WEB TEORIJA

T1

Port je softverski zadatak kanal kojim komuniciraju aplikacije putem računarskih mreža.

HTTP je skraćenica HyperText Transfer Protocol.

Fundamentalni protokol na kom se bazira rad WWW.

VERZIJE

1.0

1.1 – keep alive connection

2.0

HTTP komunikacija je zasnovana na zahtev/odogvor principu

HTTP je stateless protokol koji ne zahteva od servera čuvanje statusa klijenta ili korisničke sesije klijenta.Ovo se prevazilazi cookijem koji omogućava da se prati zahtevi od istog klijenta,kreirajući stateful protokol iznad http protokola.

U verziji 1.0 po završetku isporuke odgovora klijenta konekcija se zatvara.

U verziji 1.1 konekcija se ne zatvara tj. ostaje otvorena. Klijent će da koristi istu komunikaciju za novi zahtev ka server. Konekcija je otvorena dok jedna od strana ne odluči da je zatvori.

METODI

GET

POST itd

Dodatni redovi sadrže atribute oblika:

Ime: vrednost

AKO JE POST ZAHTEV POSLE PRAZNOG REDA IDU PARAMETRI FORME.

METODE

GET – zahteva resurs od web servera

POST - šalje parametre forme i traži odgovor.

HEAD – zahteva samo HTTP odgovor ,bez slanja samog resursa

PUT – omogućava klijentu da pošalje datoteku web server

OPTIONS – od web servera traži spisak metoda koje podržava

DELETE – omogućava klijentu da obriše resurs sa web servera

GET METODA

- Parametri iz forme se za GET METODU smeštaju u zaglavlje get zahteva.

POST METODA

- Parametra iz forme se za POST metodu smeštaju u telo POST zahteva.

Kod GET metode se parametri forme nalaze u header delu HTTP request poruke.

Kod POST metode se parametri forme nalaze u body delu HTTP request poruke.

Ukoliko je potrebno uz pomoć forme poslati datoteku sa klijenta na server koristi se HTTP Post metoda.

VRSTE WWW SADRŽAJA

- Statički (unapred uskladišteni)
- Dinamički (Generisani po zahtevu)

1.2 Servleti i Tomcat

- Tehnologija za generisanje dinamičkih sadržaja
- Podaci iz HTTP zahtev se posleđuju servletu, on ih procesira i vraća odgovor veb serveru.
- Veb server na osnovu odgovora servleta kreira HTTP odgovor koji se zatim prosleđuje Klijentu

HttpServlet.doGet() – obrada Get zahteva

Apache Tomcat je veb server otvorenog koda razvijen od strane *Apache Software Fondation*.

Tomcat predstavlja veb kontejner koji u potpunosti podržava java veb bazirane aplikacije.

Izgled *apache tomcat* web servera, struktura foldera:

- *bin* - sadrži skripte i exe fajlove koji omogućavaju upravljanje bazičnim radom samog servera (pokretanje i zaustavljanje).
    - Za Windows os iz cmd pokrećemo ***startup.bat***
    - Za Linux os iz terminala pokrećemo ***sh catalina.sh run***
- *conf* – sadrži skripte za podešavanje konfiguracije veb servera.
- *lib* – sadrži biblioteke koje koristi Tomcat prilikom rada. Bibloteke koje se nalaze u ovom folderu dostupne su za sve aplikacije koje su podignute pod Tomacat serverom
    - *servlet-api.jar* je ključna biblioteka koja se koristi u *Eclipse* za nasleđivanje *HttpServlet*
    - možemo ubaciti mysql connector jar
- *logs* - Tomcat upisuje izveštaje tokom svog rada
- temp – se koristi kao pomoćni folder za smeštanje privremenih resursa(fajlova) za sam *Tomcat*, koji nastaju tokom rada *Tomcat*, može da se briše sadržaj kada je T*omcat* isključen
- *work* – se koristi kao pomoćni folder za smeštanje privremenih resursa(fajlova) za aplikacije postavljene na *Tomcat*, koji nastaju prilikom izvršavanja aplikacije, može da se briše sadržaj kada je *Tomcat* isključen
- webapps – najbitiniji folder, zadrži war arhive i web aplikacije

- Opis veb aplikacija za potrebe servera se nalazi u *web.xml*
- Opisuju se mapiranja za Servlete, resursi i konfiguracije, kao i način kako će ih web server koristiti kada odgovara na zahteve klijenata

Klasa HttpServletResponse

- Reprezentuje HTTP odgovor
- Čuva tip odgovora (atribut Content-Type)
    - metoda setContentType(vrednost)
- Čuva cookie (atribut SetCookie)
    - metoda addCookie(cookie)
- **Omogućuje redirekciju (Location) na drugu stanicu**
    - **metoda sendRedirect(nova_lokacija)**
- Podešava proizvoljan atribut zaglavlja

    - metoda setHeader(naziv, vrednost)
- Ugrađuje ID sesije ako cookies nisu uključeni
    - metode encodeURL(url) i encodeRedirectURL(url)
- Čuva izlazni tok podataka

ServletContext

- Reprezentuje kompletnu veb aplikaciju unutar JVM, sadrži sve Servlete
- Deljena memorija za sve Servlete

2.1 Maven,Spring

- Design pattern koji se zasniva na sinhronom radu tri komponente: Model, View i Controller

KORIŠĆENJE SPRING (I BILO KOG DRUGOG) RADNOG OKVIRA

- u projektu se programiraju komponente koje će se ugraditi u već predefinisanu aplikaciju (aplikacija kao da već postoji samo je treba doraditi)
- **kontrola nije više na našoj strani, radni okvir poziva našu komponentu**
- radni okvir implementira učestale mehanizme, nizove koraka, koji se uvek izvršavaju u aplikaciji.
- kada se dođe do konkretnog koraka koji treba da se izvrši, tada je neophodno za radni okvir isprogramirati i u njemu ugraditi odgovarajuću komponentu, koja će izvršiti odgovarjuću funkcionalnost

- Inverzija kontrole (IoC)
  Princip koji se koristi kod Spring radnog okvira, da programer kreira komponente koje Spring radni okvir poziva po potrebi zove se inverzija kontrole (**Inversion of Control**) ili može se čak i zvati **Hollywood Principle**

Rad sa Mavenom

- Maven omogućva *Dependency management* tj. automatsko upravljanje dependency-ima za korišćene biblioteke.
- Kod projekta sa Maven kada bi bilo neophodno koristiti neku biblioteku neophodno bi bilo samo navesti zavisnost projekta od te biblioteke u **pom.xml** fajlu, a Maven će sam naći tu biblioteku, pruzeti je, ubaciti u projekat, dodati je u *build path* projekta.
- Maven će ponoviti isti postupak za sve biblioteke od kojih preuzeta bibliteka zavisi, i ponoviti isti process za novopreuzete bibliteke, sve dok više ne bude zavisnosti.

- Proces ponavljati sve dok više nema zavisnih biblioteka
- U Maven projektu se poštuje princip *Convention over Configuration* koji se još naziva i *coding by convention*
- Omogućava dobijanje određenog željenog ponašanja rada aplikacije bez potrebe da se pišu konfiguracioni fajlovi
- *Artifact Id* prestavlja osnovnu gradivnu jedinicu u Maven alatu.
- *Group Id* označava grupu projekta, najčešće nešto vezano za organizaciju.

2.2,3.1,4.1 Spring Boot

- Spring Boot (https://spring.io/projects/spring-boot) projekat je nastao sa idejom da se olakša kreiranje *stand-alone* Spring aplikacije koja jednostavno može da se pokrene.
- Spring Boot karateristike
    - Jednostavnije se dobija konfigurisana Spring aplikacija
    - Jednostavnije pokretanje
        - Ugrađen veb server
    - Jednostavnije upravljanje paketima
        - Skup pripremljenih Maven artefakata
    - Konfiguriše Spring kontejner automatski gde god je moguće
- Ideja je da se programer fokusira inicijalno na razvoj aplikacije umesto na njen životni ciklus (konfiguraciju, postavljanje, upravljanje projektom, ...)
- Postoji klasa koja ima *main* metodu
    - Aplikacija se pokreće kao da je stand-alone aplikacija
- **Controller** su Java klase čije metode (**Handler Methods**) su zadužene za obradu različitih HTTP zahteva. Po potrebi metode kreiraju **model** i pozivaju metode servisnog sloja.
- **View engine** može biti npr. JSP, FreeMarker ili Thymeleaf template engine
- Postoje 3 obavezne zavisnosti *spring-boot-starter-web*, *spring-boot-starter-tomcat* i *spring-boot-starter-test.*
- To su artifakti koji su napravljeni u Sring Boot sa ciljem da se prevuku svi moduli Spring i propratne konfiguracije koje su neophodne za izradu **Spring Boot veb projekta**.

- Artifakt *spring-boot-starter-thymeleaf* je potreban za uključivanje *Thymeleaf* biblioteka. *Thymeleaf* je Java XML/XHTML/HTML5 templejt **View engine** koji se koristi za veb (servlet-based) i ne-veb oruženja

ANOTACIJA @SpringBootApplication

- Povlači za sobom još tri anotacije @SpringBootConfiguration, @EnableAutoConfiguration i @ComponentScan
    - @SpringBootConfiguration – označava da se u *TestMavenVebApplication* klasi nalazi konfiguracija za Spring projekat
    - @EnableAutoConfiguration omogućava automstsku konfiguraciju *Spring Application Context*, pokušavajući da pogodi i konfiguriše binove, komponente i biblioteke koje aplikacija koristi
        - oslanja se na *pom.xml*
    - *@ComponentScan* govori Spring aplikaciji da prođe kroz kompletan projekat i za svaku Bean klasu da pročita njenu konfiguraciju iz Java koda.
- JavaBean je običan standard za pisanje klasa.
    - Standard nalaže da klasa mora zadovoljavati sledeće osobine da bi bila Bean:
    - Svi atributi klase moraju biti privatni (private) – koristiti getere i setere
    - Klasa mora imati javni (public) konstruktor bez parametara
    - Klasa implementira interfejs Serializable
- RequestMapping je anotacija koja se navodi za klasu i njom se definiše URL mapiranje za datu klasu.
- U telu kontrolera definišu se javne metode koje trebaju da predstavlju odgovor kontrolera za poziv odgovarajuće HTTP metode, te metode se nazivaju još i **Handler Methods**.
- Za svaku **Handler Method** treba da definiše ostatak URL koji se pridodaje na osnovni URL klase. Metoda je dostupna kao **adresa kontrolera + proširenje adrese navedeno u anotaciji**
    - @ResponseBody naznačava se da će se u telu metode definisati telo HTTP odgovora (u našem slučaju povratni HTML), umesto da metoda vrati logičko ime prezentacije (view).

DEPENDENCY INJECTION (DI)

- Dependency Injection je softverski obrazac/princip koji se koristi kako bi se olakšalo instanciranje objekata.
- Osnovna ideja je da programer ne instancira smamostalno objekte u kodu već da za instanciranje objekata prepusti radnom okviru u kome programira.

    - U kodu programer samo koristi promenljive a u njima će se “magično” naći objekti
- Dependency Injection je primer Inversion of Control principa

@ModelAttribute anotacija se može koristiti za preuzimanje svih parametara forme pri čemu se oni automatski povezuju sa klasom iz sloja model koja je argument metode i atributima navedene klase

5.1 PRAĆENJE SESIJE

- Kada ističe cookie?
    - eksplicitan momenat (expires)
    - nedefinisano – kada se ugasi browser

HTTP je stateless protokol koji ne zateva od servera čuvanje statusa klijenta ili korisničke sesije klijenta tj. stanja klijenta proizašlog iz niza zahteva upućenih od strane istog klijenta

- HTTP serveri prevazilaze prethodno tako što implementiraju različite metode za održavanje i upravljanje sesijom, tipično se oslanjajući na jedinstveni identifikator *cookie* ili neki drugi parametar koji omogućava praćenje zahteva koji originiraju od istog klijenta (npr. URL Rewriting mehanizam), kreirajući stateful protokol iznad HTTP protokola.

HTTP protokol ne prati sesiju – komunikacija klijenta i servera se ne prati po isporuci resursa.

- Koristi se *cookie* mehanizam:
    - Server šalje *cookie* klijentu u okviru http response
    - klijent čuva primljeni *cookie* i šalje ga uz svaki http request.
- Ako navigator ne prihvata *cookie-je*, koristi se URL Rewriting mehanizam:
- Svaki brauzer ima svoje kukije i ne deli ih sa drugim brauzerima
- Ako navigator ne prihvata cookie-je, koristi se URL Rewriting mehanizam
- u hiperlink (<a href="...">) koji "gađa" naš server ugradimo id sesije:

Klasa HttpSession

- Reprezentuje sesiju
- Čuva cookie ili ID sesije za URL redirection
    - metoda getId()
- Čuva objekte vezane za sesiju

    - metode getAttribute(ime), setAttribute(ime, objekat), removeAttribute(ime)
- Invalidira sesiju i razvezuje sve objekte vezane za nju
    - metoda invalidate()
- podešava period neaktivnosti
    - metoda setMaxInactiveInterval(sekunde)
- Spring Session ima jednostavan cilj da se oslobodi ograničenja da se HTTP sesije čuvaju na serveru.
    - Takođe omogućava deljenje podataka o sesiji između različitih servisa koji se nalaze u oblaku bez vezanosti za jedan kontejner (tj. Tomcat)
    - Podržava višestruke sesije u istom brauzeru i slanje sesije u HTTP zaglavlju

Spring Session uključuje sledeće module:

- Spring Session Core - Spring Session core APIs - obavezan
- Spring Session Data Redis – pruža repozitorijum za skladištenje sesije i upravljanje sesijom koja se oslanja na Redis bazu podataka
- Spring Session JDBC - pruža repozitorijum za skladištenje sesije i upravljanje sesijom koje se oslanja relacionu bazu podataka kao što je MySQL
- Spring Session Hazelcast - pruža repozitorijum za skladištenje sesije i Hazelcast upravljanje sesijom

- Implementacijom interfejsa HttpSessionListener i redefinisanjem metode
    - sessionCreated moguće je definisati kod koji će se izvršiti nakon kreiranja sesije,
    - a redefinisanje metode sessionDestroyed moguće je definisati kod koji će se izvršiti nakon uništavanja sesije
- Neohodno je da klasa koja implementira Interfejs bude označena sa anotacijom @Component, a Spring će prepoznati osluškivač prilikom skeniranje svih bean klasa.
- Ukoliko se ne koristi anotacija @Component, osluškivač se može registrovati na nivou aplikacije - u /WEB-INF/web.xml

6.2 PRIJAVA,ODJAVA I KEŠIRANJE

- Request.getSesion() == Request.getSesion(true)
- Request.getSession(false) – pribavlja sesiju, a ako nema sesije neće kreirati novu sesiju
- sesija se poništava – session.invalidate()

6.3 INTERNACIONALIZACIJA SADRŽAJA

Internacionalizacijom sadržaja koji se dostavlja korisnicima, aplikacije se prilagođava potrebama korisnika koji dolaze iz različitih zemalja/regiona.

Na ovaj način aplikacija postaje dostupna široj populaciji korisnika.

- Ne znaju svi korisnici predefinisani jezik na kome je pravljena apliakcije

- Sav tekstulani sadržaj koji se koristi za potrebe internacionalizacije piše se u property fajlovima, a oni se skladište u okviru **src/main/resources** direktorijuma.
- Obično se kreira defaultni fajl pod nazivom **messages.properties** u kome se navodi predefinisani/osnovni set poruka koji se koristi ako aplikacija ne podržava poruku za odabrani jezik internacionalizacije.
- Za svaki jezik internacionalizacije koji treba da podržava aplikacija se navodi set poruka u okviru zasebnog property fajla, pri čemu taj fajl sadrži **sufiks** koji predstvlja kod zemlje (npr. **messages_sr.properties**, **messages_de.properties**, **messages_nl.properties,**...)
- **MessageSource** interfejs se u Spring radnom okviru koristi za pronalaženje poruka, pružajući podršku za njihovu parametrizaciju i interncipnalizaciju.

Metoda getMessage interfejsa **MessageSource** omogućuje pribavljanje sadržaja za navedeni jezik.

U okviru poziva metode se navode:

- ključ poruke,
- dodatni parametri poruke (ako postoje)
- lokalizacija za koju se traži poruka

- LocaleResolver interfejs se u Spring radnom okviru koristi za određivanje trenutne lokalizacije koja je u upotrebi za klijenta.

7.2 SERVISNI SLOJ

Servisni sloj je tu da pruža poslovnu logiku nad podacima koji se šalju ka i od DAO sloja ka kontrolerima.

Servisi su komponente koje implementiraju **poslovnu logiku** vezanu za neki entitet.

Servisni sloj se koristi za manipulaciju podacima. Te manipulacije najčešće podrazumevaju CRUD operacije.

Servisni sloj se koristi kao omotač (wrapper) DAO sloja i predstavlja opšteprihvaćen način za korišćenje DAO sloja. Radi sa **Modelom**.

- U servisni sloj se stavljaju sve CRUD (create, retrieve, update, delete) metode koje perzistencioni sloj nudi
- Može biti izostavljen tako što ćemo koristiti DAO sloj direktno, ali njegovo uvođenje može doneti mnoge pogodnosti:
- **Zašto ne koristiti direktno perzistencioni sloj**? Zato što perzistencioni sloj nudi samo osnovne operacije. Ako želimo da proverimo poslovnu logiku u tim podacima to nije moguće. Takve provere možemo vršiti u metodama u servisnom sloju.
- **Preporuka je da se u Service sloju piše poslovna logika sistema**

- Za servisni sloj se po konvenciji koristi paket **service**.
- Paket će sadržati interfejse sa metodama koje će predstavljati **željene CRUD operacije nad podacima**
- Koristeći DI mehanizam @Autowired anotaciju Spring može automatski da prepozna i poveže implementaciju servisa sa kodom koji poziva interfejs za taj servis.
- **Ukoliko postoji više od 1 implementacije za određeni servis**, potrebno je Spring **eksplictno** naznačiti koja implementacija servisa treba da se koristiti.
- Jedan od načina bi bio da se definiše primarna implementacija za servis.
- Impelemtacija se anotira kao primarna sa @Primary.
- Anotacija @Primary ukazuje da datoj bean klasi treba dati prednost ukoliko postoje više bean klasa koji su kandidati za automatsko povezivanje sa @Autowired anotacijom.
- Kako koristiti različite imeplemntacije u različitim delovima koda?
- Kada je potrebno imati veći stepen kontrole nad procesom selekcije kandidata za DI mehanizam, moguće je koristiti @Qualifier anotaciju.

8.THYMELEAF

- softverska komponenta koja ima za zadatak da automatski u proizvoljan tekstualni šablon, napisan određenom sintaksom, ubaci odgovarajuće podatke na posebno unapred obeležena mesta i na specificirani način
- *template engine* može biti deo *framework*-a ili može biti *3rd-party* biblioteka
- programer piše statičke tekstualne šablone u pogodnom *editor*-u za odabranu sintaksu, a zatim šablone dopunjuje specifikacijom na osnovu koje će *template engine* **za vreme izvršavanja** ubaciti odgovarajuće podatke
- način pisanja specifikacije zavisi od samog *template engine*-a
- *template engine*-u je pored tekstualnog šablona sa specifikacijom potreban i izvor podataka da bi proizveo konačan rezultat
- ako je šablon statički HTML dokument, tada iz njega nastaje dinamički HTML sadržaj
- moderan *open source template engine* baziran na *Java* programskom jeziku
- može se koristiti u sklopu web aplikacija ili samostalno za druge namene

U kontekstu HTML dokumenata, šablon je moguće popuniti na sledeće načine:

- popunjavanjem tekstualnog sadržaja elemenata
- popunjavanjem vrednosti atributa
- uslovnim prikazom elemenata i atributa
- ponavljanjem elemenata ili grupe elemenata

- izračunavanje izraza rezultuje nekom vrednošću određenog tipa (tekstualnog, numeričkog, *boolean*, *null* i sl., ili može biti i referenca na objekat)
  postoji više vrsta izraza od kojih ćemo se ograničiti na:
  - *${...}* : *variable expressions* (**dodela vrednosti** na osnovu izraza koji sadrži promenljive)
  - **{...}* : selection *expressions* (**dodela vrednosti** na osnovu izraza koji pristupa delovima prethodno odabranog objekta)
  - *#{...}* : message (i18) *expressions* (**pribavljanje locale secifičnih poruka** iz spoljnjeg izvora)
  - *@{...}* : *link (URL) expressions* (**formiraju URL-ove** u odnosu na bazični URL)
  - *~{...}* : fragment expressions (**pribavljanje fragemnata drugih HTML stranica**)
- *Variable expressions* se izvršavaju nad ***context promenljivama*** *koje se u Spring-u još nazivaju i* ***model attributes*** *promenljive (u Handler metodama se u Model dodaju atibuti po principu ključ i vrednost. U parametar metode* ModelMap map *ili povratavrednost metode* ModelAndView*, a zatim se model prosleđuje pogledu)*
- Koriste se za **dodelu vrednosti** na definisanoj poziciji u HTML na osnovu izraza koji sadrži promenljive
- U okviru Thymeleaf izraza moguće je korićenje literala:
  - *Text literals: 'one text', 'Another one!',…*
  - *Number literals: 0, 34, 3.0, 12.3,…*
  - *Boolean literals: true, false*
  - *Null literal: null*

  Literal tokens: jedan, nekiTekst, moj.Tekst, moj.Tekst2, …

- Sa ciljem da se postigne veća fleksibilnost izraza OGNL jezika omogućeno je korićenje bazičnih objekata (***Basic Objects***) i pomoćnih objekata (**Utility Objects**)
- Objekti se pozivaju u oblik: *#naziv_objekta*
- Bazični objekti (***Basic Objects***) su:
  - #ctx: the context object org.thymeleaf.context.WebContext implements IWebContext. Sadrži ostale navedene ispod *${#ctx.request}*
  - #vars: the context variables.
  - #locale: the context locale. Direkna veza sa **java.util.Locale** koji je asociran sa trenutnim zahtevom *#locale.language*
  - #request: (only in Web Contexts) the javax.servlet.http.HttpServletRequest object. Pribavljanje parametra ime *${#request.getParameter('ime')}*, atributa mojAtribut *${#request.getAttribute('mojAtribut')} i atributa zaglavlja User-Agent ${#request.getHeader('User-Agent')}*
  - #response: (only in Web Contexts) the javax.servlet.http.HttpServletResponse object. Pribavljanje njegovih vrednosti iz templejta nema baš puno smisla jer još nije objekat

formiran osim ako u kontroleru prethodno ne definišemo neku vrednost *${#response.getHeader('zaglavlje1').*

- #servletContext: (only in Web Contexts) the ServletContext object. Pribavljanje objekta *statistikaFilmova ${#servletContext.getAttribute('statistikaFilmova').filmovi*.
- param: mapping for retrieving request parameters (query ili data parametri). Pribavljanje parametra ime *${param.ime[0]}*. Metode *${param.size()}*, *${param.isEmpty()}*, *${param.containsKey('ime')}*
- session: (only in Web Contexts) mapping for retrieving HttpSession object. Pribavljanje objekta prijavljeniKorisnik *${session.prijavljeniKorisnik}*.
- application: retrieving servlet context attributes *${application.statistikaFilmova}*

- Pomoćni objekti (***Utility Objects***) su:
  - #execInfo: information about the template being processed.
  - **#messages**: methods for obtaining externalized messages inside variables expressions, in the same way as they would be obtained using #{…} syntax.
  - #uris: methods for escaping parts of URLs/URIs
  - **#conversions**: methods for executing the configured conversion service (if any).
  - **#dates**: methods for java.util.Date objects: formatting, component extraction, etc.
  - **#temporals:** methods for java.time.LocalDate, java.time.LocalDateTime objects.
  - **#calendars**: analogous to #dates, but for java.util.Calendar objects.
  - **#numbers**: methods for formatting numeric objects.
  - **#strings**: methods for String objects: contains, startsWith, prepending/appending, etc.
  - **#objects**: methods for objects in general.
  - **#bools**: methods for boolean evaluation.
  - **#arrays**: methods for arrays.
  - **#lists**: methods for lists.
  - **#sets**: methods for sets.
  - **#maps**: methods for maps.
  - **#aggregates**: methods for creating aggregates on arrays or collections.

  #ids: methods for dealing with id attributes that might be repeated (for example, as a result of an iteration).

  IZRAZI – SELECTION EXPRESSIONS

- slični variable expressions
- koriste se za izračunavaje vrednosti za prethodno odabrani objekat
- oblik: **{property}*
- Obekat se mora prethodno odabrati sa **th:object** atributom

  IZRAZI – FRAGMENT EXPRESSIONS
- koriste se za pribavljanje fragmenata drugih HTML stranica
- cilj je ponovno iskorišćenje često ponavljanih delova HTML stanice (npr. sekcije *header* i fotter)
- **U posmatranu html stranicu ubacujem delove drugih stranica**
- Fragmenti ste mogu kreirati u posebim stranicama ili u jednoj stanici
  oblik: <ElementZaKojiSeFragmentDefinise th:fragment=*"nazivFragmenta"*>

- Postoje 3 osnovna načina za uključivanje sadržaj fragemnata
    - insert – u elementu se insertuje element koji je definisan kao fragment
    - replace – element se menja sa elementom koji je definisan kao fragment
    - include –uključivanje sadržaja - depricated legacy code
- Dozvoljeno je definisanje lokalnih prmenlijvih koje će biti vidljive samo u elementu u kojem su definisane
- Navedi promeljiva i njena vrednost u atributu th:with
- oblik:
- <element th:with="*nazivLokProm=${promenljiva}*">
- Uslovni izraz je namenjen za izvršavanje jednog od dva ponuđena izraza u zavisnosti od rezultata evaluacije uslova
- Sva tri dela izraza (**condition**, **then** i **else**) su izrazi za sebe

ELVIS OPERATOR – PREDEFINISANI USLOVNI IZRAZ

Predstavlja specijalizaciju uslovnog izraza u kome se ne navodi **then** deo.
Vrednost then dela je predefinisana i predstavlja zapravo vrednost varijable navedene u **condition** delu tj. u uslovu izraza.

- ako se u nadelementu navede atribut th:switch i u elementu se navede atribut th:case, tada element zajedno sa svojim podelementima se prikazuje ako vrednost elementa odgovara vrednosti promenljive definisane u nadelementu
- Odgovara sintaksi **switch** strukture u Javi
- oblik:

```
<element th:switch="${promenljiva}">
        <element th:case="vrednost1"> </element>
        <element th:case="${promenljiva2}"> </element>
</element>
```

- ako se u elementu navede atribut th:each, element zajedno sa svojim podelementima se ponavlja za svaki element kolekcije koja je određena izrazom
- oblik: <element th:each="*element, status: ${kolekcija}*">…</element>
- *element* promenljiva poprima vrednost jednog po jednog elementa kolekcije
  status je pomoćna promenljiva koja sadrži dodatne informacije o iteraciji kroz elemente kolekcije

8.2 DAL

1. *controller*:
    a) traži od servisnog sloja podatke
    b) šalje pristigle parametre servisnom sloju radi upisa
    c) vrši kontrolu toka (redirekcija i sl.)
2. servisni sloj:
    a) šalje zahtev za čitanje DAL sloju
    b) po potrebi vrši pripremu podataka i šalje ih DAL sloju radi upisa

3. DAL sloj:
    a) vrši čitanje podataka iz baze i vraća ih servisnom sloju
    b) vrši upis podataka u bazu i vraća rezultat upisa
4. servisni sloj:
    a) po potrebu vrši obradu podataka i vraća ih *controller*-u
    b) prosleđuje rezultat upisa podataka *controller*-u
5. *controller*:
    a) prosleđuje podatke *view engine*-u radi prikaza

    vrši kontrolu toka (redirekcija i sl.)

- umesto interfejsa *sql.Connection*, za komunikaciju sa bazom se koristi klasa *JdbcTemplate*
- ovu klasu *framework inject*-uje za vreme izvršavanja i pri tom vrši povezivanje sa bazom na osnovu unosa iz *application.properties* datoteke
- *RowMapper* je interfejs čija implementacija služi za mapiranje svakog pojedinačnog reda *ResultSet*-a na objekat klase modela koji je potrebno kreirati
- metoda *mapRow* se poziva za svaki red *ResultSet*-a, a njena implementacija čita kolonu po kolonu za dati red i na osnovu pročitanih vrednosti **kreira jedan objekat** i **vraća jedan objekat klase modela**
- *RowCallbackHandler* je interfejs čija implementacija služi za mapiranje jednog reda *ResultSet*-a na objekat klase modela koji je potrebno kreirati, ali tako da je programmer u kontroli kreiranja rezultata
- metoda *processRow* se poziva za svaki red *ResultSet*-a, a njena implementacija čita kolonu po kolonu za dati red i na osnovu pročitanih vrednosti **kreira/pruzima jedan objekat** i **ne vraća nijedan objekat**
- Drugi način za dobijanje podataka o povezanim entitetima bio bi korišćenje *RowMapper*.
- @Transactional anotacija služi da grupiše sve upite iz metode u transakciju
- *PreparedStatementCreator* je interfejs čija implementacija služi da se dodatno upravlja kreiranjem *PreparedStatement*-a
- *GeneratedKeyHolder* je klasa čiji objekti čuvaju ključeve, generisane od strane baze, koji su nastali pri prethodnom INSERT upitu

APLIKACIJA ARHITEKTURE TROSLOJNA

- Prednosti troslojne :
    - Smanjenje troškova vezano za hardver klijentskih mašina
    - Modularnost, “lako” izmenjivi delovi
    - Sa izdvajanjem poslovne logike koja se **odražava na brojne krajnje korisnike** u zaseban sloj u vidu **aplikativnog servera, ažuriranje i održavanje aplikacije je centralizovano**. Ovim se eliminiše problem distribucije softvera, koji je bio prisutan u dvoslojnom klijent-server modelu
    - Sa dobijenom modularnošću moguće je lako izmeniti ili zameniti neki od slojeva bez uticaja na ostale

- JDBC API - namenjen aplikativnim programerima, koji definiše komunikaciju na relaciji **Java aplikacija** – **JDBC menadžer**
- JDBC drajver API, interfejs nižeg nivoa namenjen programerima drajvera, koji definiše način komunikacije na relaciji **JDBC menadžer** – **drajver konkretnog DBMS**

Korišćenje baze u Javi - Inicijalizacija konekcije

- forName – poziv statičke metode klase Class. Statička metoda vraća incijalizovan objekat klase koji odgovara parametru.
- Klasa java.sql.DriverManager koja upravlja učitavanjem drajvera i obezbeđuje podršku za otvaranje konekcija ka bazi podataka. Njegova osnovna funkcija je održavanje liste dostupnih drajvera i učitavanje odgovarajućeg drajvera na osnovuinformacija iz URL adrese
- Struktura JDBC URL adrese je sledeća: jdbc:<podprotokol>:<ime baze>, gde podprotokol predstavlja konkretan mehanizam povezivanja na određenu bazu podataka, a koji može biti podržan od strane više drajvera. Komponente i sintaksa imena baze zavise od vrste podprotokola u upotrebi.

10.1 SCRIPT JEZICI,JavaScript,BOM,DOM

- da bi izgled HTML stranice mogao da se menja lokalno kod korisnika u *web browser*-u bez direktnog učešća servera, neophodno je da:
    1. *web browser* može da izvršava naredbe na nekom programskom jeziku koje će da upravljaju tom izmenom
    2. server sa prvim slanjem HTML stranice pošalje i pridruženi program na tom jeziku
- prethodno važi i logiku validacije unosa i eventualne druge namene (interaktivnost)
- da bi takva *web aplikacija* mogla da se izvršava na svim *web browser*-ima i na svim platformama, neophodno je da:
    1. programske naredbe, koje *web browser*-i izvršavaju, budu napisane na istom standardnom jeziku
    2. se naredbe ne prevode u izvršni oblik, već da se njihova sintaksa evaluira i izvršava *ad hoc* (da budu interpretirane)

JavaScript

- da bi izgled HTML stranice mogao da se menja lokalno kod korisnika u *web browser*-u bez direktnog učešća servera, neophodno je da:
    1. *web browser* može da izvršava naredbe na nekom programskom jeziku koje će da upravljaju tom izmenom
    2. server sa prvim slanjem HTML stranice pošalje i pridruženi program na tom jeziku

- prethodno važi i logiku validacije unosa i eventualne druge namene (interaktivnost)
- da bi takva *web aplikacija* mogla da se izvršava na svim *web browser*-ima i na svim platformama, neophodno je da:
    1. programske naredbe, koje *web browser*-i izvršavaju, budu napisane na istom standardnom jeziku
    2. se naredbe ne prevode u izvršni oblik, već da se njihova sintaksa evaluira i izvršava *ad hoc* (da budu interpretirane)
- funkcije, definisane u *script tag*-u u *head tag*-u HTML dokumenta ili u eksternim datotekama, se ne moraju pozivati samo u *script tag*-ovima u *body tagu* HTML dokumenta, već se mogu pozivati i pri korisnički izazvanim akcijama i u drugim posebnim vremenskim trenucima – događajima (npr. klik na dugme, pritisak tastera, završetak učitavanja dokumenta i sl.)
- tada se ove funkcije zovu rukovaoci događajima (*event handlers*)

Događaj	Dešava se kada...	Najčešća upotreba
onload	... se stranica učita, ako se definiše za *body* element ... se element učita, ako se definiše za bilo koji drugi element	kada se stranica ili neki element dugo učitava, a neka automatska procedura zahteva da element bude učitan
onclick	... korisnik klikne na element	validacija unosa forme
onsubmit	... korisnik *submit*-uje formu	
onmouseover	... korisnik pređe mišem preko elementa	promena izgleda elementa, ispis pojašnjenja, prikaz dodatnih opcija
onfocus	... korisnik uđe u polje za unos	
onblur	... korisnik napusti polje za unos	validacija unosa za celo polje
onkeydown	... korisnik pritisne taster	validacija unosa za svaki karakter
onkeypress	... korisnik pritisne pa otpusti taster, ili drži taster	
onkeyup	... korisnik otpusti taster	
i mnogi drugi...		

JavaScript raspolaže ugrađenim klasama:
String
Number
Array
Map
Date
Math
i mnogim drugim...

definicija *JavaScript* objekta se navodi između znakova { i }
navode se parovi nazivAtributa: vrednostAtributa
vrednost atributa može biti primitivnog tipa ili referenca na objekat
ako atributa ima više, parovi se odvajaju znakom ,

definicija *JavaScript* objekta se navodi između znakova { i }
navode se parovi nazivAtributa: vrednostAtributa
vrednost atributa može biti primitivnog tipa ili referenca na objekat

ako atributa ima više, parovi se odvajaju znakom ,

atributi mogu biti i reference na objekte i čak i kolekcije referenci

BOM (Browser Object Model)

- objektni model prozora *web browser*-a koji omogućuje programsko rukovanje njegovim elementima

Window

	Naziv	Upotreba
metode	*alert(...), confirm(...), prompt(...)*	prikaz poruka i pitanja korisniku
	back() *forward()*	povratak na prethodnu stranicu odlazak na sledeću stranicu
	moveBy(...) *moveTo(...)*	pomeranje prozora
	open(...)	otvara novi prozor sa zadatom adresom
	setTimeout("izraz", timeout) *setInterval("izraz", interval)*	zadavanje *JavaScript* izraza koji će se izvršiti nakon određenog vremena zadavanje *JavaScript* izraza koji će se izvršavati periodično
	i mnoge druge...	
atributi	*screen, document, location, history, navigator*	pristup podelementima prozora
	screenX, screenY	pozicija prozora na ekranu
	outerWidth, outerHeight	dimenzije prozora
	i mnogi drugi...	

Location

	Naziv	Upotreba
metode	*reload()* *replace(...)*	ponovo učitava tekuću adresu učitava novu adresu
atributi	*href*	kompletan URL
	protocol	protokol
	host	adresa servera
	port	port
	pathname	putanja do resursa
	search	parametri

History

	Naziv	Upotreba
metode	*back()* *forward()* *go(...)*	povratak na prethodnu stranicu odlazak na sledeću stranicu odlazak na zadatu stranicu iz liste
atributi	*current*	tekuća stranica
	length	ukupan broj zapamćenih posećenih stranica
	next	sledeća stranica
	previous	prethodna stranica

Navigator

opisuje *web browser* koji korisnik koristi da bi pristupio stranici

	Naziv	Upotreba
atributi	*appName*	naziv *web browser*-a
	appVersion	verzija *web browser*-a
	cookieEnabled	da li browser podržava *cookies*?
	language	jezik *web browser*-a
	platform	operativni system na kome je pokrenut *web browser*
	i mnogi drugi...	

DOM (Document Object Model)

- dalje opisuje objektni model HTML stranice uz pomoć kog mogu da se čitaju i menjaju njeni elementi
- objektni model je organizovan u stablo
- ranije se DOM standard organizovao u generacije (Level 0, Level 1, ...), ali to više nije slučaj

DOM Level 0

- opisuje predefinisane nizove objekata koji se mogu ili ne moraju javiti na HTML stranici
  nizovi uvek postoje, makar bili i prazni

DOM Level 1-4

- opisuje objektni model proizvoljne HTML stranice u odnosu na njen korenski element
- svaki element (čvor) stabla je opisan objektom tipa *node*
- čvor na vrhu stabla se zove korenski čvor (*root node*)
- svaki čvor ima referencu na jedan roditeljski čvor (*parent node*); referenca je prazna za korenski čvor
- svaki čvor ima niz referenci na čvorove potomke (*child nodes*); niz može biti prazan i tada se čvor naziva list
- čvorovi potomci istog roditeljskog čvora se nazivaju *siblings*

Node – opisuje 1 čvor u stablu

	Naziv	Upotreba
atributi	*nodeName*	naziv elementa ili atributa
	nodeType	tip čvora (1 za HTML tagove, 2 za atribute, 3 za tekstualne čvorove, 8 za komentar, 9 za dokument)
	nodeValue	sadržaj tekstualnog čvora ili vrednost atributa
	innerHTML	kompletan HTML kod podstabla čvora
	id	vrednost *id* atributa (ako se navede)
	className	vrednost *class* atributa (ako se navede)
	style	referenca na style objekat
	childNodes	lista čvorova potomaka
	firstChild *lastChild*	prvi čvor potomak poslednji čvor potomak
	parentNode	roditeljski čvor
	previousSibling *nextSibling*	prethodni čvor na istom nivou stabla sledeći čvor na istom nivou stabla
	i mnogi drugi...	

	Naziv	Upotreba
metode	*appendChild(node)*	dodaje novi čvor na kraj liste čvorova potomaka pozivajućeg čvora
	insertBefore(referenceNode, insertedNode)	umeće čvor u listu čvorova potomaka pre drugog čvora u podstablu pozivajućeg čvora
	removeChild(node)	uklanja čvor iz liste čvorova potomaka pozivajućeg čvora
	getAttribute(atrributeName)	direktno čita vrednost atributa čvora (iako je atribut čvor potomak pozivajućeg čvora)
	setAttribute(atrributeName, attributeValue)	direktno postavlja novu vrednost atributa čvora (iako je atribut čvor potomak pozivajućeg čvora)
	removeAttribute(atrributeName)	direktno uklanja atribut čvora (iako je atribut čvor potomak pozivajućeg čvora)
	hasAttributes()	vraća *true* ako pozivajući čvor u svom podstablu ima atribute
	i mnoge druge...	

Form – opisuje formu

	Naziv	Upotreba
metode	*reset()*	*reset*-uje sva polja forme
	submit()	programski klik na *submit* dugme
atributi	*method*	vrednost *method* atributa
	action	vrednost *action* atributa
	name	vrednost *name* atributa
	length	ukupan broj *input* elemenata u formi
	elements	niz *input* elemenata u formi
	i mnogi drugi...	

Input – opisuje input element forme

	Naziv	Upotreba
atributi	*value*	vrednost unosa
	defaultValue	početna vrednost
	type	vrednost type atributa
	name	vrednost name atributa
	form	referenca na formu kojoj *input* pripada
	i mnogi drugi…	

Metoda *document.write(...)*

- ako se pozove za vreme učitavanja stranice dopisuje sadržaj na kraj HTML dokumenta
- ako se pozove nakon učitavanja stranice, zamenjuje kompletan sadržaj HTML dokumenta

Style

- objekat uz pomoć kog se upravlja CSS atributima čvora

	Naziv	CSS atribut
atributi	*display*	*display*
	color	*color*
	backgroundColor	*background-color*
	width	*width*
	borderStyle	*border-style*
	borderColor	*border-color*
	borderWidth	*border-width*
	i mnogi drugi…	

10.2 i 11 JavaScript i Jquery biblioteke

JavaScript biblioteka obuhvata skup konstanti, klasa i funkcija koje su definisane u eksternoj *.js* datoteci
JavaScript biblioteka obuhvata jednu ili više eksternih *.js* datoteka koje se uključuju na HTML stranicu pre *JavaScript* skripte koja će je koristiti
JavaScript biblioteka kojom se pojednostavljuje programiranje na klijentskoj stani tj. pojedostavljuje se:

  - pristup elementima *web* stranice
  - izmena izgleda *web* stranice
  - izmena sadržaja *web* stranice
  - interakcija sa korisnikom
  - animacije
  - uobičajene *JavaScript* naredbe
  - Dobavljanje sadržaja sa servera bez ponovnog osvežavanja stranice (AJAX zahtevi i odgovori)

- DOM objekat ili skup objekata se zatvaraju (*wrap*-uju) u *jQuery* objekat i njima se rukuje uz pomoć metoda *jQuery* objekta

- ove metode tipično nude lakše rukovanje DOM objektima i dodatno nude složenije operacije koje DOM objekti ne implementiraju po specifikaciji
- jQuery objektima se rukuje gotovo univerzalno putem metoda
- DOM objekat do čije se reference došlo klasičnim putem preko *JavaScript*-a može se zatvoriti (*wrap*-ovati) u *jQuery* objekat i sačuvati u posebnu referencu
- od tog momenta nad tom referencom se mogu pozvati *jQuery* metode
- ako je potreban poziv samo jedne metode, prethodno se može zapisati u jednom izrazu
- **$** : koristi jQuery - aktivira jQuery
- Da li je iz JavaScript-a
- var contents = document.getElementById(**'contents'**);
- isto sa JQuery
- var contents = $(**'#contents'**);
- Nije
- Prva će vratiti HTML DOM objekat
- Druga će vratiti jQuery objekat koji se vrapuje oko HTML DOM objekta i koji pruža jQuery metode.
- Nad pozivom $() mogu se pozvati JQuery metode kao css() ili animate()

JavaScript događaji

Događaj	Dešava se kada...
onabort	se prekine učitavanje slike
onblur	element izgubi fokus
onchange	korisnik pormeni sadržaj polja
onclick	se klikne mišem na objekat
ondblclick	se dva puta klikne po objektu
onerror	se dogodi greška prilikom učitavanja dokumenta ili slike
onfocus	element dobije fokus
onkeydown	se pritisne taster
onkeypress	se pritisne, pa otpusti taster, ili se drži pritisnut
onkeyup	se otpusti taster

Događaj	Dešava se kada...
onload	se stranica ili slika učita
onmousedown	se pritisne dugme miša
onmousemove	se miš pomera
onmouseout	miš izađe izvan zone elementa
onmouseover	miš pređe preko elementa
onmouseup	se otpusti dugme miša
onreset	se klikne na reset dugme
onresize	se prozoru ili frejmu promeni veličina
onselect	je tekst selektovan
onsubmit	se klikne na dugme subit u formi
onunload	korisnik napusti stranicu
i mnogi drugi...	

Jquery događaji

Događaj	Dešava se kada...
`$(…).ready(…)`	... se dokument učita; registruje se na *document* objekat
`$(…).click(…)`	... korisnik klikne na element
`$(…).submit(…)`	... korisnik *submit*-uje formu
`$(…).mouseover(…)`	... korisnik pređe mišem preko elementa
`$(…).focus(…)`	... korisnik uđe u polje za unos
`$(…).blur(…)`	... korisnik napusti polje za unos
`$(…).keydown(…)`	... korisnik pritisne taster
`$(…).keypress(…)`	... korisnik pritisne pa otpusti taster, ili drži taster
`$(…).keyup(…)`	... korisnik otpusti taster
i mnogi drugi...	

metode pozivaju nad objektima i vraćaju objekte koji predstavljaju elemente

jQuery preskače tesktualne čvorove i čvorove koji predstavljaju atribute i njima rukuje implicitno

Pronalaženje elementa na osnovu njegovog položaja u DOM stablu, terminologija:

Ancestor – čvor prethodnih

Descendant – čvor sledbenik

Parent – čvor direktni prethodnik

Child – čvor direktni sledbenik

Sibling – čvor na istom nivou

12.jQuery,Ajax i JSON

1. *web browser*:
    a) traži od servera statičku HTML stranicu
2. server:
    a) vraća statički HTML stranicu sa pridruženim *JavaScript* programom
3. *web browser*:
    a) *JavaScript* program pravi jedan ili više uzastopnih AJAX zahteva ka *controller*-ima servera (inicijalno i na korisničke događaje)
4. server:
    a) vraća podatke *JavaScript* programu u vidu JSON objekata
5. *web browser*:
    a) *JavaScript program* ugrađuje podatke u HTML stranicu i menja je
    b) *JavaScript program* traži neku drugu statičku HTML stranicu (vrši *client-side* redirekciju)
6. *web browser* (client):
7. izvršava programsku logiku vezanu za prikaz
8. izvršava programsku logiku vezanu za kontrolu toka
9. zvršava *client-side* validaciju podataka gde je to moguće
10. *server:*
11. obavlja ulogu repozitorijuma podataka: čuva integritet podataka i omogućuje da više klijenata konzistentno rukuju istim podacima
12. i dalje može da čuva stanje korisničke sesije ili *context*-a aplikacije
13. i dalje vrši *server-side* validaciju podataka
14. i dalje obavlja poslovnu logiku (servisi)
15. i dalje vrši perzistenciju podatka (DAL, baza)
16. *Asynchronous JavaScript and* XML
17. prevod: *JavaScript* program pravi asinhrone pozive ka serveru i sa njim razmenjuje XML podatke
18. tehnika za kreiranje brzih interaktivnih dinamičkih web stranica

19. u prvim inkarnacijama ovog koncepta, zaista su se razmenjivali XML podaci, dok ih je vremenom zamenio JSON format zbog jednostavnosti i činjenice da ga *JavaScript* ima ugrađenu podršku za njega
20. asinhrona priroda zahteva podrazumeva da *web browser* nakon korisničkog događaja ne blokira interfejs i ne čeka odgovor servera da bi vratio korisniku kontrolu nad *web browser-om*, već u paralelnom toku (programskoj niti) inicira zahtev i čeka odgovor, a odgovor obrađuje tek kada on stigne;
21. to može da potraje i nekoliko sekundi, a korisnik za to vreme može da nastavi da korisiti interfejs (**ne blokira se stranica, korisnik nije izgubio osećaj kontrole, nema louder sličice koja izluđuje korisnike**)
22. *Asynchronous JavaScript and* XML
23. Kod klasičnih web stranica, potrebno je osvežiti čitavu web stranicu kako bi se promenio sadržaj.
24. Umesto da se osvežava čitava stranica, učitavaju se podaci sa servera i osvežavaju delovi stranice
25. web stranica se osvežava asinhrono, razmenom male količine podataka sa serverom. Ovo omogućava da se osvežavaju delovi stranice.
26. Nije nova tehnologija već je kombinacija postojećih tehnologija:
27. **XMLHttpRequest object** – asinhrona razmena podataka sa serverom
28. **JavaScript/DOM** – izmena strukture i sadržaja bez ponovnog učitavanja stranice
29. **CSS** – uređivanje izgleda stranice
30. **XML (češto JSON)** – format podataka koji se razmenjuju
31. XMLHttpRequest
32. Koristi se za slanje HTTP i HTTPS zahteva iz skritpe ka serveru i za slanje odgovora sa servera u skriptu.
33. Radi nezavisno od stranice
34. pozovemo metodu **send** i ona uputi HTTP zahtev nezavisno od glavne stranice
35. Koristi se za
36. Ažuriranje web stranice bez potrebe da se ona ponovno učita
37. Zatraživanje podataka sa servera – nakon što je stanica učitana
38. Prihvatanje podataka sa servera – nakon što je stanica učitana
39. Slanje podataka serveru – u pozadini
40. Povratna vrednost servera može biti: **XML, JSON, HTML, ili običan tekst**

Method	Description
new XMLHttpRequest()	Creates a new XMLHttpRequest object
open(*method, url, async*)	Specifies the type of request *method*: the type of request: GET or POST *url*: the file location *async*: true (asynchronous) or false (synchronous)
send()	Sends a request to the server (used for GET)
send(*string*)	Sends a request string to the server (used for POST)
onreadystatechange	A function to be called when the readyState property changes
readyState	The status of the XMLHttpRequest 0: request not initialized 1: server connection established 2: request received 3: processing request 4: request finished and response is ready
status	200: OK 404: Page not found
responseText	The response data as a string
responseXML	The response data as XML data

JSON

JavaScript Object Notation

Jednostavan format za razmenu podataka predstavljenih pomoću teksta

Danas primarni format podataka korišćen za asinhronu komunikaciju između klijenta i servera

JSON i JavaScript

Kada se razmenjuju sa Web serverom, podaci se razmenjuju kao string.

JSON.parse(tekst) parsira podatke (string) u JavaScript objekat.

JSON.stringify(objekat) konvertuje JavaScript objekat u string.

JSON.stringify(objekat):

Konvertuje sve datume u stringove;

Uklanja sve funkcije iz JavaScript objekta.