodo kiekį, analogiškai perkurtai sistemai. Pasak jų ataskaitos:

1. Aplikacija buvo perkurta dvigubai greičiau, lyginant su senąją JAVA kalba parašyta sistemine dalim ir tai buvo atlikta su mažiau žmonių komandoje.
2. Po migracijos, serveris galėjo priimti dvigubai daugiau užklausų per sekundę ir vidutinis atsakymo laikas buvo sumažintas 35%.
3. Buvo parašyta 33% mažiau kodo ir sukurta 40% mažiau failų, nei lyginant su senąją sistema.

Kita, ne tiek daug kam žinoma sėkmės istorija yra LinkedIn kompanijos loginės serverio dalies migracija nuo Ruby on Rails į Node.js. Po šios migracijos, jų įvertinti privalumai buvo [Lan12], [Hof12]:

1. Žymiai geresnis serverio efektyvumas ir mažesnis atminties sunaudojimas, tam tikrais atvejais net iki 20 kartų.
2. Vartotojo sąsajos programuotojai, su jau turimais JavaScript įgūdžiais galėjo prisidėti prie loginės dalies kūrimo ir dėl to skirtingos komandos galėjo būti sujungiamos į vieną.
3. Efektyvumas visumoje pakilo taip smarkiai, jog užteko tik 3 serverių vietoj anksčiau naudotų 30.
4. Programuotojai galėjo daugiau laiko skirti kurdami projektą, vietoj to, jog taisytų klaidas ar lipdytų kodo “pleistrus”.

4.1 Programinio kodo pernaudojimas

Viena iš pagrindinių vietų, iš kur atsiranda kodo efektyvumas, programuojant JavaScript, yra galimybė pernaudoti tuos pačius kodo gabalus tiek loginėje sistemos dalyje, tiek vartotojo sąsajos dalyje. Dažniausiai, tai yra daroma su įrankių ar pagalbinėmis funkcijomis. Tai darant, gali kilti tik vienas pavojus, jog kodas, kuris turėtų būti paslėptas ir nematomas paprastiems vartotojams, gali būti atskleidžiamas į viešumą. Tačiau, nuo to yra labai paprastai apsisaugoma - bendrai naudojamas kodas yra iškeliamas į atskirus failus, atskirai nuo loginių, serverio failų. Tokiu būdu, jokia delikati informacija ar kodas niekad nebus įtraukiamas į klientui užkraunamus failus. Tokio tipo kodo pernaudojimas yra aktualus WEB aplikacijose, kadangi JavaScript kalba jose turės būti naudojama bet kuriuo atveju, arba mobiliosiose ir darbalaukio aplikacijose, tuomet kai aplikacija yra kuriama su moderniais vartotojo sąsajos kūrimo karkasais kaip ReactNative ar Electron. Todėl labai dažnai, kažkurios vienos technologijos naudojimas kuriant kažkurią aplikacijos dalį - vartotojo sąsają ar

sisteminę - motyvuoja kitą dalį taip pat kurti su JavaScript, kadangi jos tarpusavyje tuomet turi puikią sinergiją. O JavaScript yra dažniausiai pasirenkama bent vienai aplikacijos daliai kurti, nes ji turi labai didelį technologinį palaikymą programuotojų bendruomenėje.

Kitas faktorius, kurį svarbu paminėti apie kodo pernaudojimą, yra galimybė pernaudoti tuos pačius, apsibrėžtus TypeScript kalbos statinius tipus tarp serverinės dalies ir kliento sąsajos dalies kodo. Nors TypeScript kalba nėra visiškai JavaScript, o tai daugiau yra JavaScript kalbos papildymas pridėtiniu funkcionalumu, tačiau tai vis tiek svarbu paminėti dėl kelių priežasčių:

1. TypeScript kuo toliau, tuo labiau tampa populiaresne [Sta18], [Sta19] ir daug JavaScript kalbos programuotojų pradeda migruoti prie TypeScript kalbos arba alternatyviai naudoja TypeScript, priklausomai nuo projekto.
2. Node.js ir TypeScript technologinis rinkinys yra vienintelis būdas, kaip galima programuoti pilnas aplikacijas, turint galimybę pernaudoti statinius tipus kuriant pilnas WEB ar mobilias aplikacijas. Išimtis šiam teiginiui, yra tik Flutter technologija, tačiau ji turi kitų minusų paminėtų anksčiau. Taigi, pagal didžiulių kompanijų raportuotas sėkmės istorijas galima teigti, kad aplikacijų kūrimas šia kalba padidina kodo rašymo ir pačios aplikacijos efektyvumą. Vieni iš prisidedančių faktorių kodo rašymo efektyvumui pakelti yra galimybė tarp kliento ir serverio pernaudoti programinį kodą ir TypeScript kalbos tipus, kas taip pat pageriną kodo palaikymą, atnaujinimą ir suvokimą.

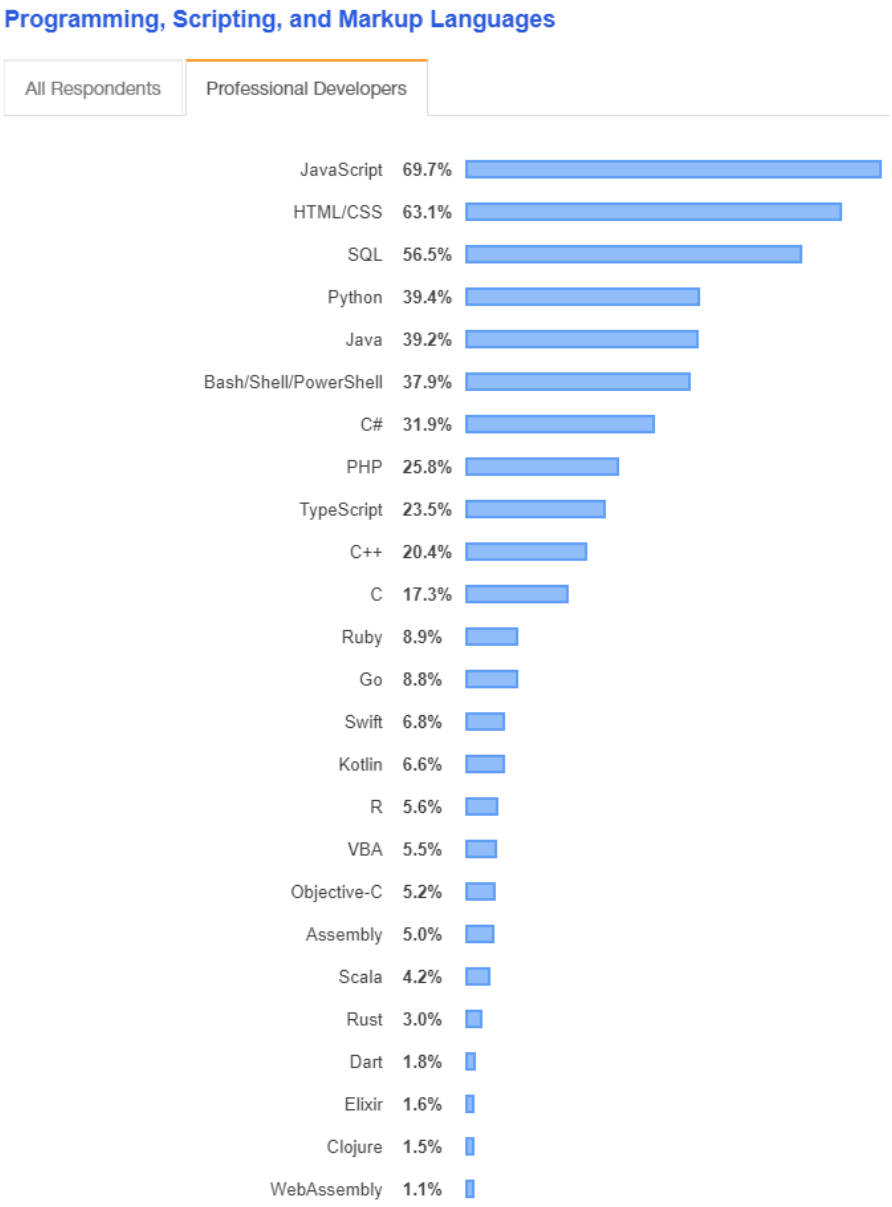
5 Technologinis populiarumas

Vertinant technologijos potencialą yra svarbu įvertinti jos populiarumą skirtingais aspektais. Nes pati technologiją, gali būti gera, tačiau jeigu nėra šia kalba dirbančių profesionalų, nėra sukurta jokių įrankių, palengvinančių ir pagreitinančių programavimo darbą ar žmonių prižiūrinčių šiuos įrankius ir taisančių jų klaidas, tuomet iš tokios kalbos ne daug praktinės naudos, didžiajai daliai naudojimo scenarijų.

5.1 JavaScript kalbos profesionalų populiarumas

Kadangi, jeigu tokių specialistų yra labai mažai, tuomet aplikacijos rašymas, palaikymas, ateities klaidų taisymas ir atnaujinimų pridėjimas gali tapti labai didele

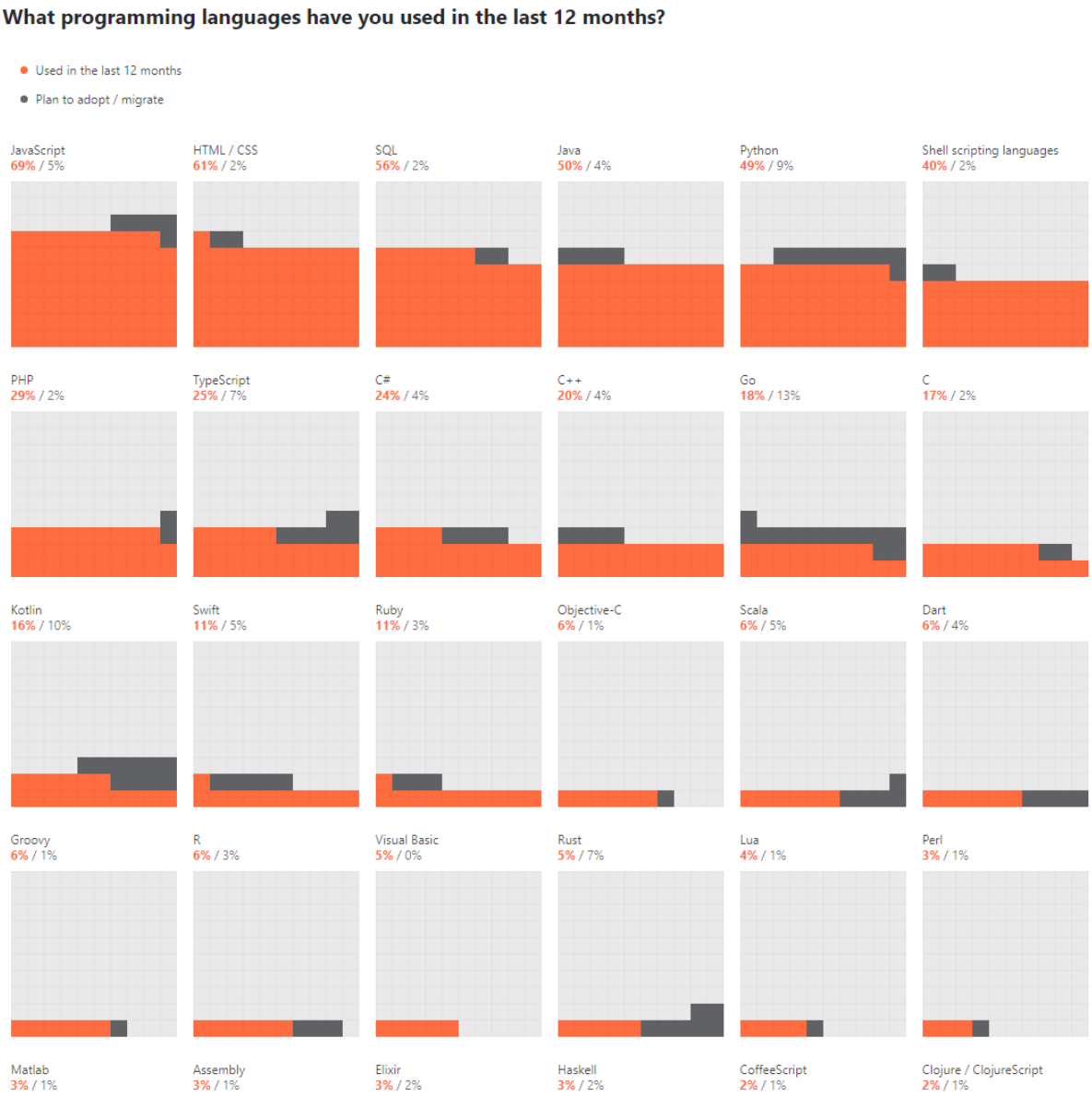
našta aplikacijos savininkams - paprastam žmogui ar verslui, kadangi surasti profesionalą, kuris tai galėtų atlikti, gali užtrukti labai ilgai ir/arba labai brangiai.



Paveiksliukas 5. [Sta19] Populiariausios programavimo, scenarijavimo ir žymėjimo kalbos

Didžiausio klausimų ir atsakymų tinklapio stackoverflow.com, skirto programuotojams, duomenimis (paveiksliukas 5) JavaScript kalba yra pati populiariausia tarp programuotojų ir yra labiausiai naudojama, jau 7 metus iš eilės. Kitos po jos einančios kalbos būtų, HTML/CSS, SQL, Python ir Java. Taip pat gan svarbu paminėti ir kalbą TypeScript, kuri yra 9 vietoje, kadangi TypeScript kalba, yra tik patobulintas JavaScript kalbos variantas, sukurtas Microsoft'o, kuris JavaScript kalbai prideda statinius duomenų tipus, kurie yra tikrinami kompiliavimo metu.

Labai panašius rezultatus rodo ir gerai žinomos įmonės, kuriančios profesionalią programinę įrangą programuotojams, JetBrains statistika.

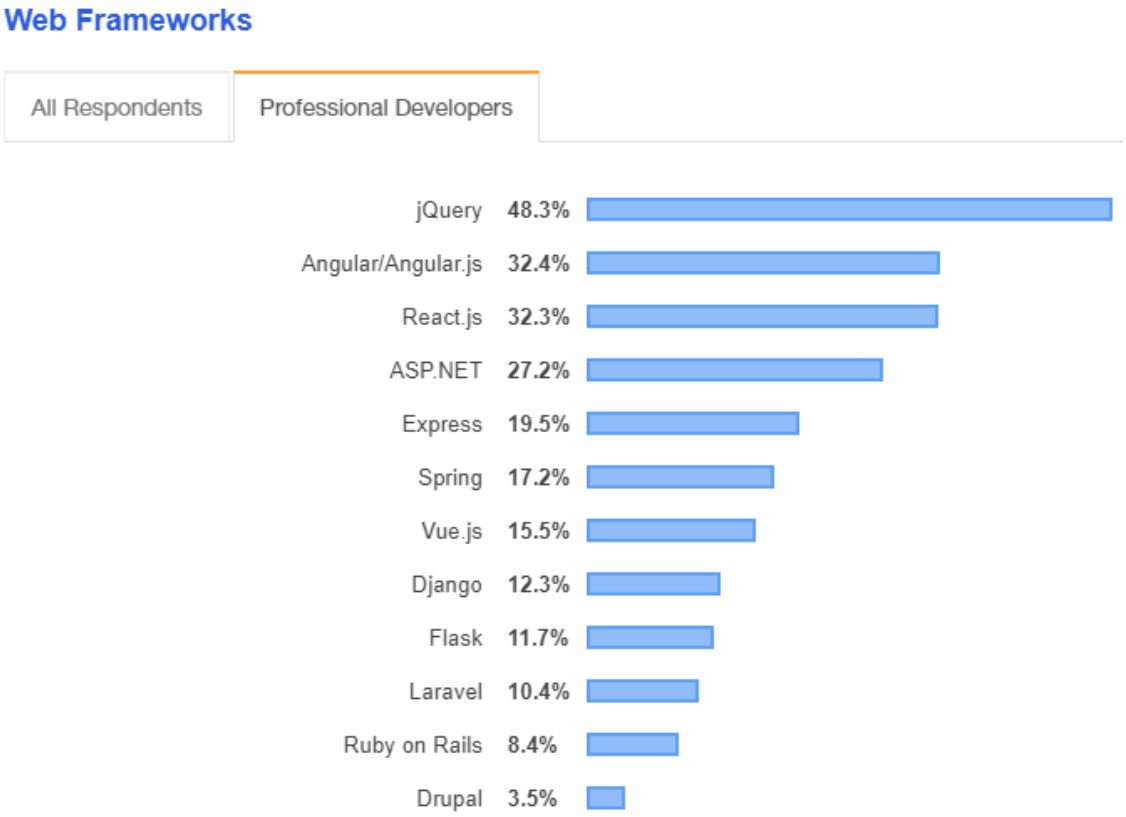


Paveiksliukas 6. [Jet19] Programuotojų labiausiai naudojamos programavimo kalbos per 2019 metus

Pagal šią statistiką (paveiksliukas 6) JavaScript kalba išlieka pirmoje vietoje, TypeScript aštuntoje. Kitos populiariausios kalbos išlieka tos pačios – HTML/CSS, SQL, Java, Python.

5.2 Populiariausios programavimo kalbų karkasinės sistemos ir kiti įrankiai

Taip pat svarbu peržvelgti programuotojų naudojamas pačias populiariausias karkasines sistemas, kurios padeda žymiai greičiau atlikti kažkokias technines užduotis - sukurti HTTP serverį, kurti vartotojo sąsajos šablonus, apjungti sisteminę dalį su vartotojo sąsaja ir t.t. Kadangi šios užduotys yra tipinės ir yra reikalingos beveik kiekvienai standartinei aplikacijai, šių karkasinių sistemų egzistavimas ir populiarumas tai specifinei programavimo kalbai puikiai parodo, ar ši kalba yra praktiška ir ar yra naudojama kitų programuotojų kuriant pilnas aplikacijas.



Paveiksliukas 7. [Sta19] Populiariausios karkasinės sistemos.

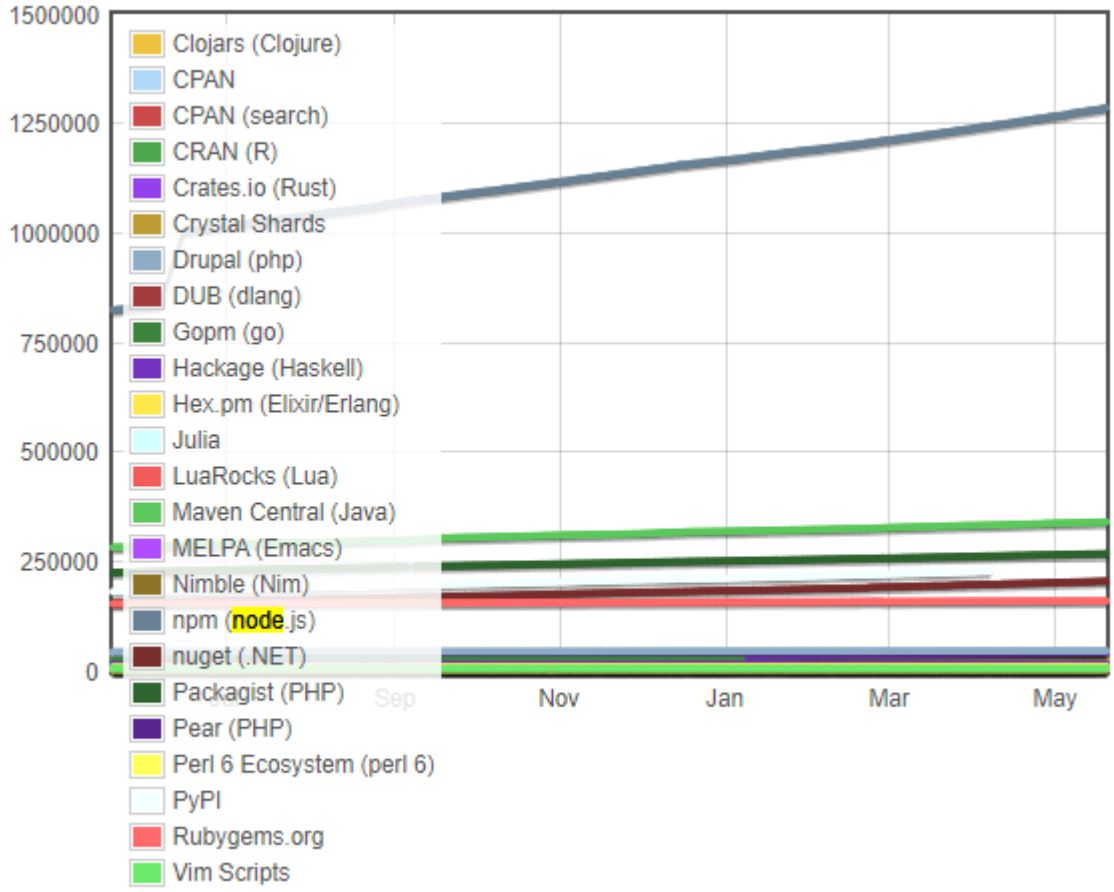
Iš pateiktų statistikų (paveiksliukas 7) galima matyti, jog tarp naudojamų karkasinių sistemų, pirmauja JavaScript kalbos sistemos. Iš pirmų penkių labiausiai naudojamų karkasinių sistemų kuriant tinklapius, net keturios - jQuery, Angular.js, React.js, Express - yra skirtos specifiškai JavaScript kalbai. Svarbu paminėti, jog visos šios JavaScript technologijos yra pritaikomos tiek WEB, tiek mobiliųjų telefonų, tiek kompiuterinių aplikacijų kūrimui, tokių technologijų pagalba kaip NativeScript ar ReactNative.

Taip pat, pažvelgus į tinklapio GitHub.com visų populiariausių viešų kodo talpyklų (angl. repositories) statistikas [Git20], net 7 iš 13 jų yra JavaScript kalbos. (Neskaičiuojant kalbai neutralių talpyklų, kurios nėra skirtos jokiai specifiniai kalbai ir įtraukiant TypeScript kalbos talpyklas)

Tinklapio modulecounts.com duomenys (paveiksliukas 8 ir paveiksliukas 9), kurie seka 24 populiariausių programavimo kalbų įrankių platformas, taip pat parodo jog Node.js įrankių valdymo platforma turi didžiausią kiekį viešų pagalbinių įrankių, ir lenkia kitą populiariausią kalbą - Java (Maven Central) - net ~3.78, o PHP (Pear, Packagist, Drupal) net ~4.1 karto savo egzistuojančių įrankių kiekiu. Vidutinis įrankių atsiradimo kiekis per dieną, JavaScript kalbos yra ~5.6 karto didesnis palyginus su Java kalba ir ~8.23 palyginus su PHP kalba. Tai ne tik parodo jos dabartinį populiarumą, tačiau ir tai jog numatomas tolimesnis JavaScript kalbos populiarėjimas ir augimas.

	May 13	May 14	May 15	May 16	May 17	May 18	May 19	Avg Growth
Clojars (Clojure)	26241	26247	26252	26261	26262	26264	26271	5/day
CPAN	41330	41336	41339	41340	41351	41354	41356	5/day
CPAN (search)								5/day
CRAN (R)	15677	15677	15677	15690	15682	15683	15683	1/day
Crates.io (Rust)	40094	40132	40186	40242	40290	40349	40394	50/day
Crystal Shards	5504	5506	5510	5513	5521	5524	5528	4/day
Drupal (php)	45400	45401	45408	45412	45419	45423	45427	6/day
DUB (dlang)	1795	1796	1798					2/day
Gopm (go)								7/day
Hackage (Haskell)	14844	14844	14848	14851	14857	14865	14868	4/day
Hex.pm (Elixir/Erlang)	9972	9979	9982	9987	9991	9997	10006	6/day
Julia	3190	3194	3202	3203	3203	3203	3203	3/day
LuaRocks (Lua)	2461	2462	2462	2463	2464	2464	2467	1/day
Maven Central (Java)	338674	338886	339029	339171	339281	339583	339730	188/day
MELPA (Emacs)	4590	4590	4590	4590	4593	4596	4596	1/day
Nimble (Nim)	1282	1283	1283	1283	1286	1289	1289	1/day
npm (node.js)	1279525	1280477	1281253	1281984	1282747	1284743	1285873	1054/day
nuget (.NET)	203405	203405	204109	204109	204273	204409	204667	209/day
Packagist (PHP)	266533	266680	266836	266896	266993	267132	267265	122/day
Pear (PHP)	603	603	603	603	603	603	603	0/day
Perl 6 Ecosystem (perl 6)	1548	1549	1549	1556	1556	1557	1557	1/day
PyPI								177/day
Rubygems.org	159446	159478	159505	159526	159538	159552	159576	22/day
Vim Scripts	5536	5536	5536	5536	5536	5536	5536	0/day

Paveiksliukas 8. [Mod20] Populiariausių įvairių programavimo kalbų įrankių talpyklų statistika, 2020m.



Paveiksliukas 9. [Mod20] Populiariausių įvairių programavimo kalbų įrankių talpyklų statistika, 2020m.

Apibendrinant, galima teigti, kad JavaScript kalba pagal populiarumo kriterijų puikiai tinka kurti pilnoms aplikacijoms, nes:

- Egzistuoja daug JavaScript kalbos profesionalų.
- Iš visų programuotojų naudojamų karkasinių sistemų ir kitų įrankių, JavaScript kalbos įrankiai yra patys populiariausi.
- Pačios populiariausios viešos pasaulio kodo talpyklos yra parašytos JavaScript kalba.
- JavaScript kalbos įrankių ir įskiepių egzistuojantis kiekis ir jų vidutinis dienos augimas parodo, jog ši kalba šiuo metu yra pati populiariausia ir yra toliau populiarėjanti greičiausiu tempu.

6 JavaScript kalbos specifiniai privalumai kuriant pilnas aplikacijas

Pilnų aplikacijų kūrimas naudojant JavaScript turi keletą specifinių privalumų, kurių neturi kitos kalbos:

1. Galimybė pasitelkti JavaScript kalbos populiariąją technologiją Webpack.
2. Galimybė kurti vartotojo sąsajas su populiariausiomis, moderniomis karkasinėmis sistemomis.

6.1 Webpack technologija

Webpack yra Node.js statinė modulių surišimo technologija, modernioms JavaScript aplikacijoms. Šios technologijos dėka JavaScript vartotojų sąsajos kodą galima rašyti nauju standartu - moduliais. Ši technologija, apdoroja visą įeities JavaScript kodą, patikrina visus kode naudojamus modulius ir išorinius failus, iš jų sukuria priklausomybių grafiką (angl. dependency graph), kurio dėka failai ar moduliai, kurie dubliuojasi kode, gali būti optimizuojami ar kitaip pertvarkomi ir įtraukiami į sukompiliuoto kodo bazę tik vieną kartą visam projektui, apart to, jog būtų įtraukiami, kiekvieną kartą, kai jie yra kur nors panaudojami ar iškviečiami ir tuomet daug skirtingų failų yra sujungiama į vieną (ar pagal pasirinkimą - kelis, tačiau taip pat optimizuotus) failą/us. Kad suprasti, kodėl tai yra svarbu, reikia suteikti šiek tiek kontekstinės informacijos apie JavaScript. Standartiškai, kuriant vartotojo sąsaja senais standartais, JavaScript kalba visuomet yra naudojama globaliu rėžimu. Nėra jokių modulių, kurie būtų įtraukiami į kitus modulius ar atliekamos priklausomybių injekcijos (angl. dependency injection). Norint panaudoti kažkokį tai modulį, jis tiesiog yra iškviečiamas globaliame kontekste ir tuomet tampa prieinamas absoliučiai visam tinklapyje naudojamam JavaScript kodui. Suprantama, tai sukelia daugybę problemų, tokių kaip:

1. **Vardų srities užteršimas** (angl. namespace pollution). Rašant kodą vienoje kodo dalyje, visiškai atsitiktinai, programuotojui to nežinant, jis gali pakeisti kažkokį jau naudojamą, svarbų, globalų kintamąjį ar funkciją ir dėl to gali kilti sunkiai atsekamų klaidų, kurių problemos šaltinio gali tekti ieškoti labai ilgai.
2. **Sunkiai suprantamas kodas.** Kodas visuomet yra suprantamas geriausiai, kai yra aiškiai matoma iš kur ateina informacija ir kai individualių kodo elementų sritis yra apribota. Kadangi globalūs kintamieji gali būti modifikuojami bet kurioje

kodo dalyje, tai žymiai labiau apsunkina kodo suvokimo ir kodo loginės sekos sekimo procesą.

3. **Krenta kodo efektyvumas**. Lokalaus kodo naudojimas, tam tikrais scenarijais gali pakelti kodo efektyvumą JavaScript kalboje net iki 82% [Web08].

Svarbu paminėti, kad šiai problemai apeiti, rašant kodą pagal senus standartus, yra naudojami tokie kodo rašymo stiliai kaip YUI modulių modelis (angl. YUI module pattern.), kurie sumažina kiekį funkcijų ir kintamųjų kuriais yra užteršiama globali vardų sritis, tačiau tai neišsprendžia problemos iki galo. Tačiau, globalaus kodo problemos išsprendimas nėra vienintelis dalykas, ką padaro Webpack. Taip pat jis atlieka šias funkcijas:

1. **Programinio kodo optimizavimas**. Webpack automatiškai aptinka, neefektyviai parašyta kodą ir jį automatiškai suoptimizuoja naujai sukompiliuotoje kodo bazėje. Tuomet, rašant kodą, galima jį rašyti taip, jog jis būtų lengviau skaitomas ir suprantamas, o kodo efektyvumu, pasirūpina Webpack.
2. **Programinio kodo minimizavimas**. Webpack technologija taip pat maksimaliai suspaudžia ir suminimizuoja kodą, panaikindamas visus nereikalingus simbolius, tokius kaip: tarpus, kablelius, naujų eilučių simbolius, komentarus. Taip pat, kaip jau minėta anksčiau, minimizavimas yra atliekant aptinkant dažnai naudojamas kodo vietas ar modulius, juos iškeliant ir pernaudojant tiesiog vieną to kodo instanciją.
3. **Nenaudojamo kodo ištrynimas**. Webpack automatiškai aptinka nepanaudoto kodo dalis ir tos dalys nėra įtraukiamos į sukompiliuoto kodo bazę, dar labiau minimizuojant kodą.
4. **Programinio kodo užmaskavimas (angl. obfuscation)**. Užmaskavimas dažniausiai yra naudojamas dėl dviejų priežasčių -
    a. Jį užmaskavus kodas pasidaro mažesnis, kadangi, visi kintamieji ir funkcijų pavadinimai yra pervadinami į kažką labai trumpo, pavyzdžiui, vieną raidę.
    b. Užmaskuotą kodą yra žymiai sunkiau skaityti, suprasti ir analizuoti potencialiems įsilaužėliams ar kitiems kenkėjams, kurie gali norėti perskaityti ir suprasti kodą, dėl jų siekiamų piktų kėslų.

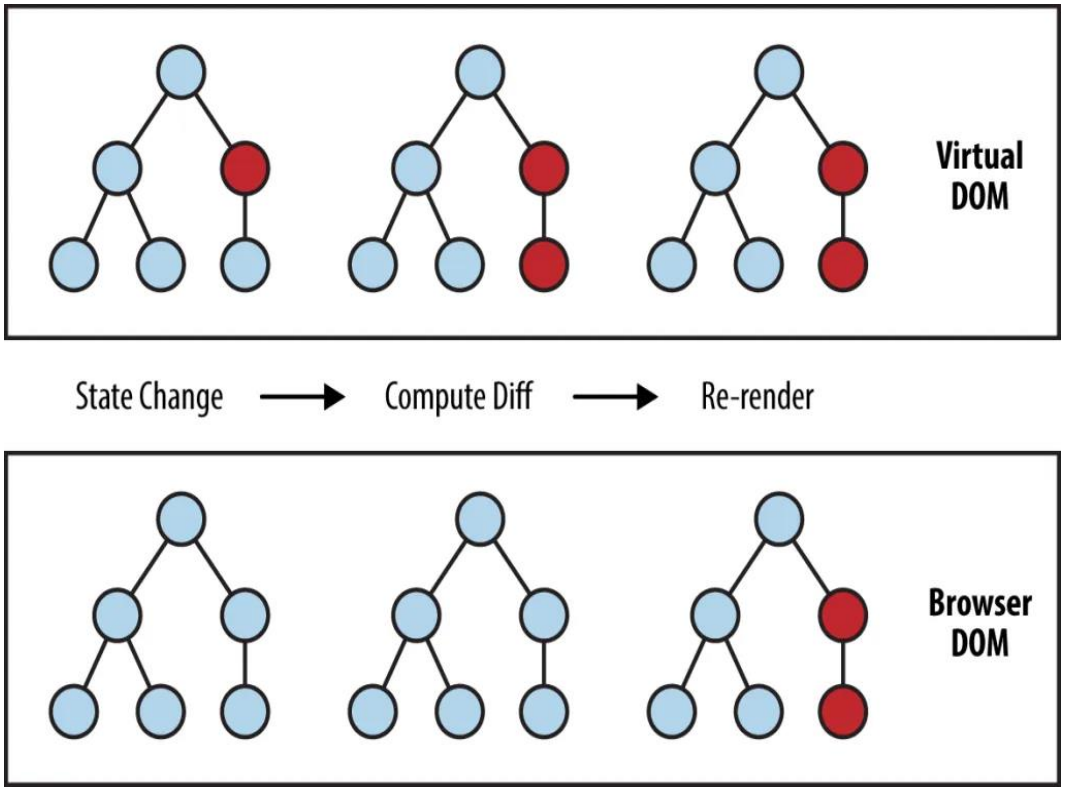
5. **Kodo išskaidymas į kelis ar vieną failus**. Priklausomai nuo projekto tipo ir jo dydžio, kodas dažnai išvedamas kaip vienas arba keli failai. Vienas iš pavyzdžių, kuomet verta skaidyti kodą į du failus, yra tuomet, kai didelę kodo dalį sudaro išoriniai naudojami moduliai. Tokiu atveju yra sukuriamas atskiras failas, kuris turi visus išorinius modulius, ir kitas failas, kuris turi visą parašytą kodo bazę. Tuomet pasinaudojant naršyklių teikiamu talpyklų funkcionalumu (angl. caching), galima smarkiai pagreitinti tinklapio užkrovimo greitį, ar atsisiunčiamų atnaujinimų dydį (mobiliosiose ar kompiuterinėse programose), kadangi įvykus kažkokiems kodo pakeitimams, tuomet reikia atnaujinti tik sukurto programinio kodo failą, kuris paprastai yra žymiai mažesnis, už naudojamų išorinių modulių failą.
6. **Automatinis kodo užkrovimas įvykus pasikeitimams (angl. live reloading)**. Šis funkcionalumas dažniausiai yra naudojamas tik kuriant aplikaciją. Jis smarkiai padidina kuriamos aplikacijos kodo rašymo efektyvumą, kadangi tik išsaugojus programinį kodą, Webpack automatiškai aptinka skirtumus, ir būtent tik tuos specifinius skirtumus “įšvirkščia” (angl. inject) į aplikaciją ir tuomet ji neturi būti papildomai iš naujo perkraunama ar perkompiliuojama norint pamatyti pakeitimus.
7. **Galimybė naudoti naujų standartų JavaScript sintaksę, kuri dar nėra palaikoma visuose įrenginiuose**. Webpack dėka, galima rašyti kodą su modernia JavaScript sintakse, tačiau kompiliacijos metu, naujoji sintaksė sukompiliuotoje kodo bazėje yra paverčiama į senąją, pažystamą visoms naršyklėms ar kitiems įrenginiams ir taip yra išvengiamos nesuderinamumo problemos.

6.2 Vartotojo sąsajos kūrimas su moderniomis karkasinėmis technologijomis.

Kitas privalumas - JavaScript kalba parašytos vartotojo sąsajos kūrimo karkasinės sistemos tokios kaip React, React Native, Angular, Vue.js, NativeScript ir t.t. Visos šios karkasinės sistemos iš esmės turi tokius bendrus pranašumus, vietoj standartinių aplikacijų kūrimo tik su HTML/CSS kombinacija ir JavaScript kalba tik interaktyvumui kurti:

1. Galimybė kurti įvairių platformų aplikacijas su viena kalba. Tai reiškia, jog su viena kalba galima pilnai sukurti modernią, efektyvią aplikaciją skirtą visoms platformoms ir aplinkoms - naršyklėse, skirtingoms operacinėms sistemoms mobiliuosiuose įrenginiuose ir kompiuteriuose.
2. Jei kodas nėra rašomas su technologija, kuri leistų tą patį kodą pritaikyti kitam įrenginiui ar operacinei sistemai, tuomet bent egzistuoja galimybe pernaudoti programinį kodą kuriant šias aplikacijas skirtingoms platformoms. Pavyzdžiui, ReactNative karkasinėje sistemoje parašytas programinis kodas, gali būti puikiai pernaudojamas React karkasinėje sistemoje, pasitelkus tam tikrus modulius, kaip “React Native for Web” [Git20a] arba rankiniu būdų pakeitus pagrindinius atvaizdavimo struktūrinius elementus į WEB sistemoje atitinkančius elementus, tik tiek. Visa loginė vartotojo sąsajos dalis, būsenų valdymo, komponentų gyvavimo ciklų kodo dalis nesiskiria ir gali būti pernaudojama.
3. Virtualaus dokumento objekto modelio (angl. virtual document object model - Virtual DOM.) optimizacijos pasitelkimas. Šios optimizacijos veikimo principas yra toks:
    a. Jog visuomet atmintyje yra laikomas virtualus dokumento objekto modelis.
    b. Tuomet, tik atlikus kokius nors pakeitimus, sukuriamas naujas virtualus dokumento objekto modelio medis.
    c. Naujasis pakeitimų medis yra palyginamas su senuoju ir tada yra surandamas pats efektyviausias būdas padaryti šiems pakeitimams.
    d. Suradus geriausiam būdui įgyvendinti šiuos pakeitimus, tie pakeitimai iš tikro yra pritaikomi realiam dokumento objekto modeliui, kuris yra atvaizduojamas vartotojui.

Visas šis procesas galėtų būti atvaizduojamas taip (paveiksliukas 10) :



Paveiksliukas 10. [Ham18] Virtualaus dokumento objekto modelio veikimo principas

Šis būdas, kaip galima suprasti, yra žymiai efektyvesnis, negu senasis, įprastas modelis - kuomet norint atlikti pakeitimus vienam ar keliems elementams, tie elementai yra ieškomi visame objekto dokumento modelyje. Juos suradus, tie pakeitimai yra tiesiogiai pritaikomi, neieškant jokių persidengiančių pakeitimų, kuriuos būtų galima sugrupuoti ir atlikti kaip vieną ar kitokių optimizacijų. Pakeitimai pritaikomi po vieną pakeitimą vienu metu, ko pasekoje, reikia atlikti kelis atnaujinimus. Dėl daugybės greitų atnaujinimų per mažą intervalą, dažnai vartotojui yra sukuriamas stringančios aplikacijos įvaizdis.

4. Galimybė pilnai kurti vartotojo sąsają su CSS/JS/HTML kalba, viename faile, naudojantis modernia “JSX” ar “TSX” (TypeScript) sintakse. Tai leidžia programuotojams kurti universalius, pernaudojamus, atominius (angl. atomic) komponentus, kai visas su vartotojo sąsaja susijęs kodas - stiliai, komponento būsenų valdymas ir pačios vartotojo sąsajos struktūrinis kodas - yra viename faile. Tokių komponentų priežiūra ir atnaujinimas yra žymiai paprastesnis. Taip pat, tokių komponentų kūrimas išsprendžia neatsekamo nenaudojamo kodo problema. Kadangi visos kodo dalys, kurios nėra naudojamos komponente yra iškarto aiškiai atsekamos, negu lyginant su standartiniu vartotojo sąsajos

modeliu, kai loginis komponentų kodas yra vienuose failuose, stilistinis kodas yra kituose failuose, ir pačio komponento struktūrinis kodas yra trečiajame faile.

Taigi, JavaScript kalbos Webpack technologiją kartu su galimybe kurti vartotojo sąsają moderniomis karkasinėmis technologijomis yra didelis privalumas, kadangi tai:

1. Leidžia pasitelkti Webpack technologiją, kuri padidina kodo rašymo efektyvumą, įgalina naudoti pačią naujoviškiausią JavaScript kalbos sintaksę, kuri galbūt paprastai nebūtų palaikoma klientų naršyklių ir automatiškai atlieka įvairias kodo optimizacijas kaip - minimizavimą, užmaskavimą, nereikalingo kodo pašalinimą, loginį optimizavimą.
2. Leidžia pasitelkti naujoviškiausias vartotojų sąsajos kūrimo karkasines sistemas, kurių dėka galima kurti pilnas aplikacijas įvairiems įrenginiams - kompiuteriams ir išmaniesiems telefonams - ir skirtingoms operacinėms sistemoms telefonams su viena kodo baze.
3. Palengvina programavimo darbą, kadangi virtualaus objekto modelio optimizacija užtikrina, kad nesvarbu kaip programuotojas parašytų dinamiškų pasikeitimų kodą, ši optimizacija užtikrins, jog vartotojui būtų atliktu pačiu efektyviausiu būdu.
4. Supatogina programinio kodo atnaujinimą ir palaikymą, nes suteikia galimybę rašyti gerai struktūrizuotą atominį vartotojo sąsajos kodą, kai visas susijęs vienos vartotojo sąsajos komponento kodas - loginis, struktūrinis ir stilistinis - yra viename faile.

2-asis ir 3-iasis paminėti punktai yra dideli JavaScript kalbos privalumai, kadangi 3-iasis punktas yra įmanomas tik su JavaScript kalba, o 2-asis turi tik vieną alternatyvą - Google sukurtą technologiją Flutter sukurtą su Dart kalba, tačiau ji neturi daugybės kitų privalumų kaip bendruomenės palaikymo, populiarumo, yra labai naują, todėl vis dar turi daug klaidų ir t.t. O 1-asis punktas, nėra didelis privalumas palyginus prieš kitas kalbas, nes kitos kalbos gali atlikti tai taip pat, tačiau JavaScript kalba šioje srityje niekuo nuo jų neatsilieka.

7 Programavimo kalbos efektyvumas

Programavimo kalba gali būti labai gera įvairiais aspektais - būti populiari, patogi programuoti, egzistuoja daug sukurtų įrankių dirbti su ta kalba, galima greitai kažką su ja sukurti ir t.t. Tačiau, gali būti tokių aplikacijų, kurioms reikia atlikti daug ir sudėtingų

skaičiavimų ir tai gali būti pagrindinis faktorius, kuris nulems, kokią programavimo kalbą pasirinkti tai aplikacijai kurti. Taip pat ir todėl, jog aplikacija gali populiarėti ateityje ir to pasekoje didės jos apkrova. Efektyvi programavimo kalba iš karto užtikrina, kad aplikacija galėtų atidėti programos plėtimosi (angl. scaling) problemos sprendimą gan ilgam laikui, kol programa pasieks ekstremaliai dideles apkrovas arba bent jau palengvinti aplikacijos plėtimąsi ateityje dėl savo techninių savybių. Taigi todėl, atliksime eksperimentus, palyginančius kalbų efektyvumą.

7.1 Eksperimento aprašymas

Taigi programavimo kalbos efektyvumui aplikacijų kūrimui ištirti atliksime keletą skirtingų testų, tarp kelių populiariausių programavimo kalbų, tokių kaip Python ir PHP. Šios kalbos buvo pasirinktos todėl, jog Python yra kita populiariausia programavimo kalba po JavaScript, o PHP dėl savo populiarumo tarp WEB aplikacijų, kurios sudaro didžiausią procentalią dalį visų aplikacijų. Visi testai atliekami su JavaScript kalba, bus atliekami Node.js aplinkoje. Bus atliekami skirtingi testai, su tikslu įvertinti efektyvumą skirtingomis sąlygomis, kurios būtų aktualios realiame pasaulyje, tikrose aplikacijose. Testai bus atliekami ant vieno kompiuterio. Su kiekviena kalba visi testai bus suprogramuoti vienodai, stengiantis išlaikyti vienodas naudojamas operacijas ir operacijų kiekį, maksimaliai iki kiek tai leis atlikti programinės kalbos funkcionalumas. Eksperimentui yra naudojamos pačios naujausios, stabilios (ištestuotos) programavimo kalbų versijos egzistuojančios eksperimento darymo metu. 1 Lentelėje yra pateikiama testavimo aplinkos techninė specifikacija.

CPU	7th Generation Intel® Core™ i7-7600k procesorius (4.0 GHz)
RAM (kompiuterio operatyvioji atmintis)	Unbuffered Corsair DDR4 16GB (2x8GB) 3000MHz CL15 1.35V
Atmintis	Crucial MX300 SSD - 525GB
Operacinė sistema	Windows 10
Node.js versija	12.14.0 LTS
Python versija	3.8.1
PHP versija	7.4.1

Lentelė 1. Testavimo aplinkos techninė specifikacija.

Atliekami eksperimentai:

1. Standartinių operacijų vykdymas.
2. HTTP užklausų apdorojimas.

7.2 Eksperimentas #1. Standartinių operacijų vykdymas.

Bus atliekami eksperimentai su tokiomis standartinėmis operacijomis:

1. Skaičių sudėjimas 100 mln. kartų.
2. Masyvo užpildymas su 5 mln. elementų.
3. 100 MB failo nuskaitymas 100 kartų.

Efektyvumo vertinimo metrika bus praėjęs sekundžių kiekis, nuo operacijos pradžios iki pabaigos.

Šio testo pseudo kodas atrodytų maždaug taip:

```
// skaičių sudėjimas 10mln. kartų
sum := 0
for(i = 1; i <= 100000000; i++) {
    sum += i
}

//masyvo užpildymas su 10mln. elementų.
array := [];
for(i = 1; i <= 100000000; i++) {
    array.push(i)
}

//100 MB failo nuskaitymas 100 kartų.
for(i = 1; i <= 100; i++) {
    nuskaityti_faila(100_MB_FAILAS)
}
```

Paveiksliukas 11. Eksperimento #1 pseudo kodas.

7.2.1 JavaScript (Node.js) eksperimentas #1

JavaScript (Node.js) eksperimento naudojamas programinis kodas yra aprašytas 1 priede. Eksperimentas buvo sėkmingas. Gautų rezultatų išvestį galima pamatyti 2 priede. Buvo gauti tokie rezultatai:

Operacija	**Laikas, sekundės**
100 mln. skaičių sumavimas	0.194
5 mln. elementų masyvo užpildymas	0.108
100MB failo nuskaitymas 100 kartų	4.068

Lentelė 2. JavaScript (Node.js) kalbos eksperimento #1 rezultatai.

7.2.2 Python kalbos eksperimentas #1

Python eksperimento naudojamas programinis kodas yra aprašytas 3 priede. Eksperimentas buvo sėkmingas. Gautų rezultatų išvestį galima pamatyti 4 priede. Buvo gauti tokie rezultatai:

Operacija	**Laikas, sekundės**
100 mln. skaičių sumavimas	12.276
5 mln. elementų masyvo užpildymas	0.523
100MB failo nuskaitymas 100 kartų	4.444

Lentelė 3. Python kalbos eksperimento #1 rezultatai.

7.2.3 PHP kalbos eksperimentas #1

PHP eksperimento naudojamas programinis kodas yra aprašytas 5 priede. Eksperimentas buvo sėkmingas. Gautų rezultatų išvestį galima pamatyti 6 priede. Buvo gauti tokie rezultatai:

Operacija	**Laikas, sekundės**
100 mln. skaičių sumavimas	1.182
5 mln. elementų masyvo užpildymas	0.219
100MB failo nuskaitymas 100 kartų	7.274

Lentelė 4. PHP kalbos eksperimento #1 rezultatai.

7.2.4 Eksperimento #1 rezultatų apibendrinimas

	JavaScript (Node.js)	**Python**	**PHP**
100mln. elementų sumavimas	194.902ms	12276.090ms	1182.358ms
5mln. elementų užpildymas į masyvą	108.342ms	523.018ms	219.442ms
100MB failo nuskaitymas 100 kartų	4068.803ms	4444.139ms	7274.437ms

Lentelė 5. Eksperimento #1 rezultatų apibendrinimas.

Iš eksperimento rezultatų (lentelė 5) galima padaryti išvadas, jog JavaScript kalba, atliekant šias tris operacijas - elementų sumavimą, elementų užpildymą į masyvą, failų nuskaitymą yra efektyviausia, kadangi visos šios operacijos užtruko mažiausiai laiko palyginus su kitomis programavimo kalbomis. Antra efektyviausia kalba būtų PHP, nes jos elementų sumavimo ir masyvo užpildymo operacijos yra efektyvesnės už Python, tačiau smarkiai atsilieka nuo Python nuskaitant failus. Ir paskutinėje vietoje pagal efektyvumą - Python kalba.

.

7.3 Eksperimentas #2. HTTP užklausų apdorojimas.

Šio eksperimento tikslas yra ištirti kaip efektyviai šios trys programavimo kalbos gali apdoroti HTTP užklausas naudojant įprastai naudojamus, populiariausius technologinius rinkinius su šiomis kalbomis. HTTP serveriai bus testuojami lokaliajame tinkle (localhost). Bus daromos HTTP GET užklausos. Visų užklausų atsakymas bus vienodas - HTTP 200 statuso atsakymas su tekstu “Hello World!”.

Testuojama bus su Apache Benchmark įrankiu, versija 2.3. Iš viso bus atliekama 100 tūkst. užklausų, vienu metu leidžiant atlikti 1000 paralelinių užklausų. Užklausos bus atliekamos su “KeepAlive” užklausos antrašte, kadangi standartiškai ši antraštė visuomet yra pritaikoma naršyklėse, o eksperimento tikslas yra sukurti kuo artimesnes sąlygas realiam naudojimo scenarijui. Ši antraštė užtikrina, jog tuo pačiu TCP prisijungimu prie serverio, būtų galima atlikti/gauti keletą HTTP užklausų/atsakymų ateityje ir nereikėtų kurti naujo TCP prisijungimo.

Testavimo komanda, naudojama su Apache BenchMark įrankiu: “ab.exe -k -c 1000 -n 100000 localhost:[port]/”

Testuojami technologijų rinkiniai bus:

1. Node.js + Express (versija 4.17.1) HTTP serveris.
2. PHP + Apache (versija 2.4.41) HTTP serveris.
3. Python + Flask (versija 1.1.1) HTTP serveris.

Vertinimo metrikos, pagal kurias bus vertinamas HTTP užklausų apdorojimas:

1. Kiek HTTP užklausų gali serveris apdoroti per sekundę
2. Kiek laiko vidutiniškai užtrunka atsakyti į HTTP užklausą.
3. Kiek užklausų nepavyko įvykdyti dėl per didelės apkrovos.

7.3.1 JavaScript (Node.js) kalbos eksperimentas #2

JavaScript (Node.js) eksperimento naudojamas programinis kodas yra aprašytas 7 priede. Eksperimentas buvo sėkmingas. Gautų rezultatų išvestį galima pamatyti 8 priede. Buvo gauti tokie rezultatai:

Metrika	**Rezultatas**
Užklausų sk. / sekundę	7153.37
Užklausa užtrunka vidutiniškai, sekundžių	0.139
Nepavykusios užklausos	0

Lentelė 6. JavaScript (Node.js) kalbos eksperimento #2 rezultatai.

7.3.2 Python kalbos eksperimentas #2

Python eksperimento naudojamas programinis kodas yra aprašytas 9 priede. Eksperimentas buvo sėkmingas. Gautų rezultatų išvestį galima pamatyti 10 priede. Buvo gauti tokie rezultatai:

Metrika	**Rezultatas**
Užklausų sk. / sekundę	295.93
Užklausa užtrunka vidutiniškai, sekundžių	3.379
Nepavykusios užklausos	13

Lentelė 7. *Python kalbos eksperimento #2 rezultatai.*

7.3.3 PHP kalbos eksperimentas #2

Aktualu paminėti, kad darant šį eksperimentą, buvo modifikuojami Apache WEB serverio nustatymai, jog jis galėtų palaikyti 1000 paralelinių prisijungimu vienu metu, vietoj 150, pagal nutylėjimą. Programinio kodo priedo šiam eksperimentui nėra, kadangi Apache WEB serveris yra išorinis įrankis, kuris nėra kuriamas su pačia PHP programavimo kalba.

Eksperimentas buvo sėkmingas. Gautų rezultatų išvestį galima pamatyti 10 priede. Buvo gauti tokie rezultatai:

Metrika	**Rezultatas**
Užklausų sk. / sekundę	2359.03
Užklausa užtrunka vidutiniškai, sekundžių	0.424
Nepavykusios užklausos	0

Lentelė 8. PHP kalbos eksperimento #2 rezultatai.

7.4 Rezultatų apibendrinimas

	JavaScript (Node.js)	**Python**	**PHP**
Užklausų sk. / sekundę	7153.37 užklausos per sekundę	295.93 užklausos per sekundę	2359.03 užklausos per sekundę
Užklausa užtrunka vidutiniškai	139.794ms	3379.174ms	424.903ms
Nepavykusios užklausos	0	13	0

Lentelė 9. Eksperimento #2 rezultatų apibendrinimas

Iš šių rezultatų (lentelė 9) galima padaryti išvadą, jog Node.js express HTTP serveris yra geriausias iš visų testuotų variantų, nes sugebėjo apdoroti daugiausiai užklausų per sekundę (net ~3.03 karto) daugiau, nei antroje vietoje esantis PHP. Ir trečioje vietoje lieka Python kalba, sugebėjusi apdoroti mažiausiai užklausų per sekundę iš visų ir be to, nesugebėjusi priimti 13 užklausų.

8 Rezultatai

Iškeltiems bakalauro darbo uždaviniams buvo pasiekti tokie rezultatai, kuriuos galima skirstyti pagal pagrindinius pilnos aplikacijos kūrimo aspektus:

- **Iš vartotojo sąsajos kūrimo perspektyvos, nustatyta kad:**
    - Tik su JavaScript kalba galima kurti WEB vieno puslapio aplikacijas (išskyrus Flutter technologiją paremtą Dart, tačiau turinčią kitų trūkumų), o iš jų – efektyviai paversti į pačio moderniausio tipo aplikacijas - progresyvias WEB aplikacijas, kurios gali būti naudojamos ir kaip mobiliosios aplikacijos.
    - JavaScript kalba leidžia efektyviau už kitas kalbas rašyti vartotojo sąsajos programinį kodą ir jį palaikyti ar atnaujinti, dėl galimybės rašyti atominį kodą, pasitelkiant modernias vartotojo sąsajos kūrimo karkasines sistemas.
    - JavaScript kalba leidžia su viena kodo baze kurti vartotojo sąsajos aplikacijas skirtingoms platformoms ar operacinėms sistemoms.
    - JavaScript kalba turi webpack technologiją, kuri padidina kodo rašymo efektyvumą, leidžia naudoti pačius naujausius kalbos standartus, kurių paprastai galbūt nepalaikytų kliento aplinka ir automatiškai atlieka svarbias optimizacijas - kodo minimizavimą, užmaskavimą, nereikalingo kodo pašalinimą, loginį optimizavimą.
    - Virtualaus objekto modelio optimizacija supaprastina kodo rašymą programuotojams ir užtikrina didesnę produkto kokybę galutiniam naudos gavėjui - aplikacijos naudotojui, klientui.
- **Iš programų sistemų kūrimo perspektyvos, nustatyta kad:**
    - Node.js turi puikią architektūrą kurti populiarėjančio mikroservisų architektūrinio stiliaus programų sistemas.
    - Node.js gali išgauti didelį efektyvumą iš vienos darbinės gijos, dėl įvykių ciklo algoritmo.
    - Node.js turi puikią architektūrą asinchroniniam programavimui, kuris šiais laikais yra naudojamas beveik visose pilnose aplikacijose.
    - Didžiųjų kompanijų sėkmės istorijos rodo, jog praktikoje Node.js gali padidinti serverinių aplikacijų efektyvumą net iki 20 kartų ir pakelti kodo rašymo efektyvumą iki 33%.

    - Pagal atliktus eksperimentus, palyginime su PHP ir Python kalbomis, turi geriausią darbinį programavimo kalbos efektyvumą atliekant standartines operacijas ir HTTP užklausų apdorojimą.
- **Pagal bendrus, technologinius aspektus buvo nustatyta kad:**
    - Tik su JavaScript (išskyrus Flutter technologiją paremtą Dart, tačiau turinčią kitų trūkumų), yra įmanoma sukurti WEB, mobilią ir darbalaukio aplikacijas skirtingoms operacinėms sistemoms, kartu su logine serverio dalimi naudojant tik vieną kalbą, naudojant tik 2 kodo bazes - vieną vartotojo sąsajai, kitą loginiai serverio daliai.
    - JavaScript yra vienintelė kalba (išskyrus Flutter technologiją paremtą Dart, tačiau turinčią kitų trūkumų), kurios dėka galima pernaudoti tą patį rašomą kodą mobiliosiose ir WEB aplikacijose tarp serverio ir kliento kodo bazių.
    - JavaScript yra vienintelė kalba (išskyrus Flutter technologiją paremtą Dart, tačiau turinčią kitų trūkumų), kurios dėka galima pernaudoti statinius kalbos tipus kuriant pilnas WEB ar mobilias aplikacijas.
    - Didžiųjų gigantų sėkmės istorijos rodo, jog aplikacijų kūrimas su JavaScript gali pakelti programuotojų produktyvumą jei kuriamos aplikacijos tiek loginė tiek vartotojo sąsajos dalys yra kuriamos su JavaScript.
    - JavaScript kalbos profesionalų yra daugiausiai, kas užtikriną profesionalų pasiūlą, greitą problemų sprendimą ir aukštesnius kokybės standartus.
    - Didžioji dalis pačių populiariausių ir labiausiai naudojamų programuotojų viešo naudojimo technologijų yra parašytos JavaScript kalba.
    - JavaScript kalba turi didžiausią kiekį viešai galimų naudoti įrankių.
    - JavaScript kalbos viešų įrankių kasdieninis prieaugis yra pats didžiausias, lenkiantis kita populiariausią kalbą net 3,78 karto, kas prognozuoja ir tolimesnį šios kalbos augimą.

9 Išvados

Apibendrinant, galima teigti, jog pagal išsikeltus tyrimo aspektus JavaScript kalba parodė puikius rezultatus. JavaScript turėjo privalumų visose pilnų aplikacijų kūrimui aktualiose kategorijose – vartotojo sąsajos kūrime, programų sistemų kūrime ir apskritai puikiai pasirodė kaip programavimo kalba pagal bendrus technologinius aspektus. Todėl JavaScript kalbos potencialas kuriant pilnas aplikacijas šiuo metu, palyginus su kitomis programavimo kalbomis, yra didelis ir kaip rodo šios kalbos populiarumo statistikos – daugiausiai turinčių viešų įrankių, greičiausiai atsirandančių įrankių per dieną, populiariausių egzistuojančių naudojamų įrankių, populiarumo tarp programuotojų – ši kalba ir toliau turėtų tokia išlikti, todėl JavaScript yra rekomenduotinas pasirinkimas norint kurti pilnas aplikacijas.

Šaltiniai

- [Ham18] Mosh Hamedani, React Virtual DOM Explained in Simple English, 2018. https://programmingwithmosh.com/react/react-virtual-dom-explained/
- [Cap19] Tomislav Capan. Why The Hell Would I Use Node.js? A Case-by-Case Tutorial, 2019. https://www.toptal.com/nodejs/why-the-hell-would-i-use-node-js
- [Kum19] Ankit Kumar. Microservices With Node.js: Scalable, Superior, and Secure Apps, 2019. https://dzone.com/articles/microservices-with-nodejs-scalable-superior-and-se
- [Pja20] JS MeetUP PWA, 2020. https://pjagajitprusty.github.io/PWD/
- [Mod20] Module Counts, 2020. http://www.modulecounts.com/
- [Nod20] About | Node.js, 2020. https://nodejs.org/en/about/
- [Api17] PROGRESSIVE WEB APPS: KEY BENEFITS, STATISTICS, USE CASES, 2017. https://apiumhub.com/tech-blog-barcelona/progressive-web-apps/
- [Ale18] The Good and the Bad of JavaScript Full Stack Development, 2018. https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-javascript-full-stack-development/
- [Pay13] Node.js at PayPal, 2013. https://medium.com/paypal-engineering/node-js-at-paypal-4e2d1d08ce4f
- [Sta19] Stackoverflow.com. Developer Survey Results, 2019. https://insights.stackoverflow.com/survey/2019
- [Sta18] Stackoverflow.com. Developer Survey Results, 2018. https://insights.stackoverflow.com/survey/2018
- [Lan12] Ikai Lan, Clearing up some things about LinkedIn mobile's move from Rails to node.js, 2012. https://ikaisays.com/2012/10/04/clearing-up-some-things-about-linkedin-mobiles-move-from-rails-to-node-js/

- [Hof12] Todd Hoff, LinkedIn Moved From Rails To Node: 27 Servers Cut And Up To 20x Faster, 2012. http://highscalability.com/blog/2012/10/4/linkedin-moved-from-rails-to-node-27-servers-cut-and-up-to-2.html
- [Ofo18] Victor Ofoegbu, The only NodeJs introduction you'll ever need, 2018. https://codeburst.io/the-only-nodejs-introduction-youll-ever-need-d969a47ef219
- [Git20] Github.com repository statistics by most stars, 2020. https://github.com/search?p=1&q=stars%3A%3E1&type=Repositories.
- [Jet19] Jetbrains.com. The State of Developer Ecosystem, 2019. https://www.jetbrains.com/lp/devecosystem-2019/.
- [Web08] Webreference.com. Performance Optimizations for High Speed JavaScript, 2018. http://webreference.com/programming/javascript/jkm3/index-2.html
- [Git20a] Github, React Native for Web, 2020. https://github.com/necolas/react-native-web

Priedai

1 priedas. Eksperimento #1 naudojamas JavaScript (Node.js) programinis kodas

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

// skaičių sudėjimas 10mln. kartų
let sum = 0;
console.time('sum')
for (i = 1; i <= 100000000; i++) {
    sum += i
}
console.log('10 mln. skaičių sumavimas užtruko:')
console.timeEnd('sum')

//masyvo užpildymas su 10mln. elementų.
console.time('array_fill')
const array = [];
for (i = 1; i <= 5000000; i++) {
    array.push(i)
}
console.log('5 mln. elementų užpildymas į masyvą užtruko:')
console.timeEnd('array_fill')


//100 MB failo nuskaitymas 100 kartų.
console.time('files_read')
for (i = 1; i <= 100; i++) {
    fs.readFileSync('../../files/100MB_FILE.dat')
}
console.log('Nuskaityti 100MB failą 100 kartų užtruko:')
console.timeEnd('files_read')
```

Paveiksliukas 12. Eksperimento #1 naudojamas JavaScript (Node.js) programinis kodas.

2 priedas. Eksperimento #1 JavaScript (Node.js) kalbos išvestis.

```
100 mln. skaičių sumavimas užtruko:
sum: 194.902ms
5 mln. elementų užpildymas į masyvą užtruko:
array_fill: 108.324ms
Nuskaityti 100MB failą 100 kartų užtruko:
files_read: 4068.803ms
```

Paveiksliukas 13. Eksperimento #1 JavaScript (Node.js) kalbos išvestis.

3 priedas. Eksperimento #1 Python kalbos naudojamas programinis kodas.

```python
import time


start = time.time()
sum = 0;
for i in range(1, 100000001):
    sum += i
end = time.time()
print("10 mln. skaičių sumavimas užtruko: " + str(end - start))

start = time.time()
array = [];
for i in range(1, 5000001):
    array.append(i)
end = time.time()
print("5 mln. elementų užpildymas į masyvą užtruko:" + str(end - start))

start = time.time()
for i in range(1, 101):
    f = open("../../files/100MB_FILE.dat")
    f.read()
    f.close()
end = time.time()
print("Nuskaityti 100MB failą 100 kartų užtruko:"  + str(end - start))

```

Paveiksliukas 14. Eksperimento #1 naudojamas JavaScript (Node.js) programinis kodas.

4 priedas. Eksperimento #1 Python kalbos išvestis.

```
100 mln. skaičių sumavimas užtruko: 12.276090860366821
5 mln. elementų užpildymas į masyvą užtruko: 0.5230183601379395
Nuskaityti 100MB failą 100 kartų užtruko: 4.444139003753662
```

Paveiksliukas 15. Eksperimento #1 Python kalbos išvestis.

5 priedas. Eksperimento #1 PHP kalbos naudojamas programinis kodas.

```php
<?php
ini_set('memory_limit', '2000M');

$time_pre = microtime(true);
$sum = 0;
for ($x = 1; $x <= 100000000; $x++) {
    $sum += $x;
}
$time_post = microtime(true);
echo "100 mln. skaičių sumavimas užtruko: " . ($time_post - $time_pre);
echo "\n";

$time_pre = microtime(true);
$arr = array();
for ($x = 1; $x <= 5000000; $x++) {
    array_push($arr, $x);
}
$time_post = microtime(true);
echo "5 mln. elementų užpildymas į masyvą užtruko: " . ($time_post - $time_pre);
echo "\n";

$time_pre = microtime(true);
$fileSize = filesize("C:/Users/artsv/Desktop/tests/1_experiment/files/100MB_FILE.dat");
for ($x = 1; $x <= 100; $x++) {
    $myfile = fopen("C:/Users/artsv/Desktop/tests/1_experiment/files/100MB_FILE.dat", "r");
    fread($myfile, $fileSize);
    fclose($myfile);
}
$time_post = microtime(true);
echo "Nuskaityti 100MB failą 100 kartų užtruko: " . ($time_post - $time_pre);

?>
```

Paveiksliukas 16. Eksperimento #1 PHP kalbos naudojamas programinis kodas.

6 priedas. Eksperimento #1 PHP kalbos išvestis.

```
100 mln. skaičių sumavimas užtruko: 1.1823589801788
5 mln. elementų užpildymas į masyvą užtruko: 0.21944284439087
Nuskaityti 100MB failą 100 kartų užtruko: 7.2744379043579
```

Paveiksliukas 11. Eksperimento #1 PHP kalbos išvestis.

7 priedas. Eksperimento #2 JavaScript (Node.js) kalbos naudojamas programinis kodas.

```
1  const express = require("express");
2  const app = express();
3
4  app.get("/", function(req, res) {
5    res.send("Hello World!");
6  });
7
8  app.listen(3000);
```

Paveiksliukas 17. Eksperimento #2 JavaScript (Node.js) kalbos naudojamas programinis kodas.

8 priedas. Eksperimento #2 JavaScript (Node.js) kalbos išvestis.

```
Server Software:
Server Hostname:        localhost
Server Port:            3000

Document Path:          /
Document Length:        12 bytes

Concurrency Level:      1000
Time taken for tests:   13.979 seconds
Complete requests:      100000
Failed requests:        0
Keep-Alive requests:    100000
Total transferred:      21600000 bytes
HTML transferred:       1200000 bytes
Requests per second:    7153.37 [#/sec] (mean)
Time per request:       139.794 [ms] (mean)
Time per request:       0.140 [ms] (mean, across all concurrent requests)
Transfer rate:          1508.91 [Kbytes/sec] received
```

Paveiksliukas 18. Eksperimento #2 JavaScript (Node.js) kalbos išvestis.

9 priedas. Eksperimento #2 Python kalbos naudojamas programinis kodas.

```python
from flask import Flask
import logging

log = logging.getLogger('werkzeug')
log.setLevel(logging.ERROR)


app = Flask(__name__)
app.logger.disabled = True;

@app.route('/')
def index():
    return 'Hello World!'

if __name__ == '__main__':
    app.run(host='0.0.0.0')
```

Paveiksliukas 19. Eksperimento #2 Python kalbos naudojamas programinis kodas.

10 priedas. Eksperimento #2 Python kalbos išvestis.

```
Server Hostname:        localhost
Server Port:            5000

Document Path:          /
Document Length:        12 bytes

Concurrency Level:      1000
Time taken for tests:   337.917 seconds
Complete requests:      100000
Failed requests:        13
   (Connect: 0, Receive: 0, Length: 13, Exceptions: 0)
Keep-Alive requests:    0
Total transferred:      16599570 bytes
HTML transferred:       1199844 bytes
Requests per second:    295.93 [#/sec] (mean)
Time per request:       3379.174 [ms] (mean)
Time per request:       3.379 [ms] (mean, across all concurrent requests)
Transfer rate:          47.97 [Kbytes/sec] received
```

Paveiksliukas 20. Eksperimento #2 Python kalbos išvestis.

11 priedas. Eksperimento #2 PHP kalbos išvestis.

```
Server Software:        Apache/2.4.41
Server Hostname:        localhost
Server Port:            80

Document Path:          /
Document Length:        43 bytes

Concurrency Level:      1000
Time taken for tests:   42.390 seconds
Complete requests:      100000
Failed requests:        0
Keep-Alive requests:    99539
Total transferred:      29678864 bytes
HTML transferred:       4300000 bytes
Requests per second:    2359.03 [#/sec] (mean)
Time per request:       423.903 [ms] (mean)
Time per request:       0.424 [ms] (mean, across all concurrent requests)
Transfer rate:          683.72 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.2      0      45
Processing:     2  422 292.6    405    1985
Waiting:        2  422 292.7    405    1985
Total:          2  422 292.6    405    1985
```

Paveiksliukas 21. Eksperimento #2 PHP kalbos išvestis.