LISTE
<ol> = uređena lista, **<ul>** = neuređena lista
<dl> = opisna lista, Član liste: **<li>** naziv **<li>**
Stilizacija člana liste (primjer):
član <li> bilokoje liste:
`li { margin-top: 0px; }`
član <li> neuređene liste:
`ul>li { margin-top: 0px; }`
član <li> koji ima bilokoji id:
`li[id] { margin-top: 0px;}`
član neuređene liste <li> koji ima klasu "klasa1":
`ul>li.klasa1{margin-top:0px;}`

CSS SELEKTORI
za elemente poput h1, h2, p, a, etc.:
`h1{background-color: green;}`
za klase u HTML-u:
ako je u HTML-u `<h1 class="klasa1">`
u CSS-u je `.klasa1{border: 1px;}`
za identifikatore:
ako je u HTML-u `<h2 id="crveno">`
u CSS-u je `#crveno {border: 1px;}`
univerzalni selector (*) – primjer:
`* {color: red;}` (→ sve je crveno)
za atribute:
`[attribute]` → sadrži atribut (npr. href, id)
`[attribute="x"]` → sadrži atribut čija je vrijednost jednaka x I samo x
`[attribute~="x"]` → sadrži atribut koji u svojoj listi sadrži x između ostalih vrijednosti (odvojenih razmakom)
`[attribute|="x"]` → sadrži atribut koji u svojoj listi sadrži x između ostalih vrijednosti (odvojenih crticom – npr. lang="en-gb")
`[attribute^="x"]` → sadrži atribut koji sadrži element koji počinje sa x
`[attribute$="x"]` → završava sa x
`[attribute*="x"]` → sadrži x u sebi

CSS SELEKTORI
pseudo-klase:
`:link | :visited | :active | :hover | :focus | :target | :lang(xy) | :first-child | :last-child | :first-of-type | :last-of-type | :nth-child(i) | :nth-last-child(i) | :nth-of-type(i) | :nth-last-of-type(i) | :only-child | :only-of-type | :empty | :root | :enabled | :disabled | :checked | :not(:first-child)`
pseudo-elementi:
`::first-line | ::first-letter | ::before | ::after`
kombinatori
`selector selector` → element + descendant
`selector > selector` → element + dijete
`selector + selector` → element + element koji slijedi neposredno nakon njega
`selector ~ selector` → element + general sibling

HTML FORMS
Šablona:
```
<form action="X" method="Y" id="Z">
    <label for="username"> Username:
<\label>
    <input type="A" name="B"
placeholder="C"/>
    <input type="submit" value="Submit">
<\form>
```
X → naziv datoteke (npr index.php) u koju se šalju podaci iz form-a nakon što se klikne submit
Y → get ILI post → get – default (šalje podatke iz form-a na URL naveden u polju action)
post – šalje podatke iz form-a kao HTTP post transakciju
Z → nije nužno imati id, služi za stilizaciju, CSS
A → button, checkbox, text (default), radio, hidden, email, password, search, datetime-local, date, etc.
B → ime elementa
C → tekst koji se pojavljuje dok korisnik još ništa nije upisao, odnosno input polje nema teksta

HTML podsjetnik: (komentar : **<!–komentar –>**
Tags
<a> (poveznica) |
 (novi redak) | <button> | , | <fieldset> (kao form, koristi se unutar form-a, ima I atribut form) | <img> (src, alt) | <legend> | <h1> , <h2> , <h3> | <script>

EXPRESS (ROUTING)
Middleware
- funkcije koje imaju pristup: → **request** objektu
- odgovara HTTP zahtjevu
- ima svojstva koja odgovaraju query stringu, parametrima, trijelu, zaglavljima, itd.
→ **response** objektu – odgovara HTTP odgovoru
- pomoću njega odgovaramo na zahtjev
→ **next()** funkciji – prosljeđuje obavljanje sljedećoj middleware funkciji u lancu
Primjer korištenja:
```
  const express = require('express');
  const app = express(); // ILI
  const router = express.Router();
```
Middleware - predlošci (ejs)
```
app.set('views', path.join(__dirname,
'views'));
app.set('view engine', 'ejs');
```
Middleware - statički resursi
```
app.use(express.static(path.join(__dirname,
'public')));
```
Middleware - dekodiranje parametara
```
app.use(express.urlencoded({ extended: true
}));
```
Pokretanje poslužitelja na portu 3000
```
app.listen(3000);
```
5 vrsta Middleware-a:
Application-level Middleware | Router-level Middleware
Error-handling Middleware | Third-party Middleware
Built-in Middleware (express.static/json/urlencoded)

EXPRESS (ROUTING)
- datoteke tipa NameOfView.routes.js
počinju sa:
```
const express = require('express');
const router = express.Router();
```
(možemo I unesti neke MVC Modele, kao npr:
`const User = require('../models/UserModel') )`
nakon toga moramo implementirati dvije funkcije:
1) Funkcija **router.get**
nju obično koristimo za općeniti prikaz stranice, ima elemente req, res, next, te u nju uglavnom ubacujemo funkciju res.render u kojoj povezujemo elemente iz .ejs-a.
Primjer:
datoteka signup.routes.js ima funkciju:
```
router.get('/', function(req,res,next)
{  res.render('signup', {
          title: 'Register a new user',
          linkActive: 'signup',
          user: req.session.user,
          err: undefined  });       });
```
datoteka signup.ejs ima liniju:
```
<% if (err !== undefined ) { %>
  <p class="alert alert-danger"><%= err %></p>
 <% } %>
```
err je definiran u routes.js-u, a sintaksa mora biti ovako zapisana kako bi ejs znao koji error se na stranici mora izbaciti.
2) Funkcija router.post
- smije ih biti više, ovisno o tome koliko mogućih redirect-ova ima
- definira ponašanje za klik na određeni gumb.
Primjer:
u datoteci promotional.ejs imamo ovu liniju:
```
<input class="btn" type="submit"
formaction="./promotional/save"
value="Save and return to cart">
```
u datoteci promotional.routes.js definiramo ponašanje kada korisnik klikne taj gumb:
```
router.post('/save', function (req, res,
next) {
    req.session.promotionType =
req.body['promo-type'];
    req.session.promotionCode =
req.body['promo-code'];
    res.redirect('/cart');   })
```
- ovaj primjer konkretno ostvaruje ponašanje da na klik gumba Save, u sjednici se napravi zapis promo tipa I promo koda, zatim redirect-a natrag na stranicu cart.
Na kraj obavezno staviti:
`module.exports = router;`

SJEDNICA (SESSION)
- slijed vremenski omeđenih I logički povezanih transakcija između pojedinog klijenta I poslužitelja
- **početak sjednice** – određen npr. zahtjevom klijenta prema poslužitelju nakon duljeg vremena neaktivnosti ili prijavom korisnika etc.
- **završetak sjednice** – određen prestankom rada klijenta ili odjavom etc.
- sjednica ima **identifikator** – session token, pridijeljen svakoj transakciji koja pripada određenoj sjednici – session token se nalazi u:
  - prvom retku zaglavlja (dio URI)
  - zaglavlju (parovi ime-vrijednost)
  - tijelu zahtjeva ili odgovoru (resursu)
Mehanizmi prenošenja podataka o stanju (obilježavanju sjednice):
- Skrivena polja (hidden fields)
- Prepisivanje URLa (URL rewriting)
- Kolačići (cookies)
SKRIVENA POLJA
→ sadržaj HTML stranice uključuje I podatak o stanju/sjednici
- dinamička stranica stvara HTML document
- skriveno polje izgleda ovako:
`<input name="naziv" type="hidden" value="SID=34cca95f43d">`
Prednosti: - neovisnost o pregledniku, obrasci podržani na svim preglednicima, ne mogu se onemogućiti na pregledniku (za razliku od kolačića), jednostavnost korištenja I performansi
Nedostaci: - lako dostupni pri pogledu na izvorni kod stranice u pregledniku, prenose se kod svake transakcije (u oba smjera), zahtijevaju korištenje obrazaca (zbog akcije submit)
PREPISIVANJE URLa
- poveznice na stranici uključuju podatak o sjednici
- mehanizam automatiziranje promjene URL dolaznog zahtjeva na poslužitelj Web-a, unutar ulaznog niza filtara
- mehanizam dodavanja informacija unutar poveznica HTML stranice dostavljene pregledniku
- podaci: parovi ime-vrijednost
- alternativan mehanizam označavanja sjednica (kada kolačići nisu dostupni)
Prednosti: - potpuna neovisnost o pregledniku, ne može se neomogućiti na pregledniku, jednostavna implementacija
Nedostaci: - podaci se prenose unutar puta upita URI, moguće koristiti vrlo ograničenu količinu podataka, podaci dio svih poveznica koje vode na dinamičke stranice izvorišnog poslužitelja, potrebna dodatna funkcionalnost kod implementacije, ekstrakcija podataka o sjednici iz polja upita URI, smanjena čitljivost poveznica
Primjer korištenja:
```
//create a function for adding sessionID to URL
function sessionURLBuilder(sessionID) {
  //add sessionID parameter to URL query segment
  return function(url) {
    let newURL = new URL(url)
    newURL.searchParams.append(sIDName, sessionID)
    return newURL.toString() } }
```

KOLAČIĆI
Ograničenje na prosljeđivanje iz drugih domena:
SameSite request → First-party cookies (None, Strict, Lax)
CrossSite request → Third-party cookies
Ograničenja (**Nedostaci**) kolačića: Vrlo ograničena količina podataka unutar kolačića (do 4kB), povećanje količina prenošenih podataka, trajnost podataka unutar kolačića, trajnost podataka unutar kolačića (privremeni kolačići, istek roka, ručno brisanje, …), kolačić je vezan na klijenta a ne korisnika (što znači da različiti preglednici ne dijele kolačiće, a različiti korisnici istog pregledника dijele kolačiće),
a što se tiče **Sigurnosti**:
informacije unutar kolačića su vidljive I čitljive kod prijenosa nesigurnim komunikacijskim kanalom, presretanje(krađa) kolačića, **Cross-site scripting**(Session cookie theft, Session hijacking), Cross-site request forgery, Cookie poisoning → third-party cookies su problem privatnosti
STVARANJE SJEDNICA
- Mogući **uzroci nevažeće sjednice**: nepostojeći uvid, istekao rok trajanja sjednice, premašeno vrijeme neaktivnosti sjednice, neodgovarajuća IP adresa klijenta
- Tablica sjednica – tablica sjednica koje su trenutno aktivne
KOMUNIKACIJA S BAZOM PODATAKA
- kako tehnički komunicirati sa BP? → treba nam upravljački program, driver → najpoznatiji ODBC
→ Driver održava **bazen (pool)** N unaprijed otvorenih konekcija s istim korisničkim računom → tzv. **CONNECTION POOLING**
Zašto connection pooling? → Uspostava konekcije je vremenski skupa (~30ms), baze podataka ne mogu podnijeti proizvoljan (tj. beskonačan) broj konekcija – optimiziranje su one za paralelni rad ALI S MJEROM!, upiti na jednoj konekciji se obavljaju serijski →
→ imamo 2 loše opcije: 1) Svaki zahtjev jedna konekcija ILI 2) svi zahtjevi jedna konekcija – Connection pooling je **kompromis** između te dvije loše opcije
*Primjer connection pooling-a u **db/index.js:***
```
const {Pool} = require('pg');
const pool = new Pool ({
  user: 'postgres',
  host: 'localhost',
  database: 'bazepodataka',
  port: 5432, });
module.exports = { query: (text, params) => {
    const start = Date.now();
    return pool.query(text, params).then(res => {
        const duration = Date.now() - start;
        return res; });     },
  pool: pool }
```
*Primjer ostvarivanja connection pooling-a u **server.js**:*
```
const app = express();
const pg = require('pg')
const db = require('./db')
const session = rewuire('express-session')
const pgSession =require('connect-pg-simple')(session)
app.use(session( {
  store : new pgSession( { pool: db.pool,
                           tableName : 'session'}),
          secret : 'nekitekst',
          resave : false,
          saveUnitialized : true,
          cookie : {maxAge : 60*60*24*1000} // 1 dan}));
```
Kako izvršavati query-e? Jedan primjer kako:
```
(async() => { let rows = await db.query('SELECT id,
      name, description
      FROM place_type
      WHERE id BETWEEN $1 AND $2', [1, 3]);
  console.log(rows); // opcionalno
})();
```

COOKIES
Mehanizam razmjene male količine slobodno definiranih podataka između klijenta I poslužitelja unutar svake transakcije HTTP
- Ukupna količina sadržaja unutar kolačića je DO 4kB
- Sadržaj kolačića: jedan par ime=vrijednost
Meta-podaci (svojstva kolačića) → domena(ako nije definirana onda je uzima ime poslužitelja npr. www.fer.unizg.hr), put (domena + put = doseg), rok valjanosti, ograničenje pristupa, ograničenje na sigurnost prosljeđivanja, ograničenja na prosljeđivanje iz drugih domena
- kolačiće **stvara poslužitelj**, definira sadržaj I svojstva kolačića
- kolačiće **pohranjuje klijent**, ako je kolačić istog imena s istog poslužitelja prethodna informacija bit će zamijenjena novom
- kolačiće **šalju** i klijent i poslužitelj, **zahtijeva** ih nitko
Primjer:
```
Set-Cookie: SID=4a57e40d0ee; Path=/nastava;
 Domain=www.fer.unizg.hr;Secure;HttpOnly;
 Expires=Wed, 07 May 2014 10:18:21 GMT
```
Sadržaj kolačića: SID=4a57e40d0ee → sadržaj je OBAVEZAN !!!
Put za koji kolačić vrijedi: Path=/nastava
Domena za koju kolačić vrijedi: Domain=www.fer.unizf.hr
Kolačić se može prosljeđivati samo sigurnim kanalima(Secure)
Kolačiću se ne može pristupati lokalno(HttpOnly)
Kolačić istječe: Expires=Wed, 07 May 2014 10:18:21
Prosljeđivanje kolačića: **UVJETI**: **1.** Poslužitelj pripada domeni za koju je kolačić definiran, **2.** Put definiran za kolačić u cjelini je sadržan unutar puta dohvaćenog resursa, **3.** Kolačiću nije istekao rok trajanja ili nema definiran rok trajanja, **4.** Kolačić koji imaju definirano svojstvo *secure* mogu biti prosljeđivani samo sigurnim kanalima (https, ne http), **5.** Samo ako je to izravno zabranjeno, kolačić neće biti prosljeđen iz druge domene.
Trajnost kolačića: **Trajni** (persistent) kolačići (imaju definirano vrijeme isteka, brišu se nakon roka isteka valjanosti) I **Privremeni** (transient) – session cookies (nemaju definirano vrijeme isteka valjanosti, najčešće imaju maxAge, brišu se nakon prestanka rada klijenta)
Brisanje kolačića – "ručno" (korisnik putem opcije u pregledniku), automatsko brisanje istekom roka valjanosti, promjenom sadržaja kolačića
Pristup kolačiću – udaljen (poslužitelj, mehanizmom prosljeđivanja kolačića putem HTTP odgovora) ili lokalni (klijent, programski)
- ako je postavljeno svojstvo HttpOnly nije moguć lokalni pristup kolačiću
- ograničenje prosljeđivanja samo na sigurne komunikacijske kanale (Secure)

METODE HTTP ZAHTJEVA
OPTIONS - informiranje o mogućnostima poslužitelja i komunikacijskim zahtjevima povezanim s resursom
GET - za dohvaćanje sadržaja resursa (najčešća metoda)
HEAD - za dohvaćanje podataka o resursu (npr. provjera da postoji, veličina)
POST - aktiviranje resursa (npr. slanje podataka obrazaca)
PUT - postavljanje entiteta (npr. promjena podataka - kod REST-a)
DELETE - brisanje resursa (npr. kod REST-a)
TRACE - za dijagnostiku
CONNECT - pokreće dvosmjernu komunikaciju s traženim resursom putem proxy-ja prema web-sjedištu koje koristi HTTPS
Razlika između GET-a i HEAD-a:
- Veličina – GET je puno veći
- Zaglavlja – identična su im
- Response – HEAD nema response
- HEAD - metoda za dohvaćanje podataka o resursu
  - razlika u odnosu na GET: poslužitelj ne vraća sadržaj resursa u tijelu odgovora
  - najčešće se koristi kako bi se provjerilo postoji li entitet na poslužitelju
  - druge uporabe:
    - provjera veličine datoteke prije dohvaćanja
    - pribavljanje metapodataka o entitetu na poslužitelju se zahtjev HEAD obrađuje jednako kao i GET
- POST - metoda za "aktiviranje" resursa znači "pomoću adresiranog resursa obradi podatke koje šaljem"
  - obično se aktivira pritiskom gumba u obrascu

JAVASCRIPT PODSJETNIK (gradivo MI)
DOM – Document Object Model
Glavni objekt DOM-a je document i ima niz podobjekata:
document.documentElement, predstavlja oznaku <html>
document.body, predstavlja oznaku <body>
može biti null ako stranica nije u potpunosti učitana
document.head, predstavlja oznaku <head>
Nad objektima DOM-a moguće je pozivati brojne metode i koristiti brojna svojstva:
.childNodes (svojstvo koje je kolekcija)
.firstChild (objekt koji je prvo dijete nekog objekta)
.lastChild (objekt koji je zadnje dijete nekog objekta)
.nextSibling (sljedeći objekt na istoj razini)
.previousSibling (prethodni objekt na istoj razini)
.parentNode (roditeljski objekt)
.children (elementi djeca)
.firstElementChild (prvo dijete element)
.lastElementChild (zadnje dijete element)
.previousElementSibling (prethodni element na istoj razini)
.nextElementSibling (sljedeći element na istoj razini)
.parentElement (roditeljski element)
document.getElementById(element id value)
Koristi se id elementa
document.querySelectorAll(CSS query) i
document.querySelector
za složenije slučajeve pretraživanja
POLJA
let persons = ["Mark", "John", "Joe"]
push(… items) – dodaje element(e) na kraj
pop() - briše i vraća element s kraja
shift() - briše i vraća element s početka
unshift(…items) - dodaje element(e) items na početak polja
splice(index, num) - od pozicije index briše num elemenata
slice(index1, index2) - vraća novo polje s elementima od pozicija indeks1 do index2
concat(…items) – nadodaje element(e) items u polje i vraća kopiju novog polja
indexOf/lastIndexOf(item, pos) – traži element item, počevši s pozicijom pos
includes(value) – provjerava je li value u polju
find/filter(func) – vraća sve vrijednosti koje zadovoljavaju uvjet (funkciju) func
findIndex(func) - isto kao i find, ali vraća indeks
forEach(func) – poziva funkciju func za svaki element polja
map(func) – zove func za svaki element polja i stvara novo polje temeljem rezultata te ga vraća
→ ove metode ne mijenjaju niz persons, već je njihova return vrijednost izmijenjeni niz.
sort(func) – sortira polje, te ga vraća
reverse() – obrće polje, te ga vraća
split()/join() – pretvara string u polje i obrnuto
JSON
U JavaScriptu postoje dvije glavne metode:
JSON.stringify - pretvara objekte u JSON format
JSON.parse - učitava objekte iz JSON formata
CookieClicker – moduleHpScore.js
```
export class HighScoreManager {
  static getHighScores() {
    const highscores =
localStorage.getItem("highscores");
    if (!highscores) {
      throw new Error("Highscores not set!") }
    return JSON.parse(highscores);
  } //Add new score for user (and remove old one),
and sort the scores for the list
  static updateHighScore(score, user) {
    const highscores =
      HighScoreManager.getHighScores();
    let foundEntry = highscores.findIndex(oneEntry
=> oneEntry.username == user);
    if(foundEntry != -1){
      highscores.splice(foundEntry, 1);}
      highscores.push({ highscore: score,
       username: user });
      highscores.sort(function(a,b)
       {return b.highscore - a.highscore});
      localStorage.setItem("highscores",
       JSON.stringify(highscores)); } }
```
CookieClicker – moduleCookieGame.js (dio)
```
export class CookieGame {
  gameStarted = false;
  cookieCount = 0;
  gameIntervalTimer = null;
  updateTimerUI = null;
  updateHighScoreListUI = null;
  resetGameUI = null;
  constructor(updateTimerUI,
    updateHighScoreListUI, resetGameUI){
    this.updateTimerUI = updateTimerUI;
    this.updateHighScoreListUI =
      updateHighScoreListUI;
    this.resetGameUI = resetGameUI;  }
```

HTTP
primjer HTTP zahtjeva:
GET /predmet/rppzw HTTP/1.1
primjer HTTP odgovora:
HTTP/1.1 200 OK
MIME Type → Multipurpose Internet Mail Extension
Struktura: **tip/podtip** (application/javascript, image/png, text/plain, video/mp4)
- Što možemo provjeriti I mijenjati vezano uz zahtjeve I odgovore? → HTTP metodu, URI, kod odgovora, veličina dostavljenog resursa, vrijeme potrebno za dohvat resursa, zaglavlje zahtjeva I odgovora, cookies
HTTPS
- sigurna verzija protokola HTTP
- koristi protocol TLS (port 443), verzija protokola SSL
- koristi se simetrična kriptografija za šifriranje, generira se ključ sjednice, nema lažnog predstavljanja
- uspostavljanje komunikacije putem TLS-a → dvije osnovne faze: faza rukovanja I faza komunikacije
Opis tijeka komunikacije:
1. Preglednik šalje zahtjev poslužitelju
2. Poslužitelj u odgovoru dostavlja svoj javni ključ i certifikat
3. Preglednik provjerava valjanost javnog ključa i certifikata
4. Ako su certifikat i javni ključ poslužitelja valjani, preglednik generira jednokratni simetrični ključ sjednice
5. Preglednik šalje u odgovoru ključ sjednice šifriran javnim ključem poslužitelja
6. Poslužitelj prima i dešifrira ključ sjednice pomoću svog privatnog ključa i šalje u odgovoru ključ sjednice šifriran samim sobom
7. Ključ sjednice koristi se za daljnje šifriranje poruka između preglednika i poslužitelja
Identifikacija resursa
URI – Uniform Resource Identifier
URN – Uniform Resource Name (npr. urn:ietf:rfc:2616)
URL – Uniform Resource Locator
Analiza URI-ja HTTP-a:
http :// www.fer.unizg.hr / predmet/rppzw
1. http → shema
2. www.fer.unizg.hr → naziv poslužitelja (host name)
3. predmet/rppzw → put I oznaka resursa
Opća sintaksa URI-ja
scheme :// userinfo @ host : port path ? query # fragment
- userinfo → obično se ne koristi
- redoslijed je uglavnom drukčiji – host se nalazi NAKON path-a, a port može biti izostavljen. Primjeri:

https://www.fer.unizg.hr/novosti?@=2tznu#news_94045				
shema	autoritet	put	query	fragment

Kako izgleda HTTP odgovor kada korisnik ispuni HTML formu I submit-a ju? Recimo da imamo ovakvu formu:
```
<form …>
  Ime i prezime: <input type="text"
id="ipr" name="ipr"><br>
JMBAG:<input type="text" id="userID"
name="jmbag"><br>
Uzoran student: <input
type="checkbox" id="usz"><br> <input
type="submit">    </form>
```
I da ju je korisnik ispunio na sljedeći način:
Ime i prezime: Marko Horvat
JMBAG: 32423423423
Uzoran student:
Submit
HTTP response izgleda ovako:
ipr=Marko%20Horvat&jmbag=324234423423
→ u odgovoru se ne nalazi ovo Uzoran student, jer checkbox nema atribut name! da ima pisalo bi: uzoranstudent=checked
HTTP odgovori:
Uspješno obavljeni zahtjevi:
200 OK – uspješno obavljen zahtjev
Redirect (preusmjeravanje) zahtjeva:
300 Multiple Choices
301 Moved permanently – resurs je TRAJNO premješten na drugi poslužitelj
302 Found – resurs je PRIVREMENO premješten na drugi poslužitelj
304 Not Modified – resurs nije modificiran od zadnjeg puta kad smo ga tražili (If-Modified-Since u headeru je false)
Pogreška na klijentu:
400 Bad Request– pogreška u sintaksi zahtjeva
401 Unauthorized – zahtjevu nedostaje autorizacija korisnika
403 Forbidden – resurs je validan, ali server je odbio akciju iz nekog razloga (možda korisnik nema dovoljne ovlasti da traži taj resurs)
404 Not Found – resurs nije dostupan, ali ne ulazi u detalje zašto – obično je greška pri zadavanju URI-ja
Pogreška na poslužitelju:
500 Internal Server Error – obično programska pogreška u resursu
503 Service Unavailable – poslužitelj je preopterećen – odgovor može sadržavati polje Retry-After
- razlika između 401 I 403 je što u 401 korisnik uopće nije izvršio prijavu, a u 403 je manjak ovlasti

-

PODSJETNIK – gradivo MI
CSS → bodovanje specifičnosti:
- Inline stilovi (1000) (span, itd.),
- #ID selektori(100),
- .class i [attribute] i :pseudo-class (10),
- <TAG> (osim inline) i ::pseudo-element (1),
- <body> i * (0)
- ako je težina ista, gleda se poredak I pobjeđuje zadnji
Relativne I apsolutne jedinice:
em → relativna veličina fonta u odnosu na roditelja
rem → relativna veličina fonta u odnosu na root element
px → apsolutna jedinica, 1 piksel
→ Primjer korištenja relativnih jedinica **rem** i **em**: (default fs je 16px)
```
<div style="font-size: 30px;"
   <span style="font-
size:.">em</span> //fs1
   <span style="font-size:
1rem;">rem</span> //fs2
</div>
```
fs1 = 1*30px = 30px, fs2 = 1*16px = 16px
→ justify-content: flex-start, flex-end, center, space-between, space-around, space-between, space-evenly
align-content: isto kao justify-content, ali po y osi!
→ Kako da odredimo ponašanje za podčlan nekog člana liste? **li li { … }**
HTML → forms – padajuća lista:
```
<select name="jezik" size="5"
(multiple)>
<option value="njemački">Njemački
jezik</option>
<option value="francuski"
selected>Francuski jezik</option>
</select>
```
→ razlika između `<a>` I `<link>` → <link> služi samo za povezivanje elemenata, dok je <a> neka poveznica koja se može pojaviti na stranici na koju možemo kliknuti. Primjeri korištenja:
```
<link rel="stylesheet"
type="text/css"
href="styles/main.css">
<a href="index.html" class="link
menu-item">Home</a>
```
→ Tablica u HTML-u – primjer:
```
<table …>
  <tr> <td colspan="2">A</td>
  <td rowspan="2">C</td> </tr>
  <tr> <td>B</td><td>B</td></tr>
</table>
```
A	C
B	B

```
router.get('/', function(req, res,
next) {
  res.render('home', {
    title: 'Home'} → će tražiti home.ejs u konfiguriranom direktoriju (views)
```

→ *"Napiši osnovne dijelove HTML-a I JS-a da bi se ostvario*

- 0 +

ovakav brojač: - Raspon su mu brojevi 0-10. Kada se stranica učita, vrijednost mora biti 0."

HTML:

```
<button onclick="decrement()"><-</button>
<input id=numInput type=number min=0 max=10
value='0'>
<button onclick="increment()">+</button>
```

JS (unutar istog HTML-a):

```
<script>
  function increment() {
    document.getElementById('numInput').stepUp();}
  function decrement() {
 document.getElementById('numInput').stepDown();}
</script>
```

→ Prilikom slanja podatka **iz obrasca na web-poslužitelj** nazivi i vrijednosti elemenata obrasca će se u HTTP-zahtjevu prenijeti: metodom **GET** u **URI/URL-u** HTTP-zahtjeva kao parametri upita metodom **POST** u **tijelu** HTTP-zahtjeva

- Za upravljanje ponašanjem priručnog spremišta koristi se polje **Cache-Control** (pojavljuje se u zahtjevu I odgovoru)

→ Koja je KONKRETNA **razlika između ** (uređene liste) **i ** (neuređene liste)? predstavlja sekvencu, dok <ul> predstavlja natuknice. U čistom HTML-u <ol> će dati brojeve (1., 2., 3., ...) a <ul> će dati bulletpoints (• natuknica \n • druga natuknica ...)

→ Primjer opisne liste **<dl>:**

```
<dl>
  <dt>name:</dt><dd>John Doe</dd>
  <dt>tel:</dt><dd>01-2345678</dd>
  <dt>fax:</dt><dd>02-3456789</dd>
  <dt>email:</dt><dd>johndoe@someemail.com</dd>
</dl>
```

"Napišite HTML kôd obrasca za unos registracijskih podataka (HTML III ejs)"

```
create table user (
   email text primary key,
   ime text not null,
   prezime text not null  )
<form action="/register" method="POST">
  <fieldset>
    <legend>Registration data</legend>
    <div>
      <label for="ime">Ime:</label>
      <input type="text" name="ime" id="ime"
size="30">
    </div>
    <div>
      <label for="prezime">Prezime:</label>
      <input type="text" name="prezime"
id="prezime" size="30">
    </div>
    <div>
      <label for="email">Email:</label>
      <input type="text" name="email" id="email"
size="30">
    </div>
      <input type="submit" value="Submit">
  </fieldset>
</form>
```

"Napišite kôd na poslužitelju (samo metodu kontrolera, nije potrebno usmjeravanje/routing) koji obrađuje zahtjev koji se šalje iz zadatka (b), tj. obavlja opisanu validaciju i registrira korisnika ako je validacija uspješna, pritom unoseći novi zapis u bazu podataka (strukturu jednostavne tablice korisnika osmislite sami)."

```
router.post('/', async function(req, res) {
 try {
  if (req.body.ime && req.body.prezime &&
      req.body.ime.length > 2 &&
      req.body.prezime.length > 2) {
   let rsp = await db.query(`SELECT COUNT(*) as
                cnt FROM users WHERE email = $1`,
                [req.body.email]);
   if (rsp[0].cnt > 0) {
     res.render('register', {
       msg: 'Email nije jedinstven.'    });
   } else {
     await db.query(`INSERT INTO user(ime,
       prezime, email) VALUES ($1,$2,$3)`,
      [req.body.ime, req.body.prezime,
       req.body.email]);
     res.render('ok');   }
 } else {
   res.render('register', {
     msg: 'Ime i prezime moraju imati barem dva
       znaka.' });   }
} catch (error) {
   res.render('register', {
     msg: 'Došlo je do pogreške prilikom
     registracije.'  });   }
});
```

Primjer padajućeg izbornika u ejs-u:

```
<div class="form-group">
  <% const promoOptions = ["Delivery", "Total Price"]%>
  <label class="form-label" for="promo-type">Promotion
  type:</label>
  <select class="form-component" id="promo-type"
  name="promo-type">
    <%for (const opt of promoOptions) { %>
    <option <%= promoType == opt ? 'selected' : '' %>>
      <%= opt%>
    </option>
    <%}%>
  </select>
</div>
```

Primjer submit-anja obrasca (EJS – promotional.ejs):

```
<input class="btn" type="submit"
formaction="./promotional/save" value="Save and return
to cart">
<input class="btn" type="submit"
formaction="./promotional/reset" value="Reset">
<input class="btn push-right" type="submit"
formaction="./promotional" value="Order now!">
```

Primjer submit-anja obrasca (cijeli promotional.routes.js):

```
const express = require('express');
const router = express.Router();
const cart = require('../models/CartModel');
const authHandler = require('./helpers/auth-handler');

router.get('/', authHandler, function (req, res, next) {
  res.render('promotional', {
    title : 'Enter promo code',
    linkActive : 'cart',
    cart : req.session.cart,
    user : req.session.user,
    promoType : req.session.promotionType,
    promoCode : req.session.promotionCode,
    err : undefined });   });
router.post('/', function (req, res, next) {
  req.session.promotionType = undefined;
  req.session.promotionCode = undefined;
  res.redirect('/checkout');  })
router.post('/save', function (req, res, next) {
  req.session.promotionType = req.body['promo-type'];
  req.session.promotionCode = req.body['promo-code'];
  res.redirect('/cart');  })
router.post('/reset', function (req, res, next) {
  req.session.promotionType = undefined;
  req.session.promotionCode = undefined;
  res.redirect('/promotional');  })
module.exports = router;
```

Primjer router.post fje pri prijavi korisnika (login.routes.js):

```
router.post('/', function (req, res, next) {
  (async () => {  let user =
    await User.fetchByUsername(req.body.user);
    if(user.id === undefined || !user.checkPassword
    (req.body.password)){
      res.render('login', {
        title : 'Log in',
        linkActive : 'login',
        user : req.session.user,
        err : 'Incorrect username or password.'});
      return;  }
      req.session.user = user;
      res.redirect('/');
  })()
});
```

Pretpostavimo da želimo provjeriti jedinstvenost e-mail adrese bez predavanja cijelog obrasca i u tu svrhu dodajemo gumb „Provjeri e-mail" koji treba korisniku dojaviti je li e-mail slobodan ili ne. Objasnite što je potrebno napraviti u kojem slolju da bi se to ostvarilo. Napišite kôd koji je potrebno dodati na klijentskoj strani (preglednik). Pretpostavite da poslužitelj vraća JSON s jednim boolean podatkom (oznakom je li adresa valjana ili ne).

```
<button id="btnCheck" type="button"> check
</button>
<script>
  document.getElementById("btnCheck").onclick =
   async function() {
     let response = await fetch
      ('http://localhost:3000/register
       /check?email=' +
       document.getElementById("email").value);
     if (response.ok) {
       let jsonResp = await response.json();
       if (jsonResp.isUnique) {
         alert("You're fine");
       } else {
         alert("Not unique, sorry.");  }
     } else {
       alert("HTTP-Error: " + response.status); }
   }
</script>
```

CartModel.js

```
const db = require('../db')
function createCart() {
  let newCart = {}
  newCart.totalAmount = 0
  newCart.items = {}
  return newCart }
function updateCart() {
  cart.totalAmount = 0
  for(let item of Object.values(cart.items))
    cart.totalAmount += item.price * item.quantity}
async function addItemToCart(cart, id, quantity) {
  let itemObject = cart.items[id]
  if( itemObject === undefined ) {
    itemObject = {id: id, name: undefined, price:
      undefined, quantity: 0, imageurl: undefined}
    cart.items[id] = itemObject }
  if(itemObject.price === undefined) {
    if( itemData = await getItemData(id) ) {
      itemObject.name = itemData.name
      itemObject.price = itemData.price
      itemObject.imageurl = itemData.image }}
  itemObject.quantity =
     itemObject.quantity + quantity
  update(cart)   }
async function removeItemFromCart(cart, id, quantity){
  let itemObject = cart.items[id]
  if( itemObject ) {
    let newQuantity =
      Math.max(0, itemObject.quantity - quantity)
    if( newQuantity )
      itemObject.quantity = newQuantity
    else
      delete cart.items[id] }
  update(cart)   }
async function getItemData(id) {
  const sql = `SELECT name, price, imageurl FROM
  inventory WHERE id = ` + id;
  try {
    const result = await db.query(sql, []);
    return {
      name: result.rows[0].name,
      price: result.rows[0].price,
      image: result.rows[0].imageurl  }
  } catch (err) {
      console.log(err);
      throw err }
  return undefined  }
module.exports = {
  createCart,
  addItemToCart,
  removeItemFromCart  }
```

Server.js

```
const express = require('express');
const app = express();
const path = require('path');
const pg = require('pg')
const db = require('./db')
const session = require('express-session')
const pgSession = require('connect-pg-
simple')(session)
//middleware - predlošci (ejs)
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'ejs');
//middleware - statički resursi
app.use(express.static(path.join(__dirname,
'public')));

const homeRouter = require('./routes/home.routes');
const orderRouter = require('./routes/order.routes');
const loginRoute = require('./routes/login.routes');
const logoutRoute = require('./routes/logout.routes');
const signupRoute = require('./routes/signup.routes'); ...
app.use(session( {
   store : new pgSession(
    { pool : db.pool,
      tableName : 'session'}),
      secret : '4. Labos',
      resave : false,
      saveUninitialized : true,
      cookie : {maxAge : 60 * 60 * 24 * 1000} } ));
app.use((req, res, next) => {
   if (req.session.cart === undefined) {
      req.session.cart = cart.createCart();}
      next();  });
//definicija ruta
app.use('/', homeRouter);
app.use('/order', orderRouter);
app.use('/login', loginRoute);
app.use('/logout', logoutRoute);
app.use('/signup', signupRoute);

app.listen(3000);
```

UserModel.js – iz 4. Labosa

```
const db = require('../db')
//razred User enkapsulira korisnika web trgovine
module.exports = class User {
  //konstruktor korisnika
  constructor(username, first_name, last_name, email, password,
role="user") {
     this.id = undefined
     this.user_name = username
     this.first_name = first_name
     this.last_name = last_name
     this.email = email
     this.password = password
     this.role = role  }
//dohvat korisnika na osnovu korisničkog imena
static async fetchByUsername(username) {
  let results = await dbGetUserByName(username)
  let newUser = new User()
  if( results.length > 0 ) {
    newUser = new User(results[0].user_name,
results[0].first_name, results[0].last_name, results[0].email,
results[0].password, results[0].role)
newUser.id = results[0].id  }
  return newUser }
//dohvat korisnika na osnovu email adrese
static async fetchByEmail(email) {
  let results = await dbGetUserByEmail(email)
  let newUser = new User()
  if( results.length > 0 ) {
    newUser = new User(results[0].user_name,
    results[0].first_name, results[0].last_name,
    results[0].email, results[0].password, results[0].role)
    newUser.id = results[0].id }
  return newUser }
//dohvat korisnika na osnovu id korisnika (tablica users)
static async fetchByUserId(id) {
  let results = await dbGetUserById(id)
  let newUser = new User()
  if( results.length > 0 ) {
    newUser = new User(results[0].user_name,
    results[0].first_name,results[0].last_name,
    results[0].email, results[0].password, results[0].role)
    newUser.id = results[0].id }
  return newUser }
//da li je korisnik pohranjen u bazu podataka?
isPersisted() {
  return this.id !== undefined }
//provjera zaporke
checkPassword(password) {
  return this.password ? this.password == password : false  }
//pohrana korisnika u bazu podataka
async persist() {
  try {
    let userID = await dbNewUser(this)
    this.id = userID
  } catch(err) {
    console.log("ERROR persisting user data: " +
JSON.stringify(this))
    throw err }  } }
//dohvat korisnika iz baze podataka na osnovu korisničog imena
(stupac user_name)
dbGetUserByName = async (user_name) => {
  const sql = `SELECT id, user_name, first_name, last_name,
email, password, role FROM users WHERE user_name = '` +
user_name + `'`;
  try {
    const result = await db.query(sql, []);
    return result.rows;
  } catch (err) {
    console.log(err);
    throw err } };
//dohvat korisnika iz baze podataka na osnovu id korisnika
(stupac id)
dbGetUserById = async (user_id) => {
  const sql = `SELECT id, user_name, first_name, last_name,
email, password, role
  FROM users WHERE id = ` + user_id;
  try {
    const result = await db.query(sql, []);
    return result.rows;
  } catch (err) {
    console.log(err);
    throw err}}
//umetanje zapisa o korisniku u bazu podataka
dbNewUser = async (user) => {
  const sql = "INSERT INTO users (user_name, first_name,
last_name, email, password, role) VALUES ('" + user.user_name +
"', '" + user.first_name + "', '" + user.last_name + "', '" +
user.email + "', '" + user.password + "', '" + user.role + "')
  RETURNING id";
  try {
    const result = await db.query(sql, []);
    return result.rows[0].id;
  } catch (err) {
    console.log(err);
    throw err } }s
```

2. Labos – primjer dodavanja proizvoda na stranicu orde putem DOM-a:

```
<template id="category template"> ...
</template id>
<script>
  let getData = async function () {
    let response = await
    fetch("https://weblab2.azurewebsites.net/
    categories");
    let data = await response.json();
    addCategories(data);  }
  let addCategories = async function
  (categories) {
    let main = document.querySelector('main');
    let categoryTemplate = document
    .querySelector('#category-template');
    for (let index = 0; index < categories.
    length; index++) {
    let category =
    categoryTemplate.content.cloneNode(true);
    let categoryTitleElement = category.
    querySelector('.decorated-title > span');
    categoryTitleElement.textContent =
    categories[index].name;
    let gallery =
    category.querySelector(".gallery");
    main.appendChild(category);
    let ProductResponse = await
    fetch("https://weblab2.azurewebsites.net/
    products?categoryId=" + (index + 1));
    let ProductData = await
    ProductResponse.json();
    let productTemplate = document
    .querySelector("#product-template");
    for (let i = 0;
     i < ProductData.length; i++) {
      let product = productTemplate
      .content.cloneNode(true);
      let productTitle = product.
      querySelector(".photo-box-title");
      productTitle.textContent =
      ProductData[i].name;
      let productImage = product
      .querySelector(".photo-box-image");
      productImage.src =
      ProductData[i].imageUrl;
      let cartButton = product
      .querySelector(".cart-btn");
      cartButton.onclick = function() {
        addToCart(ProductData[i].id);
      };
    gallery.appendChild(product); }
  }
};
updateCart();
getData();
</script>
```

Primjer obećanja:

```
let promise = new Promise(function(resolve,
reject){
  let randomNumber =
  Math.floor(Math.random() * 10);
  console.log(randomNumber);
  if(randomNumber%2==1){
    setTimeout(() => {
     resolve("An odd random number was
     generated.");
    }, 5000);
  } else {
    setTimeout(() => {
    reject(new Error("This is generated from
    the promise executor!"));
    }, 1000);
  } });
promise.catch(
  function(error){ console.log(error);}).then(
    function(result){ console.log("Resolve:" +
    result) },
    function(result){ console.log("Reject:" +
    result) }
);
```

OBECANJA

- na njima "registriramo" metode:
  - .then → aktivira se ako je izvršna funkcija vratila uspjeh ili neuspjeh
  - .catch → aktivira se **samo** ako je izvršna funkcija vratila neuspjeh
  - .finally → aktivira se ako je izvršna funkcija vratila uspjeh ili neuspjeh (ali za razliku od then ne daje mogućnost obrade uspjeha ili neuspjeha)

ASINKRONOST

Što se sve obavlja asinkrono? → IO operacije (dohvat podataka preko mreže, čitanje i pisanje na disk, ..) ; ovisi i o operacijskom sustavu ; ovisi i što programer koristi

Preglednik šalje sljedeći zahtjev poslužitelju www.fer.unizg.hr na vrata 80:

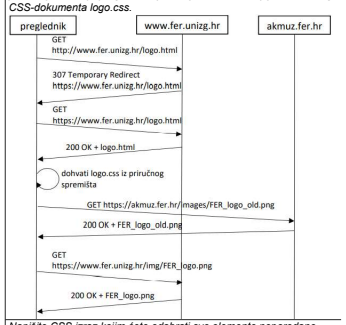
GET /logo.html HTTP/1.1

Host: www.fer.unizg.hr

Traženi resurs logo.html prikazan je:

```
<!DOCTYPE html>
<html>
  <head>
    <link rel="stylesheet" href="logo.css">
  </head>
  <body>
    <div>
     <img class="left"
     src = "https://akmuz.fer.hr/
     images/FER_logo_old.png">
     <h2>Stari logo vs. novi logo</h2>
     <p>Koji vam se više svida?</p>
     <img class="right" src="./img/FER_logo.png">
    </div>
  </body>
</html>
```

Skicirajte slijedni dijagram i označite sve akcije koje će preglednik napraviti vezano uz poslužiteljе ili priručnu memoriju te odgovore poslužitelja. Pretpostavite da je poslužitelj www.fer.unizg.hr konfiguriran tako da koristi protokol https te na prvi navedeni zahtjev preglednika odgovara s 307 Temporary Redirect. Svi poslužitelji su dostupni i mogu dostaviti sve tražene resurse, dok je u priručnoj memoriji preglednika pohranjena ažurna kopija navedenog CSS-dokumenta logo.css.



Napišite CSS izraz kojim ćete odabrati sve elemente neporedane liste koji imaju id (bilo kakav) ili imaju klasu „zeleno" te im postaviti boju pozadine na zeleno.

```
ul>li[id], ul>li.zeleno {
  background-color: green; }
```

Koristeći slijedni dijagram nacrtajte scenarij u kojem korisnik pristupa stranici za registraciju na sljedeći način: • prvo unosi neispravno (prazno) ime (što rezultira neuspješnom registracijom) • zatim popravlja ime, ali unosi e-mail koji nije jedinstven (što rezultira neuspješnom registracijom) • konačno, unosi ispravne podatke i uspješno se registrira.