Iscrtavanje na strani poslužitelja

Tomas, Karlo

Sveučilište u Splitu
Prirodoslovno-matematički fakultet
Odjel za informatiku

Karlo Tomas

ISCRTAVANJE NA STRANI POSLUŽITELJA

Završni rad

Split, 2023.

Temeljna dokumentacijska kartica

Završni rad

Sveučilište u Splitu
Prirodoslovno-matematički fakultet
Odjel za informatiku
Ruđera Boškovića 33, 21000 Split, Hrvatska

ISCRTAVANJE NA STRANI POSLUŽITELJA

Karlo Tomas

SAŽETAK

Rad se bavi analizom pristupa razvoju web aplikacija koji je opisan izrazom iscrtavanje na strani poslužitelja (SSR). Prvobitno se upoznaje sa HTTP protokolom i arhitekturom web aplikacije te sa razlikama između tradicionalne višestranične i jednostranične aplikacije. Nakon navedenog slijedi analiza različitih pristupa iscrtavanju. Iscrtavanje na strani poslužitelja nudi mogućnost optimizacije brzine učitavanja stranica, pruža bolje iskustvo korisnicima sa sporijim uređajima ili ograničenom snagom procesiranja i održava dosljednost u različitim preglednicima tako što se web stranice iscrtavaju na poslužitelju prije nego što se isporuče klijentu.

Ključne riječi: iscrtavanje, poslužitelj, SSR, web aplikacija, web razvoj
Rad je pohranjen u knjižnici Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu

Rad sadrži: 34 stranice, 14 grafičkih prikaza, 0 tablica i 5 literaturnih navoda. Izvornik je na hrvatskom jeziku.

Mentor: **Doc. dr. sc. Goran Zaharija,** *docent Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu*

Neposredni voditelj:
Doc. dr. sc. Goran Zaharija, *docent Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu*

Ocjenjivači: **Doc. dr. sc. Goran Zaharija,** *docent Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu*
Doc. dr. sc. Monika Mladenović, *docent Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu*
Mirna Marić, *asistent Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu*

Rad prihvaćen: **rujan 2023.**

Basic documentation card

Bachelor's thesis

University of Split
Faculty of Science
Department of informatics
Ruđera Boškovića 33, 21000 Split, Croatia

SERVER SIDE RENDERING

Karlo Tomas

ABSTRACT

The paper deals with the analysis of the approach to web application development described by the term server-side rendering (SSR). First there is an introduction to the HTTP protocol and web application architecture and to the differences between a traditional multi-page and a single-page application. The aforementioned is followed by an analysis of different rendering approaches. Serve-side rendering offers the ability to optimize page load speed, provides a better experience for users with slower devices or limited processing power and maintains consistency across browsers by rendering web pages on the server before they are delivered to the client.

Key words: rendering, server, SSR, web application, web development
Thesis deposited in library of Faculty of science, University of Split

Thesis consists of: 34 pages, 14 figures, 0 tables and 5 references
Original language: Croatian

Mentor: **Goran Zaharija, Ph.D.** *Assistant Professor / Associate Professor / Professor of Faculty of Science, University of Split*

Supervisor: **Goran Zaharija, Ph.D.** *Assistant Professor / Associate Professor / Professor of Faculty of Science, University of Split*

Reviewers: **Goran Zaharija, Ph.D.** *Assistant Professor of Faculty of Science, University of Split*
Monika Mladenović, Ph.D. *Assistant Professor of Faculty of Science, University of Split*
Mirna Marić, *Instructor of Faculty of Science, University of Split*

Thesis accepted: **September 2023**

SADRŽAJ

1. UVOD

Iscrtavanje je proces generiranja slike pomoću računala koji je moguć unutar web aplikacije. U ranim fazama World Wide Weba (WWW), oko 2000. godine, web stranice su već bile prikazivane korištenjem iscrtavanja na strani poslužitelja. Evolucija web razvoja može se objasniti poviješću web stranice. Od 90-ih do ranih 00-ih, internet se sastojao od onoga što se naziva Web 1.0. Web 1.0 bile su rane faze web stranica koje su više-manje izgledale poput postera gotovo bez ikakve interaktivnosti. Web stranice su tada bile ograničene samo na čitanje i korisnik nije mogao prenijeti nikakve informacije natrag na web-mjesto. Zatim je stigao Web 2.0 koji još uvijek postoji. Vrhunac Weba 2.0 bio je između 2000. i 2010. i naziva se društvenim webom ili čitaj-piši webom čije su karakteristike omogućile korisniku veću interakciju. Trenutno se nalazimo u eri Weba 3.0, koji se također naziva semantičkim webom ili čitaj-piši-izvrši web gdje se pojavljuje sve veća potreba da računala razumiju korisnike i njihove potrebe.

Glavna razlika između iscrtavanja na strani poslužitelja u prošlosti i u svijetu današnjice je interaktivnost. U Web 1.0 ili eri samo za čitanje, web stranice su bile statične što znači da su se mogle prikazati na poslužitelju da bi se kasnije prikazale. Interaktivnost web stranica dodaje sloj složenosti. Interaktivnost se mora dodati nakon što se statična web stranica prikaže na poslužitelju.

Ovaj rad započinje odgovaranjem na pitanje što je to web aplikacija i proučava njihovu strukturu odnosno arhitekturu. Navedeno je u biti uvertira za postizanje glavnog cilja, a to je kvalitetno upoznavanje s pristupima iscrtavanja na strani klijenta i, još bitnije i detaljnije, iscrtavanja na strani poslužitelja (SSR – server side rendering) te istraživanje funkcionalnosti i alata koji idu uz iste.

2. RAZRADA TEME

2.1. HTTP protokol

Internetski protokol je skup pravila koja definiraju kako uređaji na internetskoj mreži komuniciraju. HTTP (Hypertext Transfer Protocol) je protokol koji se koristi za razmjenu informacija putem interneta i u suštini je temelj World Wide Weba. To je način na koji web-preglednici i poslužitelji međusobno komuniciraju i šalju podatke poput web stranica jedan prema drugom. Za programere koji izrađuju web aplikacije je krucijalno da znaju kako funkcionira.

HTTP je "stateless" sustav zahtjeva/odgovora što znači da su zahtjevi odvojeni jedni od drugih. Veza između klijenta i poslužitelja održava se samo za trenutni zahtjev, a nakon toga se veza zatvara. Nakon što HTTP klijent uspostavi TCP vezu s poslužiteljem i pošalje mu poruku zahtjeva, poslužitelj šalje natrag svoj odgovor i zatvara vezu.

Komunikacija u HTTP-u temelji se na konceptu koji se zove ciklus zahtjeva i odgovora. Ciklus zahtjev-odgovor proces je kojim klijent komunicira s poslužiteljem kako bi dohvatio resurse ili izvršio radnje. Ciklus uključuje nekoliko koraka:

Klijent inicira zahtjev poslužitelju slanjem poruke HTTP zahtjeva, koja sadrži informacije kao što su traženi resurs i dodatni parametri.

Poslužitelj prima poruku zahtjeva i obrađuje ju te generira poruku odgovora koristeći svoje resurse.

Poslužitelj šalje natrag klijentu poruku odgovora koja obično sadrži traženi resurs i sve dodatne informacije ili metapodatke.

Klijent prima poruku odgovora i obrađuje ju, obično iscrtavanjem sadržaja u web pregledniku ili prikazivanjem u aplikaciji.

Klijent može inicirati dodatne zahtjeve prema poslužitelju, ponavljajući ciklus po potrebi.

Kada web preglednik želi dohvatiti resurs, šalje zahtjev poslužitelju koristeći URL. URL (engl. Uniform Resource Locator) koristi se za jedinstvenu identifikaciju resursa na webu. URL ima sljedeću sintaksu:

„protokol://nazivposluzitelja:prikljucak/putanja-i-naziv-datoteke“

URL se sastoji od 4 dijela:

1) Protokol: Protokol na razini aplikacije koji koriste klijent i poslužitelj, npr. HTTP, FTP i telnet.

2) Naziv poslužitelja (engl. hostname): Naziv DNS domene (npr. www.primjer.com) ili IP adresa (npr. 192.128.1.2) poslužitelja.
3) Priključak (engl. port): Broj TCP priključka koji poslužitelj osluškuje za dolazne zahtjeve od klijenata.
4) Putanja i naziv datoteke: Naziv i lokacija traženog resursa u direktoriju baze dokumenata poslužitelja

Na primjer, u URL-u http://www.primjer.com/dokumenti/index.html, komunikacijski protokol je HTTP, naziv poslužitelja je www.primjer.com. Broj porta nije naveden u URL-u i preuzima zadani broj, a to je TCP priključak 80 za HTTP. Putanja i naziv datoteke za resurs koji treba pronaći je "/dokumenti/index.html".

HTTP protokol definira skup metoda zahtjeva. Klijent može koristiti metode zahtjeva za slanje poruke zahtjeva HTTP poslužitelju. Najkorištenije metode su:

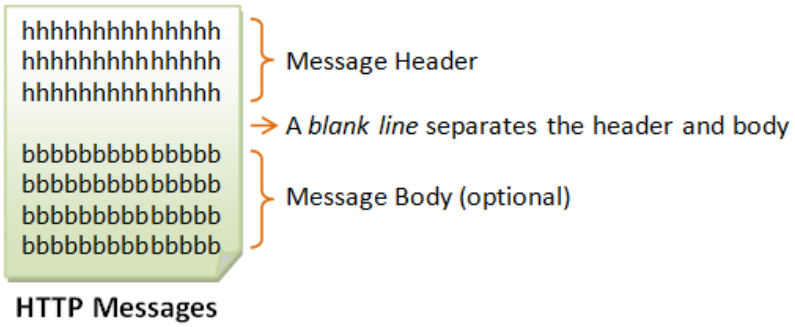
GET - Klijent može koristiti GET zahtjev za dobivanje resursa s poslužitelja

POST - Koristi se za slanje podataka na web poslužitelj

PUT – Slanje zahtjeva za pohranu podataka poslužitelju

DELETE – Slanje zahtjeva za brisanje podataka poslužitelju

HTTP klijent i poslužitelj komuniciraju slanjem tekstualnih poruka. Klijent poslužitelju šalje poruku zahtjeva, a poslužitelj vraća poruku odgovora. HTTP poruka sastoji se od zaglavlja poruke i neobaveznog tijela poruke koji su međusobno odvojeni praznim redom, kao što je prikazano:

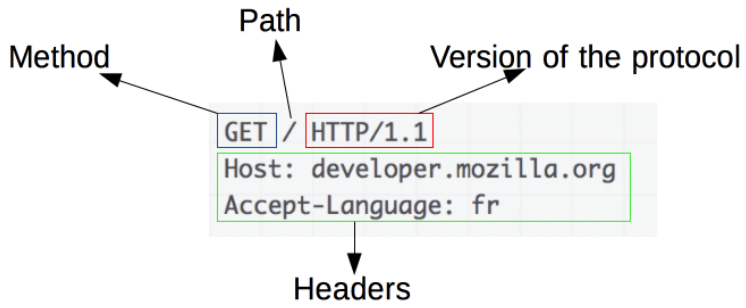


Slika 1. Struktura HTTP poruke (izvor: www3.ntu.edu.sg)

HTTP zahtjev sastoji se od sljedećih elemenata:

- HTTP metoda koja definira operaciju koju klijent želi izvesti
- Putanja resursa za dohvaćanje
- Verzija HTTP protokola

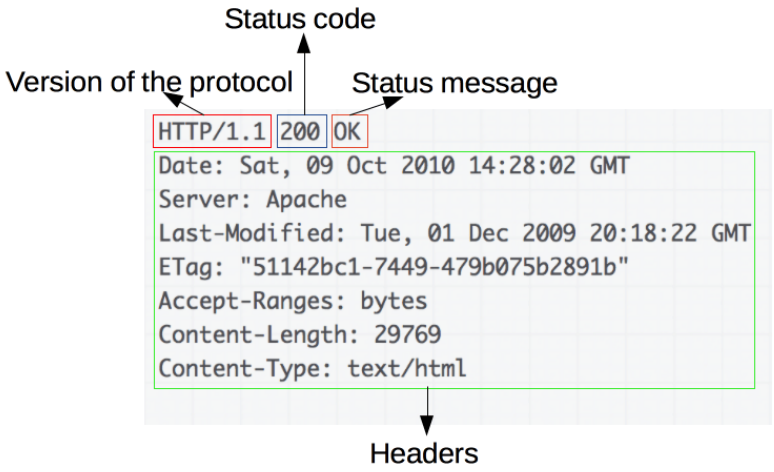
- Neobavezna zaglavlja koja prenose dodatne informacije za poslužitelje
- Tijelo, za neke metode kao što je POST, slično onima u odgovorima koji sadrže poslani resurs



Slika 2. Primjer HTTP zahtjeva (izvor: developer.mozilla.org)

HTTP odgovor se sastoji od sljedećih elemenata:

- Verzija HTTP protokola koju slijedi
- Statusni kod koji pokazuje je li zahtjev bio uspješan ili ne i zašto
- Statusna poruka, neautoritativni kratki opis statusnog koda
- HTTP zaglavlja, poput onih za zahtjeve
- Neobavezno tijelo koje sadrži dohvaćeni resurs



Slika 3. Primjer HTTP odgovora (izvor: developer.mozilla.org)

2.2. Arhitektura web aplikacije

Arhitektura web aplikacije predstavlja plan tj. sadržaj svih softverskih komponenti i njihovu međusobnu interakciju. Definira kako se podaci šalju preko HTTP-a i omogućuje da se klijent i server međusobno razumiju. Također osigurava validnost podataka u svim korisničkim zahtjevima. Kreira i upravlja zapisima dok pruža pristup baziran na dopuštenjima i autentikaciji. Web aplikacija tipično sadrži 3 ključne komponente:

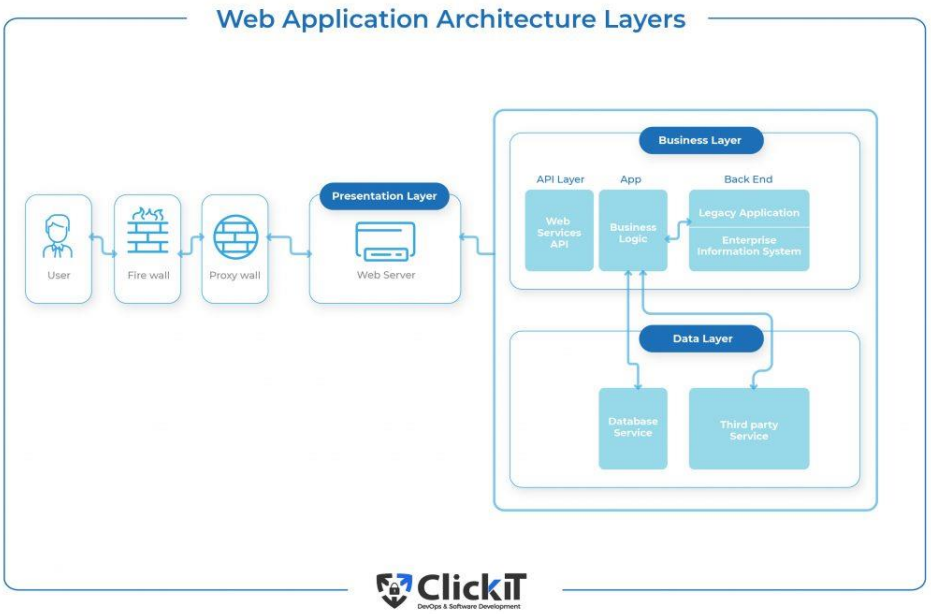
1) Web preglednik ili komponenta na strani klijenta ili tzv. front-end komponenta je ključna komponenta koja komunicira s korisnikom, prima unos i upravlja prezentacijskom logikom dok kontrolira interakciju korisnika s aplikacijom. Ako je potrebno, moguća je i validacija korisničkih unosa.
2) Web poslužitelj koji je poznat također kao back-end komponenta ili komponenta na strani poslužitelja upravlja poslovnom logikom i procesira korisničke zahtjeve usmjeravanjem zahtjeva na odgovarajuću komponentu te upravlja cjelokupnim operacijama aplikacije. Može pokretati i nadzirati zahtjeve širokog spektra klijenata.
3) Poslužitelj baze podataka daje potrebne podatke za aplikaciju. Obrađuje zadatke vezane za podatke. U višeslojnoj arhitekturi, poslužitelji baze podataka mogu upravljati poslovnom logikom uz pomoć pohranjenih procedura (engl. stored procedure).

2.2.1. Troslojna arhitektura

U tradicionalnoj dvoslojnoj arhitekturi postoje dvije komponente, a to su sustav na strani klijenta ili korisničko sučelje te pozadinski sustav koji je obično poslužitelj baze podataka. U ovakvoj arhitekturi poslovna logika je ugrađena u korisničko sučelje ili poslužitelj baze podataka. Loša strana dvoslojne arhitekture je to što s povećanim brojem korisnika opada učinkovitost izvršavanja. Navrh toga, izravna interakcija baze podataka i korisničkog sučelja također ostavlja potencijal za sigurnosne probleme (Eyskens, 2021).

Troslojna arhitektura se sastoji od sljedećih slojeva:

1) Prezentacijski sloj / sloj klijenta
2) Aplikacijski sloj / Poslovni sloj
3) Podatkovni sloj



Slika 4. Arhitektura standardne web app (izvor: clickittech.com)

Na aplikacijskom sloju se nalazi web poslužitelj koji pokreće jednu ili više web stranica. Koristi HTTP zajedno s drugim protokolima za slušanje korisničkih zahtjeva koji stižu sa preglednika. Obrađuje ih primjenom poslovne logike te isporučuje traženi sadržaj korisniku. Web poslužitelj može biti hardverski uređaj ili softverski program. Hardverski web poslužitelj je uređaj koji je spojen na internet i sadrži softver web poslužitelja te komponente web aplikacije kao što su slike, HTML dokumenti, JavaScript datoteke i CSS stilske tablice. Softverski web poslužitelj je softver koji razumije URL-ove i HTTP protokole. Korisnici mu mogu pristupiti putem domenskih naziva kako bi dobili traženi sadržaj. Dok statični web poslužitelj isporučuje pregledniku sadržaj kakav jeste, dinamički web poslužitelj ažurira podatke prije nego što ih isporuči pregledniku. Primjeri popularnih web poslužitelja su Apache i NGINX.

Na prezentacijskom sloju komponenta na strani klijenta omogućuje korisnicima interakciju s poslužiteljem i back-end servisima putem preglednika. Kod se nalazi u pregledniku, prima zahtjeve i prikazuje korisniku potrebne informacije. Tu dolaze UI/UX dizajn, obavijesti, konfiguracijske postavke, interaktivni elementi i slično. Neke od često korištenih front-end tehnologija su: HTML, CSS, JavaScript, React, Vue.js, Angular.js.

Komponenta na strani poslužitelja sadržana na aplikacijskom sloju ključna je komponenta koja prima korisničke zahtjeve, izvodi poslovnu logiku i isporučuje potrebne podatke front-end sustavima. Generalno sadrži poslužitelje, baze podataka, servise. Neke od često korištenih tehnologija na strani poslužitelja su: Node.js, Java, Python, PHP Laravel, Go, .NET, Ruby.

Dalje na aplikacijskom sloju također je sadržan tzv. API odnosno Application Programming Interface. To konkretno nije tehnologija nego koncept koji omogućuje developerima pristup određenim podacima i funkcijama softvera. Jednostavno rečeno, to je posrednik koji daje aplikacijama mogućnost međusobne komunikacije. Sastoji se od protokola, alata i podrutina potrebnih za izgradnju aplikacija. Na primjer, kada se korisnik prijavi u aplikaciju, aplikacija poziva API kako bi dohvatila podatke o računu i slično. Aplikacija će kontaktirati odgovarajuće poslužitelje kako bi primila te informacije i vratila podatke na stranu korisnika. Web API je API dostupan na webu putem HTTP protokola. Može se izgraditi pomoću tehnologija kao što su .NET i Java.

Pomoću API-a, developeri ne moraju stvarati sve od nule već koriste postojeće funkcije izložene kao API za povećanje produktivnosti i brži izlazak na tržište. API-ji značajno smanjuju troškove razvoja tako što smanjuju količinu vremena i energije utrošene na samo pisanje koda. Također rezultiraju poboljšanom suradnjom i povezanošću u cijelom ekosustavu te unaprjeđuju korisničko iskustvo.

Postoje različite vrste API-ja:

RESTful API tj. Representational State Transfer API u laganom JSON formatu. Vrlo je skalabilan, pouzdan i pruža brze performanse što ga čini najpopularnijim API-jem.

SOAP tj. Simple Object Access Protokol koji koristi XML za prijenos podataka. Zahtijeva veću propusnost i naprednu sigurnost.

XML-RPC što je skraćenica za Extensible Markup Language – Remote Procedure Calls koji koristi specifičan XML format za prijenos podataka.

JSON-RPC koristi JSON format za prijenos podataka.

Također bitna stavka aplikacijskog sloja u arhitekturi web aplikacija su poslužitelji ili instance oblaka (engl. cloud instances). Instanca oblaka je instanca virutalnog poslužitelja koja je izgrađena, isporučena i hostirana pomoću javnog ili privatnog oblaka i dostupna je putem interneta. Radi kao fizički poslužitelj koji se može neprimjetno kretati preko više uređaja ili implementirati više instanci na jednom poslužitelju. Kao takav, vrlo je dinamična, skalabilna i isplativa opcija. Moguće je automatski zamijeniti poslužitelje bez prekida rada aplikacije. Korištenje instance oblaka omogućava jednostavnu implementaciju i upravljanje web aplikacijama u bilo kojem okruženju.

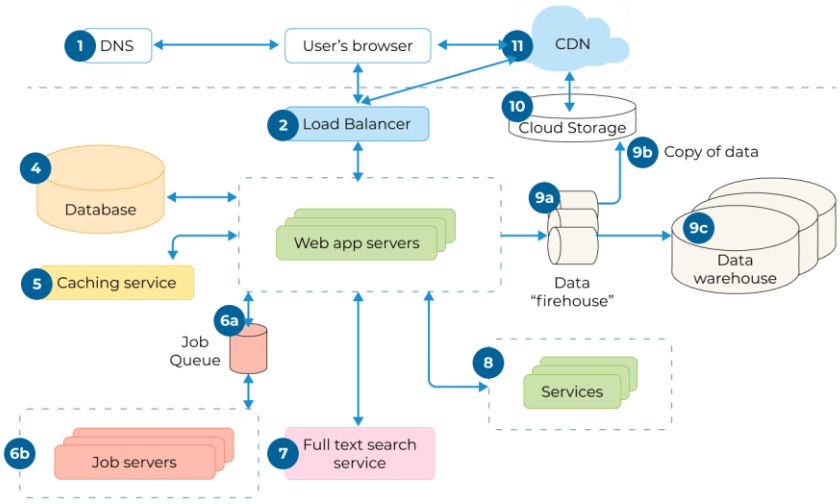
Što se podatkovnog sloja tiče, tu se nalazi baza podataka. Baza podataka je ključna komponenta web aplikacije koja pohranjuje i upravlja informacijama za web aplikaciju. Korištenjem funkcije moguće je pretraživati, filtrirati i sortirati informacije na temelju

korisničkog zahtjeva i prezentirati potrebne informacije krajnjem korisniku. Omogućuju pristup temeljen na ulogama radi održavanja integriteta podataka.

Postoje četiri važna aspekta koje treba razmotriti prilikom biranja baze podataka. To su veličina, brzina, skalabilnost i struktura. Za strukturirane podatke, baze podataka temeljene na SQL-u su dobar izbor. Odgovaraju financijskim aplikacijama u kojima je integritet podataka ključni zahtjev. Za upravljanje nestrukturiranim podacima, NoSQL je dobra opcija. Odgovara aplikacijama u kojima priroda dolaznih podataka nije predvidljiva. Tzv. Key Value baze podataka pridružuju svaku vrijednost ključu i odgovaraju situacijama gdje je potrebno pohranjivati korisničke profile, recenzije, komentare i slično. Za analitičke svrhe dobar izbor su baze podataka širokih stupaca (engl. wide column databases).

2.2.2. Arhitektura naprednih i skalabilnih web aplikacija

Postoji niz dodatnih komponenti koje ulaze u priču kada se povećavaju zahtjevi aplikacija. Također, arhitektura web aplikacija nije fiksirana nego s vremenom evoluira što je i u skladu s očekivanjima za sve vezano za web. Dobri primjeri su: sustav predmemoriranja, CDN, balanser opterećenja (Eyskens, 2021).



Slika 5. Primjer napredne arhitekture (izvor: litslink.com)

Sustav predmemoriranja je lokalna pohrana podataka koja aplikacijskom poslužitelju omogućava brzi pristup podacima umjesto da je obavezno svaki put kontaktirati bazu podataka. U tradicionalnom stilu postavljanja, podaci se pohranjuju u bazu podataka. Kada korisnik podnese zahtjev, aplikacijski poslužitelj traži te podatke od baze podataka i prezentira ih korisniku. Kada se ponovno zatraže isti podaci, poslužitelj bi trebao ponovno izvesti isti proces

koji se nepotrebno ponavlja i oduzima puno vremena. Pohranjivanjem ovih podataka u privremenu memoriju, aplikacije mogu brzo prikazati podatke korisnicima.

Sustav za predmemoriju može biti dizajniran u 4 modela:

1) Predmemorija aplikacijskog poslužitelja – predmemorija u memoriji uz aplikacijski poslužitelj što se odnosi na aplikacije koje imaju jedan čvor (engl. node)
2) Globalna predmemorija – svi čvorovi pristupaju jednom prostoru predmemorije
3) Distribuirana predmemorija – predmemorija je distribuirana po čvorovima pri čemu se koristi konzistentna hashing funkcija za usmjeravanje zahtjeva prema potrebnim podacima
4) Content Delivery Network (CDN) – koristi se za isporuku velikih količina statičkih podataka

Što se tiče alata za predmemoriju, dva najpopularnija su Redis i Memcached. Međutim, Redis nudi bogat skup vlastitih alata što ga čini primjenjivim za obavljanje raznih zadataka. Memcached, s druge strane, je jednostavan i lagan za korištenje.

CDN odnosno Content Delivery Network odnosi se na mrežu poslužitelja koja je instalirana na različitim geolokacijama kako bi korisnicima brže i bolje isporučila sadržaj. Umjesto kontaktiranja središnjeg poslužitelja, zahtjev korisnika se preusmjerava na CDN poslužitelj koji sadrži pohranjenu predmemoriranu verziju sadržaja. Zahvaljujući tome brzina i izvedba stranice se povećavaju, a smanjuje se gubitak paketa. Opterećenje poslužitelja je smanjeno. Također omogućuje segmentaciju publike i naprednu web sigurnost. Poznati CDN servisi su: CloudFront, Azure CDN, Googleov Cloud CDN te CloudFlare.

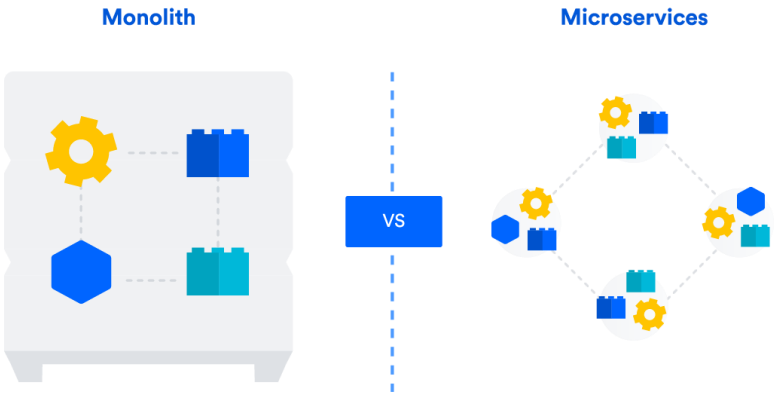
Balanser opterećenja (engl. load balancer) je usluga koja uravnotežuje prometna opterećenja distribucijom na različite poslužitelje na temelju dostupnosti ili unaprijed definiranih pravila. Kada se korisnički zahtjev primi u balanser opterećenja, on dohvaća status ispravnosti poslužitelja u smislu dostupnosti i skalabilnosti te usmjerava zahtjev na najbolji poslužitelj. Balanser opterećenja može biti hardverska komponenta ili softverski program.

Balansiranje opterećenja može se izvršiti na dva načina:

1) Balansiranje opterećenja na TCP/IP razini – balansiranje opterećenja na temelju DNS-a
2) Balansiranje opterećenja na razini aplikacije – balansiranje opterećenja na temelju opterećenja aplikacije

2.2.3. Tipovi arhitektura web aplikacija

Neka uobičajena podjela tipova arhitektura web aplikacija na osnovu stila razvoja i uzoraka implementacije je na monolitnu arhitekturu i mikroservisnu arhitekturu.



Slika 6. Monolitna i mikroservisna arhitektura (izvor: atlassian.com)

Monolitna arhitektura tradicionalni je model softverskog programa koji je izgrađen kao unificirana jedinica koja je samostalna i neovisna o drugim aplikacijama. Riječ „monolit“ često se pripisuje nečemu velikom, izgrađenom od jednog bloka kamena što nije daleko od istine o monolitnoj arhitekturi za dizajn softvera. Monolitna arhitektura jedinstvena je velika računalna mreža s jednom bazom koda koja povezuje sve poslovne probleme. Za promjenu ove vrste aplikacije potrebno je ažuriranje cijelog stacka tako što se pristupi bazi koda te izgradi i uvede ažurirana verzija sučelja na strani usluge. Zbog toga su ažuriranja dosta restriktirana i generalno dugo traju. Monoliti mogu biti prikladni u ranoj fazi života projekta zbog jednostavnosti upravljanja kodom, umjerenih kognitivnih napora i implementacije. To omogućuje da se sve u monolitu izbaci istovremeno.

Prednosti monolitne arhitekture su lakoća inicijalnog razvoja, jednostavnost implementacije, izvedba s obzirom na centraliziranu bazu koda, jednostavno testiranje i lagano debugiranje.

Monolitna arhitektura obično počinje pokazivati svoje nedostatke kada aplikacija postane prevelika i skaliranje se pojavi kao izazov. Nedostaci monolitne arhitekture su: sporiji daljnji razvoj, skalabilnost budući da se ne mogu skalirati individualne komponente, generalna pouzdanost jer greška u jednom modulu može utjecati na čitavu aplikaciju, otežan prelazak na druge ili novije tehnologije, nedostatak fleksibilnosti.

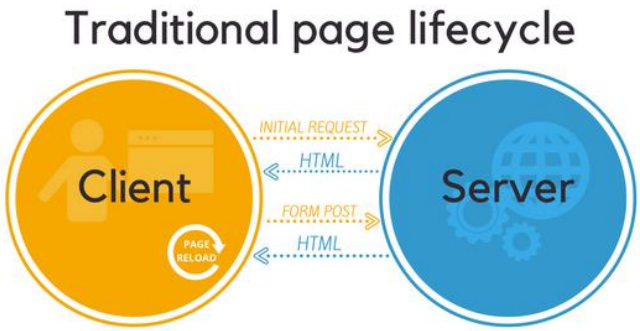
Mikroservisna arhitektura, također poznata i jednostavno kao mikroservisi, je arhitekturalna metoda koja se oslanja na niz servisa koji se mogu implementirati bez da ovise jedni o drugima. Ovi servisi imaju vlastitu poslovnu logiku i bazu podataka sa specifičnim ciljem. Ažuriranje, testiranje, implementacija i skaliranje se pojedinačno tj. odvojeno obavljaju unutar svakog servisa. Mikroservisi odvajaju ključne poslovne probleme specifične za određenu domenu u odvojene, neovisne baze koda. Mikroservisi ne umanjuju kompleksnost, ali su zahvaljujući njima stavke preglednije te kao rezultat toga omogućuju lakše rukovođenje istim kroz razdvajanje zadataka u manje procese koji funkcioniraju neovisno jedni od drugih a doprinose cjelini.

Prednosti mikroservisne arhitekture su: agilnost, fleksibilno skaliranje, CD (engl. continuous deployment) što znači češći i brži ciklusi novih izdanja, veoma održive aplikacije koje je lako testirati, neovisno izbacivanje određenih servisa, fleksibilnost glede tehnologije koja se koristi za razvoj, veća pouzdanost u smislu da se može implementirati promjene za određeni servis bez rizika rušenja cijele aplikacije.

Nedostaci mikroservisne arhitekture mogu biti: razvlačenje razvoja tj. vrijeme potrebno za razvoj može biti drastično povećano u usporedbi s monolitom ako nije razrađen dovoljno kvalitetan plan rada te menadžment projekta, potencijal za znatno veće troškove infrastrukture zbog pojedinačnih servisa, potreba za boljom i većom organizacijom rada, izazovno debugiranje, nedostatak standardizacije npr. glede korištene tehnologije ili jezika.

2.3. Jednostranična aplikacija i višestranična aplikacija

U najjednostavnijem obliku, aplikacija s više stranica sastoji se od nekoliko stranica sa statičnim informacijama (tekst, slike itd.) i poveznicama na druge stranice s njihovim vlastitim sadržajem. Tijekom prebacivanja na drugu stranicu, preglednik potpuno ponovno učitava sadržaj stranice i ponovno preuzima sve resurse, čak i komponente koje se ponavljaju na svim stranicama (npr. zaglavlje, podnožje).



Slika 7. Životni ciklus višestranične web aplikacije (izvor: mindk.com)

Korisnik dolaskom na web adresu aplikacije šalje zahtjev za primanje početne stranice aplikacije (1), poslužitelj obrađuje zahtjev i, u skladu s logikom rada, šalje HTML datoteku klijentskom dijelu za prikaz informacija korisniku (2), tada kada klijent pošalje zahtjev za dodavanje podataka ili prođe kroz interne veze (3), poslužitelj obrađuje te informacije i šalje novu HTML datoteku (4) aplikaciji na strani klijenta što dovodi do ponovnog učitavanja trenutne stranice u korisničkom pregledniku (Skólski, 2016).

Ne postoji ništa nužno loše niti pogrešno u ovom pristupu kada je riječ o potrebi za jednostavnim aplikacijama. Ali ako postoji potreba za stvaranjem bogatog korisničkog sučelja, tada stranica može postati vrlo složena i biti puna podataka, a ti se podaci prvo generiraju na poslužiteljskoj strani aplikacije, a zatim šalju klijentskoj strani. Zbog toga se povećava vrijeme učitavanja stranice i pogoršava korisničko iskustvo s aplikacijom. Potpuno drugačiji pristup razvoju web aplikacija je korištenje AJAX-a (Asynchronous JavaScript And XML), koji se pojavio 2000-ih. Prilikom korištenja AJAX-a u kodu stranice, moguće je prenijeti i dohvatiti potrebne podatke bez ponovnog učitavanja same stranice i bez trošenja nepotrebnog prometa. Ova tehnologija omogućuje integraciju u funkciju web-aplikacije s više stranica uz pomoć višestraničnih aplikacija (MPA), što poboljšava vrijeme odaziva aplikacije.

Prednosti aplikacije s više stranica:

MPA korisnicima pruža bolju vizualizaciju aplikacije. Višerazinska navigacija je glavni dio višestranične aplikacije. Budući da se MPA sastoji od mnogo stranica, lakše ih je promovirati pomoću ključnih riječi, mogu se postaviti za svaku stranicu posebno, što je trenutno najbolje rješenje za SEO (Skólski, 2016). Danas je to najpopularnije rješenje za razvoj web aplikacija. To znači da postoji širok izbor literature i generalnih izvora informacija za kvalitetan razvoj.

Nedostaci aplikacije s više stranica:

Poslovna logika višestraničnih aplikacija usko je povezana i s klijentskom i poslužiteljskom stranom, što ne dopušta promjenu poslužiteljskog dijela bez utjecaja na klijenta.

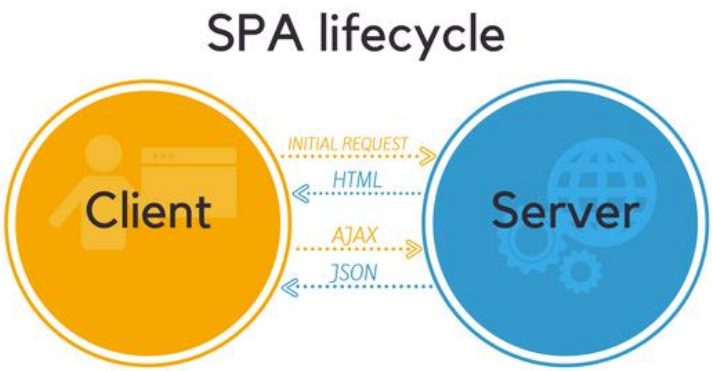
Razvoj postaje prilično složen. Programer mora koristiti okvire za klijentsku i poslužiteljsku stranu. To rezultira duljim vremenom razvoja aplikacije.

Ako se ne koristi AJAX, stranica će se ponovno pokrenuti svaki put kada korisnik klikne na poveznicu na web stranici

SPA je web aplikacija koja doslovno ima jednu stranicu - jedna HTML stranica se učitava u preglednik i ne učitava se ponovno tijekom korištenja. Drugim riječima, SPA je web aplikacija smještena na jednoj web stranici koja preuzima sav potreban kod za rad, zajedno sa samom stranicom. Aplikacija ovog tipa pojavila se relativno nedavno, s početkom ere HTML5 i SPA je tipičan predstavnik aplikacija na HTML5. Sadržaj takve stranice učitava se s poslužitelja pomoću AJAX-a. Za implementaciju rada putem AJAX-a potrebno je implementirati dio aplikacije i na strani poslužitelja. Za to se obično se koriste skriptni jezici. Budući da se HTML učitava na strani klijenta, veličina opterećenja je smanjena jer poslužitelj vraća samo JSON, a ne HTML.

Dostupne su dvije moguće implementacije SPA tehnologije: sav potreban kod (HTML, CSS, JavaScript) odmah se preuzima u klijentski dio aplikacije ili se potrebni kod dinamično učitava kao odgovor na radnje korisnika.

SPA zahtijevaju označavanje (HTML) i podatke (JSON) neovisno i prikazuju stranice izravno u pregledniku. SPA dinamički šalje samo podatke na poslužitelj, što oduzima manje vremena i pomaže smanjiti opterećenje mreže. To se radi korištenjem asinkronih zahtjeva prema poslužitelju pomoću AJAX-a.



Slika 8. Životni ciklus jednostranične aplikacije (izvor: mindk.com)

Web aplikacija na jednoj stranici uvijek radi na strani klijenta, tako da nema stalnog ponovnog učitavanja stranice i korisnik može bez problema koristiti aplikacije. SPA podržava navigaciju klijenta. Svako kretanje korisnika po stranicama-modulima jednoznačno je zapisano u povijesti navigacije, a navigacija je duboka, odnosno ako korisnik kopira i otvori poveznicu na internu stranicu-modul u drugom pregledniku ili prozoru, otići će na odgovarajuću stranicu.

SPA se nalazi na jednoj web stranici, tako da sve skripte i stilove potrebne za rad stranice treba definirati na jednom mjestu projekta - na jednoj web stranici. Jednostranične aplikacije učitavaju sve skripte potrebne za pokretanje aplikacije kada se web stranica inicijalizira.

Prednosti jednostranične aplikacije:

SPA je brza jer se većina resursa (HTML, CSS i skripte) učitava samo jednom tijekom vijeka trajanja aplikacije. Samo se podaci prenose naprijed i nazad.

Sami razvoj je pojednostavljen i poboljšan. Nema potrebe za pisanjem koda za prikaz stranica na poslužitelju. Puno je lakše započeti s razvojem jer obično je moguće započeti razvoj iz datoteke file.//URI, bez korištenja ikakvog poslužitelja.

SPA-ove je generalno lako debugirati u Chromeu. Također, lakše je izraditi mobilnu aplikaciju jer programer može ponovno upotrijebiti isti pozadinski kod za web aplikaciju i izvornu mobilnu aplikaciju.

SPA može učinkovito predmemorirati bilo koju lokalnu pohranu. Aplikacija šalje samo jedan zahtjev, pohranjuje sve podatke te zatim može koristiti te podatke i radi čak i bez mrežne konekcije.

Nedostaci jednostranične aplikacije:

Iznimno je teško napraviti SEO za jednostranične aplikacije.

Sporo se preuzima jer se na klijenta moraju učitati teški klijentski okviri.

Zahtijeva da JavaScript bude prisutan i omogućen. Ako bilo koji korisnik onemogući JavaScript u svom pregledniku, neće biti moguće prikazati aplikaciju i njezine radnje na ispravan način.

U usporedbi s „tradicionalnom“ aplikacijom, SPA je manje sigurna. Zbog Cross-Site Scriptinga (XSS), omogućuje napadačima ubacivanje skripti na strani klijenta u web aplikacije drugih korisnika.

Curenje memorije u JavaScriptu može uzrokovati usporavanje čak i snažnih sustava.

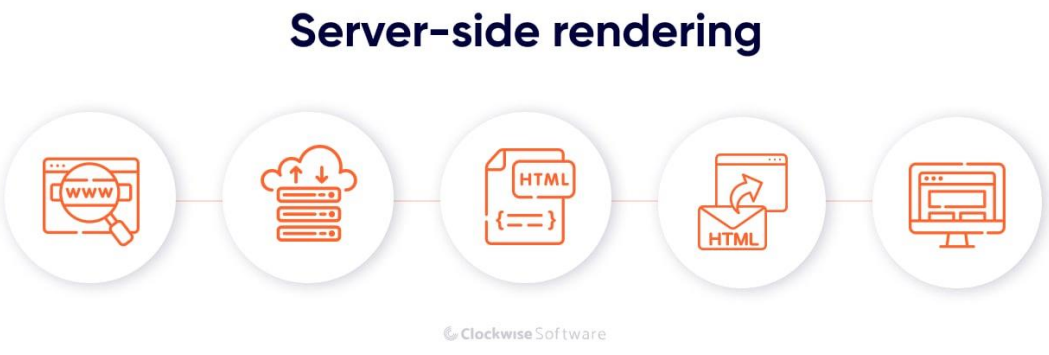
2.4. Osnovna usporedba SSR-a i CSR-a

Tijekom razvoja web aplikacije programski inženjeri moraju donijeti brojne odluke vezane za izgradnju arhitekture web aplikacije, konfiguraciju poslužitelja, odabir najprikladnijih alata i tehnologije. Uz navedeno također trebaju odlučiti o vrsti iscrtavanja (engl. rendering) aplikacije: iscrtavanje na strani poslužitelja (SSR) ili iscrtavanje na strani klijenta (CSR).

Mnogi vlasnici tvrtki izbjegavaju dubinsko istraživanje teme iscrtavanja te vjeruju svom razvojnom timu glede donošenja odluke. Međutim, važno je biti svjestan vrste iscrtavanja koja je bolja za određenu aplikaciju kako bi se izbjegao niz izazova nakon izbacivanja iste. Neki od tih izazova uključuju loše rezultate SEO-a, spor rad aplikacije i nezadovoljavajuće korisničko iskustvo. Štoviše, ako se tvrtka tj. razvojni tim odluči za pogrešnu vrstu iscrtavanja, vjerojatno će ju morati mijenjati u budućnosti, a to će naravno dovesti do izgubljenog vremena i novca.

Tijekom iscrtavanja na strani poslužitelja, kod aplikacije obrađuje se na poslužitelju. Prikaz stranice generira se na back-endu i šalje na front-end. Ovaj proces se može razložiti na sljedeće korake:

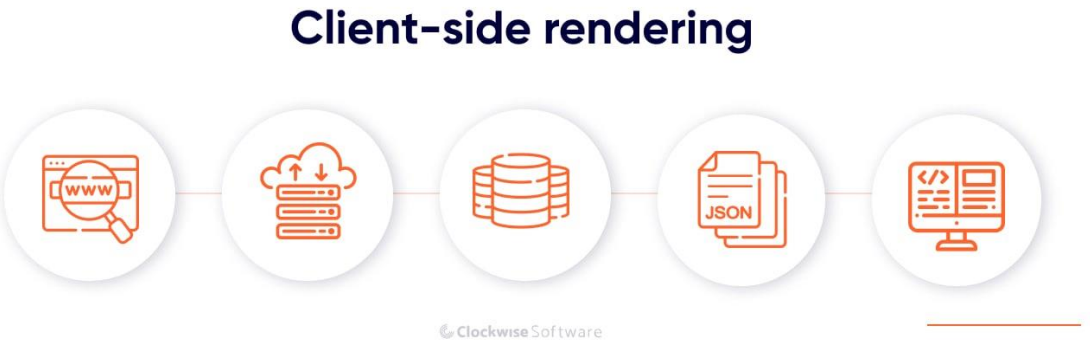
- Korisnik unosi URL u adresnu traku preglednika.
- Zahtjev za podacima šalje se poslužitelju na navedeni URL.
- Poslužitelj generira HTML datoteku s potrebnim podacima i stilovima na temelju zahtjeva koji dolazi s front-enda.
- Poslužitelj šalje HTML datoteku kao odgovor pregledniku.
- Preglednik izvršava HTML i prikazuje stranicu.



Slika 9. Koraci SSR-a (izvor: clockwise.software)

Tijekom iscrtavanja na strani klijenta, kod aplikacije se šalje u korisnički preglednik gdje se obrađuje i pretvara u vidljive i interaktivne stranice. Iscrtavanje na strani klijenta događa se na sljedeći način:

- Korisnik unosi URL u adresnu traku preglednika.
- Zahtjev za podacima šalje se poslužitelju na navedeni URL.
- Poslužitelj izvlači informacije iz baze podataka i generira odgovor u traženom formatu, kao što je JSON.
- Poslužitelj šalje tražene podatke klijentu.
- Preglednik prikazuje stranicu tako što umetne dobivene podatke u HTML kod. Preglednik čini stranicu vidljivom korisniku izvršavanjem JavaScript koda pohranjenog na strani klijenta.



Slika 10. Koraci CSR-a (izvor: clockwise.software)

Prednosti iscrtavanja na strani poslužitelja su sljedeće:

Brzo početno učitavanje. Pri korištenju SSR-a odgovor poslužitelja na zahtjev stranice je iscrtana HTML datoteka. To znači da preglednik ne treba obrađivati velike JavaScript datoteke i može prikazati sadržaj kroz nekoliko milisekundi.

Kvalitetan SEO. Kako bi došle do vrha na rezultatima tražilice, web stranice moraju biti čitljive za tražilice. Čitljive su kada odmah prikazuju sve informacije, po mogućnosti u obliku teksta, a to je upravo način na koji se SSR stranice pojavljuju u pregledniku.

Vrijeme odgovora web stranice također utječe na njezino rangiranje. Potrebno je više vremena da stranica koju je iscrtao klijent prikaže svoj sadržaj te alat za indeksiranje može napustiti stranicu prije nego što je odgovor spreman. Stranice iscrtane sa strane poslužitelja gotovo odmah prikazuju sadržaj što zadovoljava tražilice i jedan je od razloga njihovog višeg rangiranja.

Nedostaci iscrtavanja na strani poslužitelja su sljedeći:

Veliko opterećenje poslužitelja. Kako SSR troši kapacitet poslužitelja, mnogi istodobni zahtjevi od različitih korisnika mogu usporiti odgovor poslužitelja. To može dovesti do nezadovoljstva od strane korisnika i čak rezultirati višom stopom napuštanja stranice. Kako bi to izbjegli, vlasnici aplikacija trebali bi pravilno procijeniti broj korisnika i kapacitet opterećenja web stranice kako bi odabrali pružatelja usluga hostinga s dovoljnom memorijom, propusnošću i drugim karakteristikama za visoku izvedbu.

Sporiji prijelaz između stranica u usporedbi s web-aplikacijama iscrtanim u pregledniku. U slučaju SSR-a, kada se korisnik treba prebacivati između stranica, svaka se stranica iscrtava ispočetka od nule, čak i ako su razlike između stranica minimalne. Na primjer, kada korisnik prelazi s jednog članka na drugi unutar bloga, zaglavlje i podnožje se ponovno iscrtavaju čak i ako su identični na svim stranicama bloga.

Neke od prednosti iscrtavanja na strani klijenta su:

Minimalno opterećenje poslužitelja. Posebna odlika iscrtavanja na strani klijenta je to da se sadržaj web stranice prikazuje u pregledniku, što znači da skoro ni ne stvara opterećenje na poslužitelju.

Brže iscrtavanje web stranice nakon inicijalnog učitavanja. Kada se korisnik prebacuje između stranica CSR aplikacija, prikazuju se samo nove informacije.

Nedostaci iscrtavanja na strani klijenta mogu biti:

Dulje inicijalno vrijeme učitavanja. Kada korisnik prvi put posjeti web stranicu sa CSR-om, potrebno je više vremena za preuzimanje i izvođenje JavaScript datoteka u usporedbi s vremenom koje bi bilo potrebno za učitavanje web stranice prikazane na poslužitelju. JavaScript datoteke dovode do duljeg početnog vremena čekanja za posjetitelje, posebno kada imaju zastarjeli preglednik ili sporu internetsku vezu. Sporije učitavanje moglo bi natjerati neke korisnike da napuste stranicu.

Loš SEO. Iako Google napreduje u indeksiranju web stranica s JavaScriptom, još uvijek ne rangira takve web stranice tako visoko kao one prikazane na poslužitelju. Budući da početno učitavanje CSR aplikacija traje dulje, alati za indeksiranje možda neće čekati da se sadržaj učita i umjesto toga će napustiti stranicu. Stoga, web stranice prikazane na strani klijenta obično zauzimaju niže pozicije u rezultatima pretraživanja.

2.5. Detaljno o SSR-u

2.5.1. Vremenska učinkovitost SSR-a

Kako bi programski inženjeri mogli učinkovito komunicirati tijekom razrade plana razvoja i kako bi uopće razumjeli problematiku potrebno je kvalitetno poznavanje terminologije vezane za iscrtavanje i performanse. Što se samog iscrtavanja tiče, uz već upoznate termine potrebno je spomenuti rehidraciju (engl. rehydration) i prethodno iscrtavanje (engl. prerendering). U području performansi važni termini su Time To First Byte (TTFB), First Contentful Paint (FCP), Interaction To Next Paint (INP) te Total Blocking Time (TBT) (Miller i Osmani, 2023).

Rehidracija – pokretanje JavaScript view-ova na klijentu na takav način da ponovno koriste (recikliraju) HTML DOM stablo i podatke iscrtane na strani poslužitelja.

Prethodno iscrtavanje – pokretanje aplikacije na strani klijenta u vrijeme izrade kako bi se uhvatilo njeno početno stanje kao statički HTML.

Time To First Byte (TTFB) – smatra se vremenom između klika na link i prvog bita sadržaja koji dolazi.

First Contentful Paint (FCP) – vrijeme potrebno do trenutka kada traženi sadržaj (npr. tijelo članka) postane vidljiv.

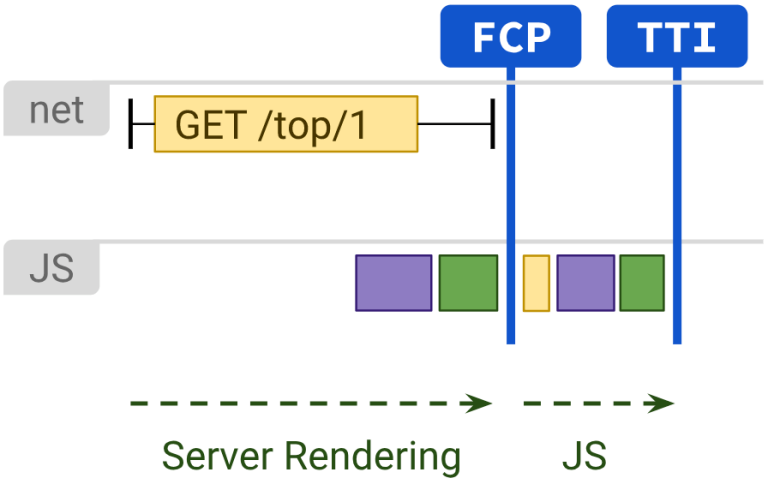
Interaction To Next Paint (INP) – opisuje se kao reprezentativna metrika koja procjenjuje reagira li stranica ujednačenom stabilnom brzinom na korisničke unose.

Total Blocking Time (TBT) – proxy metrika za INP koja izračunava količinu vremena koju je glavna nit provela blokirana tijekom učitavanja stranice.

Iscrtavanje na strani poslužitelja generira puni HTML kod za stranicu na poslužitelju kao odgovor na navigaciju. Time se izbjegavaju ponavljajuće putanje za dohvaćanje podataka i izradu predložaka na klijentu. Umjesto takvog viška to je sve riješeno unaprijed prije nego preglednik dobije odgovor.

Iscrtavanje na strani poslužitelja generalno daje brz FCP. Pokretanje logike stranice i iscrtavanje na poslužitelju omogućuju izbjegavanje slanja puno JavaScripta klijentu. To pomaže smanjiti TBT stranice što također može dovesti do nižeg INP-a jer glavna nit nije toliko često blokirana tijekom učitavanja stranice. Kada se glavna nit rjeđe blokira, korisničke

interakcije imat će više prilika da se pokrenu ranije. Ovo ima smisla budući da se pomoću SSR-a zapravo samo šalje tekst i linkovi na preglednik korisnika. Ovaj pristup može dobro funkcionirati za širok spektar uređaja i mrežnih uvjeta te omogućuje zanimljive optimizacije preglednika.



Slika 11. SSR putanja kroz vrijeme (izvor: web.dev)

Koristeći SSR manja je vjerojatnost da će korisnici morati čekati da se JavaScript koji je vezan za CPU pokrene prije nego što budu u mogućnosti koristiti stranicu. Čak i kada se third-party JavaScript ne može zaobići, korištenje SSR-a za smanjenje troškova vlastitog JavaScript sadržaja može tvrtki omogućiti veći budžet za druge svrhe. Međutim, postoji jedan nedostatak kod ovog pristupa, a to je činjenica da generiranje stranica na poslužitelju zahtijeva vrijeme što može rezultirati većim TTFB-om.

2.5.2. Hidracija

Izuzev biranja između striktnog CSR ili SSR pristupa postoji vrsta pristupa gdje se to dvoje zapravo kombinira u pokušaju da se minimiziraju nedostaci a maksimiziraju pozitivne karakteristike s obje strane. Taj pristup je generalno poznat kao hidracija (engl. hydration).

Navigacijske zahtjeve poput učitavanja ili ponovnog učitavanja cijele stranice obrađuje poslužitelj koji iscrtava aplikaciju u HTML, a zatim se JavaScript i podaci korišteni za iscrtavanje ugrađuju u rezultirajući dokument. Kada se ispravno napravi, ovim pristupom se

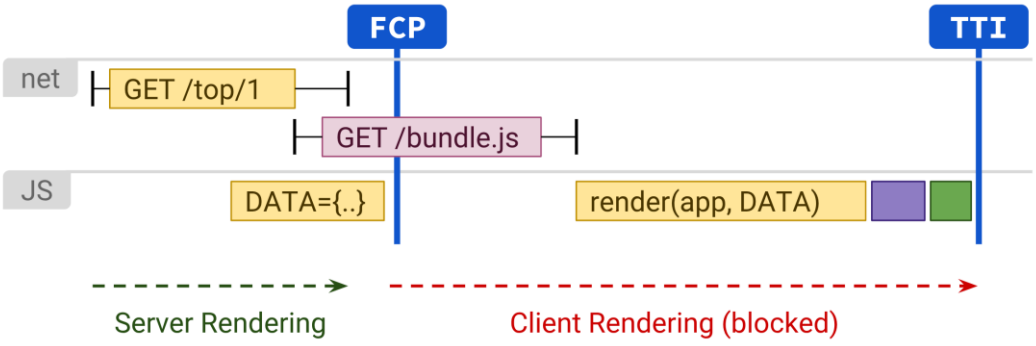
postiže brz FCP baš kao sa SSR-om, a zatim slijedi ponovno iscrtavanje na klijentu koristeći hidraciju. Ovo je učinkovito rješenje, ali može imati znatne nedostatke što se tiče performansi.

Primarna loša strana iscrtavanja na strani poslužitelja sa hidracijom je da može imati značajan negativan utjecaj na TBT i INP čak i ako poboljšava FCP. Stranice iscrtane na strani poslužitelja mogu se činiti kao da su učitane i interaktivne, ali zapravo ne reagiraju na unos dok se ne izvrše skripte za komponente na strani klijenta i dok se ne pridruže potrebni event handleri. To može potrajati nekoliko sekundi ili u slučaju mobilnog uređaja čak par minuta. No problemi s hidracijom ne staju tu i mogu zapravo biti i gori. JavaScript na stani klijenta ima izazov – ispravno i precizno nastaviti od točke na kojoj je poslužitelj stao bez da mora ponovno tražiti sve podatke koje je poslužitelj koristio za renderiranje svog HTML-a. Rješenje tog problema obično nije optimalno, tj. rezultira visokom razinom dupliciranja.

Slika 12. Hidracija i dupliciranje (izvor: web.dev)

U navedenom primjeru poslužitelj vraća opis korisničkog sučelja aplikacije kao odgovor na navigacijski zahtjev, ali također vraća izvorne podatke koji su bili korišteni za sastavljanje tog korisničkog sučelja i potpunu kopiju implementacije korisničkog sučelja koja se zatim pokreće na klijentu. Ovo korisničko sučelje postaje interkativno tek nakon što bundle.js završi s učitavanjem i izvođenjem.

Podaci o kvaliteti izvedbe prikupljeni sa stvarnih web stranica pomoću SSR-a i hidracije ukazuju na to da bi se upotreba iste trebala izbjegavati. U konačnici, razlog se svodi na korisničko iskustvo. Solidna je šansa da će korisnici zapinjati u nepoznatom teritoriju gdje se čini da nema interaktivnosti iako stranica izgleda kao da je spremna za upotrebu.



Slika 13. Hidracija, putanja kroz vrijeme (izvor: web.dev)

Doduše, postoji nekoliko modificiranih načina korištenja hidracije koji bi mogli dati nadu za učinkovitost ovog pristupa.

Streaming SSR omogućuje slanje HTML-a u komadima (engl. chunks) koje preglednik može progresivno prikazivati kako ih prima kroz vrijeme. Ovo može omogućiti brz FCP jer HTML označavanje brže stiže do korisnika.

U progresivnoj hidraciji pojedinačni dijelovi aplikacija koje je iscrtao poslužitelj se pokreću kroz vrijeme umjesto uobičajenog pristupa pokretanja cijele aplikacije odjednom. To može pomoći u smanjenju količine JavaScripta potrebnog da stranice budu interaktivne, budući da se nadogradnja dijelova stranice na strani klijenta koji su niskog prioriteta može odgoditi kako bi se spriječilo blokiranje glavne niti. Progresivna hidracija također može pomoći u izbjegavanju jedne od najčešćih zamki hidracije sa SSR-om, a to je slučaj gdje se DOM stablo koje se iscrtalo na poslužitelju uništava i zatim odmah ponovno gradi – najčešće zato što je početno sinkrono iscrtavanje na strani klijenta zahtijevalo podatke koji nisu bili sasvim spremni.

Također postoji opcija djelomične hidracije. Djelomična hidracija se pokazala kao teška za provesti u djelo. Ovaj pristup je proširenje ideje progresivne hidracije, gdje se analiziraju pojedinačni dijelovi koji će se progresivno hidrirati, a identificiraju se oni s malom interaktivnošću ili bez reaktivnosti. Za svaki od ovih većinski statičnih dijelova, odgovarajući JavaScript kod se zatim pretvara u inertne reference i dekorativnu funkcionalnost, smanjujući njihov otisak na strani klijenta gotovo na nulu. Pristup djelomične hidracije dolazi sa svojim problemima i kompomisima. Predstavlja neke zanimljive izazove za predmemoriranje, a što se tiče navigacije na strani klijenta ne može se pretpostaviti da će poslužiteljski generiran HTML za inertne dijelove aplikacije biti dostupan bez učitavanja cijele stranice.

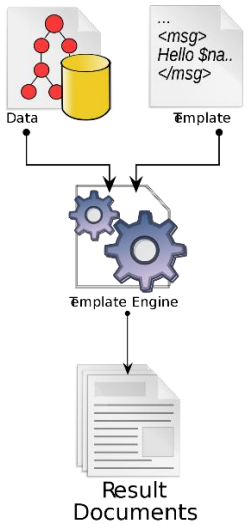
Trizomorfno (engl. trisomorphic) iscrtavanje je tehnika koja koristi streaming SSR za inicijalne navigacije ili navigacije bez JavaScripta, a zatim koristi service worker za

prikazivanje HTML-a za navigacije. To može pomoći održati predmemorirane komponente i predloške ažuriranima i omogućiti navigacije u SPA stilu za iscrtavanje novih viewova u istoj sesiji. Ovaj pristup najbolje funkcionira kada se može dijeliti isti kod za predloške i usmjeravanje između poslužitelja, stranice klijenta i service workera.

2.5.3. Korištenje predložaka

Procesor predložaka je softver koji je dizajniran za kombiniranje predložaka (engl. template) s podatkovnim modelom s ciljem dobivanja krajnjih iskoristivih dokumenata kao rezultat. Procesori predložaka se generalno koriste u raznim sferama softverskog inženjerstva, no kad je riječ o razvoju web aplikacija obično se koristi termin template engine. Sami termin se razvio kao generalizirani opis programskih jezika čija je primarna ili isključiva svrha obrada predložaka i podataka za ispis teksta. Dok se najviše spominje u kontekstu razvoja web aplikacija, nije ni u kojem slučaju isključivo na to ograničen.

Template engine obrađuje predloške za web i izvorne podatke kako bi proizveo jednu ili više web stranica ili fragmenata stranice. To radi tako što uzima datoteku predloška koja je HTML datoteka s nekim posebnim oznakama ili sintaksom i podatkovni objekt koji je objekt s podacima (obično iz baze podataka) koje je potrebno umetnuti u predložak. Template engine zatim analizira datoteku predloška, zamjenjuje rezervirana mjesta (engl. placeholders) podacima i generira konačni HTML kod koji se može poslati pregledniku (Hahn, 2016).



Slika 14. Opis template engine-a (izvor: wikipedia.com)

Generiranje stranice uz pomoć template engine-a je proces koji sadrži nekoliko koraka.

Prvo je potrebno definirati sami predložak, dakle kreirati datoteku predloška koja sadrži HTML kod s potrebnim oznakama ili sintaksom koja označava gdje treba umetnuti dinamične podatke. Kada se aplikacija pokrene, template engine kompajlira datoteku predloška u funkciju koja se kasnije koristi za iscrtavanje samog predloška.
Aplikacija dohvaća podatke koje je potrebno umetnuti u predložak te ih prosljeđuje kompajliranoj funkciji predloška koja generira konačni HTML kod tako što zamjeni rezervirana mjesta u predlošku sa stvarnim podacima.

Korištenje template engine-a dolazi sa svojim nizom i prednosti i nedostataka, kao i većina toga vezanog za razvoj web aplikacija.

Template engine-i nude nekoliko prednosti za web razvoj, kao što je odvajanje prezentacijskog sloja od podatkovnog sloja, smanjenje dupliciranja koda i suvišnog koda općenito te omogućuju ponovnu upotrebu predložaka na različitim stranicama ili komponentama. Također, template engine-i omogućuju prilagođavanje izgleda i dojma web stranica bez mijenjanja temeljnih podataka ili logike što znači da cjelokupni kod postaje kvalitetniji po pitanju sažetosti, čitljivosti, dosljednosti, skalabilnosti i prilagodljivosti.

Template engine-i mogu imati neke nedostatke kao što je uvođenje dodatne količine složenosti i apstrakcije, što može otežati otklanjanje pogrešaka i testiranje. Osim toga, programeri će možda morati naučiti novu sintaksu i pravila koja neće nužno biti kompatibilna s drugim alatima ili jezicima. Problemi ili ograničenja izvedbe mogu se pojaviti ovisno o veličini i složenosti predložaka i podataka. Također, template engine-i možda neće podržavati sve značajke ili funkcije koje su potrebne za web stranicu kao što su interaktivnost, upravljanje stanjem ili usmjeravanje.

Što se tiče template engine-a za Node.js, postoji veoma širok izbor istih. Neki od dostupnih, ali ne i svi, su: Pug, Haml.js, EJS, Squirrelly, marko, Blade, LiquidJS. U nastavku slijedi analiza dva engine-a s popisa, a to su Pug i EJS. Oba navedena imaju velik broj korisnika te su generalno kvalitetni izbori tako da nije moguće praviti usporedbe s ciljem biranja pobjednika već je usporedba isključivo informativnog karaktera. Odabir se većinski svodi na osobne preference i potrebe.

2.5.4. Pug i EJS

Pug i EJS su u suštini dvije najpopularnije opcije za predloške kad je riječ o Node.js poslužiteljskom okruženju. Slijedi usporedba na osnovu parametara kao što su čitljivost, skalabilnost i performanse.

Pug je template engine koji je originalno bio poznat pod imenom Jade. Dizajniran je s ciljem da bude sažet, pregledan, jednostavan za pisanje. Koristi sintaksu koja je slična jezicima baziranima na uvlačenju redaka kao što je Python. Jedna od glavnih prednosti Pug-a je to što dozvoljava pisanje HTML koda na način koji je veoma čitljiv i ekspresivan. To je vidljivo na primjer prilikom korištenja oznaka kao što su „div“ i „p“ gdje nema potrebe za zatvaranjem istih nego Pug to radi samostalno.

Još jedna prednost Pug-a je to što ima veliku i aktivnu zajednicu programera, što znači da je dostupna velika količina online resursa i podrške. Također je dostupan niz dodataka i biblioteka koji mogu pomoći u proširivanju funkcionalnosti i olakšavanju načina rada.

EJS, s druge strane, je template engine koji programerima omogućuje umetanje JavaScript koda direktno u HTML predloške. Koristi sintaksu koja je slična HTML-u uz dodane posebne oznake pomoću kojih se umetne spomenuti JavaScript kod.

Jedna od glavnih prednosti EJS-a je činjenica da ga je lako naučiti i koristiti, pogotovo ako programer već ima iskustva u korištenju HTML-a i JavaScripta. Osim toga, što se tiče online zajednice vezane za EJS, slična je situacija kao i sa Pug-om. Ima veliku i aktivnu zajednicu korisnika i dostupan je velik broj dodataka i biblioteka koje mogu proširivati funkcionalnosti.

Pored svih sličnosti ipak postoje i neke razlike između ova dva engine-a.

Što se tiče lakoće korištenja, oboje je relativno lako koristiti, ali Pug-ova sintaksa će vjerojatno biti poznatija programerima koji imaju iskustva s jezicima baziranima na uvlačenju redaka kao što je Python. EJS, međutim, ima sintaksu koja je slična HTML-u koja može biti poznatija drugom tipu programera.

Po pitanju čitljivosti koda možemo zaključiti da su Pug predlošci obično kraći i lakši za iščitati od EJS predložaka zato što koriste sintaksu sa uvlačenjem redaka umjesto HTML oznaka. Iz ovog razloga Pug može biti dobar izbor ako je važan cilj imati predloške koje je lako održavati i ažurirati.

Skalabilnost je još jedna bitna stavka za uzeti u obzir prilikom biranja odgovarajućeg template engine-a. Naravno, i Pug i EJS su skalabilni i mogu se koristiti za izradu web aplikacija bilo koje veličine. Međutim, sažeta sintaksa Pug-a i mogućnost korištenja varijabli i drugih

dinamičnih elemenata potencijalno ipak njega čini boljim izborom za izradu većih, kompleksnijih predložaka.

Što se tiče performansi, Pug predlošci su generalno brži i učinkovitiji od EJS predložaka jer se kompajliraju u HTML na strani poslužitelja. Ovo može biti prednost ako se radi o pravljenju velike ili kompleksne aplikacije koja treba podnijeti puno prometa.

Kao i ranije rečeno, izbor između Pug-a i EJS-a se na kraju svodi na osobne preference i specifične zahtjeve projekta. Oboje su kvalitetni engine-i koji imaju svoje karakteristike i pogodnosti te je na programerima da odaberu ono što njima najviše odgovara.

Uz sve navedeno još je preostalo pokazati neke sličnosti i razlike Pug-a i EJS-a na primjerima u kodu. Instalacija i jednog i drugog je poprilično jednostavna, to se čini pomoću „npm install pug“ ili „npm install ejs“ te uz to dodavanjem par linija koda za postavljanje expressa i odabranog template-a:

```
const express = require("express");
const app = express();
// Postavljanje Pug-a
app.set("view engine", "pug");
// Postavljanje direktorija gdje se nalaze Pug predlosci
app.set("views", "./views");
// Definiranje rute koja iscrtava "index" predlozak
app.get("/", (req, res) => {
  res.render("index");
});

// Pokretanje posluzitelja

app.listen(3000, () => {
  console.log("Posluzitelj slusa na portu 3000");
});
```

Naravno, potrebno je zapravo kreirati „views“ mapu i u njoj predložak „index.pug“. Predložak može izgledati kao sljedeće:

```
html
  head
    title Testna stranica
  body
    h1 Dobrodosli na testnu stranicu!!
    p Ovo je primjer Pug predloska.
```

Postavljanje EJS-a je u suštini identično:

```
const express = require("express");
const app = express();
// Postavljanje EJS-a
app.set("view engine", "ejs");
// Postavljanje direktorija gdje se nalaze EJS predlosci
app.set("views", "./views");
// Definiranje rute koja iscrtava "index" predlozak
app.get("/", (req, res) => {
  res.render("index");
});

// Pokretanje posluzitelja

app.listen(3000, () => {
  console.log("Posluzitelj slusa na portu 3000");
});
```

Predložak identičan prošlom primjeru bi u EJS-u izgledao ovako (s tim da je naziv datoteke „index.ejs“):

```
<html>
  <head>
    <title>Testna stranica</title>
  </head>

  <body>
    <h1>Dobrodosli na testnu stranicu!!</h1>

    <p>Ovo je primjer Pug predloska.</p>
  </body>
</html>
```

Sintaksa za korištenje petlji se također razlikuje u ova dva engine-a.

Pug:

```
html
  body
    h1= naslov
    ul
      each predmet in predmeti
        li= predmet
```

Identičan primjer u EJS-u:

```
<html>
  <body>
    <h1><%= naslov %></h1>
    <ul>
      <% predmeti.forEach(function (predmet) { %>
        <li><%= predmet %></li>
      <% } %>
    </ul>
  </body>
</html>
```

Naravno, cilj korištenja template engine-a je kombiniranje podataka sa samim predlošcima. Slijedi jednostavan primjer umetanja podataka.

U „app.js“ bi trebalo uraditi dodatak i izmjenu u sljedećem stilu:

```
const predmet = {
    naziv: 'Programiranje 1',
    ocjena: 5,
}
app.get('/', (req, res) => {
    res.render('pages/index', {
        predmet: predmet
    })
})
```

Naravno, potrebno je i omogućiti prikaz podataka u „index.ejs“ odnosno „index.pug“:

```
<p>Predmet: <%= predmet.naziv  %></p>
<p>Ocjena: <%= predmet.ocjena  %></p>
```

```
p Predmet: #{predmet.naziv}
p Ocjena: #{predmet.ocjena}
```

Ovo su svakako samo najosnovniji primjeri korištenja dva template engine-a. Upotreba istih i njihove funkcionalnosti idu mnogo dalje od navedenog, ali to izlazi van parametara ovog rada.

3. PRAKTIČNI RAD

Sljedeće navedeno je prikaz minijaturne web aplikacije koja dohvaća podatke koristeći API te zatim iste iscrtava pomoću Pug-a i EJS-a. Aplikacija je napravljena u dvije varijante, pomoću jednog i drugog template engine-a, ali je krajnji vizualni rezultat za korisnika identičan. Kao i u prethodnim primjerima u isječcima koda, glavna razlika je u predlošcima gdje se može uočiti sintaksne razlike i složenost datoteke.

„Server.js“ datoteka u kojoj se najprije definiraju međuovisnosti, template engine te API ključ koji je potreban za dohvaćanje podataka:

```
const express = require('express');
const bodyParser = require('body-parser');
const request = require('request');
const app = express()

const apiKey = 'aa703b8e22029b460fdf6a2ce2834de7';

app.use(express.static('public'));
app.use(bodyParser.urlencoded({ extended: true }));
app.set('view engine', 'ejs')
```

Osnovni GET zahtjev za početnu stranicu:

```
app.get('/', function (req, res) {
  res.render('index', {weather: null, sunrise: null, sunset: null, error: null});
})
```

POST zahtjev koji u slučaju uspjeha iscrtava predložak „result“ gdje se stranica ispunjava dohvaćenim podacima:

```
} else {
  let sunrise = format_time(weather.sys.sunrise);
  let sunset = format_time(weather.sys.sunset);
  let weatherText = `Temperatura je ${Math.round(weather.main.temp)} stupnjeva u ${city}!`;
  let sunriseText = `Sunce izlazi u ${sunrise}`;
  let sunsetText = `Sunce zalazi u ${sunset}`;

  res.render('result', {weather: weatherText, sunrise: sunriseText, sunset: sunsetText, error: null});
}
```

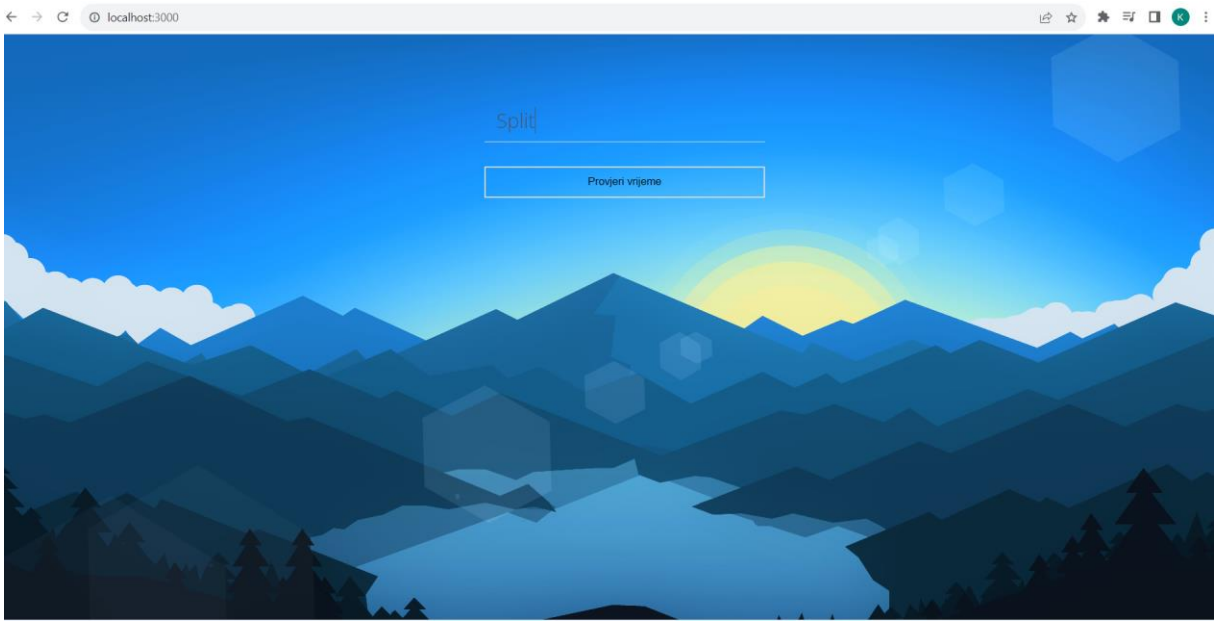
Sljedeće dvije slike su kod predloška „index“ napisanog koristeći EJS te zatim koristeći Pug.

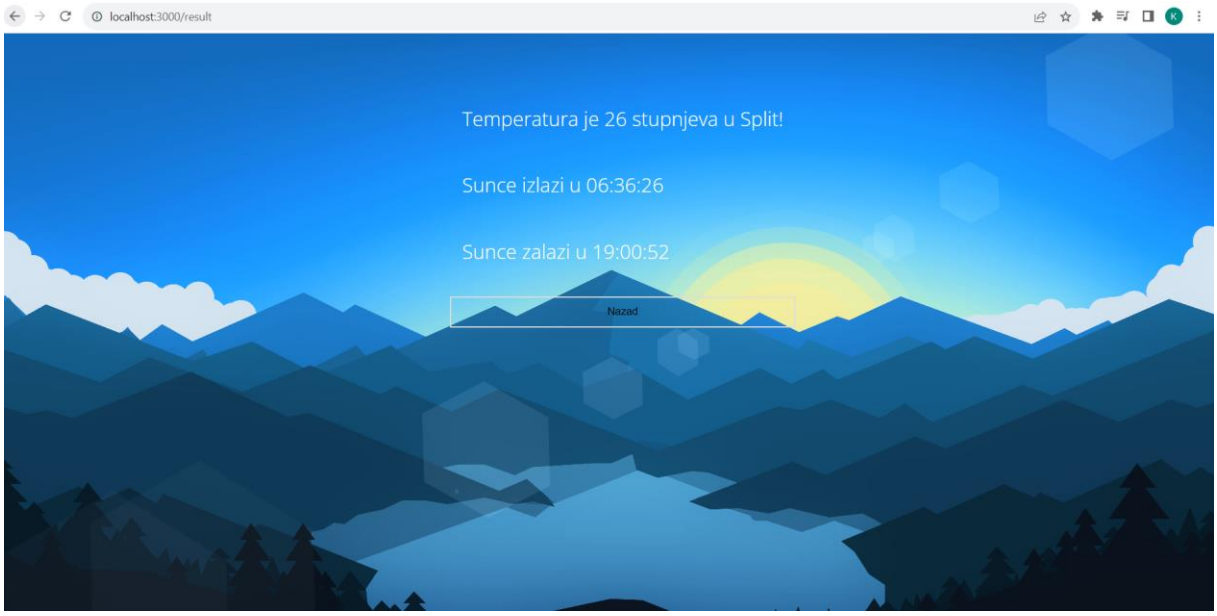
```ejs
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>EJS</title>
    <link rel="stylesheet" type="text/css" href="/css/style.css">
    <link href='https://fonts.googleapis.com/css?family=Open+Sans:300' rel='stylesheet' type='text/css'>
  </head>
  <body>
    <div class="container">
      <fieldset>
        <form action="/result" method="post">
          <input name="city" type="text" class="ghost-input" placeholder="Unesite ime grada" required>
          <input type="submit" class="ghost-button" value="Provjeri vrijeme">
        </form>
        <% if(error !== null){ %>
          <p><%= error %></p>
        <% } %>
      </fieldset>
    </div>
  </body>
</html>
```

```pug
doctype html
html
  head
    meta(charset='utf-8')
    title PUG
    link(rel='stylesheet', type='text/css', href='/css/style.css')
    link(href='https://fonts.googleapis.com/css?family=Open+Sans:300', rel='stylesheet', type='text/css')
  body.view
    fieldset
      form(action='/result', method='post')
        input.ghost-input(name='city', type='text', placeholder='Unesite ime grada', required='')
        input.ghost-button(type='submit', value='Provjeri vrijeme')
      if error
        p= error
```

Evidentno je da je predložak napisan koristeći Pug kraći i pregledniji. Bez prikazivanja koda za predloške „result“ dovoljno je spomenuti razlike u broju linija koda – predložak napisan koristeći EJS sadrži 34 linije koda dok ekvivalent u Pug-u sadrži 19. No, uvlačenje redaka koje je zaslužno za skraćenje koda također zna stvarati probleme pri rješavanju grešaka u kodu budući da je puno teže pronaći retke u kojima se iste nalaze.

Sama web aplikacija se uspješno prikazuje i radi na strani klijenta. Dohvaćeni podaci iscrtavaju se ispravno zahvaljujući upotrebi predložaka što ne bi bilo moguće koristeći isključivo HTML datoteke.


localhost:3000
Split
Provjeri vrijeme


localhost:3000/result
Temperatura je 26 stupnjeva u Split!
Sunce izlazi u 06:36:26
Sunce zalazi u 19:00:52
Nazad

4. SAŽETAK

SSR u okruženjima baziranim na JavaScript tehnologijama nudi kvalitetno rješenje za stvaranje dinamičnih, učinkovitih web aplikacija. Nudi mogućnost optimizacije brzine učitavanja stranica, pruža bolje iskustvo korisnicima sa sporijim uređajima ili ograničenom snagom procesiranja i održava dosljednost u različitim preglednicima tako što se web stranice iscrtavaju na poslužitelju prije nego što se isporuče klijentu.

SSR bi se trebao smatrati vrijednim alatom za projekte s velikom količinom dinamičkog sadržaja ili kompleksnom logikom na back-endu. Može pružiti značajne prednosti u izvedbi, korisničkom iskustvu i optimizaciji za tražilice ako se vodi računa o pravilnom dizajnu i implementaciji. Uz teoretsku analizu, ovaj rad je ponudio i praktične primjere ovog pristupa na osnovu kojih se može zamisliti korisnost pristupa za konkretnije, kompleksnije projekte.

Uz sve navedeno naravno opet stoji činjenica da odabir pristupa pri razvoju aplikacije ovisi o faktorima kao što su struktura i kompetencije razvojnog tima, zahtjevi i potrebe projekta koji se protežu od tipa i kompleksnosti aplikacije sve do količine korisnika ili čak i vrste korisnika. Nažalost (ili srećom), u području razvoja web aplikacija nije moguće dati jedinstveni odgovor adekvatan za svaki kontekst i situaciju. Za svaki projekt potrebno je donijeti jedinstvene odluke bazirane na specifičnim zahtjevima i izazovima koji se pojavljuju uz isti.

5. LITERATURA

Knjiga:

Eyskens, S. (2021). Software Architecture for Busy Developers. O'Reilly

Thakkar, M. (2020). Building React Apps with Server-Side Rendering. O'Reilly

Hahn, Evan M. (2016). Express in Action. Manning

Mrežna stranica:

Anonymus (2023). An overview of HTTP. On-Line URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview Mozilla Developer. Pristupljeno 11.9.2023.

Harris, C. (2020). Microservices vs monolothic architecture. On-Line URL: https://www.atlassian.com/microservices/microservices-architecture/microservices-vs-monolith Atlassian. Pristupljeno 11.9.2023.

Skólski, P. (2016). Single-page application vs. multiple-page application. On-Line URL: https://medium.com/@NeotericEU/single-page-application-vs-multiple-page-application-2591588efe58 Medium. Pristupljeno 11.9.2023.

Sydorkina, A. (2023). Key Differences Between Client-Side, Server-Side and Pre-rendering. On-Line URL: https://clockwise.software/blog/client-side-vs-server-side-vs-pre-rendering/ Clockwise Software. Pristupljeno 11.9.2023.

Galvez, J. (2022). A Gentle Introduction to SSR. On-Line URL: https://hire.jonasgalvez.com.br/2022/apr/30/a-gentle-introduction-to-ssr/. Pristupljeno 11.9.2023.

Jiang, A. (2023). The Future (and the Past) of the Web is Server Side Rendering. On-Line URL: https://deno.com/blog/the-future-and-past-is-server-side-rendering. Pristupljeno 11.9.2023.

Miller, J., Osmani, A. (2023). Rendering on the Web. On-Line URL: https://web.dev/rendering-on-the-web/. Web Dev. Pristupljeno 11.9.2023.

Anonymus (2023). Template engines. On-Line URL: https://expressjs.com/en/resources/template-engines.html Express. Pristupljeno 11.9.2023.

Anonymus (2023). Pug. On-Line URL: https://pugjs.org/api/getting-started.html Pug. Pristupljeno 11.9.2023.

Anonymus (2023). EJS. On-Line URL: https://ejs.co/ EJS. Pristupljeno 11.9.2023.