# Teorijske osnove

## Frontend programiranje

Odnosi se na proces razvoja korisničkog interfejsa i funkcionalnosti koje korisnici direktno vide i interaguju sa njima prilikom korišćenja *web* aplikacija, mobilnih aplikacija ili drugih vrsta softvera. Ovo uključuje sve što se prikazuje u veb *browser*-u ili na ekranu mobilnog uređaja, uključujući tekst, slike, forme za unos podataka, dugmad, menije i druge elemente.

*Frontend* programeri koriste jezike kao što su HTML (*HyperText* *Markup* *Language*) za strukturu veb stranice, CSS (*Cascading* *Style* *Sheets*) za stilizovanje i izgled stranice, i JavaScript za interaktivnost i dinamičnost. Ovi jezici rade zajedno kako bi kreirali funkcionalne i estetski prijatne korisničke interfejse.

Takođe, *frontend* programiranje uključuje rad sa različitim alatima i okvirima (npr. React, Angular, Vue.js) koji olakšavaju razvoj kompleksnih korisničkih interfejsa i omogućavaju bolje upravljanje stanjem aplikacije.

Kroz *frontend* programiranje, programeri se fokusiraju na obezbeđivanje pozitivnog korisničkog iskustva tako što pravilno organizuju informacije, olakšavaju navigaciju i osiguravaju da aplikacija bude odzivna i brza.

## HTML (HyperText Markup Language)

Standardni jezik za označavanje i strukturiranje sadržaja veb stranica. To je osnovni jezik koji se koristi za izgradnju i oblikovanje elemenata koji se prikazuju na veb stranicama. Kroz HTML, možemo definisati različite delove veb stranice, kao što su naslovi, paragrafi, slike, veze, forme i druge komponente.

HTML koristi "oznake" ili "tagove" da označi različite delove teksta i elemenata na stranici. Oznake se obično sastoje od otvarajućeg i zatvarajućeg dela, gde otvarajuća oznaka označava početak elementa, dok zatvarajuća označava kraj. Na primer, <h1> označava početak naslova prvog nivoa (*heading* *level* 1), a </h1> označava kraj tog naslova. Pravilno strukturiran HTML je ključan za pravilno prikazivanje veb stranica na različitim uređajima i veb *browser*-ima.

## CSS (Cascading Style Sheets)

Predstavlja jezik za stilizovanje veb stranica i omogućava definisanje izgleda i prezentacije elemenata ( boje, fontovi, raspored, veličine i druge vizuelne karakteristike ) koje smo strukturirali koristeći HTML.

Omogućava odvajanje dizajna veb stranice od njenog sadržaja, čime se olakšava održavanje i ažuriranje veb stranica. Umesto da stilizujemo svaki element pojedinačno u HTML-u, koristimo CSS da definišemo stilove koji se primenjuju na različite elemente. Na primer, možemo definisati stil za sve naslove (<h1>, <h2>, <h3>, itd.) koji će važiti na svim stranicama na sajtu.

Princip "kaskadnog" (*Cascading*) u CSS-u se odnosi na to da stilovi mogu biti nasleđeni od roditeljskih elemenata, ali ih takođe možemo izmeniti ili nadjačati specifičnim stilovima za određene elemente.

CSS takođe omogućava prilagodljivost i odzivnost veb stranica na različite uređaje i veličine ekrana, što je ključno za kreiranje korisničkih interfejsa koji su funkcionalni i lepo izgledaju na svim platformama.

Ukratko, CSS je jezik koji omogućava oblikovanje i stilizovanje veb stranica kako bi se postigao željeni izgled i korisničko iskustvo.

## JavaScript

Programski jezik koji se često koristi za razvoj dinamičkih i interaktivnih veb stranica. To je jezik koji se izvodi na klijentskoj strani (u veb *browser*-ima korisnika), što znači da se kod izvršava na računarima korisnika, a ne na serverima.

Omogućava programerima da kontrolišu ponašanje elemenata na veb stranici, reaguju na korisničke akcije (kao što su klikovi i unos teksta) i dinamički ažuriraju prikaz stranice bez potrebe za ponovnim učitavanjem celokupne stranice. Takođe se koristi za validaciju podataka unetih u forme, manipulaciju dokumentima i animaciju elemenata.

Pored toga, JavaScript je često korišćen i izvan veb *browser*-a, kao što je u razvoju mobilnih aplikacija koristeći okvire kao što su React Native ili u razvoju serverske aplikacije koristeći Node.js.

U suštini, JavaScript je snažan alat koji omogućava interaktivnost, dinamiku i kompleksne funkcionalnosti na veb stranicama i u drugim vrstama softvera.

## TypeScript

Jezik za programiranje koji proširuje funkcionalnosti JavaScript-a dodavanjem statičkog tipovanja. To znači da programeri mogu definisati tipove podataka (kao što su brojevi, stringovi, objekti, itd.) i koristiti ih u kodu, čime se pomaže u detekciji i sprečavanju potencijalnih grešaka prilikom razvoja softvera.

Osnovni cilj TypeScript-a je da obezbedi bolju strukturu i čitljivost koda, kao i veću sigurnost u pogledu tipova podataka. Programeri mogu definisati tipove za promenljive, funkcije, klase i druge konstrukte, a zatim TypeScript proverava da li se tipovi koriste na ispravan način u toku kompilacije.

Evo nekoliko ključnih karakteristika TypeScript-a:

* Statičko tipovanje: Omogućava programerima da definišu tipove promenljivih i drugih elemenata u kodu pre izvršavanja, čime se smanjuje broj grešaka vezanih za tipove tokom razvoja.
* Objektno-orijentisano programiranje: TypeScript podržava koncepte objektno-orijentisanog programiranja kao što su klase, nasleđivanje, interfejsi i apstraktne klase.
* Modularnost: Omogućava organizaciju koda u module, čime se olakšava ponovna upotreba i održavanje koda.
* Kompatibilnost sa JavaScript-om: TypeScript je nadskup JavaScript-a, što znači da je moguće koristiti većinu postojećeg JavaScript koda u TypeScript aplikacijama.
* Razvoj u timu: Poboljšava saradnju i razvoj u timu kroz bolje razumevanje koda zbog definisanih tipova i jasnijeg interfejsa.

TypeScript je postao popularan u razvoju veb i serverskih aplikacija, posebno u velikim projektima gde je važna stabilnost i sigurnost tipova podataka.

## Angular

Popularan open-source okvir (*framework*) za razvoj veb aplikacija. Razvila ga je kompanija Google i predstavlja moćan alat koji olakšava kreiranje dinamičkih, interaktivnih i skalabilnih veb aplikacija. To je klijentski (*frontend*) okvir koji se koristi za izgradnju modernih veb aplikacija koje mogu da rade u veb *browser*-ima.

Glavne karakteristike Angular-a uključuju:

* Komponentna arhitektura: Angular se bazira na komponentama, koje su nezavisne, ponovno upotrebljive i samostalne delove korisničkog interfejsa. Ovo olakšava organizaciju i upravljanje kodom.
* Dvosmerna veza podataka: Angular omogućava automatsko ažuriranje korisničkog interfejsa kada se podaci u modelu promene, i obrnuto. To olakšava praćenje i upravljanje podacima.
* Injekcija zavisnosti: Angular koristi koncept injekcije zavisnosti kako bi se omogućila bolja kontrola i organizacija komponenata i servisa.
* Modularnost: Kod se organizuje u module, što doprinosi boljoj organizaciji i skalabilnosti aplikacije.
* Rute i navigacija: Angular pruža mehanizam za definisanje ruta i navigaciju unutar aplikacije, omogućavajući korisnicima da prelaze između različitih delova aplikacije.
* Testiranje: Angular ima ugrađene alate za jednostavno testiranje koda, što doprinosi kvalitetu aplikacije.

Angular omogućava razvoj složenih aplikacija kao što su veb aplikacije, progresivne veb aplikacije (PWA) i veb stranice visokih performansi. Kroz svoje napredne funkcionalnosti, postao je popularan izbor za razvoj modernih veb aplikacija.

## Mikrofrontend

Arhitektonski pristup u razvoju veb aplikacija gde se aplikacija dekomponuje na manje, autonomne delove (mikrofrontendi) koji mogu da se razvijaju, testiraju, implementiraju i održavaju nezavisno. Ovi mikrofrontendi predstavljaju manje delove korisničkog interfejsa koji rade zajedno kako bi činili celinu aplikacije.

Svaki mikrofrontend je samostalan modul ili aplikacija koji je fokusiran na specifičnu funkcionalnost ili deo korisničkog interfejsa. Oni mogu biti razvijani od strane različitih timova ili pojedinaca, a zatim integrisani u glavnu aplikaciju. Ovaj pristup omogućava agilnost u razvoju, ubrzava implementaciju novih funkcionalnosti i olakšava održavanje kompleksnih veb aplikacija.

Prednosti mikrofrontend arhitekture uključuju:

* Svaki mikrofrontend može da bude razvijan od strane nezavisnog tima, omogućavajući paralelni razvoj i ubrzanje vremena do implementacije.
* Svaki mikrofrontend može da se implementira, testira i održava nezavisno, što olakšava upravljanje kodom i brže isporučivanje promena.
* Omogućava skalabilnost na nivou komponenata, gde se resursi mogu optimalno koristiti prema zahtevima svake komponente.
* Mikrofrontendi su modularni i mogu se koristiti u različitim delovima aplikacije ili u različitim aplikacijama, što doprinosi ponovnoj upotrebi koda.
* Moguće je optimizovati učitavanje stranica, jer se samo potrebni mikrofrontendi učitavaju za određenu stranicu.

Važno je naglasiti da, iako mikrofrontendi donose brojne prednosti, njihova implementacija zahteva pažljivo planiranje, upravljanje stanjem aplikacije i pravilno upravljanje komunikacijom između mikrofrontenda i glavne aplikacije.

## Mikrofrontend vs. Standardni program

Razlika između mikrofrontend i standardnog (monolitnog) pristupa programiranju leži u tome kako se veb aplikacije razvijaju, organizuju i održavaju. Evo nekoliko ključnih tačaka koje ilustruju razlike između ovih dva pristupa:

1. Arhitektura:

* Mikrofrontend arhitektura se zasniva na dekompoziciji aplikacije na manje, autonomne mikrofrontende. Svaki mikrofrontend je nezavisan modul ili aplikacija koji može da se razvija, testira i implementira nezavisno. Ovo omogućava paralelni razvoj i bržu implementaciju novih funkcionalnosti.
* Monolitna arhitektura podrazumeva razvoj celokupne aplikacije kao jedne celine, gde svi delovi i funkcionalnosti dele istu bazu koda. Ovaj pristup može postati kompleksan i težak za održavanje kako aplikacija raste.

1. Nezavisnost:

* Mikrofrontendi su nezavisni i mogu koristiti različite tehnologije, okvire ili biblioteke. To omogućava različitim timovima da rade na različitim mikrofrontendima sa svojim preferiranim tehnologijama.
* U monolitnoj arhitekturi, svi delovi aplikacije dele isti tehnološki stek, što može ograničiti izbor tehnologija i otežati rad različitim timovima.

1. Skalabilnost:

* Mikrofrontendi omogućavaju skalabilnost na nivou komponenata. Možete skalirati samo određene delove aplikacije koji zahtevaju više resursa ili brže odzive.
* U monolitnoj arhitekturi, skaliranje znači skaliranje cele aplikacije, čak iako neki delovi ne zahtevaju dodatne resurse.

1. Učitavanje i performanse:

* Mikrofrontendi omogućavaju dinamičko učitavanje samo onih delova aplikacije koji su potrebni za određenu stranicu ili funkcionalnost, što može poboljšati performanse i brzinu učitavanja.
* U monolitnoj arhitekturi, cela aplikacija se učitava pri svakom zahtevu, bez obzira na to koji deo korisniku trenutno treba.

1. Održavanje:

* Održavanje mikrofrontend aplikacije je olakšano jer se svaki mikrofrontend može ažurirati nezavisno. Ovo smanjuje rizik od narušavanja funkcionalnosti drugih delova aplikacije.
* U monolitnoj arhitekturi, promene u jednom delu aplikacije mogu imati nepredviđene posledice na ostale delove, što otežava održavanje.

Izbor između mikrofrontend i standardnog programa zavisi od specifičnih zahteva projekta, kompleksnosti aplikacije i razvojnog tima. Svaki pristup ima svoje prednosti i nedostatke, i važno je pažljivo razmotriti koji će odgovarati konkretnom projektu.

## Single-spa

JavaScript okvir (*framework*) otvorenog koda koji omogućava razvoj mikrofrontend arhitekture. Ovaj okvir omogućava integraciju različitih mikrofrontenda u jednu koherentnu veb aplikaciju, omogućavajući im da rade zajedno kao deo istog korisničkog interfejsa.

Koncept Single Spa-a se fokusira na to da svaki mikrofrontend može biti razvijen nezavisno, koristeći različite tehnologije i okvire, i da se integriše u glavnu aplikaciju bez međusobnih konflikata. Single Spa omogućava dinamičko učitavanje mikrofrontenda samo kada su potrebni, čime se smanjuje početno vreme učitavanja aplikacije i poboljšava performanse.

Ključne funkcionalnosti Single Spa-a uključuju:

* Omogućava integrisanje mikrofrontenda u glavnu aplikaciju, bez obzira na to koji je tehnološki stek koristio svaki od njih.
* Mogućnost dinamičkog učitavanja mikrofrontenda prema potrebi, što doprinosi efikasnosti i brzini učitavanja.
* Pruža mehanizam za upravljanje rutama i navigacijom unutar aplikacije kroz različite mikrofrontend delove.
* Omogućava deljenje globalnog stanja između mikrofrontenda kako bi se sačuvala koherentnost aplikacije.
* Pruža metode i događaje za upravljanje životnim ciklusom komponenata, kao što su montiranje, demontiranje i ažuriranje.

Single Spa je popularan alat u svetu razvoja mikrofrontend arhitekture jer omogućava organizaciju i integraciju više mikrofrontenda u jednu aplikaciju sa minimalnim naporima i smanjenjem kompleksnosti.

## Role-based access control (RBAC)

Ovaj model se koristi za upravljanje pravima pristupa korisnicima na osnovu njihovih uloga u organizaciji. Svaka uloga ima *set* dozvoljenih akcija i aktivnosti koje korisnici koji imaju tu ulogu mogu da obavljaju u sistemu.

Osnovne komponente RBAC modela uključuju:

* Korisnici: Osobe, sistemi ili procesi koji zahtevaju pristup informacionom sistemu.
* Uloge: Skupovi prava koji određuju tipične zadatke ili funkcije u organizaciji. Uloge grupišu korisnike na osnovu sličnih potreba za pristupom.
* Dozvole: Prava koja se dodeljuju ulogama, određujući šta korisnici sa određenom ulogom mogu ili ne mogu da rade u sistemu.

Prednosti RBAC-a uključuju jednostavno upravljanje pravima pristupa, lakše održavanje sistema, bolju kontrolu nad sigurnošću i bolje razumevanje sistema u celini. Pored toga, RBAC može značajno doprineti sigurnosti sistema ograničavanjem pristupa osetljivim informacijama samo na osnovu uloga i ovlašćenja koje te uloge imaju.

U implementaciji RBAC-a, važno je pažljivo definisati uloge, njihove ovlašćenja i način dodeljivanja uloga korisnicima kako bi se postigla efikasna i sigurna kontrola pristupa u informacionim sistemima.

## HTTP (Hypertext Transfer Protocol)

Protokol koji se koristi za prenos informacija ( komunikaciju između veb servera i veb klijenata ) na *World Wide Webu* (*WWW*).

Funkcioniše na osnovu klijent-server arhitekture, gde klijent šalje HTTP zahteve (*request*) serveru, a server odgovara na te zahteve. Zahtevi uključuju različite metode, kao što su GET, POST, PUT, DELETE, koje definišu željenu akciju nad resursima na serveru.

Osnovne karakteristike HTTP-a uključuju:

* HTTP je *stateless*, što znači da svaki zahtev koji klijent šalje serveru sadrži sve potrebne informacije i ne zavisi od prethodnih zahteva. Ovo pojednostavljuje proces obrade zahteva na serveru.
* Klijenti šalju HTTP zahteve (*request*) serveru, a server odgovara na te zahteve HTTP odgovorima (*response*), koji sadrže informacije ili tražene resurse.
* HTTP podržava različite metode, uključujući GET (čitanje), POST (slanje podataka), PUT (ažuriranje podataka) i DELETE (brisanje podataka), koje se koriste za manipulaciju resursima.
* URI (*Uniform Resource Identifier*) se koristi za identifikaciju resursa (npr. veb stranica, slika, dokument) na internetu.
* HTTP zaglavlja (*headers*) omogućavaju prenos dodatnih informacija između klijenta i servera, kao što su informacije o tipu sadržaja, sesiji, kolačići i druge relevantne informacije.

HTTP je osnova za komunikaciju na internetu i omogućava pristup veb sadržaju, slanje podataka na servere, preuzimanje resursa i obavljanje različitih interakcija na *World Wide Webu*.

## REST (Representational State Transfer)

Predstavlja arhitektonski stil za dizajniranje distribuiranih sistema, a posebno se često koristi u razvoju web servisa. Ovaj koncept se oslanja na prenos podataka putem HTTP protokola i ima za cilj da obezbedi skalabilnost, pouzdanost, performanse i laku upotrebljivost sistema.

Ključne karakteristike REST arhitekture uključuju:

* Resursi (*Resources*): predstavljaju ključne koncepte u REST arhitekturi. Svaki resurs je identifikovan jedinstvenim URI (*Uniform Resource Identifier*). Na primer, u veb razvoju, resursi mogu biti entiteti kao što su korisnici, proizvodi ili narudžbine.
* HTTP Metode (*Methods*): GET, POST, PUT i DELETE koriste se za izvršavanje operacija nad resursima. Na primer, GET se koristi za čitanje podataka resursa, POST za kreiranje novog resursa, PUT za ažuriranje postojećeg resursa i DELETE za brisanje resursa.
* Reprezentacija (*Representation*): Resursi se mogu predstaviti u različitim formatima, kao što su JSON, XML ili HTML, u zavisnosti od potreba aplikacije i klijenta.
* Bez stanja (*Statelessness*): REST je *stateless* što znači da svaki zahtev od klijenta serveru mora sadržati sve potrebne informacije za obradu tog zahteva, bez oslanjanja na prethodne zahteve.
* Jednostavan interfejs (*Uniform Interface*): REST ima jednostavan i konzistentan interfejs između klijenta i servera, što olakšava razvoj, održavanje i skaliranje aplikacija.
* Hipermediji (*Hypermedia*): REST može da koristi hipermediji (npr. linkovi) da omogući navigaciju kroz aplikaciju i informacije o resursima.

REST se često koristi za izradu API-ja (*Application Programming Interface*) koji omogućava komunikaciju između različitih sistema i servisa na internetu. Ovo je popularan pristup u veb razvoju zbog svoje jednostavnosti, skalabilnosti i široke primenljivosti.

# Specifikacija zahteva