Razvoj Web aplikacije korištenjem React i Spring Boot programskih okvira

Galina, Patrik

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Patrik Galina

RAZVOJ WEB APLIKACIJE KORIŠTENJEM REACT I SPRING BOOT PROGRAMSKIH OKVIRA

DIPLOMSKI RAD

Varaždin, 2022.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Patrik Galina

Matični broj: 46230/17-R

Studij: Informacijsko i programsko inženjerstvo

RAZVOJ WEB APLIKACIJE KORIŠTENJEM REACT I SPRING BOOT PROGRAMSKIH OKVIRA

DIPLOMSKI RAD

Mentor/Mentorica:

Prof. dr. sc. Danijel Radošević

Varaždin, travanj 2022.

Patrik Galina

Izjava o izvornosti

Izjavljujem da je moj završni/diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor/Autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

Najpoznatiji te najučestaliji oblik razmjene dobara i usluga za platežno sredstvo, odnosno novac ili drugu robu, u današnje vrijeme, su trgovine koje svoje korijene vuku od 3000.-te godine pr. Kr. Trgovinu možemo definirati kao gospodarsku djelatnost u kojoj pojedinci i tvrtke, odnosno fizičke i pravne osobe, posreduju između proizvodnje i potrošnje kao kupci i prodavatelji dobara i usluga, a samim time i organizatori tržišta. Trgovina kao zasebna djelatnost omogućuje najbržu i najuspješniju povezanost između proizvođača i potrošača. Iako trgovine u samim svojim počecima ne možemo poistovjetiti s današnjim, modernim trgovinama, one su svakako dobar uvod i oslonac u cijeli koncept trgovine. Vjeruje se kako je trgovina najstarija ljudska djelatnost koja se odvija tijekom većeg dijela ljudske povijesti. Osim tradicionalnih trgovina, odnosno trgovina koje se odvijaju uživo na unaprijed predviđenim, fizičkim lokacijama, u današnje vrijeme, obilježeno pandemijom, sve značajnije postaju virtualne trgovine, odnosno elektroničke trgovine koje se odvijaju na virtualnim lokacijama. Poslovni proces elektroničke trgovine se u svojoj osnovi ne razlikuju od poslovnog procesa klasične trgovine, međutim fleksibilnost poslovnog procesa elektroničke trgovine je ono što ih čini sve popularnijim te učestalijim oblikom trgovanja. Elektroničke trgovine otvorile su nove mogućnosti prodaje, odnosno kupnje dobara ili usluga kako za kupce tako i za prodavače.

Tema ovog diplomskog rada je teorijska konceptualizacija procesa razvoja Web aplikacija kroz sistematizaciju tehnologija koje se pritom koriste kao i ključnih koncepata Web-a. Mogućnosti tehnologija za implementaciju Web aplikacija pobliže su objašnjene njihovom detaljnom razradom te pregledom programskih primjera koji su izrađeni korištenjem programskih jezika JavaScript te Java. U uvodnom dijelu rada opisani su ključni aspekti Web-a koji imaju značaj za razvoj Web aplikacija kao i njihov povijesni razvoj. U nastavku su sistematizirane tehnologije koje se koriste prilikom razvoja Web aplikacija te je objašnjen koncept komunikacije u realnom vremenu kao jedan od ključnih koncepata prilikom razvoja modernih Web aplikacija. Rad obuhvaća i praktični dio, odnosno praktičnu primjenu teorijskih koncepata teme, a u sklopu kojeg je izrađena Web aplikacija koja krajnjim korisnicima omogućava kupnju, odnosno naručivanje proizvoda putem Web-a.

Ključne riječi: elektronička trgovina; Web aplikacija, React; Spring Boot; Web servisi; Payment Gateway

Sadržaj

1. Uvod

U današnjem modernom, digitalnom svijetu, a izrazito u doba prisutnosti globalne pandemije te globalizacije, cijeli koncept virtualnog poslovanja, pa tako i elektroničke trgovine, strelovito se brzo razvija. Provedene studije na temu utjecaja pandemije na globalno poslovanje pokazale su da se većina poduzeća reaktivno prilagodila novonastaloj situaciji te da se njihovo poslovanje nastavilo u istom tonu. U novonastaloj situaciji važno je da poduzeća kroz digitalnu transformaciju poslovanja prilagode svoje poslovne procese okvirima nove realnosti, odnosno "novog normalnog" kako bi mogle opstati na tržištu koje je izrazito dinamično te se neprestano mijenja. U cijelu tu priču ulazi i koncept virtualnog trgovanja, odnosno elektroničke trgovine. Naime, vlasnici fizičkih trgovina moraju provesti digitalnu transformaciju poslovanja kako bi opstali na tržištu u ovom vrlo zahtjevnom okruženju te održali postojeću poziciju u tržišnoj utakmici. Upravo je Web omogućio pojednostavljenje procesa trgovanja kroz Web trgovine koje krajnjeg korisnika oslobađaju od odlazaka u fizičke trgovine kako bi kupio dobro ili uslugu od interesa. Općenito, koncept elektroničke trgovine se temelji na neprekidnoj dostupnosti usluge te ideji pojednostavljenja procesa kupnje dobara ili usluga, kako za krajnje kupce tako i za prodavače. Stahl u svojoj knjizi *E-Commerce-Leitfaden* definira elektroničku trgovinu kao afirmiranu komponentu gospodarstva sa stabilnim obujmom kupnje, odnosno prodaje te visokom razinom prihvaćenosti od strane krajnjih kupaca i prodavača. [1] Osnove funkcionalnosti svake elektroničke trgovine su košarica za kupnju te procesiranje narudžbe. Pritom je važno spomenuti kako opseg funkcionalnosti elektroničke trgovine ponajviše ovisi o poslovnom modelu te grani gospodarstva unutar koje trgovina djeluje. Web trgovine variraju od sustava koji omogućava jednostavnu funkcionalnost košarice te kreiranja narudžbe za odabrane proizvode pa sve do prilagodljivih sustava koji omogućavaju integraciju s postojećim sustavima unutar poduzeća kako bi omogućili kontrolu zaliha te automatizirali logističke poslove. Elektroničke trgovine zahtijevaju visoku razinu sigurnosti te očuvanja povjerljivosti korisničkih podataka, što je posebno važno prilikom provođenja transakcija. Elektroničke trgovine postaju važnije no ikad prije, što je prvenstveno uzrokovano globalnom pandemijom koja je vlasnike poduzeća natjerala na promjene u poslovanju.

Cilj ovog rada je opisati teorijske aspekte procesa razvoja Web aplikacija kroz sistematizaciju tehnologija koje se pritom koriste, kao i prikaz osnovnih koncepata WWW-a (engl. *World Wide Web*) te provođenja transakcija u Web okruženju. U okviru praktičnog dijela rada bit će detaljno prikazan proces razvoja Web aplikacije koja krajnjim korisnicima omogućava kupnju dobara, dok s druge strane prodavaču omogućava prodaju te promoviranje svojih dobara. Motivacija za izradu ovog rada proizašla je iz promatranja aplikacijskih domena vezanih uz područje elektroničke trgovine te mogućih napredaka unutar spomenutog područja.

2. Istraživački problem

Bezić u svom znanstvenom članku *Elektronička trgovina u malim i srednjim poduzećima Republike Hrvatske* navodi kako su brzi napredak informacijsko-komunikacijskih tehnologija, rasprostranjenost računalnih mreža te sve šira uporaba Interneta temelj nove industrijske revolucije. Pritom elektronička trgovina predstavlja novu, suvremenu mogućnost stjecanja konkurentske prednosti kako na domaćem tržištu, tako i na globalnoj razini. [2] Međutim, iako bi se iz ovog moglo zaključiti kako je elektronička trgovina savršena zamjena tradicionalnom načinu trgovanja, ona u stvarnosti ima i svojih problema. Prije svega, pojedino dobro koje se prodaje putem elektroničke trgovine može ostaviti bolji dojam na krajnjeg kupca ukoliko ono ne reprezentira njegovu stvarnu sliku. Na taj je način elektronička trgovina, u samim svojim početcima, doživljavala velik neuspjeh uzrokovan nezadovoljstvom krajnjih korisnika. Međutim, suvremene tehnologije omogućile su 3D prikaz dobara te na taj način njihovu realističniju reprezentaciju čime je korisničko iskustvo elektroničkog trgovanja podignuto na jednu višu razinu što je utjecalo na smanjenje razine nezadovoljstva korisnika.

Posljednjih nekoliko godina svjedočimo učestalom, mogli bismo reći svakodnevnom, razvoju tehnologija. Tradicionalni oblici plaćanja, kao i obavljanja transakcija, odlaze u povijest, a zamjenjuju ih njihovi elektronički ekvivalenti koji na vrlo brz i efikasan način obavljaju zadatak koji se od njih zahtijeva. Dugi redovi čekanja u fizičkim trgovinama zamijenjeni su kupnjom iz vlastitog doma čime se značajno olakšava svakodnevica krajnjih kupaca te poboljšava njihovo korisničko iskustvo prilikom kupovine. Web trgovanje se tijekom godina afirmiralo što je dovelo do stvaranja virtualnog tržišta koje u današnje, suvremeno, ali istovremeno užurbano, doba predstavlja najbrži i najefikasniji način provođenja transakcija. Razna istraživanja dovela su do zaključaka kako je tehnologija promijenila svjetsku ekonomiju i potrošače te da je značajno utjecala na poslovne procese unutar poduzeća. Elektroničko poslovanja, odnosno digitalna ekonomija, postavlja ciljeve koji ističu način na koji znanje postaje fokus poslovanja, a dijeljenje informacija glavni adut u rukavu. [4]

Jedno od ključnih pitanja vezanih uz područje razvoja elektroničke trgovine koje se svakodnevno nameće svakako je sigurnost transakcija prilikom obavljanja kupnje putem Interneta. U tu priču ponovno ulaze suvremene tehnologije koje su omogućile zadovoljavajuću razinu sigurnosti prilikom obavljanja transakcija te samim time utjecale na porast korištenja usluga elektroničke trgovine. Najvažnija od njih je Payment Gateway koji je bitno utjecao na sigurnost te provođenje transakcija putem Interneta. Payment Gateway je u kontekstu Internet trgovine ono što je POS (engl. *Point of Sale*) uređaj u tradicionalnim oblicima trgovine. [5] Drugim riječima, Payment Gateway je posrednik između krajnjeg korisnika, prodavača i banke koji osigurava sigurnost prilikom obavljanja internetskih transakcija.

3. Metode i tehnike rada

Prvi dio ovog rada bavi se pregledom osnovnih koncepata vezanih uz elektroničku trgovinu, od njezine povijesti, terminologije pa sve do njezinog razvoja tijekom godina. Osim toga, u prvom dijelu rada dana je teorijska konceptualizacija procesa razvoja Web aplikacija kroz sistematizaciju tehnologija koje se pritom koriste te su sagledani osnovni koncepti Web-a. Glavne metode istraživanja i analize podataka, koje su korištene kako bi se determinirali pojedini dijelovi stručne terminologije, su metode analize, sinteze, indukcije te deskripcije. [11] Kao osnova baze znanja prilikom sistematizacije stručne terminologije prvog dijela rada korištena je znanstvena i stručna literatura, koja je relevantna s obzirom na problematiku kojom se rad bavi, koja je dostupna na Google Scholar-u te portalu Hrčak. Teorijski dio rada sadrži pregled elemenata i koncepata koji su usko vezani uz aplikacijsku domenu. Sistematizacija teorijskih koncepata konceptualizirana je kroz poglavlja pri čemu svako poglavlje definira širi spektar pojedinog dijela koncepta na kojem će se temeljiti kasniji razvoj aplikacije. U uvodnom dijelu teorijskog dijela rada obrađena je kratka povijest elektroničke trgovine te su prikazani i definirani osnovni koncepti elektroničke trgovine kao i najbitniji pojmovi koji se vežu uz sam termin elektroničke trgovine. U nastavku su definirani i opisani modeli elektroničke trgovine koji se uobičajeno koriste u današnje vrijeme te je dan kratak osvrt na digitalnu transformaciju poslovanja primjenom koncepata elektroničke trgovine. U nastavku slijedi poglavlje vezano uz Internet u kojem se navodi njegova kratka povijest te najvažnije tehnologije na kojima se temelji razvoj modernih aplikacija, od HTTP-a pa sve do Web servisa, API-a te programskih okvira.

Nakon detaljne razrade osnovnih koncepata i terminologije vezane uz teorijski dio same teme, slijedi i praktični dio rada u kojem je sukcesivno opisan proces razvoja Web aplikacije. U okviru praktičnog dijela rada bit će implementirana Web aplikacija koja će predstavljati virtualnu trgovinu koja će krajnjim korisnicima omogućiti kupnju bicikala, dijelova i opreme za bicikle kao i rezervaciju električnih bicikala, pri čemu se poseban naglasak stavlja na maksimalno zadovoljavanje korisničkih potreba, a samim time i poboljšanje korisničkog iskustva prilikom korištenja elektroničke trgovine. Prilikom razvoja Web aplikacije korišteni se programski jezici JavaScript i Java, točnije programski okviri React, za frontend dio, te Spring Boot, za backend dio Web aplikacije. Osim toga, korišten je Hibernate kako bi se omogućilo objektno-relacijsko mapiranje podataka. Drugim riječima, Hibernate pruža okvir koji omogućava mapiranje objektno orijentiranog modela podatkovne domene u model relacijske baze podataka. Kao sustav za upravljanje bazom podataka korišten je PostgreSQL koji se nameće kao idealan alat upravo zbog otvorenosti koda, jednostavnosti korištenja te velike potpore zajednice.

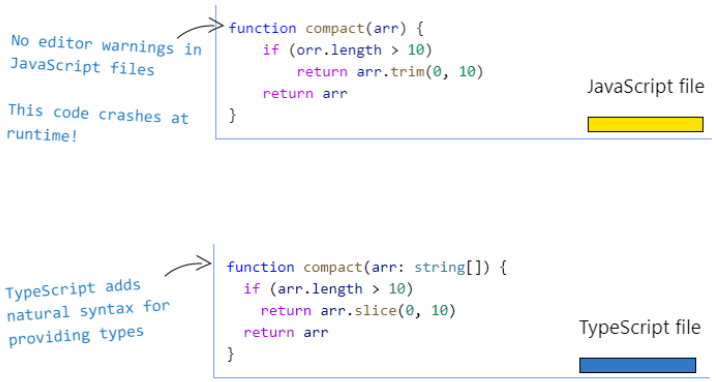
3.1. TypeScript

TypeScript je skriptni programski jezik temeljen na JavaScript-u koji je razvijen 2012.-te godine od strane Microsoft-a. Riječ je o programskom jeziku koji predstavlja generalizaciju JavaScript programskog jezika definirajući pritom strogo sintaktički skup pravila koja omogućuju statičku provjeru tipova podataka. Provjera tipova podataka tijekom kompiliranja omogućava standardizaciju i tipizaciju programskog jezika koja na taj način omogućuje jednostavnije i brže pronalaženje pogrešaka u kodu. Kako TypeScript predstavlja generalizaciju JavaScript-a, njegova primjena nije ograničena na stranu klijenta ili na stranu poslužitelja. [prema 69] Kao što je i ranije spomenuto, TypeScript nadograđuje programski jezik JavaScript uvođenjem sintakse koja omogućuje statičko tipiziranje. TypeScript na taj način provjerava programski kod te obavještava programera o svim pogreškama vezanim uz programski kod, od nazivlja varijabli pa do provjera usklađenosti jednostavnih i složenih tipova podataka. Nakon pokretanja aplikacije, kompajler prevodi TypeScript tipiziranu sintaksu u čisti JavaScript kod. [prema 69] Pritom se koristi standardni TypeScript kompilator ili Babel kompilator programskog koda. Dodatna sintaksa koju pruža TypeScript u odnosu na JavaScript omogućava čvršću integraciju s modernim uređivačima programskog koda kao što je Visual Studio Code. TypeScript datoteke moguće je prepoznati prema *.ts* ekstenziji

U svijetu informacijsko-komunikacijskih tehnologija vrlo često se nameće pitanje:

„Za koga je osmišljen programski jezik TypeScript?“

Kako nije riječ o potpuno novom programskom jeziku već o striktnijoj varijanti JavaScript programskog jezika, svaki JavaScript *developer* može vrlo jednostavno i brzo započeti s korištenjem TypeScript-a. Od velike je pomoći ukoliko više osoba radi na istome projektu ili je potrebno adaptirati stari kod postojećeg legacy sustava. Naravno, za trenutno aktivne projekte to će značiti dodatan posao kako bi se obavio prelazak na TypeScript. [70] Međutim, prelaskom na TypeScript osigurava se dugoročna stabilnost i pouzdanost programskog koda čime se znatno pospješuje i olakšava proces održavanja aplikativnog sustava.



Slika 1. Primjer funkcije u JavaScript-u i TypeScript-u [prema 69]

3.2. Java

Java je višeplatformski, objektno-orijentirani programski jezik te računalna platforma koja je prvi put objavljena 1995.-te godine od strane Sun Microsystems. Osnovna ideja programskog jezika Java je reducirati ovisnost o implementaciji kako bi se omogućila izgradnja različitih vrsta aplikacija, od mobilnih, pa sve do desktop te Web aplikacija. Od skromnih i manje značajnih početaka, Java se afirmirala kao programski jezik koji pokreće današnji digitalni svijet kroz pouzdanu platformu koja omogućuje izgradnju suvremenih usluga i aplikacija. [75] Java platforma je zbirka programa koja programerima omogućava brzo, jednostavno i učinkovito razvijanje i pokretanje aplikacija. [prema 76] Popularnost Jave u današnje vrijeme očituje se mnoštvom opcija koje pruža prilikom razvoja aplikacija. Uz pomoć Jave moguće je razvijati mobilne aplikacije, Web aplikacije, Enterprise aplikacije, sustave za Big Data analitiku, a osim toga, koristi se za programiranje različitih vrsta ugrađenih uređaja.

Radi se o programskom jeziku visoke razine koji je jednostavan, neovisan o platformi, objektno-orijentiran, prenosiv, distribuiran, skalabilan te siguran. Java omogućava dinamičko povezivanje koda primjenom rekurzije te koncepata generičkog programiranja. Java koristi koncept automatskog upravljanja radnom memorijom uz pomoć *garbage collector-a* koji omogućava dealociranje onih dijelova memorije koji više nisu u upotrebi. Osim svega navedenog, radi se o programskom jeziku visokih performansi koji pruža mogućnost višedretvenog rada (engl. *multithreading*).

U literaturi se vrlo često spominje akronim WORA (engl. *write once, run anywhere*) koji označava pisanje koda samo jednom, dok se taj isti kod može izvršavati na različitim uređajima i platformama. [prema 75] Neovisnost o platformi realizirana je na način da kompajler iz *.java* datoteke koja sadrži izvorni kod, umjesto strojnog koda, generira *bytecode*, u obliku datoteke s ekstenzijom *.class*. Bytecode se izvršava u Java virtualnom stroju, odnosno JVM-u (engl. *Java Virtual machine*), koji se temelji na interpreteru platforme i operacijskog sustava na kojem se želi izvršavati Java aplikacija. [77]

Slika 2. Proces generiranja .class datoteke [78]

4. Elektronička trgovina

4.1. Povijest elektroničke trgovine

Postoje razni dokazi koji upućuju na to kako trgovina svoje korijene vuče iz prapovijesnog kamenog doba. Tu zapravo govorimo o trampi[1] opsidijana (vulkanskog stakla) te kremena, pri čemu se trgovina temeljila na zamjeni robe za robu pošto u to vrijeme još nije razvijen novac niti njegov ekvivalent. Osim toga, poznato je da se u Egiptu 3000. godine pr. Kr. trgovalo materijalima koji su korišteni za izradu nakita. U istom povijesnom razdoblju, Sumerani počinju trgovati s Harapan civilizacijama te se na taj način razvijaju duge trgovačke rute. Uz povijest trgovine vrlo često se veže pojam Put svile koji je odigrao vrlo važnu povijesnu ulogu te bio glavni distribucijski kanal svile iz kineske civilizacije prema Zapadu. [prema 6]

S druge pak strane, elektronička trgovina svoje korijene vuče iz kraja 20. stoljeća te je njezina povijest daleko bolje popraćena. Razvoj Web trgovine započinje 1960. godine, a njezinom neprekidnom razvoju svjedočimo i danas. Ed Guilbert je u tom periodu razvio EDI (engl. *Electronic Data Interchange*) sustav koji je omogućio prijenos podataka u strukturiranom obliku između većeg broj računalnih sustava. Bjelić u svom završnom radu navodi kako se ocem elektroničke trgovine smatra Michael Aldrich koji je 1979. godine demonstrirao prvi sustav kupovine na daljinu putem kojeg su kupci, gledajući televizijski program, mogli telefonski naručiti željeni proizvod koji bi ima zatim bio dostavljen na kućnu adresu. [8] To je ujedno i prvi sustav trgovine posredstvom digitalnog medija čime se bitno mijenjaju pogledi na tada dostupne tehnologije. Međutim, Markoff u svojoj knjizi *What the Dormouse Said* kao prekretnicu trgovanja te začetak elektroničke trgovine smatra 1971. godinu kada su studenti dvaju sveučilišta, SAIL (engl. *Stanford Artificial Intelligence Laboratory*) i MIT (engl. *Massachusetts Institute of Technology*) iskoristili ARPANET (engl. *The Advanced Research Projects Agency Network*), preteču Interneta, kako bi ugovarali prodaju kanabisa. [9] Međutim, Elektronička trgovina kakvu danas poznajemo doživjela je svoj veliki napredak 1991. godine kada je Tim Berners-Lee upogonio World Wide Web te na taj način omogućio elektroničko trgovanje u punom smislu te riječi. Prvu elektroničku trgovinu ''otvorio'' je Amazon, 1995. godine, koji je iskoristio mogućnosti Web-a kako bi prodavao knjige. Nedugo zatim, 1998. godine, nastao je eBay koji je danas jedna od najuspješnijih elektroničkih trgovina. Osim toga, 1999. godine pokrenuta je elektronička trgovina pod nazivom Zappos koja nije imala trgovina na fizičkim lokacijama već se isključivo orijentirala na prodaju putem Web-a. [10]

[1] Trampa – proces razmjene jednog dobra za neko drugo dobro bez posredstva novca ili novčanih ekvivalenata. [prema 7]

4.2. Terminologija elektroničke trgovine

4.2.1. Definicija elektroničke trgovine

Razvojem World Wide Web-a, 1991. godine, dolazi do novih pogleda na informacijsko-komunikacijske tehnologije te mogućnosti koje one pružaju. Internet je masovni medij suvremenog doba koji je značajno utjecao na društvo u cjelini te se upravo zbog toga smatra najraširenijim sredstvom razmjene informacija. Rasprostranjenost Interneta kao masovnog medija utjecala je na različita područja ljudskog djelovanja mijenjajući pritom način komuniciranja, informiranja, obrazovanja te kupovine. Ružić u svojoj knjizi *E-marketing* navodi da je Internet uvelike promijenio način poslovanja, prvenstveno zbog činjenice da suvremeni komunikacijski alati, koje Internet nudi, pružaju mnoge, suvremene mogućnosti komuniciranja što je posebno zamjetno u području marketinga. [prema 12] Internet se, kroz godine, sukcesivno razvio u jedan od glavnih kanala prodaje i distribucije dobara i usluga te se na taj način etablirao kao platforma za upravljanje odnosima s kupcima te istraživanje tržišta.

Termin elektroničke trgovine u današnje, suvremeno, doba susrećemo gotovo na dnevnoj razini. Širenjem dostupnosti Interneta kao i ubrzanim razvojem tehnologija dolazi do otvaranja novih mogućnosti trgovanja te se trgovanje sve većim dijelom seli u virtualni svijet. Različiti izvori i autori iznose različite definicije termina elektroničke trgovine.
Matić u svojoj knjizi *Međunarodno poslovanje* definira termin elektroničke trgovine kao pojam koji označava kupnju i prodaju dobara i usluga te transfer novčanih sredstava posredstvom digitalne komunikacije pri čemu se koriste digitalne kartice, digitalna gotovina te svi drugi načini poslovanja dostupni putem Interneta. [prema 3]

Elektronička trgovina obuhvaća razmjenu poslovnih informacija, održavanje poslovnih odnosa i obavljanje poslovnih transakcija putem telekomunikacijskih mreža. Nju čine sve komercijalne transakcije koje se odvijaju putem otvorenih mreža te uključuje obradu i razmjenu digitaliziranih podataka, između pravnih osoba te razmjenu između poduzeća i krajnjih potrošača. [2]

Čerić pak u svom članku *Internet Economy and Electronic Commerce* navodi kako elektronička trgovina obuhvaća prodaju dobara i usluga putem komunikacijskih mreža. Ona u sebi integrira mnoštvo funkcija za podršku samoj funkciji prodaje, kao što su marketing, proizvodnja te isporuka. [prema 13]

Iz navedenih definicija vidljivo je kako elektronička trgovina obuhvaća širok spektar različitih, te pritom međusobno komplementarnih domena, od Internet bankarstva, Internet trgovine pa sve do različitih oblika promocije dobara i usluga na društvenim mrežama. Sveprisutnost elektroničke trgovine dovela je do toga da na tržištu prosperiraju samo suvremena poduzeća koja koriste sofisticirane i suvremene informacijsko-komunikacijske tehnologije.

4.2.2. Elektronička gotovina

Nacionalni CERT (engl. *Computer Emergency Response Team*) definira pojam elektroničke gotovine kao jedan od načina ostvarivanja elektroničkog oblika plaćanja koji se pojavio kao posljedica širenja dostupnosti Interneta te mogućnosti koje pružaju računalne mreže. [18] Elektronička gotovina je sredstvo koje omogućava obavljanje transakcija putem Interneta na brz i jednostavan način. Naknada za provođenje transakcije korištenjem elektroničke gotovine znatno je manja u odnosu na naknadu koja se naplaćuje u raznim bankarskim poslovnicama. Elektronička gotovina, za razliku od papirnatog novca nije prenosiva. Obična novčanica primljena u jednoj od prethodnih transakcija može se ponovno upotrijebiti u nekoj od sljedećih. Ona je prenosiva i traje više od jedne transakcije. Takvo bi svojstvo bilo vrlo poželjno za elektroničke novčanice jer se pri svakoj transakciji novčanica ne bi trebala pohranjivati u banku, smanjujući tako broj interakcija s bankom, a time i troškove sustava. [18] Iz svega navedenog, vidljivo je da elektronička gotovina čuva anonimnost osobe koja je koristi kao sredstvo plaćanja te je samim time nemoguće pratiti njezin tijek pretvorbe.

4.2.3. Elektronički novčanik

Kod tradicionalnih oblika elektroničke trgovine plaćanje se obavlja autentifikacijom te verifikacijom korisnika na temelju unesenih podataka s kreditne kartice – *ime*, *prezime*, *broj kartice*, *sigurnosni kod* te *datum valjanosti kartice* unutar polja odgovarajuće forme. Repetitivni unos poveće količine podataka prilikom obavljanja elektroničke trgovine često je uzrokovao odustajanje krajnjih korisnika od plaćanja kreirane narudžbe. U svrhu smanjenja repetitivnih unosa korisnika, a posljedično i poboljšanja korisničkog iskustva prilikom obavljanja kupovine, uveden je elektronički novčanik.

Elektronički novčanik je vrsta korisničkog profila koji je namijenjen za pohranu i zbrinjavanje elektroničkih novčanih sredstava, pri čemu se prijenos sredstava na elektronički novčanik obavlja putem terminala, bankovnih transfera ili na bilo koji drugi virtualni način. [17] Ružić u svojoj knjizi *E-marketing* navodi kako su u elektroničkom novčaniku pohranjeni svi podaci o kupcu koje je potrebno unositi prilikom obavljanja elektroničkog plaćanja. [16] Umjesto repetitivnih unosa podataka potrebnih za izvršavanje transakcije na različitim Web aplikacijama, pogodnost elektroničkog novčanika je da se navedena radnja izvršava samo jednom i to prilikom uspostavljanja usluge. Porastom dostupnosti te korištenja mobilnih uređaja, korisnici se sve više okreću korištenju usluga mobilnih elektroničkih novčanika. Revolut, KEKS Pay te Aircash samo su neke od mobilnih aplikacija koje pružaju usluge mobilnog elektroničkog novčanika

4.3. Modeli elektroničke trgovine

Elektronička trgovina obuhvaća skup procesa koji se koriste prilikom obavljanja poslovnih transakcija u elektroničkom obliku, pri čemu je skup aktivnosti i sudionika pojedinog procesa definiran u obliku modela. Modele razlikujemo sukladno kriteriju razmatranja, a tri osnovne skupine modela koje Žarković razlikuje su: *modeli prema kriteriju sudionika*, *modeli prema kriteriju proizvodnje i skladištenja* te *modeli prema kriteriju prodajnog mjesta*. [14]

Trendovi u elektroničkoj trgovini, kao i u ostalim srodnim granama, razvijaju se na svakodnevnoj razini što dovodi do fluktuacije velike količine podataka kao i stručnih termina. DeMatas navodi kako se vrlo lako oduševiti najnovijim trendovima koji vladaju u području elektroničke trgovine, ali bez poznavanja osnovnih načela te principa pojedinog trenda vrlo je lako naići na zid profitabilnosti. [prema 15] Samo područje elektroničke trgovine zahtijeva vrlo dobro poznavanje virtualnog tržišta te detaljno razrađen poslovni plan kako bi odabrani model poslovanja urodio plodom.

4.3.1. Modeli prema kriteriju sudionika

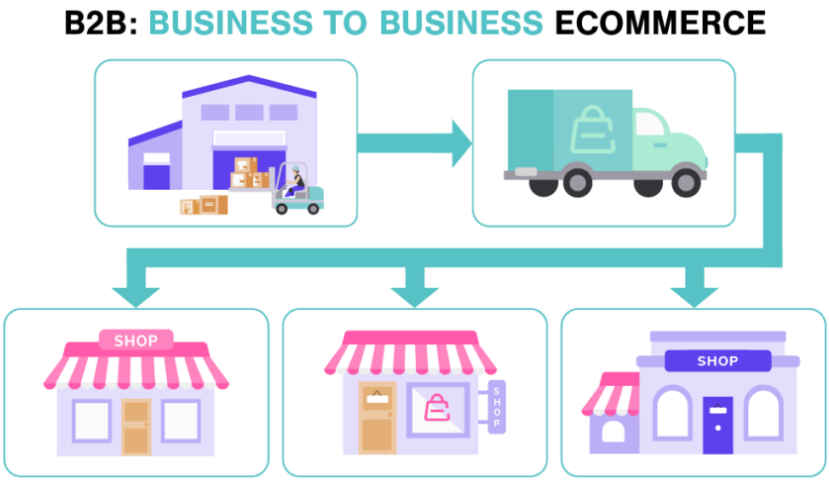
Svaki od modela elektroničke trgovine ove kategorije sadržava dvije međusobno ''sukobljene'' strane, odnosno sudionika, pri čemu se sudionici klasificiraju u jednu od standardiziranih kategorija – *potrošač* ili *poduzeće*, *Vlada*. Različiti autori definiraju različiti skup modela elektroničkog poslovanja, kao generalizacije elektroničke trgovine, međutim, osnovnih skup modela koje definira svaki od njih sastoji se od: B2B (engl. *Business to Business*), B2C (engl. *Business to Consumer*), C2C (engl. *Consumer to Consumer*), C2B (engl. *Consumer to Business*), B2G (engl. *Business to Government*), C2G (engl. *Consumer to Government*) te G2C (engl. *Government to Consumer*). Neki od autora definiraju i modele mobilne trgovine te P2P (engl. *Peer to Peer*) modele, koji su specifični za pojedine grane gospodarstva. U kontekstu elektroničke trgovine posebno su zanimljiva prva tri navedena modela, odnosno B2B (engl. *Business to Business*), B2C (engl. *Business to Consumer*), C2C (engl. *Consumer to Consumer*) koji su detaljnije opisani u nastavku.

4.3.1.1. B2B (engl. Business to Business) model

Business to Business, u neformalnom prijevodu ''poduzeće prema poduzeću'', model uključuje dva ili više poduzeća koja djeluju kao sukobljene strane. Drugim riječima, poduzeća se pojavljuju i kao prodavači i kao potrošači dobara i usluga. *Business to Business* model predstavlja elektroničku trgovinu dobrom ili uslugom, kao i njihovim kombinacijama, koje je od interesa za poduzeće koje djeluje na strani potrošača. Na taj se način poduzeća rasterećuju od poslovnih aktivnosti sa sukobljenim strana, odnosno za poduzeća se pojednostavljuje

proces obavljanja poslovnih aktivnosti vezanih uz transakcije s distributerima, dobavljačima te drugim poslovnim partnerima. Međutim, iako vrlo moćan, *Business to Business* model ima i svojih nedostataka. Prije svega, poduzeća su suočena s niskom razinom interoperabilnosti između sustava te je integracija takvih sustava vrlo kompleksna, a posebno iz razloga ostvarenja korektne komunikacije između njih. Spomenuti problem dovodi do razilaženja prilikom razmjene podataka između sustava koji bi trebali skladno komunicirati te na taj način automatizirati pojedine poslovne procese unutar organizacija. Iako je većina poduzeća, koja koriste ovaj model poslovanja, isključivo davatelj usluga, u današnje vrijeme, on se sve više veže uz IT poduzeća te srodne djelatnosti. [prema 15]

Prema DeMatas-u, Boeing i ExxonMobil najpoznatiji su primjeri poduzeća koja primjenjuju *Business to Business* model poslovanja.



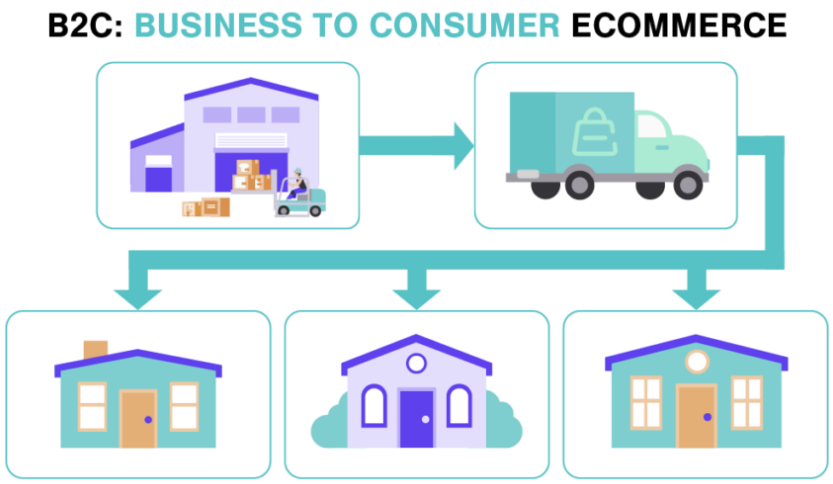
Slika 3. Business to Business model poslovanja [15]

4.3.1.2. B2C (engl. Business to Consumer) model

Business to Consumer, u neformalnom prijevodu "poduzeće prema kupcu", model uključuje stranu poduzeća kao ponuditelja dobara i usluga te stranu krajnjeg korisnika kao konzumenta dobara i usluga koje poduzeće nudi u okviru svog proizvodnog asortimana. U realnom sektoru te stvarnom životu, *Business to Consumer* model je ono na što većina ljudi pomisli prilikom razmatranja i definiranja pojma elektroničke trgovine. *Business to Consumer* model omogućava prodavačima da svoja dobra i usluge nude na elektroničkom tržištu koje obiluje brojem i raznolikošću potencijalnih krajnjih kupaca. Prodaja se uobičajeno obavlja putem Web aplikacije, a u novije vrijeme sve su popularnije i mobilne aplikacije upravo zbog svoje jednostavnosti i dostupnosti. Krajnji korisnici mogu vrlo jednostavno provjeriti cijene te karakteristike dobara i usluga u okviru Web ili mobilne aplikacije te kreirati narudžbu sukladno vlastitim preferencijama pri čemu se eliminira potreba komunikacije s prodajnim posrednikom što u globalu uvelike utječe na korisničko iskustvo. [prema 14]

Business to Consumer model pruža brojne mogućnosti i prednosti za poduzeća. Prije svega, poduzeće može ostvariti značajnu uštedu te bitno reducirati troškove pokretanja poslovanja ukoliko odabere *Business to Consumer* model. Naime, virtualno poslovanje ne zahtjeva zaposlenike kao ni troškove vezane uz fizički prostor čime se vrše značajne uštede.

Prema DeMatas-u, primjere ovog modela poslovanja susrećemo gotovo na dnevnoj bazi, a Wish te Overstock.com najpoznatiji su primjeri aplikacija koja primjenjuju te podupiru *Business to Consumer* model poslovanja.



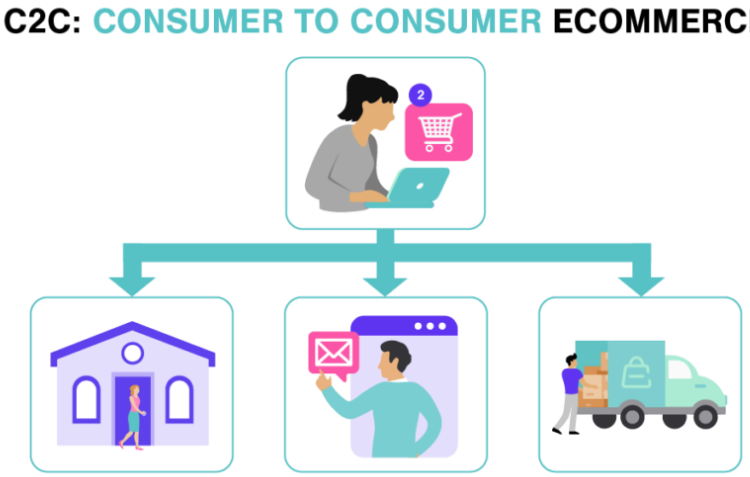
Slika 4. Business to Consumer model poslovanja [15]

4.3.1.3. C2C (engl. Consumer to Consumer) model

Consumer to *Consumer*, u neformalnom prijevodu "kupac prema kupcu", model uključuje dva ili više kupca koja djeluju kao sukobljene strane. Jednostavnije rečeno, kupci se pojavljuju i kao prodavači i kao potrošači dobara i usluga pri čemu se prodaja događa izravno između dvije sukobljenih strana, umjesto posredstvom prodajnog predstavnika. *Consumer* to *Consumer* model uobičajeno obuhvaća Web te mobilne aplikacije putem kojih korisnici mogu oglašavati vlastita dobra i usluge, dok istovremeno, potencijalni korisnici pretražuju dobra i usluge od interesa. Ovaj model poslovanja uobičajeno obuhvaća pregovaranje o cijeni te se konačna cijena dobra ili usluge određuje ostvarivanjem kompromisa između sukobljenih strana. *Consumer* to *Consumer* vrlo je popularan jer omogućuje brzu i jednostavnu prodaju diferenciranih dobara i usluga na jednom mjestu i bez potrebe za fizičkom odlaskom na aukcijska nadmetanja. Za razliku od *Business to Business* te *Business to Consumer* modela, ovaj model isključuje sudjelovanje prodajnih posrednika, odnosno omogućuje kupnju ili prodaju dobara i usluga izravno od vlasnika dobra ili pružatelja usluge.

U Republici Hrvatskoj postoji nekoliko aplikacija koje podržavaju *Consumer* to *Consumer* model, a među najpopularnijima su svakako Njuškalo.hr te Index oglasi. Međutim,

ovakav oblik poslovanja za sobom povlači i preuzimanje rizika. Naime, porastom kibernetičkog kriminala te prijevara značajno je porastao broj prijevara vezanih uz ovaj model poslovanja te se krajnji kupci sve više udaljavaju od ovakvog modela poslovanja. Porastom potražnje za *Consumer* to *Consumer* modelom poslovanja došlo je i do zaokreta poslovnog modela aplikacija koje podržavaju ovaj model poslovanja. Naime, sve veći broj aplikacija uvodi naplatu oglašavanja dobara i usluga u svoj bazični dio aplikacije te na taj način odvlači korisnike. Osim aplikacija koje podržavaju *Consumer to Consumer* model poslovanja, u novije vrijeme sve popularnije postaje i trgovanje putem društvenih mreža. Primjerice, Facebook je razvio Marketplace modul koji korisnicima omogućava oglašavanje vlastitih dobara i usluga.



Slika 5. Consumer to Consumer model poslovanja [15]

4.3.2. Modeli prema kriteriju proizvodnje i skladištenja

Modeli elektroničke trgovine ove kategorije sadržavaju koncepte vezane uz upravljanje zalihama, nabavku proizvoda i poluproizvoda te terminsko upravljanje proizvodnjom kako bi proizvodi bili dostupni u predodređeno vrijeme te na taj način ispunili korisnička očekivanja.

U modele prema kriteriju proizvodnje i skladištenja ubrajamo *posredovanje* (engl. *Dropshipping*), *veleprodaju i skladištenje* (engl. *Wholesaling and Warehousing*) te *rebrendiranje* (engl. *White Labeling*). [15]. Neki od navedenih modela imali su značajan utjecaj na razvoj elektroničke trgovine te bitno utjecali na njezinu globalnu rasprostranjenost, a posljednjih godina i sve veću popularnost. Prema DeMatas-u, modeli prema kriteriju proizvodnje i skladištenja doveli su do diferencijacije dviju vrsta osoba – *osobe koje simpatiziraju ideju vlastite proizvodnje* te *osobe koje ne simpatiziraju kreiranje zaliha proizvoda te suočavanje sa skladištem koje je prepunjeno proizvodima*. [prema 15]
U nastavku teksta korišteni su engleski nazivi pojedinog modela iz razloga jer u hrvatskom govornom području ne postoje njihovi općeprihvaćeni i standardizirani ekvivalenti termini.

4.3.2.1. Dropshipping

Ferreira u svom članku *What is Dropshipping?* definira pojam *Dropshipping* kao način maloprodaje u kojem elektronička trgovina nema skladište, a sukladno tome ni proizvode na zalihama, već ih, nakon što krajnji korisnik kreira narudžbu, naručuje od trećih strana i dostavlja izravno krajnjim kupcima. [prema 19] Pravna osoba, kao vlasnik elektroničke trgovine, na taj način "izbjegava" kontakt s proizvodom te djeluje kao posrednik između krajnjeg kupca te stvarnog vlasnika proizvoda ili proizvođača. Ono što diferencira *Dropshipping* model od standardiziranog modela elektroničke trgovine je skladištenje proizvoda. Kod *Dropshipping* modela vlasnik elektroničke trgovine ne posjeduje niti unajmljuje fizičke lokacije za skladištenje proizvoda već proizvode nabavlja sukladno trenutnoj potražnji. U suštini, osnovni koncept *Dropshipping* metode podrazumijeva da posrednik kreira narudžbu od treće strane, primjerice proizvođača, nakon što krajnji korisnik kreira narudžbu na Web trgovini posrednika. Posrednik, u kontekstu razmatrane tematike vlasnik elektroničke trgovine, na taj način izbjegava troškove skladištenja proizvoda kao i sve rizike koji iz toga proizlaze. Prednosti *Dropshipping* modela su mnogobrojne, od niskih početnih ulaganja, pa sve do ušteda vezanih uz nepostojanje troškova skladištenja te logističkih rizika. Međutim, iako vrlo popularan, ovaj model ima i svojih nedostataka koji se ponajprije manifestiraju u smanjenju ugleda elektroničke trgovine koja svoje poslovanje temelji na *Dropshipping* modelu. Naime, kako posrednik, u kontekstu naše tematike vlasnik elektroničke trgovine, ne posjeduje zalihe proizvoda vrlo često dolazi do nemogućnosti ispunjavanja kreirane narudžbe od strane posrednika. Kako posrednik naručuje proizvode tek nakon što krajnji kupac kreira narudžbu na njegovoj elektroničkoj trgovini, vrlo često dolazi od problema nedostatka proizvoda na zalihama krajnjeg proizvođača.

AliExpress, jedna od najpoznatijih platformi za kupnju dobara i usluga, svoj model trgovanja temelji na *Dropshipping* modelu.

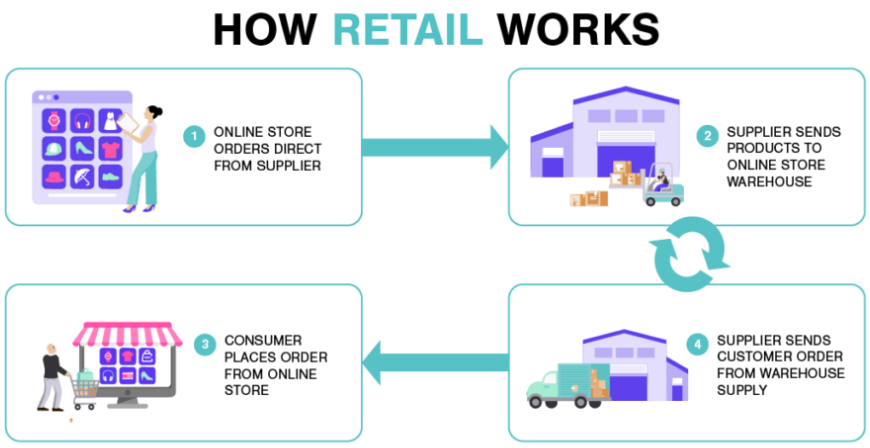
Slika 6. Dropshipping model poslovanja [15]

4.3.2.2. Wholesaling and Warehousing

Ferreira u svom članku *What is Dropshipping?* definira pojam *Wholesaling and Warehousing* kao način veleprodaje u kojem elektronička trgovina posjeduje fizičke lokacije za skladištenje proizvoda koje nudi u sklopu proizvodnog asortimana. [prema 19] Pravna osoba, kao vlasnik elektroničke trgovine, korištenjem ovog modela poslovanja osigurava zalihe proizvoda unaprijed, umjesto da ih naručuje prema potrebi, kao kod *Dropshipping* modela. Koncept *Wholesaling and Warehousing* modela temelji se na tome da vlasnik elektroničke trgovine, koji se bavi maloprodajom, naručuje proizvode od veleprodaje, primjerice proizvođača, te ih po primitku skladišti na vlastitim fizičkim lokacijama. Ferreira također navodi kako *Wholesaling and Warehousing* model zahtijeva vrlo visoka početna ulaganja koja su vezana uz upravljanje proizvodima i zalihama, upravljanje narudžbama te upravljanje samim skladišnim prostorom. Osnovni princip *Wholesaling and Warehousing* modela temelji se na ideji kupnje velike količine diferencirane robe koja se zatim prodaje po komadu uz određeni postotak marže[2]. *Wholesaling and Warehousing* model objedinjuje tri međusobno "sukobljene" strane – *proizvođač, posrednik* te *krajnji kupac* te omogućava njihovu sinergiju unutar lanca. *Wholesaling and Warehousing* model pruža jednostavnu skalabilnost tijekom ubrzanih promjena potražnje te nizak rizik, ali uz vrlo visoka početna ulaganja.

Ferreira navodi DollarDays kao jednog od najpoznatijih elektroničkih veletrgovaca koji u svom proizvodnom asortimanu ima preko 260 000 proizvoda. [19]

Alibaba, jedna od najpoznatijih platformi za kupnju dobara i usluga, svoj model trgovanja temelji na *Wholesaling and Warehousing* modelu.



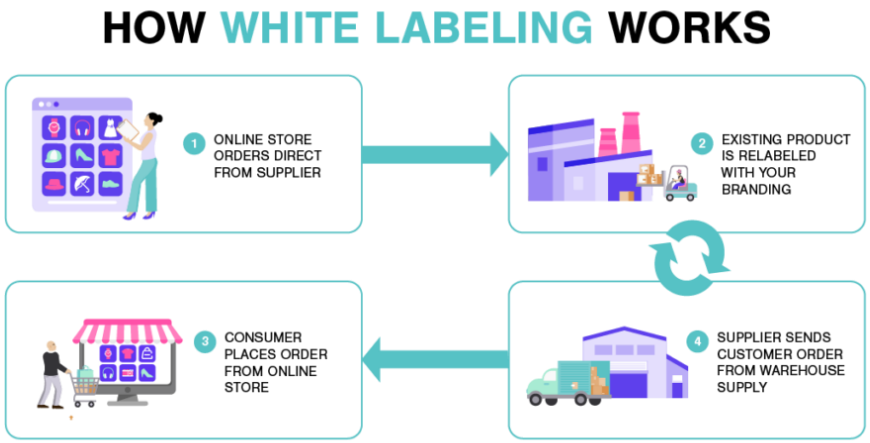
Slika 7. Wholesaling and Warehousing model poslovanja [15]

[2] Marža – razlika između nabavne i prodajne cijene robe koju ostvaruje trgovačko poduzeće koja se uobičajeno izražava u postotku. [prema 20]

4.3.2.3. White labeling

Ferreira u svom članku *What is Dropshipping?* definira pojam *White labeling* kao model elektroničke trgovine koji obuhvaća rebrendiranje proizvoda koji uslijed velike popularnosti iziskuju povećanje proizvodnje kako bi se zadovoljila potražnja na tržištu. [prema 19] Drugim riječima, proizvođač prodaje nebrendirani proizvod posredniku koji ga zatim prodaje pod krinkom vlastitog brenda. Posrednik na temelju istraživanja i analize tržišta utvrđuje proizvod(e) koji se trenutno uspješno prodaju te temeljem rezultata odabire proizvod(e) koji će prodavati u vlastitom pakiranju i pod vlastitim brendom. Drugim riječima, posrednik definira specifikacije proizvoda u vidu pakiranja te etikete, za neki od proizvoda postojećeg proizvodnog asortimana proizvođača, koje proizvođač zatim proizvodi u ime posrednika. *White labeling* model najčešće se koristi u kozmetičkoj industriji odakle je i stekao svoju popularnost. Primjerice, Alchemist lab jedan je najpoznatijih proizvođača koji primjenjuje *White labeling* model poslovanja i koji uspješno posluje s najvećim svjetskih kozmetičkim brendovima. U kontekstu elektroničke trgovine, svojevrsni *White labeling* model primjenjuje i WooCommerce dodatak koji omogućava aktivnosti elektroničke trgovine korištenjem WordPress platforme. Jedan od najvećih problema *White labeling* modela je potražnja. Naime, ukoliko potražnja nije na zadovoljavajućoj razini tada postoji rizik prekomjernog zadržavanja proizvoda na skladištu što u konačnici dovodi do zatvaranja poduzeća uslijed nedovoljne prodaje. Stoga, kako bi *White labeling* model bio uspješno primijenjen te polučio rezultate vrlo je važna detaljna analiza tržišta kako bi se utvrdili proizvodi za kojima vlada velika potražnja. *White labeling* model oslobađa proizvođača od marketinškog dijela prodaje, dok posredniku omogućava suočavanje s marketinškim dijelom prodaje te na taj način omogućava fokus na *core business*. [prema 21]

Vendasta, jedna od najpoznatijih platformi za kupnju dobara, svoj model trgovanja temelji na *White labeling* modelu, odnosno prodaji nebrendiranih proizvoda.



Slika 8. White labeling model poslovanja [15]

4.3.3. Modeli prema kriteriju prodajnog mjesta

Modeli elektroničke trgovine ove kategorije sadržavaju koncepte vezane uz upravljanje prodajom dobara i usluga, njihovom distribucijom te preuzimanjem od strane krajnjih korisnika. Osim toga, ovi modeli sadržavaju koncepte koji definiraju aktivnosti pojedinog sudionika te tako jasno diferenciraju i specificiraju odgovornosti tijekom procesa elektroničke trgovine. U modele prema kriteriju prodajnog mjesta ubrajamo *Bricks and Clicks* te *Pure Play*.
U nastavku teksta korišteni su engleski nazivi pojedinog modela iz razloga jer u hrvatskom govornom području ne postoje njihovi općeprihvaćeni i standardizirani ekvivalenti termini.

4.3.3.1. Bricks and Clicks

Kriss u svom članku *The Bricks and Clicks Business Model* definira pojam *Bricks and Clicks* kao način prodaje u kojem fizička osoba koristi fizičku te elektroničku trgovinu za promoviranje i distribuciju vlastitih dobara i usluga. [prema 22] *Bricks and Clicks* model uobičajeno obuhvaća poduzeća koja svoj poslovni model temelje na fizičkim trgovinama i lokacijama, pri čemu koriste elektroničku trgovinu kao sekundarni kanal distribucije dobara i usluga. *Bricks and Clicks* model na taj način objedinjuje tradicionalnu i elektroničku trgovinu, pružajući pritom mogućnost kombinacije različitih kanala prilikom obavljanja kupnje. Na taj način, *Bricks and Clicks* model omogućuje implementaciju i primjenu jedinstvene maloprodajne strategije kroz kombinaciju diferenciranih distribucijskih kanala koji su u međusobnoj sinergiji kroz koju ostvaruju svoju primarnu funkciju.

Walmart i Target najpoznatiji su primjeri poduzeća koji svoj poslovni model temelje na *Bricks and Clicks* modelu elektroničke trgovine.

4.3.3.2. Pure Play

Contreras u svom članku *Internet Transactions and Business models* definira pojam *Pure Play* kao način prodaje u kojem fizička osoba ne posjeduje fizičke lokacije na kojima prodaje dobra i usluga već distribuciju vlastitih dobara i usluga obavlja isključivo elektroničkim putem. [prema 23] *Pure Play* model koristi zanemariv broj poduzeća iako su istraživanja pokazala kako polučuje izvrsne rezultate. Poduzeća koja svoje poslovanje temelje na *Pure Play* modelu uobičajeno su fokusirana na specijalizirane tržišne grane. *Pure Play* model ima veliki broj prednosti u odnosu na tradicionalne trgovine. Kako se trgovina obavlja isključivo elektronskim putem, vlasnik poduzeća reducira troškove poslovanja glede troškova kupnje ili najma poslovnog prostora te infrastrukturnih i logističkih troškova.

PayPal najpoznatiji je primjer poduzeća koje svoj poslovni model temelji na *Pure Play* modelu elektroničke trgovine. Zanimljiva je činjenica kako je Amazon svoju popularnost stekao upravo na *Pure Play* modelu u okviru kojeg se bavio prodajom knjiga.

4.4. Prednosti i nedostatci elektroničke trgovine

Elektronička trgovina svoj eksponencijalni rast i veliku popularnost zahvaljuje brojnim prednostima u odnosu na klasične, tradicionalne načine trgovanja. [16] Pritom se pogodnosti, koje elektronička trgovina pruža nasuprot tradicionalnom načinu trgovanja, mogu podijeliti u dvije glavne kategorije: *prednosti za prodavače* te *prednosti za krajnje kupce (potrošače)*. Prednosti između spomenutih kategorija nalaze se u međusobnoj koherenciji te na taj način djeluju na zadovoljstvo obiju strana. U prethodnom poglavlju spomenuli smo razne načine na koje elektronička trgovina omogućava smanjenje troškova u odnosu na klasične, tradicionalne načine trgovanja. Smanjenjem troškova, prodavači su u mogućnosti smanjiti cijene dobara i usluga kako bi ostvarili stabilniji i održiviji položaj na suvremenom, dinamičnom tržištu te tako povećali potražnju. Prodavači na taj način ostvaruju konkurentsku prednost uz zadržavanje kvalitete dobara i usluga što pozitivno utječe na kupčevo zadovoljstvo kupljenim dobrima. Međutim, iako bogata prednostima, elektronička trgovina ima i svojih nedostataka koji se u današnje vrijeme pokušavaju riješiti upotrebom suvremenih tehnologija.

4.4.1. Prednosti elektroničke trgovine

Prednosti elektroničke trgovine međusobno se isprepliću između prodavača i potrošača te pružaju obostranu korist. Glavna prednost koju elektronička trgovina pruža prodavačima, u odnosu na tradicionalne načine trgovanja, svakako je smanjenje troškova vezanih uz fizičke lokacije te logističke usluge. U većini slučajeva, vlasnik elektroničke trgovine ne posjeduje fizičke lokacije na kojima skladišti proizvode već se proizvodi isporučuju izravno iz skladišta koje je u vlasništvu proizvođača pri čemu vlasnik elektroničke trgovine igra ulogu posrednika. Ružić u svojoj knjizi *E-marketing* navodi kako elektronička trgovina pruža temelj te osnovne koncepte koji prodavaču omogućuju širenje na nova tržišta. [prema 16] Elektronička trgovina je izrazito praktična te omogućava krajnjim kupcima trgovanje iz udobnosti vlastitog doma bez potrebe za odlaskom u fizičku trgovinu, pretraživanje polica te suočavanje s gužvom. Osim toga, krajnjim kupcima se na taj način omogućava vrlo jednostavna i brza usporedba dobara i usluga različitih prodavača putem njihovih elektroničkih trgovina.

U današnje vrijeme, elektroničke trgovine sve više pažnje posvećuju zadovoljavanju korisničkih potreba kroz interaktivnost i praćenje korisničkih interesa. U digitalnom okruženju, unutar kojeg djeluje i elektronička trgovina, trgovci dobivaju veću količinu povratnih informacija koje im omogućuju prilagođavanje elektroničke trgovine sukladnom korisničkim preferencijama čime se značajno utječe na povećanje korisničkog zadovoljstva. Povećanje korisničkog zadovoljstva dovodi do neizravnog marketinga koji posljedično dovodi do povećanja potražnje te reputacije elektroničke trgovine.

Ono što pozitivno utječe na korisničko iskustvo te zadovoljavanje korisničkih potreba svakako je i dostupnost usluge. Naime, elektroničke trgovine ''otvorene'' su 24 sata dnevno, 365 dana u godini čime je omogućena kupnja u bilo koje doba dana ili noći. Dok vlasnici spavaju ili odmaraju njihova elektronička trgovina brine o narudžbama krajnjih korisnika te ih na taj način oslobađa od obavljanja nepotrebnih poslova.

U prošlosti je postojao veliki problem vezan uz neinformiranost krajnjih korisnika o svojstvima i cijenama pojedine vrste proizvoda. Pojavom elektroničke trgovine krajnji korisnici postali su informiraniji no ikad prije. U svega par klikova, krajnji korisnici mogu saznati osnove informacije o pojedinom proizvodu te njegovim karakteristikama i cijenama. Upravo zbog globalne rasprostranjenosti i dostupnost Interneta loše vijest se jako brzo šire te uzrokuju pad potražnje kod prodavača koji su, iz bilo kojeg razloga, na lošem glasu. Glavna prednost koju elektronička trgovina pruža potrošačima, u odnosu na tradicionalne načine trgovanja, svakako je praktičnost te niže cijene proizvoda. Potrošačima je omogućena kupovina iz vlastitog doma uz niže cijene. Naime, smanjenjem troškova vezanih uz najam fizičkih lokacija za skladištenje i prodaju proizvoda, prodavačima se otvara dodatan prostor smanjenja cijene proizvoda kako bi privukli što veći broj potencijalnih kupaca te na taj način povećali potražnju.

Globalna dostupnost Interneta, a posljedično i elektroničke trgovine, dovela je do smanjenja troškova prodaje te povećanja potražnje. Ono što elektroničku trgovinu čini jedinstvenom je sposobnost skaliranja sukladno potrebama tržišta. Naime, povećanje broja kupaca ne zahtijeva povećanje broja zaposlenih. Drugim riječima, elektronička trgovina će na isti način rukovati desecima i tisućama korisnika. Osim toga, globalna dostupnost te sveprisutnost Interneta dovela je do pokretanja elektroničke trgovine u svega par klikova. Babić u svojoj knjizi *Dosezi elektroničke trgovine u Hrvatskoj i svijetu* navodi kako je, u današnje, suvremeno vrijeme, elektroničku trgovinu relativno lako pokrenuti uzimajući u obzir broj gotovih, tzv. *Plug and Play*, aplikacija koje podržavaju elektroničku trgovinu.

Elektronička trgovina, u odnosu na tradicionalne načine trgovanja, omogućuje smanjenje troškova obrade te izvršenja narudžbe. Smanjenje spomenutih troškova direktno je povezano sa sposobnošću elektroničke trgovine da se reaktivno prilagodi te skalira trenutnoj potražnji. Kroz automatizaciju procesa narudžbe, kao i svih popratnih aktivnosti, smanjuje se potreba za ljudskim resursima čime se izravno utječe na smanjenje troškova.

Važno je spomenuti kako digitalna transformacija te digitalne tehnologije nisu utjecale samo na djelovanje poduzeća i potrošača zasebno, već su utjecale i na njihovu međusobnu interakciju, prvenstveno putem društvenih mreža koje su poduzećima omogućile novi način promocije svojih proizvoda. Na taj se način poboljšava odnos između poduzeća i klijenta, a posebice ako se radi o kupnji proizvoda ili usluga putem interneta. [26]

4.4.2. Nedostatci elektroničke trgovine

Ružić u svojoj knjizi *E-marketing* navodi kako elektronička trgovina, iako bogata prednostima, ima i svojih nedostataka te ograničenja koja su ponajprije uzrokovana pogrešnim vođenjem posla od strane trgovaca te strogim zakonodavnim i regulatornim okvirima koji su postavljeni od strane vlasti pojedinih država. [prema 16] Uz to, postoji i određeni skup nedostataka s kojima se treba suočiti pošto su nedjeljiv pratitelj elektroničke trgovine.

S gledišta prodavača, najveći nedostatak s kojim se elektronička trgovina susreće vezan je uz brze promjene tehnologija koje uzrokuju proaktivnu primjenu istih uz vrlo visoka dodatna ulaganja. Kako krivulje učenja novih tehnologija, vezanih uz područje elektroničke trgovine, imaju eksponencijalni oblik, nedostatak tehnički obrazovanih stručnjaka javlja se kao problem na globalnoj razini. Kupnja iz udobnosti vlastitog doma nameće se kao glavna prednost elektroničke trgovine. Međutim, virtualna trgovina promatra se kao dvosjekli mač pošto je, u većini slučajeva, fizički kontakt s proizvodom glavni pokretač za njegovu kupovinu. Živković u svom stručnom radu *Prednosti i nedostatci elektronske trgovine* navodi kako je nedostatak fizičkog kontakta glavni nedostatak elektroničke trgovine. [prema 25] Naime, elektronička trgovina je u svojim početcima korištena kako bi prodavači prodavali robu niže kvalitete koju nisu uspjeli prodati u fizičkim prodavaonicama. Takav način prodaje uzrokovao je veliko nezadovoljstvo kod krajnjih kupaca nakon primitka narudžbe te negativno utjecao na korisničko iskustvo. Međutim, u današnje, suvremeno doba konkurentnost u području elektroničke trgovine je izrazito velika te se glavni naglasak stavlja na zadovoljavanje korisničkih potreba s ciljem zadržavanja kupaca. Velik broj elektroničkih trgovina omogućio je krajnjim kupcima da na jednostavan i brz način pronađu supstitut proizvoda s kojim nisu zadovoljni. U samim početcima elektroničke trgovine to nije bilo tako pošto su sami prodavači pružali veliki otpor reinženjeringu načina poslovanja, a samim time i ulasku na virtualno tržište.

Globalna rasprostranjenost Internet omogućila je međunarodnu elektroničku trgovinu koja ne poznaje zemljopisne granice, a koja je za sobom povukla razne poteškoće. Jedan od takvih problema vezan je uz oporezivanje te primjenu zakonskih regulativa. Naime, prodavač se može nalaziti u jednoj državi, isporučiti proizvod iz skladišta koje se nalazi u drugoj državi, kupcu koji se nalazi u trećoj državi. Ružić u svojoj knjizi *E-marketing* navodi kako poseban problem predstavljaju digitalni proizvodi za koje se ne može evidentirati prelazak carine, odnosno granice pojedine države.

Međutim, veliki rast popularnosti elektroničke trgovine ponukao je države članice različitih organizacija, a posebno organizacije OSCE (engl. *The Organization for Security and Co-operation in Europe*), na zajedničku suradnju te rješavanje problema s kojima se područje elektroničke trgovine susreće.

4.5. Digitalna transformacija trgovine

Spremić u svojoj knjizi *Digitalna transformacija poslovanja* definira pojam digitalne transformacije kao intenzivnu primjenu digitalne tehnologije i resursa kako bi se korišteni resursi pretvorili u nove prihode, poslovne modele te načine poslovanja. Digitalna transformacija nastupa onog trenutka kada poduzeće odluči, da u vrlo kratkom vremenskom razdoblju, želi promijeniti poslovne procese, strateški plan te strategije postizanja strateškog plana, hijerarhijsku i organizacijsku strukturu te poslovanje u globalu. [29] Navedeno se želi postići uz primjenu suvremenih digitalnih tehnologija kako bi se utjecalo na kohezivnost poslovnih procesa i strategije te time ostvarila veća konkurentnost na tržištu.

Globalna pandemija te globalizacija poslovanja utjecale su na porast značaja te utjecaja elektroničke trgovine, kako u Hrvatskoj tako i u svijetu. Prema podacima Hrvatske gospodarske komore, globalna pandemija utjecala je na značajan rast primjene elektroničke trgovine. Naime, rezultati istraživanja, koje je 2020. godine proveo *Eurostat*, ukazuju kako primjena elektroničke trgovine bilježi rast i to na razini od 10% u odnosu na godinu ranije. [27] Prema podacima *Državnog zavoda za statistiku*, Hrvatska je u istom periodu ostvarila rast primjene elektroničke trgovine na razini od 12,6% te time postala jedna od rijetkih trgovačkih struka koja bilježi rast od trenutka prisutnosti globalne pandemije. [28]

Digitalna transformacija nametnula se kao idealno rješenje za postizanje konkurentnosti u sektoru trgovine za vrijeme prisutnosti globalne pandemije. Hrvatska gospodarska komora pretpostavlja da će se pozitivne promjene u navikama kupaca te modelima prodaje zadržati i ubuduće uz pozitivan trend rasta u nadolazećem periodu. [30] Digitalna transformacija trgovačkog sektora provodi se strelovitim tempom, a sama elektronička trgovina postala je jedan od primarnih kanala prodaje. Uključivanje u digitalno, virtualno tržište više ne predstavlja mogućnost već potrebu. Poduzeća koja ne uspiju provesti digitalnu transformaciju poslovanja te istovremeno pratiti trgovinske trendove neće biti konkurentna te će teško opstati na tržištu koje je u današnje vrijeme vrlo dinamično i zahtjevno. Sukladno tome, digitalna transformacija poslovanja ne veže se samo uz velika poduzeća kao što je to bio slučaj ranije. Sve je veći broj malih, a posebice srednjih poduzeća, koja svoje poslovanje nastoje prilagoditi novom normalnom te tako kroz digitalnu transformaciju poslovanja postati konkurentniji te samim time opstati na tržištu.

Hrvatska gospodarska komora navodi kako su članovi vlasti pojedine države svjesni situacije koja je zatekla sektor trgovine te da će nastojati regulirati tržište te pomoći malim i srednjim poduzećima da se što bolje i brže prilagode novom normalnom. Pritom se i dalje nastoji fokusirati na potrošača te njegove potrebe. [30]

5. Internet i WWW (engl. *World Wide Web*)

Internet i World Wide Web dva su termina koja vrlo često susrećemo i koristimo u svakodnevnom životu. Pritom, većina ljudi smatra kako su Internet i World Wide Web ekvivalentni pojmovi te ih koristi u istom kontekstu. Međutim, radi se potpuno suprotnim pojmovima koji se vežu jedan uz drugog te djeluju u simbiozi, ali nikako ne definiraju istu stvar.

Prema definiciji koju iznosi *Leksikografski zavod Miroslav Krleža*, Internet (engl. *Inter* + *net[work]*) je svjetski sustav međusobno povezanih računalnih mreža. [31] Drugim riječima, Internet je mreža drugih mreža koja kreira metamodel komunikacije i razmjene podataka na globalnoj razini. Internet na taj način predstavlja fizičku mrežu koja povezuje milijune računala diljem svijeta koristeći pritom TCP/IP (engl. *Transmission Control Protocol/Internet Protocol*) protokol za dijeljenje i prijenos informacija. [prema 32] Internet pritom, koristeći koncept jedinstvene primjene TCP/IP protokola, omogućava uniformnu komunikaciju između različitih računala koja su rasprostranjena diljem svijeta. Popularnost Interneta prouzročena je ubrzanim razvojem informacijsko-komunikacijskih tehnologija koje su omogućile njegovu sveprisutnost te globalnu rasprostranjenost čime je postao osnova suvremene elektroničke komunikacije. Kada govorimo o samom terminu Internet, tada on ne obuhvaća samo mrežu na globalnoj razini već i sve druge mreže koje omogućuju isti koncept komunikacije i razmjene informacija i podataka. Sukladno tome, uobičajeno razlikujemo privatne, javne, akademske te vladine mreže koje su povezane uz pomoć elektronskih, optičkih ili bežičnih tehnologija umrežavanja.

Prema službenoj definiciji, koju iznosi CERN (engl*. European Council for Nuclear Research*), World Wide Web, ili kraće WWW, je inicijativa koja je pokrenuta s ciljem što bržeg, jednostavnijeg i univerzalnijeg pretraživanja i dohvaćanja hipertekstualnih[3] dokumenata. [prema 34] Neki autori definiraju pojam World Wide Web kao kolekciju međusobno povezanih multimedijskih dokumenata, pohranjenih na Internetu, na jednom ili više poslužitelja, kojima se pristupa korištenjem zajedničkog HTTP (engl. *Hypertext Transfer Protocol*) protokola. [32] Korištenje zajedničkog protokola razmjene podataka i informacija osigurava uniformnost primjene World Wide Web-a.

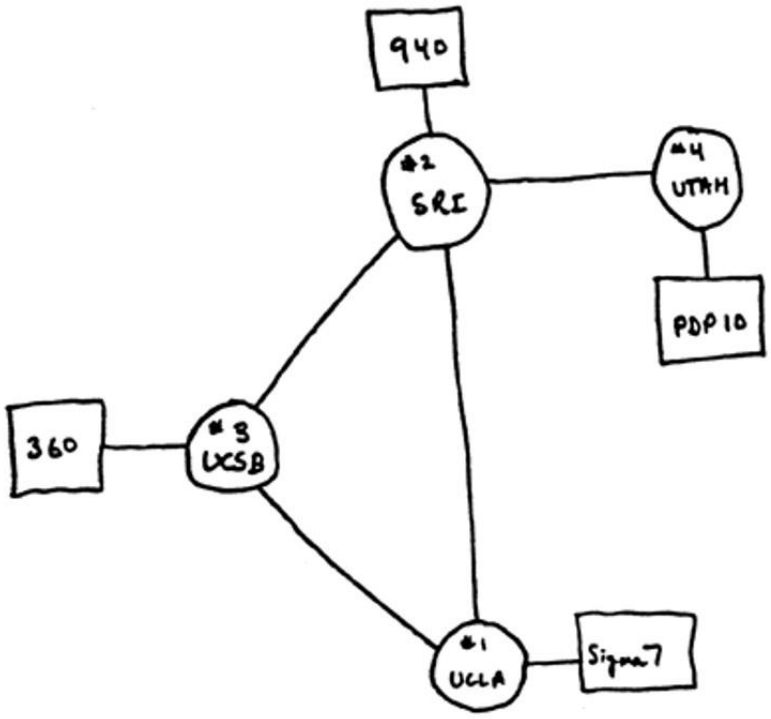
Iz svega navedenog jednostavno je zaključiti, a samim time potvrditi uvodnu hipotezu, kako Internet i World Wide Web nisu sinonimi te se ne smiju koristiti u istom kontekstu. Internet je mreža drugih mreža koja samim time predstavlja sklopovski dio cjelokupne infrastrukture, dok je World Wide Web usluga koja koristi mogućnosti koje Internet nudi te samim time predstavlja programski dio cjelokupne infrastrukture.

[3] Hipertekst – skup dokumenata u elektroničkom obliku, pretežito tekstovnog ili slikovnog sadržaja, koji su međusobno povezanih elektroničkim poveznicama (*hipervezama*, *linkovima*). [35]

5.1. Povijest Interneta i Web-a

Internet i World Wide Web imaju dugu, bogatu i zanimljivu povijest. Razvoj Interneta, kao globalne multimedijalne mreže, ovisio je ponajprije o razvoju različitih sredstava komuniciranja. [prema 37] Izumi te daljnji razvoj telegrafa, telefona, radija te računala bili su temelj pojave Interneta kakvog danas poznajemo. Internet je plod različitih istraživanja te kao takav nije nastao preko noći. Početkom 60.-ih godina prošlog stoljeća, Licklider, profesor na MIT-u (engl. *Massachusetts Institute of Technology*) bavio se istraživanjem i analizom ideje povezivanja računala na daljinu. U to vrijeme "digla se velika prašina" oko ideje povezivanja računala na daljinu, a poseban interes pokazao je DoD (engl. *Department of Defense*) koji je imao interes za razvoj decentraliziranih i distribuiranih mreža koje bi bile otporne na pogreške te omogućile dugotrajnu održivost te robusnost. [prema 38]

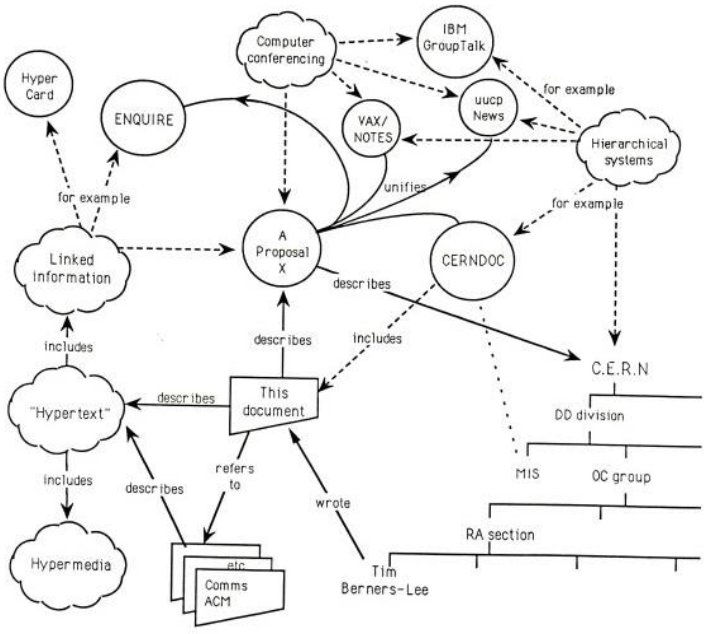
Upravo s ciljem osiguranja navedenih svojstava, 1969. godine izgrađen je ARPANET (engl. *Advanced Research Projects Agency Network*). Osnovni cilj ARPANET-a bilo je povezivanje četiriju sveučilišta i instituta – UCLA (engl. *University of California, Los Angeles*), SRI (engl. *Stanford Research Institute*), UCSB (engl. *University of California, Santa Barbara*) te UTAH kako bi istraživači s pojedinog sveučilišta i instituta uspješno razmjenjivali podatke i informacije sa svojim kolegama. Od povezivanja četiriju računala 1969. godine, Internet je dosegao milijun umreženih računala do 1992. godine te na taj način povezao veliki broj svjetskih sveučilišta. [36] ARPANET se zbog svojih karakteristika smatra pretečom Interneta.



Slika 9. Shema arhitekture ARPANET-a [39]

Internet u nadolazećim godinama nastavlja ubrzano napredovati te istovremeno stjecati popularnost na globalnoj razini. Korak koji je doveo do ubrzanog širenja Interneta u svakodnevni život ljudi svakako je uspostavljanje World Wide Web-a, kao jedne od najpoznatijih mrežnih usluga. [prema 36] Sam koncept koji promovira World Wide Web datira iz davnih 1940.-ih godina kada je Vannevar Bush u svom članku *As We May Think* iznio ideju hiperteksta kao jednog od načina međusobnog povezivanja dokumenata na računalu. Međutim, World Wide Web kakvoga danas poznajemo svoje korijene veže uz 1989. godinu kada je Tim Berners-Lee, tada istraživač na institutu CERN, objavio dokument pod nazivom *Information management: A proposal* u kojem je iznio osnovne koncepte vezane uz prijenos informacija putem Interneta korištenjem hiperteksta. [prema 36]

Razvoj World Wide Web-a uzrokovao je potrebu za definiranjem jezika za specifikaciju sadržaja dokumenata koji je kroz godine evolvirao u HTML (engl. *HyperText Markup Language*) kao i protokola za preuzimanje dokumenata i interpretiranje njihovog sadržaja koji je kroz godine evolvirao u HTTP. [prema 32] Godine 1990. razvijen je prvi Internetski preglednik pod nazivom WorldWideWeb, koji je kasnije preimenovan u Nexus kako bi se izbjegla konfuzija s terminom World Wide Web-a kao usluge. World Wide Web svoju komercijalnu dostupnost zahvaljuje 1991. godini kada postaje dostupan izvan CERN-a, odnosno široj globalnoj zajednici. Uspostava World Wide Web-a, kao globalno dostupnog servisa, dovela je do saznanja kako Internet ne mora biti korišten isključivo kao tehnička infrastruktura za razmjenu podataka i informacija o poslovanju, već se može iskoristiti kao medij u čijoj okolini će se obavljati cjelokupno poslovanje. [prema 36]



Slika 10. Shema arhitekture World Wide Web-a [40]

5.2. Prošlost, sadašnjost i budućnost Web-a

Evolucija World Wide Web-a, skraćeno Web-a, popraćena je kroz njegove generacije koje su evoluirale sukladno rapidnim napredcima tehnologija kojima smo svjedoci i danas. U svojim početcima, Web je korišten za prikaz statičnih sadržaja te kao takav nije omogućio interaktivnost s krajnjim korisnicima. Razvoj tehnologija uzrokovao je novi skup potreba te problema čija su rješenja pronađeno upravo u mogućnostima koje pruža sam Web. Dinamičnost je omogućila nove poglede na Web, a posebno u kontekstu interaktivnosti s krajnjim korisnicima. Interaktivnost Web-a smatra se prekretnicom u njegovu korištenju te predstavlja glavni razlog njegove globalne rasprostranjenosti te neprekidnog rasta. Različiti autori generaliziraju Web na različite načine. Ustaljena kategorizacija Web-a, koju spominje svaki od autora, definira tri osnove generacije Web-a – *Web 1.0*, *Web 2.0* te *Web 3.0*.

5.2.1. Generacija Web 1.0

Generacija Web 1.0 veže se uz same početke primjene Web-a te uključuje korištenje hiperteksta kao načina međusobnog povezivanja dokumenata na računalu. Web 1.0 omogućio je prikaz, odnosno pregledavanje statičnog sadržaja. Upravo zbog statičnosti sadržaja, *Read-only Web* jedan je od učestalih i široko prihvaćenih termina koji se veže uz pojam Web 1.0. Sharma u svom članku *Web 1.0, Web 2.0 and Web 3.0 with their difference* navodi kako je u početcima bilo svega nekoliko korisnika u svijetu koji su se bavili uređivanjem sadržaja Web stranica, tzv. *Webmasteri*, dok je istovremeno, broj konzumenata sadržaja rastao enormnom brzinom. [prema 41]

Portfolio, odnosno osobna Web stranica, je prva asocijacija na koju pomisli informatički pismena osoba prilikom spomena statičnog sadržaja. Generacija Web 1.0 opisuje koncepte koji definiraju statični sadržaj Web stranica koji krajnjim korisnicima omogućava isključivo pregled sadržaja Web stranice bez sposobnosti interakcije. Pojedini dokumenti smješteni su unutar direktorija te na taj način vrlo lako dostupni krajnjim korisnicima. Prema Sharmi, tri osnovna koncepta koja uobličuje Web 1.0 su *statičan sadržaj*, *posluživanje sadržaja koji se nalazi unutar datotečnog sustava poslužitelja* te *izrada Web stranica korištenjem SSI* (engl. *Server Side Includes*) ili *CGI* (engl. *Common Gateway Interface*). [prema 41]

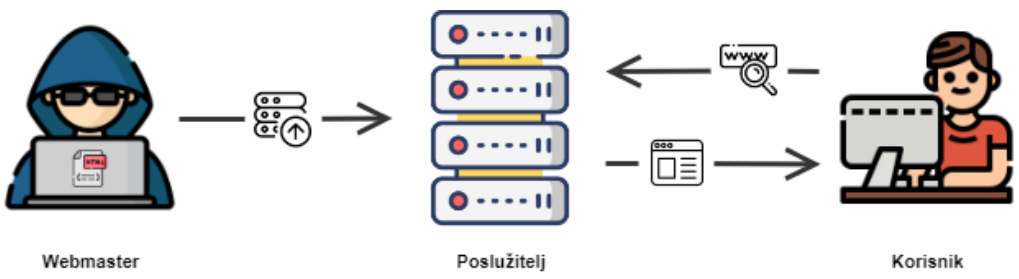
Web 1.0 imao je svoju primjenu dugi niz godina. Naime, razdoblje ''vladavine'' i nadmoći generacije Web 1.0 smatra se od 1991. pa sve do 2004. godine. Web stranice generacije Web 1.0 imale su isključivo statičan sadržaj pri čemu nije postojala nikakva komunikacija između aplikacijskog servera te poslužitelja baze podataka ili nekog drugog izvora te spremišta podataka. Na taj su način krajnji korisnici konzumirali sadržaj koji je bio aktualan u datom

trenutku vremena, a svaka nova promjena, kao što je unos nove, aktualne vijesti, zahtijevala je ažuriranje sadržaja datoteke na poslužitelju od strane *Webmaster-a*.

Hiperveze su bile jednostavne te najčešće vezane uz pojedine riječi ili kraće fraze. [42] Današnja, Web 2.0 generacija o kojoj ćemo detaljnije u poglavlju *5.1.1.2. Web 2.0*, obiluje raznolikošću dizajna. Međutim, Web 1.0 je u svojim početcima sadržavao datoteke koje nisu bili stilizirane te su sadržavale samo konkretan sadržaj. Međutim, pojavom CSS-a (engl. *Cascading Style Sheets*) 1994. godine, sadržaj dobiva na dodatnom značaju te se korisnicima omogućava njegovo jednostavnije razmatranje te razumijevanje. Primjerice, naglašavanjem naslova, odvajanjem pojedinih paragrafa, pozicioniranjem elemenata i slično.

Glavne domene te područja interesa u tom periodu bili su protokoli poput HTTP-a te FTP-a (engl. *File Transfer Protocol*) te jezici za prezentaciju i uređivanje sadržaja poput HTML-a te XML-a (engl. *Extensible Markup Language*). [prema 42] Osnovni cilj generacije Web 1.0 bio je prezentacija Web-a kao globalno dostupne usluge uz pomoć različitih sadržaja koje će konzumirati krajnji korisnici. Naravno, u današnje, moderno vrijeme i sam spomen generacije Web 1.0 se smatra zastarjelim. Međutim, u periodu od 1991. pa do 2004. Web 1.0 predstavio je novi način prezentacije informacija koji je izazvao je veliki *hype* na globalnoj razini, a posebno u istraživačkim društvenim zajednicama.

Arhitektura generacije Web 1.0 sastoji od centralnog poslužitelja na kojem su pohranjene datoteke sa statičnim sadržajem kojima krajnji korisnici pristupaju putem hiperveza. U osnovni, koncept rada koji promovira generacija Web 1.0 je vrlo jednostavan. *Webmaster* kreira datoteku sa sadržajem te ju pohranjuje u odgovarajući direktorij na centralnom poslužitelju. Krajnji korisnik putem pripremljene hiperveze pristupa datoteci, preuzima je na vlastito računalo te zatim konzumira njen sadržaj. Kako su datoteke sadržavale statičan sadržaj, svaka promjena sadržaja zahtijevala je ažuriranje sadržaja korespondentne datoteke na centralnom poslužitelju. Primjerice, ukoliko je datoteka sadržavala "najnovije" vijesti, tada bi dolaskom nove vijesti *Webmaster* morao ažurirati sadržaj datoteke te ponovno postaviti datoteku u odgovarajući direktorij na centralnom poslužitelju. Primjer Web stranice iz razdoblja generacije Web 1.0 možete pronaći na https://worldwideweb.cern.ch.



Slika 11. Pohranjivanje i dohvaćanje statične Web stranice

5.2.2. Generacija Web 2.0

Krah *dot-com*[4] balona, koji se dogodio u jesen 2001. godine, označio je prekretnicu za World Wide Web te donio nove poglede na razmjenu podataka putem Interneta. [prema 43] Sam termin Web 2.0 etablirao je Tim O'Reilly 2004. godine nakon što je na jednoj od konferencija izrekao rečenicu koja je promijenila cjelokupni koncept Web-a.

„Web 2.0 is the business revolution in the computer industry caused by the move to the Internet as a platform, and attempt to understand the rules for success on that new platform."

Dakle, razdoblje vladavine generacije Web 2.0 započinje 2004. godine, a upravo zbog svojih karakteristika traje još i danas. U periodu prelaska s generacije Web 1.0 na generaciju Web 2.0, došlo je do velike evolucije Web-a, kako u pogledu njegovog korištenja od strane krajnjih korisnika, tako i u načinu prezentacije te uređivanja sadržaja od strane *Webmaster-a*. Interaktivnost je karakteristika koja se smatra najvrjednijom posljedicom prelaska na novu generaciju Web-a. Za razliku od njezinog prethodnika, generacija Web 2.0, umjesto slijepog vraćanja odgovora u obliku informacija sadržanih u HTML datoteci, uvodi koncepte koji omogućuju pohranu korisničkih informacija koje se kasnije koriste prilikom generiranja sadržaja pojedine Web stranice i/ili Web aplikacije.

Web 2.0 je termin koji definira evolucijski napredak i svojevrsnu nadogradnju Web 1.0 generacije. Generacija Web 2.0 donijela je velike promjene u globalni, virtualni svijet, a posebno u kontekstu interaktivnosti s krajnjim korisnicima što je dovelo do poslovne revolucije u računalnoj industriji. Statičnost Web stranica, kao glavna odlika Web 1.0 generacije, zamijenjena je interaktivnim sadržajem kojeg mogu stvarati i sami krajnji korisnici. Sharma u svom članku *Web 1.0, Web 2.0 and Web 3.0 with their difference* navodi kako se, zbog svoje interaktivnosti, Web 2.0 često naziva i participativnim te društvenim Web-om. [prema 41]

Interaktivnost, kao glavna odlika Web 2.0 generacije, omogućila je izradu dinamičkih Web stranica koje omogućuju "komunikaciju" između krajnjeg korisnika te Web aplikacije. Upravo zbog dinamičnosti sadržaja, *Read-write Web* jedan je od učestalih i široko prihvaćenih termina koji se veže uz pojam Web 2.0. Nova generacija Web-a unijela je kako pozitivne, tako i negativne aspekte u cijeli koncept razmjene informacije i podataka putem Interneta. Prije svega, Web 2.0 omogućio je povećanje upotrebljivosti i interoperabilnosti za krajnje korisnike. Uloga krajnjih korisnika, prilikom "komunikacije" s Web stranicom, postaje važnija no ikad. Pojavom nove generacije Web-a, koja omogućava interaktivnost s krajnjim korisnicima, dolazi i do pojave termina Web aplikacija.

[4] Dot-com – razdoblje između 1997. te 2000. godine tijekom kojeg je zabilježen snažan rast ekonomskih vrijednosti tvrtki koje su bile povezane Internetom. [prema 44]

U poglavlju *5.1.1.1. Web 1.0* navodi se samo pojam Web stranica, dok se u poglavlju *5.1.1.2. Web 2.0* isprepliću pojmovi Web stranica te Web aplikacija. Ključne razlike između tih dvaju pojmova navedene su u poglavlju *5.3 Razlike između Web stranice i Web aplikacije*.

Centraliziranost je *keyword*[5] koji se često spominje u kontekstu generacije Web 2.0. Centraliziranost je dovela do pohranjivanja podataka na jedinstvenom poslužitelju čime je omogućeno poboljšanje korisničkog iskustva primjenom metode strojnog učenja kao grane umjetne inteligencije. Naime, strojno učenje omogućilo je dobivanje informacija o navikama te interesima samih korisnika što je omogućilo nove načine promoviranja. Tehnološke tvrtke bavile su se prodajom informacija o korisnicima te tako ostvarile veliku zaradu. [prema 42]

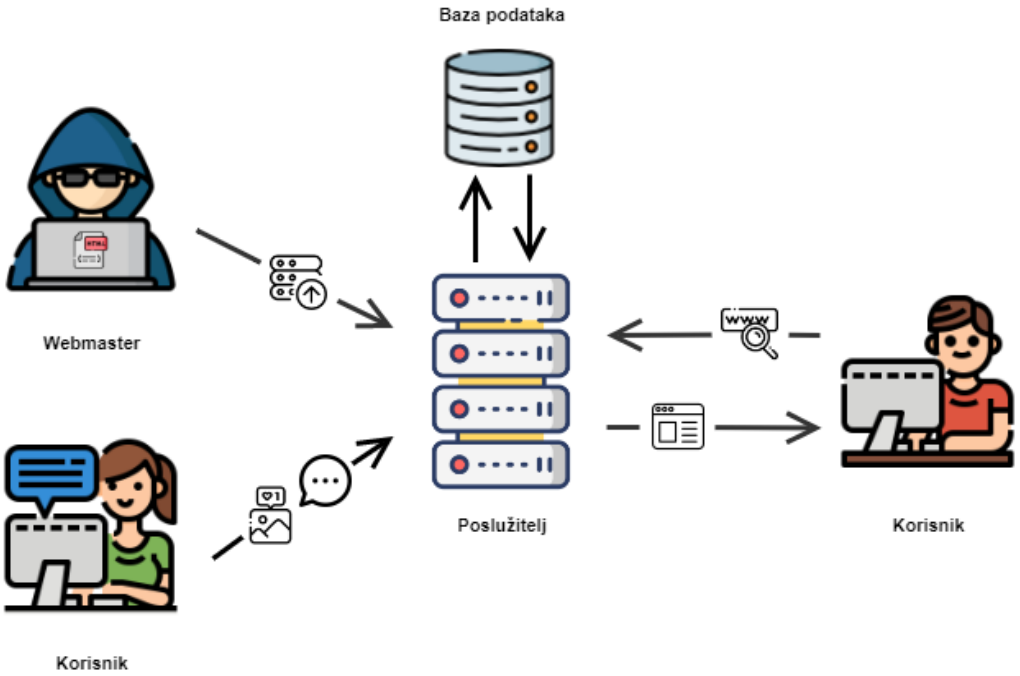
Generacija Web 2.0 omogućila je centralizirano prikupljanje korisničkih preferencija, informacija te intencija prilikom korištenja i pretraživanja Web aplikacija kao što su Facebook i YouTube. Analiza dobivenih korisničkih informacija omogućila je kreatorima Web aplikacija osiguravanje boljeg korisničkog iskustva s ciljem zadržavanja postojećih, te posljedično pridobivanja novih krajnjih korisnika. Dakle, u svojim početcima, usluge generacije Web 2.0 temeljile su se na povećanju virtualne zajednice s ciljem ostvarivanja što je moguće većeg profita. Velike korporacije uvidjele su potencijal koji donosi centraliziranost podataka o krajnjim korisnicima te su počele pakirati i prodavati informacije oglašivačkim poduzećima. Pritom je važno spomenuti kako većina krajnjih korisnika ne brine o zaštiti osobnih podataka na Web-u što je danas posebno vidljivo na društvenim mrežama. Krajnji korisnici, bez prisile, javno objavljuju vlastite korisničke podatke što velike korporacije masovno iskorištavaju. Upravo zbog spomenutih karakteristika, u literaturi se često spominje kako se razdoblje generacija Web 2.0 smatra dobom masovnog oglašavanja te nedostatka privatnosti osobnih podataka.

Kod generacije Web 2.0, dva krajnja korisnika mogu pristupiti istoj domeni, odnosno istom URL-u (engl, *Uniform Resource Locator*), primjerice facebook.com, te pritom vidjeti potpuno drugačije podatke i informacije na naslovnici s najnovijih objavama. Takav način prezentacije informacija omogućen je primjenom koncepta dinamičkog prikazivanja sadržaja koji se dohvaća na temelju korisničkih podataka trenutno prijavljenog korisnika. Drugim riječima, Web aplikacija sortira podatke temeljem podataka o trenutno prijavljenom korisniku te na temelju informacija koje je trenutno prijavljeni korisnik implicitno pohranio u centraliziranu bazu podataka prilikom svojih prijašnjih korištenja Web aplikacije. Primjerice, trenutno prijavljeni korisnik prilikom pregledavanja video materijala ili komentiranja objava na Facebook-u implicitno i nesvjesno pruža informacije koje se kasnije koriste prilikom prikazivanja oglasa te drugog (ne)edukativnog sadržaja na spomenutoj platformi.

[5] Keyword – ključna riječ, izraz ili fraza koja se često koristi u određenom kontekstu te na taj način predstavlja njegovu nadopunu ili pak nadopunu osnovnog termina.

Interaktivnost je dovela do preopterećenosti sadržajem koji se nudi krajnjim korisnicima. Svakog dana kreira se 2.5 kvintilijuna novih bajtova informacija čime sadržaj gubi na pouzdanosti. Prema Sharmi, tri osnovna koncepta koja uobličuje Web 2.0 su *dinamičan sadržaj*, *sortiranje podataka koje omogućuje korisničku klasifikaciju primljenih informacija*, *primjena Web servisa, API-a* (engl. *Application programming interface*) te *HTTP-a kao osnove za komunikaciju između klijentske i serverske strane aplikacije*. [prema 41] Kako bi se zadovoljile sve navedene karakteristike generacije Web 2.0, razvijene su nove tehnologije koje se koriste prilikom razvoja Web aplikacija – AJAX (engl. *Asynchronous JavaScript and XML*) te cijeli niz JavaScript programskih okvira, među kojima je najpoznatiji jQuery. Primjenom spomenutih mehanizama, omogućuje se generiranje sadržaja, odnosno HTML datoteke, na strani poslužitelja koja će se poslati kao odgovor krajnjem korisniku. Krajnji korisnik, nakon obavljene akcije, kreira HTTP zahtjev prema poslužitelju koji obrađuje primljeni zahtjev te generira HTML dokument koju zatim šalje kao HTTP odgovor na primljeni zahtjev.

Generacija Web 2.0 ne predstavlja tehničku nadogradnju generacije Web 1.0 već sadrži elemente koji definiraju svojevrsne kumulativne promjene koje su uzrokovale modifikacije u razvoju te načinu interakcije s krajnjim korisnicima. Drugim riječima, Web 2.0 ne mijenja u potpunosti arhitekturu World Wide Web-a te njegovih *core* koncepata već pruža nove načine komunikacije te razmjene podataka i informacija. Prema Sharmi, Web 2.0 karakterizira primjena tehnologija kao što su blogovi, društveno umrežavanje, P2P razmjena podataka te povezivanje podataka iz velikog broja različitih izvora. [prema 41]



Slika 12. Pohranjivanje i dohvaćanje dinamičke Web stranice

5.2.3. Generacija Web 3.0

U poglavlju *5.1.1.2. Web 2.0* bavili smo se osnovnim konceptima generacije Web 2.0 kao i njezinim prednostima te nedostatcima. Generacija Web 2.0 je zbog svojih karakteristika aktualna i danas te se još uvijek široko koristi. Unazad nekoliko godina pojavio se novi termin, generacija Web 3.0, koji opisuje Web nove generacije koji bi u narednim godinama trebao unaprijediti koncept Web 2.0 te riješiti njegove nedostatke ponajprije primjenom koncepta decentralizacije te sveprisutne *Blockchain* tehnologije. U suštini, *Blockchain* je tehnologija koja se temelji na distribuiranoj bazi podataka koja se javlja u obliku digitalne javne knjige (engl. *ledger*), a koja pritom koristi kriptografiju za osiguravanje povjerljivosti, autentičnosti te integriteta podataka i informacija. [prema 45] *Blockchain* je danas dominanta tehnologija koja se koristi u kriptovalutnim sustavima, kao što je Bitcoin, kako bi se osigurala sigurnost te decentralizacija transakcija. Upravo zbog svojih karakteristika, *Blockchain* je pronašao svoju primjenu i u sklopu generacije Web 3.0.

Ne postoji službena definicija pojma generacije Web 3.0, međutim, radi se o terminu koji je trenutno u velikom *hype-u* te je najpopularnija riječ mnogih IT stručnjaka, kripto entuzijasta te investitora. [46] U širem kontekstu, izraz Web 3.0 obuhvaća sve inovacije i suvremene koncepte koji će u skorije vrijeme značajno unaprijediti Web koji trenutno koristimo kao i način njegova korištenja, posebno sa stajališta sigurnosti i pouzdanosti. Web 3.0 generacija kroz svoj evolucijski rast naglasak stavlja na primjenu koncepata kao što su decentraliziranost, umjetna inteligencija te sustavi kriptovaluta. U svojoj srži, Web 3.0 nastoji riješiti probleme s kojima se susreću korisnici generacije Web 2.0. Prije svega, Web 3.0 kroz primjenu koncepta decentralizacije rješava problem kontrole nad vlastitim podacima te ovisnosti o pravilima koja nameću treće strane. [prema 46]

Web 3.0 temelji se na primjeni P2P tehnologija, kao što su *Blockchain* te Internet stvari kako bi osigurao željene karakteristike te na taj način zadovoljio korisničke potrebe. Generacija Web 3.0 svojim korisnicima nastoji prezentirati najbolje što Internet, kao globalno rasprostranjena i široko dostupna mreža, nudi. Web 3.0 se pritom oslanja na prethodne generacije, Web 1.0 te Web 2.0, kako bi, kombinirajući njihove najbolje karakteristike, kreirao Web koji će biti jedinstven te tako zadovoljio korisničke potrebe. Koncepti koje Web 3.0 promovira poznati su u engleskom govornom području pod nazivima *Permissionless* te *Trustless*. Za razliku od svojih prethodnika, koji prilikom interakcije s Web stranicama i/ili Web aplikacijama zahtijevaju autentifikaciju[6] te autorizaciju[7] putem centralnog poslužitelja, što u

[6] Autentifikacija – sigurnosni mehanizam koji omogućava jedinstvenu provjeru identiteta korisnika primjenom odgovarajućeg autentifikacijskog mehanizma (korisničko ime i lozinka, otisak prsta,…).

[7] Autorizacija – sigurnosni mehanizam koji omogućava određivanje prava pristupa te uređivačkih ovlasti pojedinog korisnika temeljem provedene autentifikacije ((ne)registrirani korisnik, administrator).

prijevodu znači zasebnu registraciju na svakoj pojedinoj stranici, generacija Web 3.0 nastoji nametnuti registraciju na Web stranice i/ili Web aplikacije bez dodatne autorizacije. *Bitcoin store* u svome članku *Što je Web 3.0* navodi kako termini *Permissionless* i *Trustless* definiraju koncepte koji omogućavaju dijeljeno pristupanje pojedinim Web stranicama i/ili Web aplikacijama koristeći pritom jedinstveni račun. [prema 46] Dobar primjer primjene spomenutih praksi svakako su digitalni novčanici koji su detaljnije definirani i opisani u poglavlju *4.2.3. Elektronički novčanik*. Nakon spajanja na elektronički novčanik omogućeno je jednostavno konzumiranje različitih sadržaja na različitim Web stranicama i/ili Web aplikacijama. Upravo zbog svojih karakteristika, *Read-write-execute Web* jedan je od učestalih i široko prihvaćenih termina koji se veže uz pojam Web 3.0

Ono što generaciju Web 3.0 čini posebnom svakako je primjena raznih tehnika umjetne inteligencije te metoda i tehnika strojnog učenja. Ovu značajku moguće je primijetiti prilikom korištenja mail ili chat servisa u obliku unaprijed pripremljenih riječi i/ili rečenica na temelju preostalog teksta koji je definiran unutar same poruke. Web 3.0 će uz očekivani rast tehnologije, a samim time i računalne snage, omogućiti razumijevanje ljudskih potreba. [46]

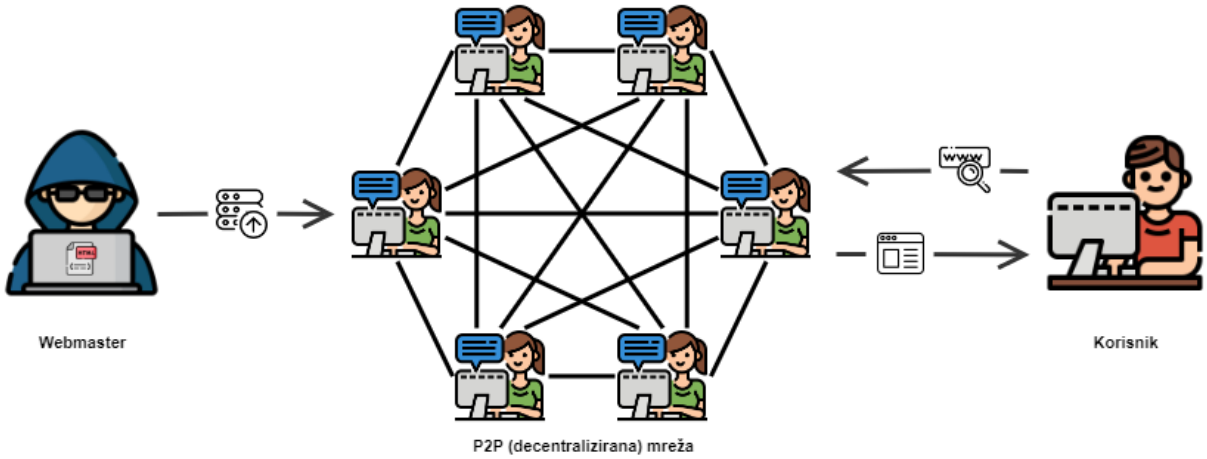
Povezanost sa svim uređajima omogućit će jednostavnu dostupnost sadržaja na svim uređajima i aplikacijama koje su povezane Internetom. [46] Napredak tehnologije uzrokovao je povećanje broja različitih vrsta izvora i uređaja koji se mogu povezati putem Interneta. Znanstvenici koji se bave istraživanjem razvoja tehnologija predviđaju kako će generacija Web 3.0 posebno pospješiti primjenu IoT-a[8] (engl. *Internet of Things*), odnosno Interneta stvari.

Slika 13. Prijelaz iz centraliziranih aplikacija na decentralizirane ekvivalente [46]

[8] Internet of Things (IoT) – kocept koji podrazumijeva spajanje bilo kojeg uređaja na Internet i/ili s drugim uređajima. [47]

Nepostojanje centralnog tijela koje nadzire i kontrolira cjelokupni mrežni sustav, *bolji protok informacija i podataka*, *učinkovitije pretraživanje i filtriranje sadržaja koji je dostupan na Web-u* te *poboljšano oglašavanje* značajke su koje različiti izvori i autori ističu kao najveće prednosti primjene Web-a 3.0, kao nove i unaprijeđene generacije Web-a. [prema 46] Osim toga, smatra se kako će generacija Web 3.0 značajno pridonijeti razvoju sigurnosti na Web-u što će značajno utjecati na poboljšanje korisničkog iskustva prilikom korištenja Web-a. U tom kontekstu, očekivano je kako će se većina globalnog poslovanja preseliti u virtualno okruženje te svoj tržišni rast temeljiti na primjeni različitih koncepata te modela elektroničke trgovine.

Nepostojanje centralnog tijela koje nadzire i kontrolira cjelokupni mrežni sustav omogućeno je uklanjanjem trećih, posredničkih strana koje su imale ulogu kontrole i pohrane korisničkih podataka. [46] Primjenom spomenutog koncepta omogućena je pohrana korisničkih podataka na više lokacija, odnosno decentraliziranost samih korisničkih podataka, umjesto da su svi podaci pohranjeni na nekom jedinstvenom, centralnom poslužitelju. Samim time, krajnji korisnici imaju slobodu glede odlučivanja o upotrebi njihovih osobnih podataka. Povezanost sa svim uređajima omogućit će širu primjenu IoT-a te bolji protok informacija i podataka. Veći broj uređaja, koji su na globalnoj razini povezani Internetom, omogućit će analizu većih, te samim time reprezentativnijih, skupova podataka što u konačnici pridonosi pružanju informacija koje su prikladnije specifičnim potrebama krajnjih korisnika. [46] Učinkovitije pretraživanje i filtriranje, odnosno kvalitetniji SEO (engl. *Search Engine Optimization*) sadržaja koji je dostupan na Web-u omogućeno je zahvaljujući metapodacima kao i primjenom umjetne inteligencije i koncepata strojnog učenja koji će omogućiti razumijevanje konteksta ključnih riječi prilikom pretraživanja i filtriranja podataka. [prema 46] Umjetna inteligencija te koncepti strojnog učenja omogućit će poboljšano oglašavanje. Generacija Web 3.0 promovira oglašavanje sukladno potrebama korisnika umjesto bombardiranja oglasima na svakom koraku i u svakom kutku Web-a. [prema 46]



Slika 14. Pohranjivanje i dohvaćanje podataka u/iz P2P mreže

5.3. Razlike između Web stranice i Web aplikacije

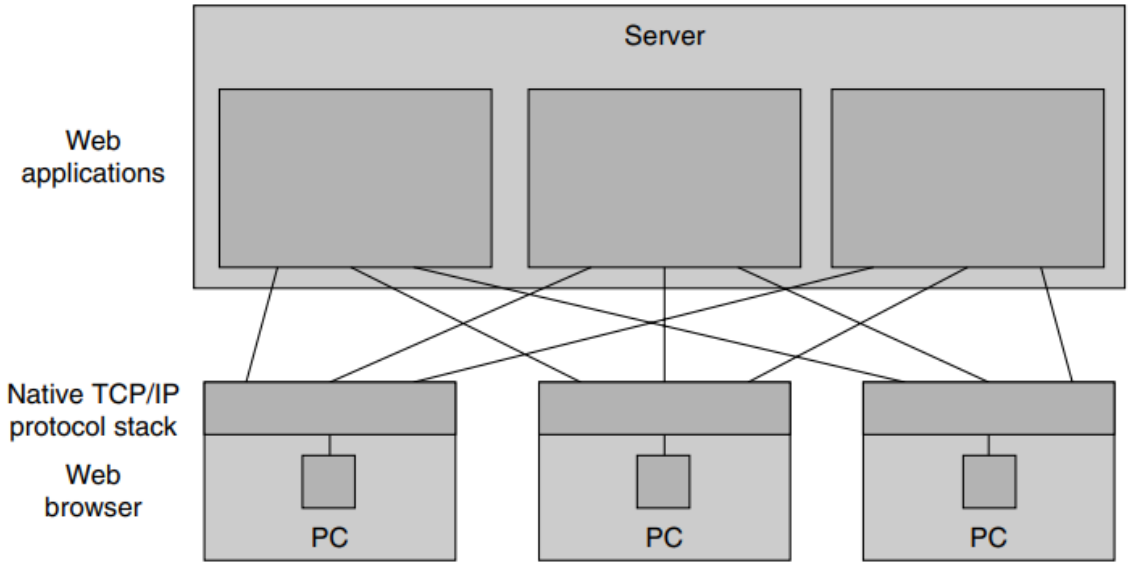
U današnjem suvremenom, modernom svijetu, koji je pod velikim utjecajem Web-a, njegovih tehnologija, usluga te servisa, vrlo često se spominju pojmovi Web stranice te Web aplikacije. U svakodnevnom govoru, u većini slučajeva, pojmovi Web aplikacija i Web stranica koriste se kao sinonimi iako se radi o terminima s dijametralno suprotnim karakteristikama. U suštini, radi se o "sukobu" dviju generacija Web-a. Pojam Web stranice vežemo uz generaciju Web 1.0 te njezinu statičnost sadržaja, dok pojam Web aplikacije vežemo uz generaciju Web 2.0 koja pruža interaktivnost s krajnjim korisnicima te dinamičnost sadržaja. Mnoge Web platforme te Web mjesta sadrže kombinaciju statičnog i dinamičnog sadržaja zbog čega linija između termina Web stranica i Web aplikacija može postati nejasna, a posebice u okviru karakteristika modernog Web-a. Uspostavljanjem novih izraza koji se koriste u kombinaciji s Internetom i povezanih tehnologijama došlo se do pitanja:

Koje su razlike između navedenih termina te u kojem kontekstu koristiti pojedini od njih?

Ključna razlika između dvaju sukobljenih termina je u statičnosti, odnosno dinamičnosti posjećenog Web mjesta. Web stranice pružaju samo statične informacije te se samim time koriste za prikaz sadržaja informativnog karaktera. S druge pak strane, Web aplikacije interaktivne su prirode te obiluju dinamičnošću sadržaja što omogućava generiranje sadržaja na zahtjev u ovisnosti o parametrima te karakteristika autentifikacije te autorizacije. [prema 48] Kao posljedica spomenute karakteristike, Web stranice ne sadrže elemente koji se mijenjaju temeljem korisničkih akcija, dok Web aplikacije svoj rad temelje upravo na interaktivnosti s krajnjim korisnicima te promjeni karakteristika gradivnih elemenata Web mjesta u ovisnosti o korisničkim akcijama. Kroz analizu domene Web mjesta restorana biti će prikazana razlika između Web stranice te Web aplikacije.

Web stranicu možemo definirati kao hipermedijski sadržaj, sastavljen od jednostavne arhitekture zasnovane na HTML-u, XHTML-u (engl. *Extensible HyperText Markup Language*) ili XML-u, spremljen kao datoteka na Web poslužitelju ili se vraća kao rezultat kreiranog zahtjeva unutar Web aplikacije [prema 32] U tom kontekstu, Web stranice se koriste za prikaz statičnih podataka informativnog karaktera koji se vrlo rijetko mijenjaju. Obzirom na svoje karakteristike, Web stranica ne omogućava interaktivnost s krajnjim korisnicima te provođenje transakcija. Veliki broj Web mjesta pojedinog restorana ubraja se u kategoriju Web stranica. Razlog tome je što većina njih sadržava informacije informativnog karaktera koje su usko povezane s djelatnošću kojom se restoran bavi. Web mjesta restorana uobičajeno sadržavaju informacije kao što su naziv restorana, adresa restorana, aktualni jelovnik, pogodnosti, broj telefona i/ili e-mail adresa i slično. Sve navedene informacije informativnog su karaktera te kao takve ne omogućavaju nikakvu korisničku interakciju.

S druge pak strane, Web aplikaciju možemo definirati kao programski sustav koji generira Web stranice i dokumente, a napisan je u nekom od programskih jezika koji se izvršavaju na strani poslužitelja. Ujedno, Web aplikacija služi za preuzimanje podataka od korisnika i njihovo permanentno spremanje u perzistentno spremište za pohranu podataka kao što je datoteka ili baza podataka. [prema 32]. Uobičajeno, Web aplikacija koristi autentifikaciju i autorizaciju putem korisničkog imena i lozinke kako bi omogućila kreiranje veze između identiteta krajnjeg korisnika te provedenih akcija (narudžba, rezervacija i slično). Web aplikacija je termin koji se koristi za aplikaciju koja se poziva iz Web preglednika te koristi Internet za komunikaciju s drugim resursima i servisima. [49] Pojednostavljeno, Web aplikacija je vrsta aplikacije koja se temelji na klijent-server arhitekturi pri čemu se Web preglednik koristi kao klijentska strana aplikacije. Web aplikacije se pojavljuju u obliku formi za unos podataka na temelju kojih se generira određeni rezultat ili pohranjuju podaci u bazu podataka, pa sve do Web aplikacija, kao što je Microsoft 365, koje nude Cloud funkcionalnosti Microsoftovih bazičnih proizvoda. Web aplikacija je pokrenuta na serveru koji upravlja zahtjevima i odgovorima koje dobiva iz samog koda aplikacije. U trenutku kada klijent pošalje zahtjev za određenim resursom, aplikacija taj zahtjev prosljeđuje na server koji nakon obrade zahtjeva vraća odgovor u obliku sadržaja koji se prikazuje unutar Internet preglednika. Uobičajeno je da Web aplikacija radi s bazom podataka koja joj nudi sve potrebne informacije o zahtijevanim resursima od strane korisnika. Najvažnija karakteristika Web aplikacija je njihova interaktivnost s krajnjim korisnicima te dinamičnost sadržaja. Ranije spomenuto Web mjesto restorana, kao primjer Web stranice, može vrlo jednostavno konvergirati u Web aplikaciju. Ukoliko bi Web mjesto restorana, uz sve ranije spomenute statične karakteristike, omogućilo kreiranje narudžbe, rezervaciju stolova ili pak ocjenjivanje zadovoljstva pruženom uslugom tada bi se ono ubrajalo u kategoriju Web aplikacija.



Slika 15. Arhitektura Web baziranih aplikacija [50]

5.4. HTTP (engl. *Hyper Text Transfer Protocol*)

Od samih početaka svog razvoja, koje seže u daleku 1990.-u godinu, World Wide Web postao je centralno, a posljedično i primarno, mjesto života mnogih ljudima, pružajući pritom korisnicima mogućnost rada, istraživanja, trgovine i društvenog povezivanja. Otvaranjem Web preglednika i pregledavanjem dostupnog sadržaja moguće je pristupiti gotovo beskonačnom svemiru informacija. Načela otvorenosti i standardizacije pravila, na kojima se temelji koncept Web-a, trebala bi omogućiti međusoban rad i komunikaciju između različitih Web aplikacija. [51] Upravo se kroz načelo otvorenosti omogućava interoperabilnost[9] koja, kao karakteristika Web-a, omogućava, uz poštivanje unaprijed definiranih pravila i dogovora, komunikaciju između različitih Web aplikacija. Bogatstvo medijskog sadržaja modernog Web-a je tijekom godina značajno evoluiralo te se više ne temelji na konceptu datoteka tekstualnog sadržaja. Međutim, unatoč svojoj evoluciji, Web je zadržao izvorni mehanizam dohvaćanja informacija te komunikacije koji se temelji na HTTP-u.

Prema definiciji, HTTP je protokol aplikacijskog sloja koji temeljem striktnog i formalno definiranog skupa pravila omogućava komunikaciju između klijenta, kao mrežnog resursa koji zahtijeva podatke, te poslužitelja, kao mrežnog resursa koji prima i procesira zahtjev te kreira korespondentni odgovor na primljeni zahtjev. [prema 51] Laički rečeno, HTTP je "jezik" virtualnog svijeta koji se, uz pomoć Interneta kao infrastrukture koja povezuje klijente i poslužitelje, koristi za ostvarenje njihove međusobne komunikacije. U osnovni, komunikacija između klijenta i poslužitelja temelji se na slanju HTTP zahtjeva, odnosno HTTP odgovora.

5.4.1. HTTP zahtjev (engl. *HTTP request*)

HTTP zahtjev je koncept putem kojeg platforme za Internetsku komunikaciju, kao što su Web preglednici, zahtijevaju informacije od krajnjih poslužitelja kako bi uspješno učitali Web stranicu ili podatke unutar Web aplikacije. [prema 53] Svaki HTTP zahtjev, kreiran u okviru Interneta kao mrežne infrastrukture, nosi sa sobom niz kodiranih podataka koji definiraju različite vrste informacija i podataka. Tipizacijom HTTP zahtjeva osigurava se razumijevanje sadržaja na strani poslužitelja, a standardizirani HTTP zahtjev sadrži:

1. HTTP verziju
2. URL poslužitelja kojem se šalje HTTP zahtjev
3. HTTP metodu
4. zaglavlja HTTP zahtjeva
5. tijelo HTTP zahtjeva (opcionalno)

[9] Interoperabilnost – sposobnost informacijskih i komunikacijskih sustava te poslovnih procesa da podržavaju razmjenu podataka te omogućuju dijeljenje informacija i znanja.

5.4.1.1. Verzije HTTP-a

U današnje vrijeme nemoguće je zamisliti inkubirani razvoj tehnologija. Drugim riječima, svaki napredak jedne tehnologije ima implicitan utjecaj na potrebu razvoja drugih tehnologija i koncepata. Kako je Web evoluirao kroz svoje generacije tako se javljala potreba za unaprjeđenjem koncepata koje pruža HTTP. Do danas je razvijeno šest verzija HTTP-a, koje se kronološkim redoslijedom dijele na HTTP/0.9, HTTPS, HTTP/1.0, HTTP/1.1, HTTP/2, HTTP/3. U današnje vrijeme najčešće se, upravo zbog svojih karakteristika, primjenjuju HTTP/1.1 te HTTP/2.0.

Zahtjevi HTTP/0.9 sastojali su se od jednoga retka i započinjali su, jedinom mogućom, GET metodom nakon čega je slijedila putanja do zahtijevanog resursa (URL). Identifikator resursa, odnosno URL, nije sadržavao apsolutnu putanju, koja bi uključivala navođenje protokola, adrese poslužitelja te porta, jer se resurs identificirao relativno nakon uspostave veze s krajnjim poslužiteljem. [prema 54]

```
GET /mypage.html
```

Slika 16. Zahtjev HTTP/0.9 protokola [54]

Odgovor HTTP/0.9 sastojao se od zahtijevanog resursa, odnosno HTML datoteke.

```
<html>
A very simple HTML page
</html>
```

Slika 17. Odgovor HTTP/0.9 protokola [54]

HTTP/0.9 je bio vrlo ograničen u pogledu mogućnosti koje nudi, tako da se ubrzo javila potreba za njegovim unaprjeđenjem i to u vidu HTTP/1.0. U sam redak zahtjeva dodana je informacija o verziji korištenog protokola. Osim toga, redak zahtjeva sadržava *User-Agent* zaglavlje koje specificira klijenta koji je kreirao samo zahtjev [prema 54] U suštini, sam format i sadržaj zahtjeva HTTP/1.0 nije značajno promijenjen. Međutim, pojavom HTTP/1.0 dolazi do uvođenja koncepta zaglavlja u HTTP zahtjeve što je imalo veliki utjecaj na daljnji smjer razvoja samog HTTP-a te njegovih mogućnosti.

```
GET /mypage.html HTTP/1.0
User-Agent: NCSA_Mosaic/2.0 (Windows 3.1)
```

Slika 18. Zahtjev HTTP/1.0 protokola [54]

Odgovor HTTP/1.0 značajno je promijenjen u odnosu na odgovor svog prethodnika te sadržava redak statusnog koda koji klijentima, odnosno Web preglednicima, omogućava prepoznavanje uspjeha, odnosno neuspjeha kreiranog zahtjeva kako bi sukladno tome prilagodili svoje ponašanje. Statusni kod igra posebnu ulogu prilikom određivanja načina korištenja lokalne predmemorije (engl. *local cache*) Web preglednika. Kao i kod zahtjeva, uveden je koncept HTTP zaglavlja koji pruža dodane informacije o samom HTTP odgovoru. Upravo primjenom koncepta zaglavlja, preciznije *Content-Type* zaglavlja, osim HTML datoteka, omogućen je i prijenos drugih tipova datoteka i podataka. [prema 54]

```
200 OK
Date: Tue, 15 Nov 1994 08:12:31 GMT
Server: CERN/3.0 libwww/2.17
Content-Type: text/html
<HTML>
A page with an image
  <IMG SRC="/myimage.gif">
</HTML>
```

Slika 19. Odgovor HTTP/1.0 protokola [54]

Paralelno s razvojem HTTP/1.0 radilo se na njegovoj standardizaciji. Prva standardizirana verzija HTTP-a, HTTP/1.1, predstavljen je široj javnosti početkom 1997. godine, samo nekoliko mjeseci nakon objave HTTP/1.0. HTTP/1.1 uveo je brojna poboljšanja od ponovne iskoristivosti jednom uspostavljene veze s krajnjim poslužiteljem pa sve do koncepta cjevovoda (engl. *pipes*) koji je omogućio asinkrono slanje HTTP zahtjeva. Asinkrono slanje zahtjeva omogućilo je da se novi HTTP zahtjev može poslati prije nego stigne odgovor na neki od ranije poslanih HTTP zahtjeva. [prema 54] Osim toga, uvedena su nova zaglavlja koja pružaju dodatne metapodatke.

```
GET /static/img/header-background.png HTTP/1.1
Host: developer.mozilla.org
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9; rv:50.0) Gecko/20100101
Firefox/50.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Referer: https://developer.mozilla.org/en-US/docs/Glossary/Simple_header
```

Slika 20. Zahtjev HTTP/1.1 protokola [54]

Odgovor HTTP/1.1 nije se značajno mijenjao u odnosu na svog prethodnika. Jedina novost je uvođenje podijeljenih odgovora. Na taj je način omogućena raspodjela odgovora s ogromnim *payload-om* na manje dijelove (engl. *chunk*) te ponovno sastavljanje HTTP odgovora na odredišnoj strani.

```
200 OK
Age: 9578461
Cache-Control: public, max-age=315360000
Connection: keep-alive
Content-Length: 3077
Content-Type: image/png
Date: Thu, 31 Mar 2016 13:34:46 GMT
Last-Modified: Wed, 21 Oct 2015 18:27:50 GMT
Server: Apache

(image content of 3077 bytes)
```

Slika 21. Odgovor HTTP/1.1 protokola [54]

Tijekom godina, Web aplikacije postajale su sve složenije te se javlja potreba za sofisticiranijim oblicima komunikacije. Upravo u svrhu podrške ubrzanom razvoju tehnologija, 2015. godine razvijen je HTTP/2.0. Prije svega, valja napomenuti kako je HTTP/2.0 binarni protokol što znači da je ljudima nečitljiv te se kao takav ne može kreirati ručno. [54] HTTP/2.0 je multipleksirani protokol koji omogućuje kreiranje paralelnih HTTP zahtjeva preko iste TCP veze. Na taj je način HTTP/2.0 uklonio ograničenja ranijih verzija, odnosno verzija HTTP/1.x. Zaglavlja HTTP/2.0 zahtjeva i odgovora komprimirana su čime se uvodi dodatna razina optimizacije te ubrzava sam proces komunikacije između klijenta te poslužitelja.

Kao okosnica različitih verzija HTTP-a, 2019. godine je javno objavljen HTTP/3.0 koji objedinjuje sve dobre karakteristike svojih prethodnika uz primjenu novih koncepata i tehnologija. Glavna razlika u odnosnu na prethodne verzije je da se na transportnom sloju, umjesto standardnog TCP-a ili UDP-a (engl. *User Datagram Protocol*), koristi QUIC (engl. *Quick UDP Internet Connections*) protokol koji pruža različite mogućnosti prijenosa podataka. U osnovi, QUIC protokol je šifrirani protokol opće namjene koji omogućava multipleksiranje više tokova podataka na jednoj vezi. [prema 55] U literaturi je HTTP/3.0 poznat pod naziv *HTTP/2.0-over-QUIC*. Glavna značajka koju uvodi QUIC je poboljšana privatnost uz zadržavanje postojeće razine sigurnosti koju pruža HTTP/2.0. Iako se HTTP/3.0 nalazi u eksperimentalnoj fazi, te prema posljednjim podacima koristi kod svega 8% Web aplikacija, pred njim je blistava budućnost, posebice u kombinaciji s generacijom Web 3.0.

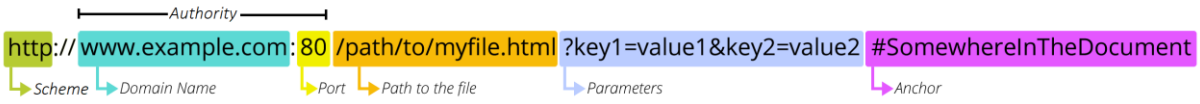
5.4.1.2. URL (engl. Uniform Resource Locator)

URL je, uz hipertekst i HTTP, jedan od ključnih koncepata modernog Web-a koji se primarno koristi kao mehanizam kojeg Web preglednici primjenjuju prilikom dohvaćanja bilo kojeg resursa koji je javno objavljen na Web-u. Kao što i samo ime govori, URL je adresa jedinstvenog resursa na Web-u. [56] URL se uobičajeno koristi za dohvaćanje resursa kao što je HTML stranica, CSS dokument, slika i slično. Budući da resursom, koji je predstavljen jedinstvenim URL-om, upravlja Web poslužitelj, na vlasniku Web poslužitelja je da pažljivo upravlja specificiranim resursom te njegovim mapiranjem na korespondentni URL.

```
https://developer.mozilla.org
https://developer.mozilla.org/en-US/docs/Learn/
https://developer.mozilla.org/en-US/search?q=URL
```

Slika 22. Primjeri URL-a [56]

URL se sastoji od nekoliko različitih dijelova među kojima su neki obavezni, a neki opcionalni. URL se može poistovjetiti s, ljudima poznatijom, poštanskom adresom. *Schema* u tom kontekstu reprezentira poštansku uslugu koja se koristi, *Domain Name* reprezentira grad ili mjesto isporuke, *Port* reprezentira poštanski broj dok *Path* reprezentira konkretnu adresu isporuke. *Schema*, *Domain Name*, *Port* i *Path* su obavezni dijelovi svakog URL-a koji u svojoj kombinaciji jednoznačno identificiraju svaki pojedini resurs koji je dostupan na Web-u. Osim obaveznih, URL adresa sadrži i nekoliko opcionalnih dijelova među kojima su najvažniji i najčešće korišteni *Parameters* te *Anchor*. U spomenutom kontekstu poštanske adrese, *Parameters* reprezentira dodatne informacije o samoj adresi isporuke. Primjerice, ukoliko je adresa isporuke povezana sa zgradom, tada se uz pomoć *Parameters* omogućuje specificiranje dodatnih informacija o samoj adresi kao što su broj stana ili konkretni kat zgrade. Uz pomoć *Anchor* specificira se stvarna osoba kao konkretni primatelj pošiljke. [prema 56]



Slika 23. Struktura URL-a [56]

URL predstavlja čitljivu ulaznu točku koja se koristi za pristupanje Web stranici ili Web aplikaciji. Upravo zbog svojih karakteristika, URL adrese se vrlo lako pamte te unose u adresnu traku Web preglednika. URL adresa definira određene restrikcije kojih se potrebno pridržavati prilikom njezine izgradnje. Prije svega, URL adresa može sadržavati slova, brojeve te znakove (,), $, –, ', _, * te +. Pojedine riječi unutar URL adrese odvajaju se upotrebom znakova – ili _ zbog restrikcija vezanih uz upotrebu razmaka. Prilikom pristupanja resursu dolazi do mapiranja URL adrese na pripadajuću IP adresu za što je zadužen DNS (engl. *Domain Name System*).

5.4.1.3. HTTP metoda

HTTP definira osam "glagola" uz pomoć kojih Web klijent može zatražiti dohvaćanje, ažuriranje ili upravljanje sadržajem resursa koji se nalazi na Web poslužitelju. U kontekstu HTTP-a i Web-a, spomenutih osam glagola naziva se HTTP metodama. [51] HTTP metoda označava radnju koja se kroz HTTP zahtjev šalje poslužitelju kako bi bila procesirana te izvršena od strane zahtijevanog poslužitelja. [53]

HTTP metode za dohvaćanje resursa (tipično Web sadržaja):

- HEAD
- GET

HTTP metode za ažuriranje resursa:

- POST
- PUT
- PATCH

HTTP metoda za brisanje resursa:

- DELETE

HTTP metoda za debugging poruke:

- TRACE

HTTP metoda za dohvaćanje informacija o Web poslužitelju (dozvoljene metode i operacije):

- OPTIONS

HTTP metoda za kreiranje mrežne veze:

- CONNECT

Od spomenutih metoda, u praksi se najčešće koriste GET, POST, PUT i DELETE HTTP metode koje omogućuju bazičnu CRUD (engl. *Create, Read, Update, Delete*) funkcionalnost Web aplikacija. HTTP zahtjev GET metodom zauzvrat očekuje informacije od Web poslužitelja, dok HTTP zahtjev POST metodom označava slanje informacija Web poslužitelju. Primjerice, HTTP zahtjev GET metodom koristi se za dohvaćanje podataka o proizvodima koji su dostupni u okviru elektroničke trgovine, dok se HTTP zahtjev POST metodom koristi za dodavanje novog proizvoda u postojeći proizvodni asortiman elektroničke trgovine. S druge pak strane, HTTP zahtjev PUT metodom koristi se za ažuriranje resursa na Web poslužitelju, dok se HTTP zahtjev DELETE metodom koristi za brisanje resursa s Web poslužitelja. Primjerice, HTTP zahtjev PUT metodom koristi se za ažuriranje sadržaja korisničke košarice, dok se HTTP zahtjev DELETE metodom koristi za brisanje proizvoda koji su dodani u korisničku košaricu. [prema 53] Neke od HTTP metoda definiraju dodatne restrikcije i preporuke prilikom konstrukcije HTTP zahtjeva koje nisu nužno obavezne. Primjerice, POST metoda zahtijeva definiranje sadržaja unutar tijela HTTP zahtjeva, dok s druge strane GET metoda zabranjuje definiranje sadržaja unutar tijela HTTP zahtjeva.

5.4.1.4. Zaglavlja HTTP zahtjeva

Zaglavlja HTTP zahtjeva sadrže tekstualne informacije pohranjene u JSON (engl. *JavaScript Object Notation*) formatu, odnosno u obliku parova oblika *ključ:vrijednost*. Zaglavlja su uključena u svaki HTTP zahtjev pružajući pritom metainformacije te metapodatke o kreiranom HTTP zahtjevu. [prema 53] U suštini, zaglavlja HTTP zahtjeva sadrže informacije koje Web poslužitelj koristi prilikom procesiranja HTTP zahtjeva kao što je informacija o Web pregledniku kojeg klijent koristi, tipu podatka kojeg klijent očekuje kao odgovor, autentifikacijske podatke i slično.

Slika 24. Primjer zaglavlja HTTP zahtjeva [53]

5.4.1.5. Tijelo HTTP zahtjeva (engl. *request body*)

Tijelo HTTP zahtjeva sadrži informacije i podatke koji se šalju Web poslužitelju s ciljem njihovog daljnjeg procesiranja na strani servera. [prema 53] Kada govorimo o tijelu HTTP zahtjeva tada ono uobičajeno sadrži podatke koji su uneseni putem formi na strani klijenta. Format podataka koji se šalju unutar tijela HTTP zahtjeva nije strogo specificiran, a neki od dozvoljenih formata su *form-data*, *x-www-form-urlencoded*, *raw (Text, JavaScript, JSON, HTML, XML)*, *binary* te *GraphQL*. Format podataka definira se putem *content-type* zaglavlja HTTP zahtjeva. Time se omogućava pružanje metainformacija o samom HTTP zahtjevu kako bi Web poslužitelj razumio te na ispravan način procesirao primljeni HTTP zahtjev.

Slika 25. Primjer tijela HTTP zahtjeva u JSON formatu [57]

5.4.2. HTTP odgovor (engl. *HTTP response*)

HTTP odgovor je koncept putem kojeg platforme za Internetsku komunikaciju, kao što su Web preglednici, na temelju kreiranih HTTP zahtjeva primaju informacije od krajnjih poslužitelja kako bi uspješno učitali Web stranicu ili podatke unutar Web aplikacije. HTTP odgovori prenose informacije o resursima koje Web preglednici zahtijevaju kreiranjem odgovarajućeg HTTP zahtjeva. HTTP zahtjev djeluje kao potvrda o uspješnosti izvršavanje kreirane akcije, odnosno HTTP zahtjeva. [prema 58] U slučaju pogreške tijekom izvršavanja HTTP zahtjeva Web poslužitelj odgovara odgovarajućom porukom pogreške. Informacije i podaci prenose se unutar tijela HTTP odgovora. Tipizacijom HTTP odgovora osigurava se razumijevanje sadržaja na strani klijenta, a standardizirani HTTP odgovor sadrži:

1. HTTP statusni kod
2. zaglavlja HTTP odgovora
3. tijelo HTTP odgovora (opcionalno)

5.4.2.1. HTTP statusni kod

HTTP statusni kod je troznamenkasti broj koji se koristi za označavanje uspjeha ili neuspjeha tijekom izvršavanja HTTP zahtjeva od strane Web poslužitelja. [prema 53] HTTP statusni kodovi podijeljeni su u pet kategorijskih blokova:

- 1xx – kategorija *informativnih* statusnih kodova
- 2xx – kategorija *uspješnih* statusnih kodova
- 3xx – kategorija statusnih kodova za *preusmjeravanje*
- 4xx – kategorija statusnih kodova za *pogreške na strani klijenta*
- 5xx – kategorija statusnih kodova za *pogreške na strani Web poslužitelja*

Oznaka "*xx*" označava raspon brojeva između 0 i 99. [prema 53]

Primjerice, statusni kodovi iz kategorije 2xx označavaju da je HTTP zahtjev uspješno izvršen. U većini slučajeva, nakon što klijent zatraži Web stranicu ili podatke Web aplikacije, HTTP odgovor sadrži status "200 OK", čime Web poslužitelj naglašava kako je kreirani HTTP zahtjev uspješno obrađen. Ako klijent kreira HTTP zahtjev primjenom POST metode te je zahtjev uspješno obrađen na strani Web poslužitelja tada HTTP odgovor sadržava status "201 Created". Statusni kodovi iz kategorije 4xx i 5xx označavaju da je došlo do pogreške, na strani klijenta ili na strani Web poslužitelja, te se Web stranica u tom slučaju neće prikazati. Ako klijent pristupa URL-u koji nije u domeni Web stranice ili Web aplikacije tada će HTTP odgovor sadržavati status "404 Not Found". Ako dođe do problema prilikom obrade zahtjeva na strani Web poslužitelja tada HTTP odgovor sadržava status "500 Internal Server Error".

5.4.2.2. Zaglavlja HTTP odgovora

HTTP odgovor, kao i HTTP zahtjev, ima definirano zaglavlje koje sadrži metapodatke kreiranog HTTP odgovora. Zaglavlja HTTP odgovora prenose važne informacija kao što su jezik i format podataka koji se prenose unutar tijela HTTP odgovora. [53] Sam format zapisa podataka unutar zaglavlja ekvivalentan je kao i kod HTTP zahtjeva. Razlika između zaglavlja HTTP zahtjeva i HTTP odgovora je u ključevima koji su sadržani u samom zaglavlju. Uz pomoć zaglavlja HTTP odgovora, Web poslužitelj eksplicitno pruža informacije koje klijent koristi prilikom daljnjeg rukovanja s primljenim podacima zahtijevanog resursa.

Slika 26.Primjer zaglavlja HTTP odgovora [53]

5.4.2.3. Tijelo HTTP odgovora (engl. *response body*)

Tijelo HTTP odgovora sadrži informacije i podatke zahtijevanog resursa. Drugim riječima, HTTP odgovor sadrži podatke koji su zahtijevani od strane klijenta na temelju kreiranog HTTP zahtjeva. Važno je spomenuti kako je tijelo opcionalni dio HTTP odgovora. Tijelo se uobičajeno koristi prilikom kreiranja HTTP odgovora koji je povezan s korespondentnim HTTP zahtjevom klijenta koji se temelji na GET metodi. U većini slučajeva, HTTP odgovor sadrži podatke koje će Web preglednik prevesti te transformirati u Web stranicu. [prema 53] Format podataka HTTP odgovora nije strogo specificiran, a neki od dozvoljenih formata su *JSON, HTML* te *XML*.

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html>
<head>
   <title>404 Not Found</title>
</head>
<body>
   <h1>Not Found</h1>
   <p>The requested URL /t.html was not found on this server.</p>
</body>
</html>
```

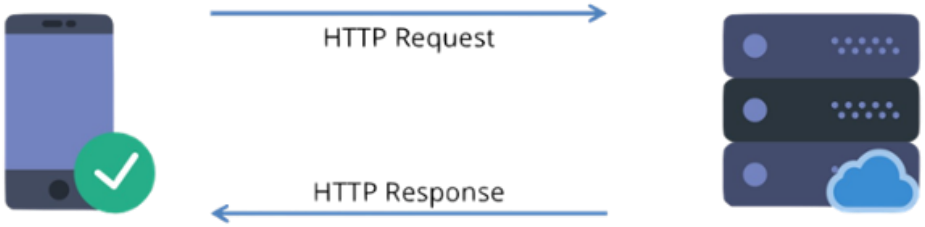
Slika 27. Primjer HTTP odgovora u HTML formatu [59]

5.5. Web servisi

Kako se Web tijekom godina razvijao, ponajprije iz statičnih Web stranica pa sve do dinamičnih Web aplikacija, tako se javljala potreba za novim i inovativnim načinima komunikacije između raspodijeljenih sustava. Razvoj i sama primjena višeplatformskih, modernih aplikacija više se ne javlja kao mogućnost već kao svakodnevna potreba. Web servisi javljaju se kao koncept koji nameće rješenje problema komunikacije između različitih sustava. Web servisi su klijentske i poslužiteljske aplikacije koje komuniciraju putem HTTP-a, pružajući pritom standardno sredstvo interoperabilnosti između različitih aplikacija koje se mogu izvršavati na različitim platformama i okvirima. [60]

Web servisi karakteriziraju se kao centralno mjesto za agregiranje podataka i informacija iz različitih izvora koje pritom služi kao primarno sredstvo interoperabilnosti između različitih klijentskih i poslužiteljskih aplikacija. Uz interoperabilnost, Web servisi karakterizirani su jednostavnom proširivošću te po računalno obradivim opisima, ponajprije zahvaljujući XML-u, a kasnije i JSON-u. [60] Pojam Web servisa definiran je od strane W3C (engl. *World Wide Web Consortium*) te stoga slijedi strogi niz tehničkih standarda i pravila. Sučelje Web servisa opisano je putem WSDL-a (engl. *Web Services Description Language*) kao formata koji omogućuje strojno procesiranje podataka i informacija. [prema 61] Korisnik i pružatelj Web servisa koriste poruke za razmjenu informacija, slanjem zahtjeva i odgovora, koje su u obliku samoodrživih dokumenata koji imaju vrlo malo pretpostavki o tehnološkim osobinama primatelja. [60]

Razvoj Web servisa utjecao je na promjene u arhitekturama koje se koriste za prijenos podataka. Unazad nekoliko godina, SOAP (engl. *Simple Object Access Protocol*) i REST su bili jedini arhitekturni modeli za razvoj Web servisa te tako prevladavali i dominirali na svjetskom tržištu Web baziranih aplikacija. U suštini, obje arhitekture koriste HTTP za prijenos podataka te URI (engl. *Uniform Resource Identifier*) standard. Razlika je u tome što se URI standard kod SOAP-a koristi za identifikaciju servisa, a kod REST-a za identifikaciju resursa. [65] U novije vrijeme, sve je popularniji GraphQL koji nudi nove mogućnosti prijenosa podataka i informacija između aplikacija.



Slika 28. Komunikacija između klijenta i poslužitelja [62]

5.5.1. REST(ful) Web servisi (engl. *Representational State Transfer*)

Termin REST akronim je za *Representational State Transfer*, u prijevodu prijenos reprezentacijskog stanja. U svojoj srži, REST je arhitekturni stil koji definira skup arhitekturnih pravila i ograničenja prilikom razvoja distribuiranih hipermedijskih sustava. Važno je naglasiti kako REST nije protokol ili standard već arhitekturni stil. Sam termin je skovao Roy Thomas Fielding davne 2000. godine. [prema 66] Roy Thomas Fielding je u svojoj disertaciji *Architectural Styles and the Design* of *Network-based Software Architectures* iznio opis novog arhitekturnog stila koji se temeljio na postojećim stilovima, ali uz dodatak ograničenja koja su specifična za taj novi stil. Svaki stil unutar mrežne arhitekture možemo definirati kao skup arhitekturnih ograničenja koja nameću dozvoljene odnose između pojedinih elemenata arhitekture. [68] Prema *Architectural Styles and the Design of Network-based Software Architectures*, svaku arhitekturu, pa tako i REST, definiraju tri osnovna tipa elemenata koji se koriste za postizanje željenih svojstava, a to su *komponente*, *konektori* te *podaci*. Komponente možemo poistovjetiti sa sustavima koji vrše manipulacije nad podacima, dok pojedine komponente predstavljaju gradivne blokove cjelokupnog sustava koji se međusobno povezuju konektorima. Gupta u svom članku REST *Architectural Constraints* navodi kako je REST arhitekturni stil koji omogućava projektiranje i razvoj slabo povezanih (engl. *loosely coupled*) aplikacija koje komuniciraju putem Interneta.

REST se zasniva na prijenosu stanja u kojem se adresirani resurs trenutno nalazi. REST Web servisi se vrlo jednostavno integriraju s HTTP-om te nisu čvrsto vezani uz korištenje XML poruka za razmjenu podataka ili WSDL-a za opis servisa. Podaci se prenose u JSON standardu kao uobičajenom formatu za razmjenu poruka tijekom komunikacije te XML-u i YAML-u (engl. *YAML Ain't Markup Language*) kao dodatnim standardima za definiranje formata poruka.

U literaturi se vrlo često isprepliću pojmovi REST Web servisi te RESTful Web servisi, pri čemu je važno naglasiti kako se ne radi o istoznačnim pojmovima. REST definira šest dodatnih arhitekturnih ograničenja temeljem kojih neki Web servis možemo proglasiti RESTful Web servisom. Dakle, RESTful je specijalizacija REST arhitekturnog stila koja definira dodatna ograničenja kao što su *klijent-poslužitelj arhitektura*, *korištenje priručne memorije, jedinstvenost sučelja*, *neovisnost zahtjeva*, *slojeviti sustav* te *kod na zahtjev*. [66] REST servisi su servisi bez stanja što znači da svaki HTTP zahtjev mora uključivati sve podatke koji su poslužitelju potrebni za obradu. Samim time, klijent je odgovoran za upravljanje stanjem aplikacije pošto se korisnički kontekst ne pohranjuje na poslužitelju. Slojevitom arhitekturom sustava, REST servisi omogućuju obradu HTTP zahtjeva kroz više različitih arhitekturnih slojeva (engl. *layers*) čime se omogućava prosljeđivanje ili preusmjeravanje kreiranog zahtjeva prema drugim Web servisima. Takav zahtjev poznatiji je kao *backend-to-backend* zahtjev. [67]

5.5.1.1. Elementi podataka

Roy Thomas Fielding, u svojoj disertaciji *Architectural Styles and the Design of Network-based Software Architectures* navodi kako se u sustavima distribuiranih objekata podaci nalaze unutar komponenata koje vrše obradu nad skupom podataka. Kod REST arhitekture, stanje pojedinog podatka iz skupa igra temeljnu ulogu prilikom slanja odgovora na primljeni aplikacijski zahtjev.

U distribuiranom sustavu podaci se prenose od mjesta na kojem su pohranjeni pa sve do mjesta na kojem su zahtijevani. Za ostvarenje prijenosa podataka koristi se koncept poveznice koji, dizajnerima distribuiranih sustava, daje razne mogućnosti prilikom implementacije Web servisa, od *skrivene implementacije programa*, pa sve do *pakiranja podataka i programa* ili *slanja neobrađenih podataka*. [prema 68]

Prva opcija implementacije omogućava distribuiranim sustavima sakrivanje implementacije programa koji vrši obradu podataka na temelju primljenog zahtjeva te nakon pronalaska traženog podatka rezultat se šalje kao odgovor klijentu. Međutim, sakrivanjem implementacije programa ograničava se paralelna obrada primljenih zahtjeva, odnosno mogućnost obrađivanja većeg broja zahtjeva istovremeno zbog potrebe vršenja lokalne obrade podataka na svaki od primljenih zahtjeva. Iako se čini idealnom opcijom implementacije, u praksi se vrlo rijetko koristi zbog činjenice da poslužitelj mora slijedno obrađivati primljene zahtjeve. Slijedna obrada na strani poslužitelja proizlazi iz ranije spomenute potrebe za lokalnom obradom podataka svakog primljenog zahtjeva. [prema 68]

Druga opcija implementacije omogućava distribuiranim sustavima da u odgovor na primljeni zahtjev upakiraju tražene podatke i program koji je potreban za prezentaciju podataka iz odgovora. Najveći nedostatak ovakvog pristupa je prijenos većeg broja datoteka u samom odgovoru. Naime, potreba za prijenosom većeg broja datoteka proizlazi iz razloga što odgovor na primljeni zahtjev sadrži podatke i program za prezentaciju tih podataka. Međutim, skrivanje podataka je još uvijek omogućeno te se kao takvo primjenjuje. [prema 68]

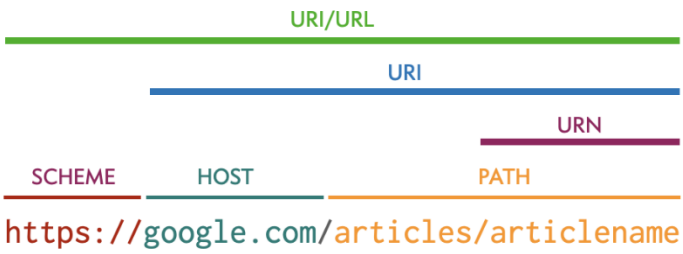
Trećom opcijom implementacije distribuiranim sustavima se omogućava da u odgovor mogu staviti neobrađene podatke i metapodatke koji su potrebni za sam opis neobrađenih podataka. Na taj se način klijentu omogućava odabir programa koji će koristiti prilikom obrade podataka koje je primio u odgovoru. Ovom opcijom, omogućava se višedretveni rad poslužitelja zbog toga što poslužitelj ne vrši lokalnu obradu podataka pa odgovor sadrži neobrađene podatke. Važno je da klijent i poslužitelj razumiju tip podatka koji se prenosi u zahtjevu ili odgovoru kako bi mogli razumjeti kontekst primljene poruke te na taj način pravilno komunicirati. Iako ima svojih prednosti, ovakav način rada dizajnerima osporava mogućnost enkapsulacije, odnosno sakrivanja podataka. [prema 68]

REST arhitektura predstavlja hibrid triju navedenih opcija koje nude distribuirani sustavi pošto daje mogućnost razumijevanja tipova podataka, ali istovremeno, uz pomoć standardiziranog sučelja, ograničava vidljivost svoje unutrašnjosti. Komponente koje definiraju REST arhitekturu imaju mogućnost odabira načina reprezentacije zahtijevanog resursa ovisno o standardnom tipu podatka ili prirodi resursa. [prema 68] Primjerice, ukoliko postoji sučelje za rad sa slikovnim sadržajima tada će se birati između JPG, JPEG ili PNG formata pri čemu je originalni tip resursa skriven iza samog sučelja. REST arhitektura omogućava odvajanje odgovornosti između klijenta i poslužitelja kroz samostalni razvoj navedenih komponenata te sakrivanje informacija iza samog sučelja.

Roy Thomas Fielding u svojoj disertaciji spominje šest temeljnih elemenata podataka među kojima su *resurs*, *identifikator resursa*, *reprezentacija resursa*, *metapodaci reprezentacije resursa*, *metapodaci samog resursa* te *kontrolni podaci*. [68]

Resurs predstavlja konceptualnu metu hipertekst reference. Resurs se uobičajeno definirati kao apstrakcija koja predstavlja imenovani materijalni i/ili nematerijalni entitet. Resurs može biti slika, zvuk, video ili drugi servis kojeg REST zahtijeva. Roy Thomas Fielding u svojoj disertaciji navodi kako se svaki od resursa preslikava na skup entiteta. Resursi se prezentiraju reprezentacijama, odnosno XML, JSON, YAML i drugim validnim formatima. Resurs se dohvaća na temelju identifikatora resursa. Konvencija nalaže da resursi budu imenice. Primjerice, kolekcija podataka o korisnicima definira se kao resurs `/users`. [prema 68]

Identifikator resursa, odnosno URI, predstavlja jednoznačni naziv pomoću kojeg se ostvaruje pristup zahtijevanom resursu. Sam URI nije reprezentacija resursa već samo oznaka kojom se jednoznačno identificira pojedini resurs na poslužitelju. Općenito razlikujemo dvije vrste identifikatora resursa, a to su *statički* i *dinamički*. Statički identifikatori pokazuju na resurse čija se reprezentacija ne mijenja tijekom vremena, dok dinamički identifikatori pokazuju na resurse čiji se parametri reprezentacije mogu mijenjati. [prema 68] Važno je naglasiti kako URL i URI nisu istoznačnice. URI se koristi kao jedinstveni identifikator resursa dok URL sadrži informacije vezane uz pribavljanje resursa. U tom pogledu, URL je specijalizacija URI-a. Primjerice, `http://localhost:8080/api/users/:id` predstavlja jednoznačni identifikator resursa.



Slika 29. Razlika između URL-a i URI-a

Reprezentacija resursa definira format podataka zahtijevanog resursa. REST obavlja radnje nad resursom kako bi oblikovao željeno stanje resursa, odnosno reprezentaciju resursa koja će se koristiti prilikom same komunikacije. Reprezentacije predstavlja skup okteta koji je dobiven od nekog resursa te skup metapodataka reprezentacije koji opisuju način formatiranja resursa. [prema 68]. Kada govorimo o reprezentaciji resursa tada REST koristi različite formate za prijenos podataka. U današnje vrijeme najčešće se primjenjuje JSON format iako ni XML te YAML nisu rijetkost. Važno je spomenuti da je reprezentacija resursa način komunikacije te da sami resursi ostaju nepromijenjeni i enkapsulirani.

Metapodaci reprezentacije referenciraju HTTP metode zaglavlja. Oni predstavljaju parove ključ-vrijednost koji se koriste za definiranje informacija o reprezentaciji koje koristi sam sustav. Obično se radi o podacima koji nisu direktno dostupni i vidljivi samom korisniku. [68] Metapodaci reprezentacije obično definiraju vrstu medija, datum izrade, datum modifikacije, broj verzije i slično. Metapodaci reprezentacije čine podatke iz zaglavlje HTTP zahtjeva ili odgovora. [80] Primjerice, metapodaci reprezentacije mogu sadržavati podatke o tipu reprezentacije resursa (engl. *Content-Type*), zadnjoj modifikaciji zahtijevanog resursa (engl. *Last-Modified*), token za autorizaciju korisnika (engl. *Authorization*), MIME tip (engl. *Multipurpose Internet Mail Extensions*) i slično.

Metapodaci resursa nisu samo opisi resursa već i dodatne informacije koje se pružaju sustavu kako bi on bio svjestan postojanja resursa na poslužitelju. Uz metapodatke resursa najčešće vežemo poveznice izvora, alternativne resurse te informacije o resursu. [81] Primjerice, metapodaci resursa mogu uključiti zamjenski tekst koji se prikazuje ukoliko dohvaćanje slike nije bilo uspješno te ju samim time moguće prezentirati. Primjer možemo poistovjetiti s *alt* atributom u HTML elementu *img*.

Kontrolni podaci namijenjeni su validaciji resursa te njegovoj reprezentaciji na strani korisnika. Kontrolni podaci obično obuhvaćaju provjeru predmemoriranja podataka, vrijeme isteka podataka ili pružaju ograničenja vezana uz korištenje podataka određenog resursa. [81] Vrlo često se koriste za osiguravanje integriteta podataka kako ne bi došlo do neželjenih izmjena tijekom transporta informacija između dviju krajnjih točaka komunikacijskog kanala. Kao što je i ranije navedeno, kontrolni podaci obuhvaćaju metode HTTP zaglavlja koje određuju način rukovanja priručnom memorijom te način reprezentacije resursa na način da ih klijent ili poslužitelj mogu prepoznati. Uobičajeni kontrolni podatak HTTP zahtjeva je *Accept* koji definira MIME tipove koje klijent može razumjeti i procesirati. To je posebno važno u slučajevima kada resurs ima više različitih reprezentacija pa sam servis mora omogućiti mehanizam koji će odrediti zahtijevanu reprezentaciju od strane korisnika. [prema 68] U suštini, kontrolni podaci predstavljaju ugrađene HTTP metode koje kontroliraju događaje na poslužiteljskoj strani te paralelno kontroliraju priručnu memoriju.

5.5.1.2. Elementi konektora

Roy Thomas Fielding, u svojoj disertaciji *Architectural Styles and the Design of Network-based Software Architectures* razlikuje pet osnovnih tipova elemenata konektora među kojima su *klijent*, *poslužitelj*, *priručna memorija*, *rješavač* (engl. *resolver*) i *tunel*. Elementi konektora koriste se za opis aktivnosti pristupa te prijenosa reprezentacije resursa. [prema 68]

Elementi konektora koriste se za kreiranje apstraktnog sučelja za komunikaciju između pojedinih REST komponenti. Primjena apstraktnog sučelja, preko kojeg klijenti pristupaju servisu, omogućuje promjene u implementaciji sučelja bez promjene samog sučelja. Na taj je način omogućeno da nove promjene sučelja ne utječu na način korištenja samog servisa već su sve promjene odrađene na način da ''podržavaju'' postojeće sučelje. Vrlo važna karakteristika konektora je da omogućavaju komponentama prijenos velike količine podataka kroz više različitih poziva. [68] Navedena karakteristika omogućena je kroz neovisnost svakog pojedinog zahtjeva, odnosno zahtjev koji je trenutno primljen, kao i njegova obrada, ne ovise o prethodno primljenim zahtjevima od strane određene komponente (engl. *statelessness*).

Konektor klijent prvi je od dvaju ključnih elemenata koji su zaduženi za komunikaciju unutar sustava. Zadužen je za slanje zahtjeva na poslužitelj kako bi dobio odgovor u obliku tražene usluge ili podataka od strane poslužitelja. Klijent je zadužen za kreiranje odgovarajućih HTTP zahtjeva sukladno potrebama korisničkog sučelja kao i obradu primljenih HTTP odgovora.

Konektor poslužitelj drugi je ključni element zadužen za komunikaciju unutar sustava. Zadužen je za osluškivanje klijentskih zahtjeva na sučelju te obradu zahtjeva i generiranje odgovora koji se zatim vraća klijentu koji je uputio zahtjev. Drugim riječima, konektor poslužitelj isporučuje informacije, podatke i datoteke koje klijent, kreiranjem HTTP zahtjeva, zahtijeva od samog poslužitelja.

Konektor priručne memorije zadužen je za analizu zahtjeva koji pristižu na poslužiteljsko ili klijentsko sučelje sa svrhom *cacheiranja* odgovora na često postavljene zahtjeve. Na taj je način omogućena otpornost poslužitelja na povećani mrežni promet jer odgovori na neke korisničke zahtjeve postoje u priručnoj memoriji iz koje se onda dohvaćaju čime se reducira vrijeme potrebno za dobivanje odgovora od strane poslužitelja. [68]

Konektor *resolver* zadužen je da iz identifikatora resursa (URI) identificira ime domene kako bi istu mogao poslati DNS (engl. *Domain Name System*) *resolver-u*, koji će na temelju primljenog naziva domene vratiti traženu IP adresu domene na koju se zatim dodaje ostatak putanje identifikatora resursa kako bi se formirao korespondentni HTTP zahtjev.

Konektor tunel zadužen je za prijenos komunikacije iz zatvorene mreže do poslužitelja preko vatrozida. [5] Ova komponenta održava direktnu vezu između klijenta i poslužitelja.

5.5.1.3. Elementi komponente

Roy Thomas Fielding, u svojoj disertaciji *Architectural Styles and the Design of Network-based Software Architectures* navodi kako elementi komponente predstavljaju dijelove koji imaju ulogu kreiranje zahtjeva, prosljeđivanja zahtjeva te odgovora kao i točke koje primaju zahtjeve klijenata te generiraju odgovore. Pritom, Roy Thomas Fielding razlikuje četiri elementa komponente među kojima su *izvorni poslužitelj*, *poveznik*, *posrednik* te *korisnički agent*. Pojedina komponenta REST arhitekture tipizirana je sukladno njezinoj ulozi u cjelokupnoj REST aplikacijskog arhitekturi.

Komponenta izvorni poslužitelj zadužena je da, uz pomoć konektora poslužitelja, nadzire resurse kojima poslužitelj raspolaže. Osim toga, komponenta izvorni poslužitelj je ''pravi'' izvor reprezentacije resursa koji su pohranjeni. Komponenta izvornog poslužitelja pruža generičko sučelje za pristup resursima koji su dostupni na samom poslužitelju. Na taj se način omogućava sakrivanje detalja implementacije zahtijevanog resursa.

Komponenta poveznik te komponenta posrednik ubrajaju se u kategoriju međukomponenata iz razloga što se navedene komponente istovremeno ponašaju i kao klijent i kao poslužitelj.

Međukomponenta poveznik (engl. *gateway*), koja je nametnuta od strane mreže ili glavnog poslužitelja, zadužena je za filtriranje zahtjeva između vanjskog svijeta i glavnog poslužitelja. [68] Osnovni cilj komponente poveznik je zaštita glavnog poslužitelja te poboljšanje performansi poslužitelja. Ujedno, komponenta poveznik služi kao prevoditelj podataka koji stižu unutar tijela HTTP odgovora.

Međukomponenta posrednik (engl. *proxy*), koja je nametnuta i odabrana je od strane korisničkog agenta, zadužena je za prosljeđivanje zahtjeva korisničkog agenta te prosljeđivanje odgovora poslužitelja do korisničkog agenta. [68] Vrlo često se koristi kao točka koja smanjuje opterećenje glavnog poslužitelja primjenom priručne memorije iz koje će se generirati odgovori na često postavljene zahtjeve.

Komponenta korisnički agent zadužena je da, uz pomoć konektora klijenta, generira klijentski zahtjev. Osim toga, korisnički agent je zadužen za primanje odgovora od strane poslužitelja na upućene zahtjeve. Iniciranje HTTP zahtjeva komponenta korisničkog agenta postaje krajnji, odnosno konačni primatelj HTTP odgovorta. U skupinu komponente korisničkih agenata ubrajaju se i web preglednici. [68]

5.5.1.4. REST arhitekturna ograničenja

REST arhitektura predstavlja hibrid raznih arhitekturnih stilova iz kojih koristi samo najbolje karakteristike te ih objedinjuje u zajedničku cjelinu. Upravo zbog sinergije karakteristika različitih stilova javlja se potreba za postavljanjem ograničenja koja će biti primijenjena na elemente REST arhitekture. Prema Roy Thomas Fielding [68] te Tutorials Point [8] ograničenja koja se postavljaju na REST arhitekturu su *klijent-poslužitelj, priručna memorija*, *jedinstvenost sučelja*, *neovisnost zahtjeva*, *slojeviti sustav* te *kod na zahtjev.* Uvođenjem arhitekturnih ograničenja omogućuje se implementacija RESTful Web servisa.

Ograničenje klijent-poslužitelj uvodi princip raspodijeljene odgovornosti čime se omogućava da komponenta klijent, neovisno o komponenti poslužitelj, razvija prikaz korisničkog sučelja i implementaciju načina komunikacije s poslužiteljem. Na taj način REST dobiva dodatnu karakteristiku kojom se omogućava prenosivost sučelja komponenata na različite platforme. Osim toga, poslužitelj više ne treba biti fokusiran na vezivanje podataka (engl. *data binding*) koje pohranjuje ili na kreiranje korisničkog sučelja. [prema 68]

Ograničenje priručne memorije omogućuje pohranu odgovora na često postavljene zahtjeve klijenata. Komponenta poslužitelj zadužena je da prilikom kreiranja odgovora implicitno ili eksplicitno dozvoljava ili zabranjuje uporabu priručne memorije za pohranu generiranog odgovora. Ukoliko komponenta poslužitelj implicitno omogući pohranu odgovora na postavljani zahtjev u priručnu memoriju tada će se pohranjeni odgovor koristiti prilikom sljedećeg, identično postavljenog zahtjeva klijenta. Priručna memorija može se nalaziti unutar komponente poslužitelja, posrednika ili klijenta. [prema 68]

Ograničenje jedinstvenog sučelja omogućava neovisnost između razvoja sučelja te apstraktnog sloja za komunikaciju s klijentom. Međutim, ovakav pristup ima i svojih nedostataka. Prije svega, zbog ovisnosti o standardiziranim metodama i tipovima podataka koji su nametnuti HTTP-a nije moguće izvršiti jednostavne prilagodbe sučelja. Prednost takvog jedinstvenog sučelja očituje se u jednostavnoj arhitekturi sustava. [68]

Ograničenje neovisnog zahtjeva zabranjuje korištenje sjednice prilikom komunikacije između komponente klijenta i komponente poslužitelja. Korištenje sjednica smanjuje broj klijentskih zahtjeva koje poslužitelj može istovremeno obrađivati. Neovisnošću zahtjeva stavlja se naglasak na postojanje svih potrebnih informacija unutar zahtjeva kako bi poslužitelj mogao obraditi zahtjev. Prednosti takvog sustava su poboljšanje vidljivosti kao i proširivost sustava.

Ograničenje slojevitog sustava omogućava neovisnost tijekom razvoja komponenti sustava uz uvjet zadržavanja postojećih sučelja za ostvarenje komunikacije. Ovakvim pristupom omogućeno je reduciranje složenosti sustava jer svaka od komponenata vidi samo ono sučelje komponente s kojom komunicira.

5.5.1.5. Implementacija REST Web servisa

REST predstavlja skup arhitekturnih principa koji nam daju mogućnost implementacije Web servisa koji su fokusirani na resurse sustava kao i načine prijenosa te adresiranja resursa dostupnih na poslužitelju korištenjem HTTP-a. U posljednjih nekoliko godina REST Web servisi prolaze kroz fazu životnog ciklusa u kojoj doživljavaju eksponencijalan rast popularnosti. Sam rast popularnosti ponukan je jednostavnošću njegove implementacije te primjene zbog čega dolazi do istiskivanja dizajna temeljenih na SOAP-u i WSDL-u. [prema 82] U svrhu standardizacije i konzistentnosti implementacije REST Web servisa definirana su četiri osnovna principa dizajna:

1. implementacija REST Web servisa koristi **isključivo** HTTP metode
2. komunikacija se odvija bez pamćenja prethodnog stanja
3. URI se primjenjuje za identifikaciju resursa
4. reprezentacije resursa mora biti u JSON i/ili XML formatu

Jedna od ključnih karakteristika REST servisa je eksplicitna upotreba HTTP metoda na način koji je definiran RFC2616 standardom. Primjerice, HTTP GET metoda je definirana za kreiranje podataka koju će korisnička aplikacija koristiti za dohvaćanje željenog resursa, dohvaćanje podataka s Web poslužitelja ili izvršavanje upita s očekivanjem da će Web poslužitelj odgovoriti skupom odgovarajućih resursa. S druge pak strane, HTTP POST metoda je definirana za pohranjivanje podataka koju će korisnička aplikacija koristiti za perzistentnu pohranu podataka koji su kreirani na korisničkoj strani aplikacije. Prilikom implementacije REST Web servisa vrlo je važna uspostava jedan na jedan mapiranja između operacija kreiranja, pregledavanja, ažuriranja, brisanja - CRUD (engl. *Create*, *Read*, *Update*, *Delete*) te HTTP metoda. Mapiranje između operacija i HTTP metoda definirano je na sljedeći način:

- za kreiranje novog resursa (C) koristi se HTTP metoda POST
- za dohvaćanje reprezentacije resursa (R) koristi se HTTP metoda GET
- za promjenu stanja resursa ili ažuriranje resursa (U) koristi se HTTP metoda PUT
- za brisanje resursa (D) koristi se HTTP metoda DELETE

Dizajneri sustava koji nisu dovoljno upoznati s konceptima RFC2616 standarda te primjenom HTTP metode GET ponekad istu koristi u pogrešne svrhe, primjerice za kreiranje resursa. U takvim slučajevima se URI zahtjeva GET ne koristi RESTfuly. Jedan od glavnih problema spomenutog korištenja HTTP metode GET vezan je uz samu semantiku zahtjeva. Web poslužitelji su konfigurirani na način da, u slučaju HTTP metode GET, vraćaju resurs koji odgovara URI-u ili parametrima koji su zadani u okviru URI-a (engl. *Query Parameters*).

```
GET /adduser?name=Robert HTTP/1.1
```

Slika 30. HTTP zahtjev za kreiranje novog resursa primjenom HTTP metode GET

REST Web servisi odlikuju se skalabilnošću kako bi zadovoljili vrlo visoke zahtjeve za performansama. Upravo se iz navedenog razloga koriste klasteri koji imaju mogućnost uravnoteženja opterećenja Web poslužitelja te preusmjeravanje zahtjeva na druge Web poslužitelje kako bi se smanjilo vrijeme odziva. *Load Balancer* djeluje kao reverzni *proxy* koji distribuira podatkovni promet, odnosno zahtjeve klijenata, između većeg broja poslužitelja, odnosno većeg broja instanci aplikacije. Pritom je važno da REST Web servis prosljeđuje cjelovite zahtjeve na posredničke Web poslužitelje kako bi se izbjeglo lokalno pamćenje stanja između zahtjeva. Cjeloviti zahtjev obuhvaća slanje tokena koji sadrži autentifikacijske i/ili autorizacijske podatke. Komponente bez pamćenja stanja omogućuju jednostavniji razvoj te distribuciju zahtjeva između poslužitelja s uravnoteženim opterećenjem.

Slika 31. Komunikacija bez pamćenja stanja

Sa stajališta klijentskih aplikacija, URI omogućuje adresiranje pojedinog resursa koji se nalazi na Web poslužitelju. U tu svrhu, URI struktura trebala bi biti jednostavna, predvidljiva te lako razumljiva. Jedan od načina postizanja visoke razine upotrebljivosti je definiranje URI-a koji nalikuju strukturi direktorija. Prema ovoj definiciji, URI nije samo niz razgraničen kosom crtom, već stablo s podređenim i nadređenim granama koje su međusobno povezane čvorovima. Primjerice, URI visoke razine upotrebljivosti mogao bi biti definiran kao

```
http://www.myservice.org/ discussion/topics/{topic}
```

Reprezentacija resursa predstavlja stanje u kojem se zahtijevani resurs nalazi u trenutku kada ga klijentska aplikacija pokušava dohvatiti. Važno je da se omogući preslikavanje reprezentacije resursa u format koji je prikladan za transfer podataka. Postavljanje standarada za komunikaciju je vrlo važno jer omogućuje nesmetanu komunikaciju između klijenata i poslužitelja ili pak između dvaju poslužitelja. JSON je lagani, tekstu orijentirani format zapisa podataka. Prema W3C [10], JSON je neovisan o programskom jeziku, samoopisujući te vrlo jednostavan za razumijevanje. XML je dizajniran kako bi bio čitljiv i računalu i čovjeku. XML se koristi za pohranu te transport podataka, a razlika u odnosu na HTML je u tome što se HTML koristi za dizajn Web stranica, a XML za pohranu i transport podataka.

5.5.2. GraphQL Web servisi

REST(ful) Web servisi godinama unazad pružaju osnovnu podršku prilikom kreiranja jednostavnih struktura upita. Međutim, REST(ful) Web servisi ne pružaju preciznu kontrolu nad podacima koja bi razvojnim programerima omogućila fleksibilnost te dohvaćanje svih potrebnih podataka i informacija bez generiranja velike količine nepotrebnih poziva. Facebook je 2015. godine razvio GraphQL upitni jezik, kao odgovor na probleme s kojima se REST(ful) arhitektura susreće. GraphQL je upitni jezik za kreiranje API-a koji razvojnim programerima omogućava odabir vrste zahtjeva te definiranje željenih svojstava odabranog entiteta kao i primanje potrebnih informacija u **jednom** odgovoru. [83]

Termin GraphQL akronim je za *Graph Query Language*, ali za razliku od drugih upitnih jezika, kao što je SQL (engl. *Structured Query Language*), to nije jezik za izravnu komunikaciju s bazom podataka, već deklarativni jezik koji definira strukturu te pravila putem kojih klijenti komuniciraju s API poslužiteljem. GraphQL je jezik koji omogućava API poslužitelju da putem sučelja izloži dostupne resurse, dok je GraphQL specifikacija otvoreni standard koji opisuje pravila i karakteristike jezika te pruža upute za izvršavanje GraphQL upita. S obzirom da je GraphQL definiran otvorenim standardom, službena implementacija samog GraphQL-a nije specificirana i implementirana već se ona može napisati u bilo kojem programskom jeziku. Osim toga, GraphQL je moguće integrirati s bilo kojom bazom podataka, bilo SQL, bilo NoSQL. Različitost klijenata ne predstavlja prepreku prilikom implementacije te primjene GraphQL-a tako da sama implementacija Web servisa može podržavati bilo koju vrstu klijenta, od mobilnih aplikacija pa sve do Web aplikacija, pri čemu je vrlo važno napomenuti da implementacija samog Web servisa mora strogo slijediti pravila navedena u GraphQL specifikaciji. [84]

Priča o nastanku i razvoju GraphQL upitnog jezika datira u 2012. godinu i to u vrijeme kada se Facebook suočio s neefikasnošću rada njihove aplikacije na mobilnim uređajima. Glavni problem bio je vezan uz kompleksnu implementaciju *News Feed-a* te međusobnu povezanost, ispripletenost te rekurzivnost objava koje se pojavljuju na samom *News Feed-u*. Kako je Facebook prvotno razvijan kao Web aplikacija, razvojni programeri iskoristili su postojeći REST API prilikom izgradnje mobilne aplikacije. Međutim, skup podataka koji prikazuje pojedina objava na aplikaciji za mobilne uređaje daleko je manji od onog koji prikazuje pojedina objava unutar Web aplikacije. Razvojni inženjeri odlučili su da je potreban reinženjering *News Feed* API-a kako bi sama mobilna aplikacija zadovoljila zahtijevane potrebe za efikasnošću i brzinom rada. U kolovozu iste godine objavljena je prva verzija Facebook mobilne aplikacije koja se temeljila na upotrebi GraphQL upitnog jezika koji je razvojnim programerima omogućio značajna poboljšanja u pogledu karakteristika performansi prilikom dohvaćanja podataka. Tijekom sljedeće dvije godine nastavilo se s razvojem GraphQL-a, a prva verzija GraphQL specifikacije javno je objavljena krajem 2015. godine.

5.5.2.1. Karakteristike GraphQL-a

GraphQL je definiran na temelju nekoliko karakteristika dizajna. Prije svega, GraphQL upiti su deklarativni, hijerarhijski te rekurzivni, dok je sama GraphQL shema strogo tipizirana te introspektivna. Navedene karakteristike omogućavaju rješavanje problema s kojima se susreće REST arhitektura – *under-fetching* i *over-fetching* podataka pojedinog entiteta.

Deklarativnost GraphQL upita znači da klijenti unutar zahtjeva deklariraju samo ona svojstva entiteta koja ga trenutno zanimaju, a sam odgovor će uključivati samo ta svojstva.

```
{
    order(id: '1') {
        buyer
        total
    }
}
```

```
{
    'data': {
        'order': {
            buyer: 'John Doe',
            total: 2999.00
        }
    }
}
```

Hijerarhičnost GraphQL upita znači da vraćeni podaci slijede strukturu kreiranog upita. Drugim riječima, podaci su prikazani u formatu strukture stabla, odnosno, svaki složeni entitet koji je dio osnovnog upita prikazan je u odgovarajućoj strukturi korespondentnog upita. [83]

```
{
    order(id: '1') {
        buyer
        products {
            name
            quantity
        }
        total
    }
}
```

```
{
    'data': {
        'order': {
            buyer: 'John Doe',
            products: [
                {
                    name: 'Cowboy 4',
                    quantity: 1
                }
            ]
            total: 2999.00
        }
    }
}
```

Stroga tipizacija GraphQL upita definirana je ključnom riječju *type* . Tip vrijednosti definira mogući skup vrijednosti pojedinog svojstva entiteta. Neki od GraphQL jednostavnih tipova su skalari, odnosno skalarne, primitivne vrijednosti, nizovi, logičke vrijednosti, dok u složene vrijednosti spadaju objekti. [83] Pritom je važno napomenuti da se *non-nullable* svojstva označavaju simbolom **!** koji dolazi na kraju definicije tipa pojedinog svojstva.

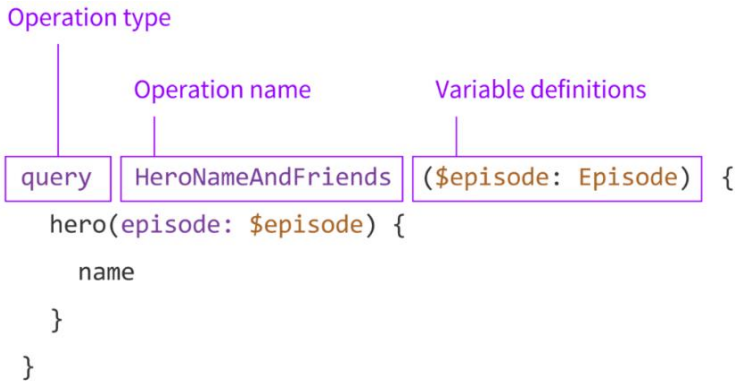
```
type Order {
    id: ID!
    products: [Product!]!
    total: Float!
}
```

5.5.2.2. Upiti i mutacije

Koncept rada REST API-a te GraphQL API-a može se usporediti s radom tradicionalnih prodajnih automata (engl. *vending machine*). Primjerice, uz tradicionalni REST, svaka tipka na automatu povezana je samo s **jednim** proizvodom Dakle, kupnja *n* različitih proizvoda iziskuje pritisak na *n* različitih tipaka kako bi dobili sve proizvode. Jasno je da je ovakav proces vrlo spor te da iziskuje puno praznog hoda. Ukoliko bi postojala tipka posebne namjene putem koje postoji mogućnost dobivanja više proizvoda odjednom, tada bi primjerice, pritiskom na tipku posebne namjene bilo moguće dobiti *n* proizvoda iz automata. Kombinacija navedenih koncepata dovodi do mehanizma automata na kojem je moguće pritisnuti točno one tipke koje želimo kako bi njihovom kombinacijom dobili sve što je potrebno u samo **jednom** zahtjevu. Upravo se na tome temelji koncept GraphQL API-a.

GraphQL je deklarativni upitni jezik koji se temelji na jedinstvenoj krajnjoj točci na koju se, putem POST metode u tijelu HTTP zahtjeva, šalje GraphQL operacija koju je potrebno izvršiti zajedno sa svim dodatnim informacijama i podacima. Jedinstvena krajnja točka nalazi se na putanji /graphql. Zahtjevi upućeni GraphQL poslužitelju nazivaju se dokumentima, a sastoje se od triju primarnih vrsta operacija – `query`, `mutation` te `subscription`. Operacija koja se koristi za dohvaćanje podataka naziva se upit te je ekvivalentna krajnjim točkama REST API-a koje obrađuju pristigle GET zahtjeve. Operacija koja se koristi za manipulaciju nad podacima naziva se mutacija te je ekvivalentna krajnjim točkama REST API-a koje obrađuju POST, PUT ili DELETE zahtjeve.

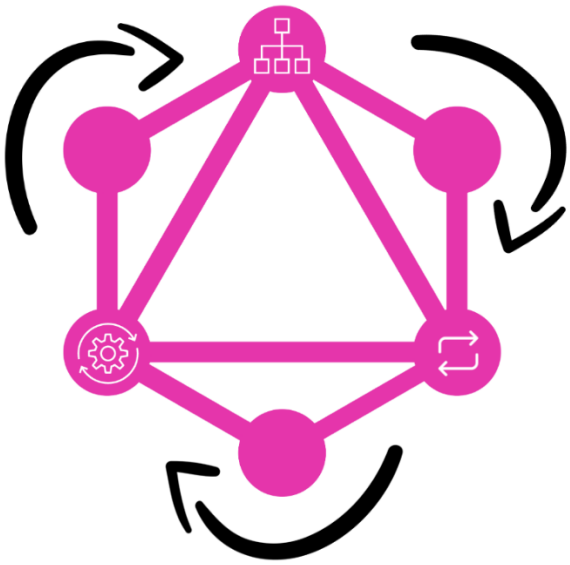
Kod operacije tipa `query` sam naziv operacije je proizvoljan te se u pravilu definira sukladno nazivu entiteta kojem se želi pristupiti. Unutar zagrada opcionalno se definiraju varijable upita, odnosno parametri kreiranog zahtjeva. U idućem redu nalazi se naziv upita koji se želi izvršiti, a koji obavezno mora biti implementiran na API-u te su potom definirana polja (eng. *fields*) koja moraju biti sadržana u odgovoru. Mutacije slijede istu strukturu kao i upiti uz bitnu razliku da se kao tip operacije koristi `mutation`. Sam naziv operacije je također proizvoljan, ali se obično radi o glagolu koji opisuje akciju koja se želi izvršiti. [85]



Slika 32. Prikaz generalne strukture GraphQL operacije

5.5.2.3. Izvršavanje upita i mutacija

Upiti i mutacije prolaze kroz tri faze tijekom životnog ciklusa obrade i izvršavanja – *parsiranje*, *validacija* te *izvršavanje*.



Slika 33. Životni ciklus izvršavanja GraphQL upita ili mutacije

Tijekom procesa parsiranja upit se raščlanjuje u AST (engl. *Abstract Syntax Tree*), odnosno stablo apstraktne sintakse. Stablo apstraktne sintakse je vrlo moćan alat, kako u matematici tako i u računalstvu, pa ga je moguće susresti kod alata kao što su ESLint, Babel i slični. U nekom od prethodnih poglavlja naveli smo kako je GraphQL, kao upitni jezik, strogo tipiziran. Za svaki od entiteta baze podataka kreira se korespondentna *schema* koja zatim predstavlja njegovu reprezentaciju. [prema 86]

Tijekom procesa validacije kreirano stablo apstraktne sintakse, iz prethodne faze životnog ciklusa, provjerava se sukladno kreiranoj *schemi* zahtijevanog entiteta. Proces validacije upita uključuje sintaktičku provjeru samog upita te provjeru koja se odnosi na ispravnost te postojanje definiranih polja. Ukoliko se tijekom procesa validacije utvrde određene nepodudarnosti i problemi cijeli proces izvršavanja kreiranog upita prestaje te korisnik dobiva odgovarajuću poruku pogreške. [prema 86]

Ukoliko tijekom faze validacije nisu pronađene sintaktičke nepravilnosti i/ili semantičke nepodudarnosti tada se prelazi na fazu izvršavanja kreiranog upita. Tijekom procesa izvršavanja upita prolazi se kroz kreirano stablo apstraktne sintakse, počevši od korijena stabla, te se za svaki pojedini čvor stabla poziva korespondentni razrješivač (engl. *resolver*), a rezultati se agregiraju te prezentiraju u JSON formatu.

5.6. API (engl. *Application Programming Interface*)

Aplikacijsko programsko sučelje, poznatije kao API, je skup pravila koja omogućavaju prijenos podataka i informacija između različitih klijentskih i serverskih aplikacija. [63] Primarna prednost te odlika API-a je mogućnost ponovnog korištenja postojećih funkcija. API omogućava programeru izbjegavanje nepotrebnog rada tako što nudi skup funkcija koje su već implementirane te javno dostupne. Programeri tako imaju mogućnost inkorporacije postojećih eksternih funkcija, koje nudi zahtijevani API, u vlastite aplikacije čime dolazi do smanjenja nepotrebnog koda, a time posljedično i primjene koncepata ''čistog koda'' (engl. *Clean code*). [prema 63]

Poziv API-a, odnosno API zahtjev, se definira kao poruka usmjerena na API s ciljem okidanja neke od specifičnih funkcionalnosti koje pruža. Svaki API zahtjev šalje se prema API krajnjoj točci (engl. *API endpoint*) koja predstavlja jedan od dvaju dijelova komunikacijskog kanala. [prema 63] Ujedno, API krajnja točka predstavlja mjesto s kojeg potječe kreirani API odgovor. Primjerice, *Fetch API* koristi se za kreiranje AJAX zahtjeva unutar programskog jezika JavaScript. S druge pak strane, *Stream API* koristi se za vrlo jednostavno procesiranje kolekcija i objekata unutar programskog jezika Java. Svaki od ovih API-a ima definirane funkcije koje sadrže implementacije često korištenih programskih blokova. Primjerice, *Fetch API* pruža vrlo jednostavno kreiranje HTTP zahtjeva uz pomoć *fetch* funkcije. Na taj se način programer oslobađa od pisanja *boilerplate* koda.

Velik broj autora, prilikom definiranja pojma API, služi se primjerom procesa naručivanja jela u restoranu. U tom kontekstu, API predstavlja jelovnik u restoranu. Jelovnik sadrži popis jela koje možemo naručiti zajedno s njihovim opisima. Popis jela predstavlja skup funkcija, zajedno sa svojim definicijama, odnosno brojem i tipom argumenata koje svaka od njih prima, koje API stavlja na raspolaganje programerima. Poziv svake od API funkcija stavlja se u korelaciju s kreiranjem narudžbe za odabrano jelo. Naime, nakon poziva API funkcije programer ne zna što se točno događa u pozadini, ali zna što će dobiti kao rezultat, odnosno kao API odgovor na kreirani API poziv. U kontekstu naručivanja jela u restoranu, korisnik po završetku procesa narudžbe ne zna što se točno događa u kuhinji te ima vrlo malo informacija o načinu na koji kuhari pripremaju odabrano jelo. I u kontekstu API-a i u kontekstu naručivanja jela u restoranu, krajnjeg korisnika ne zanimaju detalji procesa već samo konačan plod kreiranog zahtjeva. [prema 64]

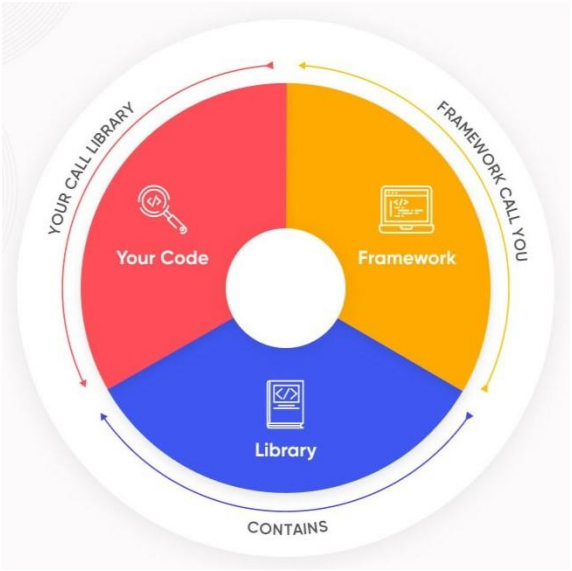
API se vrlo često poistovjećuje s pojmom Web servisa. Za razliku od Web servisa, API ne mora biti dostupan putem Web-a već se može koristiti kao samostojeća biblioteka koja pruža odgovarajuće funkcionalnosti. Sukladno tome, svaki Web servis je ujedno i API, međutim obrat ovog teorema ne vrijedi jer API ne mora nužno biti dostupan putem Web-a.

5.7. Programski okvir (engl. *software framework*)

Programski okvir je skup programskih alata na temelju kojih se grade dobro strukturirani te pouzdani programski sustavi. Službena, rječnička definicija programskog okvira koja se najčešće susreće u literaturi glasi:

„Programski okvir je alat koji pruža gotove komponente ili rješenja koja se prilagođavaju, a sve u cilju bržeg i jednostavnijeg razvoja složenih aplikativnih sustava.“ [87]

Programski okvir može uključivati niz drugih biblioteka koje mu olakšavaju sam rad. Prilikom izgradnje programskog okvira važno se pridržavati IoC (engl. *Inversion of Control*) načela, odnosno načela inverzije kontrole. Primjenom IoC načela, programski okvir poziva prilagođene dijelove, odnosno komponente sustava, kada je to potrebno. [prema 87] Sam razvoj programskih okvira značajno je utjecao na područje razvoja programskih sustava posebice iz perspektive brzine razvoja te ponovne iskoristivosti komponenata.



Slika 34. Koncept rada programskog okvira [87]

Programski okvir može uključivati druge biblioteke programskog koda, koristiti potpornu programsku podršku, setove programskih alata ili vanjske API-e koji nude potrebne informacije i podatke. Pojedine komponente programskog okvira moguće je prilagoditi vlastitim potrebama. Svrha programskog okvira je pripomoći razvoju aplikativnih sustava pružajući standardnu funkcionalnost niske razine kako bi se razvojni inženjeri mogli usredotočiti na elemente koji pojedini projekt čine jedinstvenim. U konačnici, primjena programskog okvira reducira troškove razvoja sustava.

Slika 35. Sastavni dijelovi programskog okvira [87]

5.7.1. React

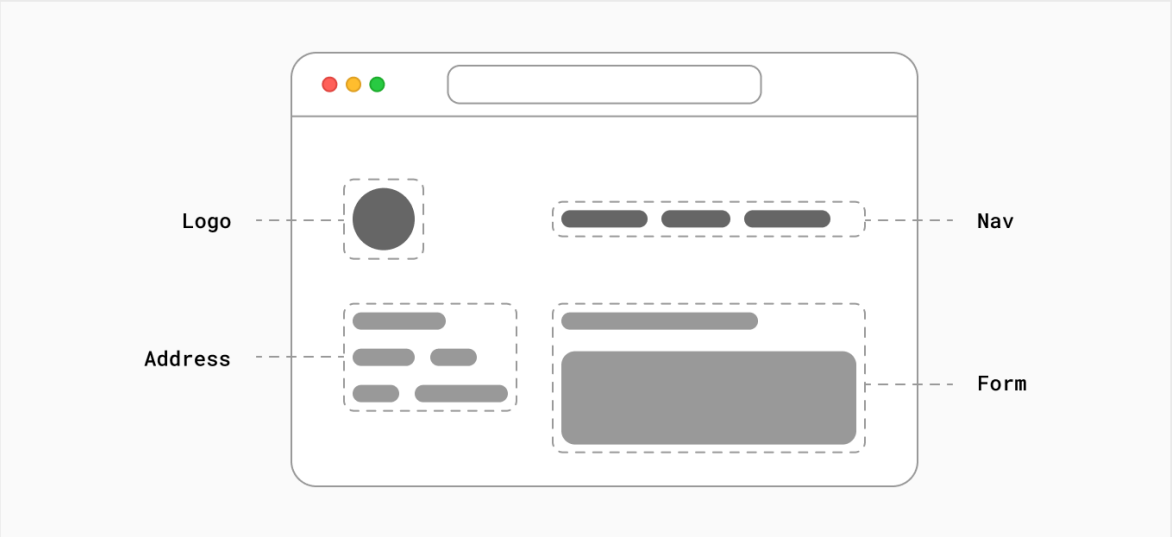
React, također poznat kao React.js te ReactJS, je deklarativna i fleksibilna JavaScript biblioteka otvorenog koda koja se koristi za izgradnju modularnog korisničkog sučelja sastavljenog od komponenata. [71] Komponente omogućuju izradu samostalnih dijelova programskog koda koji promiču koncept višestruke i ponovne iskoristivosti. Svaka komponenta može se sastojati od neograničenog broja drugih komponenata koje predstavljaju djecu osnovne komponente roditelja. Deklarativnost programskog okvira omogućava programerima kreiranje opisa koji definira što aplikacija mora raditi umjesto da se definiraju koraci kako će se aplikacija izvršavati te samim time ponašati. React je razvijen davne 2013.-te godine od strane Facebook-a, a svoju veliku popularnost stekao je 2016.-te godine. React omogućava izgradnju SPA (engl. *Single Page Applications*) te PWA (engl. *Progressive Web Application*), odnosno Web aplikacija koje ne zahtijevaju ponovno učitavanje stranice u slučaju promjene sadržaja već se promjene sadržaja obavljaju dinamički. React omogućava vrlo jednostavno zaključivanje o trenutnom stanju korisničkog sučelja kroz dekomponentizaciju korisničkog sučelja u zajedničku zbirku komponenata. Zahvaljujući svojim karakteristikama, React omogućava programerima razvoj Web aplikacija koje dinamički mijenjaju svoj sadržaj bez potrebe za osvježavanjem same stranice. Posljednje spomenuta karakteristika predstavlja srž React-a te omogućava razvoj Web aplikacija koje će se isticati brzinom, jednostavnošću te skalabilnošću. [prema 71]

Komponente korisničkog sučelja definirane su u dokumentima s ekstenzijom *.jsx* ukoliko se koristi programski jezik JavaScript, odnosno *.tsx* ukoliko se koristi programski jezik TypeScript. JavaScript XML, ili kraće JSX, omogućava programerima da na vrlo jednostavan i brz način podešavaju DOM[10] (engl. *Document Object Model*) pomoću osnovnog HTML koda. Dinamična promjena sadržaja potpomognuta je primjenom JSX-a koji omogućava manipulaciju DOM-a. Rukovanje promjenama unutar DOM-a ne predstavlja problem ukoliko razvijamo jednostavne Web aplikacije ili statičke Web stranice. Međutim, kod dinamičnih Web aplikacija koje iniciraju neprestanu interakciju s krajnjim korisnicima rukovanje DOM-a primjenom Vanilla JavaScript-a može izazvati velike probleme. JSX omogućava manipulaciju DOM-a kroz rukovanje virtualnim DOM-om koji je kreiran od strane React-a. Virtualni DOM predstavlja kopiju originalnog DOM-a koju React koristi prilikom utvrđivanja dijelova stvarnog DOM-a koji zahtijevaju dinamično ažuriranje sadržaja. [prema 72] React je tijekom svog sazrijevanja prošao kroz mnoge verzija, a trenutno je u primjeni verzija 18.2. Svaka verzija nudi koncepte koji zamjenjuju ili nadograđuju postojeće funkcionalnosti programskog okvira.

[10] DOM – objektni model dokumenta je višeplatformsko programsko sučelje koje tretira HTML i XML u obliku strukture stabla. [73]

Prilikom izrade Web aplikacija, primjenom React programskog okvira, potrebno je instalirati te izvršiti *import* dviju vanjskih paketa: *React* te *ReactDOM*. Paketi se instaliraju primjenom *yarn* ili npm (engl. *Node Package Manager*) komandi koje se koriste unutar istoimenih alata za manipulaciju paketima. Razlog separacije biblioteka *React* i *ReactDOM* je u mogućnosti korištenja React-a prilikom razvoja drugih vrsta aplikacija. React omogućava razvoj mobilnih aplikacija kroz njegovu kombinaciju s React Native programskim okvirom. React Native u tom kontekstu sadrži funkcionalnosti koje su vezane uz platformu na kojoj će se aplikacija izvršavati.

React razlikuje dva tipa komponenata, odnosno klasne komponente (engl. *Class based components*) te funkcijske komponente (engl. *Functional based components*). Razlika između dviju vrsta komponenata je u sintaksi prilikom kreiranja i korištenja same komponente te upravljanju stanjem komponente kroz njen životni ciklus. U literaturi se vrlo često koristi termin *Stateless component* za funkcijske komponente, dok se za klasne komponente koristi dijametralno suprotni termin, *Stateful component*. Klasne komponente upravljaju njenim životnim ciklusom kroz Vanilla JavaScript funkcije, dok funkcijske komponente koriste *hooks* za upravljanje životnim ciklusom komponente. *React Hooks*, koji su uvedeni s verzijom 16.8, omogućuju upravljanje stanjem funkcijske komponente kroz njen životni ciklus bez potrebe za kreiranjem odvojene klasne komponente. [prema 71] Iako klasne komponente predstavljaju korijen primjene React-a kao programskog okvira, u današnje vrijeme se u većini slučajeva koriste funkcijske komponente upravo zbog vrlo jednostavne sintakse te sukladnosti sa svim konceptima koje pružaju i klasne komponente. React dozvoljava upotrebu ES6 sintakse čime se dodatno pojednostavljuje te ubrzava razvoj Web aplikacija.

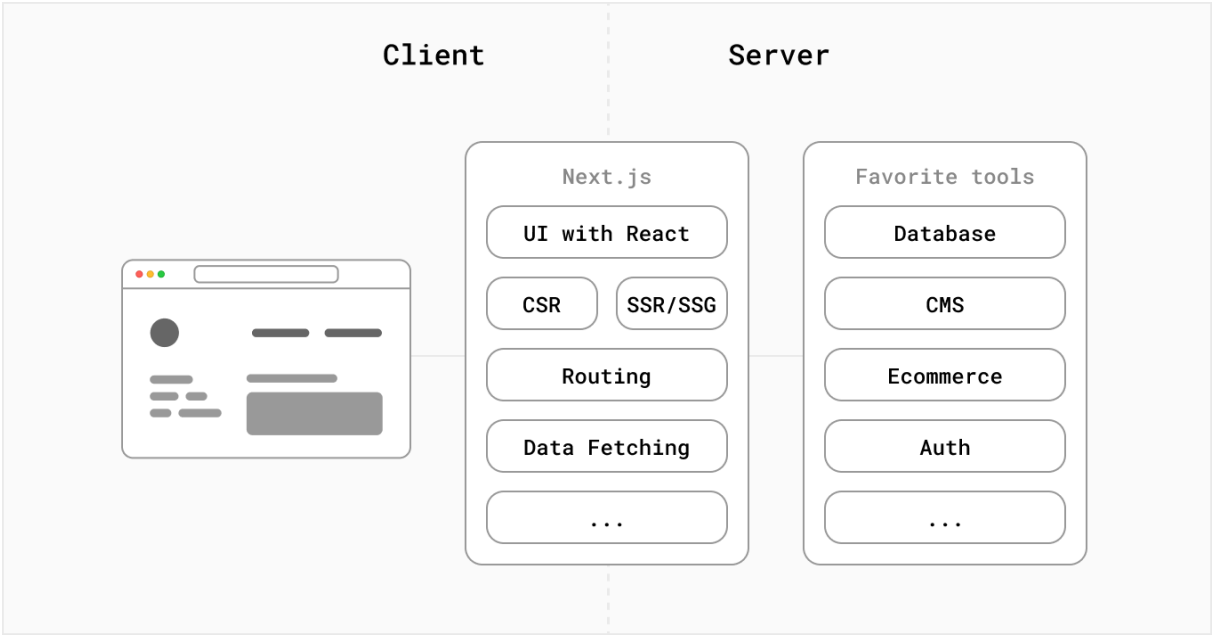


Slika 36. Primjer raščlambe aplikacije na komponente [74]

5.7.2. Next.js

Next.js je programski okvir kojeg je 2016. godine kreirao Zeit, a koji se zasniva na React-u, Babel-u te Webpack-u. Next.js omogućava jednostavno i intuitivno strukturiranje projekta te navigiranje kroz sam projekt jer eliminira potrebu za konfiguriranjem bazičnih dijelova projekta. Primjerice, putanja do pojedine stranice ekvivalentna je strukturi direktorija datotečnog sustava čime se omogućava vrlo jednostavno kreiranje nove stranice. Na taj se način eliminira potreba za konfiguriranjem React Router-a koji će prikazivati ispravne stranice u ovisnosti o trenutnoj ruti, odnosno URL-u.

Next.js je fleksibilan programski okvir temeljen na React-u razvijen s ciljem potpore rapidnom razvoju aplikativnih sustava. U suštini, Next.js, kao i React, se temelji na raščlambi aplikacije na manje dijelove, odnosno komponente čijim se sastavljanjem grade određene funkcionalnosti aplikacije. Next.js u pozadini koristi React programski okvir kojeg proširuje *built-in* funkcionalnostima i alatima. Next.js pokušava riješiti probleme s kojima se susreće React. Prije svega, React pojedinu stranicu renderira u potpunosti na strani klijenta tako da je teško provesti SEO (engl. *Search Engine Optimization*). Next.js u potpunosti rješava taj problem na način da omogućuje renderiranje na strani poslužitelja čime se klijentu isporučuje pripremljen HTML kod bez potrebe za dodatnim popunjavanjem dinamičkog sadržaja unutar same stranice. Neke od bitnijih karakteristika samog okvira su *optimizacija slika*, *minimalna konfiguracija*, *internacionalizacija*, *integrirani routing baziran na strukturi datotečnog sustava*, *inkrementalna statička regeneracija (eng. Incremental Static Regeneration)* te *podrška za Hot Reload i TypeScript.* [88]

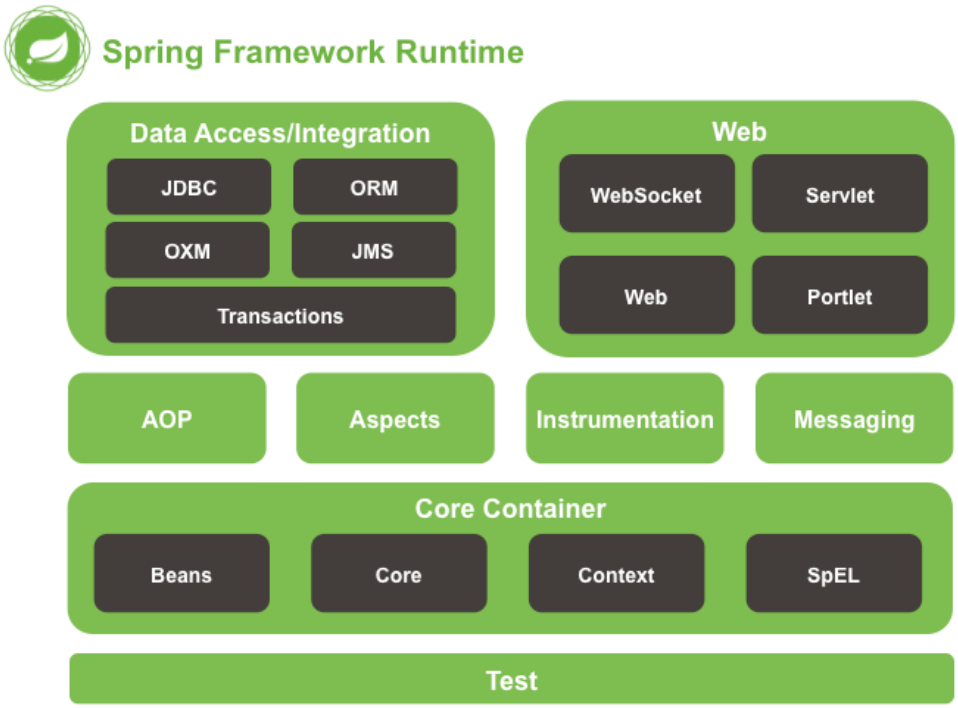


Slika 37. Komponente Next.js programskog okvira

5.7.3. Spring i Spring Boot

Spring Framework, Java razvojno okruženje otvorenog koda kojeg odlikuje jednostavnost, fleksibilnost, modularnost te *backward* kompatibilnost, razvijen je 2002. godine od strane Roda Johnsona i javio se kao odgovor na kompleksan razvoj *servlet* aplikacija te dotad korišteni JEE (engl. *Java Enterprise Edition*) standard za izradu robusnih Web aplikacija. Spring je pridobio naklonost razvojnih inženjera trima ključnim karakteristikama kojima se izdvojio u odnosu na JEE – *inverzija kontrole* koja omogućava da sam programski okvir kontrolira izvršavanje programskog koda, *injektiranje ovisnosti* (engl. *Dependency Injection*) o kojima ovise pojedini dijelovi programskog koda te podrška *aspektno orijentiranom programiranju* (engl. *Aspect Oriented Programming*). [prema 90]

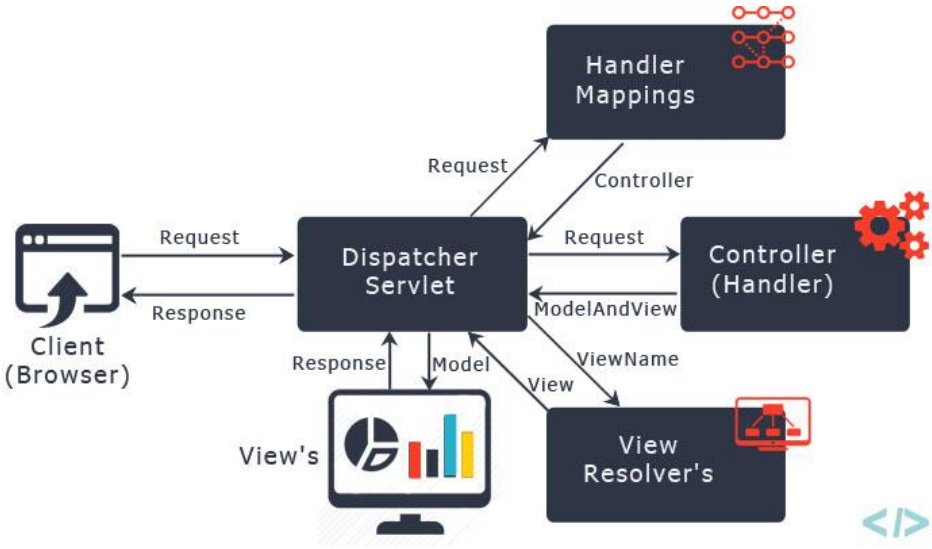
Nastanak Spring-a, osim što je olakšao kreiranje robusnih Java aplikacija, i dalje je zahtijevao veliku količinu konfiguracijskih klasa prilikom inicijalizacije projekata. Osnovu Spring programskog okvira čini *Spring container* koji vodi brigu o pojedinim komponentama aplikacije tijekom njezinog životnog ciklusa te omogućava koncept inverzije kontrole. Sam programski okvir sastoji se od velikog broja različitih modula među kojima valja istaknuti *spring-security-core*, *spring-security-web*, *spring-jdbc*, *spring-core* te *spring-web*. Spring programski okvir kontinuirano se nadograđuje i usavršava kako bi pratio nove tehnološke trendove te zadovoljio potrebe zahtjevnog tehnološkog tržišta koje svakodnevno raste i napreduje.



Slika 38. Pregled dijelova Spring programskog okvira [91]

Spring Boot jedan je od modula koji je nastao kako bi projekt bio inicijaliziran i spreman za daljnji razvoj uz čim manje truda i nepotrebnih repetitivnih konfiguriranja. Spring Boot nudi novu paradigmu razvoja Spring aplikacija koja omogućava agilniji pristup razvoju te fokusiranje razvojnih inženjera na razvoj konkretne programske logike. Inicijalizaciju Spring Boot projekta moguće je izvršiti putem službenog te javno dostupnog Spring Initializer alata koji generira osnovnu strukturu Spring Boot projekta. Koncept rada Spring Boot programskog okvira temelji se na anotacijama te dvjema ključnim datotekama – `pom.xml` kao konfiguracijska datoteka *maven* projekata te klasa anotirana sa `@SpringBootApplication` koja sadrži *main* metodu koja predstavlja ulaznu točku projekta. Anotacija `@SpringBootApplication` zamjenjuje ranije korištene `@Configuration`, `@ComponentScan` i `@EnableAutoConfiguration` anotacije. [91]

Spring MVC (engl. *Model-View-Controller*) dio je radnog okvira koji je namijenjen izradi Web aplikacija. Putem MVC modela, Spring je odvojio poslovnu logiku od logike prikaza i navigacije. Obrada zahtjeva provodi se kroz nekoliko faza, a *Dispatcher Servlet* ima početnu i završnu riječ. Proces započinje slanje zahtjeva od strane klijenta prema *DispatcherServlet-u* koji u nastavku inicira komunikaciju s *Handler Mapping* komponentama kako bi saznao koja klasa kontrolera, odnosno klasa anotirana sa `@RestController`, je zadužena za obradu zahtjeva. *DispatcherServlet* zatim prosljeđuje zahtjev odgovarajućoj kontroler klasi. *Controller* obrađuje zahtjev i vraća, odnosno podatke i informacije, u HTML ili JSON formatu. *DispatcherServlet* uz pomoć *ViewResolver-a* mapira logički naziv stranice u specifičnu implementaciju koja predstavlja konkretnu stranicu unutar aplikacije. Nakon toga se dovršava izgled stranice koja se šalje klijentu pomoću implementacije *View* komponente čime i proces obrade zahtjeva završava. [prema 91]



Slika 39. Proces obrade zahtjeva kod Spring MVC arhitekture

Razvoj REST API-a temelji se na primjeni nekoliko anotacija među kojima su ključne – `@Component`, `@Service`, `@Repository`, `@RestController`, `@GetMapping`, `@PostMapping`, `@PutMapping`, `@DeleteMapping`, `@PathVariable`, `@RequestParam` te `@RequestBody` koje bitno reduciraju vrijeme potrebno za izradu krajnjih točaka te njezinih pomoćnih servisa. Anotacija `@RestController` je centralna komponenta koja omogućava kreiranje API-a. [prema 91]

Slika 40. Spring Boot i REST API [92]

Prilikom prihvaćanja zahtjeva, na odgovarajućoj krajnjoj točci REST API-a, dolazi do deserijalizacije poruke JSON formata u korespondentni POJO (engl. *Plain old Java object*). Povratni tip metode klase anotirane sa `@RestController` može biti domenski objekt koji će sam programski okvir pretvoriti u JSON format ili objekt klase `ResponseEntity` pri čemu `ResponseEntity` predstavlja cijeli HTTP odgovor (uključujući statusni kod, zaglavlja i tijelo HTTP odgovora,…).

```
@RestController
@RequestMapping("/api/orders")
@RequiredArgsConstructor
public class OrderController {

    private final OrderService orderService;

    @GetMapping
    public ResponseEntity<OrderPage> fetchAllOrdersForUser(
        @RequestParam(defaultValue = "1") Integer page,
        @RequestParam(defaultValue = "9999") Integer size,
        @RequestParam Long idUser
        ) {
        return new ResponseEntity<>(orderService.getAllOrdersForUser(
            page, size, idUser), HttpStatus.OK);
    }

    @PostMapping
    public ResponseEntity<?> saveOrder(@RequestBody OrderForm form) {
        orderService.createOrder(form);

        return new ResponseEntity<>(HttpStatus.CREATED);
    }
}
```

5.7.4. Hibernate

Hibernate je Java programski okvir koji pojednostavljuje komunikaciju između aplikacije i baze podataka, odnosno ORM (engl. *Object Relational Mapping*) alat otvorenog koda čija implementacija slijedi JPA (engl. *Java Persistence API*) specifikacije s ciljem osiguranja perzistencije podataka. [prema 93] Njegova je glavna namjena omogućavanje mapiranja POJO-a na pojedine retke baze podataka čime se izbjegava ručno popunjavanje objekata prilikom dohvata iz baze te ručno popunjavanje upita prilikom perzistencije podataka u samu bazu podataka.

Klasa čiji se podaci spremaju u bazu podataka naziva se klasom entiteta. Svaka klasa entiteta mora imati konstruktor bez parametara. Modifikator pristupa konstruktora može biti `public`, `protected`, `package protected`, ali nikako `private`. Osim konstruktora bez parametara, za svaku je klasu entiteta važno definirati njezin jedinstveni identifikator. Anotacijom `@Id` označava se atribut tablice, odnosno svojstvo entiteta, koje predstavlja njegov jednoznačni identifikator. Uz pomoć anotacije `@Table` definira se tablica na koju se klasa entiteta veže, dok `@Entity` služi kao jedan od metapodataka prilikom učitavanja klase. Anotacija `@Column` služi za definiranje naziva atributa na kojeg se veže pojedino svojstvo objekta, pri čemu je važno napomenuti kako anotacija nije obavezna ukoliko naziv svojstva entiteta odgovara atributu tablice na koju se klasa entiteta, odnosno konkretno svojstvo entiteta, veže. Veze između pojedinih entiteta definiraju se uz pomoć anotacija `@JoinColumn`, `@OneToOne`, `@OneToMany`, `@ManyToOne` te `@ManyToMany`. [prema 93]

```
 1  @Setter
 2  @Getter
 3  @Entity
 4  @Table(name = "order", schema = "public")
 5  public class Order extends BaseAuditEntity {
 6
 7      @Id
 8      @GeneratedValue(strategy = GenerationType.IDENTITY)
 9      @Column(name = "id_order")
10      private Long idOrder;
11
12      @ManyToOne
13      @JoinColumn(name = "id_user", referencedColumnName = "id_user")
14      private User user;
15
16      @OneToMany(mappedBy = "order")
17      private List<OrderProduct> products;
18  }
```

5.8. Komunikacija u realnom vremenu

Komunikacija u realnom vremenu, u kontekstu Web-a, predstavlja zaprimanje podataka na strani klijenta odmah po njihovu nastanku na strani poslužitelja. Klijent dobiva novokreirane podatke i informacije bez potrebe za eksplicitnim kreiranjem zahtjeva prema poslužitelju, odnosno bez potrebe za osvježavanjem stranice. Komunikacija u realnom vremenu na Web-u je dvosmjerna komunikacija između klijenta i poslužitelja koja može biti *half-duplex* ili *full-duplex*. Početci razvoja komunikacije u realnom vremenu vežu se uz, tada jedino dostupnu tehnologiju koja se mogla iskoristiti u tu svrhu, HTTP. Međutim, HTTP nije osmišljen kao protokol za dvosmjernu komunikaciju tako da su razvojni inženjeri krenuli s osmišljavanjem koncepata i tehnika kojima će zaobići navedeno ograničenje. Iako je ograničenje dvosmjerne komunikacije uspješno razriješeno nizom tehnika, pri čemu je veliku ulogu imala AJAX tehnologija, zbog arhitekturne strukture HTTP-a nije moguće ostvariti potpuno dvosmjernu komunikaciju. Kako bi se ograničenje dvosmjerne komunikacije kod HTTP-a zaobišlo, razvijen je WebSocket protokol. [prema 94]

Web aplikacija može biti u potpunosti temeljena na komunikaciji u realnom vremenu – primjerice aplikacije za razmjenu poruka, VoIP (engl. *Voice over IP*) aplikacije kao što su Skype, Viber ili WhatsApp i slično. Osim toga, komunikacija u realnom vremenu može biti korištena samo u sklopu nekog dijela aplikacija kako bi se pozitivno utjecalo na korisničko iskustvo – primjerice funkcionalnost generiranja dokumenta nakon čega slijedi njegovo digitalno potpisivanje koje se pokreće klikom na gumb koji se nalazi na istoj stranici ili pak zaprimanje obavijesti unutar aplikacije.

Komunikacija u realnom vremenu ubraja se u kategoriju *duplex*, odnosno dvosmjerne komunikacije zato što obje strane unutar komunikacijskog kanala, odnosno klijent i poslužitelj, mogu međusobno primati i slati poruke. Nadalje, dvosmjerna komunikacija se tipizira na polu dvosmjernu (engl. *half-duplex*) te potpuno dvosmjernu (engl. *full-duplex*) komunikaciju. Kod polu dvosmjerne komunikacije primatelj i pošiljatelj mogu primati i slati poruke, ali ne istovremeno. Drugim riječima, komunikacija između primatelja i pošiljatelja omogućena je samo u jednom smjeru u datom trenutku vremena tako da pojedina strana komunikacijskog kanala ne može istovremeno slati i primati poruke. [95] Primjer *half-duplex* komunikacije može se poistovjetiti s komunikacijom putem *Walkie-Talkie* uređaja kod koje pojedina strana može govoriti, odnosno slati poruke tek kada suprotna strana završi s govorom. Kod potpuno dvosmjerne komunikacije omogućava se istovremeno slanje i primanje poruka između primatelja i pošiljatelja putem dva paralelna komunikacijska kanala. Primjer *full-duplex* komunikacije može se poistovjetiti s komunikacijom uživo kod koje pojedina strana komunikacijskog kanala može istovremeno slati te primati poruke drugih sugovornika. [95]

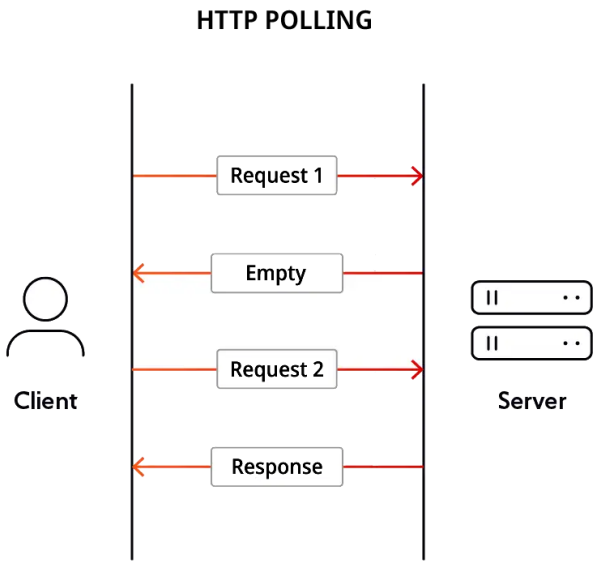
5.8.1."Komunikacija u realnom vremenu" korištenjem HTTP-a

Komunikacija između klijenta i poslužitelja na Web-u obavlja se posredstvom HTTP-a na način da klijentska strana aplikacije šalje HTTP zahtjev kojeg poslužitelj obrađuje te vraća korespondentni HTTP odgovor. Komunikacija kakvu promiče HTTP zahtijeva iniciranje korisničke akcije kako bi se kreirao zahtjev prema poslužitelju. Primjerice, ukoliko unutar aplikacije za razmjenu poruka korisnik inicira akciju slanja poruke tada je vrlo jednostavno, po završetku obrade zahtjeva, pokrenuti zahtjev za dohvaćanje poruka koje su namijenjene određenom korisniku kako bi se vidjele ažurirane poruke koje su mu uputili drugi korisnici. Međutim, ako korisnik ne pokrene nikakvu akciju koja bi pokrenula zahtjev prema poslužitelju tada poruke drugih korisnika nikada neće biti ažurne. [96] Tijekom godina razvijene su razne metode koje koriste HTTP kako bi oponašale komunikaciju u realnom vremenu – *standardni polling (engl. regular polling)*, *dugi polling (engl. long polling)* i *SSE (engl. Server-Sent Events)*.

5.8.1.1. Regular polling

Ukoliko ste ikada bili u šetnji sa djetetom onda ste se sigurno susreli s pitanjem: Jesmo li stigli? Vi ste vjerojatno ljubazno odgovorili, međutim, pitanje se nastavilo ponavljati svakih nekoliko sekundi. Roditelji širom svijeta prepoznat će se u ovoj kratkoj priči. [97]

Upravo je uvodna priča koncept na kojem se temelji standardni polling. Naime, polling je način "komunikacije u realnom vremenu" koristeći HTTP koji funkcionira na način da klijent periodički, odnosno prema unaprijed definiranim intervalima, šalje HTTP zahtjev kako bi preuzeo najnovije stanje. Poslužitelj zatim kreira odgovor s aktualnim stanjem podataka, a u većini slučajeva vraća prazan odgovor pošto ne raspolaže s novim podacima. Na taj je način moguće simulirati komunikaciju u realnom vremenu, međutim, vidljivo je da postoji određeni razmak između pojedinih zahtjeva što dovodi do problema kašnjenja. Naime, ovakav način komunikacije u realnom vremenu ne može biti primijenjen kod aplikacije koje se koriste za razmjenu poruka, primjerice Viber, WhatsApp, pošto bi navedena implementacija uzrokovala kašnjenje između slanja i primitka poruke. [prema 97]

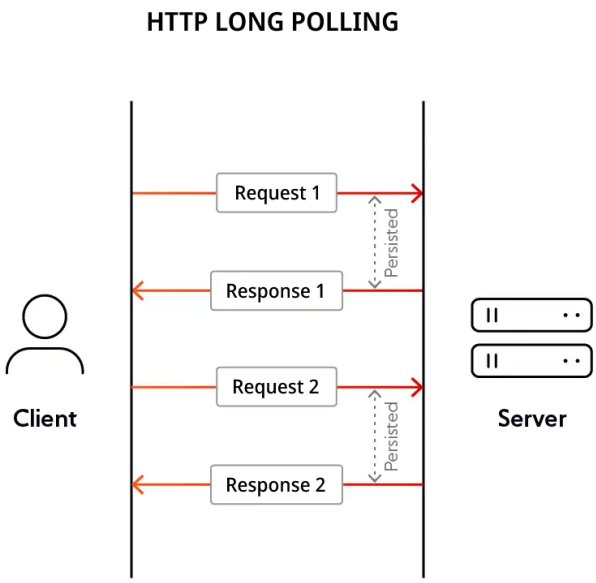


Slika 41. HTTP standardni polling [prema 98]

5.8.1.2. Long polling

Umjesto kratke šetnje zamislite dugu vožnju automobilom sa svojim djetetom. Dijete ponovno postavlja pitanje: Jesmo li stigli? Ovog puta, umjesto da ljubazno odgovorite, jednostavno šutite i ne odgovarate dok ne stignete na odredište (ili dok niste prisiljeni odgovoriti jer dijete sprema izljev bijesa). [97]

Upravo je uvodna priča koncept na kojem se temelji long polling. Long polling, umjesto konstantnog *pinganja* poslužitelja primjenjuje koncept spremanja kreiranog klijentskog zahtjeva. Zahtjev ostaje spremljen tako dugo dok se ne dogodi akcija koja će uzrokovati slanje odgovora klijentu ili dok sam zahtjev ne istekne. Istek zahtjeva obično se rješava implementacijom koja automatski generira novi zahtjev od strane klijenta. Na taj je način omogućeno da poslužitelj čuva zahtjev te na taj način omogući komunikaciju u realnom vremenu u trenutku pokretanja akcije koja uzrokuje slanje korespondentnog odgovora. Iako daleko moćniji od standardnog, long polling zahtjeva savršenu sinergiju između klijenta i poslužitelja kao i orkestraciju svakog pojedinog dijela unutar komunikacijskog kanala. S obzirom da veza ostaje otvorena tako dugo dok poslužitelj ne pošalje odgovor (ukoliko se radi o implementaciji koja automatizira slanje novog klijentskog zahtjeva nakon što trenutno aktualni zahtjev bude poništen uslijed *timeout-a*) moguće je da će veza ostati otvorena jedan poduži vremenski period. Iako je long polling metoda optimalnija od standardnog polling.a, u pogledu potrošnje mrežnih i procesorskih resursa, ona još uvijek ima neke nedostatke koji su neposredno uzrokovani arhitekturom samog HTTP-a. Problem koji je i dalje prisutan vezan je uz ponavljajuće slanje zahtjeva koji uzrokuju veliku potrošnju podatkovnog prometa, a posebno u uvjetima velikih, skalabilnih aplikacija. Međutim, za razliku od standardnog polling-a, long polling šalje podatke odmah čim su dostupni, odnosno bez ikakvih kašnjenja. [prema 97]



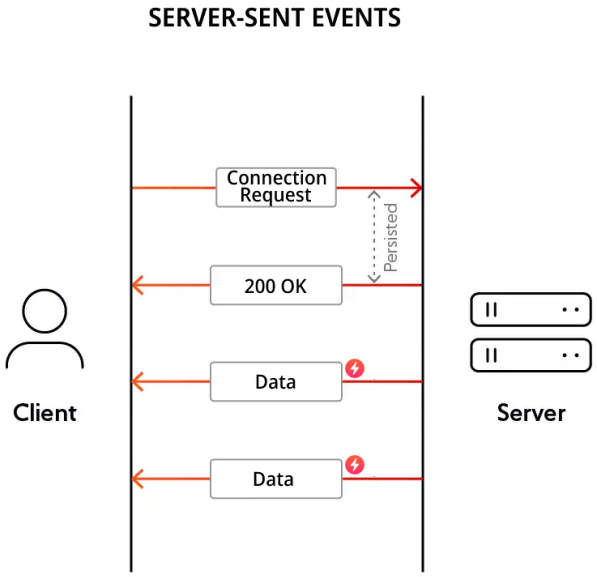
Slika 42. HTTP long polling [prema 98]

5.8.1.3. Server-Sent Events

Server-Sent Events je tehnologija koja omogućava automatsko ažuriranje stanja, odnosno slanje obavijesti, poruka i događaja od poslužitelja do klijen(a)ta putem HTTP veze. Sama tehnologija omogućava asinkronu komunikacija koja se temelji na protoku događaja između poslužitelja i klijen(a)ta. Radi se o standardiziranoj HTML5 tehnologiji koja omogućava isporuku podataka klijentima u realnom vremenu. Za razliku od *regular* te *long polling-a* koji kreiraju nove veze za svaki pojedinačni zahtjev klijenta, Server-Sent Events koristi samo jednu, uvijek istu, vezu prilikom razmjene podataka i informacija. [prema 99]

Server-Sent Events je standard koji opisuje način na koji poslužitelji mogu pokrenuti prijenos podataka prema klijentima nakon uspostave veze. Sam standard dopušta implementaciju XHR-a (engl. *XMLHttpRequest*) koji značajno štedi memoriju te time podiže efikasnost rada. Kao što i samo ime kaže, radi se o događajima koji se koriste za automatsko ažuriranje Web stranice koje je potaknuto određenom akcijom ili aktivnošću.

Samu komunikaciju inicira klijent kreiranjem HTTP zahtjeva za otvaranje veze. Klijent kreira novi JavaScript EventSource objekt putem kojeg prosljeđuje URL krajnje točke, odnosno klijenta. Klijent zatim čeka na odgovor poslužitelja. Poslužitelj ostavlja vezu otvorenom sve dok više nema događaja koji bi inicirali slanje HTTP odgovora. Veza se, osim po isteku predviđenog vremena trajanja, može zatvoriti slanje odgovarajućeg zahtjeva od strane klijenta. Održavanje veze otvorenom omogućeno je na način da poslužitelj u inicijalni HTTP odgovor, kojim uspostavlja vezu s klijentom, dodaje HTTP zaglavlje s vrijednošću *Connection: keep-alive* kojim se specificira da kreirana veza ostaje trajno otvorena. Osim toga, u svaki pojedini HTTP odgovor poslužitelj dodaje HTTP zaglavlje *Content-Type: text/event-stream* kako bi osigurao raspoznavanje poruke od strane klijenta. [prema 100] Specifičnost Server-Sent Events tehnologije je da umjesto konstantnog *pinganja* servera koristi koncept odašiljanja (engl. *push*) poruke nakon što se dogodi akcija ili aktivnost koja uzrokuje promjenu trenutnog stanja.



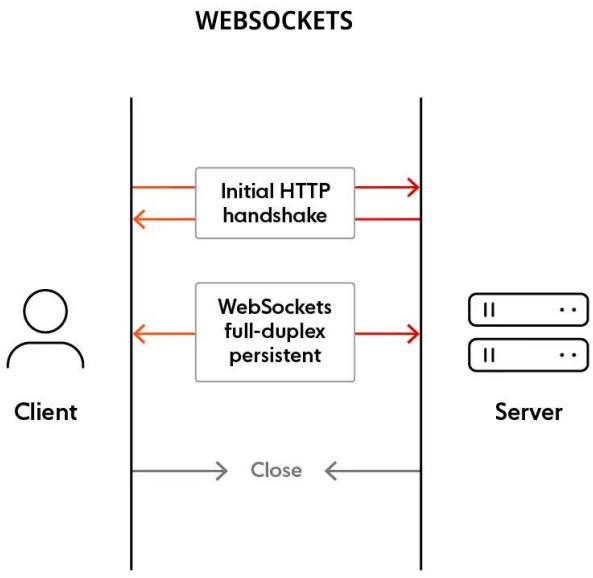
Slika 43. Server-Sent Events [prema 98]

5.8.2. Komunikacija u realnom vremenu korištenjem WebSocket-a

HTTP metode ostvarenja komunikacije u realnom vremenu temelje se na simulaciji što je ujedno i glavni razlog njihove neefikasnosti te vrlo uskog dijapazona primjene. Iako se Server-Sent Events čini kao idealan izbor ostvarenja komunikacije u realnom vremenu, on je i dalje suočen s problemima implementacije, koji su za uspostavu komunikacije u realnom vremenu pomoću HTTP-a neizbježni. Kao odgovor na probleme s kojima se susrela implementacija komunikacije u realnom vremenu korištenjem HTTP-a, 2011. godine predstavljen je WebSocket protokol koji je uzrokovao evoluciju komunikacije u realnom vremenu na Web-u. [94]

WebSocket protokol omogućuje i klijentu i poslužitelju slanje i primanje poruka u bilo kojem trenutku vremena bez potrebe za pamćenjem prethodnih zahtjeva. Bitna značajka korištenja WebSocket protokola je da gotovo svaki preglednik, u današnje vrijeme, podržava njegovo korištenje. WebSocket rješava sve probleme s kojima se susreo HTTP prilikom ostvarenja komunikacije u realnom vremenu. Za razliku od HTTP-a, kod kojeg zahtjev uvijek inicira klijent, a poslužitelj obrađuje zahtjev te šalje odgovor, WebSocket je dvosmjerni protokol što omogućuje da obje strane komunikacijskog kanala, i klijent i poslužitelj, mogu odigrati ulogu i primatelja i pošiljatelja. WebSocket protokol ne koristi `http://` ili `https://` shemu već koristi vlastitu `ws://`, odnosno `wss://` shemu ako se želi osigurati uspostava sigurne WebSocket veze. Kako je WebSocket protokol s pamćenjem stanja, veza ostaje otvorena sve dok je jedna od strana u komunikaciji, poslužitelj ili klijent, ne prekine. [94]

Komunikacija između klijenta i poslužitelja započinje na način da klijent inicira zahtjev za otvaranje WebSocket veze. Otvaranje WebSocket veze obavlja se postupkom TCP rukovanja (engl. *TCP Handshake*) između klijenta i poslužitelja, a sve s ciljem nadogradnje HTTP veze na WebSocket protokol. Nakon uspješne uspostave WebSocket veze, poslužitelj šalje odgovor sa statusnim kodom "101 Switching Protocols". Samim time, proces rukovanja je završen te se može krenuti s potpuno dvosmjernom razmjenom podataka između klijenta i poslužitelja.



Slika 44. WebSockets [prema 98]

5.9. Sigurnost Web aplikacija

Područje sigurnosti Web aplikacija posebno je vruća tema ukoliko aplikacija obrađuje strogo povjerljive podatke kao što su brojevi kreditnih kartica. U takvim situacijama je važno da se osigura anonimnost podataka koristeći pritom metode kao što je enkripcija podataka.

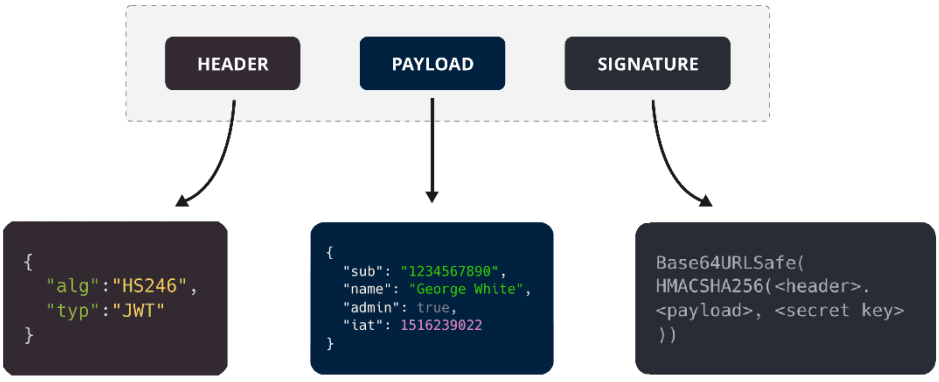
> *„Sigurnost Web aplikacija je koncept koji se temelji na izgradnji Web aplikacija na način da one funkcioniraju u skladu s očekivanjima čak i kada su napadnuta.“* [101]

Koncept sigurnosti Web aplikacija uključuje skup sigurnosnih kontrola ugrađenih u same Web aplikacije koje vrše zaštitu od raznih zlonamjernih agenata. Web aplikacija, kao i svaki drugi software, neizbježno sadrže nedostatke koji predstavljaju ranjivost Web aplikacije koju potencijalni napadači mogu vrlo jednostavno iskoristiti. Sigurnost Web aplikacija usmjerena je na sprječavanje takvih incidenata uz istovremeno smanjenje rizika od njihovog nastanka. [101]

5.9.1. JWT (engl. *JSON Web Token*)

JWT, odnosno JSON Web Token je otvoreni standard, predstavljen 2010. godine, koji se koristi za siguran i pouzdan prijenos informacija i podataka između dviju ili više strana. JWT se ponajprije koristi za autorizaciju korisnika prilikom komunikacije sa servisima koji ne čuvaju stanje, odnosno *stateless* servisima. Jedan od primjera *stateless* Web servisa je i REST API. Podaci, odnosno *claim*, koje JWT prenosi spremaju se u JSON formatu te digitalno potpisuju kako bi se očuvao integritet podataka. [102]

JWT se sastoji od nekoliko odvojenih JSON objekata između kojih se nalazi točka. Iako postoje razne implementacije JWT-a, najčešći oblik je digitalno potpisani JWT koji se sastoji od tri glavna dijela – *zaglavlje (engl. header)*, *korisni teret (engl, payload)* te *digitalni potpis (engl. digital signature)*. Svaki od triju navedenih dijelova se šifrira pomoću Base64 kodiranja te zatim sastavlja u predefiniranu strukturu JWT-a. [103] Prilikom kreiranja HTTP zahtjeva JWT se dodaje u zaglavlje kao vrijednost ključa *Authorization*.



Slika 45. Struktura JWT-a [104]

5.9.2. Transakcije u Web aplikacijama

Razvoj te sama primjena suvremenih informacijsko-komunikacijskih tehnologija omogućila je jednostavnije te brže obavljanje poslovnih procesa. S obzirom na trenutno stanje u svijetu, uzrokovano pandemijom Covid-a 19, poslovanje se sve više počinje obavljati putem trenutno dostupnih tehnologija. Prije svega, tu govorimo o raznim Web trgovinama koje su uslijed pandemije postale nezamjenjiv način provođenja poslovanja te održavanja stabilnog tržišnog položaja. Međutim, provođenje transakcija u elektroničkom obliku nije tako jednostavno kao što bi se moglo očekivati. Prije svega radi se o riziku izlaganja vlastitih podataka, kao što su brojevi kreditnih kartica i slično, koji zatim putuju nesigurnim kanalom zbog čega je važno da se osigura integritet podataka.
Primjerice, prilikom izrade Web trgovine vrlo važno pitanje koje se postavlja je:

„*Koje načine plaćanja uključiti u aplikaciju?“*

Postoje razni načini plaćanja putem Web trgovine koji prije svega uključuju korištenje suvremenih informacijsko-komunikacijskih tehnologija, a među najpopularnijima su:

1. **PayPal** – jedan od najpopularnijih i najčešće korištenih oblika plaćanja u Web aplikacijama. Primjerice, neke od najpoznatijih Web trgovina, kao što su Amazon i eBay, koriste upravo PayPal za provođenje transakcija prilikom kupovine određenog proizvoda. PayPal kao takav pruža određenu fleksibilnost i sigurnost. U slučaju da dođe do određenih problema s nepoštivanjem dogovorenih uvjeta između platitelja i primatelja, moguće je tražiti povrat novca te će zatim PayPal podrška razmotriti zahtjev, prikupiti sve potrebne dokaze kako bi takav slučaj riješila na korektan način.
2. **Kartično plaćanje** – kupac završava kupovinu na način da unosi broj kartice te CVV, odnosno sigurnosti kod kako bi obavio plaćanje kupljenih dobara ili usluga. Međutim, implementacija kartičnog plaćanja zahtjeva korištenje posrednika, odnosno *Payment Gateway-a*, koji stoji između aplikacije i banke te igra ulogu treće strane kojoj se vjeruje. Među najpopularnijim posrednicima su svakako *WSPay*, *Corvus Pay*, *MyCheckout* te *T-com PayWay*. Implementacija ovakvog načina plaćanja, plaća se na mjesečnoj ili godišnjoj razini. Prosječna cijena iznosi 200 kuna mjesečno, ovisno o *Payment Gateway-u* (posredniku). Također103r, prema zakonu, u Republici Hrvatskoj kartično plaćanje putem Web se smatra se kao promet gotovinom te je potrebno izdati fiskalizirani račun.

5.9.2.1. Payment Gateway

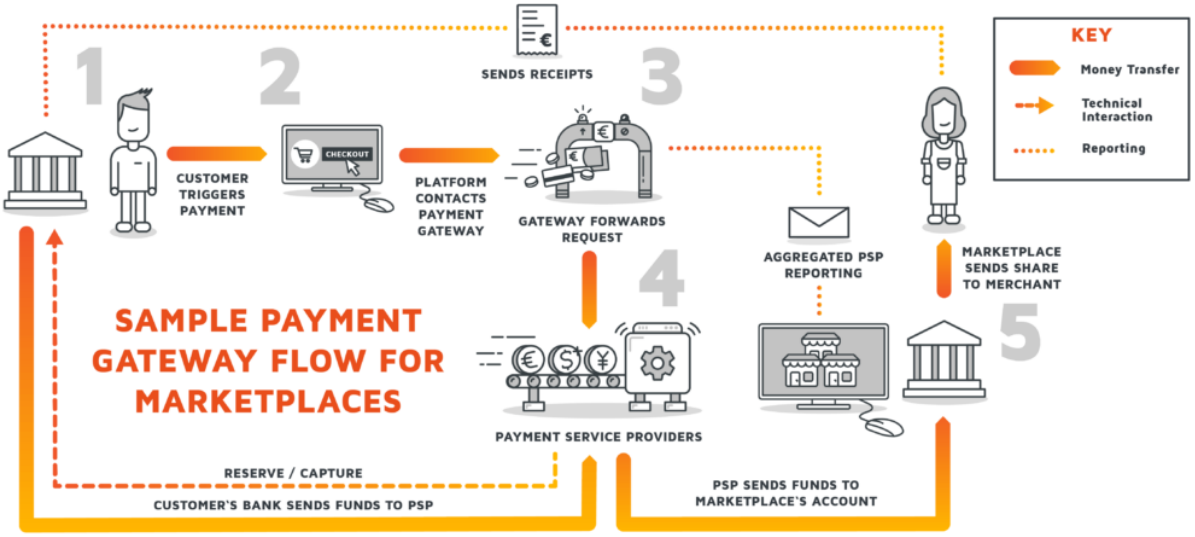
Izrada Web trgovine otvara niz pitanja, a odabir Payment Gateway-a jedna je ključnih odluka za buduće poslovanje. Prema najnovijim istraživanjima trend kupovine putem Interneta u Hrvatskoj nezaustavljivo raste. Prema najnovijim istraživanjima čak 45% Hrvata se odlučuje na ovaj oblik kupovine te se očekuje kako bi do kraja 2023. taj udio mogao narasti i do 75%. Isto tako, svijest o prednostima korištenja online trgovine nezaustavljivo raste te stoga ne čudi činjenica kako se sve više poduzetnika slaže da je izrada web trgovine jedna od najprofitabilnijih poslovnih odluka današnjice.

Pojam posrednika, odnosno Payment Gateway-a, odnosi se na sustav sigurne autentifikacije, autorizacije i naplate karticama putem Interneta. Na taj je način omogućena sigurna, brza i jednostavna kupovina za same krajnje korisnike koja se provodi putem treće strane kojoj se vjeruje. Payment Gateway je naziv za platformu koja služi za autorizaciju online financijskih transakcija koje se odvijaju u realnom vremenu. [105] Dakle, riječ je o integriranom softverskom rješenju koje automatizira proces naplate proizvoda, odnosno usluga između kupca (naručitelja), Web trgovine i banke.

Proces naplate se odvija tako da Payment Gateway zaprima podatke klijentskih kreditnih kartica te ih prosljeđuje prema banci koja zatim autorizira ili odbija transakciju. Informacija o odobrenju, odnosno odbijanju transakcije povratno se, posredstvom Payment Gateway sustava, dostavlja platitelju i banci, odnosno kartičnoj tvrtki. Ukratko, kada se radi o internetskoj trgovini, Payment Gateway obavlja identičnu svrhu kao i POS (engl. *Point Of Sale*) uređaj u fizičkim trgovinama, odnosno posreduje između prodavača i banke kako bi se omogućila realizacija sigurne naplate. S obzirom da ovakav način provođenja transakcija podrazumijeva obradu osjetljivih informacija, kao što su brojevi kreditnih kartica, kontrolni broj kartice i slično, posebna pažnja se posvećuje mjerama zaštite podataka kupaca u procesu online kupovine.

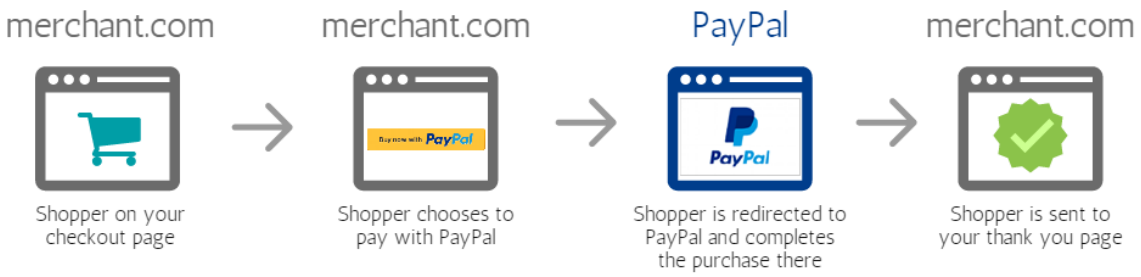
Payment Gateway sustavi, osim brzine izvršenja transakcije, imaju integriranu zaštitu podataka kupaca. To znači da funkcioniraju prema najsuvremenijim sigurnosnim standardima kartičnog plaćanja po principu nekoliko različitih slojeva enkripcije podataka – stoga podaci koje kupac upisuje prilikom kupnje nisu dostupni čak niti trgovcu. Uz to, svaka ozbiljnija Web trgovina mora imati, osim vidljivo istaknutih informacija o zaštiti osobnih podataka, integriran neki od SSL (engl. *Secure Sockets Layer*) certifikata čija je osnovna funkcija osiguravanje sigurnog prijenosa podataka između web preglednik (klijenta) te poslužitelja. Zaštita podataka se odvija tako što SSL kriptira unesene podatke i tako sprječava neovlašteno korištenje, presretanje i zlouporabu osobnih podataka nužnih za realizaciju online kupovine. Uz to, banke vlasnicima kartica omogućuju dodatnu zaštitu prilikom online kupnje kao što je, primjerice, korištenje 3-D sigurnosnog standarda, odnosno tokena koji se mogu koristiti u posebno

označenim Web trgovinama (npr. MasterCard SecureCode, Verified by Visa itd.). U osnovi, cijeli proces obavljanja transakcije temelji se na prisustvu neovisne treće strane koja provodi autentifikaciju i autorizaciju korisnika temeljem unesenih podataka. Sam proces implementacije plaćanja u Web aplikacijama nije univerzalan već on ovisi o pružatelju usluge posredništva. Svaki posrednik definira vlastita sučelja i način rada zbog čega je važno detaljno proučiti dokumentaciju koja je dostupna za odabrani Payment Gateway.



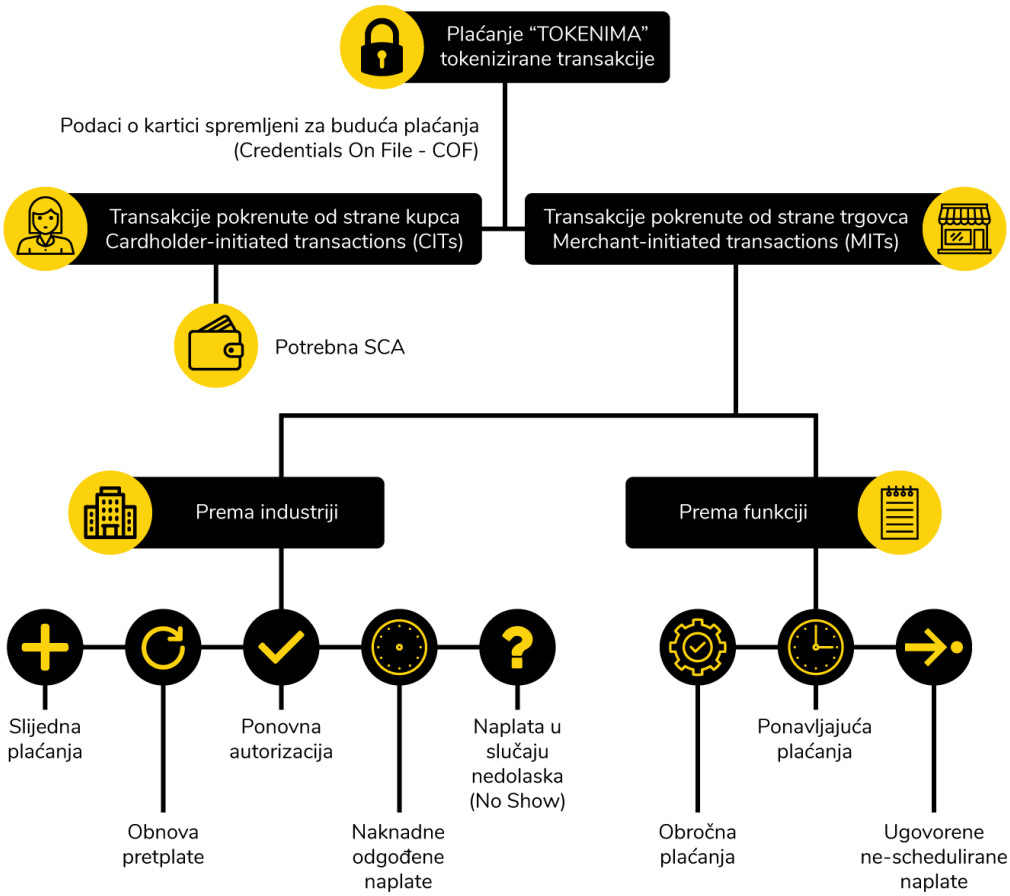
Slika 46. Koncept rada Payment Gateway-a [106]

PayPal je jedan od najpopularnijih i najčešće korištenih Payment Gateway sustava za plaćanja u Web aplikacijama. Neke od osnovnih karakteristika PayPal-a su visoka razina anonimnosti prilikom kupnje kreditnim karticama te jednostavno rješavanje pritužbi i reklamacija. PayPal usluga je, uz cijenu od samo 2.9% + 30¢ po transakciji, jedna od trenutačno najisplativijih opcija dostupnih na tržištu. Otvaranje korisničkog računa je besplatno, dodatno se naplaćuju samo napredne opcije. Sustav odobrava uplate u 26 stranih valuta, među kojima su i kune dok je korisnicima iz Hrvatske omogućena registracija uz MasterCard, Visa i Visa Electron kartice. [106]



Slika 47. Koncept rada PayPal Payment Gateway-a [107]

WSPay je jednostavan i praktičan Payment Gateway koji omogućuje vrlo brzu integraciju s bilo kojim sustavom, a postoje čak i gotovi moduli za Open Source rješenja. WSPay je jedan od najpoznatijih hrvatskih softverskih rješenja za online naplatu kod kojega se koriste unaprijed pripremljeni moduli koji su kompatibilni s velikim brojem Open Source platformi. Važno je napomenuti da se provizija za korištenje WSPay-a plaća na godišnjoj razini u iznosu od 2 500 kn i to neovisno o broju realiziranih transakcija. Ovaj sustav naplate koristi se u svim aplikacijama Fakulteta organizacije i informatike u kojima se provode transakcije. WSPay koristi sustav tokenizacije transakcija. Tokenizacija je proces pohrane kartičnih podataka na WSPay-u, odnosno u sigurnom PCI DSS L1 okruženju, uz prethodnu provjeru ispravnosti unesenih podataka i uz SCA (engl. *Strong Customer Autentification*) s ciljem da se kupcima omogući brzo plaćanje, bez unosa kartičnih podataka, kod ponovne kupnje na online prodajnom mjestu. [108] Rezultat tokenizacije je *token* te *token broj* (engl. *token number)* koji se koriste za provođenje brzih transakcija kod sljedećih plaćanja, a prema pravilima koja propisuje PSD2 regulativa i kartične sheme. Tokenizirana transakcija, koja se provodi na WSPay-u, je transakcija koja je napravljena korištenjem tokena i token broja uz dodatnu SCA (engl. *Strong Customer Authentication*) ili bez SCA.



Slika 48. Koncept rada WSPay Payment Gateway-a [108]

6. Implementacija aplikacije

Prilikom razvoja Web aplikacije korišteni se programski jezici JavaScript i Java, točnije programski okviri *React*, za *frontend* dio, te *Spring Boot*, za *backend* dio Web aplikacije. Osim toga, korišten je *Hibernate* kako bi se omogućilo objektno-relacijsko mapiranje podataka. Drugim riječima, *Hibernate* pruža okvir koji omogućava mapiranje objektno orijentiranog modela podatkovne domene u model relacijske baze podataka. Kao sustav za upravljanje bazom podataka korišten je *PostgreSQL* koji se nameće kao idealan alat upravo zbog otvorenosti koda, jednostavnosti korištenja te velike potpore zajednice. Valja spomenuti kako prilikom samog razvoja Web aplikacije nije korištena vanjska biblioteka koja pruža gotove elemente, odnosno komponente, već je razvijena vlastita biblioteka komponenata. Programski kod, zajedno s tehničkom dokumentacijom za pokretanje dostupan je u repozitorijima `ChainReaction`, `ChainReactionAPI` te `ChainReactionWS`.

6.1. Opis aplikacijske domene

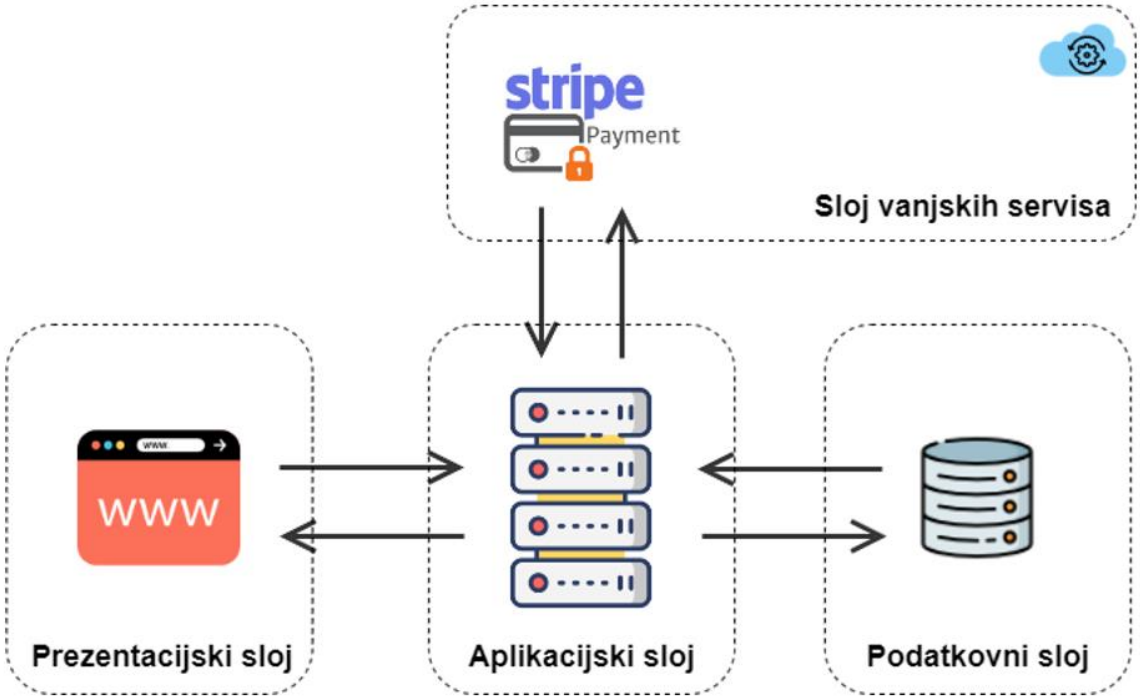
U sklopu praktičnog dijela rada sukcesivno je opisan proces razvoja Web aplikacije u okviru kojeg je implementirana Web aplikacija koja krajnjim korisnicima omogućuje kupnju bicikala, dijelova i opreme za bicikle kao i rezervaciju elektroničkih bicikala, pri čemu se poseban naglasak stavlja na maksimalno zadovoljavanje korisničkih potreba, a samim time i poboljšanje korisničkog iskustva prilikom korištenja elektroničke trgovine. Web aplikacija krajnjim korisnicima omogućava rezervaciju posebne kategorije bicikala, točnije elektroničkih bicikala, u jednom od dostupnih rezervacijskih centara. U budućnosti razvoja aplikacije pojedini rezervacijski centri bit će prikazani na Google karti, s obzirom na trenutnu lokaciju prijavljenog korisnika te odabrani pojas pretraživanja dostupnih rezervacijskih centara, zajedno s brojem bicikala koji su dostupni za rezervaciju na svakoj od lokacija. Osim toga, krajnji korisnici imaju mogućnost filtriranje proizvoda kao i mogućnost pregleda detalja pojedinog proizvoda. Najbitnija funkcionalnosti same aplikacije je korisnička košarica koja služi kao primarno spremište artikala koje krajnji korisnik planira naručiti u bližoj ili daljoj budućnosti. Unutar same Web aplikacije implementiran je koncept obavijesti koji se temelji na komunikaciji klijentskog i poslužiteljskog dijela aplikacije u realnom vremenu, a čiji je primarni cilj izvještavanje korisnika o promjenama statusa rezervacije, odnosno narudžbe.. Krajnjim je korisnicima omogućeno plaćanje pouzećem, odnosno gotovinom prilikom preuzimanja narudžbe, kao i plaćanje kreditnim karticama. Za svaku od kreiranih narudžbi omogućen je zaseban pregled koji pruža uvid u detalje svakog od artikla od kojih je narudžba sastavljena.

6.2. Arhitektura aplikacije

Odabir odgovarajuće arhitekture aplikacije ključni je korak u ranim fazama razvoja i dizajna aplikativnog sustava. Arhitektura pojedinog sustava temelji se na odgovarajućem arhitekturnom uzorku dizajna koji daje osnovne smjernice vezane uz strukturiranje i razvoj aplikacije.

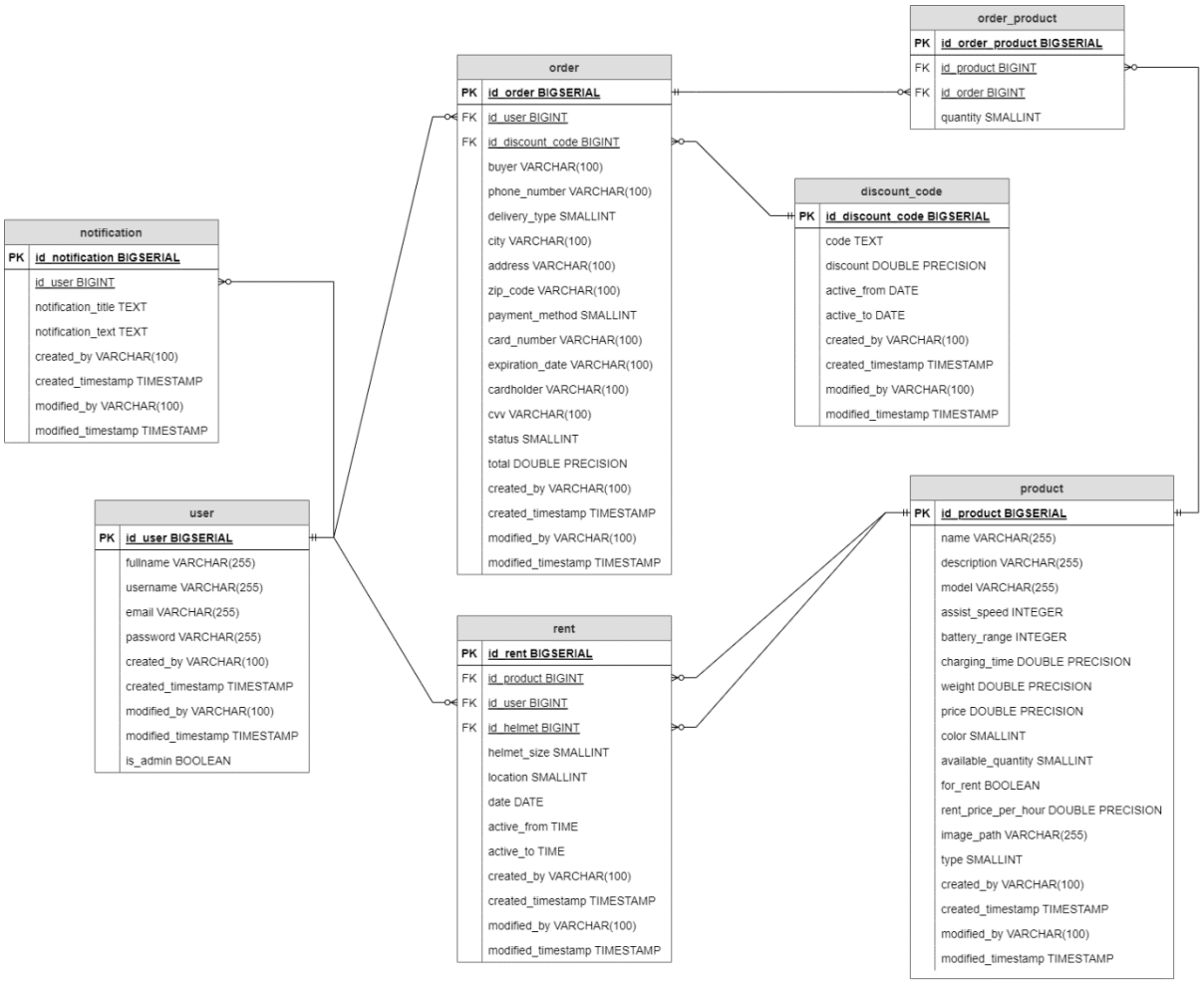
„Arhitekturni uzorak dizajna izražava temeljnu organizacijsku shemu strukture aplikativnog sustava. On pruža skup predefiniranih podsustava, specificira njihove uloge i odgovornosti te uključuje pravila i smjernice za organizaciju odnosa između njih.“ [109]

Povećanje složenosti i robusnosti aplikativnih sustava dovelo je do razvoja velikog broja arhitekturnih uzoraka dizajna, a među najznačajnijima su *layers*, *broker*, *MVC (engl. Model-View-Controller)* te *PAC (engl. Presentation-Abstraction-Control)*. Aplikacije koja je razvijena u sklopu praktičnog dijela ovog rada temelji se na Model-View-Controller arhitekturnom uzorku dizajna koji je ujedno i najčešće korišteni arhitekturni uzorak dizajna. Prema navedenom uzorku, aplikacija se dijeli u tri sloja visoke razine apstrakcije, odnosno u tri zasebne komponente koje djeluju kao cjelina – komponenta *model* koja definira strukturu podataka pojedinog entiteta u relacijskoj bazi podataka, komponenta *view* koja prikazuje informacije i podatke krajnjim korisnicima tek komponenta *controller* koja obrađuje pristigle zahtjeve.



Slika 49. Arhitektura aplikacije

6.3. Podatkovni model



Slika 50.ERA model baze podataka

ERA model prikazuje osnovnu strukturu baze podataka, odnosno relacije u koje se pohranjuju podaci koji se generiraju unutar Web aplikacije – `obavijesti`, `korisnici`, `narudžbe`, `rezervacije`,... Kao sustav za upravljanje bazom podataka odabran je PostgreSQL, dok je za objektno-relacijsko preslikavanje odabran programski okvir Hibernate. U svrhu očuvanja konzistentnosti baze podataka na lokalnoj razvojnoj okolini pojedinog developera, kao i na testnoj te produkcijskoj razvojnoj okolini, korišten je alat Liquibase koji omogućava verzioniranje baze podataka.

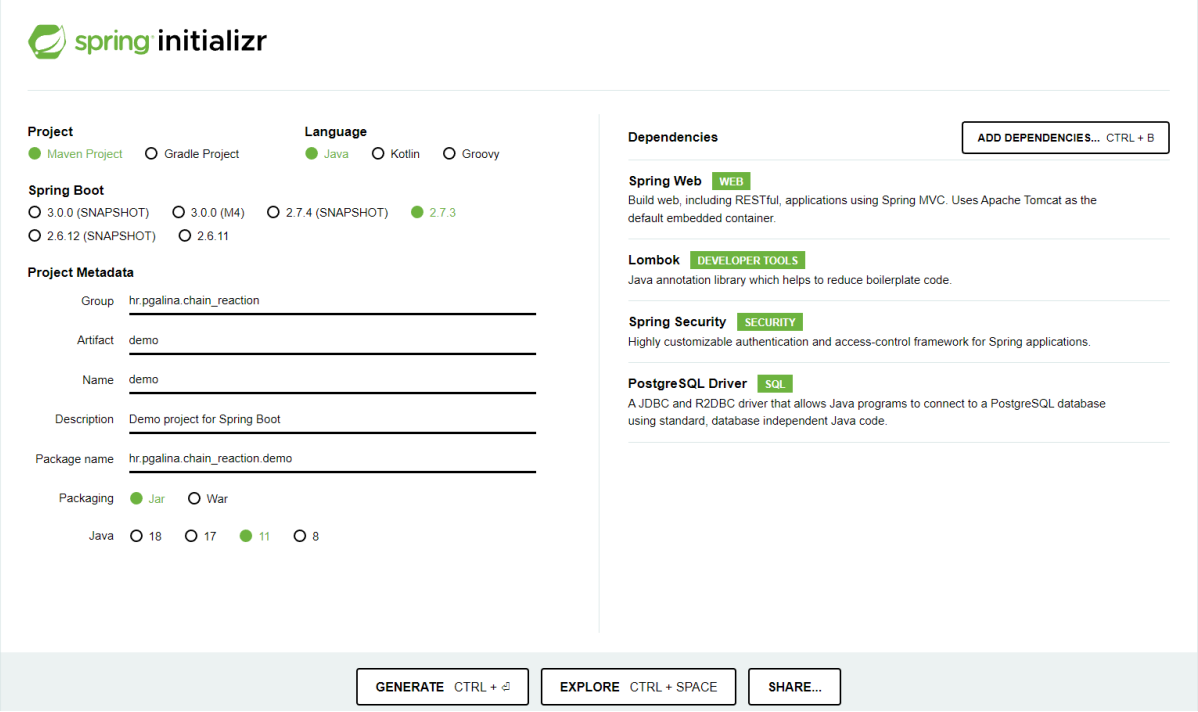
Svaka narudžba (`order`) vezana je na pripadajućeg korisnika (`user`) te opcionalno na kod za popust (`discount_code`) ukoliko je isti primijenjen prilikom kreiranja korisničke narudžbe. Stavke pojedine narudžbe, zajedno s naručenim količinama, pohranjuju se u relaciji slabog entiteta `order_product`. Svaka rezervacija (`rent`) vezana je na pripadajućeg korisnika (`user`) te proizvod (`product`) koji se želi rezervirati. Svaka obavijest (`notification`) vezana je na korisnika (`user`) kojem treba biti prikazana.

6.4. Razvoj na strani poslužitelja

Poslužiteljski dio Web aplikacije razvijen je korištenjem programskog jezika Java, točnije programskog okvira Spring Boot koji omogućava rapidni razvoj aplikacija koristeći pritom koncept anotacija te injektiranja ovisnosti. Sam poslužiteljski dio aplikacije sastoji se od dvaju odvojenih modula koji su implementirani kao zasebne, *standalone* aplikacije – REST API poslužitelj koji podržava klijentsku komunikaciju putem unaprijed definiranih krajnjih točaka te WebSocket poslužitelj koji omogućava komunikaciju u realnom vremenu između klijentskog i poslužiteljskog dijela aplikacije.

6.4.1. Inicijalizacija projekta

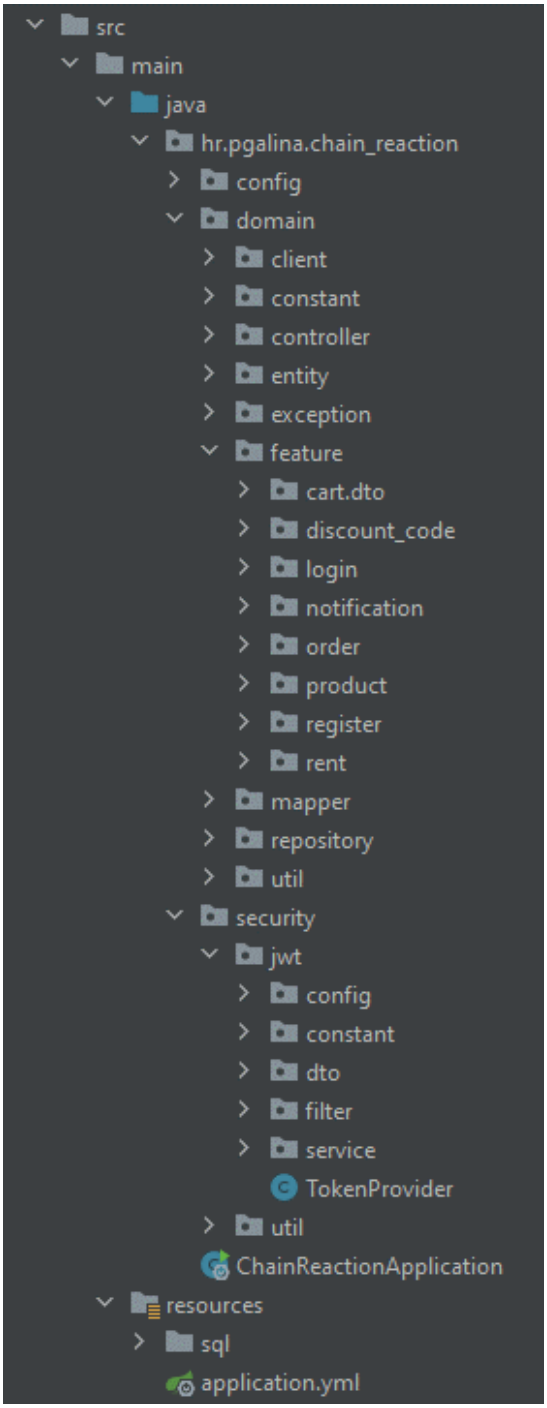
Projekt je inicijaliziran korištenjem *Spring Initializr* alata koji generira osnovnu strukturu Spring Boot projekta zajedno sa svim odabranim ovisnostima. Nakon generiranja projekta dobiva se *.rar* datoteka koja sadrži sve datoteke koje su potrebne za otvaranje projekta u odabranom razvojnom okruženju (IntelliJ IDEA, Eclipse, Netbeans,…). *Spring Initializr* dostupan je u obliku Web aplikacije putem linka https://start.spring.io/.



Slika 51. Sučelje Spring Initializr alata

6.4.2. Struktura projekta

Struktura direktorija na poslužiteljskoj strani aplikacije prati suvremene trendove razvoja aplikacija te je osmišljena s ciljem pridržavanja *clean code* koncepata te jednostavnog održavanja sustava u kasnijim fazama razvoja projekta. `ChainReactionApplication`, odnosno klasa anotirana sa `@SpringBootApplication` primarna je klasa svakog Spring Boot projekta koja ujedno sadrži `main` metodu putem koje se vrši konfiguracija te pokretanje same aplikacije.



Slika 52. Struktura poslužiltejskog dijela aplikacije

6.4.3. Konfiguracijske datoteke

Konfiguracijske datoteke, odnosno klase Spring Boot projekta dekoriraju se anotacijom `@Configuration` koja označava da klasa imaju jednu ili više metoda dekoriranih anotacijom `@Bean`. Na taj se način pružaju metapodaci koje Spring kontejner koristi prilikom učitavanja konfiguracijskih datoteka tijekom pokretanja aplikacije. Spring verzije 2 zahtijevao je deklariranje *bean* klasa unutar *web.xml* datoteke, dok je od verzije 3 dopušteno definiranje konfiguracijskih datoteka unutar samih Java klasa korištenjem ranije spomenutih anotacija. [110] Aplikacija se sastoji od triju konfiguracijskih datoteka – `CORSConfiguration`, `Jackson Configuration` te `SecurityConfiguration`.

Klasa `SecurityConfiguration`, točnije klasa koja proširuje apstraktnu klasu `WebSecurityConfigurerAdapter` te je anotirana sa `@Configuration`, primarna je i najvažnije konfiguracijska datoteka projekta. Njezina primarna svrha je definiranje pravila vezanih uz sigurnost, od ulazne točke za autentifikaciju pa sve do detalja vezanih uz pojedini HTTP zahtjev kao što je propuštanje HTTP zahtjeva pristiglih OPTIONS metodom. Ujedno, klasa sadrži anotacije `@EnableWebSecurity` te `@EnableGlobalMethodSecurity` kojima se pružaju metapodaci vezani uz uključivanje sigurnosti na razini cijele aplikacije.

```
1  @Configuration
2  @EnableWebSecurity
3  @EnableGlobalMethodSecurity(
4      prePostEnabled = true,
5      securedEnabled = true
6  )
7  public class SecurityConfiguration extends WebSecurityConfigurerAdapter
8  {
9
10     private final JWTAuthenticationEntryPoint
11         jwtAuthenticationEntryPoint;
12
13     private final TokenProvider tokenProvider;
14
15     public SecurityConfiguration(
16         JWTAuthenticationEntryPoint jwtAuthenticationEntryPoint,
17         TokenProvider tokenProvider
18     ) {
19         this.jwtAuthenticationEntryPoint = jwtAuthenticationEntryPoint;
20         this.tokenProvider = tokenProvider;
21     }
22
23     @Bean
24     public PasswordEncoder passwordEncoder() {
25         return new PasswordEncoder() {
26             final BCryptPasswordEncoder encoder =
27                 new BCryptPasswordEncoder();
28
29
```

```
30              @Override
31              public String encode(CharSequence rawPassword) {
32                  return encoder.encode(rawPassword);
33              }
34
35              @Override
36              public boolean matches(
37                  CharSequence rawPassword,
38                  String encodedPassword
39              ) {
40                  return encoder.matches(
41                      rawPassword,
42                      encodedPassword
43                  );
44              }
45          };
46      }
47
48      @Bean
49      public RestTemplate restTemplate() {
50          return new RestTemplate();
51      }
52
53      @Override
54      public void configure(WebSecurity web) {
55          web
56              .ignoring()
57                  .antMatchers(HttpMethod.OPTIONS, "/**");
58      }
59
60      @Override
61      protected void configure(HttpSecurity security) throws Exception
62      {
63          security
64              .csrf().disable()
65              .cors().disable()
66              .exceptionHandling()
67                  .authenticationEntryPoint(jwtAuthenticationEntryPoint)
68              .and()
69              .authorizeRequests()
70                  .antMatchers(HttpMethod.POST, "/api/users").permitAll()
71                  .antMatchers(HttpMethod.POST,"/api/authentication")
72                      .permitAll()
73                  .antMatchers(
74                       HttpMethod.POST,"/api/authentication/refreshToken"
75                   ).permitAll()
76                  .antMatchers("/api/**").authenticated()
77              .and().httpBasic()
78              .and().sessionManagement()
79                  .sessionCreationPolicy(SessionCreationPolicy.STATELESS)
80              .and().apply(securityConfigurerAdapter());
81      }
82
83      private JWTConfigurer securityConfigurerAdapter() {
84          return new JWTConfigurer(tokenProvider);
85      }
86  }
```

Klasa `CORSConfiguration` definira pravila vezana uz CORS (engl. *Cross-Origin Resource Sharing*), odnosno propuštanje ili blokiranje pristiglih zahtjeva temeljem informacija o domeni, vrsti zahtjeva i slično.

```
@Configuration
public class CORSConfiguration {

    @Bean
    public FilterRegistrationBean<CorsFilter> corsFilterBean(){
        FilterRegistrationBean<CorsFilter> filterRegistrationBean =
            new FilterRegistrationBean<>();

        UrlBasedCorsConfigurationSource source =
            new UrlBasedCorsConfigurationSource();
        CorsConfiguration configuration = new CorsConfiguration();

        configuration.addAllowedOrigin(CorsConfiguration.ALL);
        configuration.addAllowedHeader(CorsConfiguration.ALL);
        configuration.addAllowedMethod(CorsConfiguration.ALL);
        source.registerCorsConfiguration("/**", configuration);

        filterRegistrationBean.setFilter(new CorsFilter(source));
        filterRegistrationBean.setOrder(Ordered.HIGHEST_PRECEDENCE);

        return filterRegistrationBean;
    }
}
```

Klasa `JacksonConfiguration` služi za definiranje vanjskih modula koji će se koristiti prilikom serijalizacije i deserijalizacije podataka.

```
@Configuration
public class JacksonConfiguration {

    @Bean
    public JavaTimeModule javaTimeModule() {
        return new JavaTimeModule();
    }

    @Bean
    public Jdk8Module jdk8TimeModule() {
        return new Jdk8Module();
    }
}
```

6.4.4. Autentifikacija zahtjeva

U sklopu teorijskog dijela rada spomenut je JSON Web Token, skraćeno JWT, kao jedan od načina uspostave sigurnosti na Webu. Upravo je JWT korišten kao metoda uspostave sigurnosti unutar implementirane Web aplikacije. Spring Boot pruža unaprijed pripremljenu, `spring-boot-starter-security` biblioteku koja bitno olakšava implementaciju sigurnosti unutar aplikacija. Autentifikacija na strani poslužitelja implementirana je s ciljem ostvarenja sigurnosti pristupa pojedinoj krajnjoj točci s izuzetkom zahtjeva kreiranih POST metodom prema krajnjim točkama `/api/users` te `/api/authentication`. Implementacija autentifikacije na strani poslužitelja temelji se na primjeni filtera koji zaprima zahtjev prije njegovom prosljeđivanja sljedećem filteru unutar *filter chain-a*. Ukoliko HTTP zahtjev pristiže na zaštićenu krajnju točku bez tokena u zaglavlju zahtjeva, odnosno bez vrijednosti pod ključem `Authorization` ili s tokenom koji nije validan ili je istekao tada poslužitelj vraća HTTP odgovor sa statusom "401 Unauthorized". Ukoliko HTTP zahtjev pristiže na zaštićenu krajnju točku s validnim tokenom tada se zahtjev prosljeđuje sljedećem filteru u nizu ili se odobrava pristup krajnjoj točci ukoliko je autentifikacijski filter posljednji u *filter chain-u*. Primarne klase koje se bave sigurnošću krajnjih točaka su `JWTFilter` te `TokenProvider`.

Klasa `JWTFilter` djeluje kao filter pristiglih HTTP zahtjeva.

```
1  @Component
2  public class JWTFilter extends GenericFilterBean {
3
4      public static final String AUTHORIZATION_HEADER = "Authorization";
5
6      private final TokenProvider tokenProvider;
7
8      public JWTFilter(TokenProvider tokenProvider) {
9          this.tokenProvider = tokenProvider;
10     }
11
12     @Override
13     public void doFilter(
14         ServletRequest servletRequest,
15         ServletResponse servletResponse,
16         FilterChain filterChain
17     ) throws IOException, ServletException {
18         HttpServletRequest request =
19             (HttpServletRequest)servletRequest;
20        String jwt = resolveToken(request );
21        if (StringUtils.hasText(jwt)) {
22            if (this.tokenProvider.validateToken(jwt)) {
23                Authentication authentication =
24                    this.tokenProvider.getAuthentication(jwt);
25                 SecurityContextHolder
26                     .getContext()
27                     .setAuthentication(authentication);
28            }
```

```
            else {
                throw new BadRequestException(
                    ERROR,
                    HttpStatus.UNAUTHORIZED,
                    UNAUTHORIZED_REQUEST
                );
            }
        }

        filterChain.doFilter(servletRequest, servletResponse);
    }

    private String resolveToken(HttpServletRequest request) {
        String bearerToken = request.getHeader(AUTHORIZATION_HEADER);
        if (StringUtils.hasText(bearerToken) &&
            bearerToken.startsWith("Bearer ")) {
            return bearerToken.substring(7);
        }

        return null;
    }

```

Klasa `TokenProvider` sadrži metode koje omogućuju kreiranje JSON Web tokena, dekodiranje tokena, odnosno dohvaćanje objekta klase `Authentication`, koji sadrži osnove informacije, odnosno *claimove*, o korisniku koji pristupa krajnjoj točci, dohvaća *claimove*, autorizira korisnika te validira pristigli token.

```
@Component
public class TokenProvider {
    private Key key;

    private Long tokenValidityInMilliseconds;

    private JwtParser jwtParser;

    @Value("${jwt.token-validity-seconds}")
    private Long tokenValiditySeconds;


    @Value("${jwt.base64-secret}")
    private String base64Secret;

    @PostConstruct
    public void init() {
        byte[] keyBytes;
        keyBytes = Decoders.BASE64.decode(base64Secret);
        this.key = Keys.hmacShaKeyFor(keyBytes);
        this.tokenValidityInMilliseconds = 1000 *
            tokenValiditySeconds;
        this.jwtParser =
            Jwts.parserBuilder().setSigningKey(key).build();
    }
```

```
26      public String createToken(
27          User user,
28          Authentication authentication) {
29           String authorities =
30              authentication
31                  .getAuthorities()
32                  .stream()
33                  .map(GrantedAuthority::getAuthority)
34                  .collect(Collectors.joining(","));
35
36          long now = (new Date()).getTime();
37          Date validity = new Date(now +
38              this.tokenValidityInMilliseconds);
39
40          return Jwts
41              .builder()
42              .claim(JWT_CLAIM_ID_USER, user.getIdUser())
43              .claim(JWT_CLAIM_FULL_NAME, user.getFullname())
44              .claim(JWT_CLAIM_USERNAME, user.getUsername())
45              .claim(JWT_CLAIM_EMAIL_ADDRESS, user.getEmail())
46              .claim(AUTHORITIES_KEY, authorities)
47              .signWith(key, SignatureAlgorithm.HS512)
48              .setExpiration(validity)
49              .compact();
50      }
51
52      public String createToken(String refreshToken) {
53          long now = (new Date()).getTime();
54          Date validity = new Date(now +
55              this.tokenValidityInMilliseconds);
56
57          return Jwts
58              .builder()
59              .claim(JWT_CLAIM_ID_USER,
60                   getClaims(refreshToken).get(JWT_CLAIM_ID_USER))
61              .claim(JWT_CLAIM_FULL_NAME,
62                   getClaims(refreshToken).get(JWT_CLAIM_FULL_NAME))
63              .claim(JWT_CLAIM_USERNAME,
64                   getClaims(refreshToken).get(JWT_CLAIM_USERNAME))
65              .claim(JWT_CLAIM_EMAIL_ADDRESS,
66                   getClaims(refreshToken).get(JWT_CLAIM_EMAIL_ADDRESS))
67              .claim(AUTHORITIES_KEY,
68                   getClaims(refreshToken).get(AUTHORITIES_KEY))
69              .signWith(key, SignatureAlgorithm.HS512)
70              .setExpiration(validity)
71              .compact();
72      }
73
74
75      public Authentication getAuthentication(String token) {
76          Claims claims = getClaims(token);
77
78          Collection<? extends GrantedAuthority> authorities = Arrays
79              .stream(claims.get(AUTHORITIES_KEY).toString().split(","))
80              .map(SimpleGrantedAuthority::new)
81              .collect(Collectors.toList());
82          org.springframework.security.core.userdetails.User principal =
```

```
 83              new org.springframework.security.core.userdetails.User(
 84                  claims.get(JWT_CLAIM_USERNAME).toString(),
 85                  "",
 86                  authorities
 87          );
 88
 89         return new UsernamePasswordAuthenticationToken(
 90             principal,
 91             token,
 92             authorities
 93         );
 94     }
 95
 96     public Claims getClaims(String token) {
 97         return jwtParser
 98             .parseClaimsJws(token)
 99             .getBody();
100     }
101
102     public Long getCurrentUserId() {
103         Claims claims = getClaims(
104             SecurityUtils
105                 .getCurrentUserJWT()
106                 .orElseThrow(
107                     () -> new BadRequestException(UNKNOWN_JWT_TOKEN))
108         );
109
110         return claims.get(JWT_CLAIM_ID_USER, Long.class);
111     }
112
113     public boolean isCurrentUserAdmin() {
114         Claims claims = getClaims(
115             SecurityUtils
116                 .getCurrentUserJWT()
117                 .orElseThrow(
118                     () -> new BadRequestException(UNKNOWN_JWT_TOKEN))
119         );
120
121         return claims.get(
122             AUTHORITIES_KEY,
123             String.class).equals(ADMIN);
124     }
125
126     public boolean validateToken(String token) {
127         try {
128             jwtParser.parseClaimsJws(token);
129             return true;
130         } catch (JwtException | IllegalArgumentException ex) {
131             log.info(INVALID_JWT_TOKEN);
132         }
133         return false;
134     }
135 }
```

6.4.5. Podatkovni modeli

Podatkovni model prva je od triju komponenti MVC uzorka dizajna na kojem se temelji razvijeni aplikativni sustav. Za svaku od relacija unutar relacijske baze podataka kreira se POJO objekt, odnosno entitet koji definira objektnu strukturu relacije te na taj način omogućuje odrađivanje objektno-relacijskog preslikavanja od strane programskog okvira Hibernate. Klasa entiteta dekorira se anotacijom `@Entity`. Anotacija `@Table`, koja dodatno dekorira klasu, definira relaciju unutar relacijske baze podataka koju pripadni entitet reprezentira, odnosno na koju se pripadni entitet veže. Svaka klasa entiteta **mora** imati definirano svojstvo anotirano sa `@Id` koje predstavlja njegov jednoznačni identifikator. Anotacija `@Column` služi za definiranje naziva atributa na kojeg se veže pojedino svojstvo objekta, pri čemu je važno napomenuti kako anotacija nije obavezna ukoliko naziv svojstva entiteta odgovara atributu tablice na koju se klasa entiteta, odnosno konkretno svojstvo entiteta, veže. Veze između pojedinih entiteta definiraju se uz pomoć anotacija `@JoinColumn`, `@OneToOne`, `@OneToMany`, `@ManyToOne` te `@ManyToMany`.

Klasa entiteta `Order` predstavlja objektnu reprezentaciju relacije `order`.

```
1  @Setter
2  @Getter
3  @Entity
4  @Table(name = "order", schema = "public")
5  public class Order extends BaseAuditEntity {
6
7      @Id
8      @GeneratedValue(strategy = GenerationType.IDENTITY)
9      @Column(name = "id_order")
10     private Long idOrder;
11
12     @ManyToOne
13     @JoinColumn(
14         name = "id_user",
15         referencedColumnName = "id_user"
16     )
17     private User user;
18
19     @ManyToOne
20     @JoinColumn(
21         name = "id_discount_code",
22         referencedColumnName = "id_discount_code"
23     )
24     private DiscountCode discountCode;
25
26     @OneToMany(
27         mappedBy = "order",
28         cascade = CascadeType.ALL,
29         orphanRemoval = true
30     )
31     private List<OrderProduct> products;
```

```
32      @Column(name = "buyer")
33      private String buyer;
34
35      @Column(name = "phone_number")
36      private String phoneNumber;
37
38      @Column(name = "delivery_type")
39      private Short deliveryType;
40
41      @Column(name = "city")
42      private String city;
43
44      @Column(name = "address")
45      private String address;
46
47      @Column(name = "zip_code")
48      private String zipCode;
49
50      @Column(name = "payment_method")
51      private Short paymentMethod;
52
53      @Column(name = "card_number")
54      private String cardNumber;
55
56      @Column(name = "expiration_date")
57      private String expirationDate;
58
59      @Column(name = "cardholder")
60      private String cardholder;
61
62      @Column(name = "cvv")
63      private String cvv;
64
65      @Column(name = "status")
66      private Short status;
67
68      @Column(name = "total")
69      private Double total;
70  }
```

6.4.6. Aplikacijsko programsko sučelje

Poslužiteljski dio aplikacije povezan je s klijentskom stranom putem REST API-a koji pruža krajnje točke putem kojih klijenti pristupaju željenim informacijama i podacima. REST API predstavlja ugovor između poslužitelja i klijen(a)ta koji osigurava sigurnu i pouzdanu komunikaciju temeljenu na REST arhitekturi. Aplikacijsko programsko sučelje zaduženo je za rukovanje pristiglim HTTP zahtjevima te vraćanje korespondentnih HTTP odgovora.

6.4.6.1. Kontroler

Ulazne točke pojedinog klijentskog HTTP zahtjeva nazivaju se REST kontrolerima. REST kontroler posljednja je od triju komponenti MVC uzorka dizajna na kojem se temelji razvijeni aplikativni sustav. Svaki pristigli zahtjev, nakon prolaska kroz definirani *filter chain*, dolazi do odgovarajućeg kontrolera koji vrši daljnju obradu zahtjeva. Neka od načela *Java najboljih praksi* nalažu da kontroler prosljeđuje zahtjev pozivom metode servisa (engl. *service*) ili fasade (engl. *facade*) u ovisnosti o primijenjenom uzorku dizajna. Klasa kontrolera dekorira se anotacijom `@RestController`. Anotacija `@RequestMapping`, koja dodatno dekorira klasu, definira krajnju točku, odnosno strukturu URL-a za koju je pripadni kontroler nadležan. Pojedine specijalizacije primarne putanje kontrolera definiraju se zasebnim metodama koje se označavaju anotacijama `@GetMapping`, `@PostMapping`, `@PutMapping` te `@DeleteMapping` u ovisnosti od vrste HTTP metode za koju su nadležni. Spring Boot zajednica, kao i njezini tvorci, još uvijek nije suglasna s pozicijom poziva validacijskih metoda. Prema nekim autorima, metode validatora pozivaju se odmah po ulasku u pojedinu metodu kontrolera uz argument da se na taj način skraćuje životni vijek HTTP zahtjeva koji nisu validni. S druge strane, postoje autori koji validatorske metode pozicioniraju unutar servisa, odnosno fasada uz argument da se na taj način čuva čistoća koda kontrolera.

Klasa `ProductController` igra ulogu kontrolera za zahtjeve koji pristižu na krajnju točku `/api/products` kao i njezine specijalizacije.

```
 1  @Slf4j
 2  @RestController
 3  @RequestMapping("/api/products")
 4  @RequiredArgsConstructor
 5  public class ProductController {
 6
 7      private final ProductValidator productValidator;
 8
 9      private final ProductService productService;
10
11      @GetMapping
12      public ResponseEntity<List<ProductDto>>
13      fetchProductsByProductTypeAndProductName(
14          @RequestParam String productType,
```

```
15          @RequestParam(required = false) String productName
16      ) {
17          log.info("Entered '/api/products' with product type {} and
18          product name {} [GET].",
19              productType,
20              productName
21          );
22
23          productValidator.validateProductType(productType);
24
25          return new ResponseEntity<>(
26              productService
27                  .getProductsByProductTypeAndProductName(
28                      ProductType.findByValue(productType),
29                      productName
30                  ), HttpStatus.OK
31          );
32      }
33
34      @GetMapping("/{idProduct}")
35      public ResponseEntity<ProductDto> fetchProductById(@PathVariable
36      Long idProduct) {
37          log.info("Entered '/api/products/{idProduct}' with product ID
38          {} [GET].", idProduct);
39
40          return new  ResponseEntity<>(
41              productService
42                 .getProductById(idProduct), HttpStatus.OK);
43      }
44
45      @PostMapping("/filter")
46      public ResponseEntity<ProductPage> fetchProductsByFilter (
47          @RequestParam(defaultValue = "1") Integer page,
48          @RequestParam(defaultValue = "9999") Integer size,
49          @RequestParam(required = false) ArrayList<Short> productTypes,
50          @RequestBody ProductFilter filter
51      ) {
52          log.info("Entered '/api/products/filter' with page {}, size {},
53          productTypes {} and filter {} [POST].",
54              page,
55              size,
56              productTypes,
57              filter
58          );
59
60          return new ResponseEntity<>(
61              productService
62                  .getProductsByFilter(
63                      page,
64                      size,
65                      productTypes,
66                      filter
67                  ), HttpStatus.OK
68          );
69      }
70  }
```

6.4.6.2. Validator

Validator, kao komponenta, osigurava ispravnost podataka koji dolaze putem HTTP zahtjeva. Spring Boot razlikuje dvije vrste validatora – ugrađeni (engl. *built-in*) validatori koji se uključuju dodavanje paketa `spring-boot-starter-validation` te vlastito implementirati (engl. *custom*) validatori.

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-validation</artifactId>
</dependency>
```

Ugrađeni validatori temelje se na anotacijama kojima se dekoriraju, odnosno ograničavaju pojedina svojstva POJO-a. Validacija putem ugrađenih validatora obavlja se prilikom serijalizacije vrijednosti JSON formata u korespondentni POJO. Najčešće korištene anotacije za validaciju su `@NotNull`, `@NotEmpty`, `@NotBlank` te `@Pattern`.

Custom validatori dekoriraju se anotacijom `@Component` kako bi mogli biti injektirani u klase u kojima će biti korišteni. Uobičajeno se koriste za provjeru duplikata zapisa te ispravnosti pristiglih podataka u odnosu na šifrarnike ili vrijednosti iz baze podataka.

Klasa `OrderValidatorImpl` obavlja validaciju podataka prilikom kreiranja nove narudžbe.

```
@Component
@RequiredArgsConstructor
public class OrderValidatorImpl implements OrderValidator {

    private final ProductRepository productRepository;

    @Override
    public void validateOrderForm(OrderForm orderForm) {
        DeliveryType deliveryType =
            DeliveryType
                .findByIdDeliveryType(
                    orderForm.getDeliveryType().getIdDeliveryType()
                );

        if (Objects.isNull(deliveryType)) {
            throw new BadRequestException(
                ErrorTypeConstants.ERROR,
                HttpStatus.NOT_FOUND,
                DELIVERY_TYPE_DOES_NOT_EXIST
            );
        }

        PaymentMethod paymentMethod =
            PaymentMethod
                .findByIdPaymentMethod(
                    orderForm.getPaymentMethod().getIdPaymentMethod()
                );

```

```
29          if (Objects.isNull(paymentMethod)) {
30              throw new BadRequestException(
31                  ErrorTypeConstants.ERROR,
32                  HttpStatus.NOT_FOUND,
33                  PAYMENT_METHOD_DOES_NOT_EXIST
34              );
35          }
36
37          orderForm
38              .getProducts()
39              .stream()
40              .forEach(cartItem -> {
41                  Product product = productRepository
42                      .findById(cartItem.getIdProduct())
43                      .orElseThrow(
44                          () -> new BadRequestException(
45                              ErrorTypeConstants.ERROR,
46                              HttpStatus.NOT_FOUND,
47                              PRODUCT_DOES_NOT_EXIST)
48                      );
49
50                  if (!productRepository
51                          .isProductQuantityAvailable(
52                              cartItem.getIdProduct(),
53                              cartItem.getQuantity()
54                          )
55                  ) {
56                      throw new BadRequestException(
57                          ErrorTypeConstants.ERROR,
58                          HttpStatus.BAD_REQUEST,
59                          String.format(
60                              PRODUCT_IS_NOT_AVAILABLE_
61                              IN_SPECIFIED_QUANTITY,
62                              product.getName(),
63                              product.getModel()
64                          )
65                      );
66                  }
67              });
68      }
69  }
```

6.4.6.3. Servis

Među populacijom razvojnih inženjera koja se bavi arhitekturom aplikativnih sustava vodi se diskusija na temu položaja servisa unutar strukture složenih aplikacija te se postavlja vječito pitanje:

„Poziva li komponenta kontrolera metode servisa ili metode fasade koje zatim pozivaju metode odgovarajućih servisa?“

U praksi se isprepleteno koriste oba pristupa. Velik broj razvojnih inženjera koristi fasadu ukoliko je poslovna logika dohvaćanja ili modifikacije podataka vezana uz poveći broj drugih servisa, validatora, repozitorija i slično. U ostalim slučajevima, komponenta kontrolera prosljeđuje zahtjev odgovarajućoj servisnoj komponenti. Servis je komponenta koja se koristi za odvajanje poslovne logike dohvaćanja ili modifikacije podataka izvan koda samih kontrolera. Klasa servisa dekorira se anotacijom `@Service` ili `@Component`. Anotacija `@Component` generički je stereotip koji se koristi za definiranje komponenti koje su upravljane od strane Spring kontejnera. S druge pak strane, anotacija `@Service` ima istu ulogu kao i anotacija `@Component` uz razliku da pruža dodatne metapodatke.

Klasa `OrderServiceImpl` implementira poslovnu logiku dohvaćanja i modifikacije podataka vezanih uz narudžbe.

```
 1  @Slf4j
 2  @Service
 3  @RequiredArgsConstructor
 4  public class OrderServiceImpl implements OrderService {
 5
 6      private static final String PRIMARY_SORT_COLUMN =
 7          "createdTimestamp";
 8
 9      private final OrderValidator orderValidator;
10
11      private final ProductService productService;
12      private final NotificationService notificationService;
13
14      private final OrderRepository orderRepository;
15      private final OrderCustomRepository orderCustomRepository;
16
17      private final OrderMapper orderMapper;
18
19      @Override
20      @Transactional(readOnly = true)
21      public OrderPage getAllOrdersForUser(
22      Integer page, Integer size,  Long idUser) {
23          log.info("Entered getAllOrdersForUser in OrderServiceImpl with
24          page {}, size {}, idUser {}.", page, size, idUser);
25
26          PageRequest pageable = PageRequest.of(
27              page - 1,
28              size,
```

```
29              Sort.by(Sort.Direction.ASC, PRIMARY_SORT_COLUMN)
30          );
31
32          return orderCustomRepository.findAllByPageable(
33              pageable,
34              idUser
35          );
36      }
37
38      @Override
39      @Transactional
40      public synchronized void createOrder(OrderForm orderForm) {
41          log.info("Entered createOrder in OrderServiceImpl.");
42
43          orderValidator.validateOrderForm(orderForm);
44
45          Order order =
46              orderRepository.saveAndFlush(
47                  orderMapper.mapToEntity(orderForm)
48              );
49
50          order
51              .getProducts()
52              .forEach(
53                  orderProduct -> productService
54                      .updateProductQuantity(
55                          orderProduct
56                              .getProduct()
57                              .getIdProduct(),
58                  orderProduct
59                      .getProduct().getAvailableQuantity() -
60                  orderProduct.getQuantity()));
61
62          AsyncExecutor.executeAfterTransactionCommits(() ->
63              notificationService
64                  .createNotificationForSuccessfullyCreatedOrder(
65                      orderForm.getIdUser(),
66                      orderForm.getDeliveryType()
67                  )
68          );
69      }
70  }
```

6.4.6.4. Repozitorij

Repozitorij je komponenta aplikacije koja omogućava dohvaćanje te perzistenciju podataka. Anotacijom `@Repository` dekorira se klasa koja pruža mehanizam pretraživanja, dohvaćanja, modifikacije, kreiranja te brisanja entiteta iz baze podataka. Repozitorij uobičajeno proširuje `JPARepository` i/ili `QuerydslPredicateExecutor` sučelja koja pružaju implementacije osnovnih operacija nad entitetima.

Klasa `ProductRepository` pruža operacije koje omogućuju dohvaćanje te manipulaciju nad podacima baze podataka.

```
1  @Repository
2  public interface ProductRepository extends
3      JpaRepository<Product, Long>,
4      QuerydslPredicateExecutor<Product> {
5
6      List<Product> findProductsByTypeAndName(
7          Short productType,
8          String name
9      );
10
11     Optional<Boolean> existsByIdProduct(Long idProduct);
12
13     @Query(value =
14         "SELECT (count(product.*) > 0) FROM Product product " +
15         "WHERE product.id_product =:idProduct " +
16             "AND product.available_quantity >=:quantity",
17         nativeQuery = true)
18     boolean isProductQuantityAvailable(
19         Long idProduct,
20         Short quantity
21     );
22
23     @Modifying
24     @Query(value =
25         "UPDATE Product SET available_quantity =:availableQuantity " +
26         "WHERE id_product =:idProduct",
27         nativeQuery = true)
28     void updateQuantityOfProduct(
29         Long idProduct,
30         Integer availableQuantity
31     );
32 }
```

6.4.6.5. Mapper

Mapper je komponenta koja omogućava, ako što i samo ime govori, mapiranje podataka. Njezina primarna zadaća je konverzija podataka između različitih POJO-a te entiteta. Primjerice, *mapper* komponenta može se koristiti za mapiranje podataka entiteta na korespondentni DTO (engl. *Data Transfer Object*). Metode mapiranja podataka implementiraju se u sklopu pojedinog POJO-a ili se izdvajaju u metode zasebne *mapper* klase.

```
1  @Component
2  @RequiredArgsConstructor
3  public class ProductMapper {
4
5      private final ProductColorMapper productColorMapper;
6      private final ProductTypeMapper productTypeMapper;
7
8      public ProductDto mapToDto(Product product) {
9          ProductDto productDto = new ProductDto();
10
11         productDto.setIdProduct(product.getIdProduct());
12         productDto.setName(product.getName());
13         productDto.setDescription(product.getDescription());
14         productDto.setModel(product.getModel());
15         productDto.setAssistSpeed(product.getAssistSpeed());
16         productDto.setBatteryRange(product.getBatteryRange());
17         productDto.setChargingTime(product.getChargingTime());
18         productDto.setWeight(product.getWeight());
19         productDto.setPrice(product.getPrice());
20         productDto.setColor(
21             productColorMapper
22                 .mapToDto(
23                     ProductColor
24                         .findByIdProductColor(product.getColor())
25                 )
26         );
27         productDto.setAvailableQuantity(product.getAvailableQuantity());
28         productDto.setForRent(product.getForRent());
29         productDto.setRentPricePerHour(product.getRentPricePerHour());
30         productDto.setImagePath(product.getImagePath());
31         productDto.setType(
32             productTypeMapper
33                 .mapToDto(
34                     ProductType
35                         .findByIdProductType(product.getType())
36                 )
37         );
38         return productDto;
39     }
40
41     public List<ProductDto> mapToDtos(List<Product> products) {
42         return products
43             .stream()
44             .map(this::mapToDto)
45             .collect(Collectors.toList());
46     }
47 }
```

6.4.7. WebSocket poslužitelj

WebSocket poslužitelj implementiran je kao zasebna aplikacija s primarnom funkcijom osiguranja komunikacije u realnom vremenu između poslužiteljskog i klijentskog dijela aplikacije. Komunikacija u realnom vremenu temelji se na akcijama koje rezultiraju slanjem *backend-to-backend* zahtjeva iz aplikacijskog programskog sučelja prema WebSocket poslužitelju koji zatim prosljeđuje primljenu poruku odgovarajućim aktivnim sjednicama. Kako je WebSocket protokol prilično niske razine apstrakcije, koji definira transformaciju toka bajtova u odgovarajuće *frame-ove*, implementacija složenijih aplikacija je vrlo zahtjevna. Srećom, Spring Boot dopušta korištenje potprotokola koji radi na višoj razini apstrakcije, odnosno aplikacijskoj razini. U tu svrhu bit će korišten STOMP (engl. *Simple Text Oriented Messaging Protocol*) kao jedan u uobičajenih WebSocket potprotokola za uspostavu komunikacije u realnom vremenu.

STOMP je tekstualno orijentiran protokol koji podržava razmjenu poruka između klijenata i poslužitelja, odnosno brokera, koji su razvijeni u različitim programskim jezicima. STOMP definira pregršt tipova *frame-ova* koji se mapiraju na odgovarajuće WebSocket *frame-ove* – `CONNECT`, `SUBSCRIBE`, `UNSUBSCRIBE`, `ACK` te `SEND`. S jedne strane, ove naredbe su vrlo praktične za upravljanje komunikacijom, dok nam, s druge strane, omogućuju implementaciju rješenja sa sofisticiranijim značajkama među kojima i je potvrda prijema poruke (ACK). [111]

Kako bi omogućili komunikaciju u realnom vremenu putem WebSocket protokola potrebno je definirati klasu dekoriranu anotacijom `@EnableWebSocketMessageBroker`. Kao i sam naziv anotacije sugerira, WebSocketMessageBroker zadužen je za rukovanje WebSocket porukama te njihovo jednostavno te jednoznačno usmjeravanje. Sučelje `WebSocketMessageBrokerConfigurer` definira metode koje omogućavaju konfiguraciju rukovanja porukama koje se temelje na STOMP-u, kao jednostavnom protokolu za slanje poruka.

```
1  @Configuration
2  @EnableWebSocketMessageBroker
3  public class WebSocketConfiguration implements
4   WebSocketMessageBrokerConfigurer {
5
6     @Override
7     public void registerStompEndpoints(StompEndpointRegistry registry) {
8         registry
9             .addEndpoint("/ws")
10            .setAllowedOriginPatterns("*")
11            .setHandshakeHandler(new UserHandshakeHandler())
```

```
            .withSockJS()
            .setDisconnectDelay(60000);
    }
    @Override
    public void configureMessageBroker(MessageBrokerRegistry registry) {
        ThreadPoolTaskScheduler taskScheduler = new
            ThreadPoolTaskScheduler();
        taskScheduler.setPoolSize(1);
        taskScheduler.setThreadNamePrefix("wss-heartbeat-thread-");
        taskScheduler.initialize();

        registry.enableSimpleBroker("/queue")
            .setHeartbeatValue(new long[]{25000, 25000})
            .setTaskScheduler(taskScheduler);

        registry.setApplicationDestinationPrefixes("/app");
    }
}
```

Klijent se na WebSocket poslužitelj spaja postupkom koji je poznat pod nazivom *TCP rukovanje*, poznatije kao *TCP Handshake*, te mu, zajedno sa zahtjevom, dostavlja ID korisnika koji želi ostvariti dvosmjernu komunikaciju. Proslijeđeni korisnički ID omogućuje WebSocket-u da odredi korisničku sjednicu kojoj je potrebno proslijediti poruku. Naime, standardna implementacija `HandshakeHandler` klase pohranjuje `uuid`. Takav pristup nije dovoljno fleksibilan prilikom implementacije funkcionalnosti obavijesti pošto se pojedina obavijest veže na korisnički ID.

```
@Slf4j
public class UserHandshakeHandler extends DefaultHandshakeHandler {

    public static final String ID_USER_PARAM = "idUser";

    @Override
    protected Principal determineUser(
        ServerHttpRequest request,
        WebSocketHandler wsHandler,
        Map<String, Object> attributes
    ) {
        String userId = ((ServletServerHttpRequest) request)
            .getServletRequest()
            .getParameter(ID_USER_PARAM);

        log.info("User with id {} successfully logged in.", userId);

        return new UserPrincipal(userId);
    }
}
```

Zahtjevi pristigli na REST API WebSocket poslužitelja prosljeđuju se prema zahtijevanom klijentu te odgovarajućoj temi. Temeljni koncepti rada WebSocket poslužitelja zasniva se na

primitku poruke na odgovarajuću krajnju točku. Primitak poruke predstavlja akciju koja zahtijeva slanje poruke ili prema svim aktivnim sjednicama (engl. *broadcast*) unutar određene teme ili pak prema pojedinačnom korisniku. Ranije je spomenuto da se prilikom povezivanja na WebSocket poslužitelj, točnije prilikom procesa TCP rukovanja, dostavlja ID korisnika koji inicira kreiranje dvosmjerne komunikacije s WebSocket poslužiteljem. Na taj je način omogućena distinkcija pojedinom prijavljenog korisnika te prosljeđivanje poruka odgovarajućim sjednicama.

```
1  @Slf4j
2  @RestController
3  @RequestMapping("/ws/notifications")
4  @RequiredArgsConstructor
5  public class NotificationController {
6
7      private final RequestValidator requestValidator;
8
9      private final SimpMessagingTemplate simpMessagingTemplate;
10
11     @PostMapping("/update")
12     public ResponseEntity<?> updateNotificationsCount(
13         @RequestHeader(value = "Authorization") String token,
14         @RequestParam Long idUser
15     ) {
16
17         if (!requestValidator.validatePrivilegedToken(token)) {
18             log.error("Token {} is not valid.", token);
19
20             return new ResponseEntity<>(HttpStatus.UNAUTHORIZED);
21         }
22
23         simpMessagingTemplate
24             .convertAndSendToUser(
25                 String.valueOf(idUser),
26                 "/queue/topic/private-notifications",
27                 Boolean.TRUE
28             );
29
30         return new ResponseEntity<>(HttpStatus.OK);
31     }
32 }
```

6.5. Razvoj na strani klijenta

Klijentski dio Web aplikacije razvijen je korištenjem programskog jezika JavaScript, točnije programskog okvira Next.js kao React razvojnog okvira koji olakšava rapidni razvoj aplikacija. Centralni dio klijentskog dijela aplikacije čini vlastito implementirana biblioteka komponenata kojom se promiče načelo ponovne iskoristivosti programskoga koda (engl. *code reusability*).

6.5.1. Inicijalizacija projekta

Projekt je inicijaliziran korištenjem `npx` CLI (engl. *command-line interface*) alata koji omogućava kreiranje inicijalne strukture projekta zajedno sa osnovnim paketima koji su nužni razvoj Web aplikacije. Kako bi se kreirao novi projekt potrebno je izvršiti jednu od komandi:

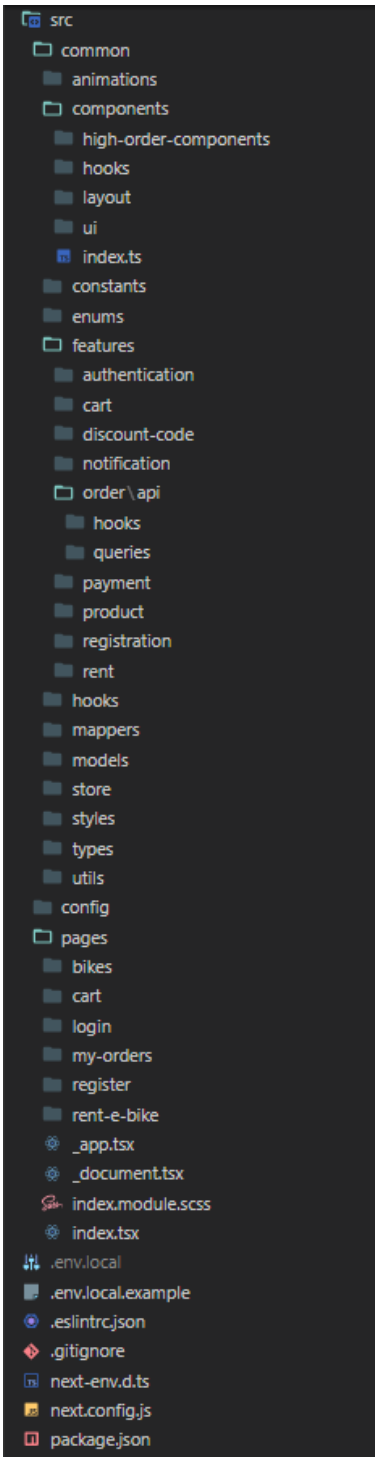
`npx create-next-app@latest –ts` ili `yarn create next-app --typescript`

```
C:\Users\Patrik\Desktop\next-test>yarn create next-app --typescript
yarn create v1.22.17
[1/4] Resolving packages...
[2/4] Fetching packages...
[3/4] Linking dependencies...
[4/4] Building fresh packages...

warning Your current version of Yarn is out of date. The latest version is "1.22.19", while you're on "1.22.17".
info To upgrade, run the following command:
$ curl --compressed -o- -L https://yarnpkg.com/install.sh | bash
success Installed "create-next-app@12.2.5" with binaries:
      - create-next-app
√ What is your project named? ... chainreaction
Creating a new Next.js app in C:\Users\Patrik\Desktop\next-test\chainreaction.

Using yarn.

Installing dependencies:
- react
- react-dom
- next

yarn add v1.22.17
info No lockfile found.
[1/4] Resolving packages...
[2/4] Fetching packages...
[3/4] Linking dependencies...
[4/4] Building fresh packages...
success Saved lockfile.
success Saved 16 new dependencies.
info Direct dependencies
├─ next@12.2.5
├─ react-dom@18.2.0
└─ react@18.2.0
```

Slika 53. Inicijalizacija Next.js projekta

6.5.2. Struktura projekta

Struktura direktorija na klijentskoj strani aplikacije prati suvremene trendove razvoja aplikacija te je osmišljena s ciljem pridržavanja *clean code* koncepata te jednostavnog održavanja sustava u kasnijim fazama razvoja projekta. `_app.tsx` primarna je klasa Next.js projekta čija je glavna namjena učitavanje odgovarajućih komponenti (stranica) ovisno o trenutnom URL-u, odnosno implementacija React Router-a u pozadini.



Slika 54. Struktura klijentskog dijela aplikacije

6.5.3. Biblioteka komponenata

Umjesto oslanjanja na već postojeće biblioteke komponenata, kao što su *Reactstrap*, *MaterialUI* ili *PrimeReact*, razvijena je vlastita biblioteka komponenata. Primarni cilj biblioteke je omogućiti ponovnu iskoristivost programskog koda, odnosno komponenata. Svaka od komponenata implementirana je na način da dozvoljava nadogradnju i prilagodbe. Komponente su stilizirane primjenom *Tailwind CSS* programskog okvira koji definira širok spektar klasa koje ubrzavaju proces razvoja samih komponenata. Biblioteka se neprestano razvija te proširuje novim komponentama ovisno o potrebama samog projekta. Velik broj razvojnih inženjera prilikom razvoja mobilnih i Web aplikacija koristi već razvijene biblioteke komponenata koje na taj način pružaju podršku rapidnom razvoju aplikativnih sustava.

Komponenta `Radio` nadograđuje `<input type='radio' />` element proširujući ga dodatnim funkcionalnostima koje sirovi HTML element ne predviđa niti omogućuje.

```
interface RadioProps {
  className?: string;
  helper?: string;
  helperText?: string;
  isChecked: boolean;
  isDisabled?: boolean;
  onChange: any;
}

const Radio = ({
  className,
  helper,
  helperText,
  isChecked,
  isDisabled = false,
  onChange,
}: RadioProps) => {
  return (
    <div className={`${className} flex`}>
      <div className="flex items-center h-5">
        <input
          id="helper-radio"
          aria-describedby="helper-radio-text"
          type="radio"
          disabled={isDisabled}
          className={declassify(
            "w-4 h-4 text-blue-600 bg-gray-100 border-gray-300
            focus:ring-blue-500 dark:focus:ring-blue-600
            dark:ring-offset-gray-800 focus:ring-2 dark:bg-gray-700
            dark:border-gray-600",
            { "cursor-not-allowed": isDisabled }
          )}
          checked={isChecked}
          onChange={() => onChange()}
        />
      </div>
```

```
37        <div className="ml-2 text-sm">
38          <label
39            htmlFor="helper-radio"
40            className={declassify(
41              "font-medium",
42              {
43                "text-gray-400 dark:text-gray-400": isDisabled,
44              },
45              { "text-gray-100 dark:text-gray-100": !isDisabled }
46            )}
47          >
48            {helper}
49          </label>
50          <p
51            id="helper-radio-text"
52            className="text-xs font-normal text-gray-400
53                       dark:text-gray-300"
54          >
55            {helperText}
56          </p>
57        </div>
58      </div>
59    );
60  };
61
62  export default Radio;
```

S ciljem pojednostavljena uvjetovane primjene klasa razvijene su *classify* te *declassify* metode koje primjenjuju pojedinu klasu u ovisnosti od istinitosti postavljenog uvjeta.

```
1   export function classify(...classes: Classifiable[]): ClassesAsArray {
2     const result = classes
3       .map((value) => {
4         if (isString(value)) {
5           return value.split(" ");
6         } else if (isArray(value)) {
7           return value
8             .map((className) => classify(className))
9             .reduce(arrayReducer, []);
10        }
11        return Object.keys(value)
12          .map((key) => (value[key] ? classify(key) : []))
13          .filter((value) => value.length > 0)
14          .reduce(arrayReducer, []);
15      })
16      .reduce(arrayReducer, []);
17    return noDupes(noEmpty(result));
18    function arrayReducer(array: string[], item: string[]) {
19      return [...array, ...item];
20    }
21  }
22
23  export function declassify(...classes: Classifiable[]): string {
24    return classify(...classes).join(" ");
```

```
25 }
```

6.5.4. Komunikacija s REST API poslužiteljem

Kako bi se olakšala komunikacija s REST API poslužiteljem kreirana je instanca `Axios` klase koja je dio istoimenog paketa. `Axios` paket implementira HTTP klijenta koji omogućava jednostavno kreiranje `XMLHttpRequests` zahtjeva. `Axios` omogućava definiranje presretača (engl. *interceptor*) za zahtjeve i odgovore čime omogućuje dodatne modifikacije zahtjeva i odgovora prije stizanja na odredište. Osim toga, `Axios` automatski transformira podatke odgovora u JSON format kako bi uspostavio razumijevanje između klijenta i poslužitelja.

```
1  import axios from "axios";
2
3  import setupAxiosInterceptors from "@config/axios-interceptor";
4
5  let instance = axios.create();
6
7  const TIMEOUT = 1 * 60 * 1000;
8  instance.defaults.timeout = TIMEOUT;
9  instance.defaults.baseURL = process.env.NEXT_PUBLIC_API_URL;
10
11 export default setupAxiosInterceptors(instance);
```

Samo upravljanje stanjem zahtjeva kao i pristiglim podacima implementirano je korištenjem `React Query` biblioteke. `React Query` bitno olakšava upravljanje stanjem zahtjeva te predmemorijom (engl. *cache*) kao i sinkronizaciju te ažuriranje stanja. Operacije dohvaćanja i kreiranja podataka omogućene su pomoću dviju *hook* metoda – `useQuery` te `useMutation`.

Hook metoda `useFetchOrdersByIdUser` koristi se za dohvaćanje korisničkih narudžbi.

```
1  const fetchOrdersByIdUser = (idUser: number, pagination: Pagination) =>
2  {
3    return async () =>
4      await axios.get<OrderPage>(
5       `/orders?page=${pagination.page}&size=${pagination.size}&
6       idUser=${idUser}`
7      );
8  };
```

```
1  import { useMutation } from "react-query";
2
3  import { fetchOrdersByIdUser } from "../queries";
4
5  import { FETCH_ORDERS_BY_ID_USER } from "../queries/constants";
6
7  const useFetchOrdersByIdUser = (idUser: number): any => {
8    return useMutation(FETCH_ORDERS_BY_ID_USER, (data: any) => {
9      return fetchOrdersByIdUser(idUser, data?.pagination)();
10   });
```

```
11 };
12
13 export default useFetchOrdersByIdUser;
```

6.5.5. Komunikacija s WebSocket poslužiteljem

Poslovna logika vezana uz povezivanje te komunikaciju klijenta s WebSocket poslužiteljem odvojena je u zasebnu datoteku koja sadrži sve potrebne metode. Na taj je način omogućena vrlo jednostavna nadogradnja te prilagodba same komunikacije između klijenta te WebSocket poslužitelja. Nakon uspješne prijave vrši se povezivanje na WebSocket poslužitelj te pretplata na *private-notifications* temu (engl. *topic*). Sama instanca objekta klase `SockJS`, koja održava vezu s WebSocket poslužiteljem, implementirana je primjenom *Singleton* uzorka dizajna kako bi se osigurala jedinstvenost veze između klijenta i WebSocket poslužitelja.

```
1  let subscriber: Subscriber<any>;
2  let connection: Promise<any>;
3  let connectedPromise: any = null;
4  let listener: Observable<any>;
5  let listenerObserver: Subscriber<any>;
6  let alreadyConnectedOnce = false;
7
8  export let stompClient: any;
9
10 export const PRIVATE_NOTIFICATION_TOPIC =
11  "/user/queue/topic/private-notifications";
12
13 const createConnection = (): Promise<any> =>
14   new Promise((resolve) => (connectedPromise = resolve));
15
16 const createListener = (): Observable<any> => {
17   return new Observable((observer) => {
18     listenerObserver = observer;
19   });
20 };
21
22 export const createStompClient = (idUser: number) => {
23   const socket = new SockJS(
24     process.env.NEXT_PUBLIC_WEB_SOCKET_URL + "?idUser=" + idUser
25   );
26
27   return Stomp.over(socket);
28 };
29
30 export const connect = (idUser: number) => {
31   if (connectedPromise !== null || alreadyConnectedOnce) {
32     return;
33   }
34
35   connection = createConnection();
36   listener = createListener();
37
38   stompClient = createStompClient(idUser);
```

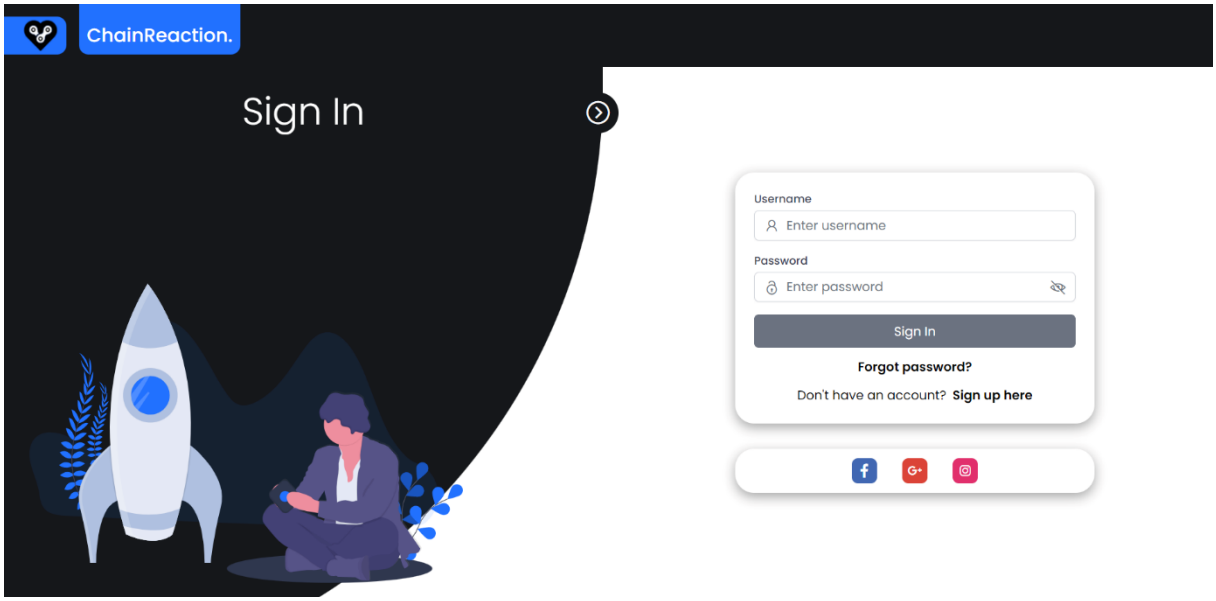
```
39    stompClient.debug = null;
40    stompClient.connect({}, () => {
41      connectedPromise("success");
42      connectedPromise = null;
43      alreadyConnectedOnce = true;
44      subscribe(PRIVATE_NOTIFICATION_TOPIC);
45    });
46
47    return stompClient;
48  };
49
50  export const receiveWebSocketMessage = () => listener;
51
52  export const subscribe = (topic: string) => {
53    listener = createListener();
54    connection.then(() => {
55      subscriber = stompClient.subscribe(topic, (payload: any) => {
56        listenerObserver?.next(JSON.parse(payload.body));
57      });
58    });
59  };
60
61  export const disconnect = () => {
62    if (stompClient !== null) {
63      if (stompClient.connected) {
64        stompClient.disconnect();
65      }
66      stompClient = null;
67    }
68    alreadyConnectedOnce = false;
69  };
70
71  export const unsubscribe = () => {
72    if (subscriber !== null) {
73      subscriber.unsubscribe();
74    }
75  };
```

6.5.6. Funkcionalnosti aplikacije

Skup trenutno razvijenih funkcionalnosti aplikacije obuhvaća *prijavu*, *registraciju*, *pregled i pretraživanje artikala*, *pregled detalja artikla*, *korisničku košaricu*, *obavijesti u aplikaciji*, *iznajmljivanje bicikla*, *kreiranje narudžbe* te *pregled kreiranih narudžbi*. Kako je sam projekt u fazi razvoja trenutno razvijeni skup funkcionalnosti ulazi u inicijalni *scope* projekta dok će se ostale funkcionalnost razvijati nakon što aplikacije prijeđe na produkcijsku okolinu te se krene s njezinim korištenjem.

6.5.6.1. Prijava i registracija

Krajnji korisnici trenutno imaju mogućnost prijave u Web aplikaciju putem korisničkog imena i lozinke. Prijava putem društvenih mreža biti će implementirana u kasnijim fazama razvoja projekta. Autentifikacija korisnika obavlja se putem krajnje točke `/api/authentication` kreiranjem korespondentnog HTTP zahtjeva POST metodom.



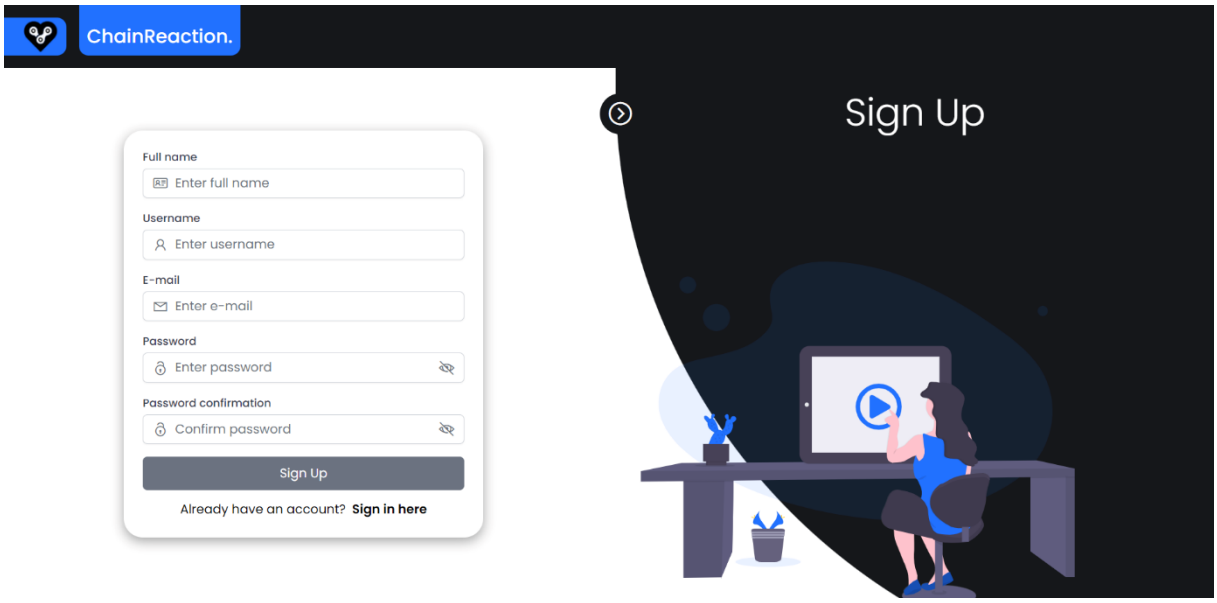
Slika 55. Zaslon prijave u Web aplikaciju

`useMutation` *hook* koji se koristi za rukovanje autentifikacijskim zahtjevom.

```
export const authenticateUser = ({ username, password }: User) => {
  return async () =>
    await axios.post<JwtToken>("/authentication", { username, password
 });
};
```

```
const useAuthenticateUser = (user: User): any => {
  return useMutation(AUTHENTICATE_USER, authenticateUser(user));
};

export default useAuthenticateUser;
```

Krajnjim korisnicima je omogućeno i kreiranje korisničkog računa putem forme za registraciju. Registracija, odnosno kreiranje korisničkog računa, obavlja se putem krajnje točke `/api/users` kreiranjem korespondentnog HTTP zahtjeva POST metodom.



Slika 56. Zaslon registracije u Web aplikaciju

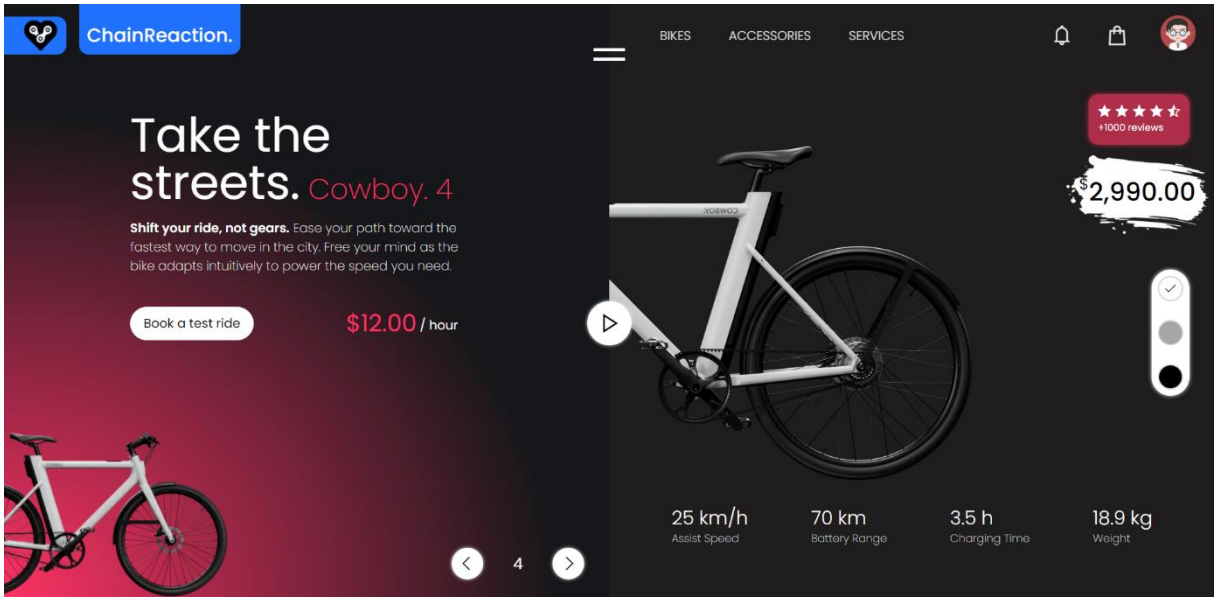
`useMutation` *hook* koji se koristi za rukovanje zahtjevom za kreiranje novog korisnika.

```
export const registerUser = ({ fullname, username, email, password }:
User) => {
  return async () =>
    await axios.post<void>("/users", { fullname, username, email,
    password });
};
```

```
const useRegisterUser = (user: User): any => {
  return useMutation(REGISTER_USER, registerUser(user));
};

export default useRegisterUser;
```

6.5.6.2. Početna stranica

Autentificirani korisnici imaju mogućnost pregleda početne stranice koja prikazuje aktualne Cowboy električne bicikle zajedno s osnovnim informacijama o samom biciklu te cijenom. Klikom na gumb "Book a test ride" omogućena je rezervacija električnog bicikla. Dohvaćanje podataka obavlja se slanje HTTP zahtjeva GET metodom na `/api/products?productType&productName`.



Slika 57. Zaslon početne stranice

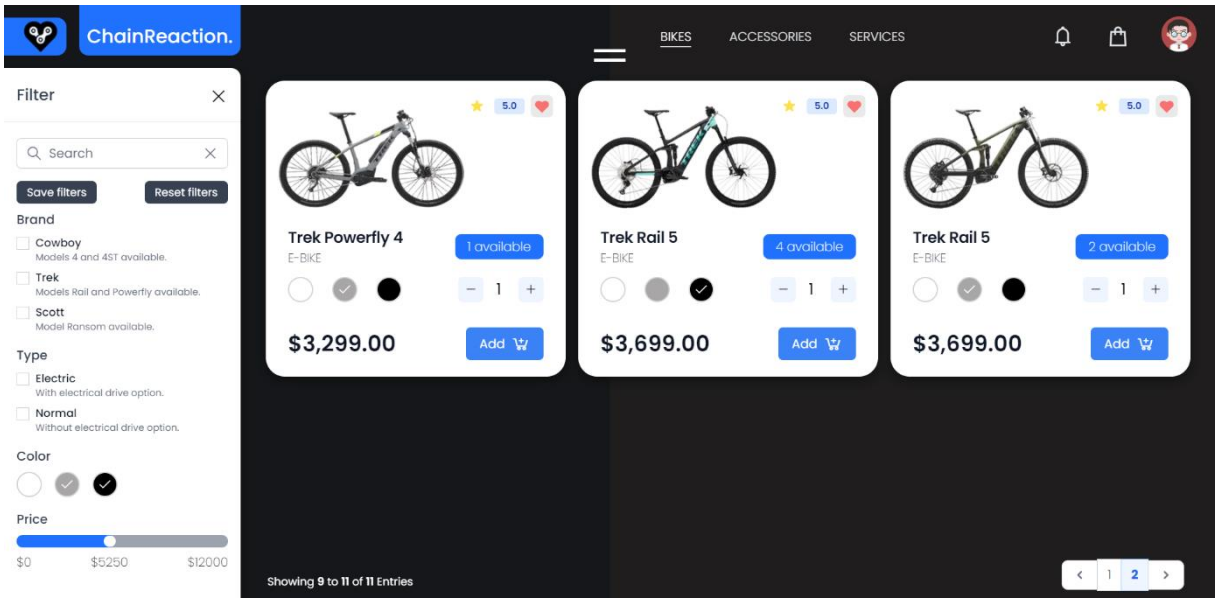
`useQuery` *hook* koji se koristi za dohvaćanje podataka o proizvodima koji se prikazuju na početnoj stranici.

```
export const fetchProductsByProductTypeAndProductName = (
   productType: string, productName?: string | null) => {
   return async () =>
     await axios.get<Product[]>(
`/products?productType=${productType}&productName=${productName}`
     );
 };
```

```
const useFetchProductsByProductTypeAndProductName = (
    productType: number, productName?: string): any => {
    return useQuery(
        FETCH_PRODUCTS_BY_PRODUCT_TYPE_AND_PRODUCT_NAME,
        fetchProductsByProductTypeAndProductName(
        getProductTypeValue(productType)!,productName||null
    ),
    {enabled: false,refetchOnWindowFocus: false}
  );
} ;

export default useFetchProductsByProductTypeAndProductName;
```

6.5.6.3. Pregled i pretraživanje artikala

Autentificirani korisnici imaju mogućnost pregleda kao i pretraživanja proizvoda prema vlastitim željama i preferencijama. Klikom na pojedini proizvod otvara se forma s detaljima odabranog proizvoda. Dohvaćanje podataka obavlja se slanje HTTP zahtjeva POST metodom na `/api/products/filter?page&size&productType`.



Slika 58. Zaslon pregleda artikala

`useMutation` *hook* koji se koristi za dohvaćanje podataka o proizvodima koji se prikazuju na stranici pregleda artikala.

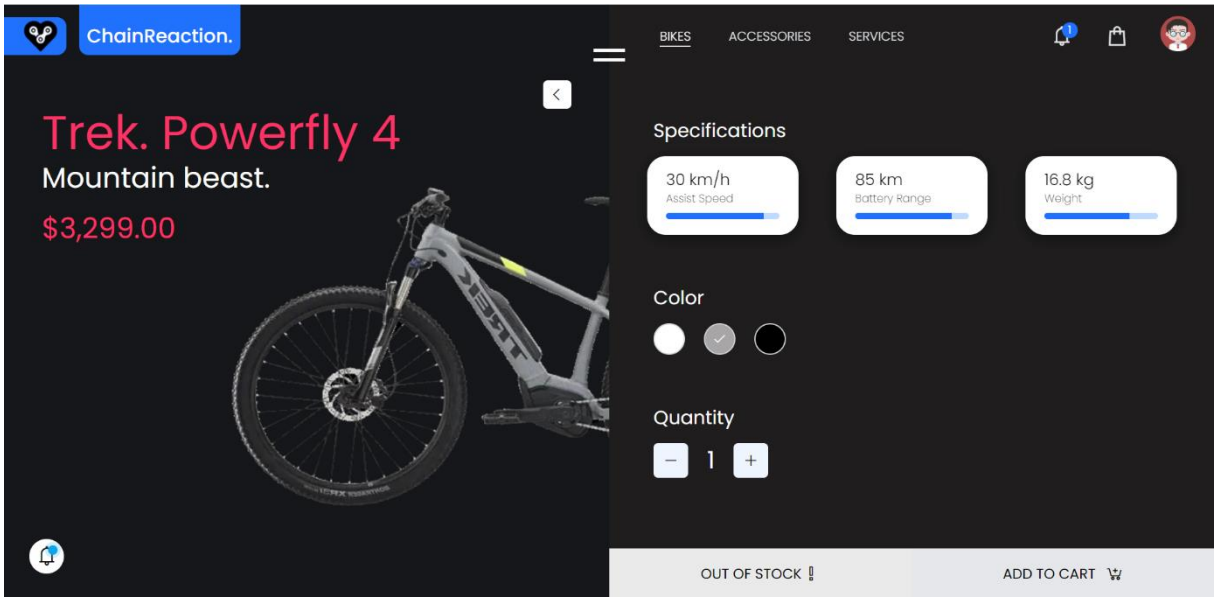
```
export const fetchProductsByFilter = (
  productTypes: number[], pagination: Pagination,
  filter: ProductFilter) => {
  return async () =>
    await axios.post<ProductPage>(
      `/products/filter?page=${pagination.page}&
        size=${pagination.size}&
        productTypes=${productTypes}`, filter
    );
};
```

```
const useFetchProductsByFilter = (productTypes: number[]): any =>
{
  return useMutation(FETCH_PRODUCTS_BY_FILTER,(data: any): any =>{
    return fetchProductsByFilter(
      productTypes,
      data?.pagination,
      data?.productFilter)();
  });
};
```

```
10
11  export default useFetchProductsByFilter;
```

6.5.6.4. Pregled detalja artikla

Klikom na karticu pojedinog proizvoda otvara se nova forma koja prikazuje detalje odabranog proizvoda. Forma za pregled detalja omogućava i dodavanje proizvoda u korisničku košaricu ukoliko je on trenutno dostupan. Ukoliko proizvod trenutno nije dostupan tada se korisnik može pretplatiti na proizvod kako bi bio obaviješten po dolasku novih proizvoda na stanje. Dohvaćanje podataka obavlja se slanje HTTP zahtjeva GET metodom na `/api/products/{idProduct}`.



Slika 59. Zaslon pregleda detalja artikla

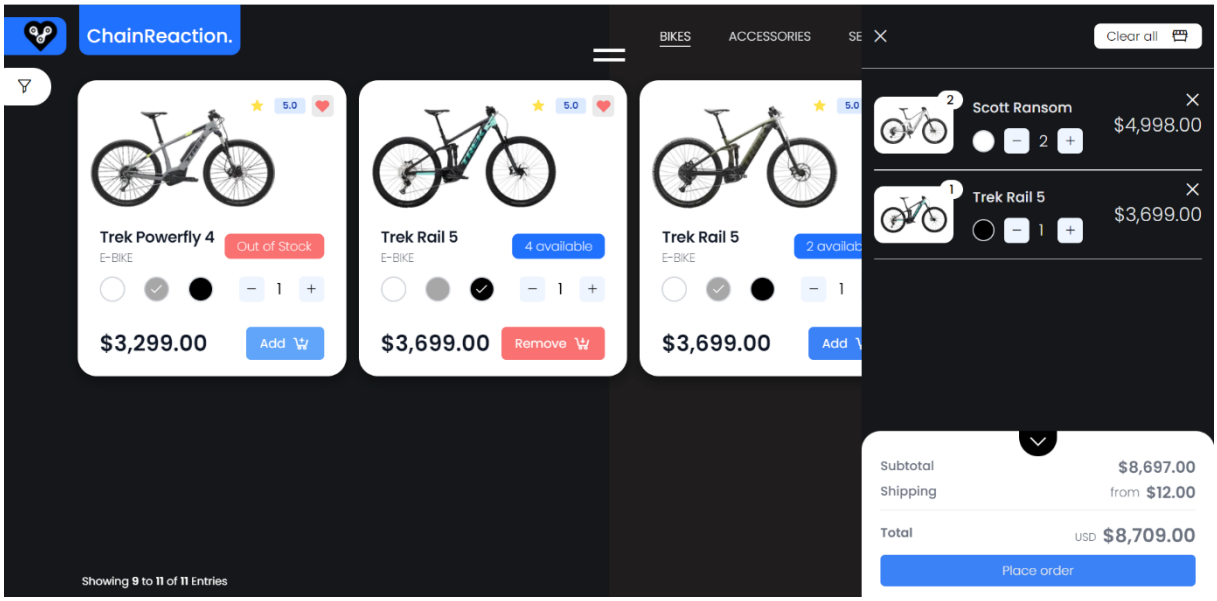
`useQuery` *hook* koji se koristi za dohvaćanje podataka o odabranom proizvodu.

```
1  export const fetchProductById = (idProduct: string) => {
2    return async () => await axios.get<Product>(`/products/${idProduct}`);
3  };
```

```
1  const useFetchProductById = (idProduct: string): any => {
2    return useQuery(FETCH_PRODUCT_BY_ID, fetchProductById(idProduct), {
3      enabled: false,
4      refetchOnWindowFocus: false,
5    });
6  };
7
8  export default useFetchProductById;
```

6.5.6.5. Korisnička košarica

Funkcionalnost korisničke košarice omogućava dodavanje proizvoda tijekom pregledavanja te njihovu sistematizaciju na jedinstvenom mjestu. Korisnička košarica sadrži popis svih proizvoda koji su dodani u nju zajedno s odabranim količinama. Osim pregledavanja dodanih proizvoda, korisnička košarica dopušta provođenje akcija vezanih uz ažuriranje količine odabranog proizvoda kao i brisanje odabranog proizvoda iz korisničke košarice. Dohvaćanje podataka obavlja se putem lokalnog spremišta (engl. *Local Storage*) Web preglednika koje omogućava njihovu perzistenciju te dohvaćanje prema odgovarajućem ključu.



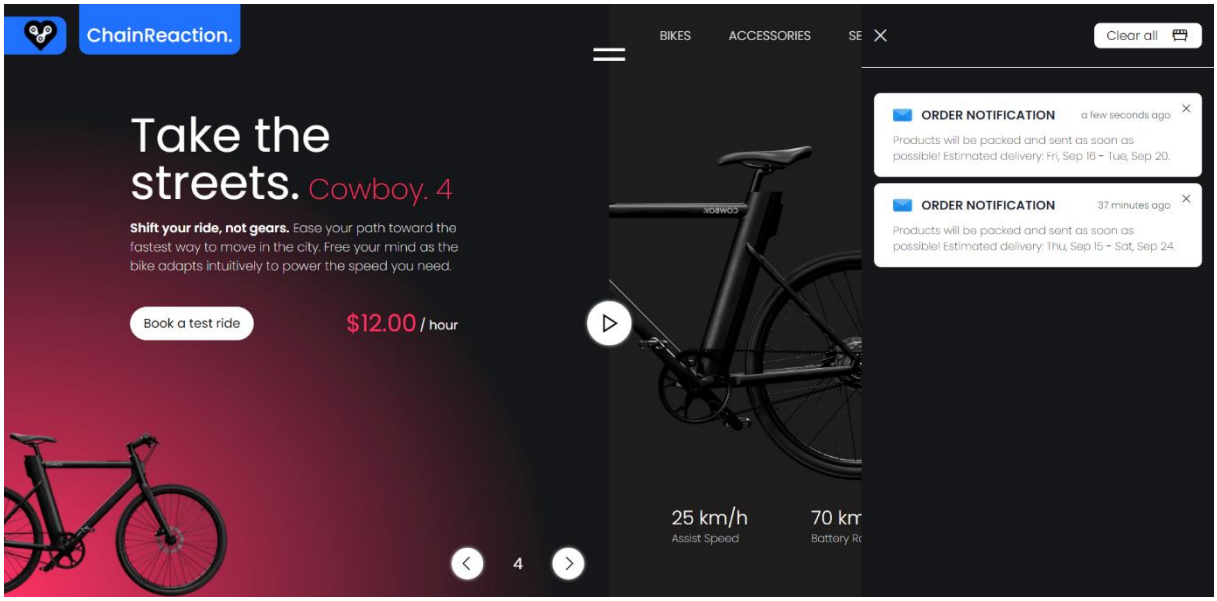
Slika 60. Okno za pregled detalja korisničke košarice

Pristupanje lokalnom spremištu Web preglednika izdvojeno je zasebnu datoteku koja sadrži sve potrebne metode za rukovanje lokalnim spremištem podataka.

```
1  import { LocalStorageKeys } from "@enums/local-storage-keys";
2
3  export const clearLocalStorage = () => {
4    Object.values(LocalStorageKeys).forEach((value) =>
5      localStorage.remove(value)
6    );
7  };
8
9  export const setValue = (key: string, value: string) => {
10   localStorage.setItem(key, value);
11 };
12
13 export const getValueByKey = (key: string): string | null => {
14   return localStorage.getItem(key) || null;
15 };
```

6.5.6.6. Obavijesti u aplikaciji

Izvršavanjem značajnijih akcija, kao što je kreiranje narudžbe ili rezervacija bicikla, od strane korisnika ili pak izvršavanjem značajnih akcija, kao što je promjena statusa narudžbe, od strane aplikacije kreira se odgovarajuća obavijest za korisnika. Obavijesti starije od 30 dana automatski se brišu iz baze podataka putem kreiranog *cron job-a*. Dohvaćanje podataka obavlja se slanje HTTP zahtjeva GET metodom na `/api/products/{idProduct}`.



Slika 61. Okno za pregled korisničkih obavijesti

`useQuery` *hook* koji se koristi za dohvaćanje podataka o korisničkim obavijestima.

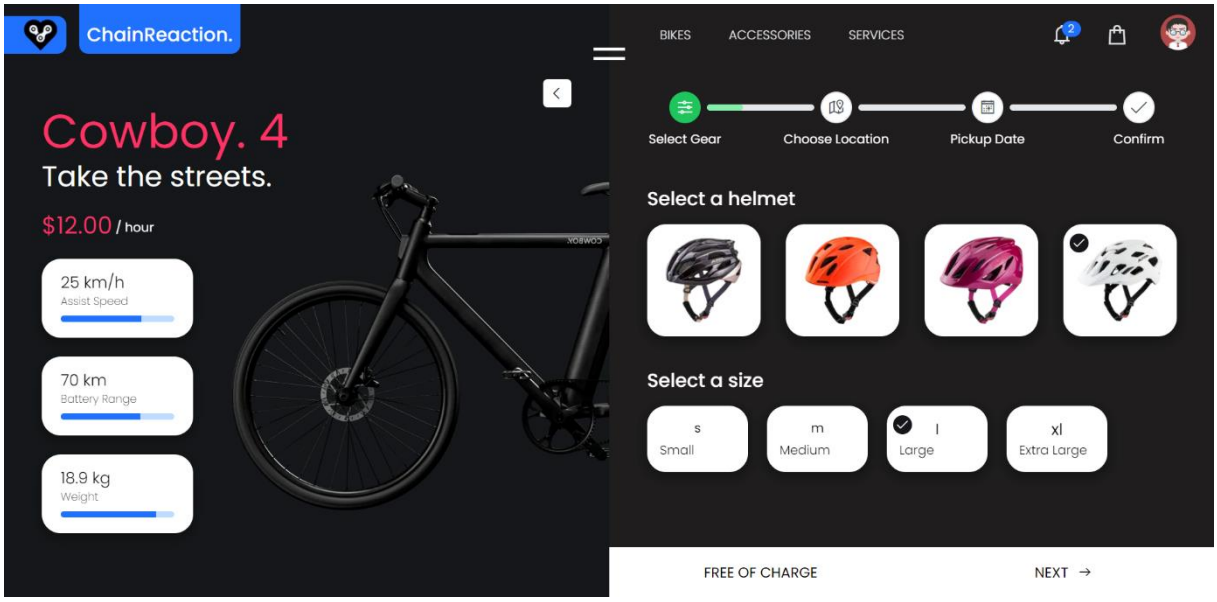
```
export const fetchNotifications = (idUser: number) => {
  return async () =>
    await axios.get<Notification[]>(`/notifications?idUser=${idUser}`);
};
```

```
const useFetchNotifications = (idUser: number): any => {
  return useQuery(FETCH_NOTIFICATIONS, fetchNotifications(idUser), {
    enabled: false,
    refetchOnWindowFocus: false,
  });
};

export default useFetchNotifications;
```

6.5.6.7. Rezervacija bicikla

Osim kupnje pojedinog bicikla, unutar aplikacije omogućeno je i rezerviranje bicikala na određenih vremenski period pri čemu je važno naglasiti kako se plaćanje kreirane rezervacije ne obavlja putem aplikacije već unutar same poslovnice. Kreiranje rezervacije obavlja se slanje HTTP zahtjeva POST metodom na `/api/rents`.

Rezervacija bicikla započinje odabirom primarne sigurnosne opreme.



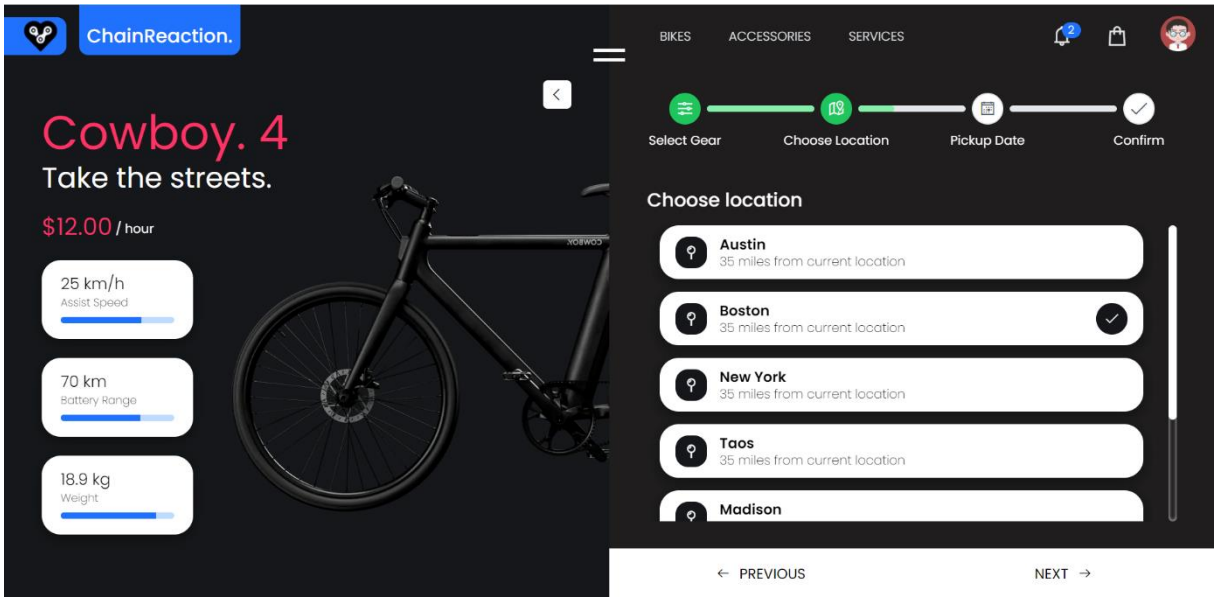
Slika 62. Zaslon za odabir opreme

`useQuery` *hook* koji se koristi za dohvaćanje podataka o sigurnosnoj opremi.

```
export const fetchProductsByProductTypeAndProductName = (
    productType: string, productName?: string | null) => {
    return async () => await axios.get<Product[]>(
`/products?productType=${productType}&productName=${productName}`
    );
};
```

```
const useFetchProductsByProductTypeAndProductName = (
    productType: number, productName?: string): any => {
    return useQuery(
        FETCH_PRODUCTS_BY_PRODUCT_TYPE_AND_PRODUCT_NAME,
        fetchProductsByProductTypeAndProductName(
        getProductTypeValue(productType)!,productName||null
    ),
    {enabled: false,refetchOnWindowFocus: false}
  );
};

export default useFetchProductsByProductTypeAndProductName;
```

Nakon odabira odgovarajuće sigurnosne opreme potrebno je odabrati rezervacijski centar.



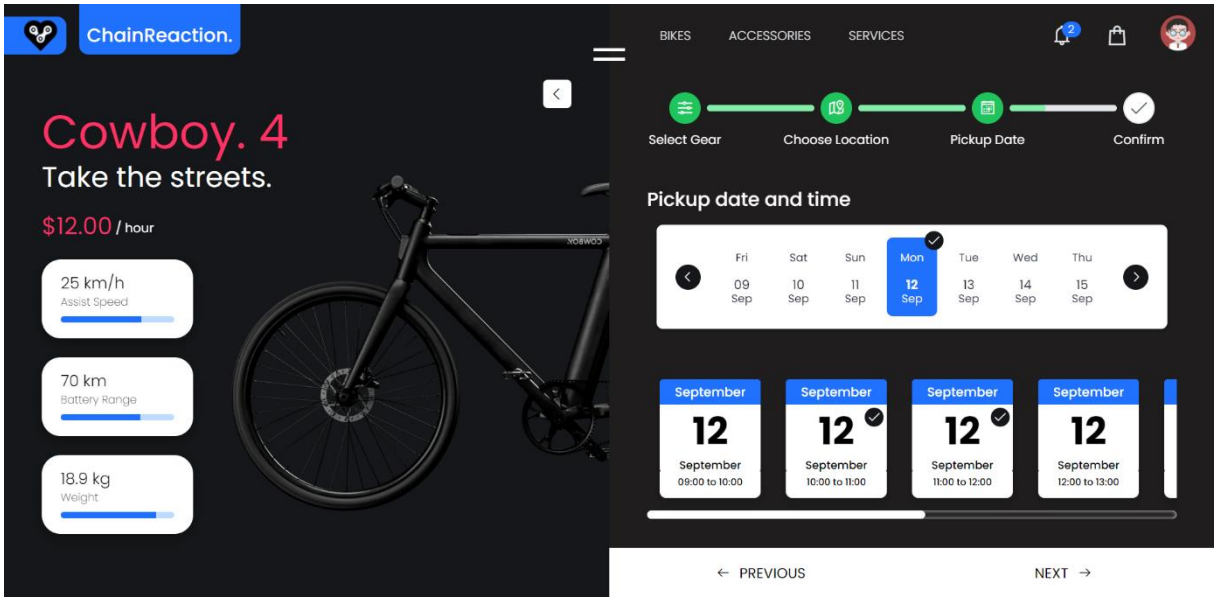
Slika 63. Zaslon za odabir rezervacijskog centra

`useQuery` *hook* koji se koristi za dohvaćanje podataka o rezervacijskim centrima.

```
export const fetchRentLocations = () => {
  return async () => await axios.get<string[]>("/rents/available-
  locations");
};
```

```
const useFetchRentLocations = (): any => {
  return useQuery(FETCH_RENT_LOCATIONS, fetchRentLocations(), {
    enabled: false,
    refetchOnWindowFocus: false,
  });
};

export default useFetchRentLocations;
```

Zatim je potrebno odabrati datum i vremenskog perioda rezervacije.



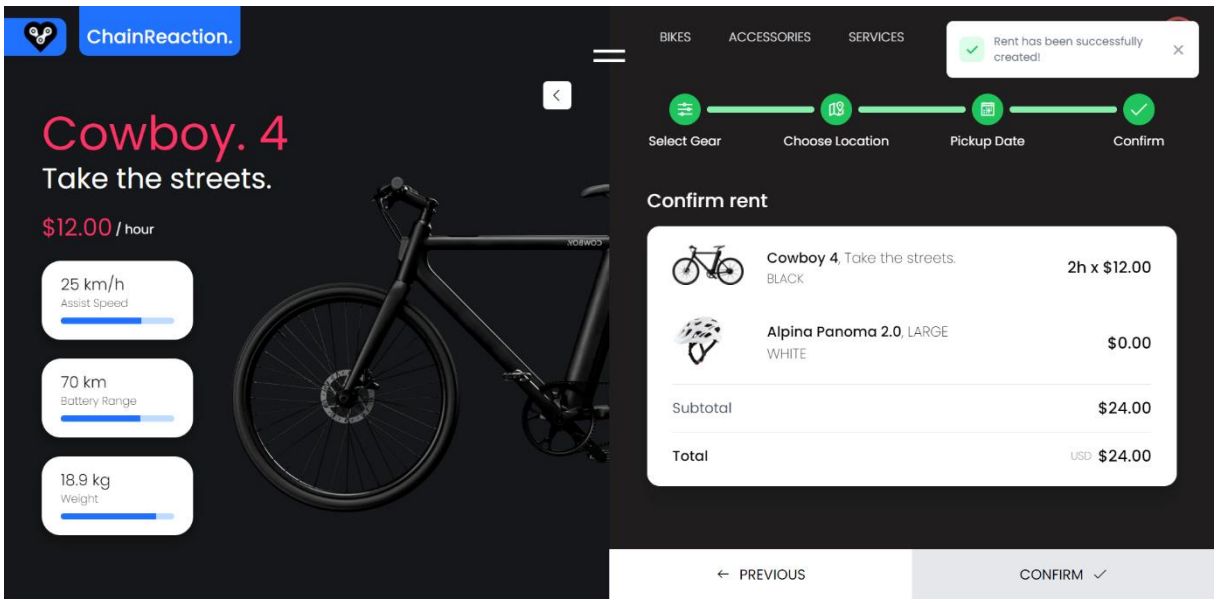
Slika 64. Zaslon za odabir datuma i vremenskog perioda rezervacije

`useQuery` *hook* koji se koristi za dohvaćanje podataka o dostupnim terminima rezervacije.

```
export const fetchAvailableTimeslots = (
  idProduct: string,
  location: string,
  date: Dayjs
)  => {
  return async () =>
    await axios.get<string[]>(
      `/rents/${idProduct}?idLocation=${location}&date=${date.format(
        APP_LOCAL_DATE_FORMAT
      )}`
    );
};
```

```
const useFetchAvailableTimeslots = (
  idProduct: string,
  location: string,
  date: Dayjs
): any => {
  return useQuery(
    FETCH_AVAILABLE_TIMESLOTS,
    fetchAvailableTimeslots(idProduct, location, date),
    {
      enabled: false,
      refetchOnWindowFocus: false,
    }
  );
};

export default useFetchAvailableTimeslots;
```

Kako bi rezervacija bila kreirana istu je potrebno potvrditi klikom na gumb "Confirm". Uspješno kreiranje rezervacije rezultira *toast* notifikacijom u gornjem desnom kutu Web preglednika.



Slika 65. Zaslon za potvrdu rezervacije

`useMutatuon` *hook* koji se koristi za spremanje kreirane rezervacije.

```
export const saveRent = (rentForm: RentForm) => {
  return async () =>
    await axios.post<void>("/rents", {...rentForm,
      timeslots: formatTimeArray(rentForm.timeslots, APP_TIME_FORMAT),
    });
};
```

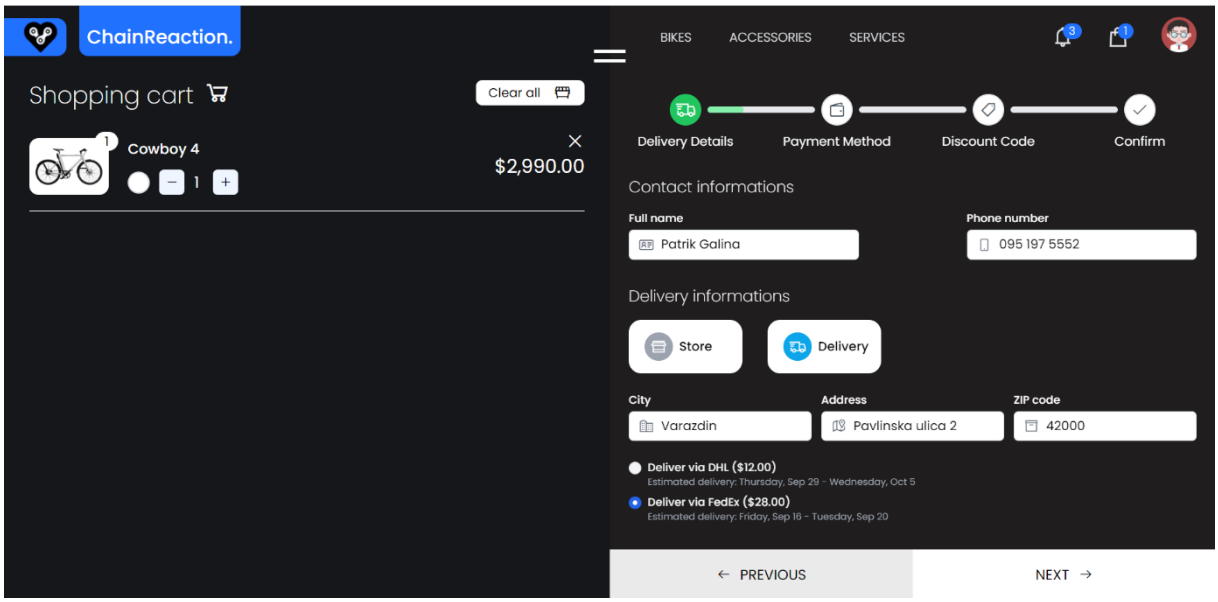
```
const useSaveRent = (rentForm: RentForm): any => {
  return useMutation(SAVE_RENT, saveRent(rentForm));
};

export default useSaveRent;
```

6.5.6.8. Kreiranje narudžbe

Proizvode dodane u korisničku košaricu moguće je naručiti putem forme za kreiranje narudžbe. Plaćanje je moguće obaviti pouzećem, odnosno gotovinom prilikom dostave paketa, ili kreditnom karticom slijedeći upute navedene na samoj formi.

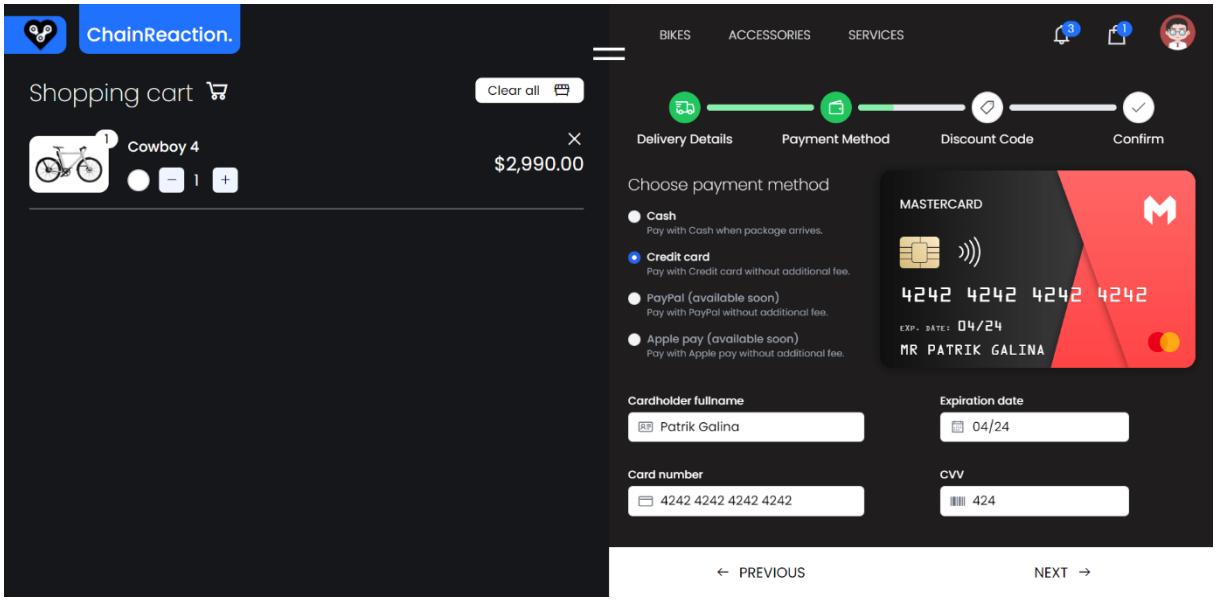
Dohvaćanje podataka obavlja se putem lokalnog spremišta (engl. *Local Storage*) Web preglednika koje omogućava njihovu perzistenciju te dohvaćanje prema odgovarajućem ključu.

Kreiranje narudžbe započinje unosom detalja vezanih uz dostavu artikala.



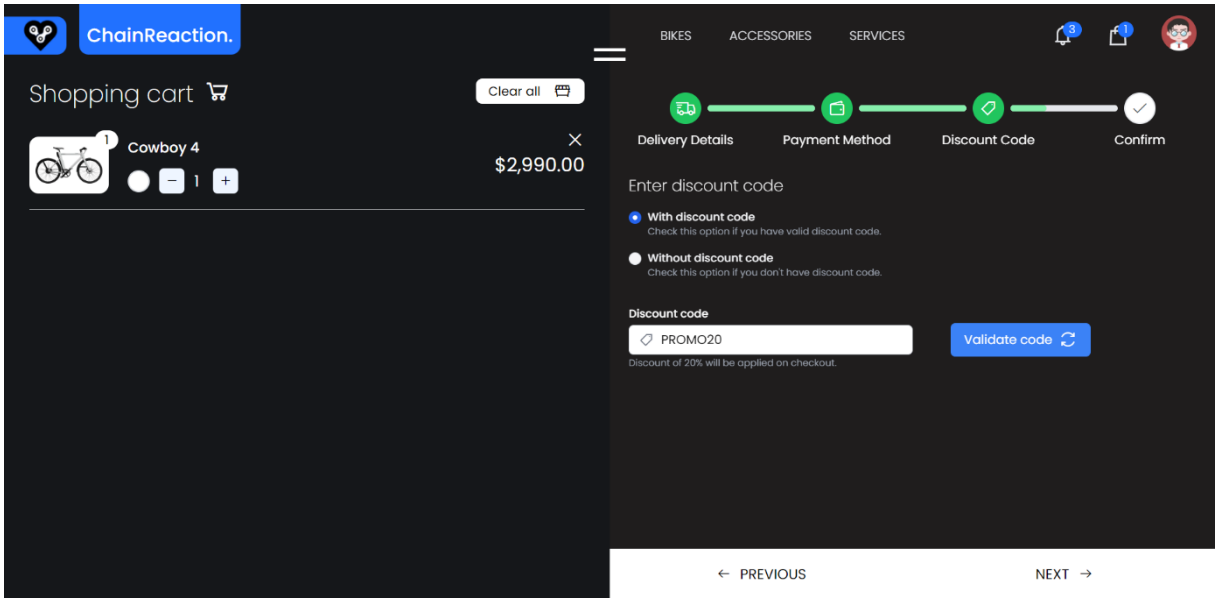
Slika 66. Zaslon za unos detalja dostave

Nakon unosa podataka vezanih uz dostavu potrebno je odabrati način plaćanja te unijeti zahtijevane podatke u slučaju odabira plaćanja kreditnim karticama.



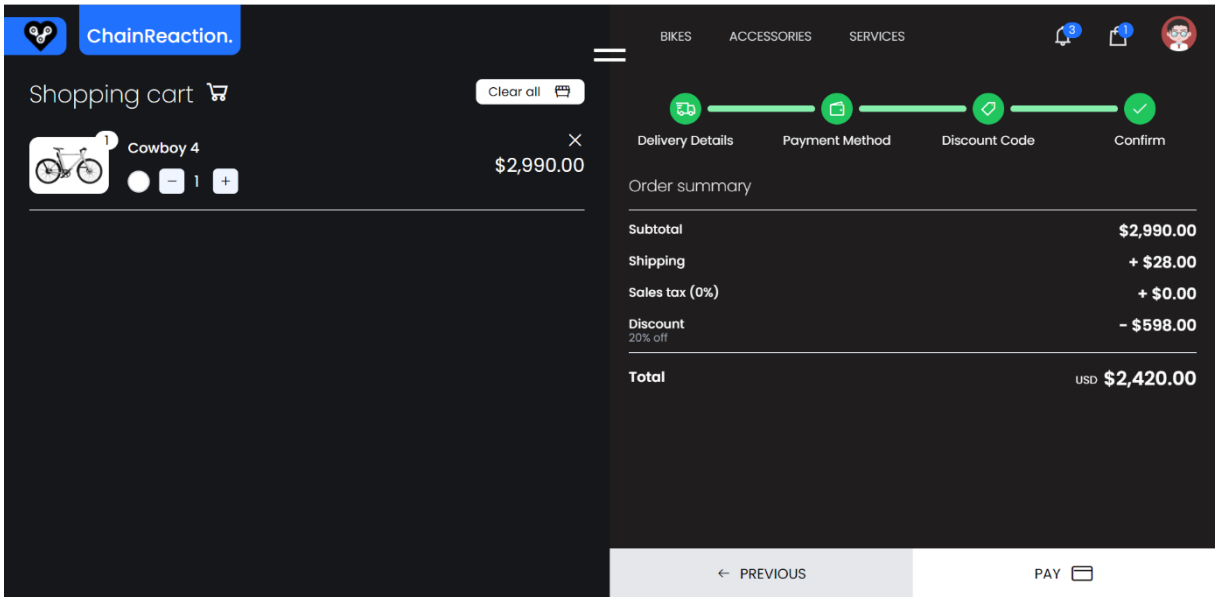
Slika 67. Zaslon za odabir načina plaćanja

Korisnik zatim ima mogućnost unosa promo koda koji osigurava popust na ukupan iznos narudžbe. Unos promo koda nije obavezan te korisnik može preskočiti unos promo koda odabirom opcije "Without discount code".



Slika 68. Zaslon za unos promo koda

Kako bi narudžba bila kreirana istu je potrebno potvrditi klikom na gumb "Pay". Klikom na gumb korisnik se preusmjerava na *payment gateway*, odnosno URL putem korisnik obavlja plaćanje unosom zahtijevanih podataka.



Slika 69. Zaslon za potvrdu rezervacije

Autorizaciju prilikom plaćanja kreditnim karticama obavlja vanjski servis koji nakon uspješne transakcije vraća *autorizacijski token* temeljem kojeg se vrše dodatne akcije na strani poslužitelja. Vanjskom servisu se iz tog razloga prosljeđuje *callback* metoda putem koje vanjski servis vraća autorizacijski token.

`handleStripeToken` metoda koja se koristi za dohvaćanje autorizacijskog tokena.

```
export const handleStripeToken = async (
  token: Token,
  amount: number,
  onSuccess: any,
  onError: any
) : Promise<void> => {
  await axios
    .get<void>(`/payments/charge?amount=${amount}`, {
      headers: { token: token?.id },
    })
    .then(() => onSuccess())
    .catch((error) => onError(error));
};
```

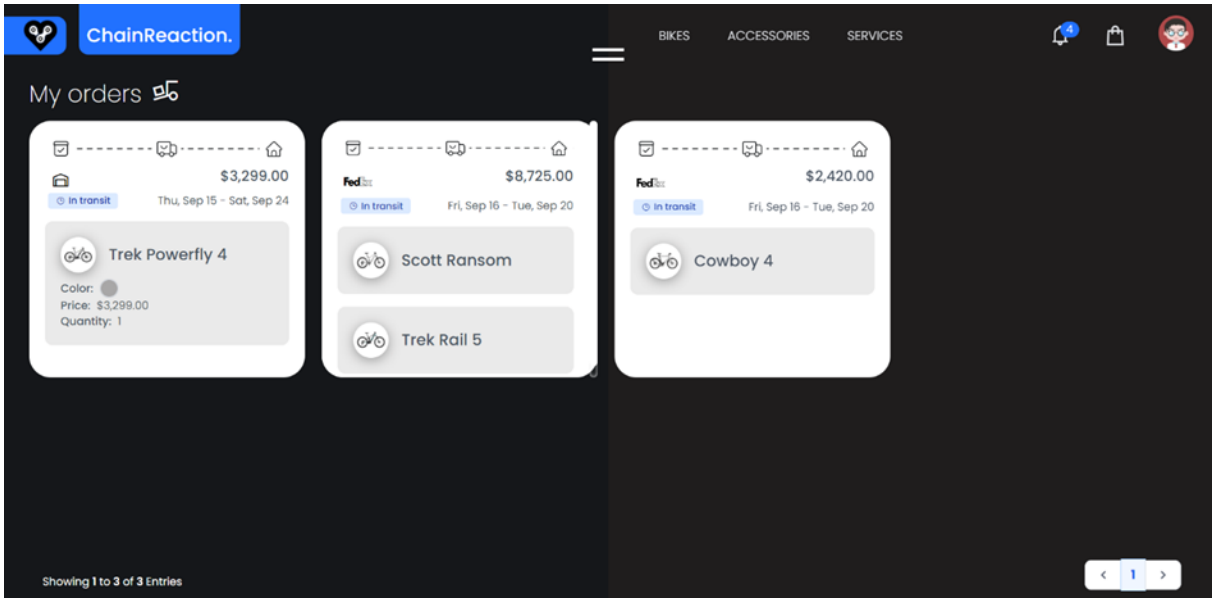
Ukoliko je plaćanje bilo uspješno tada se poziva `onSuccess` metoda koja obavlja perzistenciju kreirane narudžbe.

```
export const saveOrder = (orderForm: OrderForm) => {
    return async () => await axios.post<void>("/orders", orderForm);
};
```

```
const useSaveOrder = (orderForm: OrderForm): any => {
    return useMutation(SAVE_ORDER, saveOrder(orderForm));
};

export default useSaveOrder;
```

6.5.6.9. Pregled narudžbi

Korisnicima je omogućen pregled pojedine kreirane narudžbe zajedno s detaljima svake od njih. Za svaku stavku narudžbe navedene su osnovne informacije koje uključuju specifikacije stavke te naručenu količinu. Dohvaćanje podataka obavlja se slanje HTTP zahtjeva GET metodom na `/api/orders?page&size&idUser`.



Slika 70. Zaslon za pregled korisničkih narudžbi

`useQuery` *hook* koji se koristi za dohvaćanje podataka o korisničkim narudžbama.

```
export const fetchOrdersByIdUser = (
    idUser: number,
    pagination: Pagination) => {
        return async () => await axios.get<OrderPage>(
`/orders?page=${pagination.page}&size=${pagination.size}&
idUser=${idUser}`
    );
};
```

```
const useFetchOrdersByIdUser = (idUser: number): any => {
  return useMutation(FETCH_ORDERS_BY_ID_USER, (data: any) => {
    return fetchOrdersByIdUser(idUser, data?.pagination)();
  });
};

export default useFetchOrdersByIdUser;
```

7. Zaključak

Primarni cilj ovo rada bio je opisati teorijske aspekte procesa razvoja Web aplikacija kroz sistematizaciju tehnologija koje se pritom koriste te primijeniti spomenute koncepte u okviru praktičnog dijela rada. Glavne metode istraživanja i analize podataka, koje su korištene kako bi se determinirali pojedini dijelovi stručne terminologije, su metode analize, sinteze, indukcije te deskripcije. Sistematizacija teorijskog dijela rada konceptualizirana je kroz poglavlja pri čemu svako od poglavlja definira širi spektar pojedinog dijela koncepta na kojem se temeljio kasniji razvoj aplikacije. U uvodnom dijelu obrađena je kratka povijest elektroničke trgovine te su definirani osnovni koncepti elektroničke trgovine. U nastavku su opisani modeli elektroničke trgovine koji se uobičajeno koriste u današnje vrijeme te je dan kratak osvrt na digitalnu transformaciju poslovanja primjenom koncepata elektroničke trgovine. U nastavku slijedi poglavlje vezano uz Internet u kojem se navodi njegova kratka povijest te najvažnije tehnologije na kojima se temelji razvoj modernih aplikacija, od HTTP-a pa sve do Web servisa, API-a te programskih okvira. Na samom kraju prikazan je koncept komunikacije u realnom vremenu primjenom simulativnih tehnika, odnosno metoda temeljenih na HTTP-u, pa sve do suvremenih tehnika, odnosno metoda temeljenih na WebSocket tehnologiji. Teorijski dio rada zaključen je kratkim osvrtom na sigurnost Web aplikacija.

Nakon detaljne razrade osnovnih koncepata i terminologije, slijedi praktični dio rada u kojem je sukcesivno opisan proces razvoja Web aplikacije. U okviru praktičnog dijela razvijena je Web aplikacija koja predstavlja virtualnu trgovinu koja krajnjim korisnicima omogućava kupnju bicikala, dijelova i opreme za bicikle kao i rezervaciju elektroničkih bicikala, pri čemu je poseban naglasak stavljen na maksimalno zadovoljavanje korisničkih potreba, a samim time i poboljšanje korisničkog iskustva prilikom korištenja elektroničke trgovine. Prilikom razvoja Web aplikacije korišteni se programski jezici JavaScript i Java, točnije programski okviri *React*, za *frontend* dio, te *Spring Boot*, za *backend* dio Web aplikacije. React je deklarativna i fleksibilna JavaScript biblioteka otvorenog koda koja se koristi za izgradnju modularnog korisničkog sučelja sastavljenog od komponenata. Spring Framework, Java razvojno okruženje otvorenog koda kojeg odlikuje jednostavnost, fleksibilnost, modularnost te *backward* kompatibilnost, javio se kao odgovor na kompleksan razvoj servlet aplikacija te dotad korišteni JEE standard za izradu robusnih Web aplikacija.

Razvijeno programsko rješenje prikazuje način na koji se, kroz digitalnu transformaciju poslovanja, mogu reducirati problemi s kojima se susreće suvremena trgovina. Reduciranje problema suvremene trgovine eksplicitno rezultira povoljnijim tržišnim položajem te samim time utječe ma postizanje željenih tržišnih rezultata te materijalnih i nematerijalnih ciljeva koji su postavljeni od strane rukovodstva.

Popis literature

[1] E. Stahl, T. Krabichler, M. Breitschaft, i G. Wittmann, *ECommerce-Leitfaden: Erfolgreicher im elektronischen Handel*, 2. izdanje, Regensburg: Univ.-Verl. Regensburg, 2009.

[2] H. Bezić, A. Gašparini, i L. Bagarić, *Elektronička trgovina u malim i srednjim poduzećima Republike Hrvatske*, Ekonomski vjesnik, vol.XXII, br. 2, str. 266-281, 2009.

[3] B. Matić, *Međunarodno poslovanje*. Zagreb: Sinergija nakladništvo d.o.o., 2014.

[4] I. Butigan, *Razvoj e-trgovine i njezin utjecaj na gospodarstvo Hrvatske* [Diplomski rad], Sveučilište u Splitu, Ekonomski fakultet, Split, 2019. Dostupno: http://urn.nsk.hr/urn:nbn:hr:124:455638 [pristupano: 27.04.2022.]

[5] J. Fernando, *Payment Gateway* [Na internetu], 2021. Dostupno: http://www.investopedia.com/terms/p/payment-gateway.asp [pristupano: 27.04.2022.]

[6]] N. Pleša Puljić, M. Celić, i M. Puljić, *Povijest i budućnost prodavaonica* [Stručni rad], Visoka škola za menadžment u turizmu i informatici u Virovitici, Virovitica, 2017. Dostupno: http://hrcak.srce.hr/195829 [pristupano: 27.04.2022.]

[7] P. A. Samuelson, W. D. Nordhaus, *Ekonomija*, 19. izdanje, Zagreb: Mate, 2011.

[8] J. Bijelić, *Aktualni trendovi u razvoju elektroničke trgovine* [Završni rad], Sveučilište Jurja Dobrile u Puli, Pula, 2017. Dostupno: http://urn.nsk.hr/urn:nbn:hr:137:357715 [pristupano: 28.04.2022.]

[9] J. Markoff, *What the Dormouse Said: How the Sixties Counterculture Shaped the Personal Computer Industry*, 3. izdanje, New York: Penguin Books, 2011.

[10] Purple Co., *The history of online shopping* [Blog], 2021. Dostupno: http://purple.ai/blogs/the-history-of-online-shopping [pristupano: 28.04.2022.]

[11] T. Čendo Metzinger, M. Toth, *Metodologija istraživačkog rada za stručne studije* [Priručnik], Veleučilište Velika Gorica, Velika Gorica, 2020. Dostupno :http://www.bib.irb.hr/1058026 [pristupano: 28.04.2022.]

[12] D. Ružić, A. Biloš, D. Turkalj, *E-marketing*, 3. izdanje, Factum d.o.o.: Osijek, 2014.

[13] V. Čerić, *Internet economy and electronic commerce*, Journal of Information and Organizational Sciences, 2. izdanje, str. 143-161, 2000. Dostupno: http://hrcak.srce.hr/78699 [pristupano: 29.04.2022.]

[14] S. Žarković, *Elektronička trgovina* [Završni rad], Sveučilište u Zadru, Zadar, 2018. Dostupno: http://urn.nsk.hr/urn:nbn:hr:162:364215 [pristupano: 30.04.2022.]

[15] D. DeMatas, *10 Types of Ecommerce Business Models That Work Right Now* [Na internetu], 2022. Dostupno: http://www.ecommerceceo.com/types-of-ecommerce-business-models [pristupano: 02.05.2022.]

[16] D. Ružić, A. Biloš, i Davorin Turkalj, *E-marketing*, 2. izdanje, Sveučilište Josipa Jurja Strossmayera u Osijeku, Ekonomski fakultet u Osijeku: Osijek, 2011.

[17] BizzPortal.ru, *Elektronski novčanik: vrste, funkcije i njihove mogućnosti* [Na internetu]. Dostupno: http://hr.bizzportal.ru/elektronski-novcanik-vrste-funkcije-i-njihove-mogucnosti [pristupano: 03.05.2022.]

[18] CARNet i suradnici, *Elektronski novac* [Na internetu], 2010. Dostupno: http://www.cis.hr/www.edicija/LinkedDocuments/NCERT-PUBDOC-2010-09-311.pdf [pristupano: 03.05.2022.]

[19] C. Ferreira, *What Is Dropshipping and How Does It Work?* [Na internetu], 2022. Dostupno: http://www.cis.hr/www.edicija/LinkedDocuments/NCERT-PUBDOC-2010-09-311.pdf [pristupano: 04.05.2022.]

[20] Enciklopedija.hr, *Pojam marže* [Enciklopedija]. Dostupno: http://www.enciklopedija.hr/natuknica.aspx?id=39247 [pristupano: 05.05.2022.]

[21] B. Andony, *White-Label vs Private Label: What are They?* [Na internetu], 2018. Dostupno: http://www.vendasta.com/blog/white-label-vs-private-label [pristupano: 09.05.2022.]

[22] R. Kriss, *The Bricks and Clicks Business Model: Everything You Need to Know* [Na internetu], 2020. Dostupno: http://www.fundera.com/blog/brick-and-click [pristupano: 09.05.2022.]

[23] G. Contreras, *Internet transactions and business models* [Na internetu]. Dostupno: http://dialnet.unirioja.es/descarga/articulo/5137650.pdf [pristupano: 09.05.2022.]

[24] R. Babić, A. Krajnović, i A. Radman-Peša, *Dosezi elektroničke trgovine u Hrvatskoj i svijetu* [Stručni rad], Sveučilište u Zadru, Odjel za ekonomiju Sveučilišta u Zadru, Zadar, 2011. Dostupno: http://hrcak.srce.hr/75179 [pristupano: 10.05.2022.]

[25] Z. Živković, Z. Milosavljević, *Prednosti i nedostatci elektronske trgovine* [Stručni rad], Fakultet za industrijski rnenadžment Kruševac, Kruševac, 2008. Dostupno: http://scindeks-clanci.ceon.rs/data/pdf/0354-6829/2008/0354-682908022052.pdf [pristupano: 11.05.2022.]

[26] S. Franc, I. Dužević, *Digitalna transformacija i trgovina*, Sveučilište u Zagrebu, Ekonomski fakultet, 2020.

[27] Eurostat, *Online shopping ever more popular in 2020* [Članak]. Dostupno: http://ec.europa.eu/eurostat/web/products-eurostat-news/-/ddn-20210217-1 [pristupano: 12.05.2022.]

[28] Državni zavod za statistiku, *Arhiva objavljenih podataka* [Na internetu]. Dostupno: http://web.dzs.hr/arhiva.htm [pristupano: 12.05.2022.]

[29] M. Spremić, *Digitalna transformacija poslovanja*, Sveučilište u Zagrebu, Ekonomski fakultet, Zagreb, str. 53, 2017.

[30] Hrvatska gospodarska komora, *Trgovina se ubrzano digitalizira, tko ne prati te promjene neće biti konkurentan* [Članak], 2021. Dostupno: http://www.hgk.hr/odjel-trgovinu/trgovina-se-ubrzano-digitalizira-tko-ne-prati-te-promjene-nece-biti-konkurentan-najava [pristupano: 12.05.2022.]

[31] Enciklopedija.hr, *Pojam Interneta* [Enciklopedija]. Dostupno: http://www.enciklopedija.hr/natuknica.aspx?ID=27653
[pristupano: 13.05.2022.]

[32] D. Kermek, *Internet, Web, protokoli* [prezentacija s kolegija *Web dizajn i programiranje*], Sveučilište u Zagrebu, Fakultet organizacije i informatike, Varaždin, 2019.

[33] J, Gillies, R. Cailliau, *How the Web was Born: The Story of the World Wide Web*, Oxford University Press: Oxford, 2000.

[34] CERN, *World Wide Web* [Na internetu]. Dostupno: http://info.cern.ch/hypertext/WWW/TheProject.html [pristupano: 13.05.2022.]

[35] Enciklopedija.hr, *Pojam hiperteksta* [Enciklopedija]. Dostupno: http://www.enciklopedija.hr/natuknica.aspx?ID=25636
[pristupano: 13.05.2022.]

[36] Nacionalni centar za učenje na daljinu "Nikola Tesla", *Uvod u Internet* [Na internetu]. Dostupno: http://tesla.carnet.hr/mod/book/view.php?id=5428&chapterid=883 [pristupano: 14.05.2022.]

[37] M. Popović, *Prikazi nacionalnih parkova na Web-u* [Diplomski rad], Sveučilište u Zagrebu, Geodetski fakultet, Zagreb, 2002. Dostupno: http://www.kartografija.hr/old_hkd/obrazovanje/diplomski/popovic/2.htm [pristupano: 14.05.2022.]

[38] J. Naughton, *The evolution of the Internet: from military experiment to General Purpose Technology* [Članak], 2016. Dostupno: http://www.tandfonline.com/doi/full/10.1080/23738871.2016.1157619 [pristupano: 14.05.2022.]

[39] SRI International, *75 Years of Innovation: ARPANET* [Članak], 2020. Dostupno: http://medium.com/dish/75-years-of-innovation-arpanet-4c23a0162d25
[pristupano: 14.05.2022.]

[40] T. Berners-Lee i suradnici, *Information Management: A Proposal* [Na internetu], 1990. Dostupno: http://www.w3.org/History/1989/proposal.html
[pristupano: 14.05.2022.]

[41] M. Sharma, *Comparison Between Web 1.0, Web 2.0 and Web 3.0* [Članak], 2022. Dostupno: http://www.geeksforgeeks.org/web-1-0-web-2-0-and-web-3-0-with-their-difference [pristupano: 17.05.2022.]

[42] E. Mičetić, *Web 1.0 i Web 2.0 bile su etape u razvoju interneta* [Članak], 2022. Dostupno: http://pcchip.hr/internet/digitalni-mediji/web-1-0-i-web-2-0-bile-su-etape-u-razvoju-interneta-procitajte-koje-su-njihove-karakteristike-i-sto-bi-u-buducnosti-trebao-donijeti-web-3-0
[pristupano: 22.05.2022.]

[43] T. O'Reilly, *What is Web 2.0* [Članak], 2005. Dostupno: http://www.oreilly.com/pub/a/web2/archive/what-is-web-20.html [pristupano: 23.05.2022.]

[44] Economy-Pedia.com, *Dot-com balon* [Na internetu]. Dostupno: http://hr.economy-pedia.com/11031534-dot-com-bubble
[pristupano: 23.05.2022.]

[45] A. Hayes, *Blockchain Facts: What is it, how it works and how it can be used* [Na internetu], 2022. Dostupno: http://www.investopedia.com/terms/b/blockchain.asp [pristupano: 24.05.2022.]

[46] Bitcoin Store, *Što je Web 3.0? Sve što trebate znati o Internetu budućnosti* [Članak], 2021. Dostupno: http://www.bitcoin-store.hr/blog/sto-je-web3-i-kako-funkcionira [pristupano: 24.05.2022.]

[47] OFIR, *IoT ili Internet stvari* [Na internetu], 2019. Dostupno: http://www.ofir.hr/iot-ili-internet-stvari-2 [pristupano: 27.05.2022.]

[48] NetBIT, *Web aplikacija ili web stranica, kako prepoznati razliku?* [Članak], 2022. Dostupno: http://netbit.hr/web-aplikacija-ili-web-stranica-2 [pristupano: 30.05.2022.]

[49] M. Jazayeri, Some Trends in Web Application Development [Članak], Conference "Future of Software Engineering", FOSE 2007. Dostupno: http://ieeexplore.ieee.org/abstract/document/4221621 [pristupano: 30.05.2022.]

[50] L. Shklar, *Web Application Architecture Principles, protocols and practices*, John Wiley & Sons, Inc: Hoboken, 2007.

[51] neeva, *What is HTTP? How It Works and Related Safety Concerns* [Na internetu], 2021. Dostupno: http://neeva.com/learn/what-is-http [pristupano: 01.06.2022.]

[52] EIF, *EIF Annual Report 2004* [Na internetu], 2005. Dostupno: http://www.eif.org/news_centre/publications/EIF_annual-report-2004.htm [pristupano: 01.06.2022.]

[53] Cloudflare, *What is HTTP?* [Članak]. Dostupno: http://www.cloudflare.com/learning/ddos/glossary/hypertext-transfer-protocol-http/ [pristupano: 01.06.2022.]

[54] MDN Web docs, *Evolution of HTTP* [Na internetu]. Dostupno: http://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/Evolution_of_HTTP [pristupano: 02.06.2022.]

[55] J. Leyden, *HTTP/3: Everything you need to know about the next-generation web protocol* [Članak], 2021. Dostupno: http://portswigger.net/daily-swig/http-3-everything-you-need-to-know-about-the-next-generation-web-protocol [pristupano: 03.06.2022.]

[56] MDN Web docs, *What is a URL?* [Na internetu]. Dostupno: http://developer.mozilla.org/en-US/docs/Learn/Common_questions/What_is_a_URL [pristupano: 06.06.2022.]

[57] Mixed Analytics, Add Request Body [Na internetu], 2022. Dostupno: http://mixedanalytics.com/knowledge-base/add-body-post-requests [pristupano: 07.06.2022.]

[58] G. Kaushik, *What is HTTP Response?* [Na internetu], 2022. Dostupno: http://www.toolsqa.com/client-server/http-response [pristupano: 09.06.2022.]

[59] TutorialsPoint, *HTTP - Responses* [Na internetu]. Dostupno: http://www.tutorialspoint.com/http/http_responses.htm [pristupano: 11.06.2022.]

[60] D. Kermek, *Web servisi* [prezentacija s kolegija *Napredne Web tehnologije i servisi*], Sveučilište u Zagrebu, Fakultet organizacije i informatike, Varaždin, 2021.

[61] T. Bush, *What Is The Difference Between Web Services and APIs?* [Na internetu], 2019. Dostupno: http://nordicapis.com/what-is-the-difference-between-web-services-and-apis/ [pristupano: 13.06.2022.]

[62] M. Hernandez, *Top 3 Online Tools for Simulating HTTP Requests* [Članak], 2017. Dostupno: http://ubidots.com/blog/top-3-online-tools-for-simulating-http-requests [pristupano: 13.06.2022.]

[63] Cloudflare, *What is an API?* [Na internetu]. Dostupno: http://www.cloudflare.com/en-gb/learning/security/api/what-is-an-api [pristupano: 14.06.2022.]

[64] C. Hoffman, *What Is an API, and How Do Developers Use Them?* [Na internetu], 2021. Dostupno: http://www.howtogeek.com/343877/what-is-an-api [pristupano: 14.06.2022.]

[65] P. Marić, *Razvoj RESTFUL web usluge i aplikacijskog programskog sučelja u .NET okviru* [Završni rad], Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek, Osijek, 2018. Dostupno: http://urn.nsk.hr/urn:nbn:hr:200:425618 [pristupano: 15.06.2022.]

[66] L. Gupta, *REST Architectural Constraints* [Na internetu], 2022. Dostupno: http://restfulapi.net/rest-architectural-constraints [pristupano: 15.06.2022.]

[67] RedHat, *What is a REST API?* [Na internetu], 2020. Dostupno: http://www.redhat.com/en/topics/api/what-is-a-rest-api [pristupano: 15.06.2022.]

[68] R. T. Fielding, *Architectural Styles and the Design of Network-based Software Architectures* [Doktorska disertacija], 2000. Dostupno: http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm [pristupano: 15.06.2022.]

[69] Typescriptlang.org, *What is TypeScript?* [Na internetu]. Dostupno: http://www.typescriptlang.org [pristupano: 17.06.2022.]

[70] Weardevelopers, *Sve što trebate znati o programskom jeziku TypeScript* [Na internetu], 2021. Dostupno: http://www.debug.hr/sve-sto-trebate-znati-o-programskom-jeziku-typescript [pristupano: 17.06.2022.]

[71] Reactjs.org, *React* [Na internetu]. Dostupno: http://reactjs.org [pristupano: 18.06.2022.]

[72] A. M. Vipul, P. Sonpatki, *ReactJS by Example - Building Modern Web Applications with React*, Packt Publishing: Birmingham, 2016.

[73] B. Allen, *Što je DOM?* [Na internetu], 2020. Dostupno: http://gocoding.org/hr/what-is-dom-document-object-model [pristupano: 18.06.2022.]

[74] Vercel, *From JavaScript to React* [Na internetu]. Dostupno: http://nextjs.org/learn/foundations/from-javascript-to-react/building-ui-with-components [pristupano: 18.06.2022.]

[75] Java, *What is Java technology and why do I need it?* [Na internetu]. Dostupno: http://www.java.com/en/download/help/whatis_java.html [pristupano: 20.06.2022.]

[76] J. Hartman, *What is Java? Definition, Meaning & Features of Java Platforms* [Na internetu], 2022. Dostupno: http://www.guru99.com/java-platform.html [pristupano: 20.06.2022.]

[77] D. Kermek, *Uvod u programski jezik Java* [prezentacija s kolegija *Napredne Web tehnologije i servisi*], Sveučilište u Zagrebu, Fakultet organizacije i informatike, Varaždin, 2021.

[78] Elzawawy, *Java bytecode generator* [Na internetu], 2020. Dostupno: http://github.com/Elzawawy/java-bytecode-generator [pristupano: 20.06.2022.]

[79] D. Miessler, *The difference between URL and URI* [Na internetu] 2022. Dostupno: http://danielmiessler.com/study/difference-between-uri-url [pristupano: 22.06.2022.]

[80] SymfonyCasts, *REST: Resources and Representations* [Na internetu]. Dostupno: http://symfonycasts.com/screencast/rest/rest [pristupano: 22.06.2022.]

[81] Paessler, *What is REST?* [Na internetu]. Dostupno: http://www.paessler.com/it-explained/rest [pristupano: 22.06.2022.]

[82] TutorialsPoint, *Data Flow Architecture* [Na internetu]. Dostupno: http://www.tutorialspoint.com/software_architecture_design/data_flow_architecture.htm [pristupano: 25.06.2022.]

[83] B. Clark, *What is GraphQL: History, Components, and Ecosystem* [Članak], 2019. Dostupno: http://levelup.gitconnected.com/what-is-graphql-87fc7687b042 [pristupano: 30.08.2022.]

[84] T. Rascia, *An Introduction to GraphQL* [Članak], 2021. Dostupno: http://www.taniarascia.com/introduction-to-graphql [pristupano: 30.08.2022.]

[85] ApolloDOCS, *Schema basics* [Na internetu]. Dostupno: http://www.apollographql.com/docs/apollo-server/v2/schema/schema [pristupano: 30.08.2022.]

[86] M. Stuart, *GraphQL Resolvers: Best Practices* [Na internetu], 2018. Dostupno: http://medium.com/paypal-tech/graphql-resolvers-best-practices-cd36fdbcef55 [pristupano: 31.08.2022.]

[87] R. Ranjan, *What is a Framework in Programming & Why You Should Use One* [Članak], 2021. Dostupno: http://www.netsolutions.com/insights/what-is-a-framework-in-programming [pristupano: 31.08.2022.]

[88] Vercel, *What is Next.js?* [Na internetu]. Dostupno: http://nextjs.org/learn/foundations/about-nextjs/what-is-nextjs [pristupano: 31.08.2022.]

[89] S. Hayani, *Nextjs for everyone* [Na internetu], 2018. Dostupno: http://www.freecodecamp.org/news/an-introduction-to-next-js-for-everyone-507d2d90ab54 [pristupano: 31.08.2022.]

[90] A. Danzante, *Korištenje razvojnog okvira Spring Boot za implementaciju servisa SOAP i REST* [Diplomski rad], Sveučilište u Zagrebu, Fakultet organizacije i informatike, Varaždin, 2019. Dostupno: http://urn.nsk.hr/urn:nbn:hr:211:511795 [pristupano: 14.05.2022.]

[91] Spring.io, *Introduction to Spring Framework* [Na internetu]. Dostupno: http://docs.spring.io/spring-framework/docs/4.1.2.RELEASE/spring-framework-reference/html/overview.html [pristupano: 31.08.2022.]

[92] A. Saputra, *Build a RESTful CRUD API With Spring Boot and JPA* [Članak], 2021. Dostupno: http://medium.com/codestorm/build-a-restful-crud-api-with-spring-boot-and-jpa-fd97e7f02615 [pristupano: 31.08.2022.]

[93] Hibernate.org, *Hibernate* [Na internetu]. Dostupno: http://hibernate.org [pristupano: 01.09.2022.]

[94] K. Johannessen, *Real Time Web Applications: Comparing frameworks and transport mechanisms* [Diplomski rad], University of Oslo, Oslo, 2014. Dostupno: http://www.duo.uio.no/handle/10852/42049 [pristupano: 01.09.2022.]

[95] R. Selmer, *What Is Real Time Communications?* [Na internetu], 2017. Dostupno: http://www.vonage.com/resources/articles/real-time-communications [pristupano: 01.09.2022.]

[96] A. Z. Mcginnis, *Real-Time Communication Over Web Applications* [Članak], 2021. Dostupno: http://medium.com/@alex.z.mcginnis/real-time-communication-over-web-applications-2b2ee21176b2 [pristupano: 01.09.2022.]

[97] S. Basu, *Real-Time Communication Techniques* [Na internetu], 2019. Dostupno: http://www.telerik.com/blogs/real-time-communication-techniques [pristupano: 01.09.2022.]

[98] M. Fietkiewicz, *WebSockets vs. HTTP* [Članak], 2021. Dostupno: http://ably.com/topic/websockets-vs-http [pristupano: 01.09.2022.]

[99] G. Ayrancıoğlu, *What is Server-Sent Events (SSE) and how to implement it?* [Članak], 2022. Dostupno: http://medium.com/yemeksepeti-teknoloji/what-is-server-sent-events-sse-and-how-to-implement-it-904938bffd73 [pristupano: 01.09.2022.]

[100] ably, *Server-Sent Events (SSE): A Conceptual Deep Dive* [Na internetu], 2020. Dostupno: http://ably.com/topic/server-sent-events [pristupano: 01.09.2022.]

[101] Synopysy, *Web application security* [Članak]. Dostupno: http://www.synopsys.com/glossary/what-is-web-application-security.html [pristupano: 02.09.2022.]

[102] JWT.io, JWT [Na internetu]. Dostupno: http://jwt.io [pristupano: 02.09.2022.]

[103] Računala forenzika FER, *JSON Web Token (JWT)* [Na internetu], 2021. Dostupno: http://racfor.zesoi.fer.hr/doku.php?id=racfor_wiki:json_web_token [pristupano: 02.09.2022.]

[104] SuperTokens Team, *What is a JWT? Understanding JSON Web Tokens* [Članak], 2022. Dostupno: http://supertokens.com/blog/what-is-jwt [pristupano: 02.09.2022.]

[105] Virtualna tvornica, 5 najboljih Payment Gateway sustava [Na internetu]. Dostupno: http://www.virtualna-tvornica.com/payment-gateway [pristupano: 02.09.2022.]

[106] C. Laurer, *How to build a BNPL solution* [Na internetu], 2022. Dostupno: http://trimplement.com/blog/author/christoph-laurer [pristupano: 03.09.2022.]

[107] BlueSnap developers, *PayPal* [Na internetu]. Dostupno: http://developers.bluesnap.com/docs/paypal [pristupano: 03.09.2022.]

[108] WSPay, *Tokenizirane transakcije - transakcije sa spremljenim kartičnim podacima* [Članak]. Dostupno: http://www.wspay.info/cd/137/tokenizirane-transakcije-transakcije-sa-spremljenim-karticnim-podacima [pristupano: 03.09.2022.]

[109] D. Kermek, *Arhitektonski uzroci dizajna* [prezentacija s kolegija *Uzorci dizajna*], Sveučilište u Zagrebu, Fakultet organizacije i informatike, Varaždin, 2021.

[110] L. Gupta, *Spring @Configuration annotation* [Na internetu], 2020. Dostupno: http://howtodoinjava.com/spring-core/spring-configuration-annotation [pristupano: 05.09.2022.]

[111] T. Dąbrowski, *Using Spring Boot for WebSocket Implementation with STOMP* [Na internetu], 2019. Dostupno: http://www.toptal.com/java/stomp-spring-boot-websocket [pristupano: 06.09.2022.]

Popis slika