Određivanje mjerila izvedbe Poissonovskih i ne-Poissonovskih podvorbenih sustava

Herout, Tomislav

SVEUČILIŠTE U ZAGREBU
FAKULTET PROMETNIH ZNANOSTI

Tomislav Herout

ODREĐIVANJE MJERILA IZVEDBE POISSONOVSKIH I NE-POISSONOVSKIH PODVORBENIH SUSTAVA

DIPLOMSKI RAD

Zagreb, 2019.

SVEUČILIŠTE U ZAGREBU
FAKULTET PROMETNIH ZNANOSTI
POVJERENSTVO ZA DIPLOMSKI ISPIT

Zagreb, 10. travnja 2019.

Zavod: **Zavod za informacijsko komunikacijski promet**
Predmet: **Podvorbeni sustavi**

DIPLOMSKI ZADATAK br. 5396

Pristupnik: **Tomislav Herout (0069051667)**
Studij: Promet
Smjer: Informacijsko-komunikacijski promet

Zadatak: **Određivanje mjerila izvedbe Poissonovskih i ne-Poissonovskih podvorbenih sustava**

Opis zadatka:

Analizirati značajke različitih Poissonovskih i ne-Poissonovskih podvorbenih sustava te definirati osnovna mjerila izvedbe takvih sustava. Osnovna mjerila izvedbe trebaju uključivati mjerila poput prosječnog vremena posluživanja, prosječnog vremena zadržavanja u sustavu, prosječnog vremena čekanja (ukoliko se radi o sustavu koji se sastoji i od reda, odnosno repa), prosječan broj korisnika u sustavu ili redu. Temeljem dosegnutih spoznaja o mjerilima izvedbe, napraviti internet aplikaciju koja će omogućiti korisnicima računanje podržanih mjerila izvedbe, za određene modele podvorbenih sustava.

Mentor:

doc. dr. sc. Marko Matulin

Predsjednik povjerenstva za diplomski ispit:

Sveučilište u Zagrebu
Fakultet prometnih znanosti

DIPLOMSKI RAD

ODREĐIVANJE MJERILA IZVEDBE POISSONOVSKIH I NE-POISSONOVSKIH PODVORBENIH SUSTAVA

DETERMINING PERFORMANCE MEASURES OF POISSON AND NON-POISSON QUEUEING SYSTEMS

Mentor: doc. dr. sc. Marko Matulin

Student: Tomislav Herout
JMBAG: 0069051667

Zagreb, rujan 2019.

SAŽETAK

U okviru diplomskog rada napravljena je web aplikacija koja pojednostavljuje određivanje mjerila izvedbe poissonovskih i ne-poissonovskih podvorbenih sustava. Podvorbeni sustav je model koji se sastoji od korisnika, reda, poslužitelja i zbirke pravila prema kojima se obavlja posluživanje dolazećeg korisnika. Za izradu aplikacije prethodno je napravljena analiza određenih podvorbenih sustava. Aplikacija izračunava mjerila izvedbe na temelju ulaznih parametara i deterministčkih pravila. Klijentska i poslužiteljska strana razvijene su u najnovijim web tehnologijama. Učinkoviti podvorbeni sustavi oduvijek su bili od velike važnosti velikim kompanijama. One ulažu znatne količine novca kako bi smanjile vrijeme odgovora poslužitelja. Ljudi će se i dalje truditi što više optimizirati sustave čekanja, te je iz toga razloga ovo područje jedan od glavnih generatora potencijala u budućnosti.

Ključne riječi: poissonovski podvorbeni sustavi, ne-poissonovski podvorbeni sustavi, web aplikacija za određivanje mjerila izvedbe podvorbenih sustava

ABSTRACT

This Thesis shows implementation of the web application which simplifies determining benchmarks of the Poisson's and non-Poisson's queueing systems. Queueing system is a model that consists of client, queue, server and different rules based on which active user gets served. Prior to development of the web application, analyses of the specific queueing systems were made. Application calculates metrics of performance based on input parameters and determined rules. Client-side and server-side were developed with cutting edge technologies. Efficient queueing systems were always of great importance for big companies. They invest substantial amount of money in order to reduce the server response time. People will continue to strive to optimizing queueing systems as much as possible, and therefore this area remains one of the main generators of potential in the future.

Keywords: Poisson queueing systems, non-Poisson queueing systems, web application for determining performance measures of queueing systems

SADRŽAJ

1. UVOD

U ovom diplomskom radu potrebno je izraditi web aplikaciju za određivanje mjerila izvedbe poissonovskih i ne-poissonovskih podvorbenih sustava. Podvorbeni sustavi su sustavi koji se sastoje od korisnika, reda, poslužitelja i pravila koja definiraju ponašanje u redu. Navedeni se sustavi koriste svakodnevno u različitim granama poslovanja, odnosno postoje u različitim izvedbama. Primjer podvorbenog sustava može biti sustav posluživanja u bankama, poštanskim uredima ili telekomunikacijskim sustavima, poput telefonskih centrala ili internetskih čvorova. Poissonovski podvorbeni sustavi su sustavi kod kojih se razdioba međudolaznih vremena i razdioba trajanja posluživanja ravna po eksponencijalnoj razdiobi. U slučaju kada se jedna ili obje razdiobe vremena ne mogu opisati eksponencijalnom razdiobom tada se radi o takozvanim ne-poissonovskim podvorbenim sustavima.

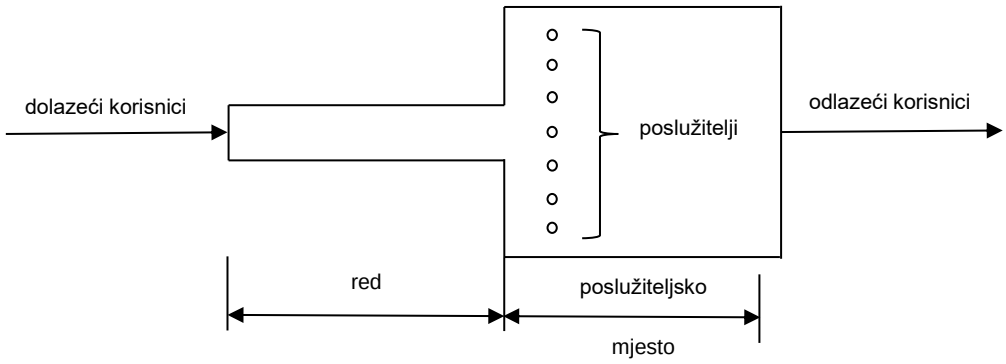
Cilj aplikacije je pružiti računsku pomoć kod određivanja mjerila izvedbe podvorbenog sustava u fazama njegovog planiranja i kapacitiranja. Diplomski rad se sastoji od šest poglavlja:

1. Uvod
2. Podvorbeni sustavi
3. Programski alati i jezici
4. Arhitektura sustava
5. Određivanje performansi podvorbenih sustava
6. Zaključak.

U drugom poglavlju dan je općeniti uvod u to što su podvorbeni sustavi. Osim toga, opisane su značajke različitih poissonovskih i ne-poissonovskih podvorbenih sustava i definirana su osnovna mjerila izvedbe takvih modela. U sljedećem poglavlju opisani su programski alati i jezici korišteni za izradu aplikacije. Četvrto poglavlje prikazuje arhitekturu sustava, odnosno dan je opis klijentske i poslužiteljske strane aplikacije. U petom poglavlju prikazano je kako se mogu odrediti performanse podvorbenih sustava koristeći izrađenu aplikaciju. Zadnje poglavlje, odnosno zaključak donosi osvrt na razrađenu temu.

2. PODVORBENI SUSTAVI

Podvorbeni sustav je sustav koji se sastoji od korisnika, reda, poslužitelja i zbirke pravila prema kojima se obavlja posluživanje dolazećeg korisnika. Shema podvorbenog sustava prikazana je na slici 1. Teorija podvorbenih sustava bavi se jednim od najneugodnijih iskustava za korisnika neke usluge, a to je čekanje. Redovi čekanja se često pojavljuju u različitim područjima: telekomunikacijskim sustavima, trgovinama, na benzinskoj postaji, u pošti i slično, [1, 2].



Slika 1 Shematski prikaz podvorbenog sustava, [1]

Korisnik je osoba ili stvar koja zahtjeva podvorbu. Korisnici dolaze iz „izvorišta“, gdje se nalazi *dolazno pučanstvo*. Izvorište se smatra beskonačno velikim, ako je broj korisnika u izvorištu toliki, da na vjerojatnost dolaska korisnika na posluživanje ne utječe broj korisnika koji su prije otišli u podvorbeni sustav i nalaze se bilo u redu bilo kod poslužitelja. U suprotnome izvorište se smatra konačnim.

Red je skup korisnika koji čekaju na posluživanje. Kapacitet reda može biti beskonačan ili konačan. Korisnik ne mora čekati u redu na posluživanje ako je red prazan i ako poslužitelj nije zauzet. Ako je poslužitelj zauzet ili red nije prazan, korisnik mora čekati neko vrijeme kako bi došao na red za posluživanje. Najvažnija značajka podvorbenog sustava je podvorbena stega. Podvorbena stega je način na koji korisnici ulaze u red, kako se ponašaju u redu, i kako odlaze iz reda poslužitelju. Sastoji se od triju pravila: pravilo ulaska u red, pravilo boravka u redu, i pravilo odlaska iz reda poslužitelju.

Poslužiteljsko mjesto je dio podvorbenog sustava gdje se nalaze poslužitelji koji obavljaju podvorbu. Za poslužiteljsko mjesto je važno utvrditi broj poslužitelja i način rada ili organizaciju posluživanja.

Kada se koriste kao uporabni objekt za opis stvarnih podvorbenih situacija podvorbeni sustavi se nazivaju modelima. Prema [1, 3], glavna obilježja modela opisuju se Kendallovom oznakom koja se sastoji od šest simbola:

$$A / B / C / K_q / K_i / P.S$$

gdje su:

- A: oznaka za razdiobu međudolaznih vremena
- B: oznaka za razdiobu trajanja posluživanja
- C: broj poslužitelja
- K_q: kapacitet sustava (najveći dopušteni broj korisnika u podvorbenom sustavu)
- K_i: kapacitet izvora (najveći dopušteni broj korisnika u izvoru)
- P.S: podvorbena stega, npr. FCFS (eng. *First Come First Served*).

2.1. Poissonovski podvorbeni sustavi

Podvorbeni sustavi kod kojih su procesi dolazaka i posluživanja Poissonovi slučajni procesi, nazivaju se poissonovskima. Ovakvi sustavi još se nazivaju i Markovljevi procesi jer trenutci prijelaza među stanjima ovise samo o sadašnjem stanju. Najjednostavniji poissonovski podvorbeni sustav je *M/M/1/∞/∞/FCFS,* [1, 2]. U nastavku su opisani sustavi i njima pripadajuća mjerila izvedbe koja aplikacija podržava.

2.1.1. Sustav M/M/1/∞/∞/FCFS

Sustav *M/M/1/∞/∞/FCFS* je najjednostavniji jednoposlužiteljski sustav. Ulazne veličine ovog sustava mogu se iščitati iz Kendallove oznake:

- M: funkcija razdiobe međudolaznih vremena je eksponencijalna
- M: funkcija razdiobe trajanja posluživanje je eksponencijalna

- 1: poslužiteljsko mjesto ima jednog poslužitelja
- ∞: beskonačni spremnik
- ∞: izvorište nema ograničenja
- FCFS: podvorbena stega prvi ušao prvi poslužen.

Za svaki sustav na temelju zadanih podataka mogu se izračunati određene izlazne veličine sustava. U ovome radu, bit će za svaki sustav opisane formule, koje su korištene u aplikaciji za računanje izlaznih veličina na temelju zadanih vrijednosti. Detaljnije informacije o izlaznim veličinama i izvodima se mogu pronaći u [1]. Iz navedenog izvora preuzeti su i svi ovdje prezentirani izrazi.

Za trenutni sustav i poznate vrijednosti intenziteta dolazaka λ i intenziteta posluživanja μ mogu se izračunati mjerila izvedbe koja su opisana u nastavku.

Ponuđeni promet α koji se računa kao umnožak intenziteta dolazaka λ i prosječnog vremena posluživanja T_s ili kao omjer intenziteta dolazaka λ i intenziteta posluživanja μ:

$$a = \lambda \cdot T_s = \frac{\lambda}{\mu} \tag{1}$$

Prosječan broj korisnika u sustavu L_q može se izračunati pomoću sljedeće formule:

$$L_q = \frac{a}{1-a} = \frac{\lambda}{\mu - \lambda} \tag{2}$$

Prosječan broj korisnika u redu L_w:

$$L_w = \frac{a^2}{1-a} \tag{3}$$

Prosječan broj korisnika u poslužiteljskom mjestu L_s, može se izračunati kao razlika prosječnog broja korisnika u sustavu L_q i prosječnog broja korisnika u redu L_w:

$$L_s = L_q - L_w \tag{4}$$

Prosječno vrijeme posluživanja korisnika T_s se može izračunati kao omjer ponuđenog prometa α i intenziteta dolazaka λ:

$$T_s = \frac{a}{\lambda} \tag{5}$$

Prosječno vrijeme boravka u sustavu T_q se može izračunati pomoću formule:

$$T_q = \frac{1}{\mu - \lambda} \tag{6}$$

Prosječno vrijeme čekanja T_w se može izračunati kao razlika prosječnog vremena boravka u sustavu T_q i prosječnog vremena posluživanja T_s ili kao omjer prosječnog broja korisnika u redu L_w i intenziteta dolazaka λ:

$$T_w = T_q - T_s = \frac{L_w}{\lambda} \tag{7}$$

Iskoristivost poslužitelja ρ' je obavljeni promet na jednoga poslužitelja u stacionarnom stanju sustava. Zato što je navedeni sustav, sustav bez gubitaka i sustav s jednim poslužiteljem, iskoristivost poslužitelja ρ' jednaka je jakosti promet ρ koji je pak jednak ponuđenom prometu a:

$$\rho' = \rho = a \tag{8}$$

Vjerojatnost da je n korisnika u sustavu Q_n računa se formulom:

$$Q_n = a^n \cdot (1 - a) \tag{9}$$

2.1.2. Sustav M/M/1/K/∞/FCFS

Ovaj podvorbeni sustav modifikacija je prethodno objašnjenog sustava. U njemu može boraviti najviše $K_q = K$ korisnika. To bi značilo da je jedan korisnik kod poslužitelja i K - 1 korisnika u redu. Takvi podvorbeni sustavi s ograničenim brojem korisnika u sustavu se katkada nazivaju „potkresani“ (eng. *truncated*), [1, 4].

Ulazne veličine ovog sustava opisane su Kendallovom oznakom, te se iz nje mogu iščitati sljedeći podaci:

- M: funkcija razdiobe međudolaznih vremena je eksponencijalna
- M: funkcija razdiobe trajanja posluživanje je eksponencijalna
- 1: poslužiteljsko mjesto ima jednog poslužitelja
- K: u sustavu ima mjesta samo za K korisnika
- ∞: izvorište nema ograničenja

- FCFS: podvorbena stega prvi ušao prvi poslužen.

Uz poznate vrijednosti intenziteta dolazaka λ, intenziteta posluživanja μ, i kapaciteta sustava *K*, mogu se izračunati mjerila izvedbe koja su opisana u nastavku.

Ponuđeni promet *α* se može izračunati kao i kod prethodnog sustava, koristeći izraz (1).

Posljedica ograničenog kapaciteta sustava, a time i reda, jest da nakon popunjavanja reda više nijedan korisnik ne može ući u sustav. Stoga u takvim sustavima treba razlikovati korisnike koji *dolaze pred* podvorbeni sustav od onih koji *ulaze* u podvorbeni sustav. Oni prvi čine *brzinu dolazaka pred* podvorbeni sustav, λ, a ovi drugi *brzinu ulazaka u* podvorbeni sustav, λ', [1]. Kako su korisnici gubitak onda kada je sustav u stanju [$n_q = K$], brzina ulazaka je:

$$\lambda' = \lambda \cdot (1 - Q_k) \tag{10}$$

gdje je $Q_k = P$ [$n_q = K$] funkcija gustoće vjerojatnosti slučajne varijable „broj korisnika u sustavu“.

Zbog gubitaka korisnika iz gore navedenog razloga obavljeni promet *α′*, se razlikuje od obavljenog prometa *α*:

$$a' = a \cdot (1 - Q_k) \tag{11}$$

Slično vrijedi i za faktor iskoristivosti:

$$\rho' = \rho \cdot (1 - Q_k) \tag{12}$$

Kako se ovdje radi o jednoposlužiteljskom sustavu, ponuđeni promet *α* i jakost prometa *ρ* su jednaki. Sukladno tome jednaki su i obavljeni promet *α′* i faktor iskoristivosti *ρ′:*

$$a = \rho \tag{13}$$
$$a' = \rho' \tag{14}$$

Prosječan broj korisnika u sustavu L_q, može se izračunati koristeći formulu:

$$L_q = \frac{a}{1-a} - \frac{(K+1)\cdot a^{K+1}}{1-a^{K+1}} \tag{15}$$

Prosječan broj korisnika u poslužiteljskom mjestu L_s, jednak je obavljenom promet a':

$$L_s = a' = a \cdot (1 - Q_k) \tag{16}$$

Prosječan broj korisnika u redu L_w jednak je razlici prosječnog broja korisnika u sustavu L_q i prosječnog broja korisnika u poslužiteljskom mjestu L_s :

$$L_w = L_q - L_s \tag{17}$$

Prosječno vrijeme čekanja u redu T_w omjer je prosječnog broja korisnika u redu L_w i brzine ulaska λ':

$$T_w = \frac{L_w}{\lambda'} = \frac{L_w}{\lambda \cdot (1 - Q_k)} \tag{18}$$

Slična formula vrijedi i za prosječno vrijeme boravka u sustavu T_q:

$$T_q = \frac{L_q}{\lambda'} = \frac{L_q}{\lambda \cdot (1 - Q_k)} \tag{19}$$

Prosječno vrijeme posluživanja korisnika T_s, se može izračunati kao razlika prosječnog vremena boravka u sustavu T_q i prosječnog vremena čekanja T_w:

$$T_s = T_q - T_w \tag{20}$$

Vjerojatnost da je n korisnika u sustavu Q_n:

$$Q_n = \frac{(1-a)\cdot a^n}{1-a^{K+1}} \tag{21}$$

2.1.3. Sustav M/M/c/∞/∞/FCFS

Za razliku od prethodno dva opisana podvorbena sustava koji su bili jendoposlužiteljski, ovaj podvorbeni sustav je višeposlužiteljski. Ovaj sustav ima sljedeća svojstva:

- korisnici dolaze na idealno slučajan način brzinom λ, što znači da je ulazni proces Poissonov slučajni proces

- razdioba trajanja posluživanja jednog poslužitelja je eksponencijalna s parametrom μ
- u poslužiteljskom mjestu se nalazi *c* identičnih, paralelnih poslužitelja
- poslužitelji uvijek poslužuju samo jednog korisnika koji im je dodijeljen, tj. ne postoji mogućnost da jednog korisnika poslužuje više poslužitelja odjednom
- izvorište i kapacitet sustava nemaju ograničenja
- prilikom dolaska korisnici neposredno ulaze najbližem slobodnom poslužitelju. Kada su svi poslužitelji zauzeti, formira se red dolazećih korisnika
- podvorbena stega je prvi došao prvi poslužen.

Sustav nema gubitaka što znači da je obavljeni promet *α′,* jednak ponuđenom prometu *α* i može se izračunati pomoću formule (1). Iz istog razloga faktor iskoristivosti *ρ′* jednak je jakosti prometa *ρ*.

Zbog *c* poslužitelja, ponuđeni promet *α* se razlikuje od jakosti promet *ρ*:

$$\rho = \frac{a}{c} = \frac{\lambda}{c \cdot \mu} \tag{22}$$

Poznajući vrijednosti intenziteta dolazaka λ, intenziteta posluživanja μ, i broja poslužitelja *c*, mogu se izračunati prethodno definirane izlazne veličine i mjerila izvedbe koja su opisana u nastavku.

Vjerojatnost da je sustav prazan Q_0 :

$$Q_0 = \frac{1}{Q_0^{-1}}$$
$$Q_0^{-1} = \sum_{n=0}^{c-1} \frac{a^n}{n!} + \frac{a^c}{c! \cdot (1-\rho)} \tag{23}$$

Vjerojatnost čekanja $E_{2,c}(a)$:

$$E_{2,c}(a) = \frac{\frac{a^c}{c!} \cdot Q_0}{1-\rho} = \frac{\frac{a^c}{c!}}{(1-\rho)\sum_{j=0}^{c-1} \frac{a^j}{j!} + \frac{a^c}{c!}} \tag{24}$$

Prosječan broj korisnika u redu L_w:

$$Lw = \frac{\rho \cdot E_{2,c}(a)}{1-\rho} \tag{25}$$

Prosječan broj korisnika u sustavu L_q, jednak je zbroju ponuđenog prometa α i prosječnog broja korisnika u redu L_w:

$$L_q = a + L_w \tag{26}$$

Prosječan broj korisnika u poslužiteljskom mjestu L_s, se može izračunati prema već definiranoj formuli (4), kao razlika prosječnog broja korisnika u sustavu L_q i prosječnog broja korisnika u redu L_w.

Prosječno vrijeme čekanja T_w :

$$T_w = \frac{E_{2,c}(a) \cdot T_s}{c \cdot (1-\rho)} \tag{27}$$

Prosječno vrijeme posluživanja T_s računa se kao omjer ponuđenog prometa α i intenziteta dolazaka λ, (5).

Prosječno vrijeme boravka u sustavu T_q zbroj je prosječnog vremena posluživanja T_s i prosječnog vremena čekanja u redu T_w:

$$T_q = T_s + T_w \tag{28}$$

Vjerojatnost da je n korisnika u sustavu, Q_n:

$$Q_n = \begin{cases} \frac{a^n}{n!} \cdot Q_0 & za\ n = 0,1,2 \ldots (c-1) \\ \frac{a^n}{c! \cdot c^{n-c}} \cdot Q_0 & za\ n \geq c \end{cases} \tag{29}$$

2.1.4. Sustav M/M/c/c/∞/FCFS

Ovaj podvorbeni sustav se razlikuje od prethodnog podvorbenog sustava po tome što ne postoji mogućnost stvaranja reda jer za njega nema mjesta. To je temeljni podvorbeni sustav s gubitcima. Podvorbeni sustav s gubitcima funkcionira tako da korisnici koji u trenutku dolaska pred podvorbeni sustav nailaze na sve poslužitelje zauzete, nepovratno odlaze iz sustava, [1]. Sustav ima sljedeća svojstva:

- funkcija razdiobe međudolaznih vremena je eksponencijalna
- funkcija razdiobe trajanja posluživanje je eksponencijalna
- poslužiteljsko mjesto ima *c* paralelnih istovrsnih poslužitelja
- ne postoji mogućnost stvaranje reda, jer kapacitet sustava je *c*
- kapacitet izvora je neograničen
- podvorbena stega je prvi došao, prvi poslužen.

Kako kod ovakvog podvorbenog sustava nema mogućnosti stvaranje reda, prosječan broj korisnika u redu L_w i prosječno vrijeme čekanja T_w jednaki su nuli:

$$Lw = Tw = 0 \tag{30}$$

Poznajući vrijednosti intenziteta dolazaka λ, intenziteta posluživanja μ i broja poslužitelja *c* mogu se izračunati mjerila izvedbe koja su opisana u nastavku.

Ponuđeni promet *α* se računa prema već dobro poznatoj formuli (1).

Jakost prometa *ρ* se računa jednako kao i za prethodni podvorbeni sustav, pomoću formule (22).

Vjerojatnost da se u sustavu nalazi *n* korisnika, Q_n:

$$Q_n = \frac{\frac{a^n}{n!}}{\sum_{j=0}^{c} \frac{a^j}{j!}} \tag{31}$$

Vjerojatnost da dolazeći korisnik ne može ući u podvorbeni sustav, $E_{1,c}(\alpha)$:

$$E_{1,c}(a) = \frac{\frac{a^c}{c!}}{\sum_{j=0}^{c} \frac{a^j}{j!}} = Q_c \tag{32}$$

Kako u ovom sustavu može doći do gubitaka korisnika, obavljeni promet *α′* se razlikuje od ponuđenog prometa *α:*

$$a' = a \cdot (1 - E_{1,c}(a)) \tag{33}$$

Iskoristivost poslužitelja *ρ′* omjer je obavljenog prometa *α′* i broja poslužitelja *c:*

$$\rho' = \frac{a'}{c} \tag{34}$$

U ovom sustavu treba razlikovati brzinu dolazaka korisnika pred podvorbeni sustav λ od brzine ulazaka korisnika u podvorbeni sustav λ':

$$\lambda' = \lambda \cdot (1 - E_{1,c}(a)) \tag{35}$$

Prosječni broj korisnika u sustavu L_q jednak je prosječnom broju zaposlenih poslužitelja L_s, a taj je jednak obavljenom prometu *α′*:

$$L_q = L_s = a' = a \cdot (1 - E_{1,c}(a)) \tag{36}$$

Prosječno vrijeme boravka u sustavu T_q jednako je prosječnom vremenu boravka kod poslužitelja T_s, koje je omjer prosječnog broja korisnika u sustavu L_q i brzine ulazaka korisnika u sustav λ':

$$T_q = T_s = \frac{L_q}{\lambda'} \tag{37}$$

2.2. Ne-poissonovski podvorbeni sustavi

Temeljni ne-poissonovski podvorbeni sustavi su *M/G/1/∞/∞/FCFS* i *G/M/1/∞/∞/FCFS.* Njihova zajednička značajka je da proces posluživanja ili dolaska više nije Poissonov, što znači da se u rješavanju tih sustava ne može rabiti svojstvo zaboravljivosti eksponencijalne razdiobe. Trenutci prijelaza među stanjima ne čine Markovljev proces jer ne ovise samo o sadašnjem, nego i prethodnim stanjima, [1, 2]. Od ova dva temeljna ne-Poisonovska sustava u ovome radu bit će detaljnije objašnjen samo *M/G/1/∞/∞/FCFS,* jer sustav *G/M/1/∞/∞/FCFS* nije podržan u aplikaciji.

2.2.1. Sustav M/G/1/∞/∞/FCFS

Sustav *M/G/1/∞/∞/FCFS* je najjednostavniji ne-poissonovski podvorbeni sustav koji se najčešće koristi u područjima računalstva i telekomunikacijama. Sustav ima definirana sljedeća svojstva:

- M: proces dolazaka je Poissonov s prosječnom brzinom dolazaka λ

- G: vrijeme posluživanja je opisano nekom općom razdiobom *G*
- 1: jednoposlužiteljski sustav
- ∞: neograničena duljina reda
- ∞: izvor nema ograničenja
- FCFS: podvorbena stega prvi ušao, prvi poslužen.

Poznajući ponuđeni promet *α*, funkciju gustoće vjerojatnosti vremena posluživanja i vremenski interval promatranja, mogu se izračunati mjerila izvedbe koja su opisana u nastavku. Svi ovdje prezentirani izrazi preuzeti su iz izvora [1].

Prosječno vrijeme posluživanja korisnika T_s:

$$T_s = \int_{-\infty}^{\infty} t \cdot f_s(t)\, dt \tag{38}$$

Varijanca $Var(\tau_s)$:

$$Var(\tau_s) = \int_{-\infty}^{\infty} t^2 \cdot f_s(t) dt - (T_s)^2 \tag{39}$$

Standardna devijacija $\sigma\tau_s$:

$$\sigma_{\tau_s} = \sqrt{Var(\tau_s)} \tag{40}$$

Koeficijent Varijacije $V\tau_s$ omjer je standardne devijacije $\sigma\tau_s$ i prosječnog vremena posluživanja T_s :

$$V_{\tau_s} = \frac{\sigma_{\tau_s}}{T_s} = \frac{\sqrt{Var(\tau_s)}}{T_s} \tag{41}$$

Prosječan broj korisnika u sustavu L_q:

$$L_q = a + \frac{a^2}{2 \cdot (1-a)} \cdot \left[1 + \left(\frac{\sigma_{\tau_s}}{T_s}\right)^2\right] \tag{42}$$

Prosječan broj korisnika u redu L_w razlika je prosječnog broja korisnika u sustavu L_q i ponuđenog prometa *α*:

$$L_w = L_q - a \tag{43}$$

Prosječan broj korisnika u poslužiteljskom mjestu L_s jednak je ponuđenom prometu *α:*

$$L_s = a \tag{44}$$

Prosječno vrijeme čekanja u redu T_w:

$$T_w = \frac{L_q}{\lambda} - \frac{1}{\mu} = \frac{\lambda^2 \cdot Var(\tau_s) + a^2}{2 \cdot \lambda \cdot (1 - a)} = \frac{a \cdot T_s}{2 \cdot (1 \quad -a)} \cdot \left[1 + \left(\frac{\sigma_{\tau_s}}{T_s}\right)^2\right] \tag{45}$$

Prosječno vrijeme boravka u sustavu T_q zbroj je prosječnog vremena čekanja T_w i prosječnog vremena posluživanja korisnika T_s:

$$T_q = T_w + T_s \tag{46}$$

Iskoristivost poslužitelja ρ' jednaka je ponuđenom prometu *α,* jer je sustav jednoposlužiteljski bez gubitaka:

$$\rho' = \rho = a = a' \tag{47}$$

2.2.2. Sustav M/E_k/1/∞/∞/FCFS

U ovom podvorbenom sustavu posebnost se očituje u načinu posluživanja. Posluživanje se odvija po Erlangovoj razdiobi *k*-tog reda s funkcijom gustoće vjerojatnosti.

Mjerila izvedbe koja se mogu izračunati uz poznate vrijednosti intenziteta dolazaka λ, standardne devijacije σ_{τ_s} i reda Erlangove razdiobe *k* opisana su u nastavku.

Prosječno vrijeme čekanja T_s:

$$\frac{\sigma_{\tau_s}}{T_s} = \frac{1}{\sqrt{k}} \qquad gdje\ je\ k\ red\ Erlamgove\ razdiobe \tag{48}$$

Ponuđeni promet *α* umnožak je prosječnog vremena čekanja T_s i intenziteta dolazaka λ:

$$a = \ \lambda \cdot T_s \tag{49}$$

Prosječan broj korisnika u redu L_w:

$$L_w = \frac{a^2 \cdot \left(1 + \frac{1}{k}\right)}{2 \cdot (1 - a)} \tag{50}$$

Prosječan broj korisnika u sustavu L_q:

$$L_q = L_w + a \tag{51}$$

Prosječan broj korisnika u poslužiteljskom mjestu L_s jednak je ponuđenom prometu α (44).

Prosječno vrijeme čekanja u redu T_w omjer je prosječnog broja korisnika u redu L_w i intenziteta dolazaka λ :

$$T_w = \frac{L_w}{\lambda} \tag{52}$$

Prosječno vrijeme boravka u sustavu T_q, zbroj je prosječnog vremena čekanja T_w i prosječnog vremena posluživanja korisnika T_s, (46).

Iskoristivost poslužitelja ρ' jednaka je ponuđenom prometu α, jer je sustav jednoposlužiteljski bez gubitaka, (47).

3. PROGRAMSKI ALATI I JEZICI

Za izradu aplikacije odabran je JavaScript programski jezik. JavaScript je odabran iz razloga što omogućuje razvoj, i klijentske, i poslužiteljske strane aplikacije. Za razvijanje aplikacije odnosno pisanje koda korišten je Visual Studio Code uređivač kodova.

3.1. JavaScript

JavaScript je jednostavan, interpretiran programski jezik. Najpoznatiji je kao skriptni jezik za izradu web stranica. Većina današnjih preglednika ima ugrađen JavaScript stroj (eng. *engine*) koji izvršava JavaScript kod. Kako popularnost JavaScripta raste tako ga sve više radnih okruženja koristi, poput NodeJS, Apache CouchDB, i Adobe Acrobat radnih okruženja. JavaScript na klijentskoj strani omogućuje dinamičko upravljanje stranicom, odnosno dizajniranje ponašanja aplikacije koje se temelji na događajima, [6].

Često se misli da je JavaScript jezik sličan Javi što je pogrešno. JavaScript je jezik koji tipove varijabli određuje dinamički, i koji podržava kreiranje objekata zasnovanih na prototipu. Sintaksa mu je namjerno slična Javi i C++ kako bi se smanjio broj novih koncepata potrebnih za učenje jezika. Primjer JavaScript koda prikazan je na slici 2.

```
1  function changeBackgroundColor(element, color) {
2    element.style.backgroundColor = color;
3  }
4
5  const element = document.getElementById('first-section');
6
7  changeBackgroundColor(element, 'blue');
```

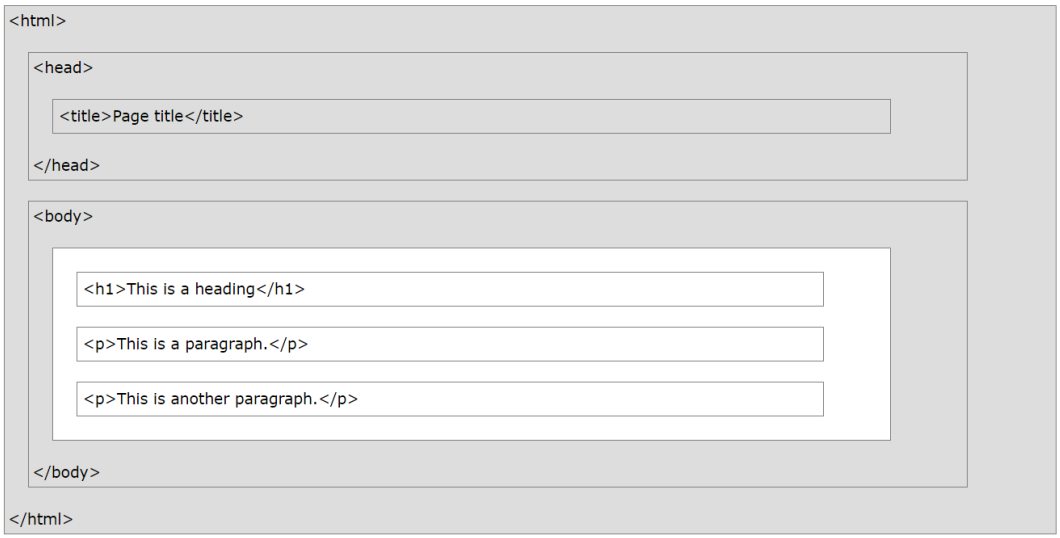
Slika 2 Primjer JavaScript koda

JavaScript je nastao 1995. godine, razvio ga je je Brendan Eich koji je u to vrijeme radio u *Netscape Communications* korporaciji, [7]. Od 1996. godine za standardizaciju JavaScripta zadužena je organizacija *ECMA* (eng. *European Computer Manufacturers Association*). Standardi se objavljuju pod nazivom *ECMAScript* i do sada je objavljeno deset inačica standarda *ECMA*-262.

3.2. HTML

HTML (eng. *HyperText Markup Language*) je jezik pomoću kojega se opisuje struktura i izgled web stranica, on nije programski jezik, već on daje upute web pregledniku na koji način treba prikazati stranicu. Ne pruža mogućnost definiranja funkcionalnosti osim osnovnih zadataka poput navigacije i slanja formi. On služi samo za opis hipertekstualnih dokumenata. Kako se HTML datoteke, tj. stranice, sastoje od običnog tekstualnog koda, mogu se pisati u najjednostavnijim tekstualnim uređivačima, kao što je i Notepad. HTML stranice sadrže standardnu ekstenziju .html ili .htm, [8].

Strukturu HTML dokumenta čine elementi kao što je vidljivo na slici 3. Elementi govore web pregledniku kako i što web stranica treba sadržavati. Općeniti elementi se sastoje od početnog taga, nekog sadržaja i završnog taga. Ponegdje se početni i završni tag nazivaju i početna, odnosno završna oznaka elementa. Tagovi su oznake web pregledniku za početak i kraj nekog elementa. Svi tagovi počinju sa znakom manje od "<" a završavaju znakom veće od ">". Općenito govoreći, postoje dvije vrste tagova: otvarajući tagovi npr. <html> i zatvarajući tagovi </html> Jedina razlika između otvarajućeg taga i zatvarajućeg taga je kosa crta "/". Sadržaj teksta se stavlja između otvarajuće i zatvarajuće oznake taga. Valja napomenuti da postoje i elementi koji se sastoje od samo jednog taga kao na primjer horizontalna linija (</ hr>) ili novi red (</ br>), [9].



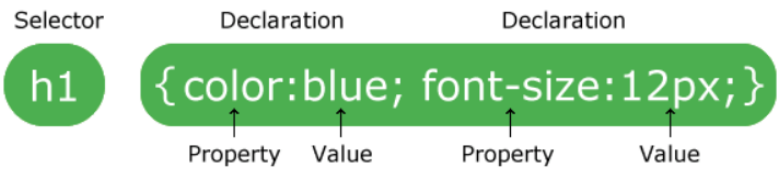
Slika 3 Struktura HTML koda, [10]

HTML5 je zadnja verzija HTML-a, nastao je kao nasljednik HTML 4.01 i XHTML (eng. *extensible HTML*) 1.1 standarda. Iz obje specifikacije je uzet dio koji se najčešće koristi te je zamijenjen način zapisivanja sintakse sa SGML-a (eng. *Standard Generalized Markup Language*) u zaseban opisni jezik. Dizajniran je sa svrhom isporuke skoro svih *online* sadržaja bez dodatnih programa kao što su dodaci pretraživačima i vanjske skripte. Također je *cross-platform*, što znači da radi jednako na svim platformama i preglednicima koji podržavaju HTML5 standard. Uvođenjem novih elemenata za pozicioniranje olakšava se izrada dizajna cijele web stranice, [11].

3.3. CSS

CSS (engl. *Cascading Style Sheets*) se pojavio relativno davne 1996. godine i predstavio potpuno novi način gledanja na web stranice i prikaz sadržaja. Omogućen je mnogo složeniji dizajn i jednostavan način održavanja na svim stranicama weba. CSS na jednostavan način omogućava razdvajanje izgleda web stranice od samog sadržaja. Kao standard CSS je donesen od strane W3C-a (engl. *World Wide Web Consortium*). Korištenjem CSS-a definirano je kako se prikazuje pojedini element web stranica. CSS omogućava definiranje klasa koje određuju svojstva nekog elementa. Pomoću CSS moguće je označiti različite elemente istom klasom i prikazivati ih na isti način, [8].

Pomoću CSS-a se određuje kako će se određeni element (ili element određene klase) prikazivati. Ti podaci upisuju se samo jednom u jednu CSS datoteku, a datoteku se mora povezati sa svakom stranicom na čiji izgled se utječe. CSS podaci su zapravo običan tekst spremljen u datoteku sa sufiksom .css. Gramatika CSS jezika sastoji se od raznih svojstava kojima se utječe na prikaz nekog elementa (npr. veličina fonta, pozadinska boja i sl.).



Slika 4 CSS sintaksa, [10]

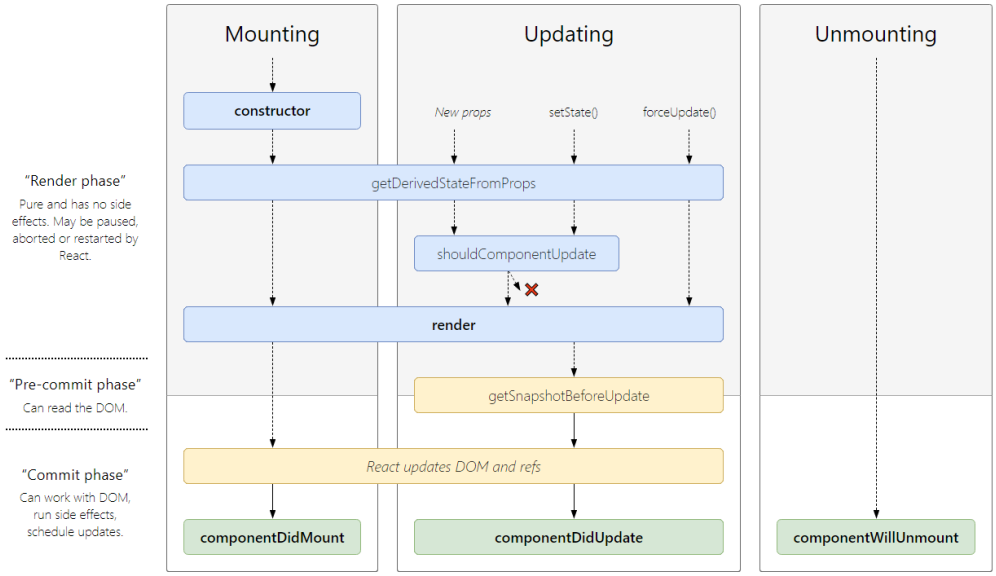
Na slici 4 prikazana je struktura CSS definicije. Svaka CSS definicija mora imati selektor i deklaraciju. Deklaracija dolazi odmah poslije selektora i sadržana je unutar vitičastih zagrada. Deklaracija se sastoji od jednog ili više svojstava odvojenih točka-zarezom. Svako svojstvo se sastoji od naziva, dvotočke i vrijednosti tog svojstva. Svojstvo može sadržavati više vrijednosti koje se odvajaju zarezom, a uz vrijednost može biti i jedinica kao na primjer px (eng. *pixel*). Između vrijednosti i jedinice ne smije biti razmak. Pri pisanju CSS-a mogu se koristiti razmaci i sve ostalo što može učiniti definicije čitljivijima, [10, 12].

3.4. ReactJS

ReactJS je učinkovita i fleksibilna JavaScript biblioteka koja se koristi za izgradnju korisničkih sučelja. Biblioteka je razvijena i održavana od strane Facebooka, a ističe se svojom jednostavnošću, brzinom i skalabilnosti. Navedene karakteristike ju čine jako popularnom u posljednje vrijeme. Kada bi se gledalo iz aspekta oblikovnog MVC (eng. *Model View Controller*) obrasca, React bi predstavljao Pogled (eng. *View*) komponentu, koja je zadužena za prikazivanje elemenata odnosno podataka korisniku, [13].

Komponente su osnovni dio aplikacija izrađenih u React biblioteci. Svaka komponenta je zasebna i višestruko upotrebljiva cjelina. Komponente se mogu definirati na dva načina: kao funkcije i kao klase. Najlakši način je definiranje komponente kao funkcije, koja prima objekt pod nazivom *props,* a vraća React element. Svakom promjena *props* objekta pokreće novi *render* komponente, kako bi prikazani podaci bili ažurni.

Komponente definirane pomoću klasa trenutno pružaju više opcija od funkcijskih komponentni. Svaka komponenta definirana pomoću klase može biti *statefull* što znači da može imati svoje interno stanje. Ako se želi koristiti interno stanje, ono se treba inicijalizirati u konstruktoru klase, a može se promijeniti metodom „*setState“*. Svaki „*setState“* poziv okida novi prolazak kroz životni ciklus komponente. Postoji nekoliko faza životnog ciklusa (slika 5) pomoću kojih se mogu izvršiti određene akcije nad podacima ili samim komponentama.



Slika 5 Dijagram životnog ciklusa React komponente, [14]

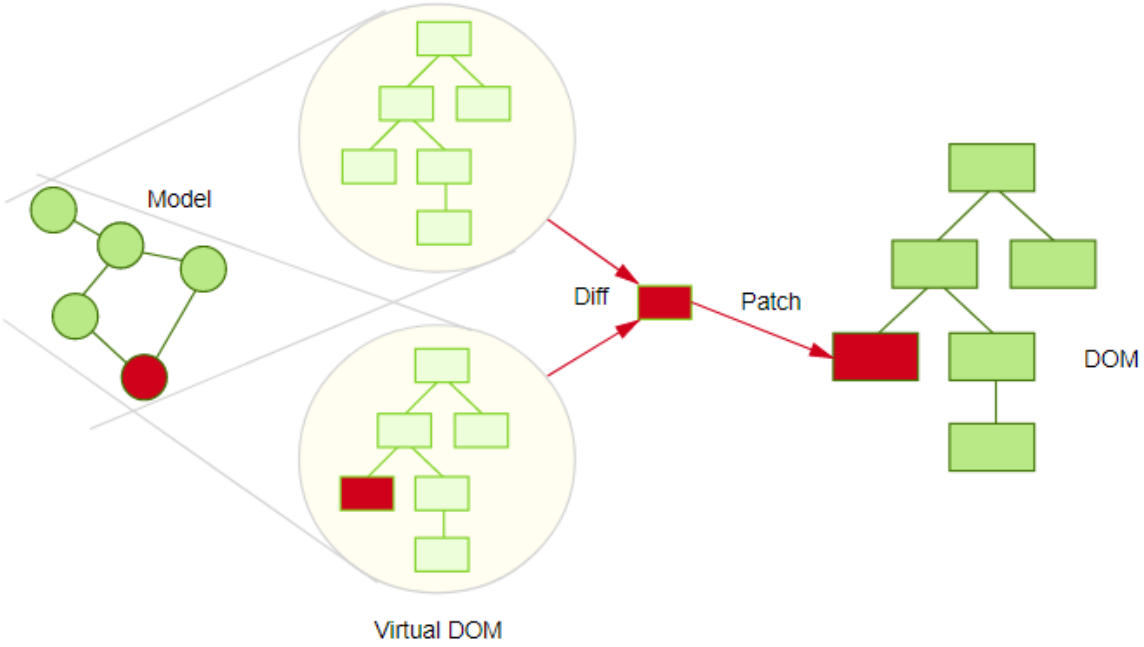
Iako komponente definirane pomoću klase pružaju više opcija, radi jednostavnosti i „*čišćeg*“ koda preporučuje se korištenje funkcijskih komponenti. Iz navedenog razloga React radi na novim značajkama, poput nedavno uvedenih *hook*-ova: *state hook* i *effect hook. State hook* omogućava stvaranje internog stanja unutar funkcijske komponente, dok *effect hook* omogućava *side effects* izvođenje unutar funkcijske komponente.

```
1   function Button (props) {
2     const {type, name} = props;
3     return <button type={type} name={name}>Click Me!</button>
4   }
5
6   class Button extends React.Component {
7     render() {
8       const {type, name} = this.props;
9       return <button type={type} name={name}>Click Me!</button>
10    }
11  }
```

Slika 6 Implementacijski primjer React komponente

JSX je ekstenzija sintakse JavaScripta koju je napravio Facebook. Ona omogućuje pisanje HTML-a unutar JavaScript koda (slika 6), a koristi se za opisivanje korisničkog sučelja. Datoteke pisane JSX sintaksom se moraju transformirati u izvorni JavaScript kod, a za transformaciju se preporučuje Babel kompajler.

Kako bi optimizirao manipulaciju DOM (eng. *Document Object Model*) elementima, React koristi virtualni DOM. Virtualni DOM je koncept zasnovan na ideji da se virtualna reprezentacija korisničkog sučelja čuva u memoriji te se sinkronizira s pravim DOM-om. Takav proces se naziva se pomirenje (eng. *reconciliation*). Glavne karakteristike virtualnog DOM-a su efikasnost i jednostavnost.



Slika 7 Dijagram React Virtualnog DOM-a, [15]

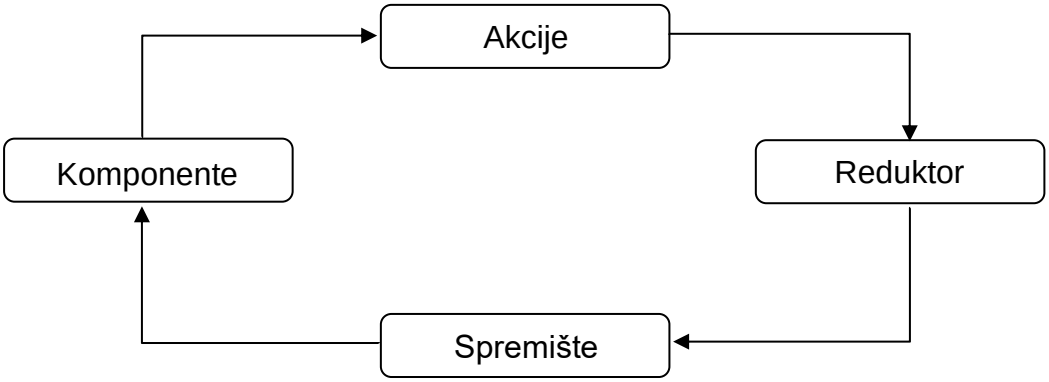
Kao što je vidljivo na slici 7, React koristi dvije instance virtualnog DOM-a, jedna instanca predstavlja ažurno stanje stabla, dok druga predstavlja prethodno stanje stabla. Te dvije instance se uspoređuju kako bi se dobio *diff*, odnosno kako bi se našle razlike između ažuriranih i prethodnih podataka. Pomoću *diff*-a, u DOM-u se ažuriraju samo elementi nad kojima su se dogodile promjene.

3.5. Redux

Redux je JavaScript biblioteka koja služi za skladištenje stanja (eng. *state*) web aplikacija. U današnje vrijeme većina web aplikacija manipulira velikom količinom podataka. Potreba za što jednostavnijim i boljim upravljanjem podataka odnosno stanja aplikacije bila je glavna motivacija za izradu Redxa. Prema [16] Redux se može opisati s tri temeljna principa:

- jedan izvor istine
- stanje se može samo čitati (eng. *read-only*)
- promjene stanja se ostvaruju čistim (eng. *pure*) funkcijama.

Na slici 8 prikazana je arhitektura Reduxa. Stanje cijele aplikacije, odnosno svi podaci spremljeni su u objektu koji se nalazi u spremištu (eng. *store*). Ovakav pristup omogućava da različite komponente koriste isti skup podataka. Podaci unutar skladišta, izravno se mogu samo čitati, njihova promjena moguća je jedino preko akcija (eng. *action*). Kako spremište može sadržavati veliki broj podataka, preporučuje se normalizacija istih kako se ne bi stvarale redundancije podataka.



Slika 8 Arhitektura Reduxa

Izvor: [16]

Akcije su jednostavni objekti koji opisuju promjenu koja se želi izvršiti nad stanjem. Slične su objektima događaja (eng. *event objects*). Svaka akcija mora sadržavati tip (eng. *type*), a opcionalni dio akcije su podaci (eng. *payload*) koji se žele proslijediti reduktoru (eng. *reducer*). Da bi se akcija izvršila mora se poslati reduktoru pomoću „*dispatch“* metode.

Reduktori su čiste funkcije koje se koriste za izmjenu podataka, odnosno stanja spremišta. Reduktori se okidaju svaki puta kada je poslana akcija pomoću „*dispatch“* metode. Primaju dva parametra, trenutno stanje i poslanu akciju. Na temelju podataka iz akcije reduktor kreira novo stanje koje se šalje spremištu. Svako spremište može imati samo jedan reduktor. Radi kompleksnosti današnjih aplikacija, i jednostavnijeg održavanja, Redux omogućava kreiranje više malih reduktora koji se ugrađenom metodom „*combineReducers“* spajaju u jedan.

Redux radi sinkrono što znači da asinkrone akcije nisu podržane u čistom Reduxu. Iz tog razloga razvijeno je više dodataka koji to omogućuju, a ovome radu će biti korišten Redux Thunk.

Kako bi se Redux mogao koristiti s Reactom, što mu je i razlog nastanka, razvijena je React Redux biblioteka. Biblioteka React-Redux služi za povezivanje Redux spremišta i React komponenti. Biblioteka omogućuje React komponentama čitanje podataka iz Redux spremišta i slanje (eng. *dispatch*) akcija u spremište kako bi se podaci ažurirali. Najjednostavnije povezivanje komponenti sa spremištem je pomoću *Provider* komponente i „*connect*“ metode koji su dio ove biblioteke. *Provider* komponenta ima jedan atribut koji mora biti postavljen, a to je „*store*“ koji bi trebao biti referenca na Redux spremište. Metoda „*connect*“ povezuje React komponente sa spremištem proslijeđenim *Provider* komponenti, [17].

3.6. NodeJS

NodeJs asinkrono je, događajima pokrenuto (eng. *event driven*) višeplatformsko radno okruženje, dizajnirano za izgradnju skalabilnih mrežnih aplikacija. Za izvršavanje JavaScript koda NodeJS koristi Chrome-ov V8 JavaScript stroj (eng. *engine*). Na slici 9 prikazan je „*Hello World*“ primjer, kod kojeg se više konekcija može izvršavati istovremeno. Unutar svake konekcije pokrenut je povratni poziv (eng. *callback*). Ako nema posla koji treba obaviti, NodeJS će biti u stanju mirovanja, [18].

```
1   const http = require('http');
2
3   const hostname = '127.0.0.1';
4   const port = 3000;
5
6   const server = http.createServer((req, res) => {
7     res.statusCode = 200;
8     res.setHeader('Content-Type', 'text/plain');
9     res.end('Hello World\n');
10  });
11
12  server.listen(port, hostname, () => {
13    console.log(`Server running at http://${hostname}:${port}/`);
14  });
```

Slika 9 NodeJS "Hello World" primjer, [18]

Vidljiva je suprotnost današnjim uobičajenim paralelnim modelima, kod kojih glavnu funkciju imaju dretve operacijskih sustava. Komunikacije zasnovane na dretvama su relativno ne efikasne i vrlo kompliciranе za koristiti. Osim toga, korisnici NodeJS-a oslobođeni su briga oko zagušenja (eng. *dead-locking*) procesa jer ne postoje zaključavanja (eng. *locks*). Gotovo nijedna funkcija u NodeJS-u ne izvršava izravno ulazno izlazne (eng. *Input/Output*) operacije i zato procesi nikada nisu blokirani.

3.6.1. NPM

NPM (eng. *Node Package Manager*) je najveći svjetski softverski registar. NPM registar je velika javna baza JavaScript paketa i njihovih meta informacija. Trenutno registar ima više od 800.000 paketa. NPM je vrlo koristan jer je programerima omogućeno dijeljenje koda, odnosno softverskih rješenja koja rješavaju određeni problem, [19].

3.6.2. Koa

Koa je web radni okvir (eng. *framework*), kojemu je cilj biti malen, ekspresivan i robustan temelj za izradu web aplikacija i API-a (eng. *Application Programming Interface*). Koristeći asinkrone funkcije, Koa omogućuje odbacivanje povratnih poziva (eng. *ditch callbacks*) i povećava efikasnost kontroliranja pogrešaka (eng. *error-handling*), [20].

3.7. Visual Studio Code

Visual Studio Code je lagan (eng. *lightweight*), ali moćan uređivač izvornih kodova. *Visual Studio Code* je aplikacija koja se izvodi lokalno kod korisnika i dostupna je za sljedeće operacijske sustave: Windows, macOS i Linux. Dolazi s ugrađenom podrškom za JavaScript, TypeScript i NodeJS i zato je jako popularan kod programera web aplikacija, [21].

3.8. JSON

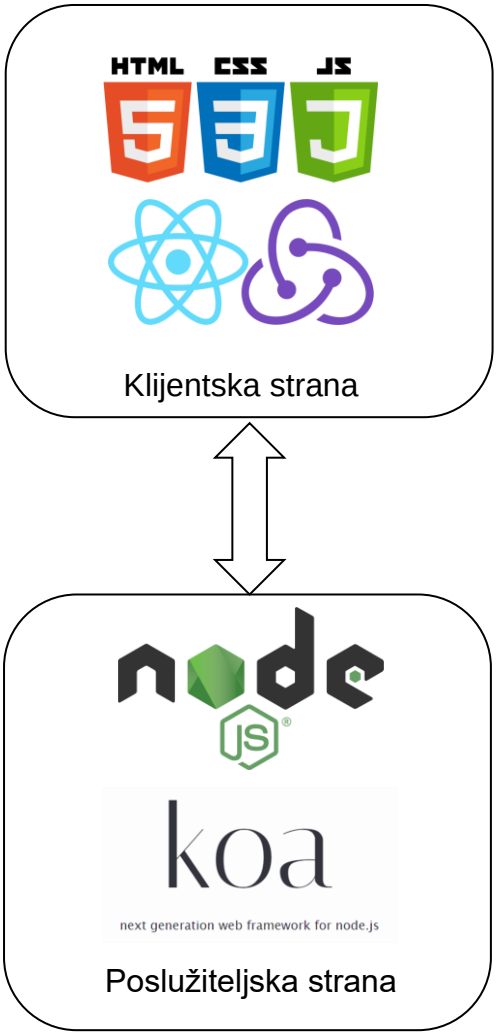
JSON (eng. *JavaScript Object Notation*) je jednostavni format za razmjenu podataka koji je ljudima razumljiv za čitanje i pisanje. Dizajniran je za jednostavno parsiranje, potpuno neovisan od bilo kojeg programskog jezika. Uz XML jedan je od najkorištenijih podatkovnih formata za razmjenu podataka. JSON se sastoji od dviju struktura: kolekcija parova ime/vrijednost i poredane liste vrijednosti, [22].

Slika 10 Primjer JSON zapisa

U JSON-u glavne forme su objekti, niz (eng. *array*), vrijednost (eng. *value*), niz povezanih znakova (eng. *string*) te brojevi (eng. *numbers*). Na slici 10 prikazan je primjer JSON zapisa.

4. ARHITEKTURA SUSTAVA

Model sustava web aplikacije za određivanje mjerila izvedbe poissonovskih i ne-poissonovskih sustava sastoji se od međusobne komunikacije dvije temeljne komponente: klijentske komponente i poslužiteljske komponente (slika 11).



Slika 11 Arhitektura sustava

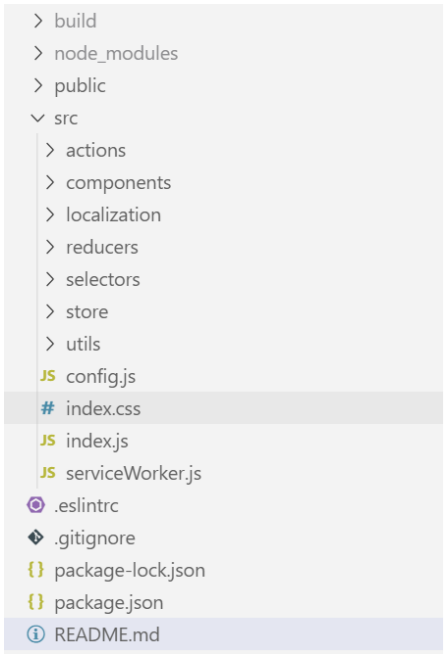
Glavnina logike poput računskih operacija i generiranja CSV (eng. *Comma-Separated Values*) datoteka nalazi se na poslužiteljskoj strani aplikacije. Klijentska strana je zadužena za postavljanje zahtjeva poslužitelju, odnosno traženja neke usluge od istog. Komunikacija između poslužiteljske i klijentske strane odvija se pomoću HTTP (eng. *Hypertext Transfer Protocol*) protokola.

4.1. Klijentska strana aplikacije

Klijentska strana aplikacije izrađena je pomoću ReactJS i Redux JavaScript biblioteka. Kako bi se omogućila višejezičnost klijentske strane aplikacije, korištena je biblioteka React Intl. To je biblioteka koja omogućuje korištenje gotovih React komponenti i API za oblikovanje datuma, brojeva, nizova znakova i rukovanje lokalizacijom, odnosno prijevodima, [23]. Za izradu klijentske strane aplikacije još su korištene biblioteke:

- Material-UI, koja pruža korištenje gotovih, stiliziranih komponenti
- Redux Form, koja olakšava kreiranje formi i validaciju polja
- Ky, biblioteka koja olakšava izradu asinkronih poziva prema poslužitelju.

Kako bi se što lakše i brže postavila okolina potrebna za razvoj aplikacije, korišten je alat Create React App. Pomoću Create React App alata stvoren je inicijalni projekt koji već sadrži najpotrebnije alate za daljnji razvoj aplikacije. Na slici 12 je prikazana struktura klijentske strane aplikacije.



Slika 12 Struktura klijentske strane aplikacije

Korijenska (eng. *root*) datoteka „*index.js*“ , je datoteka s kojom aplikacija započinje izvođenje. U korijenskoj datoteci se kreira korijenska React komponenta i uvozi (eng. *import*) spremište inicijalizirano u „*store*“ datoteci kako bi se moglo proslijediti *Provider* komponenti. Sadržaj korijenske datoteke prikazan je na slici 13.

```
1   import React from 'react';
2   import ReactDOM from 'react-dom';
3   import './index.css';
4   import App from './components/App/App';
5   import * as serviceWorker from './serviceWorker';
6   import store from './store/store';
7   import { Provider } from 'react-redux';
8
9   ReactDOM.render(
10    <Provider store={store}>
11      <App />
12    </Provider >,
13    document.getElementById('root')
14  );
```

Slika 13 Korijenska datoteka "index.js"

Komponenta *App* je kontejnerska (eng. *container*) komponenata od koje počinje iscrtavanje aplikacije (slika 14). Unutar navedene komponente smještene su sve ostale komponente aplikacije.

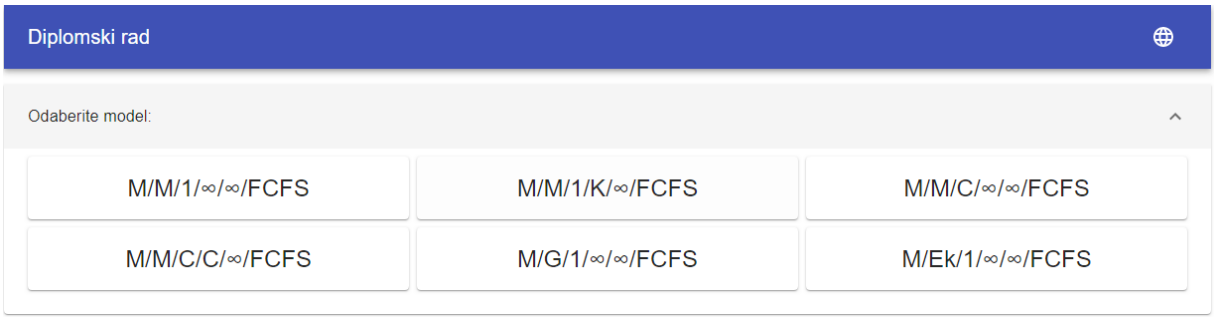
```
10  class App extends React.Component {
11    componentDidMount() {
12      this.props.fetchQueueingModels();
13    }
14
15    render() {
16      const { locale } = this.props;
17      return (
18        <IntlProvider
19          locale={locale}
20          messages={localization[locale]}
21        >
22          <Container>
23            <AppBar />
24            <Panels />
25          </Container>
26        </IntlProvider>
27      );
28    }
29  }
30
31  function mapStateToProps({ localization, queueingModels }) {
32    return {
33      locale: localization.locale,
34      queueingModelsFetched: queueingModels.fetched
35    };
36  }
37
38  const mapActionsToProps = {
39    fetchQueueingModels
40  };
41
42  export default connect(mapStateToProps, mapActionsToProps)(App);
```

Slika 14 Kontejnerska "App" komponenta

Inicijalno kada se aplikacija otvori u web pregledniku, korisničko sučelje sadrži samo alatnu traku (eng. *toolbar*) i panel naziva „Odaberite model“ (slika 15). Unutar alatne trake nalazi se naslov aplikacije i gumb koji otvara padajući izbornik za izbor jezika. Inicijalno jezik je postavljen na hrvatski, a podržan je i engleski. Unutar panela se nalazi *loader* koji označava asinkroni događaj unutar aplikacije.

Slika 15 Inicijalno stanje aplikacije

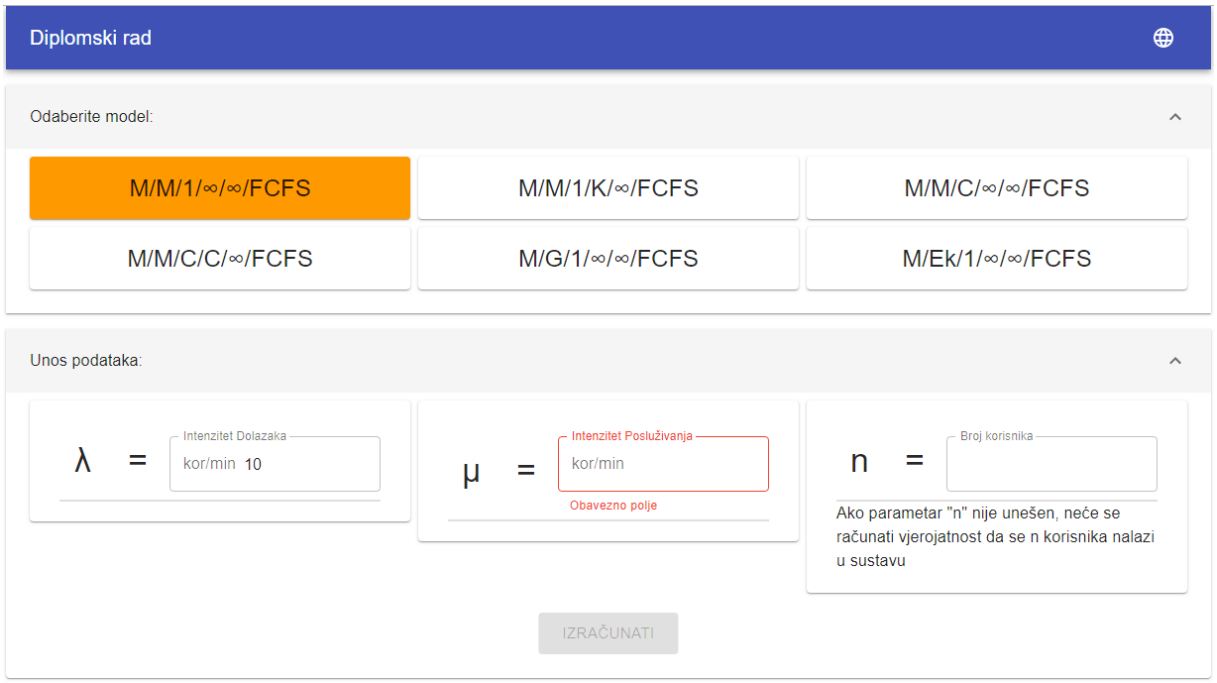
Kao što je vidljivo na slici 14, nakon što se aplikacija inicijalno iscrta, poziva se asinkrona akcija *fetchQueueingModels* koja dohvaća podatke o modelima podvorbenih sustava s poslužitelja. Kada se podaci dohvate, spremaju se u Redux spremište i komponente se ažuriraju. Nakon ažuriranja komponenti panel „Odaberite model“ izgleda kao na slici 16.



Slika 16 Panel za odabir modela

Unutar panela „Odaberite model“ nalazi se popis svih modela koje aplikacija podržava. Odabirom pojedinog modela otvara se novi panel koji se zove „Unos podataka“ (slika 17). Unutar tog panela nalazi se forma za unos podataka potrebnih za računanje izlaznih veličina. Formu sačinjavaju polja koja predstavljaju zadane veličine modela i gumb pomoću kojeg se radi zahtjev prema poslužitelju za računanje izlaznih veličina. Gumb je onemogućen (eng. *disabled*) sve dok sva polja ne prođu

zadane uvjete. Svakim novim unosom podataka, Redux forma radi njihovu validaciju, te za svako polje koje ne zadovoljava zadane uvijete ispisuje poruku greške.



Slika 17 Primjer forme za unos podataka za sustav M/M/1

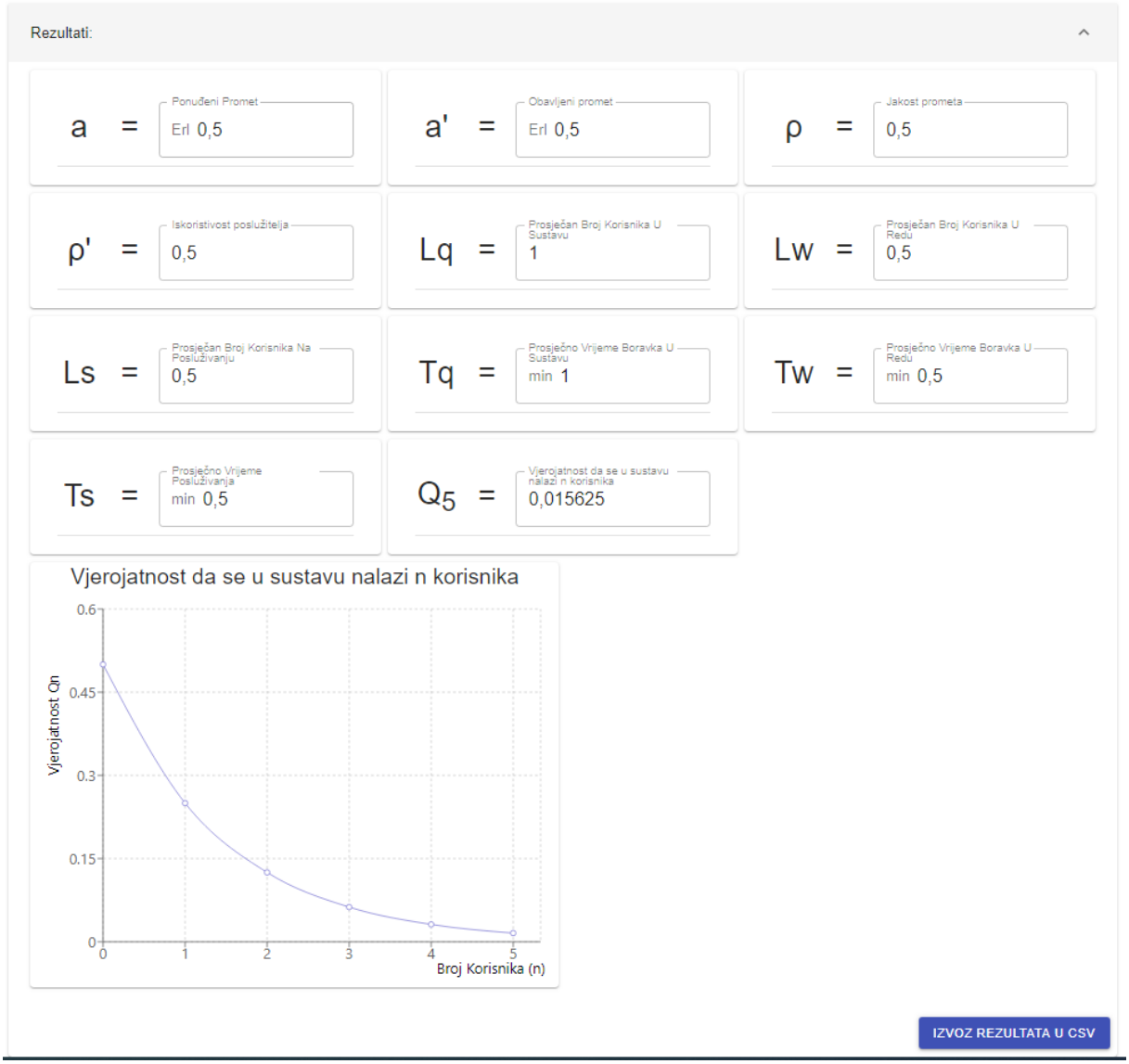
Gumb je omogućen (eng. *enabled*) kada sva polja zadovolje zadane uvjete. Pritiskom gumba, šalje se HTTP POST zahtjev (eng. *request*) prema poslužitelju. Unutar tijela (eng. *body*) zahtjeva nalaze se uneseni podaci korisnika u JSON formatu (slika 18).

```
{
  "arrivalRate": 10,
  "serviceRate": 20,
  "numberOfCustomers": 5
}
```

Slika 18 Primjer tijela zahtjeva

Kada poslužitelj primi zahtjev čita podatke iz njegovog tijela, računa izlazne veličine, te šalje odgovor (eng. *response*) klijentskoj aplikaciji. Unutar tijela odgovora nalaze se izračunate izlazne veličine u JSON formatu. Kada klijentska strana primi odgovor od poslužitelja, ažurira svoje spremište novim podacima i pokreće novo

ažuriranje komponenti. Dodavanjem podataka o izlaznim veličinama u spremište, iscrtava se novi panel naziva „Rezultati“. Unutar tog panela nalaze se izračunate izlazne veličine, što je vidljivo na slici 19.

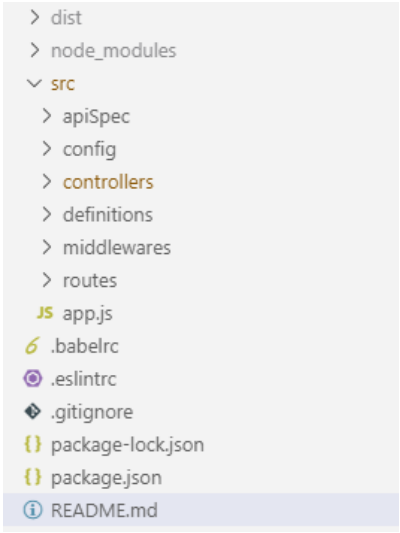


Slika 19 Primjer prikaza rezultata za model za sustav M/M/1

Izlazne veličine prikazane unutar panela „Rezultati“ mogu se izvesti u datoteku CSV formata, pritiskom na gumb koji se nalazi u donjem desnom kutu. Za kreiranje CSV podataka zadužen je poslužitelj.

4.2. Poslužiteljska strana aplikacije

Poslužiteljska strana aplikacije izrađena je u NodeJS tehnologiji koristeći Koa biblioteku. Za dokumentiranje API-a, i validaciju zahtjeva i odgovora korišten je Swagger alat. Za izradu aplikacije korištena je i biblioteka got koja služi za manipuliranje HTTP zahtjevima. Got biblioteka je potrebna zbog komuniciranja s Wolfram Alpha platformom koja je korištena za računanje integrala kod *M/G/1* modela. Struktura poslužiteljske strane aplikacije prikazana je na slici 20.



Slika 20 Struktura poslužiteljske strane aplikacije

Glavna datoteka „app.js“ zaslužena je za inicijalizaciju Koa aplikacije i postavljanja svih potrebnih dodataka. Sadržaj „*app.js*“ datoteke prikazan je na slici 21.

```
1  import Koa from 'koa';
2  import cors from '@koa/cors';
3  import router from './routes/router'
4  import bodyParser from 'koa-bodyparser';
5  import * as swagger from 'swagger2';
6  import { validate, ui } from 'swagger2-koa';
7  import apiSpec from './apiSpec';
8  // validate document
9  if (!swagger.validateDocument(apiSpec)) {
10   throw Error('./apiSpec.js does not conform to the Swagger 2.0 schema');
11 }
12
13 const app = new Koa();
14
15 app.use(cors());
16 app.use(bodyParser());
17 app.use(validate(apiSpec));
18 app.use(router.routes());
19 app.use(ui(apiSpec, '/api', ['/api/v1']));
20
21 app.listen('6060');
```

Slika 21 Glavna datoteka "app.js"

Unutar datoteke „router.js“ (slika 22) nalazi se inicijalizacija Koa Routera i definicija putanja. Koa router je dodatak Koa aplikaciji koji olakšava kreiranje putanja i upravljanje istima.

```
1   import Router from 'koa-router';
2   import * as Models from '../controllers/models';
3   import * as DataToCsv from '../controllers/dataToCsv';
4
5   const router = new Router({
6     prefix: '/api/v1'
7   });
8
9   router.get('/models', Models.getModels);
10
11  router.post('/models/metrics/:model', Models.metrics);
12
13  router.post('/exportDataToCsv', DataToCsv.dataToCsv);
14
15  export default router;
```

Slika 22 Datoteka "router.js"

Svaka definirana putanja popraćena je i Swagger specifikacijom. Swagger na temelju specifikacije radi validaciju zahtjeva i odgovora. U slučaju da za neku putanju nije napisana specifikacija ta putanja neće biti dostupna. Svaka putanja ima definiranu *middleware* funkciju koja se naziva kontroler (eng. *controller*). Kontroler se okida kada se pojavi zahtjev prema putanji. Na slici 23 je prikazana datoteka „models.js“ koja sadrži kontroler za dohvaćanje modela i kontroler za računanje izlaznih veličina podvorbenih sustava.

```
1  import modelsConfig from '../config/models';
2  import queueingModelsMetrics from './queueingModelMetrics';
3
4  export function getModels(ctx) {
5    ctx.body = modelsConfig;
6  }
7
8  async function calculateMetrics(model, params) {
9    return await queueingModelsMetrics[model.toLowerCase()](params);
10 }
11
12 export async function metrics(ctx) {
13   try {
14     const { model } = ctx.params;
15     const metrics = await calculateMetrics(model, ctx.request.body);
16     ctx.body = metrics;
17   } catch (error) {
18     console.error(error);
19     if (error.name === 'CalculationError'){
20       ctx.throw(400, error);
21     } else {
22       ctx.throw(500);
23     }
24   }
25 }
```

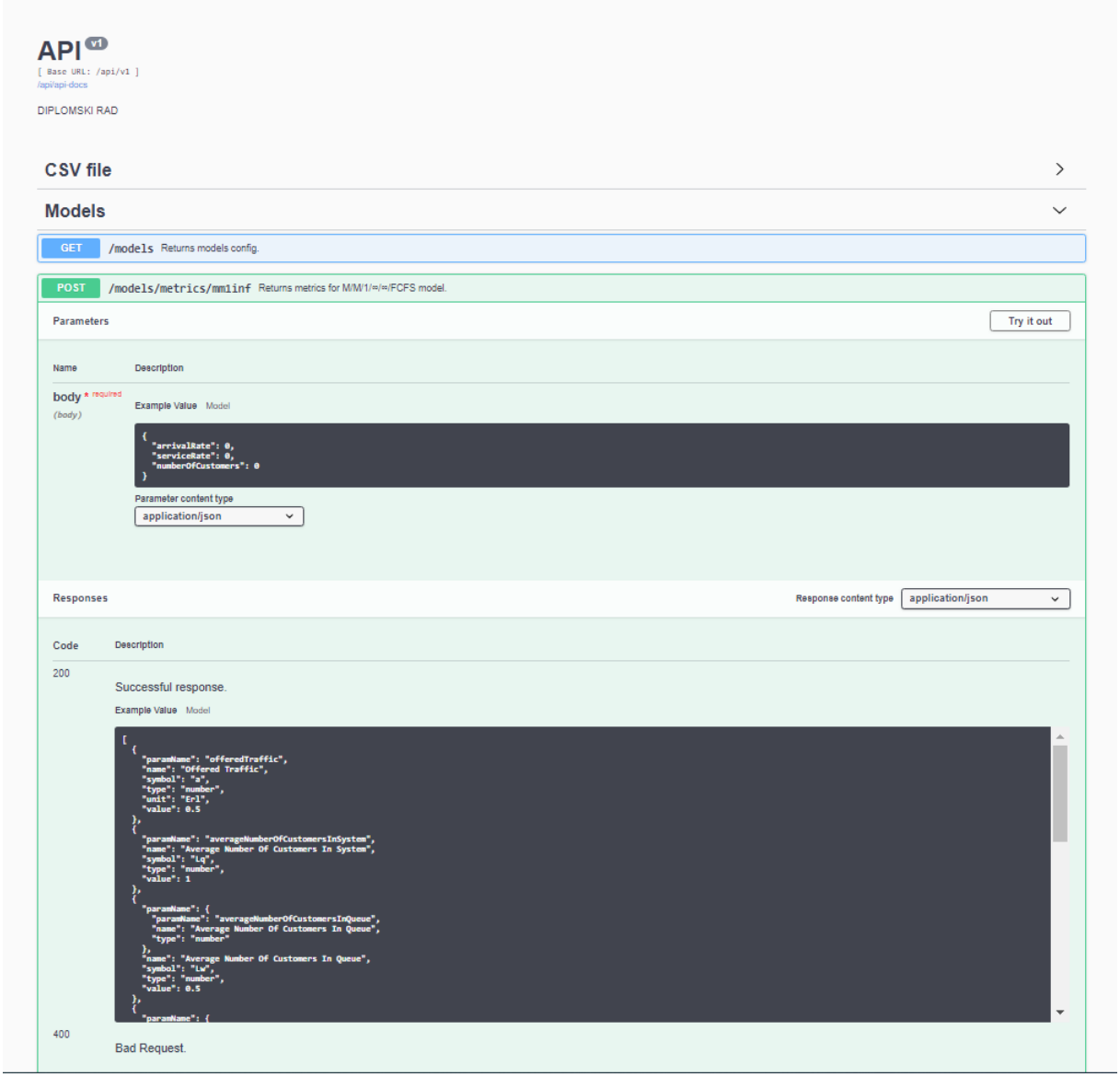
Slika 23 Datoteka „models.js“

Kada se pojavi zahtjev za računanje izlaznih veličina, okida se „*metrics*“ kontroler. Kontroler „*metrics*“ poziva pomoćnu funkciju „*calculateMetrics*“, koja poziva funkciju za računanje izlaznih veličina iz modula koji je definiran tipom modela. Za svaki model je kreiran jedan modul koji sadrži jednu glavnu i nekoliko pomoćnih funkcija za računanje izlaznih veličina. Primjer takvog modula dan je na slici 24.

```
import { calculateOfferedTraffic, calculateAverageServingTime, generateOutput, round, generateProbabilityNCustomersInSystemChartData} from './utils';

export default function calculatePerformanceMetrics(params) {

  const { arrivalRate, serviceRate, numberOfCustomers } = params;
  const offeredTraffic = calculateOfferedTraffic(arrivalRate, serviceRate);
  const averageNumberOfCustomersInSystem = calculateAverageNumberOfCustomersInSystem(arrivalRate, serviceRate);
  const averageNumberOfCustomersInQueue = calculateAverageNumberOfCustomersInQueue(offeredTraffic);
  const averageNumberOfCustomersOnServing = calculateAverageNumberOfCustomersOnServing(averageNumberOfCustomersInSystem, averageNumberOfCustomersInQueue);
  const averageTimeSpentInSystem = calculateAverageTimeSpentInSystem(arrivalRate, serviceRate);
  const averageServingTime = calculateAverageServingTime(offeredTraffic, arrivalRate);
  const averageTimeWaitingInQueue = calculateAverageTimeWaitingInQueue(averageTimeSpentInSystem, averageServingTime);

  const metrics = {
    offeredTraffic,
    carriedTraffic: offeredTraffic,
    trafficVolume: offeredTraffic,
    serverUtilization: offeredTraffic,
    averageNumberOfCustomersInSystem,
    averageNumberOfCustomersInQueue,
    averageNumberOfCustomersOnServing,
    averageTimeSpentInSystem,
    averageTimeWaitingInQueue,
    averageServingTime,
  };

  if (numberOfCustomers || numberOfCustomers === 0) {
    metrics.probabilityNCustomersInSystem = calculateProbabilityNCustomersInSystem(offeredTraffic, numberOfCustomers);
    const dataPoints = calculateProbabilityNCustomersInSystemDataPoints(offeredTraffic, numberOfCustomers);
    metrics.chartData = [generateProbabilityNCustomersInSystemChartData(dataPoints)];
  }

  return generateOutput(metrics);
}

function calculateAverageNumberOfCustomersInSystem(arrivalRate, serviceRate) {
  return arrivalRate / (serviceRate - arrivalRate);
}

function calculateAverageNumberOfCustomersInQueue(offeredTraffic) {
  return Math.pow(offeredTraffic, 2) / (1 - offeredTraffic);
}

function calculateAverageNumberOfCustomersOnServing(averageNumberOfCustomersInSystem, averageNumberOfCustomersInQueue) {
  return averageNumberOfCustomersInSystem - averageNumberOfCustomersInQueue;
}

function calculateAverageTimeSpentInSystem(arrivalRate, serviceRate) {
  return 1 / (serviceRate - arrivalRate);
}

function calculateAverageTimeWaitingInQueue(averageTimeSpentInSystem, averageServingTime) {
  return averageTimeSpentInSystem - averageServingTime;
}

function calculateProbabilityNCustomersInSystem(offeredTraffic, numberOfCustomers) {
  return Math.pow(offeredTraffic, numberOfCustomers) * (1 - offeredTraffic);
}

function calculateProbabilityNCustomersInSystemDataPoints(offeredTraffic, numberOfCustomers) {
  const values = [];

  for (let i = 0; i <= numberOfCustomers; i++) {
    const probability = round(calculateProbabilityNCustomersInSystem(offeredTraffic, i));
    values.push([i, probability]);
  }
  return values;
}
```

Slika 24 Modul za računanje izlaznih veličina za sustav M/M/1

Ako je računanje prošlo uspješno, kontroler kreira odgovor s tijelom u kojemu se nalaze izlazne veličine. U slučaju da se dogodila neka greška, kontroler kreira odgovor s neuspjelim statusom.



Slika 25 Swagger API dokumentacija

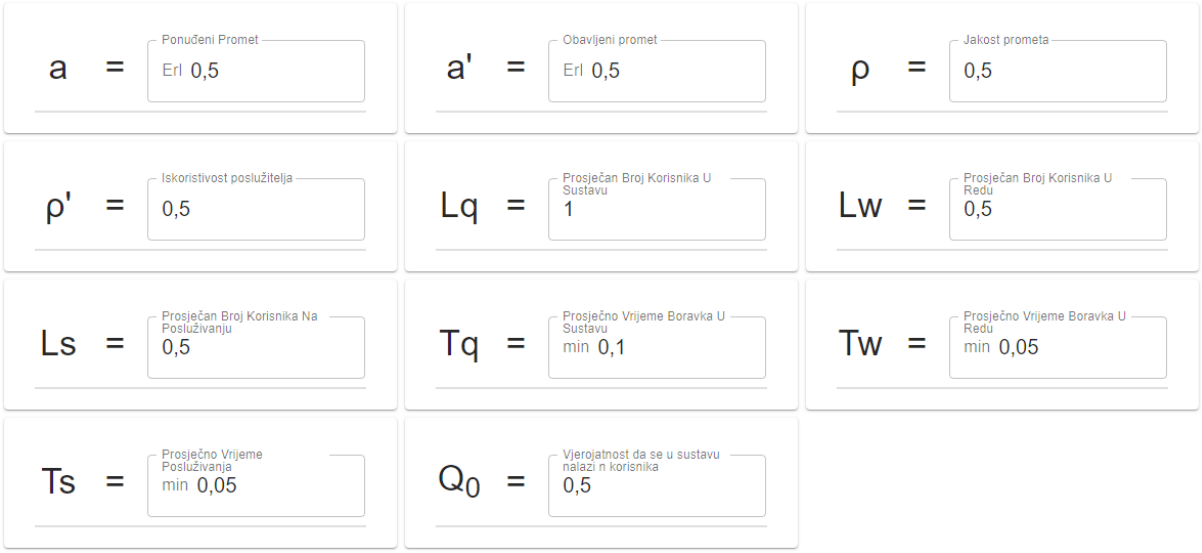
Na slici 25 je prikazano kako izgleda API dokumentacija generirana pomoću Swagger alata. Sve putanje za koje postoji specifikacija su i prikazane. Za svaku od njih mogu se vidjeti njezine definirane specifikacije poput: parametara, opisa uspješnog odgovora, opisa neuspješnog odgovara i slično. Osim opisa putanja, Swagger alat pruža i mogućnost kreiranja zahtjeva prema putanjama.

5. ODREĐIVANJE PERFORMANSI PODVORBENIH SUSTAVA

Za određivanje mjerila izvedbe pojedinog sustava posluživanja važan je ispravan odabir prometnog modela. Odabir se temelji na razdiobi međudolaznih vremena i razdiobi vremena posluživanja, ali i na kapacitetu poslužiteljskog mjesta, kapacitetu sustava i izvora kao i na disciplini posluživanja. Odabir ispravnog modela za opis pojedine podvorbene situacije ključan je korak za određivanje njegovih performansi. U osnovi redovi čekanja se pojavljuju kada je nekome ili nečemu potrebna neka usluga, a postoji ograničeni broj poslužitelja koji su zauzeti. Osnovni cilj upravljanja redovima čekanja je minimiziranje ukupnog troška. Ukupni trošak tvore troškovi kapaciteta i troškovi čekanja korisnika u redu. Cilj kod modeliranja sustava je naći optimalnu razinu poslužiteljskog kapaciteta. Optimalno rješenje ne znači da neće postojati redovi, jer to bi onda značilo da je sustav predimenzioniran, odnosno da je iskoristivost poslužitelja vrlo mala, a time i povećani troškovi kapaciteta.

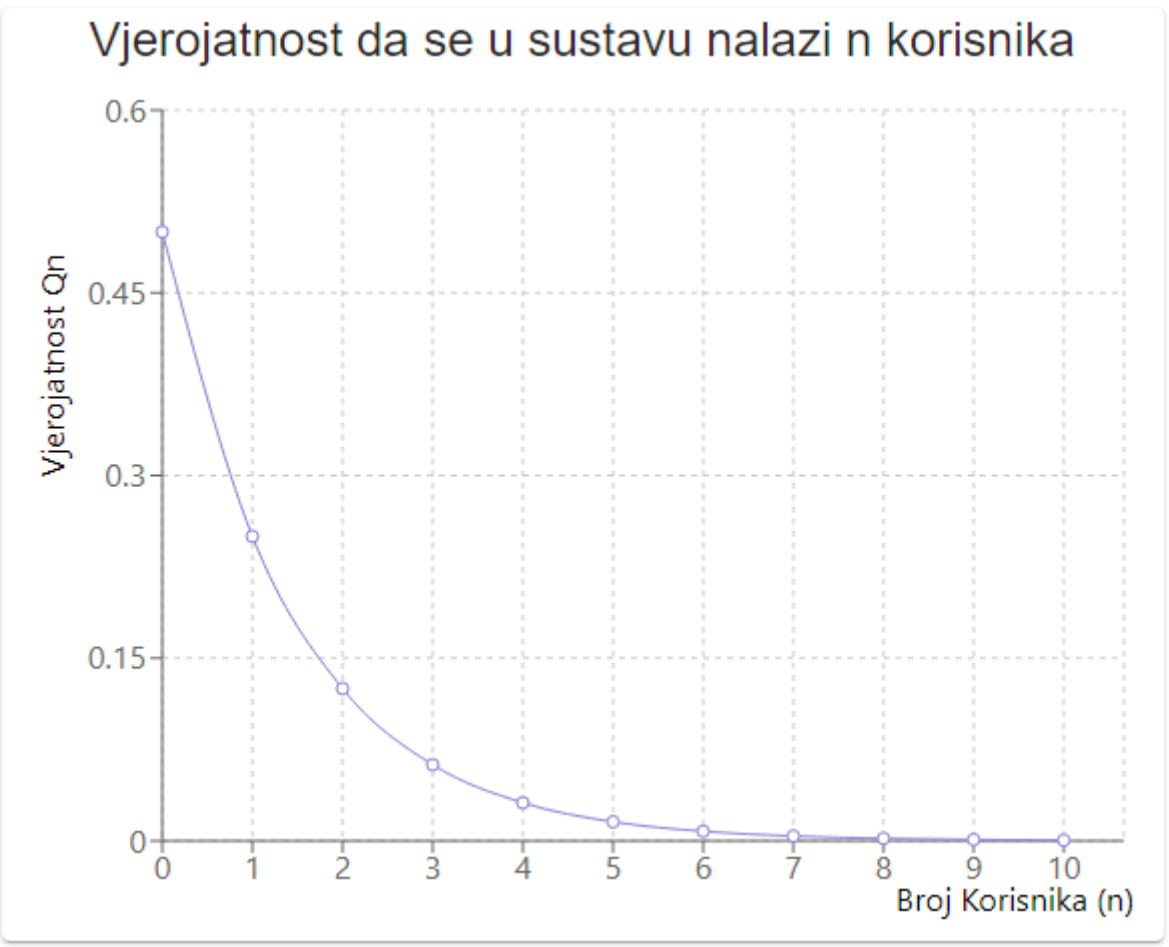
5.1. Određivanje performansi poissonovskih podvorbenih sustava

U ovom poglavlju razmatrat će se performanse različitih poissonovskih sustava za istu podvorbenu situaciju. Sustav *M/M/1/∞/∞/FCFS* je prvi promatrani sustav. Ako se pretpostavi podvorbena situacija u kojoj intenzitet dolazaka λ iznosi 10 korisnika po minuti, a intenzitet posluživanja μ iznosi 20 korisnika po minuti dobiju se rezultati mjerila izvedbe prikazani na slici 26.



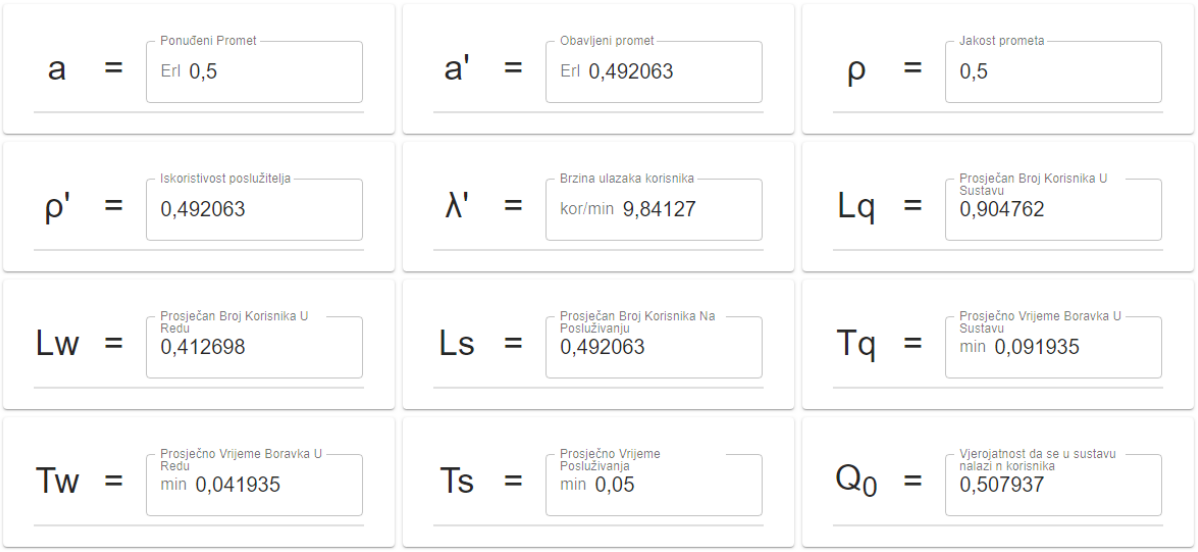
Slika 26 Mjerila izvedbe za sustav M/M/1 za ponuđeni promet od 0.5 Erl

Iz ovih rezultata je vidljivo je da je iskoristivost poslužitelja jednaka ponuđenom prometu jer je ovo sustav bez gubitaka i iznosi 50%. Prosječno vrijeme čekanja je relativno malo i iznosi 0,005 minuta, odnosno 3 sekunde. Vjerojatnost da u sustavu nema korisnika također iznosi 50%, što je ujedno i vjerojatnost da će korisnik pri ulasku u sustav naići na slobodan poslužitelj. Na slici 27 prikazan je graf s vjerojatnostima da se u sustavu nalazi od nula do deset korisnika.



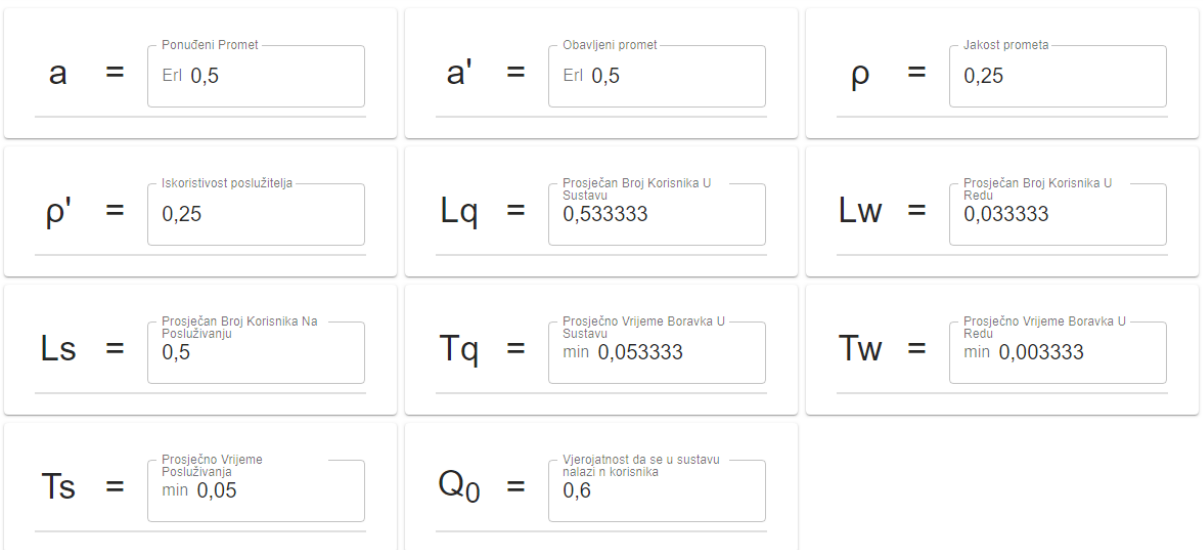
Slika 27 Vjerojatnost da se u sustavu M/M/1 nalazi od nula do deset korisnika

Sljedeći promatrani sustav je *M/M/1/5/∞/FCFS,* koji se od prethodnog razlikuje ograničenim kapacitetom sustava koji je u ovom primjeru ograničen na 5 korisnika. Za ovaj model dobiveni su rezultati prikazani na slici 28.



Slika 28 Mjerila izvedbe za sustav M/M/1/5 za ponuđeni promet od 0.5 Erl

Kod ovog sustava iskoristivost poslužitelja iznosi 41,27%, što je malo manje nego kod sustava *M/M/1*. Broj korisnika koji su ostvarili podvorbu je manji od intenziteta dolazaka jer je ovo sustav s gubicima. Sukladno tome može se vidjeti i da je ostvareni promet manji od ponuđenog, i da je iskoristivost poslužitelja manja od jakosti prometa. Prosječno vrijeme čekanja i prosječno vrijeme boravka u sustavu je nešto manje nego kod sustava *M/M/1*, dok je vrijeme posluživanja jednako.



Slika 29 Mjerila izvedbe za sustav M/M/2 za ponuđeni promet od 0.5Erl

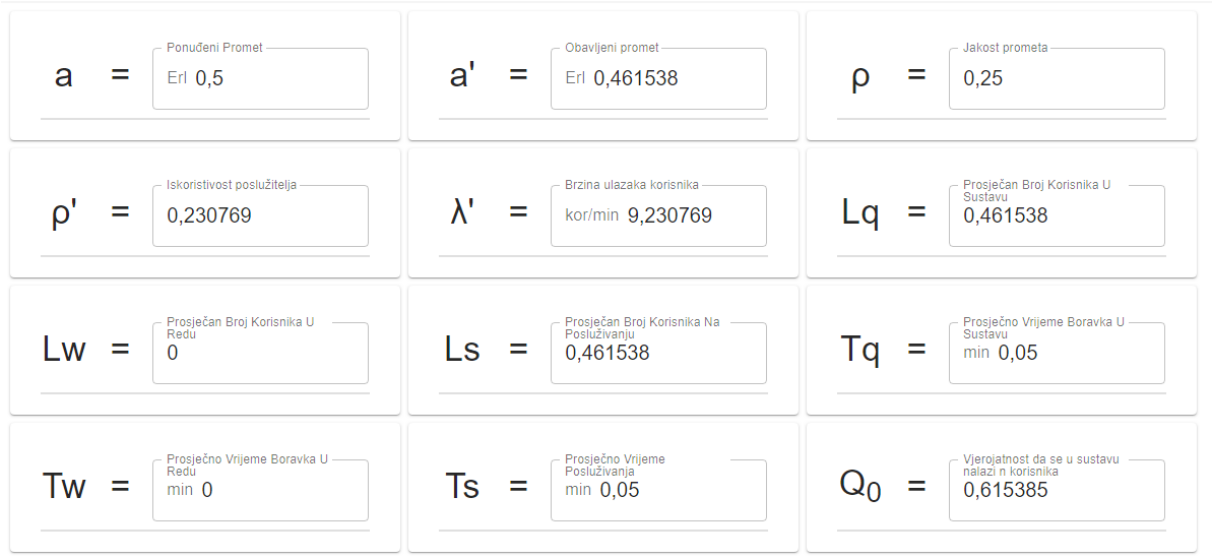
Na slici 29 vide se mjerila izvedbe dobivena za sustav *M/M/2/∞/∞/FCFS*. Ovaj sustav je višeposlužiteljski te se po tome i razlikuje od prethodna dva sustava. Model

ima neograničen kapacitet sustava što znači da kao i kod prvog sustava nema gubitaka. To je i vidljivo na prikazanim rezultatima: ostvareni promet je jednak ponuđenom, a iskoristivost poslužitelja jakosti prometa. Iskoristivost je 25% što je upola manje nego kod modela *M/M/1,* što je i logično jer je i broj poslužitelja dvostruko veći. Prosječno vrijeme posluživanja je jednako kao i kod *M/M/1* modela.

Slika 30 Vjerojatnost da se u sustavu M/M/2 nalazi od nula do deset korisnika

Kao što je vidljivo na slici 30, graf koji prikazuje vjerojatnost da se u sustavu nalazi *n* korisnika je strmiji nego kod prethodna dva modela, što je rezultat većeg broja poslužitelja. Vjerojatnost da se u sustavu nema korisnika iznosi 60%.

Zadnje promatrani poissonovski model je *M/M/2/2* koji se od prethodnih razlikuje što ima ograničen kapacitet sustava i broj poslužitelja. Kapacitet sustava jednak je broju poslužitelja i ograničen je na 2 korisnika. Ovaj sustav je temeljni sustav s gubitcima.



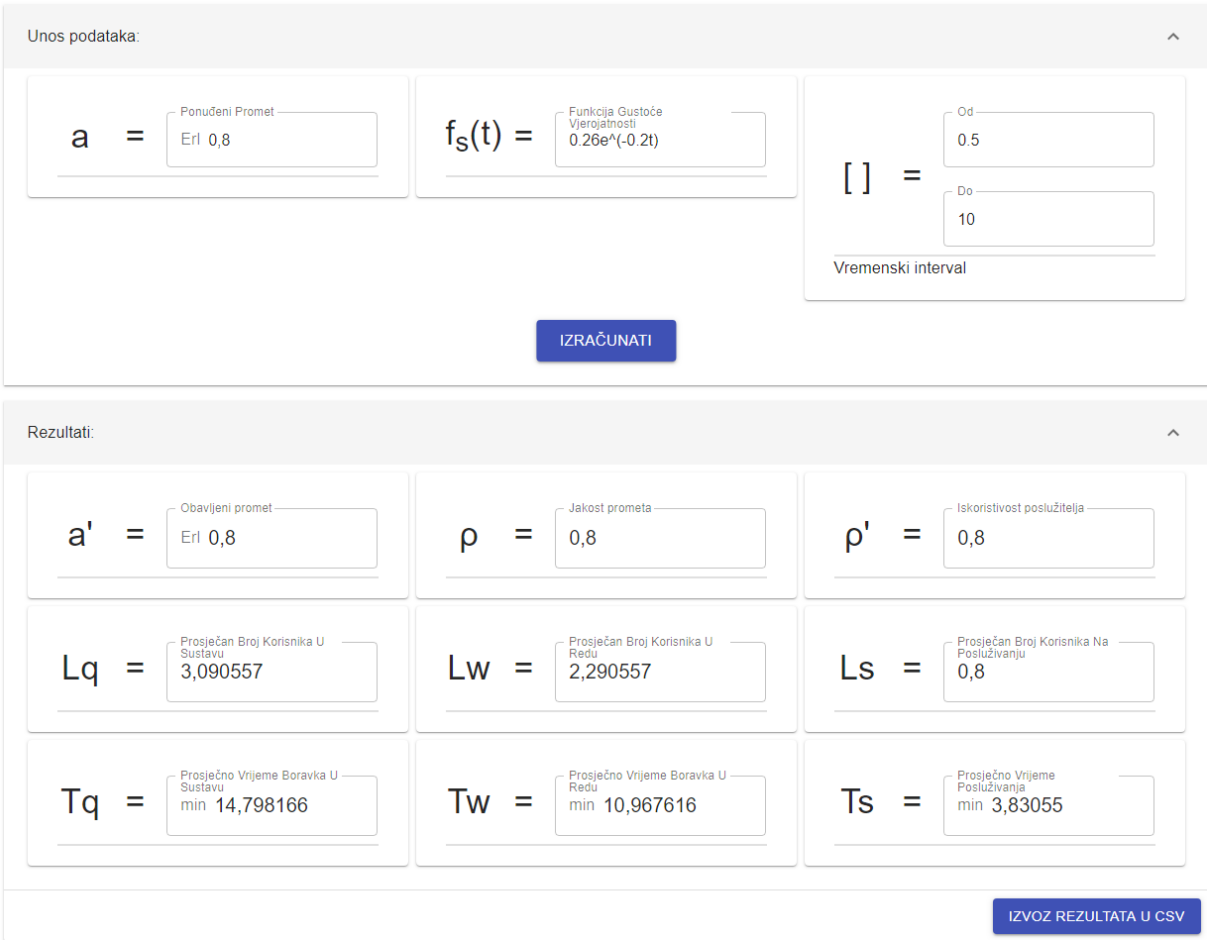
Slika 31 Mjerila izvedbe za sustav M/M/2/2 za ponuđeni promet od 0.5Erl

Na slici 31 prikazani su rezultati dobiveni za ovaj model. Kod ovog modela ne postoji red te iz tog razloga prosječan broj korisnika u redu i prosječno vrijeme čekanja u redu su jednaki nuli. Iskoristivost poslužitelja je mala i iznosi svega 23,08%.

Za zadanu podvorbenu situaciju najpogodniji model je *M/M/1*, jer ima najveću iskoristivost, nema gubitaka i u prosjeku samo jedan korisnik čeka u redu, s prosjekom čekanja od 3 sekunde.

5.2. Određivanje performansi ne-poissonovskih podvorbenih sustava

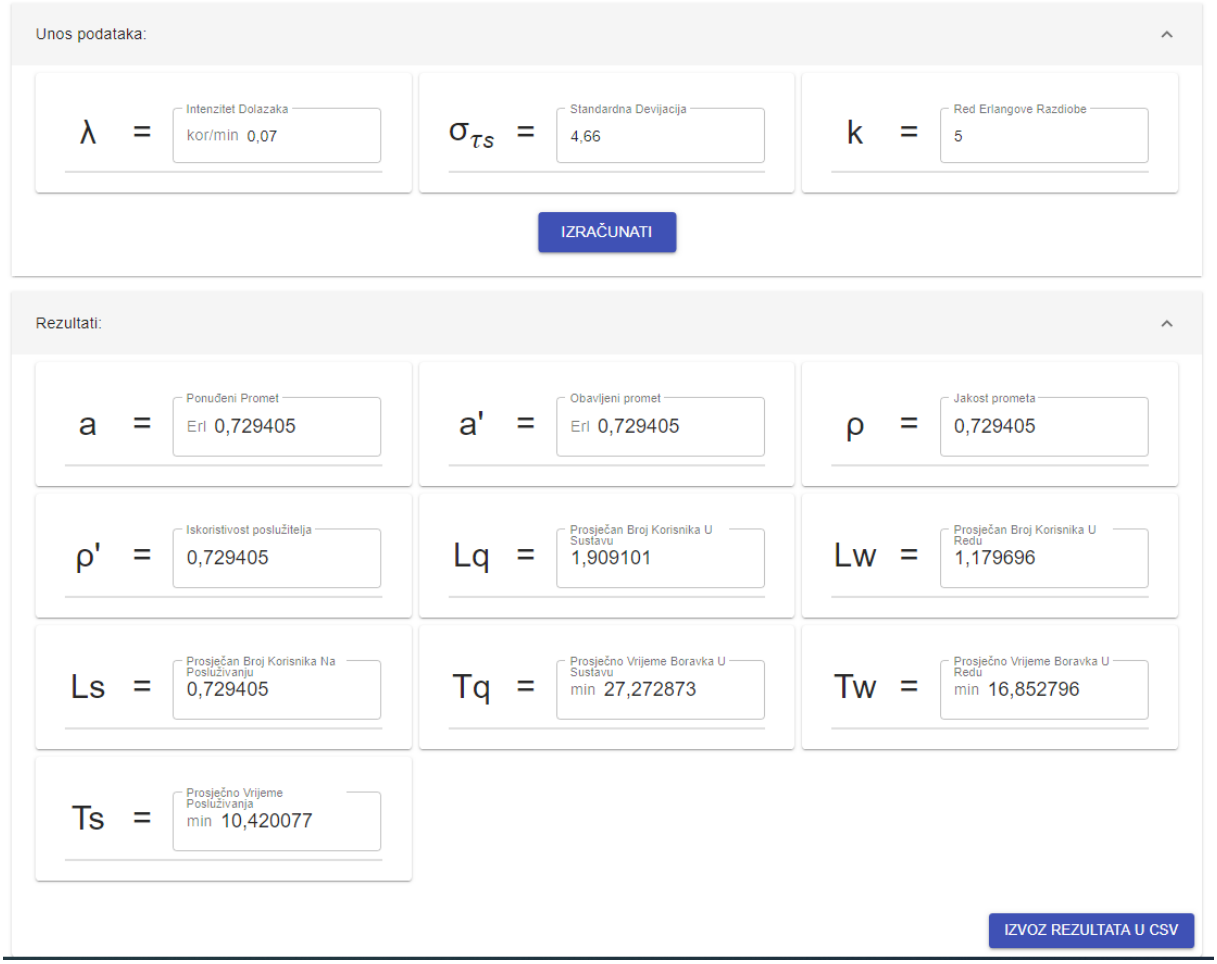
Aplikacija podržava računanje mjerila izvedbe za dva ne-poissonovka sustava a to su: *M/G/1* i *M/E$_k$/1* sustavi. Kako bi se izračunala mjerila izvedbe za sustav *M/G/1* putem aplikacije potrebno je poznavati veličinu ponuđenog prometa i funkciju gustoće vjerojatnosti vremena posluživanja, što je vidljivo na slici 32.



Slika 32 Mjerila izvedbe za sustav M/G/1

U danom primjeru izračunata su mjerila izvedbe za ponuđeni promet od 0,8 Erl i funkciju gustoće vjerojatnosti vremena posluživanja $f_s(t) = 0.26 \cdot e^{-0.2t}$ $za\ t \in [0.5, 10] min$. Kako je ovaj sustav bez gubitaka, iskoristivost poslužitelja jednaka je jakosti promet, a jakost prometa jednaka je ponuđenom prometu. Iskoristivost poslužitelja ovakvog modela je veća nego u prethodno analiziranim sustavima i iznosi 80%.

Kako bi se mogla izračunati mjerila izvedbe za sustav *M/Eₖ/1* potrebno je poznavati vrijednosti intenziteta dolazaka, standardne devijacije i parametar oblika koji označava red Erlangove razdiobe. Primjer računanja mjerila izvedbe ovakvog sustava putem aplikacije prikazan je na slici 33.



Slika 33 Mjerila izvedbe za sustav M/Eₖ/1

Mjerila izvedbe ovog sustava izračunata su za intenzitet dolazaka od 0,07 korisnika po minuti, standardnu devijaciju čija je vrijednost 4,66 i peti red Erlangove razdiobe. Iskoristivost poslužitelja je jednaka prosječnom broju korisnika na posluživanju koji je jednak ponuđenom prometu, i iznosi 72,94%.

6. ZAKLJUČAK

Ovim je radom dan sažeti pregled poissonovskih i ne-poissonovskih podvorbenih sustava. Podvorbeni sustavi su sustavi sačinjeni od korisnika, reda, poslužitelja i pravila po kojima se poslužuju dolazeći korisnici. Podvorbeni sustavi se nazivaju modelima kada predstavljaju stvarnu podvorbenu situaciju. Podvorbene situacije se nalaze svuda oko nas, u banci, poštanskom uredu, prometu na cestama itd.

Redovi nastaju kada korisnici čekaju podvorbu. Čekanje je rezultat ograničenog broja poslužitelja i ima negativan učinak na ljude, poslovanja, troškove i slično. Kako bi se ti negativni učinci ublažili potrebno je modelirati podvorbu i čekanje pomoću podvorbenih sustava.

Kod planiranja podvorbenih modela bitno je odabrati odgovarajući podvorbeni sustav. Svaki takav sustav ima određena svojstva koja se definiraju Kendallovom oznakom. U radu su opisani određeni podvorbeni modeli te njihove karakteristike pomoću kojih je izrađena web aplikacija za određivanje mjerila izvedbe. Pravilan odabir tehnologija i koncizna definicija korisničkog sučelja bili su poseban izazov kod izrade web aplikacije. Također, tijekom procesa planiranja uočen je problem implementacije integrala te je isti riješen uz korištenje Wolfram Alpha platforme. Aplikacija je podijeljena na klijentsku i poslužiteljsku stranu. U radu je prikazana implementacija klijentske strane koja koristi API definiran na poslužiteljskoj strani. Spomenuti API omogućuje lakšu integraciju budućih projekata s postojećom poslužiteljskom stranom.

Proces planiranja i kapacitiranja podvorbenog sustava zahtjevan je posao i iziskuje određeno vrijeme. Isti proces može se pojednostaviti i ubrzati koristeći izrađenu web aplikaciju što je bio i cilj ovog diplomskog rada.

LITERATURA

[1] Begović, M.: *Podvorbeni sustavi,* Fakultet prometnih znanosti, Zagreb, 2006.

[2] Sztrik, J.: *Basic Queueing Theory,* Globe, 2016.

[3] Schwartz, B.: *The Essential Guide to Queueing Theroy,* VividCortex, 2016.

[4] Mrvelj, Š., Matulin, M.: *Autorizirana predavanja iz kolegija Podvorbeni sustav*, Fakultet prometnih znanosti, Zagreb, 2017.

[5] Kleinrock, L.: *Queueing Systems, Volume 1: Theory*, Wiley-Interscience, 1975.

[6] URL: *https://developer.mozilla.org/en-US/docs/Web/JavaScript (pristupljeno: kolovoz*, 2019.)

[7] URL: *https://medium.com/@benastontweet/lesson-1a-the-history-of-javascript-8c1ce3bffb17* (pristupljeno: kolovoz, 2019.)

[8] Šimec, A.: *Autorizirana predavanja iz kolegija Oblikovanje web stranica*. TVZ, Zagreb, 2011.

[9] URL: *https://www.mojwebdizajn.net/skriptni-jezici/vodic/html/html-elementi.aspx* (pristupljeno: kolovoz, 2019.)

[10] URL*: www.w3schools.com* (pristupljeno: kolovoz, 2019.)

[11] URL: *http://www.webtech.com.hr/html5.php* (pristupljeno: kolovoz, 2019.)

[12] URL: *https://slideplayer.com/slide/14037399* (pristupljeno: kolovoz, 2019.)

[13] URL: *https://reactjs.org* (pristupljeno: kolovoz, 2019.)

[14] URL: *http://projects.wojtekmaj.pl/react-lifecycle-methods-diagram* (pristupljeno: kolovoz, 2019.)

[15] URL: *http://teropa.info/blog/2015/03/02/change-and-its-detection-in-javascript-frameworks.html* (pristupljeno: kolovoz, 2019.)

[16] URL: *https://redux.js.org* (pristupljeno: kolovoz, 2019.)

[17] URL: *https://react-redux.js.org/* (pristupljeno: kolovoz, 2019.)

[18] URL: *https://nodejs.org* (pristupljeno: kolovoz, 2019.)

[19] URL: *https://www.npmjs.com* (pristupljeno: kolovoz, 2019.)

[20] URL: *https://koajs.com* (pristupljeno: kolovoz, 2019.)

[21] URL: *https://code.visualstudio.com* (pristupljeno: kolovoz, 2019.)

[22] URL: *https://www.json.org* (pristupljeno: kolovoz, 2019.)

[23] URL: *https://github.com/formatjs/react-intl* (pristupljeno: kolovoz, 2019.)

POPIS SLIKA

Sveučilište u Zagrebu
Fakultet prometnih znanosti
10000 Zagreb
Vukelićeva 4

IZJAVA O AKADEMSKOJ ČESTITOSTI I SUGLASNOST

Izjavljujem i svojim potpisom potvrđujem kako je ovaj diplomski rad isključivo rezultat mog vlastitog rada koji se temelji na mojim istraživanjima i oslanja se na objavljenu literaturu što pokazuju korištene bilješke i bibliografija.

Izjavljujem kako nijedan dio rada nije napisan na nedozvoljen način, niti je prepisan iz necitiranog rada, te nijedan dio rada ne krši bilo čija autorska prava.

Izjavljujem također, kako nijedan dio rada nije iskorišten za bilo koji drugi rad u bilo kojoj drugoj visokoškolskoj, znanstvenoj ili obrazovnoj ustanovi.

Svojim potpisom potvrđujem i dajem suglasnost za javnu objavu diplomskog rada pod naslovom **Određivanje mjerila izvedbe poissonovskih i ne-poissonovskih podvorbenih sustava** na internetskim stranicama i repozitoriju Fakulteta prometnih znanosti, Digitalnom akademskom repozitoriju (DAR) pri Nacionalnoj i sveučilišnoj knjižnici u Zagrebu.

Student/ica:

U Zagrebu, 9.9.2019 Tomislav Hrsak
(potpis)