Veb-programiranje

SKRIPTA ZA USMENI ISPIT

Lazar Ljubenović

lazar.ljubenovic.1995@gmail.com

15. juni 2016

Sadržaj

Glava 1

Lista pitanja

Najčešća pitanja iz veb-programiranja:

1. Web aplikacije
2. Arhitektura Web aplikacija
3. HTML, opste, struktura strane
4. HTML5 vs HTML4
5. HTML5, semanticki elementi
6. CSS, opste
7. CSS selektori
8. Javascript (JS), opste
9. JS objekti
10. DOM, definicija i najcesce funkcije API-ja
11. progresivno poboljsanje
12. tipovi dizajna strane
13. Responsive dizajn
14. XML tehnologije
15. XML namespace
16. DTD
17. DTD, konkretan primer

18. XML Schema
19. XPath
20. XML vs HTML
21. XSL
22. JSON vs XML
23. JSON
24. AJAX
25. CGI
26. PHP, opste
27. PHP superglobalne promenljive
28. prenos podataka sa klijenta na server
29. PHP, klase i objekti
30. PHP, nasledjivanje
31. ORM alati
32. Web servisi
33. RESTful servisi
34. SOAP servisi
35. RPC
36. JAVA opste
37. Java virtuelna masina
38. Java biblioteke
39. Java Beans
40. JSP
41. Java apleti
42. Model2

Glava 2

Odgovori

2.1 Frontend tehnologije

> **Pitanje 1** Veb-aplikacije.

Odgovor. Veb-aplikacija je aplikacija kojoj se pristupa preko veb-čitača korišćenjem interneta ili intranet mreže.

Veb-aplikacija je softverska aplikacija bazirana na klijent-server modelu kod koje se klijent (ili korisnički interfejs) izvršava u veb-čitaču. Veb-aplikacije su popularne zbog sveprisutnosti veb-čitača. Jednostavnost korišćenja veb-čitača kao klijenta koji ažurira i održava veb-aplikaciju bez distribuiranja i potencijalnog instaliranja softvera na velikom broju računara, kao i kompatiblnost među platformama, predstavljaju ključni razlog njihove popularnosti. Neki česti primeri veb-aplikacija su veb-mejl, onlajn aukcije, viki, servisi za razmenu trenutnih poruka, itd.

Opštu razliku između **veb-sajta** i bilo kog tipa **veb-aplikacije** nije moguće precizno definisati. Veb-sajtovi za koje će se češće govoriti kao o "veb-aplikacijama" su oni koji po funkcionalnost ili izgledu podsećaju na desktop softverske aplikacije ili mobilne aplikacije.

Jednostranične aplikacije (SPA, *Single Page Applications*) više podsećaju na aplikacije jer odbacuju tipičnu veb-paradigmu navigacije kroz više različih stranica preko različitih URL-ova. Postoji više frejmvorkā, kao što je Angular, koji se mogu koristiti da bi se brže razlive ovakve aplikacije.

Veb-aplikacije se najčešće organizuju u takozvane rangove (*tier*), i obično ih ima tri: prezentacija, aplikacija i skladište, redom. Veb-čitač je prvi rang

(**prezentacija**). Endžin koji koristi neku veb-tehnologiju (npr. ASP, CGI, Dart, Node.js, PHP, Pajton) je srednji rang (**aplikaciona logika**). Baza podataka je treći rang (**skladište**). Veb-čitač šalje zahteve srednjem rangu, koji ih servira tako što šalje zahteve i naredbe bazi podataka i generiše korisnički interfejs.

Pitanje 2 Arhitektura veb-aplikacija.

Odgovor. MVC (*Model View Controller*) je softverski arhitekturni obrazac koji se često koristi za implementiranje korisničkih interfejsa; stoga predstavlja popularan izbor za kreiranje arhitekture veb-aplikacija. Najopštije rečeno, MVC deli aplikacioni logiku na tri odvojena dela, promovišući modularnost, jednostavnu međusobnu saradnju i mogućnost ponovnog korišćenja. Sem toga, aplikacije koje koriste ovaj model postaju fleksiblnije.

Možemo posmatrati aplikaciju za kupovinu. Ona može biti vrlo jednostavna: treba da sadrži listu imena, kvantita i cene za svaki proizvod koji treba kupiti ove nedelje.

Model definiše šta aplikacija treba da sadrži. Ako se stanje ovih podatka promeni, onda model uglavnom treba da obavesti pogled (kako bi se prikazala nastala promena), a ponekad kontroler (za slučaj da je potreba drugačija logika da bi se kontrolisao pogled koji se ažurira). U primeru aplikacije za kupovinu, model bi odredio koje podatke sadrži lista (proizvod, cena, itd) i koje proizvode već sadrži.

Pogled definiše kako se prikazuju podaci aplikacije i prima podatke koje treba da prikaže od modela. U primeru aplikacije za kupovinu, pogled bi definisao kako se lista prezentuje koristinu.

Kontroler sadrži logiku koja ažurira model i/ili pogled kao odgovor na promene koje korisnik unosi u aplikaciju. Na primer, aplikacija za kupovina bi imala polja za unos novih proizvoda i dugmad koja nam dopuštaju da dodamo nove proizvode ili obrišemo postojeće. Ove akcije zahtevaju da se model ažurira. Zato se unos šalje do kontrolera, koji zatim manipuliše modelom na odgovarajući način, i zatim šalje ažurirane podatke nazad pogledu. Pored toga, aplikacija može pružati i mogućnost jednostavnog ažuriranja pogleda u smislu drugačije prezentacije podataka. Na primer, listu proizvoda za kupovinu treba sortirati po nekom drugom kriterijumu, kontroler može direktno da obavi ovu operaciju bez ažuriranja modela.

Kod veb-aplikacija, model podataka je najčešće sadržan u nekoj bazi podataka (što može biti tradicionalna baza podataka kao što je MySQL, ili rešenje

na klijentskoj strani kao što je IndexedDB). Kod koji kontroliše aplikaciju je uglavnom napisan korišćenjem HTML-a i JavaScripta, dok je korisnički interfejs (pogled) definisan HTML-om, CSS-om i možda JavaScriptom.

U ranim fazama razvoja veba, MVC arhitektura je najčešće bila implementirana na strani servera, pri čemu je klijent zahtevao ažuriranje podataka korišćenjem formi ili putem linkova, a primao je ažurirane podatke kao stranice koje su se prikazivale u brauzeru. Međutim, u moderno doba, sve više logike se prebacuje na stranu klijenata zahvaljujući napretku skladištā podataka na klijenata. Pored toga, objekti tipa `XMLHttpRequest` dopuštaju ažuriranje samo dela stranice.

Popularni veb frejmvorci kao što su AngularJS, Ember.js i Backbone implementiraju MVC arhitekturu, mada svaki na malo drugačiji način.

> **Pitanje 3** Opšte o HTML-u. Struktura stranice.

Odgovor. HTML (**HyperText Markup Language**) je standardni jezik za obeležavanje koji se koristi za kreiranje veb-stranica. Uz CSS i JavaScript, predstavlja temeljnu tehnologiju za izgradnju veb-stranica, kao i korisničkih interfejsa za mobilne i veb-aplikacije. Veb-čitači čitaju HTML fajlove i renderuju ih, što ga čini **jezikom za obeležavanje** (*markup*), a ne programskim jezikom. HTML semantički opisuje strukturu veb-stranice, a pre pojave CSS-a sadržao je i opis izgleda stranice.

Gradivni elementi HTML stranice su **HTML elementi**. Pomoću njih je moguće kreiranje strukturnih dokumenata obeležanjem naslovā, paragrafā, listi, linkova, citatā, itd. HTML elementi su omeđeni tagovima, koji se pišu u uglasnim zagradama. Na primer, tekst *primer* je omeđen parom tagova (tag i anti-tag) na sledeći način: `<p>primer</p>`. Postoje i tagovi koji nemaju svoj anti-tag, već zatvaraju sami sebe, kao što je `<img />` (slika) ili `<input />` (polje za unos).

Primer minamalne HTML stranice dat je u nastavku.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Naslov</title>
  </head>
  <body>
    <p>Hello world!</p>
  </body>
</html>
```

Tekst između `<html>` i `</html>` opisuje veb-stranicu, a tekst između `<body>` i `</body>` opisuje vidljivi deo stranice. Zaglavlje HTML dokumenta nalazi se između para tagova `<head>` i `</head>` i u njemu se između ostalog može naći naslov stranice (u ovom primeru `<title>Naslov</title>`).

> **Pitanje 4** HTML5 vs HTML4.

Odgovor.

- Novi semantički elementi: `<header>`, `<footer>`, `<article>`, `<section>`, `<menu>`, `<figure>`, `<figcaption>`, `<audio>`, `<video>`, `<canvas>`. Tu su i `<mathml>` i `<svg>` elementi.
- Izbačeni nesemantički elementi: `<frame>`, `<center>`, `<big>`.
- Izbačeni mnogi nesemantički atributi, npr. `bgcolor` i `marginleft` na `body`.
- Favorizuje se korišćenje se semantičkih elemenata spram nesemantičkih kada za tim postoji potreba, npr. `<strong>` umesto `<b>`; `<emph>` umesto `<i>`; ubačeni i drugi kao `<code>`, `<mark>`, itd.
- Nova pravila za parsiranje. HTML4 namerno dozvoljava tzv. "supu tagova" (npr. nekorektno `<p>Ovo je <strong>tekst.</p></strong>`), a brauzeri se staraju o tome da ga ipak prikažu ispravno. Međutim, za ovo nigde nisu bila jasno definisana pravila, pa je svaki tim koji razvija novi veb-brauzer morao da pokreće testove u starijim veb-čitačima (pogotovo u IE), i da primeni tehniku obnutog inženjerstva kako bi imao istu detekciju grešaka. U suprotnom bi se mnoge stranice ne bi prikazivane ispravno u čitaču. Procenjuje se da je oko 90% veb-stranica bar u neku ruku loše formatirano.

  HTML5 pokušava da ovo traženje greška pretoči u kod kako bi se svi čitači mogli da preuzmu ovaj standard i tako više ne ulažu vreme i novac u konzistentnost prikaza stranica.
- Parser više ne ignoriše zadnje blanko znake u `id` atributu – više nisu uopšte dozvoljeni; ali sada postoje manje restrikcije: jedini uslovi su da mora da postoji bar jedan karakter i da ne smeju da postoje blanko znaci. Dozvoljeni su i znaci kao `#`, ali se ne preporučuje njihovo korišćenje jer se onda oni moraju eskejpovati u CSS selektorima (`#\#`). Takođe, u HTML4 `id` nije smeo da počne cifrom, već isključivo velikim ili malim slovom.

- Pruža mogućnost za jednostavnijim `DOCTYPE`-om.

```
<!-- HTML4 -->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
          "http://www.w3.org/TR/html4/strict.dtd">

<!-- HTML5 -->
<!DOCTYPE html>
```

Doktajp iz ranijih verzija HTML-a je bio duži jer je tada HTML bio baziran na SGML-u[1] i zato je morao da ima referencu na DTD.

Naime, HTML u verzijama HTML+, HTML 2.0, HTML 3.2 i HTML 4.01 je zvanično bio definisan kao SGML aplikacija. Međutim, nikad nije bio *implementiran* tako; ne postoji objavljena aplikacija koja HTML interpretira kao SGML aplikaciju. Razlog za ovo navodi i sama HTML 4.01 specifikacija (odeljak B.3.3).

> Od SGML sistema [...] se očekuje da prepoznaju određeni broj odlika koje nisu široko podržane od strane HTML korisničkih agenata. Preporučujemo da autori izbegavaju njihovo korišćenje.

HTML+, HTML 2.0, HTML 3.2 su pisani korišćenjem SGML sintaksnih pravila. HTML 4 je imao dve sintakse: SGML (HTML 4.01) i XML (XHTML 1.0).

S druge strane, HTML5 je apstraktni jezik. Čini ga *opis vokabulara* koji se može pisati koristeći dve različite sintakse: **html** i **XML**. HTML 5 definiše novu serijalizaciju koju prosto naziva *html*, koja namerno dosta podseća na SGML iz ranijih verzija.

Drugim rečima, HTML5, kao apstraktni jezik, ima dva parsera (odnosno dve serijalizacije): *html* i *XML*, koje redom imaju internet medija tipove `text/html` i `application/xhtml+xml`.

```
<!-- text/html -->
<!DOCTYPE html>
<html lang="en">
    <!-- ... -->
</html>
```

[1] SGML (*Stanrd General Markup Language* je ISO specifikacija napravljena radi definisanja deklarativnih jezika za obeležavanje. Na vebu, HTML4, XHTML i XML su popularni jezici bazirani na SGML-u.

```
<!-- application/xhtml+xml -->
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
    <!-- ... -->
</html>
```

Malo ljudi u praksi koristi XHTML. Ako se sadržaj sačuva kao internet medija tip `application/xhtml+xml`, sem toga što neće raditi u starijim verzijama Internet Explorera, i najmanja greška će kao posledicu imati neprikazivanje stranice čak i u brauzerima koji podržavaju XHTML. Zato, ono što ponekad izgleda kao XHTML (`<br/>` je *XHTML*, `<br>` je *html*) se zapravo servira i interpretira kao HTML.

Pitanje 5 HTML5, semantički elementi.

Odgovor. Semantički elementi jasno opisuju svoje značenje, kako veb-čitaču, tako i programeru. Pored bolje čitljivosti koda, važni su zbog pristupačnosti (npr. za audio-čitače veb-stranica za slepe) i za SEO (optimizacija za veb-pretraživače).

Dva najčešće korišćena **nesemantička** taga su `<div>` i `<span>`.

HTML Document Division Element (`<div>`) je **generički kontejner** za sadržaj, koji nema posebno značenje i ne predstavlja ništa. Može da se koristi za grupisanje elemenata radi stilizovanja (koristeći atribute `id` i `class`), ili za elemente koji dele zajedničke atribute, kao što je `lang`. Treba se koristiti samo kada nijedan drugi semantički element ne odgovara svrsi.

HTML Document Span Element (`<span>`) je **generički kontejer** koji se prostire u okviru linije teksta (inlajn element), i ne predstavlja ništa. Koristi se kao i `<div>`, s tim što je `<div>` element na nivou bloka, dok je `<span>` na nivou linije.

Prema MDN-u, semantički elementi se dele na osnovne elemente, metapodatke, elemente za izdvajanje sadržaja, elemente koji sadrže tekst, inlajn elemente, elemente za slike i druge multimedijalne sadržaje, elemente za ugrađeni (*embedd*) sadržaj, elemente za skripte, elemente za demarkaciju, elemente za tabelarne podatke, elemente za forme, interaktivne elemente i veb-komponente.

Osnovni elementi Jedini osnovni element koji svaka stranica mora da ima da bi se mogla smatrati HTML stranicom jeste `<html>`. Ovaj element predstavlja koren HTML dokumenta. Svi elementi moraju biti njegovi potomci.

Metapodaci o dokumentu Element `<base>` specificira osnovni URL koji treba da koriste svi relativni URL-ovi iz dokumenta.

Element `<head>` daje opšte informacije o dokumentu, uključujući naslov i linkove do skirpti i stajlšitova.

Element `<link>` specificira vezu između trenutnog dokumenta i eksternog izvora; koristi se najčešće za povezivanje sa stajlšitovima.

Element `<style>` koristi se za specifikaciju stilova koje treba primeniti na trenutni dokument. Podrazumevano se radi o CSS jeziku.

Element `<title>` govori o naslovu stranice koja će biti prikazana u veb-čitaču.

Element `<meta>` koristi se za podatke koji se ne mogu svrstati u gorenavedene metapodatke.

Izdvajanje sadržaja Ovi elementi organizuju dokument u logičke celine. Pomoću njih se brzo kreira opšta kontura stranice.

Element `<article>` predstavlja celinu koja može postojati sama za sebe i može se ponovo koristiti u drugom kontekstu. To može biti poruka na forumu, članak u časopisu, post na blogu, objekat, ili bilo koji drugi nezavistan sadržaj. Svaki `<article>` treba da bude identifikovan, obično dodavnjem elemenata za naslov (npr. `<h1>`) kao deteta artikla.

Element `<section>` predstavlja generičku sekciju dokumenta, npr. tematsko grupisanje sadržaja, najčešće sa naslovom. Kao i `<article>`, svaki `<section>` treba da bude identifikovan.

Elementi od `<h1>` do `<h6>` predstavljaju šest nivoa naslovā u dokumentu, pri čemu manji redni broj značava veću važnost. Ovaj element kratko opisuje temu sekcije koju predstavlja. Ove informacije agenti mogu da koriste da bi automatski napravili sadržaj. Svaki `<section>` ima svoju hijerarhiju naslovā, tako da sledeći isečak HTML stranice predstavlja hijerarhiju u kome je *Opet naslov* podnaslov naslova *Naslov*, iako su oba sa istim rednim brojem.

```
<section>
    <h1>Naslov</h1>
    <section>
        <h1>Opet naslov</h1>
    </section>
</section>
```

Element `<header>` predstavlja grupu sadržaja uvodnog ili navigacionog karaktera. Može sadržati nasove, ali i druge elemente koji predstavljaju npr. logo, formu za pretragu, itd.

Element `<footer>` predstavlja futer za najbližu roditeljsku sekciju (ili za ceo dokument, ako je najbliža sekcija predstavljena `<body>` tagom). Futer najčešće sadrži podatke o autoru sekcije, informacije o zaštiti autorskih prava i slično.

Element `<nav>` koristi se za deo stranice koji sadži linkove to drugih stranica ili do raznih delova iste stranice. Dakle, to je sekcija sa navigacionim linkovima.

Element `<address>` dostavlja kontakt-inforamcije u vezi sa najbližim `<article>` roditeljem. Ako takav ne postoji, podaci se tiču celog dokumenta.

Tekst sadržaj Ovi elementi se koriste radi organizacije blokova ili sekcija. Važni su zbog pristupačnosti i SEO jer identifikuju značaj sadržaja koji se u njima sadrži.

Element `<ul>` predstavlja neuređenu listu stavki, odnosno kolekciju stavki koje nisu ni na koji način numerisani. Slično, `<ol>` predstavlja uređenu listu. Unutar ovih elemanta se mogu naći isključivo `<li>` elementi, i njihov sadržaj predstavlja stavke koje treba nabrojati. Brauzeri stavke najčešće renderuju jednu ispod druge, dodajući simbole ispred stavki: bulet u slučaju neuređene liste i redni broj u slučaju uređene liste. Kastomizacija ovih osobina brši se putem CSS-a, a ne atributima liste.

Element `<main>` predstavlja glavni sadržaj elementa `<body>` u dokumentu ili aplikaciji. Sadržaj ovog elementa treba da bude od suštinske važnosti, i da ne uključuje delove koji se ponavljaju kroz stranice. Treba da bude jedinstven na stranici.

Element `<p>` predstavlja paragraf u tekstu.

Element `<pre>` predstavlja ranije formatirani tekst. Obično se renderuje korišćenjem neproporcionanog (monospejs) fonta i znaci beline se ne sažimaju kao kod ostalih elemenata.

Element `<dl>` predstavlja listu opisa i omeđuje parove termina i njihovih opisa. Obično se koristi za implementaciju rečnika ili za prikaz metapodataka. Elementi `<dt>` i `<dd>` mogu da se nađu samo u listi opisa i predstavljaju, redom, termin i opis. Može ih biti više izastopno (više termina ima isti opis ili jedan termin ima više opisa). Najpre se navodi termin (ili termini)

pa opis (ili opisi). Brauzeri termine obično renderuju boldirano, a opise u novom redu, uvučeno.

Element `<figure>` pretstavlja sadržaj koji može postojati sam za sebe, najčešće uz natpis koji je predstavljen elementom `<figcaption>`. Mada je usko vezan za glavni tok, njegova pozicija ne zavisi od glavnog toka. Ovo je obično slika, ilustracija, dijagram, isečak koda ili šema koja je referencirana u tekstu, ali se može pomeriti na sledeću stranicu ili u apendiks, a da ne naruši glavni tok.

Element `<hr>` po HTML5 specifikaciji predstavlja tematsku pauzu između elemanta na nivou paragrafa (na primer, promena scene u priči ili promena teme u članku). U ranijim verzijama HTML-a, ovaj element je predstavljao horizonalnu liniju bez semantičkog značenja. Mada ga veb-čitači i dalje najčešće predstavljaju na isti način, on sada ima semantičko značenje, a ne prezentaciono.

Itd, itd, itd.

> **Pitanje 6** CSS, opšte.

Odgovor. CSS je **jezik za formiranje** koji omogućava da se definiše izgled elemenata prvenstveno veb-stranice, mada se može koristiti i za bilo koje druge XML dokumente, uključujući npr. SVG. Uz HTML i JavaScript, predstavlja temeljnu tehnologiju za izgradnju veb-stranica.

Poboljašnje mogućnosti prezentacije podataka na vebu bile su tema mnogih onlajn foruma, a devet rešenja su između 1993. i 1996. bila dominantna u *www-talk* mejling listama. Od ovih devet rešenja, dva su imala poseban uticaj na razvoj CSS-a: CHSS (Cascading HTML Style Sheets) i SSP (Stream-based Style Sheet Proposal). “H” je izbačeno iz imena jer se CSS standard mogao koristiti i za druge jezike za označavanje osim HTML-a.

Primer CHSS:

```
font.family = times
h1.font.family = helvetica
```

Primer SSP:

```
HTML.justify: full
*H1.justify: center
*OL.LI.label: A
```

CSS je dizajniran kako bi pre svega odvojio sadržaj dokumenta od prezentacije istog, pri čemu se pod prezentacijom najčešće podrazumeva raspored

elemenata (lejaut), boje i fontovi. Ovakva struktura omogućuje **veću fleksibilnost** pri specifikaciji karakterisika prezentacije, jer omogućuje da više stranica dele isti eksterni `.css` fajl, čime se smanjuje ponavljanje istih podataka u strukturi samih dokumenata. Izmena grafičkog izgleda neograničenog broja stranica svodi se na promenu nekoliko linija CSS koda u jednom fajlu. Pored toga, moguće je isti sadržaj prezentovati na različite načine, npr. za prikaz na veb-stranici ili za štampanje, koristeći dva različita CSS-a.

Pitanje 7 CSS selektori.

Odgovor. **Selektori** služe da se definišu na koje se delove dokumenta odnosi deklaracioni blok koji sledi. Selektori se mogu odnositi na:

- sve elemente istog zadatog tipa (npr. naslov drugog nivoa `<h2>`) – **selektor tipa** (npr. `h2 {color: red}`);
- elemente koji imaju zadati atribut ili zadatu vrednost atributa, ili su određenom odnosu za zadatom vrednošću atributa – **selektori atributa** (npr. `[checked]`, `[lang="en"]`, `[data-tag~="post"]`);
- elemente koji imaju specificiran identifikator ili klasu:
  - identifikator koji je jedinstven u celom dokumentu – **selektor identifikatora** (`#logo {color: red}`)[2],
  - identifikator koji može da se odnosi na više elemenata u dokumentu – **selektor klase** (`.card {color: red}`)[3];
- sve elemente (`*`) – **univerzalni selektor**.

U tesnoj vezi sa selektorima su tzv. **kombinatori**.

- Kombinator ` ` (spejs) bira čvorove koji su **naslednici** (ne obavezno direktni naslednici, tj. deca) prethodno specificiranog elementa. Na primer, `div span` se odnosi na sve elemente `<span>` koji se nalaze u elementu `<div>`.
- Kombinator `+` selektira čvorove koji se nalaze odmah nakon specificiranog elementa (**susedna braća**). Na primer, `img + span` se odnosi na sve elemente `<span>` koji se nalaze odmah nakon elementa `<img>` (pa se tako mogu npr. selektirati opisi slika).

[2]Oktotrop sa nazivom identifikatora (`#logo`) je skraćen oblik za `[id="logo"]`.

[3]Tačka sa nazivom klase (`.card`) je skraćen oblik za `[class="card"]`.

- Kombinator `~` selektira čvorove koji se nalaze nakon (ne obavezno odmah nakon) specificiranog elementa (**opšta braća**). Na primer, `p ~ span` se odnosi na sve elemente `<span>` koji se nalaze nakon elementa `<p>`, a imaju azjedničkog roditelja.
- Kombinator `>` selektira **decu**, odnosno **direktne potomke** prethodno specificiranog elementa. Na primer, `ul > li` se odnosi na sve elemente `<li>` koji su direktni potomci `<ul>` (pa se tako npt. mogu definisati stilovi za različite "dubine" ugnježdenih listi).

Od selektora, koriste se još i pseudo-klase i pseudo-elementi. **Pseudo-klase** dozovljavaju formatiranje u skladu sa informacijama koje nisu dostupne u stablu dokumenta; na primer, deklaracioni blok selektora `:hover` će se primeniti na element samo kad korisnik postavi kursor na element. Neke pseudo-klase mogu imati i argumente, kao što su `:nth-child()`, pa tako `:nth-child(4)` selektira samo četvrti element u nizu. Moguće je čak i selektirati samo svaki treći element koristeći posebnu promenljivu n, `:nth-child(3n)`.

Pseudo-elementi, za razliku od pseudo-klasa, ne opisuju posebna stanja, već omogućuju stilizovanje samo nekih delova dokumenta. Na primer, pseudo-element `::first-line` gađa samo prvu liniju elementa specificiranog selektorom. Zvanično ih je podržano šest: `::after`, `::before`, `::first-letter`, `::first-line`, `::selection` i `::backdrop`.

Pitanje 8 Javascript (JS), opšte.

Odgovor. Javascript (*javaskript* ili *džavaskript*) je skriptni dinamički netipizirani interpretacioni programski jezik visokog nivoa. Baziran je na prototipovima sa funkcijama prve klase, što znači da podržava više paradigmi, tj. objektno-orijentisano, imperativno i funkcionalno programiranje. Ima API za rad sa tekstom, nizovima, datumima i regularnim ekspresijama, ali ne podržava nikakvu vrstu ulaza/izlaza i oslanja se na okruženje domaćina u kom je ugrađen (najčešće veb-pregledač, mada to mogu biti i Adobe Acrobat, Adobe Photoshop, SVG slike, serverska okruženja kao što je Node.js, NoSQL baze podataka kao Apache CouchDB, itd). Uz HTML i CSS, predstavlja temeljnu tehnologiju za izgradnju veb-stranica. Svi moderni veb-pretraživači ga podržavaju bez potrebe za instaliranjem dodatnih programa.

Standardizovan je ECMAScript specifikacijom. Od 2012, svi moderni veb-pretraživači u potpunosti podržavaju **ECMAScript 5.1**. Sedamnaestog juna 2015 objavljena je šesta verzija ECMAScripta, zvanično nazvana ECMAScript 2015, mada se češće sreće naziv ES6. Od tada se standard obna-

valja u godišnjim ciklusima.

Kreirao ga je Brendan Ajk 1995. godine dok je radio kao inženjer u Netskejpu. Prvi put je objavljen uz Netscape 2 početkom 1996. Prvenstveno je trebalo da se zove LiveScript, ali je preimenovan u JavaScript kao (loša) marketinška strategija sa namerom da se osloni na tadašnju popularnost jezika Java, i pored toga što jezici imaju malo toga zajedničkog.

Nekoliko meseci kasnije, Majkrosoft je izdao JScript uz IE3, koji je po mnogim karakteristikama bio kompatibilan sa JavaScriptom. Ubrzo potom, Netskejp je JavaScript prosledio evropskoj organizaciji za standrde Ecma International, što je kao rezultat iste godine dalo prvu ediciju ECMAScript standarda. Standard je doživeo značajno unapređenje 1999. godine objavom treće edicije, i od tada je ostao manje-više stabilan. Mada je četvrta edicija odbačena zbog političkih neslaganja o kompleksnosti jezika, mnogi delovi njeni delovi su poslužili kao osnova za petu ediciju (objavljena 2009), kao i za šestu (jun 2015).

JavaScript podržava sledeće tipove podataka: `Number`, `String`, `Boolean`, `Object` (gde spadaju `Function`, `Array`, `Date` i `RegExp`), kao i `null` i `undefined`.

JavaScript pravi razliku između `null` i `undefined`. `null` je vrednost koja ukazuje da je programer namerno postavio vrednost na "ništa" (i to se može uraditi jedino korićšćenjem ključne reči `null`). S druge strane, `undefined` je konstanta koja se dodeljuje neinicijalizovanim (već samo deklarisanim) promenljivama.

Svaka vrednost se može kastovati u logičku vrednost, pri čemu `false`, `0`, prazni stringovi `""`, `NaN`, `null` i `undefined` postaju `false`, dok sve ostalo postaje `true`.

Pitanje 9 JS objekti.

Odgovor. Objekti u JavaScriptu su parovi ključ-vrednost. Slični su heš-tablicama iz jezika C i C++ ili asocijativnim nizovima iz PHP-a.

Postoje dva osnovna načina da se napravi prazan objekat.

```
1 var obj1 = new Object();
2 var obj2 = {};
```

Ova dva načina su međusobno semantički ekvivalentna, mada se preporučuje korišćenje drugog načina. Ova sintaksa je takođe osnova za JSON format i uvek se preferira njeno korišćenje kada je to moguće. Pored toga, druga verzija omogućuje instanciranje celog objekta izjedna.

```
var obj = {
  name: "Carrot",
  "for": "Max",
  details: {
    color: "orange",
    size: 12
  }
}
```

Vrednostima se sada može pristupiti na dva načina: `obj.details.color` ili `obj["details"]["color"]`. Oba načina su semantički ekvivalentna. Kod druge metode, parametar je string pa se to može iskoristiti kao prednost jer omogućuje dinamički izbor ključa kome se pristupa. Pored toga, može se koristiti da se pristupi vrednostima svojstava koje kao ključ imaju ključnu reč. Na primer, `obj.for` je sintaksna greška, dok je `obj["for"]` validna sintaksa.

Pitanje 10 DOM, definicija i najčešće funkcije API-ja.

Odgovor. **Document Object Model**, skraćeno DOM, je programski interfejs za HTML, XML i SVG dokumente. Pruža struktuiranu reprezentaciju stabla dokumenta. DOM definiše metode za pristup stablu, i na taj način omogućuje promenu strukture dokumenta, stilova i sadržaja. DOM pruža i reprezentaciju dokumenta kao strukturne grupe čvorova i objekata, od kojih svaki ima različita svojstva i metode. Za čvorove mogu biti vezani i rukovodioci događajima (event handlers); kad se događaj okine, rukovodioci se izvršavaju. Najkraće rečeno, DOM povezuje veb-stranice sa skriptama ili programskim jezicima.

Mada se DOM najčešće koristi uz JavaScript, on *nije* njegov sastavni deo. Može mu se pristupiti i drugim jezicima. Na primer, može se koristiti i uz Pajton.

```
import xml.dom.minidom as m
doc = m.parse("C:\\doc.html");
doc.nodeName # DOM property of document object;
p_list = doc.getElementsByTagName("para");
```

Neke od najčešće korišćenih funkcija su date u nastavku:

- `document.getElementById(id)` – Vraća referencu na element zadat svojim identifikatorom (string). Identifikator se može postaviti direktno iz HTML-a kroz atribut `id` ili kroz skriptu. Ako ne postoji element sa specificiranim identifikatorom, vraća se `null`.
- `Element.getElementsByTagName(name)` – Vraća tzv. "živ" niz elemenata

tipa `HTMLCollection` sa zadatim imenom taga. Pretraga se obavlja kroz sve potomke elementa `Element`, isključujući njega samog. Lista koja je vraćena je "živa", što znači da će se automatski ažurirati zajedno sa DOM stablom.

- `document.createElement(name)` – Kreira specificirani HTML element ili `HTMLUnknownElement` ako je prosleđeno ime elementa nepoznato.
- `Node.appendChild(node)` – Dodaje čvor na kraj liste dece specificiranog roditeljskog čvora. Ako je dato dete referenca na postojeći čvor u dokumentu, ova metoda ga pomera sa trenutne pozicije na novu. Ovo znači da čvor ne može istovremeno biti na dva mesta u dokumentu. Prema tome, ako čvor već ima dete, čvor se prvo uklanja, a onda se dodaje na novu poziciju. Metoda `Node.cloneNode()` se može iskoristiti za kreiranje kopije čvora pre nego što se on doda pod novog roditelja. Kopije napravljene na ovaj način neće automatski biti međusobno sinhronizovane. `appendChild()` ne podržava prenos čvorova među različtim dokumentima; umesto nje se koristi `document.importNode()`.
- `Element.innerHTML` – Služi za postavljanje ili dobavljanje HTML sintakse koja opisuje sadržaj elementa, odnosno njegove potomke. Spram toga, postoje i dve slične (mada sa nekim bitnim razliakma) metode `Node.innerText` (nije po standardu) i `Node.textContent` (po standardu), koje služe za dobavljanje teksta iz čvora i njegovih potomaka. Drugim rečima, pokušava da aproksimira ono što bi korisnik dobio kada bi selektirao sadržaj kursorom i kopirao sadržaj na klipbord.
- `Element.setAttribute(name, value)` i `Element.getAttribute(name)` služe za postavljanje i pribavljanje vrednosti postojećeg atributa specificranog elementa.
- `HTMLElement.stlye` – Vraća objekat tipa `CSSStyleDeclaration` koji reprezentuje samo inlajn atribut `style` specificiranog elementa i ignoriše sva druga CSS pravila.
- `window.onload` – Rukovodilac događajem za događaj `load` nad objektom `window`. Prosleđuje mu se referenca na funkciju. Događaj se okida u trenutku kada se završi proces učitavanja dokumenta – to je trenutak u kome su DOM, sve slike, skripte, linkovi i frejmovi u potpunosti učitani.

Pitanje 11 Progresivno poboljšanje.

Odgovor. Progresivno poboljšanje (PE, *Progressive Enhancement*) je **stra-**

tegija u veb-dizajnu koja u prvi plan stavlja pristupačnost, semantički HTML i korišćenje eksternih stajlšitova i skripti. PE koristi veb-tehnologije na **slojevit način**, odnosno omogućuje svakome da pristupi osnovnom sadržaju i funkcionalnosti veb-stranice, kroz bilo koji veb-pregledač ili sa bilo kakvom internet-konekcijom, ali i omogućava pristup naprednoj verziji stranice za korisnike sa naprednijim veb-pregledačima ili sa većim protokom podataka.

Tehnika se oslanja na ranije popularniju strategiju poznatu kao "**elegantna degradacija**" (*Graceful Degradation*), gde dizajneri kreiraju veb-stranice namenjene najnovijim veb-pregledačima, ali imaju u vidu i njihove ranije verzije. Elegantna degradacija treba da dopusti stranici da "degradira", odnosno da održi svoju prezentabilnost i pored toga što neke tehnologije koje dizajner želi da koristi nisu implementirane na korisnikovoj verziji veb-pregledača.

PE čine sledeći **osnovni principi**:

- osnovni sadržaj treba da bude dostupan na svim veb-pregledačima,
- osnovna funkcionalnost treba da bude dostupna na svim veb-pregledačima,
- sadržaj je označen bez suvišnih tagova i na semantički način,
- napredni izgled stranice se oslanja na eksterno povezani CSS,
- napredna funkcionalnost je ugrađena koristeći nenametljiv (*unobtrusive*) JavaScript,
- podrazumevana podešavanja na veb-čitačima krajnjih korisnika se poštuju.

Pitanje 12 Tipovi dizajna strane.

Odgovor. Za lejaut stranice se često koriste termini *fiksni*, *statički*, *tečni*, *adaptivni*, *responzivni*, itd. Svako ime opisuje kako se lejaut ponaša kada se stranica posmatra sa ekrana različitih veličina: širina stranice se može menjati u zavinsoti od rezolucije monitora, ali i od toga na kom uređaju se stranica posmatra: računar, lap-top, tablet, mobilni telefon ili pametni sat.

Kod **statičkih** stranica (još i: fiksne stranice, stranice fiksne širine), širina veb-stranice je definisana u **pikselima** i ne menja se zajedno sa širinom ekrana na kome se posmatra. Ovaj pristup se inetnzivno se koristio do početka 2010, pri čemu je najčešća širina stranice bila 960 px. Ukoliko je širina ekrana manja od fiksno definisane veličine, ponašanje može da bude različitio od implementacije platforme ili veb-pregledačā. Na računarima će ivice (ili samo jedna ivica) stranice najčešće biti odsečena, ali će na Eplovim mobilnim uređajima stranica biti smanjenja i korisnik će moći da zumira delove stranice, i tako navigira kroz sajt.

Tečne veb-stranice koriste **relativne** umesto fiksnih veličina. Pored procenata, to mogu biti em-ovi ili (odskora) i rem-ovi. Veb-stranice dizajnire ovom tehnikom će se adaptirati bilo kojoj veličini ekrana. Mana kod ovakog prilaza je to što će na veoma širokim monitorima sadržaj biti previše razvučen (na primer, čitav paragraf može biti samo jedna linija teksta), dok na malim ekranima sadržaj može biti previše zbijen i nepregledan.

Adaptivni lejaut stranice koristi CSS **medija-kverije** kako bi se odredila širina brauzera i da bi se u skladu s tim promenili stilovi. Kao i kod statičkih lejauta, koriste se fiksne veličine ali izbor veličine zavisi od širine brauzera. Na primer, kveri može biti definisan sledećim iskazom: "ako je širina stranice do 500 px, postavi glavni kontejner na 400 px; ako je širina stranice do 1000 px, postavi glavni kontejner na 960 px". Pored širine glavnog kontejnera, mogu se menjati i drugi parametri, pa i raspored elemenata. Na primer, sajt može smanjiti broj kolona na manjim ekranima.

Responzivni lejaut predstavlja kombinaciju statičkog i adaptivnog pristupa, odnosno koristi i dužine izražene **relativnim** jedinicama i **medija-kverije**. Najčešće će ovakvom načinu dizajniranju pristupa polazeći od najmanjih veličina ekrana, odnosno tzv. *mobile-first* pristupom, jer je najčešće lakše naći način da se proširi manji lejaut nego da se smanji postojeći. Danas predstavlja najčešći pristup kreiranju veb-stranica, zbog velikog procenta korisnika koji sajtovima pristupa preko mobilnih uređaja kao što su pametni telefoni i tableti (od početka 2014 preko 50%).

> **Pitanje 13** Responzivni dizajn.

Odgovor. Responzivni veb-dizajn (RWD) je pristup dizajniranju veb-stranica koji treba da omogući optimalni doživljaj veb-stranice – čitljivost i jednostavnu navigaciju bez potrebe za mnogo zumiranja i skrolovanja – bez obzira na uređaj sa kojeg krajnji korisnik pristupa sajtu (od desktop računara do mobilnih telefona). U cilju postizanja RWD-a, koriste se razne tehnike, među kojima su: fluidna koordinatna mreža, fleksibilne slike, CSS3 medija-kveriji, odnosno `@media` pravila, itd.

RWD je postao veoma važan koncept u dizajniranju veb-stranica otkako su statistike počele da pokazuju da preko polovine saobraćaja na internetu dolazi sa mobilnih uređaja. Pored toga, Gugl je 21. aprila 2015. pustio u rad Mobilegeddon, algoritam koji prilikom sortiranja rezultata pretrage uzima u obzir i to da li je sajt optimizovan za mobilne uređaje.

Pored promišljenog korišćenja CSS-a i medija-kverija u njemu, neophodno je u HTML uvesti odogvarajuće meta-podatke koji brauzeri mogu da pročitaju

o stranici. Na primer, metatag `viewport` se koristi da bi se kontrolisala širina i skaliranje oblasti prikaza brauzera. Navođenjem `width=device-width`, širina ekrana se usklađuje sa pikselima nezavisno od uređaja, a pomoću `initial-scale=1` uspostavlja se 1:1 odnos između CSS piksela i piksela nezavisno od uređaja. Pored `initial-scale`, mogu se podesiti i drugi parametri kao što su `minimum-scale`, `maximum-scale` i `user-scalable`.

2.2 XML

> **Pitanje 14** XML tehnologije.

Odgovor. XML (**Extensible Markup Language**) je jezik za označavanje koji definiše skup pravila za šifriranje dokumenata u formatu koji je **lako razumljiv** i ljudima i mašinama. Definisan je XML 1.0 specifikacijom koju je objavio W3C. Dizajniran je s namerom da naglasi jednostavnost, generalnost i upotrebljivost širom interneta. Radi se o formatu u kome se podaci čuvaju u tekstualnom obliku koji dobro podržava Junikod, pa samim tim i razne ljudske jezike. Mada je fokus na dokumentima, često se koristi i za reprezentaciju proizvoljnih struktura podataka (npr. postupkom serijalizacije podataka iz programa).

Na stotine formata je bazirano na XML sintaksi, među kojima su RSS, Atom i XHTML. XML formati su postali standard i za mnoge alate poput Microsoft Office ili LibreOffice. Koristi se i kao osnova za neke komunikacione protokle, kao što je XMPP. Aplikacije iz Majkrosoftovog .NET frejmvorka koriste XML sintaksu za konfiguracione fajlove.

Po definiciji, XML dokument je **string karakterā**. Skoro svaki legalni Junikod karakter se može pojaviti u XML dokumentu. **Preprocesor** analizira sadržaj dokumenta i predaje struktuiranu informaciju **aplikaciji**. Specifikacija nalaže pravila o tome šta preprocesor mora ili ne sme da uradi, ali aplikacija je van njenog opsega. Dokumentacija aplikaciju naziva **procesor**, a često se kolokvijalno kaže da je ona **XML parser**.

Karakteri koji čine XML dokument se dele na **znake za obeležavenje** i **sadržaj**. Razlika između njih se može veoma lako odrediti pomoću nekoliko jednostavnih sintaksnih pravila. U opštem slučaju, znak za obeležavanje ili počinje znakom `<` i završava se znakom `>` (tagovi), ili počinje znakom `&` i završava se znakom `;` (glifovi). Stringovi koji nisu znaci za obeležavanje predstavljaju sadržaj.

Postoje tri vrste **tagova**: otvoren tag (na primer, `<section>`), zatvoren tag (na primer, `</section>`), tag praznog elementa (na primer, `<line-break />`). Logička reprezentacija komponente koja se sastoji od otvorenog taga, sadržaja i zatvorenog taga naziva se **element**. Ukoliko sadržaj sadrži druge tagove koji čine jedan ili više elemenata, takvi elementi se nazivaju **elementi-deca**. U opštem slučaju, i deca mogu imati svoju decu, pa se svi oni jednim imenom nazivaju **potomci**. Deca se još nazivaju direktni potomci.

Konstrukcija u okviru taga koja se sastoji od para ime-vrednost naziva se **atribut**. U primeru `<step number="3">Connect A to B.</step>` izdvaja se atribut `name` sa vrednošću `3`.

XML dokumenti mogu počinjati i nekim meta-podacima, odnosno informacijama o sebi samima.

```
<?xml version="1.0" encoding="UTF-8"?>
```

XML je format koji se koristi za asinhronu razmenu podataka između brauzerā i serverā i naziva se AJAX.

Pitanje 15 XML namespace.

Odgovor. XML nejmspejsovi se koriste kako bi XML dokumentu obezbedili elemente sa jedinstvenim imenom. XML instanca može da sadrži element ili imena atributa iz više od jednog XML rečnika. Ako se svakom rečniku dodeli nejmspejs, dvosmislenost između elemenata sa istim imenom se može rešiti. Jednostavan primer je XML instanca koja ima podatke o kupcu i naručenom proizvodu. I element koji predstavlja kupca i element koji predstavlja proizvod bi mogli da imaju element-dete sa imenom `id`. Reference na element `id` bi stoga bile dvosmislene; ako se postave u druge nejmspejsove, dvosmislenost nestaje.

Ime nejmspejsa je URI. URI izabran za nejmspejs datog XML rečnika obično opisuje resurs koji je pod kontrolom autora ili organizacije koja definiše rečnik, na primer URL autorovog veb-servera. Međutim, nejmspejs specifikacija ne zahteva niti predlaže da se nejmspejs URI koristi za dobavljanje informacija; XML parser ga samo tretira kao string. Ipak, korišćenje URI-ja umesto običnog imena smanjuje verovatnoću da postoji više različitih nejmspejsova sa istim identifikatorima.

Mada se za ime nejmspejsa skoro isključivo koristi *namespace URI*, W3C Rekomendacija koristi termin *namespace name*. Specifikacija ne daje jasan odgovor na pravila za imenovanje nejmspejsova, niti nalaže da parseri moraju

da odbiju dokument čije ime nejmspejsa nije validan URI, i mnogi XML parseri dozvoljavaju korišćenje bilo kakvog stringa.

XML nejmspejs se **deklariše** korišćenjem rezervisanog XML atributa `xmlns` ili `xmlns:`*`prefiks`*, čija vrednost mora biti validno nejmspejs ime. Na primer, sledeća deklaracija mapira prefiks `xhtml:` na XHTML nejmspejs.

```
xmlns:xhtml="http://www.w3.org/1999/xhtml"
```

Svaki element ili atribut čije ime počinje prefiksom `xhtml:` se svrstava pod XHTML nejmspejs, ako on ili prethodnik ima gorenavedenu deklaraciju nejmspejsa. Ako se ne navede prefiks, smatra se da se radi o podrazumevanom nejmspejsu. U tom slučaju se za svaki element koji nema nejmspejs-prefiks smatra da pripada XHTML nejmspejsu, ako on ili prethodnik ima gorenavedenu deklaraciju nejmspejsa.

Pitanje 16 DTD.

Odgovor. DTD (**Document Type Definition**) je skup deklaracija za obeležavanje (markup declaration) koji definišu tip dokumenta (document type) za SGML porodicu jezika za označavanje (SGML, XML, HTML). DTD definiše legalne gradivne blokove XML dokumenta. Definiše strukturu dokumenta sa listom legalnih elemenata i atributa. Može biti deklarisan unutar samog XML dokumenta, ili koristeći eksternu referencu. XML koristi podskup SGML DTD-a.

DTD se sa XML ili SGML dokumentom povezuje koristeći deklaraciju tipa dokumenta, doktajp (`DOCTYPE`). Doktajp se pojavljuje u blizini sintaksičkog fragmenta *doctypedecl* blizu početka XML dokumenta. Deklaracijom se utvrđuje to da je dokument instanca tipa koji je definisan referenciranim DTD-om.

DTD-ovi opisuju strukturu klase dokumenata koristeći deklaracije elemenata i liste atributa. **Deklaracije elemenata** imenuju dozovljeni skup elemanata u dokumentu, i specificiraju kako deklarisani dokumenti i nizovi karaktera (podataka) mogu da se sadrže u svakom elementu. **Dekalracije liste atributa** imenuju dozvolejni skup atributa za svaki deklarisani element, uključujući i tip svake vrednosti atributa, ili čak i eksplicitni skup validnih vrednosti.

Pitanje 17 DTD, konkretan primer.

Odgovor. U sledećem primeru je definisan element `img`. Dozovljan sadržaj mu je podešen na `EMPTY`, što znači da element mora biti prazan.

```
<!ELEMENT img EMPTY>
```

Sledi primer jednostavnog eksternog XML DTD-a.

```
<!ELEMENT people_list (person)*>
<!ELEMENT person (name, birthdate?, gender?, socialsecuritynumber?)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT birthdate (#PCDATA)>
<!ELEMENT gender (#PCDATA)>
<!ELEMENT socialsecuritynumber (#PCDATA)>
```

- `people_list` je vaildno ime elementa, i instanca takvog elementa sme da sadrži bilo koji broj `person` elemenata. Zvezdica označava da može biti 0 ili više `person` elemenata unutar `people_list` elementa.
- `person` je validno ime elementa, i instanca takvog elementa sme da sadrži jedan element `ime`, zatim opciono jedan element `birthday`, opciono jedan element `gender` i opciono jedan element `socialsecuritynumber`. Znak pitanja označava da element nije obavezan. Referenca na element `name` nema `?},`što znači da `person` *mora* da sadrži element `name`.
- `name`, `birthdate`, `gender` i `socialsecuritynumber` su validna imena elemenata, i instance tih elemanta sadrže parsirane podatke-karatere (parsed character data), `(#CDATA)`.

Sledi primer XML dokumenta koji koristi goreopisan DTD. DTD je ovde referenciran kao eksterni podskup, pomoću `SYSTEM` specifikatora i URI-ja.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE people_list SYSTEM "example.dtd">
<people_list>
  <person>
    <name>Fred Bloggs</name>
    <birthdate>2008-11-27</birthdate>
    <gender>Male</gender>
  </person>
</people_list>
```

Sledi primer iz HTML4 specifikacije, za neuređenu listu i stavke u njoj.

```
<!ELEMENT UL - - (LI)+          -- unordered list -->
<!ATTLIST UL
  %attrs;                       -- %coreattrs, %i18n, %events --
  >

<!ELEMENT LI - O (%flow;)*      -- list item -->
<!ATTLIST LI
  %attrs;                       -- %coreattrs, %i18n, %events --
  >
```

Pitanje 18 XML Schema.

Odgovor. **XML Schema** je jezik za iskazivanje ograničenja XML dokumenata. Postoji nekoliko različitih šematskih jezika koji su u širokoj upotrebi, ali glavni su DTD (*Document Type Definitions*), Relax-NG, Schematron i W3C XSD (*XML Schema Definitions*).

Šema može da se koristi u nekoliko svrha.

- Obezbeđuje listu elemenata i atributa u rečniku.
- Vezuje tipove (bilo da su to generalni tipovi kao `integer`, `string`, itd, ili određeniji kao `hat_size`, `sock_color`, itd) sa vrednostima iz dokumenta.
- Ograničava gde se mogu naći elementi i atributi, kao i šta sme da se nađe unutar tih elemenata. Na primer, može se definisati da se naslov poglavlja može naći samo unutar poglavlja, ili da za naslovom poglavlja mora da sledi jedan ili više paragrafa teksta.
- Nudi informacije koje su čitljive ljudima, a procesabilne mašinama.
- Daje formalan opis jednog ili više dokumenata.

Informacije iz šematskih dokumenata se često koriste od strane editora koji razumeju XML kako bi korisnicima ponudili elemente koji imaju najveću verovatnoću korišćenja, na osnovu trenutnog konteksta. Proveravanje da li se dokument pridržava specifikaciji šeme je poznato kao **validacija** dokumenta spram šeme; ako se drugačije ne naglasi, podrazumeva se da se radi o DTD validaciji (a ne o npr. XSD validaciji).

XSD (*XML Schema Definition*), predlog W3C-a, specificira kako da se formalno opisuju elementi u XML dokumentu. Mogu ga koristiti programeri da bi verifikovali svaki deo sadržaja dokumenta. Mogu da provere da li se pridržavaju opisima elemanta u koje su smešteni. XSD može da se koristi da iskaže skup pravila kojih se XML dokument mora pridržavati kako bi se mogao smatrati "validnim" u skladu sa tom šemom. Ipak, za razliku od većine drugih šematskih jezika, XSD je takođe definisan sa namerom da određivanje validnosti dokumenta kao izazni produkt ima skup informacija koji se drže nekih tipova podataka. Takav post-validacioni **infoset** može biti koristan pri razvitku softvera za procesiranje XML dokumenata.

Pitanje 19 XPath.

Odgovor. XPath (**XML Path Language**) je jezik baziran na upitima koji

služi za selektiranje čvorova iz XML dokumenta. Pored toga, XPath se može koristiti za izračunavanje vrednosti (npr. stringova, brojeva ili logičkih vrednosti) iz sadržaja XML dokumenta. Definisan je od strane W3C-a. Baziran je na činjenici da se XML dokument može predstaviti kao **stablo** i pruža mogućnost navigacije kroz njega, selektirajući čvorove na osnovu različitih krterijuma.

Sintaksa je dizajnirana tako da mimikuje URI i putanje do fajlove na UNIX operativnim sistemima. Primer jednostavnog upita je `/A/B/C`, koji će selektirati elemente `C` koji su deca elemenata `B` koji su deca elemenata `A`. Mogući su i nešto komplikovaniji upiti, na primer `A//B/*[1]` selektira prvo dete (`[1]`), štagod bilo njegovo ime (`*`), svakog elementa `B`, koji je dete ili dublji potomak (`//`) elementa `A` koje je dete trenutnog kontekstnog čvora (ne počinje sa `/`).

Može se koristiti i puna, **proširena sintaksa**. Gorenavedeni se u ovoj proširenoj sintaksi mogu napisati na sledeće načine: `!/child::A/child::B/child::C!` i `!child::A/descendant-or-self::node()/child::B/child::*[position()=1]`.

Pitanje 20 XML vs HTML.

Odgovor. XML pruža pravila za kreiranje, struktuiranje i enkodiranje dokumenata. XML se često koristi da sadrži podatke i da obezbedi komunikaciju među aplikacijama. HTML je dirajniran za kreiranje strukturnih dokumenata koji imaju semantičko značenje. Ima fleksibilnija pravila (dopušta supu od tagova).

Ukratko, XML je razvijen s namerom da *opiše* podatke, a HTML da ih *predstavi*.

Pitanje 21 XSL.

Odgovor. XSL (*Extensible Stylesheet Language*) je porodica jezika koji se koriste za transformaciju i renderovanje XML dokumenata. Obuhvata tri jezika: XSLT (*XSL Transformation*) – XML jezik za transformisanje XML dokumenata, XSL-FO (*XSL Formatting Objects*) – XML jezik za specificiranje vizuelnog formatiranja XML dokumenta i XPath (*XML Path Language*) – jezik koji koristi XSLT, mada je dostupan i u drugim kontekstima, i služi za adresiranje delova XML dokumenata. U nastavku odgovora se pod XSL-om misli na kombinaciju XSLT-a i XSL-FO-a (v. drugi isečak koda).

XSL stajlšit je, kao CSS, fajl koji opisuje kako treba prikazati XML dokument datog tipa. Deli funkcionalnost i kompatibilan je sa CSS2, mada koristi drugačiju sintaksu. Pored toga pruža sledeće mogućnosti.

- Transformacioni jezik za XML dokumente: **XSLT**. Prvobitno namenjen za izvođenje kompleksnih operacija za manipulaciju stilovima, npr. generisanje tabela iz sadržaja i indeksā, sada se koristi kao XML jezik za obradu opšte namene. XSLT se prema tome često koristi van konteksta XSL-a, npr. za generisanje HTML veb-stranice koriseći podatke date u XML-u.

- Napredne mogućnosti stiliziranja, izražene XML doktajpom koji definiše skup elemenata koji se naziva **Objekti za formatiranje** (*Formatting Objects*), i atribute (delimično pozajmljeni od CSS2 svojstava, uz neke dodate složenije).

Stiliziranje zahteva izvorni XML dokument, koji sadrži informacije će stajlšit prikazati i sâm stajlšit koji opisuje kako treba prikazati dokument datog tipa.

```
<scene>
    <FX>General Road Building noises.</FX>
    <speech speaker="Prosser">
        Come off it Mr Dent, you can't win
        you know. There's no point in lying
        down in the path of progress.
    </speech>
    <speech speaker="Arthur">
        I've gone off the idea of progress.
        It's overrated
    </speech>
</scene>
```

Ovaj XML fajl ne sadrži nikakve informacije o prezentaciji – one se nalaze u stajlšitu. Razdvajanje sadržaja dokumenta i informacija o stilu dokumenta dopušta prikaz istog dokumneta na različim medijumima (ekran, papir, mobilni telefon, itd). Pored toga, omogućava da više različitih XML dokumenata imaju isti stajlšit bez dupliranja podataka nad svakim dokumentom. Slede neki delovi stajlišta koji bi mogao biti povezan za gorenavedeni XML.

```
<xsl:template match="FX">
    <fo:block font-weight="bold">
        <xsl:apply-templates/>
    </fo:block>
</xsl:template>

<xsl:template match="speech[@speaker='Arthur']">
    <fo:block background-color="blue">
        <xsl:value-of select="@speaker"/>:
        <xsl:apply-templates/>
    </fo:block>
</xsl:template>
```

Za razliku od CSS-a, koji koristi svoju notaciju, XSL koristi već poznatu XML notaciju. Većina modernih pretraživača podržavaju i CSS i XSLT. Pored toga, postoje mnoge druge XSLT implementacije.

XSL nije zamena za CSS: XSL je namenjen složenom formatiranju gde bi sadržaj dokumenta trebalo predstaviti na više mesta (na primer, naslovi treba da postanu deo sadržaja); CSS je namenjen dinamičkom formatiranju onlajn dokumenata za više medijuma i striktno je deklarativan, što ga čini relativno ograničenim, mada jednostavnim za korišćenjem, generisanjem i modifikovanjem. Mogu se koristiti i u kombinaciji: XSL na serveru za pojednostavljivanje XML dokumenata, a CSS na klijentu za stilizovanje.

Pitanje 22 JSON vs XML.

Odgovor. Slede primeri JSON objekta i njegovog ekvivalenta u XML-u.

```json
{"employees": [
    {"firstName": "John",  "lastName": "Doe"},
    {"firstName": "Anna",  "lastName": "Smith"},
    {"firstName": "Peter", "lastName": "Jones"}
]}
```

```xml
<employees>
    <employee>
        <firstName>John</firstName>  <lastName>Doe</lastName>
    </employee>
    <employee>
        <firstName>Anna</firstName>  <lastName>Smith</lastName>
    </employee>
    <employee>
        <firstName>Peter</firstName> <lastName>Jones</lastName>
    </employee>
</employees>
```

JSON je format za razmenu podataka, a XML je jezik.

Pitanje 23 JSON.

Odgovor. JSON (**JavaScript Object Notation**, čita se *džejson*) je lak (lightweight) format za razmenu podataka. Ljudima je lako da ga čitaju i pišu, a računarima da ga analiziraju i generišu. Prvi ga je specifikovao i popularizovao Daglas Krokford.

U pitanju je tekstualni format podataka koji je potpuno nezavistan od jezika, ali koristi konvenzije koje su poznate programerima koji programiraju

u jezicima iz C-porodice (C, C++, C#, Java, JavaScript, Perl, Python, itd). Zvaničan internet medija tip je `application/json`, a ekstenzija je `.json`.

Zasniva se na dve strukture:

- **Skup** parova ime-vrednost. U zavisnosti od jezika, ovo može biti realizovano kao **objekat**, slog, struktura, rečnik, heš-tablea, lista sa ključevima ili asocijativni niz.
- **Uređena lista** vrednosti. U većini jezika ona je realizovana kao **niz**, vektor, lista ili sekvenca.

JSON je format koji se najčešće koristi za asinhronu razmenu podatka između brauzerā i serverā, AJAJ (mada je u široj upotrebi termin AJAX, iako se ne korsiti XML već JSON).

Pitanje 24 AJAX.

Odgovor. Ajaks (AJAX, *ejdžeks*, skraćenica od **asinhroni JavaScript i XML**) je skup tehnika za razvoj na vebu koje koriste veb-tehnologije na klijentskoj strani radi kreiranja asinhronih veb-aplikacija. Koristeći Ajaks, veb-aplikacije mogu da asinhrono šalju i primaju podatke od servera, bez uticaja na trenutni izgled i trenutno ponašanje stranice. Sloj razmene podataka je u potpunosti odvojen od prezentacionog sloja, što znači da je moguće dinamički menjati sadržaj stranice bez potrebe za ponovnim učitavanjem cele. Uprkos svom imenu, ne mora se koristiti XML – često se koristi JSON u AJAJ varijanti.

Ajaks nije tehnologija, već **skup tehnologija**. HTML i CSS se mogu zajedno koristiti za obeležavanje i stilizovanje informacija. DOM-u se pristupa pomoću JavaScripta radi dinamičkog prikaza informacija. JavaScript i objekat `XMLHttpRequest` pružaju metode za asinhronu razmenu podataka između brauzera i servera.

Tokom ranih 1990-ih, većina veb-stranica je bila bazirana samo na HTML dokumentima. Svaka interakcija korisnika sa stranicom koja za posledicu ima promenu nekog dela stranice zahtevala je ponovno slanje čitave stranice. Godine 1996, Internet Explorer je uveo tag `<iframe>` koji je dopuštao asinhrono učitavanje sadržaja, a 1999. je ta tehnologija iskorišćena za dinamičko ažuriranje vesti i citata na podrazumevanoj početnoj stranici na Internet Exploreru. Uz IE5 je uvedena kontrola XMLHTTP, koju su kasnije Mozilla, Safari i Opera pretočili u JavaScript objekat `XMLHttpRequest`. Počev od IE7, i Majkrosoft je adaptirao `XMLHttpReuqest`. Stara ActiveX verzija je i dalje dostupna u novijim verzijama IE-a, ali ne i u Microsoft Edge.

Gugl je 2004. i 2005. načinio veliki korak iskoristivši asinhrone tehnologije za Gmail i Google Maps. Sajt Kayak.com je u svojoj beta verziji koristio slične tehnologije, u to vreme zvane "ona xml http stvar". Termin *ajax* je u javnosti prvi put iskorišćen 18. februara 2005 u članku *Ajax: Novi pristup veb-aplikacijama*, gde su opisane tehnike korišćene na Guglovim aplikacijama.

Ajaks zahtev se može jednostavno poslati korišćenjem JS frejmvorka jQuery. Za serversku stranu je u primeru iskorišćen PHP, mada se može raditi o bilo kojoj drugoj tehnologiji.

```
$.get('send-ajax-data.php', function(data) {
    alert(data);
}); $
```

```
<?php
header('Content-Type: text/plain');
echo "This is the returned text.";
?>
```

2.3 Serverske tehnologije

Pitanje 25 CGI.

Odgovor. CGI (**Common GateWay Interface**) je standardni način na koji veb-serveri mogu sarađivati sa izvršnim programima instaliranim na serveru radi dinamičkog kreiranja veb-stranica. Takvi programi se zovu CGI skripte i najčešće su napisati nekim skritpnim jezikom, mada mogu biti napisani i bilo kojim programskim jezikom.

Svaki veb-server pokreće HTTP serverski softver, koji odgovara na zahteve koje dobija od brauzerā. Uopšteno, HTTP server ima direktorijum koji predstavlja kolekciju dokumenata – fajlova koji mogu biti poslati brauzerima koji su povezani na ovaj server. Na primer, ako veb-server ima domen `example.com`, a kolekcija dokumenata se čuva na `/usr/local/apache/htdocs` u lokalnom fajl-sistemu, onda će veb-server na zahetev za stranicom na adresi `http://example.com/index.html` odgovoriti brauzeru slanjem (ranije već napisanog) fajla `/urs/local/apache/htdocs/index.html`.

CGI sistem proširuje ovaj pristup tako što dopušta vlasniku veb-servera da direktorijum u kolekciji dokumenata posveti izvršnim skriptama (ili binarnim fajlovima) umesto već napisanih stranica – ovo je CGI direktorijum; na primer `/usr/local/apache/htdocs/cgi-bin`. Ako brauzer zahteva URL koji pokazuje na fajl u CGI direktorijumu (npr. Pajton skripta na lokaciji

`http://example.com/cgi-bin/printenv.pl`), umesto da server samo pošalje nazad taj fajl, HTTP server će pokrenuti specificiranu skriptu, a brauzeru će vratiti izlaz skripte. Drugim rečima, sve što skripta šalje na standardni izlaz se prosleđuje veb-klijentu umesto da se ispiše na terminalu.

Sledi primer jednostavnog CGI programa u Pajtonu, sa pridruženim HTML-om. Program se koristi za izračunavanje zbira vrednosti dva broja unesena u polja za unos.

```
<!DOCTYPE html>
<html>
 <body>
  <form action="add.cgi" method="POST" >
   Enter two numbers to add:<br />
   First Number: <input type="text" name="num1" /><br />
   Second Number: <input type="text" name="num2" /><br />
   <input type="submit" value="Add" />
  </form>
 </body>
</html>
```

```
#!/usr/bin/python

import cgi
import cgitb
cgitb.enable()

input_data=cgi.FieldStorage()

print 'Content-Type:text/html' #HTML is following
print                          #Leave a blank line
print '<h1>Addition Results</h1>'
try:
  num1=int(input_data["num1"].value)
  num2=int(input_data["num2"].value)
except:
  print '<p>Sorry, we cannot turn your inputs into numbers (integers).</p>'
  return 1
sum=num1+num2
print '<p>{0} + {1} = {2}</p>'.format(num1,num2,sum)'
```

Pitanje 26 PHP, opšte.

Odgovor. PHP je serverski **skriptni jezik** dizajniran za razvoj na vebu, mada se može koristiti i kao programski jezik opšte namene. Kreiran je 1994. godine. Prvobitno je PHP je bila skraćenica od *Personal Home Page*, ali je sada u pitanju rekurzivni bekronim *PHP: Hypertext Preprocessor*. Može

se ugraditi u HTML kod, ili se može kroistiti u kombinaciji sa nekim šablonskim sistemima (npr. Smarty) i drugim frejmvorcima. PHP kod se obično procesira pomoću **PHP interpretatora** koji je implementiran kao modul na veb-serveru ili kao CGI izvršni fajl. Veb-server kombinuje rezultate interpretiranog i izvršenog PHP koda (što može biti bilo koji tip podataka, uključujući slike) sa generisanom veb-stranicom. PHP se sve do 2014. godine razvijao bez zvanične formalne specifikacije i bez standarda.

PHP interpretator izvršava samo kod koji se nalazi unutar **delimiterā**. Najčešće se koriste `<?php` i `?>`, mada postoji i skraćena verzija otvorenog delimitera, `<?`. Postoji i takozvana eho-skraćenica otvorenog taga, `<?=`, što je ekvivalentno sa `<?php echo` (`echo` je funkcija za štampanje).

Sve promenljive imaju prefiks `$` (npr. `$var`), a tip podatka ne mora da bude unapred specificiran. Za razliku od imena funkcija i klasa, imena promenljivih prave razliku između velikih i malih slova. Korišćenje navodnika (") umesto apostrofa (') pri radu sa stringovima omogućava interpolaciju vrednosti promenljive u string. Sintaksa podseća na C.

> **Pitanje 27** PHP-ove superglobalne promenljive.

Odgovor. Superglobalne promenljive su postale deo PHP-a sa verzijom 4.1.0. To su ugrađene promenljive koje su **uvek dostupne** – može im se pristupiti iz bilo koje funkcije, klase ili fajla. Ima ih devet: `$GLOBALS`, `$_SERVER`, `$_REQUEST`, `$_POST`, `$_GET`, `$_FILES`, `$_ENV`, `$_COOKIE` i `$_SESSION`.

`$GLOBALS` se koristi za čuvanje svih globalnih promenljivih iz bilo kog dela PHP skripte (kao i iz funkcija i metoda). U sledećem primeru, `$x` i `$y` su globalne promenljive, pa im se iz funkcije može pristupiti preko `$GLOBALS['x']` i `$GLOBALS['y']`. Takođe, kako je rezultat smešten u globalnu promenljivu, on se kasnije može koristiti bez pristupa nizu `$GLOBALS`.

```
1  <?php
2  $x = 75; $y = 25;
3  function addition() {
4      GLOBALS['z'] = $GLOBALS['x'] + $GLOBALS['y'];
5  }
6  addition();
7  echo $z; // 100
8  ?>
```

`$_SERVER` sadrži informacije o hederima, putanjama i lokacijama skripti. Na primer, `$_SERVER['SERVER_NAME']` vraća ime servera (što može biti npr. `www.elfak.ni.ac.rs`).

`$_REQUEST` se koristi za prikupljanje podataka nakon prosleđivanja HTML forme, kao u sledećem primeru. Kada korisnik prosledi formu, podaci iz nje se šalju fajlu specificiranom pod atributom `action` u tagu `<form>` (što je u ovom primeru sâm taj fajl).

```
<html>
  <body>
    <form method="post" action="<?php echo $_SERVER['PHP_SELF'];?>">
      Name: <input type="text" name="fname">
      <input type="submit">
    </form>
    <?php
      if ($_SERVER["REQUEST_METHOD"] == "POST") {
        // collect value of input field
        $name = $_REQUEST['fname'];
        if (empty($name)) echo "Name is empty";
        else echo $name;
      }
    ?>
  </body>
</html>
```

Slično, `$_POST` i `$_GET` služe za sakupljanje podataka iz formi nakon njihovog prosleđivanja.

> **Pitanje 28** Prenos podataka sa klijenta na server.

Odgovor. Prenos podataka preko interneta počinje **događajem**. Događaj može biti rezultat interakcije čoveka sa računarom; npr. pokretanje brauzera (*klijentski* program) i zahtevanje neke informacije (tipično HTML stranice) koja se nalazi na udaljenom računaru. Pored toga, događaj može biti generisan i instrukcijama u programu; npr. može se automatizovati proces preuzimanja i kačenja fajlova.

Recimo da je korisnik zahtevao HTML stranicu sa udaljenog računara koristeći brauzer. Brauzer traži udaljeni ruter i, kada ga pronađe, predaje zahtev programu koji se naziva **server** na udaljenom računaru. Server onda proverava zahtev i pokušava da locira HTML fajl na disku i kada ga nađe, šalje ga nazad. Ako stranica uz sebe ima i multimedijalan sadržaj (slike, video, audio) informacije o njima se takođe šalju brauzeru.

Kada primi podatke od servera, klijent (u ovom slučaju brauzer) prikazuje veb-stranicu. Server nema učinka u samom priakzu – on samo prosleđuje podatke. Nakon prenosa svih informacija, veza između klijenta i servera se gubi. Ako klijent ponovo zahteva iste podatke, ceo ciklus će se odviti iznova;

ovo znači da je klijent-server interkacije stejtles, jer svaki novi zahtev kao rezultat ima novi odgovor.

Pitanje 29 PHP, klase i objekti.

Odgovor. Osnovna definicija klase počinje ključnom rečju `class`, za kojom seldi ime klase, zatim par vizičastih zagrada koji obuhvataju definicije svojstava i metoda koje pripadaju klasi. Ime klase može biti bilo koja validna labela, pod uslovom da nije rezervisana reč PHP-a. Validno ime klase počinje slovom ili donjom crtom, i zatim sledi bilo koji broj cifara, slova i donjih crta. Klasa može da sadrži svoje konstane, promenljive (svojstva) i funkcije (metode). Pseudo-promenljiva `$this` je dostupna kada se metoda zove iz konteksta nekog objekta. `$this` je referenca na objekat koji se zove (oično objekat kome metoda pripada).

```php
<?php
class SimpleClass
{
    // deklaracija svojstva
    public $var = 'podrazumevana vrednost';

    // deklaracija metode
    public function displayVar() {
        echo $this->var;
    }
}
?>
```

Instanca klase kreira korišćenjem ključne reči `new`. Objekat će se uvek kreirati osim ako objekat ima definisan konstruktor koji može da baci izuzetak zbog greške. Klase treba da budu definisane pre instanciranja.

```php
<?php
$instance = new SimpleClass();

// Može i pomoću stringa.
$className = 'SimpleClass';
$instance = new $className(); // new SimpleClass()
?>
```

Konstruktor se definiše korišćenjem magične metode `__construct`. Može se definsiati i **destruktor** (`__destruct`) koji se poziva kada više nema referenci na određeni objekat, ili tokom sekvence gašenja, u bilo kom redosledu.

Vizibilitet svojstva ili metode se može definisati prefiksom u deklaraciji, korišćenjem ključnih reči `public` (javni), `protected` (zaštićeni) i `private` (privatni). Svojstva klase se moraju definisati kao javni, zaštićeni ili privatni.

Ako se svojstvo deklariše pomoću `var`, a funkciji se izostavi prefiks, biće definisani kao javni. Objekti istog tipa će imati pristup privatnim i zaštićenim članovima jedni drugih iako nisu deo iste instance.

Pitanje 30 PHP, nasleđivanje.

Odgovor. Klasa može da nasledi metode i svojstva druge klase korišćenjem ključne reči `extends` u deklaraciji klase. Nije moguće naslediti više klasa; klasa može da ima samo jednu osnovnu klasu.

Nasleđene metode i svojstva se mogu **preklopiti** njihovom ponovnom deklaracijom sa istim imenom definisanim u roditeljskoj klasi. Međutim, nije moguće preklopiti metodu koja je u roditeljskoj klasi definisana kao `final`. Pristup preklopljenim metodama ili statičkim svojstvima je moguć korišćenjem `parent::`. Kod preklapanja metoda, potpis parametara treba da ostane isti, inače će PHP generisati `E_STRICT` grešku. Ovo ne važi i za konstruktore; dopušteno je preklapanje korišćenjem različitih parametara.

Roditeljski konstruktori se ne zovu implicitno ako nasleđena klase definiše konstruktor. Potrebno načini eksplicitni poziv `parent::__construct()` unutar konstruktora nasleđene klase. Ako nasleđena klasa ne definiše konstruktor, on će biti nasleđen kao i svaka druga metoda iz roditeljske klase, sem ako nije deklarisan kao privatni.

Pitanje 31 ORM alati.

Odgovor. ORM (**Object-Relational Mapping**) je programska tehnika za pretvaranje podataka objektno-orijentisanim programskim jezicima između međusobno nekompatiblnih struktura. Na ovaj način se kreira "virutelna objekta baza podataka" koja se može koristiti iz programskog jezika.

U objekto-orijentisanom programiranje, upravljanje podacima bazira se na objektno-orijentisanim objektima koji su skoro uvek ne-skalarne veličine. Na 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 progmrakog jezika, i mže mu se pristupiti jednom promenljivom (na primer, pokazivačem koji pokazuje na ubjekat). 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 konverzuje pri upisu 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 atomičku 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**.

```
// Bez mapiranja
String sql = "SELECT ... FROM persons WHERE id = 10";
DbCommand cmd = new DbCommand(connection, sql);
Result res = cmd.Execute();
String name = res[0]["FIRST_NAME"];

// Sa mapiranjem
Person p = repository.GetPerson(10);
String name = p.getFirstName();
```

2.4 Veb-servisi

> **Pitanje 32** Veb-servisi.

Odgovor. W3C definiše veb-servis kao "softverski sistem dizajniran tako da podrži interoperabilnu interakciju između mašina preko mreže". Za razmenu podatka se koristi XML ili JSON.

Veb-servis obično pruža objektno-orijentisani interfejs serveru baze podataka, koji se koristi od strane drugog veb-servera ili mobilne aplikacije, koji nudi korisnički interfejs krajnjem korisniku. Poznati primeri su Google Maps API i EBay Shopping API.

Veb-servisi mogu biti SOAP (Simple Object Access Protocol, tipičan primer `service1.getPerson()`) ili REST (Representational State Trasnfer, tipičan primer `http://example.com/persons`).

Šta još?

Pitanje 33 RESTful servisi.

Odgovor. REST (**Representational State Transfer**) je arhitekturni stil koji se sastoji od koordinišućeg skupa komponenti, konektora i elemenata u distribuiranom hipermedija sistemu, gde je fokus na ulogama komponenti i određenom skupu interakcija među elementima, a ne na implementacionim detaljima. REST je arhitekturni obrazac koji se koristi na WWW-u. Predstavio ga je Roj Filding 2000. godine.

Ima za svrhu da poboljša sledeće:

- **performanse** – interakcija komponenti može biti dominantan faktor u korisnikovom doživljaju o performansama,
- **prilagodljivost** (skalabilnost) – treba da može da podrži veliki broj komponenti i interakcijā među njima,
- **jednostavnost** interfejsā,
- **promenljivost** (modifikabilnost) komponenti kako bi se udovoljilo zahtevima za promenama, čak i dok je aplikacija pokrenuta,
- **vidljivost** (vizibilitet) komunikacije među komponentama od strane servisnih agenata,
- **prenosivost** (portabilnost) komponenti prenošenjem programskog koda sa podacima, i
- **pouzdanost**, tj. otpornost na otkaz na nivou celog sistema u prisustvu otkaza njegovih pojedinih komponenti, konektora ili 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 resursā, kao što su tekst, JSON ili XML. Danas je JSON najpopularniji format.

Za veb-aplikacije koje su u skladu sa gorenavedenim osobinama se kaže da su RESTful.

Pitanje 34 SOAP servisi.

Odgovor. SOAP (**Simple Object Access Protocol**) je protokol računarskih mreža za razmenu struktuiranih informacija u implementaciji veb-servisa. Koristi XML Information Set kao format poruka. Za transmisiju i pregovore se oslanja se na protokole apliaktivnog nivoa, najčešće HTTP ili SMTP.

SOAP može biti osnovni sloj za stek protokola veb-servisa, jer može da pruži osnovnu frejmvork za razmenu poruka veb-servisima. Baziran je na XML-u i sastoji se od tri dela:

- **koverta**, koja definiše strukture poruke i načinna koji se one obrađuju,
- **skup pravila** za šifrovanje za iskazivanje instanci tipova podatak definisanih od strane aplikacije i
- **konvenciju** za reprezentaciju poziva procedura i odgovora.

Tri glavne karakteristike SOAP-a su sledeće:

- **proširljivost**,
- **neutralnost** (može koristiti bilo koji protokol za transport, kao što su HTTP, SMTP, TCP, UDP, itd) i
- **nezavisnost** (dozvoljava bilo koji programski model).

Na primer, aplikacija može da pošalje SOAP zahtev serveru na kome su omogućeni veb-servisi – kao što je baza podataka sa cenama nekretnina – sa parametrima pretrage. Server zatim vraća SOAP odgovor (dokument formatiran kao XML koji sadrži rezultate), npr. cenu, lokaciju i osobine. Kako generisani podaci dolaze u standardizovanom formatu koji mašine mogu parsirati, aplikacija koja je pružila zahtev može direktno integrisati odgovor.

```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: 299
SOAPAction: "http://www.w3.org/2003/05/soap-envelope"

<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
  <soap:Header>
  </soap:Header>
  <soap:Body>
    <m:GetStockPrice xmlns:m="http://www.example.org/stock/Surya">
      <m:StockName>IBM</m:StockName>
    </m:GetStockPrice>
  </soap:Body>
</soap:Envelope>
```

Prednosti SOAP-ova neutralnost ga ekspliticno čini pogodnim za korišćenje sa bilo kojim transportnim protokolom. Implementacije često koriste HTTP, ali se mogu koristiti i drugi (SMTP). SOAP, kad ase kombinuje sa HTTP post/odgovor razmenama, lako kanališe kroz postojeće fajervolove i proksije, i stoga ne zahteva modifikaciju viroko rasprostranjenih komunikacionih infrastruktura koje već postoje za obradu HTTP razmena.

Mane Kada se koriste standardne implementacije i podrazumevane kombinacije SOAP/HTTP, XML infoset je serijalizovan kao XML. Ako je neophodno ugraditi binarne objekte u XML, radi boljih performansi se koristi MTOM (Message Transmission Optimization Mechanism).

Pitanje 35 RPC.

Odgovor. 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 objektno-orijentisanom 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 proed 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. Udaljenin server šalje odgovor klijentu, i aplikacija nastavlja voj proces. Dok server obrađuje poziv, klijent je blokiran (čeka dok mu server ne odgovori), osim ako je zahtev bio psolat 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 neznajući 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 dokom RPC-a je sledeća.

1. Klijent poziva klijentski stab (*stub*). To je deo kdoa koji se kroisti za konvertovanje parametara koji se prenose tokom RPC-a. Na primer, to može biti prevođenje celih brojeva iz adresiranja glave u adresiranje repa. Ovaj poziv je klasičan lokalni poziv procedure, i parametri se potavlja na stek na uobičajen način.
2. Klijentska stab procedura pakuje parametre u poruku i inicira sistemski poziv kako bi se poruka poslala. Pakovanje parametara se naziva maršaling (marshalling).
3. Klijentov lokalni operativni sistem šalje poruku od klijentske mašine ka serverskoj mašini.
4. Lokalni operativni sistem na serveru mašine prenosi dobijene pakete na serverski stab.
5. Serverski stab raspakuje parametre iz poruke (tzv. demaršaling).
6. Konačno, serverski stab poziva proceduru. Odgovor prati iste korake, samo u obrnutom redosledu.

Pitanje 36 Java, opšte.

Odgovor. Java je računarski programski jezik opšte namene. Konkurentan je, baziran na klasama, objektno-orijentisan i posebno dizajniran tako da ima

što je manje moguće implementacionih zavisnosti. Napisan je s namerom da dozvoli pisanje kodova pod mantrom "napiši jednom, pokreni svuda" (*write once, run anwhere*, WORA), što znači da iskompajliran Java kod može da se porkene na svim platformama koje podržavaju Javu, bez potrebe za rekompajliranjem. Java aplikacije se obično kompjaliraju u **bajtkod** koji se može pokrenuti na svakoj **Java virtuelnoj mašini** (JVM), bez obzira na arhitekturu računara. Prvu verziju je razvio Sun Microsystems 1995. godine, a od 2010. godine je vlasništvo korporacije Oracle.

Sintaksa Jave umnogome podseća na C++. Ipak, dok C++ kombinuje sintaksu za strukturno, generičko i objektno-orijentisano programiranje, Java je izgrađena s namerom da bude isključivo **objektno-orijentisani jezik**. Sav kod se piše unutar klasa. Svaki podatak je objekat (nema pokazivača), izuzev primitivnih tipova podataka (celi brojevi, brojevi s pokretnom zapetom, logičke vrednosti i karakteri), koji nisu objekti zbog performansi. Pored toga, Java ne podržava preklapanje operatora niti višestruko nasleđivanje klasa, mada je višestruko nasleđivanje interfejsā (tzv. implementacija interfejsā) dozvoljena. Ima automatsko upravljanje memorijom, tj. postoji *garbage collector*.

Primer *Hello, world!* programa je dat u nastavku.

```
1 class HelloWorldApp {
2     public static void main(String[] args) {
3         // Štampa string u konzolu.
4         System.out.println("Hello World!");
5     }
6 }
```

Pitanje 37 Java virtuelna mašina.

Odgovor. **Virtuelna mašina** predstavlja simulaciju mašine koja se obično razlikuje od ciljne mašine. Virtuelne mašine mogu biti zasnovane na specifikacijama hipotetičkog računara ili mogu da emuliraju arhitekturu računara i funkcije "stvarnog" kompjutera.

JVM (**Java Virtual Machine**) je apstraktna mašina za izračunavanje koja omogućuje računaru da pokrene Java program. Postoje tri predstave JVM-a: specifikacija, implementacija i instanca.

Specifikacija je dokument koji formalno opisuje šta se zahteva od JVM implementacije. Postojanje jedinstvene specifikacije obezbeđuje to da su sve implementacije interoperabilne. Specifikacija namerno obilazi implementacione detaljee koji nisu od važnosti za interoperatiblnost, npr. algoritam koji se

koristi za *garbage collection*. JVM **implementacija** je računarski program koji ispunjava zahteve postavljene JVM specifikacijom. **Instanca** JVM-a je implementacija pokrenuta u vidu procesa koji izvršava računarski program iskompajliran u Java bajtkod.

JRE (Java Runtime Environment) je softverski paket koji sadrži sve što je potrebno za pokretanje Java programa. Sadrži implementaciju JVR-a zajedno sa implementacijom JCL-a (Java Class Library). **JDK** (Java Development Kit) je nadskup JRE-a i sadrži alte za Java programere, npr. kompajler `javac`.

Još od ranih faza razvitka, Java (i JVM) su reklamirane kao veb-tehnologija za kreiranje RIA (Rich Internet Applications). Na klijentskoj strani, brauzeri mogu biti nadograđeni NPAPI Java plaginom koji izvršava takozvane **Java aplete** ugređene u HTML stranice. Apletu je dopušteno crtanje u pravougaonom prostoru stranice koji mu je dodeljen i koristi ograničen skup API-ja koji npr. pružaju mogućnost korišćenja mikrofona ili 3D hardverske akseleracije. Java apleti su po performansama i mogućnostima bili superiorniji od JavaScripta sve do 2011, kada su JavaScript endžini u brauzerima postali značajno brži (između ostalog zbog JIT-a) i kada je HTML5 počeo da obogaćuje JavaScript novim API funkcijama.

Od aprila 2014, Google Chrome ne dopušta korišćenje NPAPI plagina. Mozilla Firefox će takođe zabraniti korišćene NPAPI-ja krajem 2016. Oracle je januara 2016. objavio da će plagin biti zvanično ukinut uz izdanje Jave 9.

Pitanje 38 Java biblioteke.

Odgovor. Java biblioteke se mogu svrstati u tri grupe.

- **Osnovne biblioteke** služe za rad sa kolekcijama (liste, stabla, redovi, dekovi), za obradu XML-a, za bezbednost i za lokalizaciju.
- **Biblioteke za integraciju** se koriste za rad sa bazom (JDBC, Java DataBase Connectivity), za pretragu i otkrivanje (JNDI, Java Namind and Directory Interface), za rad u distribuiranom okruženju (RMI i CORBA) i za monitoring aplikacija (JMX).
- Od **GUI biblioteka**, najdominantnije su AWT (Abstract Window Toolkit) i Swing.

Sledi spisak nekih često korišćenih biblioteka, sa nekim često korišćenim klasama i interfejsima iz njih.

- `java.lang` se koristi za sve osnobne klase koje se automatski (implicitno) importuju (sadrži klase `String`, `Integer`, `Double`, `Byte`, `Long`, `Math`, `Short`, `Thread`, `Void`, itd).
- `java.util` sadrži mnoge korisne strukture podataka (`Collection<E>`, `Enumeration<E>`, `Iterator<E>`, `Map<K, V>`, `Observer`, `Set<E>`, `SortedSet<E>`, itd).
- `java.io` se koristi za čitanje fajlova na niskom nivou, dok je osnovna podrška za čitanje je dostupna iz `java.util.Scanner`.
- `java.math` se koristi za matematičke operacije sa proizvoljnom tačnošću (`BigDecimal`, `BigInteger`, itd).
- `java.net` se koristi za sokete, konekcije, itd (`SocketOption<T>`, `ProtocolFamily`, `CookieStore`, `URLStreamHandlerFactory`, itd).
- `javax.swing` sadrži komponente za crtanje, tj. za kreiranje GUI-ja (npr. `Action`, `Icon`, `MenuElement`, `Painter<T>`, `Box`, `ButtonGroup`, `JToggleButton`, `ProgressMonitorInputSteram`, itd).

Pitanje 39 JavaBeans

Odgovor. JavaBeans su klase koje enkapsuliraju mnoge objekte u jedan (zrno, *bean*). Svaka klasa koja ima sledeće tri karakteristike smatra se da pošutje Beans standard:

- mogu se **serijalizovati** (ovo omogućava aplikacijama i frejmvorcima da pouzdano čuvaju, skladište i čitaju stanje zrna na način koji je potpuno nezavistan od VM-a i platforme),
- imaju **podrazumevani konstruktor** (konstruktor bez argumenata) i
- dopuštaju **pristup svojstvima** samo preko *set* i *get* (ili *is* za logičke promenljive) metoda (poštujući konvencije o imenovanju ovih metoda, omogućena je automatska inspekcija zrnā iz editora).

Prednosti Svojstva, događaji i metode u zrnu koje su dostupne drugim aplikacijama se mogu kontrolisati. Sem toga, zrno može da registruje i prima događaje od drugih objekata, kao i da generiše događaje koji se mogu poslati drugim objektima. Konfiguraciona podešavanja zrna se mogu **trajno sačuvati** i obnoviti.

Mane Klasa sa konstruktorom bez argumenata može biti **inicijalizovana u nevalidnom stanju**. Ako takvu klasu progarmer instancira ručno (a ne automatski, putem nekog frejmvorka), programer možda ne shvati da ju je nepravilno inicijalizovao. Kompajler ne može da detektuje takav problem, čak i ako je on dokumentovan. Pored toga, kreiranje getera i setera za svako svojstvo vodi do mnogo **bojlerplejt** koda.

Primer JavaBean klase i njena upotreba u JSP-u je dat u nastavku.

```
package player;

public class PersonBean implements java.io.Serializable {
    private String name = null;
    private boolean awesome = false;

    public PersonBean() {
    }

    public String getName() { return name; }
    public void setName(final String value) {
        name = value;
    }

    public boolean isAwesome() { return awesome; }
    public void setAwsome(final boolean value) {
        awesome = value;
    }
}
```

```
<% // Use of PersonBean in a JSP. %>
<jsp:useBean id="person" class="player.PersonBean" scope="page"/>
<jsp:setProperty name="person" property="*"/>

<html>
    <body>
        Name: <jsp:getProperty name="person" property="name"/><br/>
        Awesome? <jsp:getProperty name="person" property="awesome"/><br/>
        <br/>
        <form name="beanTest" method="POST" action="testPersonBean.jsp">
            Enter a name: <input type="text" name="name" size="50"><br/>
            Choose an option:
            <select name="awesome">
                <option value="false">Very awesome</option>
                <option value="true">Not really awesome at all</option>
            </select>
            <input type="submit" value="Test the Bean">
        </form>
    </body>
</html>
```

Pitanje 40 JSP.

Odgovor. JSP (**JavaServer Pages**) je serverska tehnologija koja pomaže developerima da kreiraju dinamički generisane veb-stranice bazirane na HTML-u, XML-u ili drugim vrstama dokumenata. Izdat je 1999. godine od strane Sun Microsystems, i sličan je PHP-u i ASP-u, ali koristi programski jezik Java. Za pokretanje JSP-a neophodan je veb-server sa servlet kontejnerom, kao što je Apache Tomcat ili Jetty.

S arhitekturne tačke gledišta, JSP se može posmatrati kao apstrakcija Java servletā[4] na visokom nivou. JSP-ovi se prevode tokom izvršenja; svaki JSP servlet se kešira i ponovo se koristi sve dok se prvobitni JSP ne izmeni. JSP se može koristiti nezavisno ili kao *view* komponenta MVC modela, pri čemu JavaBeans obično predstavlja model, a Java servlet – kontroler. Ovo je vrsta Model 2 arhitekture.

JSP dozvoljava da Java kôd i neke predefinisane akcije budu izmešane sa statičkim obeležanim sadržajem, kao što je HTML; pri tome se rezultuja stranica kompajlira i izvršava na serveru, čiji je rezultat gotova HTML stranica. Kompajlirane stranice, kao i Java biblioteke od kojih kôd zavisi, sadrže Java bajtkod, a ne mašinski kod. Kao i svaki drugi Java program, moraju se izvršiti unutar JVM-a koji interaguje sa operativnim sistemom instaliranim na serveru. Na taj način je obezbeđeno apstraktno okruženje nezavisno od platforme.

JSP se obično koristi za generisanje HTML i XML dokumenata, ali se pomoću klase `OutputStream` mogu dobiti i drugi tipovi podataka.

Koristi se nekoliko **delimitera** za pisanje skripti (funkcija). Najosnovniji su `<%` i `%>`, koji omeđuju JSP **skriptlet**. Skriptlet je deo Java koda koji se pokreće kada korisnik uputi zahtev za stranicom. Pored njega, često se koriste i `<%=` i `%>` za **izraze**, gde se skriptlet i delimiteri koji ga okružuju zamenjuju rezultatom koji daje navedeni izraz; koriste se i `<%@` i `%>` kojima se obeležavaju **direktive**.

Java kod ne mora da se ceo nađe u jednom bloku omeđenom delimiterima, što pokazuje sledeći primer.

[4]Servleti su programi koji proširuju funkcionalnost servera. Iako mogu da obrade bilo koji tip zahteva, najčešće implementiraju aplikacije na veb-serveru.

```
<p>Brojanje:</p>
<% for (int i = 0; i <3; i++) { %>
    <p>Ovaj broj je <%= i %>.</p>
<% } %>
<p>Gotovo.</p>
```

Pitanje 41 Java apleti.

Odgovor. Java aplet je mala aplikacija napisana u Javi, dostavljena korisnicima u obliku bajtkoda. Korisnik pokreće Java aplet iz veb-stranice, aplet se izvršava u JVM-u u procesu koji je odvojen od samog brauzera, slično kao Adobe Flash. Java aplet se može pojaviti u frejmu na stranici ili kao novi aplikacioni prozor. Zahtevaju da korisnik ima instaliran JRE.

Java apleti su predstavljeni još uz prvu verziju programskog jezika Java, 1995. godine. Mogu se napisati u bilo kom programskom jeziku koji se može iskompajlirati u Javin bajtkod. Mada je to najčešće Java, mogu se koristiti i Jython, JRuby, Pascal, Scala ili Eiffel.

Java apleti se izvršavaju veoma brzo, i do 2011. su bili nekoliko puta brži od JavaScripta. Za razliku od JavaScripta, apleti imaju pristup 3D hardverskoj akseleraciji (ubrzanju), što ih čini idealnim za netrivijalne i računski zahtevne vizuelzacije. Međutim, brauzeri su dobili podršku za hardverom ubrzanu grafiku zahvaljujući kanvas tehnologiji (konrektno WebGL za 3D grafiku), kao i JIT kompajlere za JS, pa je razlika u brzini postala manje primetna.

Sledeći primer ilustruje korišćenje Java apleta kroz paket `java.applet`. Primer koristi klase iz Javinog AWT-a (Abstract Window Toolkit) za ispisivanje poruke “Hello, World!”.

```
import java.applet.Applet;
import java.awt.*;

// Ovaj fajl treba da se snimi pod imenom "HelloWorld.java".
// Nasleđuje se objekat Applet.
public class HelloWorld extends Applet {

// Crtanje po ekranu.
    public void paint(Graphics g) {
        // Ispis stringa na poziciji (20, 10).
        g.drawString("Hello, world!", 20, 10);
        // Crtanje kruga na poziciji (40, 30).
        g.drawArc(40, 30, 20, 20, 0, 360);
    }
}
```

Posle kompajliranja, rezultujuća `.class` datoteka se može postaviti na verb-serveru i pozvati iz HTML-a koristeći tag `<applet>` ili `<object>`.

```
<!DOCTYPE html>
<html>
<head>
  <title>HelloWorld_example.html</title>
</head>
<body>
  <h1>Primer Java apleta</h1>
  <p>Evo ga aplet:
    <applet code="HelloWorld.class" height="40" width="200">
     Ovde se izvršava HelloWorld.class.
    </applet>
  </p>
</body>
</html>
```

Pitanje 42 JSP Model 2 arhitektura.

Odgovor. JSP Model 2 je kompeksni projektni obrazac koji se koristi pri projektovanju Java veb-aplikacija, u kome se prikaz sadržaja odvaja od logike kojom se pribavljaju podaci i njima manipuliše.

Kako Model 2 razdvaja logiku i prikaz, često se povezuje sa MVC paradigmom. Mada tačan oblik "modela" iz MVC-a nikad nije precizno definisan prema Modelu 2, veliki broj publikacija predlaže formalizovani sloj koji sadrži kôd MVC modela. Model 2 se preporučuje za aplikacije srednje i veće veličine.

U Model 2 aplikacijama, zahtevi iz brauzera klijenta se prenose kontroleru. Kontroler obavlja logiku kojom pribavlja sadržaj koji treba prikazati. Zatim taj sadržaj stavlja u zahtev (obično u obliku JavaBeana) i odlučuje kojem pogledu će proslediti zahtev. Pogled zatim renderuje sadržaj koji mu je prosledio kontroler.