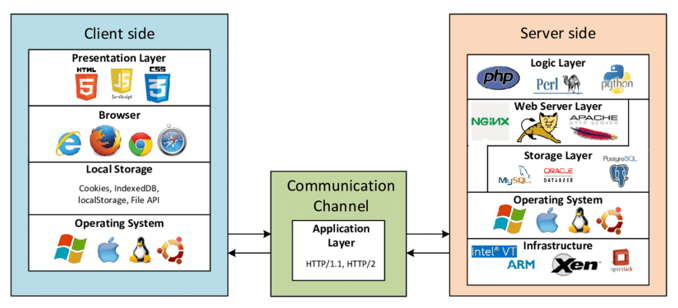
Najčešća pitanja iz Web programiranja:

1. **WWW - osnove (HTTP, HTML, URL)**
   * + World Wide Web, ili samo Web je informacioni sistem koji služi za jednostavno deljenje dokumenata i drugih resursa preko Interneta.
     + URL (Uniform Resourse Locator) – služi za identifikovanje resursa Web-a.
     + HTML (Hyper Text MarkUp Language) – služi za ulančavanje većeg broja URL-ova i njihovo grupisanje za lakši pristup.
     + HTTP (Hypertext Transfer Protocol) – služi za deljenje (transfer) resursa Web-a preko interneta između korisnika Web-a.
2. **Web aplikacije**
   * Web aplikacija je softverska aplikacija pokrenuta na web serveru, a korisnici aplikacije pristupaju aplikaciji uz pomoć pretraživača.
   * Bazirana na klijent-server modelu kod koje se klijent (korisnički interfejs) izvršava u njegovom web pretraživaču. Upravo ova činjenica je prednost web aplikacija jer developer ne mora da se brine o podršci za svaku arhitekturi i operativni sistem, već samo za web pretraživač(e) koji postaju sve fleksibilniji, tako da je dostupnost sve više zagarantovana.
3. **Desktop vs mobilne vs web aplikacija**
   * Desktop:
     + Zavisi od platforme
     + Ne zahteva internet konekciju
     + Zahteva instalaciju
     + Update zahteva ponovno instaliranje
     + Omogućava visoke multimedijalne performanse
     + Komplikovano dodavanje multimedijalnog sadržaja
   * Web:
     + Ne zavisi od platforme
     + Zahteva Internet koncekciju
     + Nije potrebna instalacija
     + Update je jednostavan refresh
     + Limitiran GUI (performanse)
     + Jednostavno dodavanje multimedijalnog sadržaja
   * Mobile:
     + Zavisi od platforme
     + Zahteva instalaciju
     + Update zahteva ponovno instaliranje
     + Verzije su uklopljenje u okruženje (OS) tako da pružaju dobre performanse
     + Veći nivo personalizacije i iskorišćenja arhitekture uređaja
4. **Arhitektura Web aplikacija (dijagram, cloud)**
   * Arhitektura web aplikacije se može razložiti na tri segmenta:
     + Client side (Front-end)
       - Prezentacioni nivo: HTML, CSS, JS
       - Pretraživač: Chrome, Firefox, Safari
       - Keširanje podataka servera: Cookies, File API, localStorage...
       - Operativni sistem: Windows, MAC, Linux, IOS, Android...
     + Communication channel (Komunikacioni kanal)
       - Aplikativni nivo (HTTP/1.1, HTTP/2...)
     + Server side (Back-end)
       - Logika: PHP, Node.js, Perl, Python, ASP .NET Core
       - Web serveri: Nginx, Apache, IIS...
       - Skladištenje: Relacione baze, NoSQL, graf, in-memory, ORACLE, MySQL...
       - Operativni sistem
       - Infrastruktura: Intel, ARM...
   * Cloud (cloud computing / računarstvo u oblaku) predstavlja deljenje (distribuciju) kompjuterskih resursa pri zahtevu korisnika, bez njegovog učešća u održavanju istih. Ovo su u stvari kompjuterski centri (serveri) koji uz pomoć Interneta omogućavaju korisnicima da im pri slanju zahteva pristupaju i koriste njihove resurse. Deljenje memorije (cloud storage), deljenje snage računanja (computing power), itd...
5. **Klijent strana (tehnologije, izazovi)**
   * Klijentska strana se odvija u web pretraživaču. Za njenu izgradnju koriste se: HTML (struktura i sadržaj), CSS (izgled) i JavaScript (ponašanje).
   * Progressive enchancement – koriste se posebni fajlovi za svaku od tehnologija. Princip izrada klijentske strane gde je fokus pre svega na osnovu (HTML) da bude dostupan i pregledan na što je moguće većem broju uređaja, a onda se dalje vrši nadogradnja te osnove (kićenje) izgledom (CSS) i dalje responsiveness (JavaScript) koji je podržan od manjeg broja uređaja. Suprotan pristup bi bio izgradnja složene strane, a onda uslovljavanje iste da pri korišćenju različitih uređaja isključi neke od funkcionalnosti.
   * Izazovi:
     + UX (korisničko iskustvo)
     + UI – korisnički interfejs
     + Performanse
     + Pružanje usluga svakom potencijalnom korisniku (cross-browser, cross-platform, rezolucija ekrana...)
6. **Server strana (tehnologije, izazovi)**
   * Tehnologije:
     1. Logika: PHP, Node.js, Perl, Python, ASP .NET Core
     2. Web serveri: Nginx, Apache, IIS...
     3. Skladištenje: Relacione baze, NoSQL, graf, in-memory, ORACLE, MySQL...
   * Izazovi:
     1. Skalabilnost (ukoliko preko noći eksplodira popularnost i svi navale na sajt, koriste se kluster, load balanseri koji prave još instance web servera kako bi se pružila svima usloga)
     2. Sigurnost (autorizacija, autentifikacija, sql injectin, DDOS)
     3. Prezentacija podataka
7. **Protokoli (HTTP 1.0, 1.1, 2.0, Websocket)**
   * HTTP – (port 80) Hypertext Transfer Protocol (1.0) gde server ne pamti korisnika, odnosno radi se o serveru sesije
   * HTTPS – (port 443) bidirekciona enkripcija uz pomoć Transport Layer Security / Secure Socket Layer koji omogućavaju izdavanje sertifikata preko trećeg lica (distribucija ključeva) – na osnovu ovoga znamo ko su osobe uključene u komunikaciji
   * HTTP 1.1 – perzistentne konekcije, pipelining (zadavanje više zahteva jedan za drugim), multipleksiranje
   * HTTP 2.0 – omogućeno je serveru da izvršava push (standardni http je pull protocol)
   * Websocket – omogućavaju full-duplex (obostrano slanje poruka client=>server i server=>client u istom trenutku) preko jedne TCP konekcije, kao i slanje sadržaja klientu bez prethodnih zahteva.
8. **HTTP - opis, metode zahteva, poruke odgovora**
   * Hypertext Transfer Protocol je protokol aplikativnog nivoa i koristi TCP port 80 (HTTPS je na portu 443), koristi se za razmenjivanje podataka, nebitno kog tipa preko Intrerneta.
   * Metode:
     + GET – koristi se kao zahtev za podatke, ili preciziranog resursa na serveru
     + HEAD – identičan GET methodu, ali odgovor na head NE SME da sadrži body

OVE DVE METODE JE U OBAVEZI DA IMPLEMENTIRA SVAKI WEB SERVER I ONE NIKAD NE VRAĆAJU GREŠKE POPUT (405 - Method Not Allowed, ili 501 – Not Implemented)

* + - POST – zahtev koji obično u sebi sadrži i podatke (unutar body-a) kako bi se prosledile servu i on obavio njihovo procesiranje, eventualno njihovo skladištenje.
    - PUT – zahtev koji kreira novi resurs, ili zamenjuje već postojeći resurs podacima koji se nalaze u body-ju zahteva. Ovaj zahtev se razlikuje od POST zahteva jer je IDEMPOTENTANT (koliko god puta primenjen, ishod je isti kao poziv prvog puta)
  + Odgovori:
    - 1xx – Informacioni odgovori
    - 2xx – Uspešno primljeni obrađeni zahtevi
    - 3xx – Redirekcije
    - 4xx – Greške na klijentskoj strani
    - 5xx – Greške na serverskoj strani

1. **HTML, opšte, struktura strane**
   * HTML – Hypertext Markup Language je standardni jezik za obeležavanje koji se koristi za kreiranje strukture web stranica.
   * Semantički opisuje strukturu web strane.
   * Osnovna struktura HTML strane:
     + DOCTYPE – informacije o dokumentu (npr verzija) pravilima XML-a
     + Zagravlje (head) – Metapodaci, stylesheet
     + Telo (body) – sadržaj, skup html elemenata (script na kraju)
   * Primer:

<!DOCTYPE html>

<html>

    <head>

        <meta>

        <link rel=stylesheet href=stil.css>

    </head>

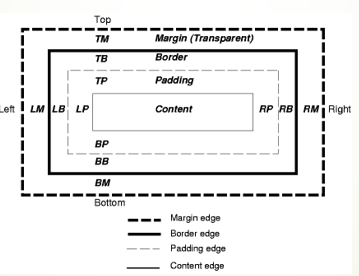
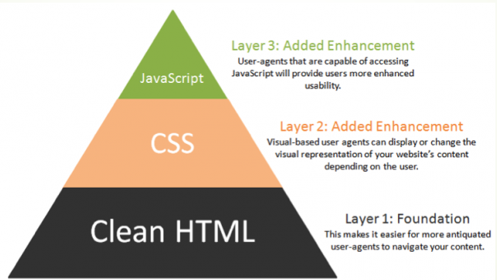
    <body>

        <script src=main.js></script>

    </body>

</html>

* + HTML pisanje elemenata:
    - Elementi sa početnim tagom i (anti-tag-om) krajnjim tagom (<div></div>, itd...)
    - Elementi samo sa početnim tagom kome je pridružen / nakon tipa taga (<br />, ...)
  + HTML prikaz elemenata:
    - Neposredni (inline, text-level) – ne počinju novim redom, već se samo nadovezuju i mogu unutar sebe da sadrže samo inline elemente
    - Blokovni (block-level) – počinju novim redom i mogu u sebi da sadrže i block i inline elemente

1. **HTML5, semantički elementi** 
   * Header, Footer, Section, Article, Aside, Nav
   * Samostalni deo strane ili aplikacije koji može da se distribuira i koristi
   * Ovi elementi nemaju nikakav specifičan izgled od <div> elementa, već pomažu parserima, pretraživačima da prepoznaju o kojim segmentima strane se radi.
   * Article => RSS (RDF (Resource Description Framework) Site Summary / Really Simple Syndication) unos
   * Section => Međusobno povezan sadržaj
   * Div => Ne postoji semantička veza između elemenata
   * Aside => Odnosi se na article (može biti unutar istog), ali prikaz može biti negde sa strane
   * Nav => Koristi se za izlistavanje navigacionih linkova između strana. Jedan nav, sa glavnim navigacionim blokovima
   * Footer => Copyright linkovi
2. **CSS, opšte**
   * Cascading Style Sheets je jezik za formiranje koji omogućava definisanje izgleda bilo kog XML dokumenta. Uz HTML i JavaScript predstavlja temeljnu tehnologiju za izgradnju web stranice.
   * Osnovne kategorije CSS objekta:
     + Border
     + Margin
     + Padding
     + Font
     + Background
   * Deklarisanje stilova:
     + Postoje tri načina povezivanja stilova (CSS-a) i HTML-a
       - U style atributu uokviru HTML elementa
       - U <style></style> elementu uokviru HTML strane
       - U eksternom CSS fajlu (<link rel=stylesheet href=DST />)
3. **CSS selektori**
   * CSS selektori su šabloni koji se koriste za izbor određenih elemenata u XML dokumentu koje biramo kako bi im menjali izgled.
   * Postoje više načina za selekciju elemenata:
     + Selektijom tipa (\_XLM\_Tip\_Elementa\_{})
     + Selekcijom klasa (.\_Ime\_Klase\_{})
     + Selekcijom identifikatora (#\_Ime\_Atributa\_{})
     + Selekcijom HTML atributa ([\_Ime\_Atributa\_='\_Vrednost\_Atributa\_'])
       - \*= => Sadrži
       - ^= => Počinje sa
       - $= => Završava se sa
     + Relacionim operatorima (+, ~, :, itd)
     + Kombinacijom navedenih pravila
4. **Progresivno poboljšanje** 
   * Progressive enchancement – koriste se posebni fajlovi za svaku od tehnologija. Princip izrada klijentske strane gde je fokus pre svega na osnovu (HTML - markup) da bude dostupan i pregledan na što je moguće većem broju uređaja, a onda se dalje vrši nadogradnja te osnove (kićenje) izgledom (CSS) i dalje responsiveness (JavaScript) koji je podržan od manjeg broja uređaja. Suprotan pristup bi bio izgradnja složene strane, a onda uslovljavanje iste da pri korišćenju različitih uređaja isključi neke od funkcionalnosti.
   * Osnovni sadržaj i funkcionalnost treba da rade u svim pretraživačima
   * Semantičko obeležavanjem celog sadržaja
   * Izgled i raspored preko CSS-a
   * Ponašanje definisano preko JS
   * Uzimanje u obzir korisničkih podešavanja definisanih u pretraživaču
5. **Dizajn strane - css grid, flexbox**
   * CSS grid layout module omogućava grid zasnovano uređenje elemenata, sa redovima i kolonama, što znatno omogućava dizajniranje i manipulisanje pozicijama elemenata na stranici
   * Display property omogućava izbor načina kojim se manipuliše grid-om, jedan od njih je flex
   * Flexbox (flexible box) omogućava korišćenje dodatnih properties-a koje pružaju organizaciju elemenata:
     + Justify-content
     + Align-items
     + Align-content
     + Flex-Flow (flex-direction, flex-wrap)
     + Flex (Flex-grow, flex-shrink, flex-basis)
6. **Responsive dizajn**
   * Responsive design je pristup web dizajniranju koji omogućava web stranicama da se prilagođavaju uređajima koji im pristupaju, odnosno njihovim ekranima i veličinama prozora. Uz pomoć CSS-u, fluid grid-ova zasnovanim na proporcionalnosti (koriste procenat umesto konkretne vredosti za dimenziju elemenata) i media queries, vrlo je lako osposobiti website da prepozna okruženje u kome se prikazuje i tom okruženju se prilagodi, kako bi sadržaj bio vidljiv, povezan, razumljiv, čitljiv....
7. **Javascript (JS), opšte**
   * Javascript je skriptni, dinamički, netipizirani, prototipski jezik visokog nivoa, sa ECMAScript specifikacijama.
   * Prototipski – stil objektno orijentisanog programiranja gde se nasleđivanje vrši kloniranjem postojećih objekata, koji služe kao prototipi
   * Skriptni – Koristi interpretator (ne kompajler)
   * Dinamički – Tokom izvršavanja se vrši interpretacija koda
   * Netipizirani – Nema jake tipove (const, let i var su jedini tipovi promenljivih koji postoje)
   * Sa HTML-om i CSS-om, ovo je jedna od osnovnih tehnologija WWW-a. Omogućava interaktivne strane i osnovni je deo web aplikacija.
   * Uglavnom je korišćen za izgradnju klijentske strane website-a
   * Omogućava:
     + Dinamičko menjanje sadržaja HTML dokumenta, kao i primenu CSS stilova
     + Validaciju korisničkih podataka (pre slanja na server)
     + Obradu događaja koji se odnose na dokument
     + Komunikaciju sa web serverom
   * JavaScript se može vezati za HTML dokument na dva načina:
     + Intrerno (unutar <script></script> elementa)
     + Eksterno (<script src=\_\_\_ type=\_\_\_>< /script>)

**JS** je univerzalni jezik klijent strane. Script se izvršava na klijentskoj mašini pošto se dokument učitava ili u nekom drugom trenutku (npr. kada se aktivira link). Skripte pružaju autoru sredstva za proširenje funkcionalnosti i interaktivnosti HTML dokumenta:

* dinamička promena sadržaja dokumenta
* validacija korisničkih podataka u obrascu, kako bi se detektovale greške pre slanja podataka serveru
* komunikacija sa serverom (AJAX, Web sockets)
* obrada događaja koji se odnose na dokument kao što su učitavanje i zatvaranje dokumenta, pokreti kursora miša, fokus elementa, itd.

JS je najpopularniji jezik Web-a, zastupljen na klijentu (Web browser-u), na serveru (Node.js), ali i van web-a: PDF, desktop widget-i.

Klijentska strana JS-a proširuje jezik dodavajući objekte za kontrolu browsera i njegovog Document Object Modela (DOM). Ekstenzija klijentske strane dozoljava aplikaciji da postavlja elemente na HTML formu i da odgovori na evente korisnika poput click-a, input-a i navigacije stranice.

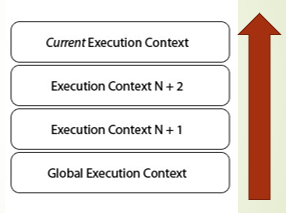
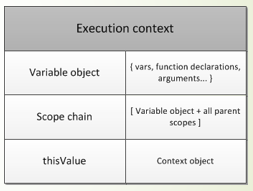
Serverska strana JS-a proširuje jezik dodavajući objekte relevantne pokretanju JavaScript-a na serveru. Ekstenzija serverske strane dozvoljava komunikaciju aplikacije sa bazom podataka, pruža kontinuitet informacija od jednog pokretanja aplikacije do sledećeg, ili izvršava manipulaciju fajlovima na serveru.

Standardizovan je ECMA Script specifikacijom. JS je prototipski skrip jezik dinamičkih tipova.

* Prototipski bazirano programiranje - stil OOP gde se nasleđivanje postiže kloniranjem postojećih objekata, koji služe kao prototip

Postoje 3 tipa deklaracije u JS-u:

* **var** Globalne ili lokalne promenljive.
* **let** Block-scope(validna je unutar bloka).
* **const** Block-scope(validna je unutar bloka), read-only.
* Dinamički tipovi: ista promenljiva može da menja tip podatka.

1. **Funkcije**
   * Funkcije u JavaScript-u mogu biti definisane na više načina:
     + Function declaration
     + Function Expression (FE)
     + Arrow Functions (x=>x\*x)
     + IIFE (Immediately Invoked Function Expression)
     + Lambda functions
2. **Razlika između deklaracije i izraza fje (FE)**
   * Glavna razlika između funkcijskog izraza i deklaracije funkcije u JavaScript-u se javlja zbog mehanizma Hoisting, naime, ukoliko deklarišemo funkciju, mi tu funkciju možemo pozivati bilo gde u skripti, pa čak i pre njene definicije, baš zbog mehanizma Hoisting, koji sve deklaracije diže na vrh poziva i ti su naredbe koje se prvo izvršavaju. Dok ukoliko koristimo funkcijski izraz, mi taj izraz pridružujemo promenljivoj, odnosno dodeljujemo ga. Dodele se izvršavaju u drugom prolazu, nakon deklaracija, tako da je funkcijske izraze dodeljene promenljivama moguće pozivati samo nakon njihovog definisanja.
   * Ostale razlike su te da je funkcijski izraz definisan svojom promenljivom i anonimnom funkcijom koja se dodeljuje toj promenljivoj (Anonimne funkcije nemaju naziv).
3. **Hoisting**
   * Hoisting je mehanizam koji definiše ponašanje u ECMAScript jezicima. Hoisting omogućava da pri interpretiranju koda se deklaracije izvršavaju među prvim naredbama, bez obzira na to u kom delu koda su napisane, tako da omogućava izvršavanje funkcije pre njihove deklaracije.
4. **Lambda fje**
   * Lambda funkcije su jedna od vrsta funkcija u JavaScript-u, a prepoznaju se po tome što su argumenti drugih funkcija. Bilo koja funkcija može biti lambda funkcija, ukoliko se prosleđuje nekoj drugoj funkciji kao argument.
5. **Closure i konteksti izvršenja funkcija**
   * Closures (Zatvaranja/Pregrađivanja) jeste kombinovanje funkcija tako da se definicija jedne funckije (unutrašnja) nalazi u definiciji druge funkcije (spoljašnja).
   * Ovakva kombinacija omogućava unutrašnjoj (ugnježdenoj) funckiji da pristupa globalnim promenljivama (scope-u) spoljne funkcije (ovo funkcioniše iterativno, tako da krajnje ugnježdena funckija ima pristup svim scope-ovima kreiranim pre njenog), naziva se leksičko okruženje.
   * Kontekst izvršenja funkcije (execution stack) jeste mehanizam kojim se kontekst funckije (Variable object, Scope chain i this) stavlja na stack i tu je dostupan svim funckijama koje su takođe pozvane (pridružene stack-u)

let executionContextObj =

{

    'scopeChain':       { /\* variableObject + all parent execution context's variableObject \*/ },

    'variableObject':  {  /\* function arguments / parameters, inner variable and function declarations \*/  },

    'this': {}

}

* + Closure predstavlja unutrašnju f-ju koja ima pristup promenljivama spoljašnje (roditeljske) f-je. Suština je kad se f-ja poziva, ona osim liste parametara napravi i kontekst tj. spisak promenljivih koje f-ja koristi, a koje su definisane izvan nje. Postoji 1 globalni kontekst i više njih za f-je. Svaki poziv f-je kreira novi kontekst. Closure ima 3 lanca opsega (scope chain):
    - ima pristup sopstvenom opsegu (promenljivama definisamnim unutar {}
    - ima pristup promenljivama i parametrima roditeljkse f-je
    - ima pristup globalnim promenljivama

1. **IIFE**
   * Immediately invoked function expression jeste funkcijski izraz u JavaScript-u kome su dodate ˝()˝ i time omogućen neposredan poziv.
   * Omogućava enkapsulaciju (privatne atribute i metode)
   * Aliasing članova (dodavanje drugih imena)
   * Globalni objekat (Zavisi od okruženja u kom je pozvano)
2. **Nizovi**
   * Nizovima (linearnim kolekcijama objekata) u JavaScriptu su pridodate funkcije osmišljene za lakši rad nad njima:
     + Niz.Map(\_lambda funkcija koja ima jedan argument [element niza] i nad njime primenjuje nešto i vraća modifikovanu vrednost\_) – N => N preslikavanje
     + Niz.Filter(\_lambda funkcija koja ima jedan argument [element niza] i kao povratnu vrednost TRUE/FALSE čime ispituje karakteristike elementa\_) – N => M, gde je M <= N
     + Niz.Find(\_lambda funkcija koja ima jedan argument [element niza] i kao povratnu vrednost TRUE/FALSE čime ispituje da li je element traženi ili nije, ukoliko lambda ne vrati TRUE ni za jedan element, vraća null, u protivnom vraća prvi element koji je zadovoljio uslov\_) – N => 1/0
     + Niz.Reduce(\_lambda funkcija koja ima dva argumenta [accumulator, element] i vrši modifikaciju accumulator-a sa elementom niza (suma, proizvod, max, min,...) i vraća izmenjen accumulator\_, \_drugi argument reduce funckije jeste početna vrednost akumulatora\_) – N => 1
3. **Prototipi**
   * Prototip je mehanizam koji koristi JavaScript pri nasleđivanju atributa objekata. Klonira se postojeći objekat i koristi se kao prototip.
   * \_\_proto\_\_ pokazuje na prototip iz kog je objekat kreiran (roditeljska klasa)
   * Konstruktorska funkcija ima property pod nazivom 'prototype' koji se može koristi pri dodavanju novih atributa i metoda već definisanim klasama, a ujednom i već instanciranim objektima tih klasa (dinamičko dodeljivanje). (Ne dodaje se klasama, već upravo tom prototipu (maketi), ali pošto je iz nje kreiran i objekat i klasa, onda oni imaju pristup toj novododatoj metodi/atributu) Uz ovo sve mora priemer u JS da bi bilo jasno, tako da manje proseravanja, više pisanja u JS....
4. **Konstruktori preko IIFE-a**

let OrderNotifier = (function()

{

    //  private stuff would go here

    //  shared by all instances

    return function(/\* constructor args go here \*/) {

        // per-instance private vars here

        return {

            notifyApproved: function(orderId, notifier, recipient)

            {

                // use the notifier object passed as ctor param to send notifications

                // related to orderId to recipients

            },

            notifySomeOtherEvent: function(orderId, notifier, recipient)

            {

                // use the ctor params again

            }

        };

    }

})();

// sometime later in your code

let notifier = new OrderNotifier(/\* args here \*/);

2. **Objekat literali**
   * Lista parova name-value odvojena zarezima i upakovana u vitičaste zagrade (JSON, basically)
     + Primer:

const proba = new Object(***{***

***name: "Stefan",***

***surname: "Aleksic",***

***class: "RII 20/21"***

***});***

console.log(proba);

1. **ES6 Klase (+ primer)**

class Animal

{

    constructor(ime, godine)

    {

        this.ime = ime;

        this.godine = godine;

    }

    ostari()

    { this.godine++; }

    podmladi()

    { this.godine--; }

}

1. **ES6 Nasleđivanje (+ primer)**

class Dog extends Animal

{

    constructor(ime, godine, zvuk)

    {

        super(ime, godine);

        this.zvuk = zvuk;

    }

    laj()

    { console.log(`${ this.ime }: ${ this.zvuk }!`); }

}

class Bichon extends Dog

{

    constructor(ime, godine, zvuk, boja)

    {

        super(ime, godine, zvuk);

        this.boja = boja;

    }

    ofarbaj(boja)

    { this.boja = boja; }

}

class Foo extends Bichon

{

    constructor(ime, godine, zvuk, boja, brzina)

    {

        super(ime, godine, zvuk, boja);

        this.brzina = brzina;

    }

    treniraj(brzina)

    { this.brzina = brzina; }

}

1. **ES5 klase (+ primer)**

function Animal(ime, godine)

{

    this.ime = ime;

    let starost = godine;

    this.getAge = () => starost;

    this.setName = function (ime) { this.ime = ime; };

    this.ostari = function () { starost++; };

    this.podmladi = function () { starost--; };

};

// function Dog(ime, godine, zvuk)

// {

//     this.\_\_proto\_\_ = Animal.prototype;

//     this.\_\_proto\_\_.constructor(ime, godine);

//     let sound = zvuk;

//     this.pustiZvuk = function () { console.log(sound) };

// };

function Dog(ime, godine, zvuk)

{

    this.Animal = Animal;

    this.Animal(ime, godine);

    let sound = zvuk;

    this.pustiZvuk = function () { console.log(sound) };

};

1. **Typescript**
   * **Typescript** je besplatan programski jezik otvorenog koda napravljen od strane Microsoft-a. Typescript je sintaksički nadskup JS-a (ECMAScript 6), pa je svaki validan JS program istovremeno validan Typescript program. Za Typescript se često kaže da je to JS sa tipovima što i jeste istina jer je glavna novina koju Typescript donosi statička tipizacija. Typescript je jezik čiji se izvorni kod sa ekstenzijom .ts kompajlira u JS (ECMAScript 5). To znači da praktično i dalje imamo dinamičku tipizaciju na nivou JS koda dobijenog kompilacijom tj. da se tip svake promenljive određuje u vreme izvršenja, s tim da ovde imamo jedan sloj iznad koji nam pruža prividnu statičku tipizaciju tako što će sva neuklapanja tipova (najčešće kada se funkciji prosleđuju argumenti neodgovarajućeg tipa ili prilikom dodele) biti prijavljenja kao greška (greška u vremenu prevođenja).
   * Ducktyping (ukoliko ima listu properties koja sadrži određene, onda odgovara kao argument)

function max(a: number, b: number): number

{

    return a > b ? a : b;

}

interface Coords

{

    x: number,

    y: number,

    z: number

}

interface CoordsPredicate

{

    (x: Coords, y: Coords): number

}

class Coordinates

{

    x: number;

    public y: number;

    public z: number;

    constructor(x: number, y: number, z: number)

    {

        this.x = x;

        this.y = y;

        this.z = z;

    }

    public getDistance: CoordsPredicate;

    getDotProduct(point: Coords)

    {

        return this.x \* point.x + this.y \* point.y + this.z \* point.z;

    }

}

let PointA = new Coordinates(2.5, 1, -7);

let PointB = new Coordinates(7, -3.5, 2);

console.log(PointA.getDotProduct(PointB));

console.log(PointB.getDotProduct(PointA));

let calc = (point1: Coords, point2: Coords) =>

{

    return Math.sqrt(Math.pow(point1.x - point2.x, 2) + Math.pow(point1.y - point2.y, 2) + Math.pow(point1.z - point2.z, 2));

}

PointA.getDistance = calc;

PointB.getDistance = calc;

console.log(PointA.getDistance(PointA, PointB));

console.log(PointB.getDistance(PointB, PointA));

enum LoveTowardsYou

{

    High = 100,

    Great = 75,

    Alright = 50,

    Faint = 25,

    NonExistant = 0

}

const lOvE = LoveTowardsYou.High;

console.log(lOvE);

1. **Typescript klase i nasleđivanje (+ primer)**

export class Person

{

    public name: string;

    public surname: string;

    private ID: string;

    public ig: string;

    private container: HTMLElement;

    public constructor(name: string, surname: string, ID: string, ig: string)

    {

        this.name = name;

        this.surname = surname;

        this.setID(ID);

        this.ig = ig;

    }

    private setID(id: string): void

    {

        this.ID = id;

    }

    public getID = (): string => this.ID;

    public print(canvas: HTMLElement)

    {

        this.container = document.createElement("div");

        this.container.classList.add("Person");

        canvas.appendChild(this.container);

        [ [ "Name", this.name ],

  [ "Surname", this.surname ],

[ "ID", this.ID ],

  [ "Instagram", this.ig ] ].forEach(el => {

            const label = document.createElement("label");

            label.htmlFor = el[ 0 ];

            label.innerHTML = el[ 1 ];

            this.container.appendChild(label);

        });

    };

}

export class Student extends Person

{

    private indexID: string;

    public year: number;

    public constructor(name: string, surname: string, ID: string, ig: string, indexID: string, year: number)

    {

        super(name, surname, ID, ig);

        this.setIndexID(indexID);

        this.year = year;

    }

    private setIndexID(indexID: string)

    {

        this.indexID = indexID;

    }

    public getIndexID = (): string => this.indexID;

    public print(canvas: HTMLElement)

   {

        super.print(canvas);

        const studentInfo = document.createElement("div");

        studentInfo.classList.add("Student");

        [ [ "Index", this.indexID ], [ "Year", this.year ] ].forEach(el =>

        {

            const label = document.createElement("label");

            label.htmlFor = el[ 0 ].toString();

            label.innerHTML = el[ 1 ].toString();

            studentInfo.appendChild(label);

        });

    }

}

1. **Typescript nasleđivanje it is up, it is up it is up and it's stuck (+ primer)**
2. **Typescript šabloni (+ primer)**

import { Student } from "./classes";

import { Person } from "./classes";

export class Community<T extends Person>

{

    private people: T[];

    private name: string;

    private container: HTMLElement;

    constructor(name: string)

    { this.setName(name); }

    private setName = (name: string): void => { this.name = name };

    public getName = (): string => this.name;

    public addMember(person: T)

    { this.people.push(person); };

    public printCommunity = (canvas: HTMLElement): void =>

    {

        this.container = document.createElement("div");

        this.container.classList.add("Community");

        canvas.appendChild(this.container);

        const info = document.createElement("label");

        info.htmlFor = "Name";

        info.innerHTML = this.name;

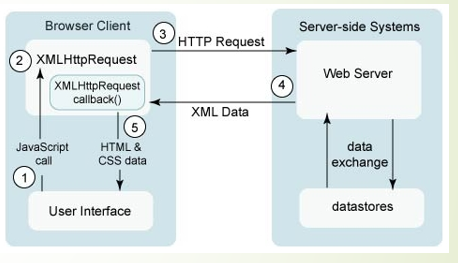
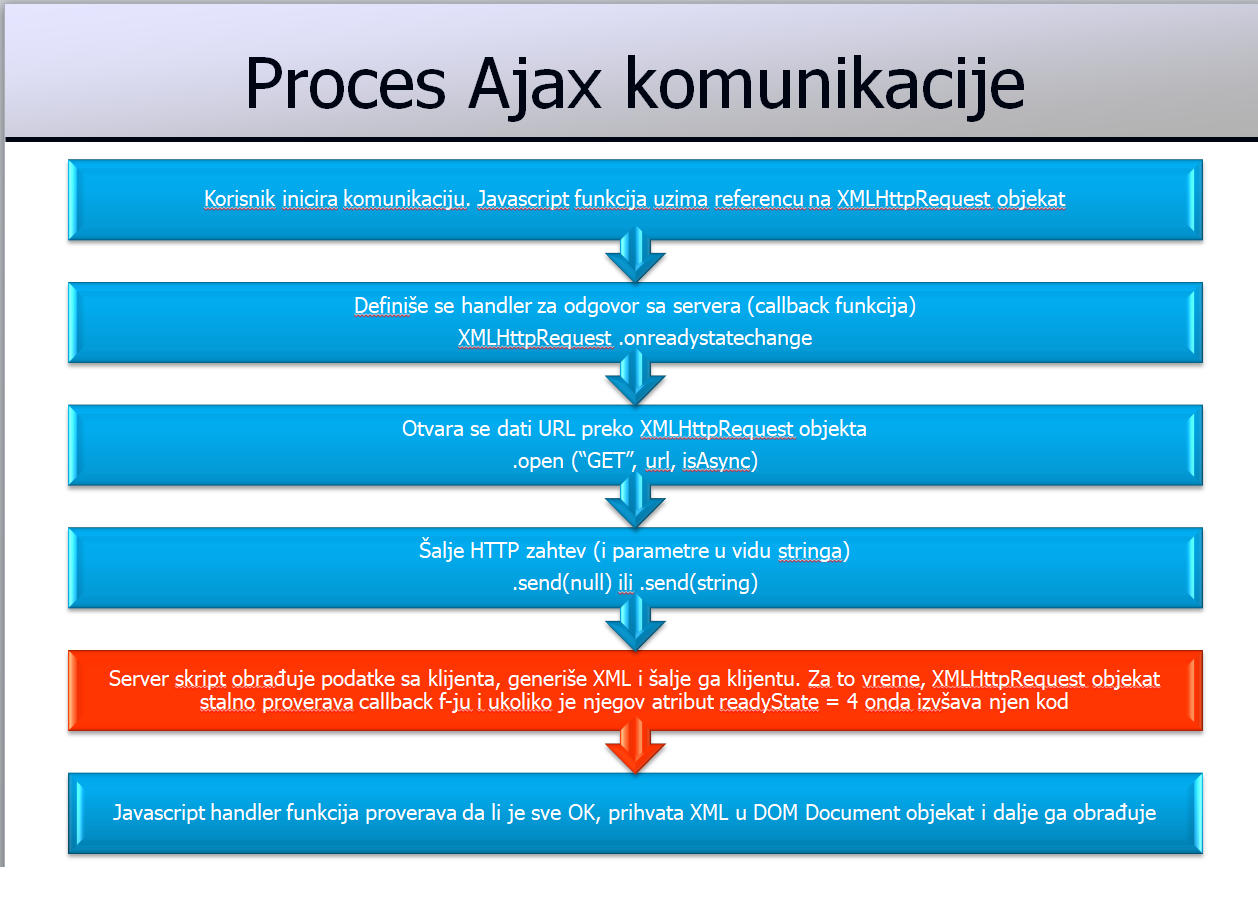
        this.container.appendChild(info);

        this.people.forEach(person =>

        { person.print(this.container) });

    };

}

1. **DOM, definicija i najčešće funkcije API-ja**
   * Dom (Document Object Model) je cross-platform standardni interfejs koji ne zavisi od jezika i koristi se za tretiranje XML dokumenata kao strukturu stabla.
   * Napravljen je od strane W3C konzorcijuma
   * DOM nudi standardni skup objekata za predstavljanje HTML i XML dokumenata, kao i standardni interfejs za njihovu manipulaciju i pristup
   * DOM omogućava univerzalnost za svaki pretraživač koji čita DOM podatke
   * Svaki element kao stablo (Primer jedan html u stablo)
   * Pretraga elemenata:
     + Document.getElementById(˝\_ID\_˝);
     + Docuemnt.getElementsByTagName(„button“);
     + Docuemnt.querySelectorAll(„div[name='hello']“) - CSS
     + Document.querySelector(„input[name='izbor']:checked“) - CSS
2. **Modeli klijent-server komunikacije**
   * Dve vrste modela:
     + Pull – klijent zahteva podatke:
       - **HTTP, HTTPS** (port 80/443)
       - POP3 (port 110)
       - RSS (nema preciziran port)
     + Push – server šalje podatke bez zahteva klijenta (publish/subscribe/notifications) model:
       - Web push – notifikacije i nakon izlaska sa sajta. Agresivne reklamne kampanje
       - HTTP/2 server push – server samoinicijativno šalje podatke klijentu (pretpostavka da su klijentu potrebni ti resursi, css, js, itd). Klijent pristigle resurse čuva u karantin. Pre svakog slanja zahtva za resursom, klijent proverava da li se resurs nalazi u karanitnu
       - Long polling – simulacija push-a ali za zahteve koji se dugo čekaju
       - **Websockets**
       - Server-sent events (HTML5)
3. **Websocket protokol**
   * Komunikacijski protokol aplikativnog nivoa koji pruža punu dupleks bidirekcionu komunikaciju – obe strane simultano šalju svoje poruke.
   * Koristi HTTP za handshake, a TCP za komunikaciju (jednu)
   * Brža komunikacija od HTTP-a (manji overhead)
   * Koristi Upgrade header kako bi HTTP/HTTPS-om naznačio drugoj strani da želi da se pređe na websocket protocol (handshake)
   * Nakon uspostavljanja protokola, vrši se komunikacija sve dok je tcp konekcija uspostavljena
4. **WebRTC**
   * Web Real Time Communication je besplatni, open-source projekat koji omogućava pretraživačima i
   * Omogućava audio i video komunikaciju na web stranama uz pomoć peer-to-peer komunikacije (direktne komunikacije), bez plug-in-a i native aplikacija.
   * Uspostavlja peer-to-peer konekciju između klijenata i nakon toga nije neophodan server za njihovu komunikaciju
   * Problem nastaje ukoliko u komunikaciji učestvuje više od dvoje ljudi, jer se onda uspostavljaju dodatne veze
5. **Prenos podataka sa klijenta na server**
   * Klijent serveru šalje podatke preko HTTProtocol-a i to uz pomoć metoda:
     + GET – querystring je deo URL-a (adresa sadrži podatke) i samim tim je limitaran sadržaj koji se prenosi, u vidu tipo podataka koji se prenose, kao i njihove dužine. Ova metoda ne vrši nikakve izmene na serverskoj strani, već samo (kako ime kaže) nabaljva potrebe podatke sa servera. Potrebno ju je implementirati i nikad ne treba da vrati greške poput (405 - Method Not Allowed, ili 501 – Not Implemented) [https://www.google.com/search?sourceid=chrome&ie=UTF-8&q=web+programming] query string podvučen (? Počinje query string, ali ? nije deo query string-a)
     + HEAD – identična GET metodi, s tim što odgovor servera ne sme da sadrži ništa u body-ju
     + PUT – metoda koja se koristi kada je potrebno izvršiti neko ažuriranje podataka ili kreiranje, slična post-u
     + POST – metoda koja podatke šalje kroz svoj body i uglavnom podrazumeva obradu tih podataka na serveru (kreiranje ili izmenu podataka servera). Slanje veće količine podataka, osetljivih pdoataka, binarnih fajlova (slike, itd), podaci se ne vide u adresi, mada je moguće i da ova metoda doda query string u URL
     + DELETE – methoda koja izvršava neki vid birsanja podataka na serveru
6. **AJAX**
   * Asynchronous JavaScript And XML je skup tehnika za web razvijanje koje koriste mnoge web tehnologije na klijentskoj strani kako bi kreirale asinhrone web aplikacije.
   * AJAX omogućava klijentskoj strani da prima i šalje podatke serveru bez osvežavanja strane, remećenja sadržaja strane, ponašanja itd...
   * Postoji ugrađen XMLHttpRequest objekat (fetch() od 2017.) koji se koristi za izvršenje Ajax-a na web stranama.
   * Format odgovora zavisi od servera i uglavnom je JSON, ne XML
7. **CORS (primeri za same origin)**
   * Cross-Origin Resource Sharing (CORS) je http-header bazirani mehanizam koji dozvoljava serveru da prepozna bilo koji drugi izvor (origin - poreklo) (protocol, host, port) koji nije njegov sopstveni i dozvoli / zabrani pristup resursima / metodama / itd...
   * Primeri za različito poreklo:
     + Adrese koje se razlikuju po protokolu
       - [**https**://localhost:5000/webProg/resurs](https://localhost:5000/webProg/resurs)
       - [**http**://localhost:5000/webProg/resurs](http://localhost:5000/webProg/resurs)
     + Adrese koje se razlikuju po host-u/domenu
       - [https://**gais.**cs.umass.edu:5000/webProg/resurs](https://gais.cs.umass.edu:5000/webProg/resurs)
       - [https://**notgaia.**cs.umas.edu:5000/webProg/resurs](https://notgaia.cs.umas.edu:5000/webProg/resurs)
     + Adrese koje se razlikuju po portu
       - [https://localhost:**5001**/webProg/resurs](https://localhost:5001/webProg/resurs)
       - [https://localhost:**5000**/webProg/resurs](https://localhost:5000/webProg/resurs)
8. **Formati odgovora servera**
   * HTML – nova strana
   * XML – web servisi
   * JSON – web servisi (90%)
   * Protocol buffer (Protobuf) – metod serilizacije strukturisanih podataka. Nije podržan od strane pretraživača, koristi se u mikroservisima
9. **XML tehnologije**
   * Extensible Markup Language je markup jezik (jezik za označavanje) koji služi za enkodiranje dokumenata (podataka/struktura podataka) u xml strukturu koja je čitljiva i mašinama i ljudima. Format odgovora web servisa.
   * Ovo je nadskup HTML-a
   * Modeluje se kao DOM (Document Object Model) stablo
   * Moguće je odrediti definicije kako se elementi grupišu XML Scheme/Document Type Definition
   * Sintaksa:

<?xml version="1.0" encoding="utf-8"?>

<note>

    <to>Tove</to>

    <from>Jani</from>

    <heading>Reminder</heading>

    <body>Don't forget me this weekend!</body>

</note>

1. **XML namespace**
   * XML namespace, kao i generalni namespace se koristi da označi kom setu (namespace-u) pripadaju tipovi objekata koji se koriste. Odnosno odakle da povuče njihovo mesto definisanja.
   * Pisanjem xml-a možemo doći u situaciju gde elementi sadrže isti tag, pa samim tim je potrebno uvesti i kojoj sintaksi (pravilima) svaki od tagova pripada.
   * Defaultni (podrazumevani) namespace je definisan kao ˝ **xlmns='\_\_izvor\_\_'** ˝, a ukoliko želimo označiti i dodati više namespace-a/ova...
   * XML namespace se definiše sa ˝ **xlmns:\_\_ime\_namespacea\_\_='\_\_izvor\_\_'** ˝ i dalje se pridodato ime namespace-a piše ispred naziva taga koji se koristi, kao u primeru:

<root xmlns:h="http://www.w3.org/TR/html4/"

xmlns:f="https://www.w3schools.com/furniture">

<h:table>

  <h:tr>

    <h:td>Apples</h:td>

    <h:td>Bananas</h:td>

  </h:tr>

</h:table>

<f:table>

  <f:name>African Coffee Table</f:name>

  <f:width>80</f:width>

  <f:length>120</f:length>

</f:table>

</root>

1. **XML Schema**
   * Extensible Markup Language Schema (pored Document Type Definition [DTD]) predstavlja jezik kojim se definiše struktura XML dokumenata.
   * Piše se u XML-u
   * Proširiva je
   * Podržava tipove podataka i namespace-ove
   * Moćnija od DTD

<?xml version="1.0"?>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="note">

<xs:complexType>

<xs:sequence>

<xs:element name="to" type="xs:string"/>

<xs:element name="from" type="xs:string"/>

<xs:element name="heading" type="xs:string"/>

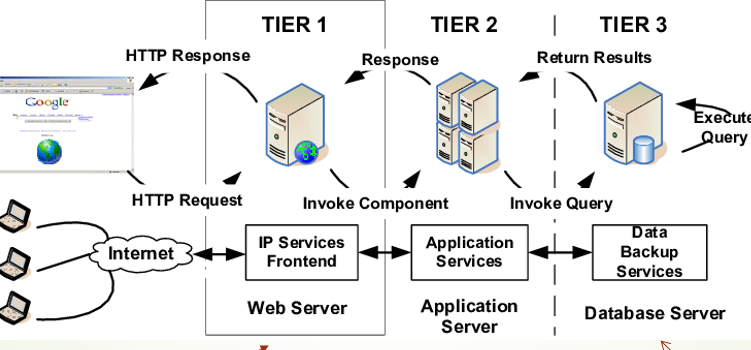
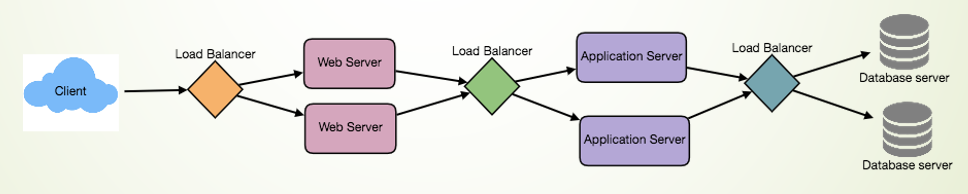
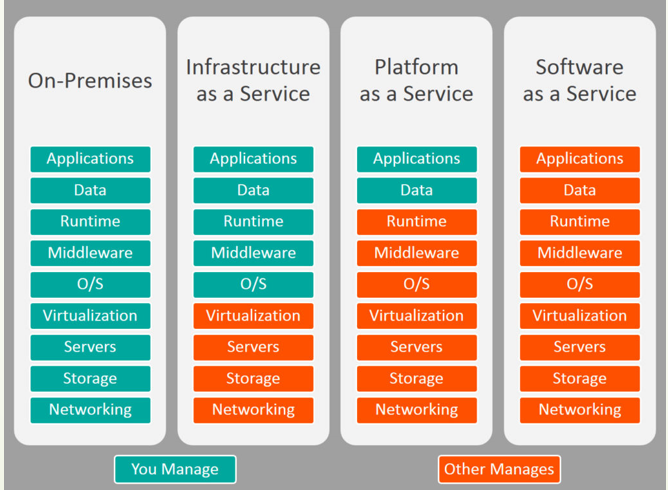
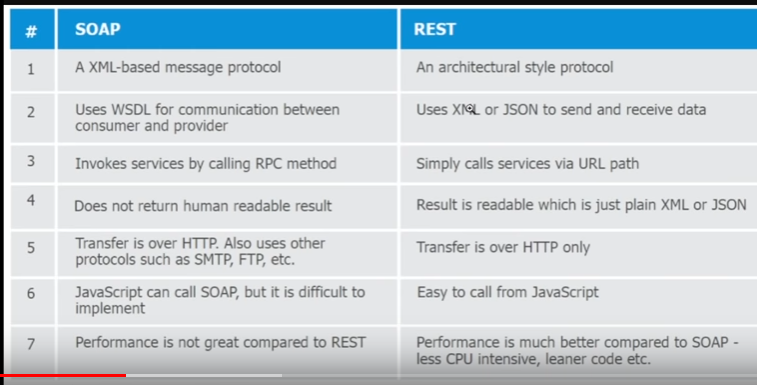
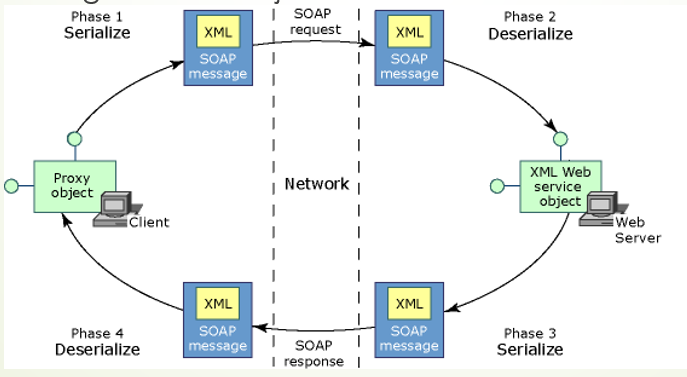
<xs:element name="body" type="xs:string"/>

</xs:sequence>

</xs:complexType>

</xs:element>

</xs:schema>

1. **XPath**
   * XML Path Language je query jezik za odabir elemenata iz XML dokumenata. Takođe se može koristiti za proračun vrednosti iz sadržaja XML dokuemnta.
   * Definisan je od strane W3C (World Wide Web Consortium)
   * Zasniva se na tome da XML ima strukturu stabla, pa se XPath-om selektuju određeni čvorovi u zavisnosti od njihovih karakteristika (atributa), grupacije itd....
   * Ključna komponeneta XSL-a (Extensible Stylesheet Language) – porodica jezika koji se koriste za transformaciju xml-a.
2. **Obrada XML dokumenta**
   * Obrada XML dokumenata sa vrši uz pomoć:
     + DOM (Document Object Model) – a:
       - XML se modeluje kao stablo
       - Apliakcije koriste Dom stablo za manipulaciju XML dokumentom
       - Kompletan XML dokument mora biti učitan u memoriju kako bi se formiralo DOM stablo
     + SAX (Simple ApplicationProgrammingInterface for XML) – a:
       - To je protočni parser (u jednom trenutku postoji samo trenutno pročitani čvor), ne povlači celo xml stablo, već samo deo
       - Baziran je na događajima (detekcija početnog, krajnjeg taga, sadržaja, atributa generiše odgovarajući događaj)
       - Brži i manje zahtevan od DOM-a
       - Pogodan za pretragu podataka u okviru XML dokumenta.
3. **XML vs HTML**
   * XML je nadskup HTML, dakle HTML dokument je validan XML dokument, određen svojom sintaksom <!DOCTYPE html>
   * HTML se koristi za vizualno predstavljanje podataka (web stranice)
   * XML se koristi za skladištenje podataka u strukturu koja je laka za čitanje
4. **XSL**
   * XSL (Extensible Stylesheet Language) je termin koji se odnosi na porodicu jezika koji se koriste za transfomraciju i prikaz XML dokumenata.
   * Može se podeliti na tri dela:
     + XSL Transformation (EXtensible Stylesheet Language Transformation): XML jezik za transformaciju XML dokumenta
     + XSL Formatting Objects (XSL-FO): XML jezik za opisivanje vizualnog formatiranja (predstavljanje) XML dokumenta
     + XML Path Language (XPath): Query jezik koji se koristi za selektovanje delova XML dokumenta.
5. **JSON**
   * JSON (JavaScript Object Notation) je jezički nezavisan format podataka koji se koristi za skladištenje (transfer) podataka koji su tipa atribut-vrednost i nizovnih tipova. Koristi se kao odgovor web servisa.
   * Jednostavan je i kraći od XML-a
   * Prasiranje u JS je trivijalno, jer je JSON validan JS objekat
   * Ne postoji podrška za metapodatke
   * Ne postoje ograničenja u tipovima podataka
   * Nema podršku za namespace
   * Koristi se za serilizaciju podataka/objekata
6. **JSON vs XML**
   * Podaci su dostupni u vidu JSON objekata
   * XML podaci moraju biti parsirani
   * JSON je podržane od većine pretraživača
   * Cross-browser XML parsiranje može biti zahtevno
   * JSON nema mogućnosti vizuelnog predstavljanja podatak
   * XML nudi mogućnosti za vizuelno predstavljanje pdataka jer je markup jezik.
7. **Web server**
   * Web server je serverski (uslužni) program, ili sistem od jednog ili više kompjutera na kojima je pokrenut serverski program koji pruža usluge klijentima korišćenjem protokola aplikativnog nivoa (HTTP 80, HTTPS 443, SMTP 25, POP3 110, DNS 53, DHCP 67/8 itd).
   * Komunikacija se ostvaruje slanjem zahteva i odgovora, odnosno razmenom resursa/podataka/izračunavanja itd...
   * Često korišćeni web serveri:
     + Nginx (najpopularniji)
     + Apache
     + Microsoft IIS
   * Sadržaj se može generisati statički (postojeći resurs isporučen klijentu i kao takav renderovan) ili dinamički (kreiran od strane aplikacije na serveru)
   * Popularne web tehnologije su:
     + Node.js + Express (JavaScript / TypeScript)
     + PHP (ranije: Personal Home Page, danas: Hypertext Preprocessor)
     + Microsoft .NET
     + Python + Django (framework)
8. **Troslojna arhitektura Web aplikacije**
   * Troslojna (generalnije n-slojna) arhitektura ili višeslojna arhitektura je klijent-server arghitektura koja fizički odvaja funkcije prezentovanja, aplikativnog procesiranja i skladištenja podataka na nivoe.
   * Tri sloja (nivoa) su:
     + Web server – IP servisi i frontend – odgovoran za:
       - Komunikaciju sa klijentom:
         * HTTP/S request – zahtev koji šalje klijent
         * HTTP/S response – odgovor koji server šalje klijentu (JSON, XML, HTML, ...)
       - Komunikaciju sa nivoom iznad sebe:
         * Invoke Component – buđenje višeg nivoa i zahtevanje dodatnih proračunavanja/resursa
         * Response – primanje podataka/odgovora od višeg nivoa
     + Application server – Model i logika domena – odgovoran za:
       - Komunikaciju sa nižim nivoom:
         * Invoke Component – primanje dodatnih zahteva
         * Response – prosleđivanje traženog odgovora
       - Komunikaciju sa višim nivoom:
         * Invoke Query – Zahtev za podacima iz baze podataka
         * Return of results – Prihvatanje rezultata baze
     + Server baze podataka – Sloj perzistencije – odgovoran za:
       - Komunikaciju sa nižim nivoom:
         * Prihvatanje Query-ja
         * Vraćanje rezultata
       - Execute Query – Izvršavanje zahteva
   * Ovakav pristup je usko povezan sa MVC šablonom:
     + Model-View-Controller (ASP .NET MVC)
       - Model – POCO klasa
       - Controller – API Controller (polazna tačka)
       - View – generiše HTML (Razor/Web strane)
9. **ORM alati**
   * ORM (**O**bject-**R**elational **M**apping) je programska tehnika za pretvaranje podataka u objektno-orijentisanim programskim jezicima između međusobno nekompatiblnih struktura. Na ovaj način se kreira “virtuelna objektna baza podataka” koja se može koristiti iz programskog jezika. U objekto-orijentisanom programiranju upravljanje podacima bazira se na objektima koji su skoro uvek neskalarne veličine.
   * **Primer**: Stavka u adresaru reprezentuje jednu osobu, koja ima nula ili više brojeva telefona, i nula ili više adresa. Ovo se u objekto-orijentisanom modelu može implementirati definisanjem objekta **Person** koji ima atribute koji drže podatke o njemu: ime osobe, lista brojeva telefona i lista adresa. Lista brojeva telefona bi u sebi sadržala objekte **PhoneNumber**, i tako dalje. Stavka u adresaru se tretira kao jedan objekat od strane programskog jezika, i može mu se pristupiti jednom promenljivom (npr, pokazivačem koji pokazuje na objekat). Nad ovim objektom se mogu primenjivati različite metode, npr. vraćanje podrazumevanog broja telefona, itd.
   * Međutim, mnogi popularni alati za rad sa bazama podataka, kao što je **SQL DBMS**, mogu da čuvaju i manipulišu samo skalarnim veličinama kao što su celi brojevi i stringovi, organizovani u tabele. Programer mora ili da konvertuje vrednosti objekata u grupe sa jednostavnijim vrednostima za čuvanje u bazi (i da ih ponovo konvertuje pri čitanju nazad), ili da koristi samo jednostavne skalarne vrednosti u programu. Objektno-orijentisano mapiranje implementira prvi prilaz. Srce problema leži u prevođenju logičke reprezentacije objekata u atomičnu formu koju je moguće skladištiti u bazi podataka, održavajući pritom svojstva objekata i njihove veze tako da se mogu ponovo učitati kao objekiti kada se za tim javi potreba. Ako se implementira ova funkcionalnost za pribavljanje i skladištenje, za objekte se kaže da su perzistentni.
   * Preslikava objekte u relacionu bazu
     + Klasa => Tabla
     + Objekat => Red u tabeli
   * Većina Hypertext Preprocessor (PHP) podržava Object-Relation Mapping
   * .NET => Entity Framework
10. **Raspoređivanje opterećenja servera (Load Balancing)**
    * Raspoređivanje opterećenja (balansiranje tereta / saobraćaja) je proces distribucije (podele) skupa naredbi na grupu kompjuterskih resursa, umesto opterećivanja jednog resursa velikom količinom naredbi.
    * Podela zaduženja na veći broj izvršitelja naredbi/zahteva (servera / aplikativnog softvera / baze podataka / kompjutera / procesora...)
    * Koristi se u svrhe opsluživanja velikog broja zahteva kako bi vreme izvršenja bilo što ujednačenije za svakog i naravno što kraće.
    * U slučaju najezde naredbi/klijentskih zahteva se aktivira ovaj mehanizam koji kreira dodatne instance kako bi opslužio novodostigle zahteve.
11. **Računarstvo u oblaku**
    * Računarstvo u oblaku je dostupnost računarskih resursa preko mreže na zahtev klijenta.
    * Podela na osnovu resursa koji se iznajmljuju:
      + Software as service – korisnik dobija pristup aplikativnom softveru i bazama podataka, nema nikakvog kontakta sa održavanjem infrastrukture i platforme na kojoj se ovaj software izvršava.
      + Platform as service – Korisnicima (uglavnom razvijači aplikativnog software-a) je na korišćenje dostupna platforma u vidu okruženja za razvoj: operativni sistem, programski jezik, debugger, baza podataka, web server... U tom okruženju se gradi software, vrši testiranje, pokretanje, execution bez potrebe održavanja hardware/arhitekture.
      + Infrastracture as service – Application Programming Interfaces visokog nivoa koji se koriste za apstrahovanje raznovrsnih detalja mrežne infrastrukture poput: fizičkih resursa za računicu, lokaciju, particiju podataka, skaliranje, bezbednost, backup itd..
        - Dostupna je mreža, skladištenje podataka, serveri i virtuelizacija okruženja
12. **Serverless arhitektura**
    * Serverless arhitektura (Serveless computing BackEnd as a Service) je model računarstva u oblaku gde korisnik iznajmljuje virtualne mašine neophodne za ispunjavanje zahteva frontend-a. Ne naplaćuje se eksplicitno infrastruktura, već broj poslatih zahteva, količina podataka itd. Korisniku je na raspolaganju kompletan backend, a on je zadužen da napravi samo svoj frontend
    * FaaS (Function as a Service) je deo serverless computing-a u širem smislu i podrazumeva na raspolaganju funkcije (procedure) koje omogućavaju ispunjavanje određenih zahteva.
    * Ovde je vrlo bitna optimalnost, jer se naplaćuje po zahtevu, a ne po satnici.
13. **Web servisi**
    * Web servis je aplikacija smeštena na nekom serveru, koja je pored osnovne namene dizajnirana da podrži interakciju između dve mašine preko mreže i omogući razmenu informacija između njih. Smatra se da je svaki servis i web servis ako je:
      + Dostupan preko interneta ili (interne mreže)
      + Koristi standardizovan sistem poruka
      + Prepoznatiljv je od strane mehanizma za pretragu
      + Nije vezan za operativni sistem ili programski jezik
    * Glavni protokol za transport podataka između web servisa i klijenta je HTTP iako je moguće koristiti i druge protokole. Za razmenu podatka se koristi XML ili JSON. Web servis obično pruža objektno-orijentisani interfejs serveru baze podataka, koji se koristi od strane drugog web servera ili mobilne aplikacije, koji nudi korisnički interfejs krajnjem korisniku. Poznati primeri su Google Maps API i EBay Shopping API.
    * Web servisi mogu biti :
      + SOAP (Simple Object Access Protocol) – Protokol – tipičan primer **service1.getPerson()**
      + REST (Representational State Transfer) – Arhitektura – tipičan primer [**http://example.com/persons**](http://example.com/persons).
    * Prednosti u odnosu na web aplikaciju:
      + **Štedljiviji su po pitanju opterećenja mreže i resursa servera** jer pri komunikaciji šalju samo odgovor dok web aplikacije pored odgovora šalju i HTML sa formom i opisom kako odgovor treba da bude prikazan. Web servisi su idealni za “male” uređaje koji nisu PC tj. mobilne Pocket PC… jer je na takvom uređaju instalirana samo front-end aplikacija a teži deo se odradjuje na serveru.
      + **Lakši su za razvoj, testiranje i održavanje**. Kod Web aplikacije pored neophodne funkcionalnosti je potrebno istestirati i dizajn na svim browser-ima i platformama dok kod web servisa autor brine samo o funkciji dok o prikazu brine klijentska aplikacija.
    * Aplikacioni softver sa različitim protokolima
14. **RPC**
    * Remote Procedure Call predstavlja pozivanje (izvršavanje) procedure (podrutine) sa klijenta, koja se nalazi u drugom adresnom prostoru (na drugom računaru / mreži itd) na nekom serveru npr. I korisnik ne mora da kodira detalje njenog poziva.
    * Stub – klijentska verzija procedure koja se poziva
    * RPC (Remote Procedure Call) je opis situacije u kojoj kompjuterski program primorava izvršenje procedure u drugom adresnom prosotru (često na drugom računaru u mreži), pri čemu je kod napisan kao da se radi o klasičnom (lokalnom) pozivu procedure, odnosno pri čemu programer nije u dužnosti da eksplicitno kodira detalje takve udaljene konekcije. Drugim rečima, programer piše isti kod bilo da se procedura poziva lokalno u odnosu na program koji se izvršava, ili ne. Ovo je primer klijent-server interakcije (pozivalac je klijent, izvršilac je server), i obično se implementira sistemom za razmenu poruka “zahtev-odgovor”. Pandam RPC-u u objektnoorijentisanom programiranju je RMI (Remote Method Invocation).
    * RPC-ovi su vrsta inter-procesne komunikacije (IPC), u kojoj različiti procesi imaju različite adresne prostore: ako su na istoj mašini, imaju različite virtuelne adresne prostore i pored toga što je fizički adresni prosotor isti; ako su na različitm hostovima, fizički adresni prostor je različit.
    * RPC započinje klijent, koji šalje poruku poznatom udaljenom serveru u kojoj nalaže izvršenje specificirane procedure koristeći priložene parametre. Udaljeni server šalje odgovor klijentu, i aplikacija nastavlja svoj proces. Dok server obrađuje poziv, klijent je blokiran (čeka dok mu server ne odgovori), osim ako je zahtev bio poslat asinhrono, kao što je XHTTP poziv.
    * Važna razlika između udaljenih poziva procedura i lokalnih poziva jeste to što RPC može da ne uspe zbog nepredviđenih problema sa mrežom. Sem toga, u opštem slučaju, pozivalac mora da se nosi sa takvim padovima ne znajući da li je udaljena procedura uopšte pozvana.
    * Idempotentne procedure (procedure koje nemaju dodatne efekte ako se pozovu više od jedanput, npr. Metoda DELETE) ne predstavljaju problem, ali ostale procedure ostaju problem.
    * Sekvenca događaja tokom RPC-a je sledeća.
      + Klijent pozove stub (verzija funkcije na klijentskoj strani). Arumenti/Parametri se smeste na stack, kao i u regularnom pozivu procedure.
      + Parametri bivaju upakovani u poruku (marshalling) i poslati serveru od strane operativnog sistema.
      + Operativni sistem servera prima poruku. Otpakuje paket, preuzme parametre (unmarshalling)
      + Konačno server izvrši funkciju i istim postupkom vraća rezultate serveru.
15. **SOAP servisi**
    * Simple Object Access Protocol predstavlja protokol razmene poruka baziranih na XML-u uz pomoć HTTP ili ponekad SMTP-a
    * U osnovi je RPC (remote procedure call, prepoznaje se na slici)
    * Pronalaze se preko UDD (Universal Description, Discovery and Integration)
    * Interfejs se opisuje preko WSDL-a
      + Web Services Desctiption Language:
        - Opis podataka – XML Schema
        - Interface/portType – operacije i koje poruke se koriste
        - Binding – format poruka i detalje protokola web servisa
        - Service – skup sistemskih funkcija koje su izložene web protokolima
    * Klijentske aplikacije komuniciraju sa web servisom preko SOAP poruka
    * Komunikacija ne zavisi od jezika i platforme
    * Komplikovani i ugalvnom jedino čitljivi od strane mašina
16. **RESTful servisi**
    * REST (Representanional State Transfer) servisi predstavljaju de-facto standard za software-sku arhitekturu interaktivnih aplikacija koje koriste veći broj web servisa.
    * REST je arhitekturni stil (nije protokol) koji omogućava komunikaciju između web servisa.
    * Pronalaženje resursa uz pomoć HTTP i URI (Uniform Resource Identifier)
    * Da bi REST bio RESTful on treba da omogući aplikativni pristup svojim resursima uz pomoć protokola koji ne pamti stanje (HTTP) podržavajući sve operacije koje omogućavaju kreiranje, čitanje, izmenu i brisanje podataka (CRUD), čime se ostvaruje perzistentnost podataka (pamćenje).
    * HTTP definiše operaciju, dok URI definiše nad kojim resursom se ta operacija obavlja
    * JSON za prenos podataka
    * U REST arhitekturi, REST server samo pruža pristup resursima, a REST klijent pristupa i predstavlja resurse. Svaki resurs je identifikovan preko URI-ja ili preko globalnog ID-ja. Na primer, URI /UserService/users prosleđene preko GET može da vrati listu svih korisnika, dok navođenje konkretnog parametra (/UserService/users/1) može da vrati konkretnog korisnika (sa ID-jem 1). REST koristi razne reprezentacije za predstavljanje resursa, kao što su tekst, JSON ili XML. Danas je JSON najpopularniji format. Za web aplikacije koje su u skladu sa gore navedenim osobinama se kaže da su RESTful.
17. **GraphQL**
    * GraphQL je open-source query jezik za čitanje i manipulaciju API-ova koji klijentu omogućava de definiše schemu/format (odgovora) koji očeuje od servera
    * Za razliku od REST-a, koji nabavlja kompletne entitete, GraphQL nabavlja samo podatke koje je korisnik precizirao, a to mogu biti i pojedine stavke većeg broja entiteta, za samo jedan zahtev (entrypoint)
    * Podaci su opisani schemom kao custom objekti koja je ogledalo json formata koji vraća server
    * BackEnd developer mora da napiše resolver za svaki entitet, kako bi omogućio ovakve upite

type Query{ // Main entry point

    videos: [Video]

    creator(id: String!): Creator

}

type Mutation{ // Manipulacije

    createVideo(url: String): Video

    deleteVideo(url: String): String

}

type Creator{ // Zahtev za određenim stavkama entiteta

    id: ID!

    name: String

    subscribers: Integer

    videos: [Video]

}

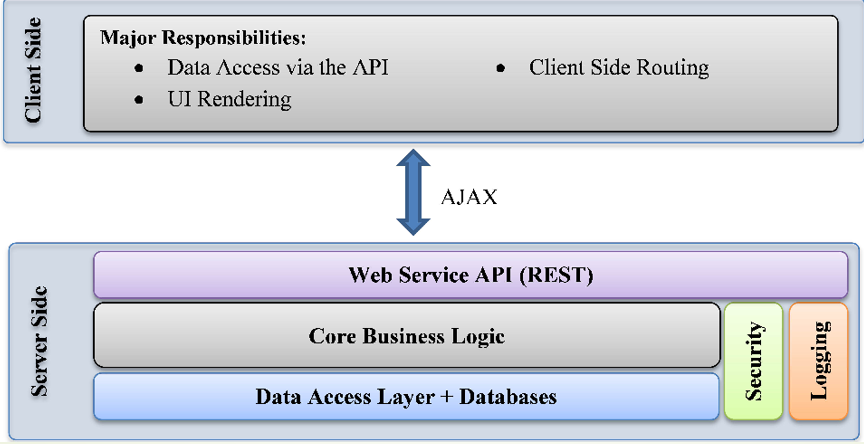
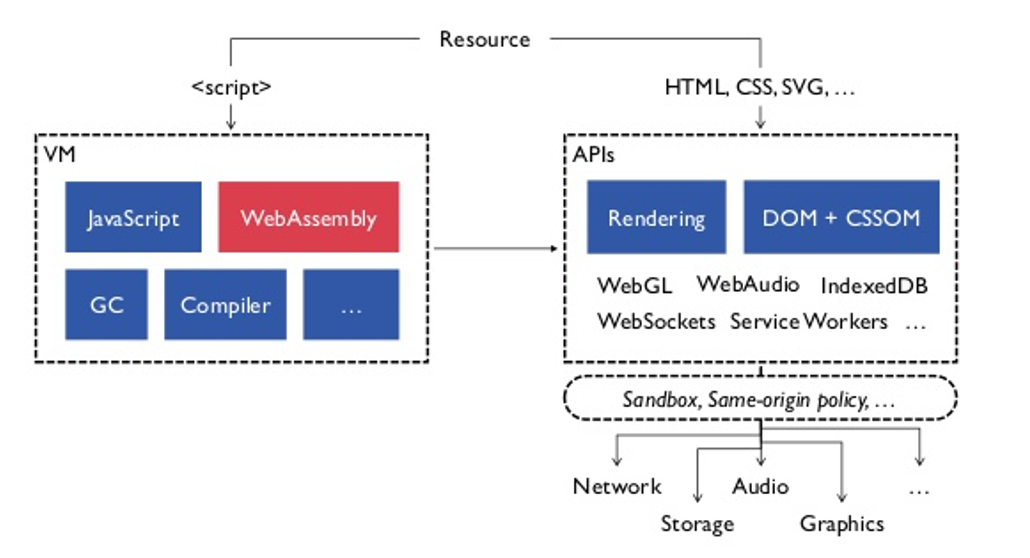
type Video{

    url: String!

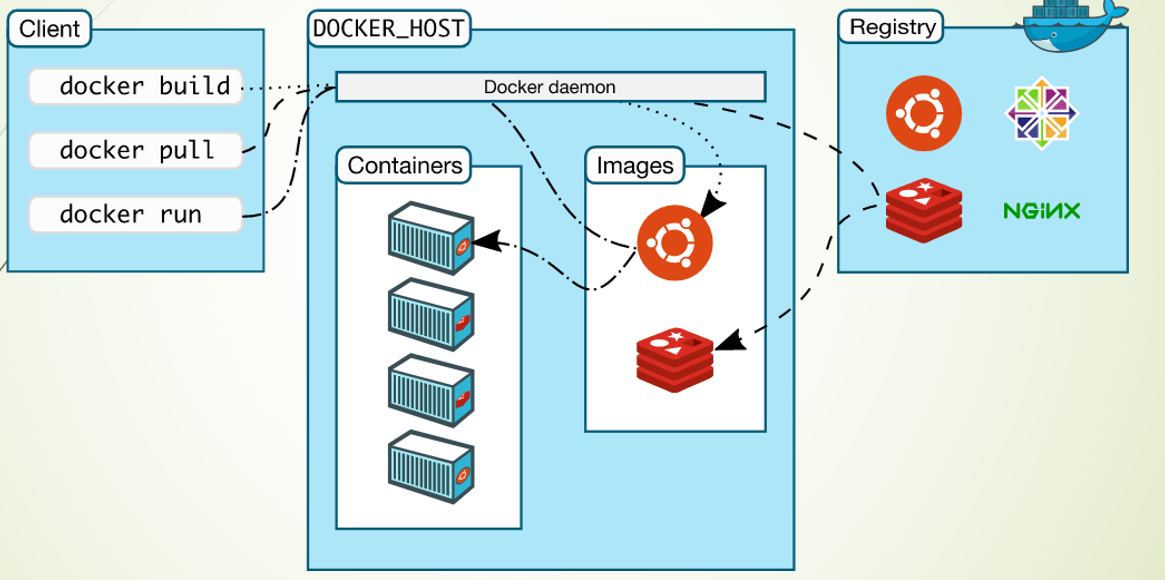
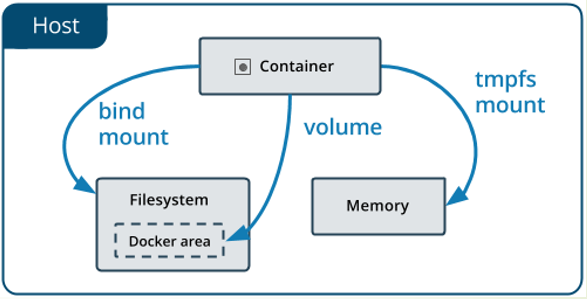
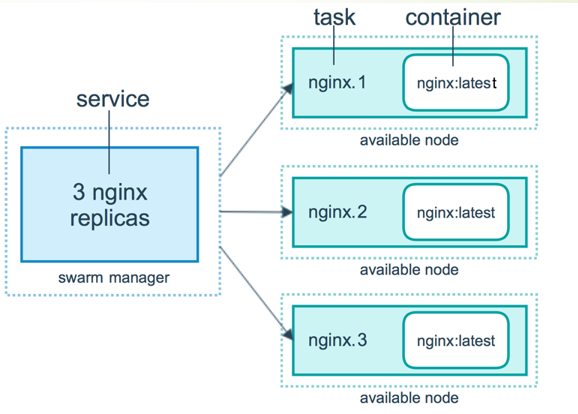
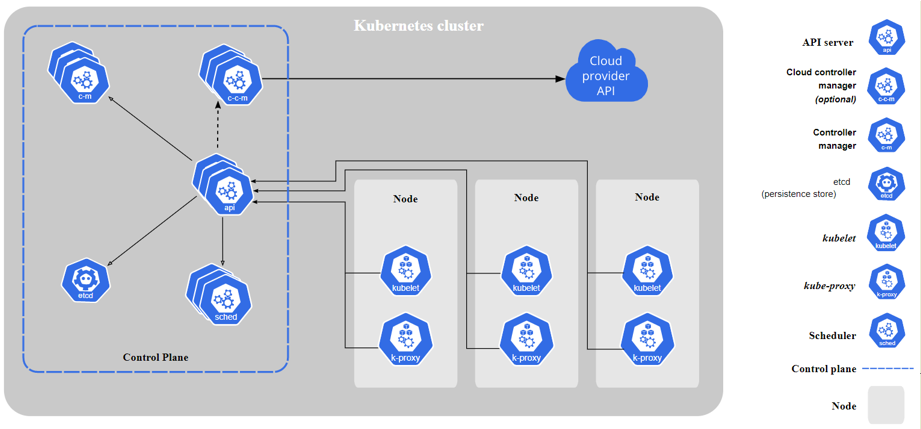
    creator: Creator

}

* + Kreiran od strane Facebook – a
  + Jezik za upite ka API-ju + servirski runtime za izvršavanje upita nad definisanim tipovima
  + Klijent definiše format traženih podataka
  + Nije vezan za bazu
  + Jedan endpoint i HTTP zahtev
  + Resolver za svaki entitet
  + Postupak
    - Klijent pošalje zahtev formiran kao schema-u sa stvarima koje su mu potrebne
    - Server deli koristi resoulver-e za svaki enitite i formira JSON odgovor koji je identičan zahtevu, samo sada sadrži potrebne podatke
    - Klijent dobija svoje podatke, bez redundantnih stavki

1. **gRPC**
   * gRPC je Google-ova verzija Remote Procedure Calls-a (komunikacija između web servisa)
   * Koristi HTTP/2
   * Koristi Protocol Buffers (ProtoBuffs):
     + Binarni format poruka
     + Nije čitljiv programerima
     + Dosta efikasniji od JSON-a
   * Pruža:
     + Autentifikaciju,
     + Bidirekcionalno stream-ovanje
     + Flow Control
     + Blocking/Nonblocking bindings
     + Cancellation
     + Timeouts etc
2. **Mikorservisi**
   * Mikroservisna ahitektura je tip servisu-orijentisane arhitekture (SOA), strukturni stil, koji aplikaciju uređuje kao kolekciju manjih servisa (procesa) sa sopstvenim funkcionalnostima i bazama koji svi zajedno služe za izvršavanje zajedničog zahteva
   * Svaki servis je manja aplikacija sa bazom
   * API Gateway je posrednik između servisa i klijenta
   * Prednosti:
     + Brži razvoj (manja kompleksnost pojedinačnih servisa)
     + Paralelizacija razvoja (jedan tim jedan servis)
     + Sopstveni skup tehnologija (jedan mikroservis == technology stack)
     + Nezavisan deploy
     + Nezavisno skaliranje
   * Mane:
     + Komplikacije pri međusobnoj komunikaciji
     + Složenije tranzakcije / konzistentnost zbog većeg broja baza podataka
     + Testiranje
     + Složeni deploy celine
   * Koristi Websockets
3. **SPA**
   * Single Page Application (SPA) je web aplikacija kod koje se celokupna interakcija sa korisnikom odvija na jednoj stranici. Takva stranica se učita jednom i dinamički ažurira tokom korišćenja. Jedna od najpopularnijih tehnologija za razvoj SPA su JavaScript framework AngularJS i React.
   * Pri interakciji sa tradicionalnim web aplikacijama server odgovara zahtevu klijenta slanjem nove stranice koja se učitava na strani klijenta. Kod SPA aplikacija serveru se upućuje AJAX zahtev i on odgovara potrebnim podacima, najčešće u JSON formatu. Posledice ovakvog pristupa su:
     + Razdvajanje prezentacionog dela od logike i podataka, što iziskuje bolje projektovanje aplikacije, postiže se stabilnost i aplikacija se lakše proširuje.
     + Projektovanje klijenta je u potpunosti nezavisno od projektovanja servera.
     + Klijent samo treba da zna koji zahtev da pošalje, prikazivanje tih podataka radi samostalno.
     + Komplikovaniji je klijentski deo.
     + Server ne daje nikakve informacije o prikazu podataka koje šalje.
     + Klijent dobija podatke od servera i ceo user interface realizuje sam. Iz tog razloga je potrebno uložiti više rada u client-side i frontend i u tu svrhu se koristi JavaScript.
   * Single Page Applications – JavaScript + RESTful service
   * Angular, Rect, Vue...
   * Klijentsa strana renderuje jednu stranu, dok korisnik interaguje sa elementima
   * Šalju se AJAX (Asynchronous JavaScript and XML) zahtevi i dobijaju odgovori od servera koji samo menjaju sadržaj već postojeće strane
4. **PWA**
   * Progressive Web Application
   * Nezavisna od browsera, uređaja i konekcije (Progressive Enhancement, Responsive, Cache)
   * Nalik na native apliakcije (Izgled, interakcija, navigacija, itd)
   * Ažurna (Service workeri nabavljaju najnoviju verziju), Sigurna (HTTPS) i Vidljiva (Pretraživači je mogu pronaći)
   * Obaveštava o novom sadržaju (notifikacije i ostali mehanizmi koji skreću pažnju korisniku, čak iako ne koristi aplikaciju)
   * Instalabilna i Linkabilna (Bez app store-a, ima web app manifest i lako je podeliti s drugima)
   * Offline podrška
   * Service Workeri (JavaScript skripta koja se izvršava u pozadini u posebnoj niti)
     + Keširanje podataka
     + SInhronizacija u pozadini
     + Push notifikacije
   * Web App Manifest (JSON fajl koji opisuje aplikaciju i koje resurse ona koristi)
   * PWA vs Native (Percentage Revenue koji uzimaju App Store-ovi od InAppPurchases)
5. **Web Assembly**

* WebAssembly (abbreviated Wasm) is a binary instruction format for a stack-based virtual machine. Wasm is designed as a portable compilation target for programming languages, enabling deployment on the web for client and server applications.
* Efficient and fast - The Wasm [stack machine](https://webassembly.org/docs/semantics/) is designed to be encoded in a size- and load-time-efficient [binary format](https://webassembly.org/docs/binary-encoding/). WebAssembly aims to execute at native speed by taking advantage of [common hardware capabilities](https://webassembly.org/docs/portability/#assumptions-for-efficient-execution) available on a wide range of platforms.
* Safe - WebAssembly describes a memory-safe, sandboxed [execution environment](https://webassembly.org/docs/semantics/#linear-memory) that may even be implemented inside existing JavaScript virtual machines. When [embedded in the web](https://webassembly.org/docs/web/), WebAssembly will enforce the same-origin and permissions security policies of the browser.
* Open and debuggable - WebAssembly is designed to be pretty-printed in a [textual format](https://webassembly.org/docs/text-format/) for debugging, testing, experimenting, optimizing, learning, teaching, and writing programs by hand. The textual format will be used when [viewing the source](https://webassembly.org/docs/faq/#will-webassembly-support-view-source-on-the-web) of Wasm modules on the web.
* Part of the open web platform - WebAssembly is designed to maintain the versionless, feature-tested, and backwards-compatible [nature of the web](https://webassembly.org/docs/web/). WebAssembly modules will be able to call into and out of the JavaScript context and access browser functionality through the same Web APIs accessible from JavaScript. WebAssembly also supports [non-web](https://webassembly.org/docs/non-web/) embeddings.
* Web Assembly je jezik niskog nivoa, nalik na assembler, sa kompaktnim binarnim formatom instrukcija (asemblerski jezik), namenjen za stack virtualnu mašinu.
* Koristi se kao meta za prevođenje, odnosno kod pisan u nekom drugom jeziku može da se prevede u .wasm binarni format I kao takav da se izvršava u bilo kom pretraživaču.
* Moguće je I kucati dorektno AssemblyScript I onda ga prevesti u binarni format .WAT => .WASM
* Omogućava izvršenje aplikacija visokih performansi u okviru pretraživača ili na JS virtualnoj mašini.
* Služi kao most između JavaScript-a i drugih tehnologija višeg nivoa. JS okruženja mogu koristiti Wasm uvođenjem njegovih modula u JS kod.
* Moguće je pisati program na C#, C++ itd i prevesti ga u .WASM
* Podržan je od strane svih pretraživača

1. **Blazor**
   * Blazor je besplatan, open-source web framework za kreiranje web aplikacija uz pomoć C#-a i HTML-a, može biti kombinovano sa JavaScript-om.
   * Pet verzija (samo prve dve dostupne):
     + Blazor Server:
       - Hostovane su na ASP .NET serveru u ASP .NET Razor formatu
       - Klijenti skidaju malu (u smislu podataka) stranu čiji se UI update-uje koristeći SignalR konekciju (besplatna, open-source softverska biblioteka za Microsoft ASP .NET koja omogućava serverskom kodu da šalje asinhrone notifikacije klijentskoj strani. Uključuje i serverske i klijenstke strane JavaScript-a).
     + Blazor WebAssembly:
       - Ovo su Single Page Apps skinute na klijenta pre pokretanja. Veličina skinute strane je zatno veće od one na serveru i potpuno procesiranje se odvija na klijentskom hardware-u.
       - Zahteva rapid response time
       - Aplikacija se nalazi u WebAssemly formatu (može biti kombinovano sa JavaScriptom)
     + Blazor PWA
     + Blazor Hybrid
     + Blazor Native – platform native framework
2. **Virtualizacija**
   * Predstavlja čin kreiranja virtualne verzije nečega, u našem slučaju kompjuterske arhitekture, koja omogućava pokretanje većeg broja operativnih sistema koji funkcionišu na istom uređaju (arhitekturi).
   * Hypervisor (monitor virtualne mašine) je kompjuterski software, firmware ili hardware koji kreira virtualnu mašinu i na kome se ona pokreće. Uređaj na kome se izvršava hypervisor se naziva host, a svaka od virtualnih mašina se naziva guest machine.
3. **Kontejnerizacija, Docker**
   * Kontejnerizacija ili virtualizacija operativnog sistema predstavlja odliku operativnog sistema da jezgro (kernel OS-a) dozvoljava postojanje većeg broja izolovanih instanci korisničkog prostora.
   * Ove instance, koje se nazivaju kontejneri, particije, virtualna okruženja, zatvori... pokreće Container engine.
   * Svi kontejneri dele zajedničko jezgro operativnog sistema, kao i neke bibilioteke i izvršne fajlove
   * U odnosu na VM zauzimaju znatno manje memorije i pokreću se znatno brže, jer su ovo procesi, a ne kompletan operativni sistem
   * Jedan kontejner se vezuje za jednu komponentu i sadrži sve što je neophodno da bi se komponenta izvršavala
     + Fajlovi,
     + Promenljive okruženja
     + Zavisnosti
     + Biblioteke...
4. **Orkestracija kontejnera (Kubernetes, Docker Swarm**
   * Orkestracija kontejnera predstavlja proces upravljanja nad životnim ciklusom kontejnera, pogotovo u velikim dinamičkim okruženjima. Kontrola i automatizovanje poslova:
     + Obezbeđivanje i postavljanje (pokretanje) kontejnera
     + Redundantnost i raspoloživost kontejnera
     + Umnožavaje (skaliranje) ili uklanjanje kontejnera kako bi se pravilno rasporedilo opterećenje na infrastrukturi
     + Premeštanje kontejnera sa jednog host-a na drugi ukoliko na prvom nema dovoljno resursa za pokretanje
     + Alokacija resursa između kontejnera
     + Konfiguracija aplikacije koja je pokrenuta na kontejnerima...
   * Docker
     + Kontejner je instanca jedne docker slike
     + Može se pokrenuti, stopirati ili izbrisati
     + Prilikom pokretanja mu se dodeljuje i jedan Read-Write file system, kako bi mogao da manipuliše fajlovima
     + Stateless, promene se gube pri brisanju instance, ali mogu biti i stateful
     + Mogu se povezati i na persistence storage (sve promene koje nisu tu sačuvane se gube)
     + Može se kreirati novi docker image na osnovu trenutnog stanja kontejnera (load balancing)
     + Docker image:
       - Read-Only šablon koji se build-uje i kasnije instancira kontejnere na osnovu slike
       - Zabeležene su sve neophodne stavke koje kontejner treba da zadovolji
       - Jedan image se bazira na drugi
       - Sastoji se od nivoa (svak docker instrukcija je jedan nivo)
       - Pri promeni jedne instrukcije, ne build-uje se ceo image, već samo nivoi od nivao na kom je nastala promena i svi nivoi nakon njega.
     + Perzistencija u Docker-u
       - Podaci se gube kad kontejner prestane da postoji
       - Layer je striktno vezan za mašinu
       - Volume je deo file system-a koji pripada docker-u, procesi van docker-a ne bi trebalo da menjaju taj deo
       - Bind mount može biti bilo koji deo file system-a. Veoma važni fajlovi (konfiguracija) i direktorijumi, procesi van docker-a menjaju taj deo
       - Tmpfs je u memoriji sistema hosta, ne prepisuje se u file system
       - Jedan volume se može koristiti od strane više kontejnera istovremeno (komuniakcija između kontejnera)
       - Npipe za komunikaciju između hosta i kontejnera, služi da se 3rd party alat pokrene u kontejner
   * Docker Swarm
     + Predstavlja grupu fizičkih ili virtuelnih mašina pokrenutih od strane docker aplikacije koje su konfigurisane da rade zajedno kao cluster. (Grupisani nodovi)
     + Node može biti worker, manager ili oba
     + Task je kontejner koji se izvršava i njime upravlja manager Node
     + Pri promeni konfiguracionog fajla (servisa) docker swarm će se postarati da zaustavi taskove koje koriste staru konfiguraciju i pokrene nove
     + Orkestracioni sloj, load balancing, rolling updates,
     + Karakteristike:
       - Jednostavno instaliranje, jedinstvena tačka pada
       - Brzo skaliranje (2000+ nodes)
       - Ručno skaliranje
       - Optimizovano za manje cluster-e
       - Ručne provere ispravnosti
   * Kubernetes
     + Open-source sistem za automatski deployment, skaliranje i upravljanje kontejnerizovanih aplikacija
     + Load balancing
     + Orkestracija smeštajnog sistema – mogućnost postavljanja po izboru (lokalna baza, cloud, provajderi, itd)
     + Samoizlečenje – restartovanje, zamena, gašenje kontejnera
     + Komponente:
       - Čaura (Pod) – skup kontejnera koji pokreću aplikaciju. Kontejneri imaju deljeni kontekst (mreža, filesystem, itd). Replikacija je na nivou čaura.
       - Čvor (node) – radna mašina koja pokreće kontejnerizovanu aplikaciju i sačinjena je od čaura
       - Kluster – skup čvororva
       - Kontrolna ravan – upravlja čvorovim i podovima
     + Specifikacije:
       - Kompleksna instalacija, jak otporan cluister
       - Sporo skaliranje zbog kompeksnosti sitema (5000+ nodes)
       - Automatizovano skaliranje
       - Optimizovano za jedan veliki cluster
       - Regularne provere ispravnosti