HTML

Globalni atributi: definirani za sve elemente (id, class, lang, title, style, hidden, data-*)

Uključivanje vanjskog CSS-a: <link rel="stylesheet" href="site.css">
Uključivanje vanjskog JS-a: <script src="index.js"></script>

Blok: započinje u novom retku i proteže se cijelom širinom stranice (div, list, section, pre…)
Inline – započinje od prvog praznog mjesta i širok je koliko mu treba za prikaz (span, b, del…)

Komentar: <!-- -->

Slike: inline-block (nema line break nakon elementa, stoje u istom redu s tekstom, ali mogu se namještati visina i širina), definiranje dimenzija preko atributa width i height ili preko glob. atributa style (preporuka), element figure odvaja sliku od ostatka stranice <figure><img><figcaption>…

Liste: blok, <ul style="list-style-type:square;">, <ol type="1" start="0" reversed>, <li>, <dl>, <dt>, <dd>

Tablice: blok, <table><tr><td>, colspan, rowspan

Obrasci: parovi name1=value1&name2=value2, ako name nije naveden, podatak se ne šalje
- atributi elementa form – action (uri resursa kojem se podaci šalju), target (cilj odgovora, _self, _blank, itd.), method (GET/POST)
- <input type=text, password, number radio, checkbox, button, submit, date, file
- required, checked, hidden, disabled
- svi radio iz iste grupe imaju isti name, a checkboxovi trebaju imati različit name
- dropdown: <select name=…><option value=… selected>
- umjesto razmaka +, ć - %C4%87
- ne šalju se elementi koji su disabled i buttoni koji nisu aktivirani (submit je aktiviran, ako ima name i value poslat će se)

CSS

Grupiranje jednostavnih selektora:
h1.naslov – elementi h1 koji imaju klasu naslov (logičko i)
h1, li – svi elementi h1 i li (logičko ili)

Atributni selektori: li[id="z2"] – elementi li kojima je atribut id jednak "z2"

Kombinirani selektori:
- div span – svi elementi span unutar div, bez obzira koliko duboko
- div>span – el. span koji su neposredna djeca el. div
- div+span – el. span koji ima istog roditelja kao div i stoji neposredno poslije njega
- div~span – el. span koji ima istog roditelja kao div i stoji poslije njega

Selektori pseudo-klasa: :hover, :first-child(el. koji su prvo dijete svog roditelja), :required(el. koji imaju atribut required)

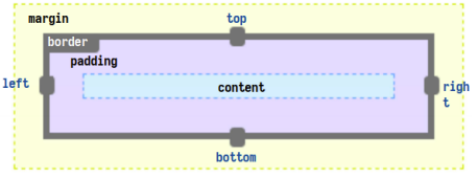
Selektori pseudo-elemenata: ::after, ::before, ::first-letter, ::first-line, ::selection

Kaskadnost:
1. izvor (author, user, user-agent)
2. specifičnost:
 - inline (1000)
 - #id (100)
 - .class, :pseudo-class, [attribute] (10)
 - <tag>, ::pseudo-element (1)
 - <body>, * (0)
3. last wins
- naslijeđena svojstva imaju najmanji prioritet

Font: dohvaćanje fonta - <link href="…" rel="stylesheet">
- 1 rem: vel. fonta korijenskog elem.
- 1 em: vel. fonta roditeljskog elem.

Boja: rgb(255, 255, 255), rgba(255, 255, 255, 1)

Box-model:



- display: block|inline|inline-block, za inline margin-top i bottom nemaju učinka, ne može se namještati width i height
- *{box-sizing: border-box;} – visina i širina odnose se na cijeli box, inače bi bilo samo na sadržaj

Pozicioniranje:
- static: default
- relative: u odnosu na izvornu poziciju
- fixed: ne miče se skrolanjem
- absolute: s obzirom na najbližeg pozicioniranog pretka

Responzivnost:
<meta name="viewport" content="width=device-width, initial-scale=1">
- @media screen and (min-width: 768px){…} – primjenjuje se na ekrane šire od 768px

Flex i grid: - display: flex, flex-flow – flex-direction(row|column|row-reversed…) + flex-wrap(nowrap|wrap|wrap-reverse)
- display: grid, grid-template-columns: 1fr 1fr 1fr

JS

- var ima scope tijela funkcije, let i const bloka u kojem se nalaze
- for…of iterira po elementima, for…in iterira po indeksima

Objekti:
- kopija: let personClone = Object.assign({}, person);
- Object.keys(obj), Object.values(obj), Object.entries(obj)
- klase mogu imati samo jedan konstruktor
- obj = new MyClass();
- obj instanceof MyClass

Polja:
- push(items), pop, shift, unshift(items), splice(index, num) – briše num elemenata od pozicije index, slice(index1, index2) – vraća novo polje elem. od index1 do index2, concat(items), indexOf(item, pos), includes(item), filter(func), forEach(func), map(func) – vraća novo polje s rezultatima pozivanja func nad svakim elementom, reverse(), split(…)/join(…) – pretvara string u polje i obrnuto

DOM:
-document.getElementById(id), document.querySelectorAll(cssSelector)
- promjena stila: document.getElementById('boja').style.color = ''red'';
- dodavanje elementa: document.querySelector(''body'').appendChild(document.createElement(''div''));
- pisanje u neki element: el.innerHTML = 'nesto' ili innerText
- kloniranje elementa: document.querySelector(…).content.cloneNode(true);
- BOM: window – newWindow = window.open(url), newWindow.close(), location – trenutni url (location.search)
- alert, prompt, confirm

JSON: .stringify – objekt u string, .parse – string u objekt
LocalStorage: setItem(key, value), getItem(key)

Asinkronost: kada je call stack prazan, event loop uzima callbackove iz message queue i stavlja ih na call stack. Obećanja imaju veći prioritet od callbackova.

- uključivanje modula: const modul = require(…), module.exports = {a: b}

HTTP

- zahtjev: 1. metoda uri verzija, 2. host
- odgovor: 1. verzija status
- ostala polja su formata ključ: vrijednost, prazan red prije tijela
- Content Type: tip/podtip
- HTTP 80, HTTPS 443
- URN i URL su podskupovi URI-ja
- 100 informativni, 200 ok, 300 redirect (304 not modified, 301 moved permanently, 302 found), 400 greška klijenta, 500 greška servera
- opća struktura URI-ja: <shema>//<autoritet><put>?<upit>
- <autoritet> = <host>:<vrata> (definira klijent)
- za URI http://www.fer.unizg.hr/predmet/rppzw shema (koristi klijent): http:, hostname (koristi klijent): www.fer.unizg.hr, oznaka resursa (koristi server): /predmet/rppzw

NODE.JS/EXPRESS

Pg: const {Pool} = require('pg');
const pool = new Pool({user, pass…});
module.exports = {query: (text,…) => {await pool.query(text,…);}}

app.get('/', (req, res) => {res.send(…)});
Middleware: app.use(function(req, res, next) {…});
- ugrađene middleware funkcije: express.static('direktorij'), express.json(), express.urlencoded()

MVC: model – dohvat podataka iz baze, view – prezentacija, controller – prima zahtjeve, routing

EJS: <% code %>, <%= code %>
res.render('ime viewa', {model});

SJEDNICE

- mehanizmi prenošenja podataka o stanju sjednice – skrivena polja, prepisivanje URL-a, kolačići

Kolačići:
- sadržaj: 1 par ime=vrijednost (do 4kB)
- metapodaci: domena, put, rok valjanosti, ograničenja na prosljeđivanje
- ako nije navedena domena/put, default je domena/put servera koji postavlja kolačić
- generira ih server i uključuje u zaglavlje HTTP odgovora (npr. Set-Cookie:sid=123)
- pohranjuje ih klijent, ako je kolačić istog imena s istog servera već definiran prepisuje se
- HttpOnly kolačićima se ne može pristupati lokalno
- klijent pri svakom slanju zahtjeva pretražuje spremište kolačića, ako pronađe kolačić koji odgovara uvjetima (domena kolačića sadržana u domeni servera, put kolačića sadržan u putu zahtjeva, rok trajanja nije istekao ili nije definiran) šalje ga u zaglavlju zahtjeva
- od klijenta prema poslužitelju šalje se samo ime=vrijednost, bez metapodataka
- u istom zahtjevu može biti poslano više kolačića (ako imaju različita imena)
- kolačići s maxAge=0 brišu se nakon završetka sjednice (sjednički ili tranzijentni kolačići)
- kolačić s definiranim svojstvom Secure može biti poslan samo preko HTTPS
- SameSite=strict: kolačić se prosljeđuje samo u sklopu zahtjeva istom sjedištu (npr. neće biti proslijeđen kod dohvata resursa s druge stranice), SameSite=lax: kolačić se prosljeđuje kod navigacijskih zahtjeva na druge stranice, ali ne kod dohvata resursa, SameSite=none: kolačići se uvijek prosljeđuju, ali onda moraju biti Secure

Sjednice korištenjem kolačića:
1. kada dođe zahtjev bez kolačića, stvori id sjednice i pohrani ga u bazu, a zatim vrati id u kolačiću
2. kada dođe zahtjev sa sjedničkim kolačićem, iz baze izvuci kondekst sjednice s odgovarajućim id-om

Korisničke sjednice:
1. nakon prijave korisnika stvori asocijaciju sjednice i korisnika
2. za zahtjeve koji sadrže sjedničke kolačiće provjeri je li sjednica asocirana s nekim korisnikom, ako je obnovi sjednički i korisnički kontekst

Express-session:
- sprema podatke o sjednici u tablicu s 3 stupca: sid, sess i expire
- kolačić koji se prosljeđuje klijentu sadrži sid
- u stupcu sess nalaze se podaci o sjednici, npr. user, cart… i oni se asociraju s id-om sjednice
- u stupac sess možemo dodati proizvoljne podatke tako da ih zakačimo na session objekt koji generira express-session middleware na temelju kolačića kojeg dobije od klijenta

```
const {Pool} = require('pg');

const pool = new Pool({
    user: 'postgres',
    host: 'localhost',
    database: 'web1-lab3',
    password: 'bazepodataka',
    port: 5432,
});
```

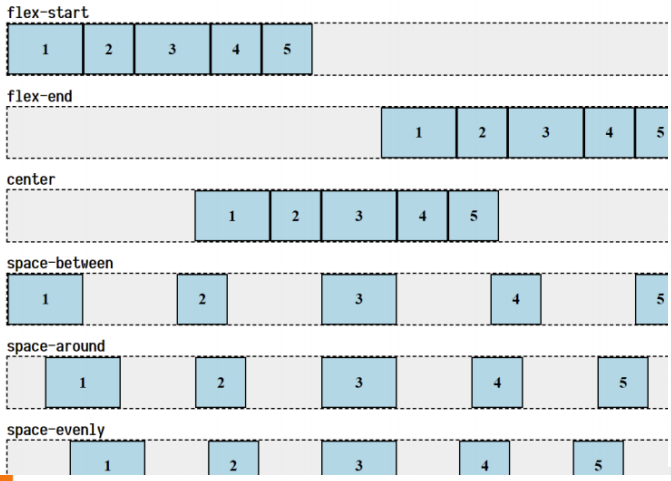
```
// fetch products for category
let response = await fetch(https://web1lab2.azurewebsites.net/products?categoryId=
         + categories[index].id);
let products = await response.json()
// create product elements from template and append them to category element
for (productObject of products) {
   let product = productTemplate.content.cloneNode(true);
   product.querySelector('.photo-box-image').src = productObject.imageUrl;
   product.querySelector('.photo-box-title').textContent = productObject.name;
   // add action to cart button
   id = productObject.id;
   product.querySelector('.cart-
btn').onclick = (function(id) {return function() {addItemToCart(id)}}(id));
   category.querySelector('.gallery').appendChild(product);
}
```

```
<form target="_self" action="/processForm.php" method="GET">
 <p><label>Korisničko ime: <input type="text" name="username" value="enter your username"
 size="30"></label></p>
 <fieldset>
  <legend>Uloga</legend>
  <p><label><input type="radio" name="role" value="admin" checked>Administrator</label></p>
  <p><label><input type="radio" name="role" value="user">Korisnik</label></p>
  <p><label><input type="radio" name="role" value="guest">Gost</label></p>
 </fieldset>
</form>
```

```
class MyClass {
  // declared property
  prop1 = "Value";
  // constructor
  constructor(prop2Value) {
    this.prop2 = prop2Value;
  }
  // member methods
  method1() { ... }
  method2() { ... }
  //getter method
  get getterName(...) {}
  //setter method
  set setterName(...) {}
}
```

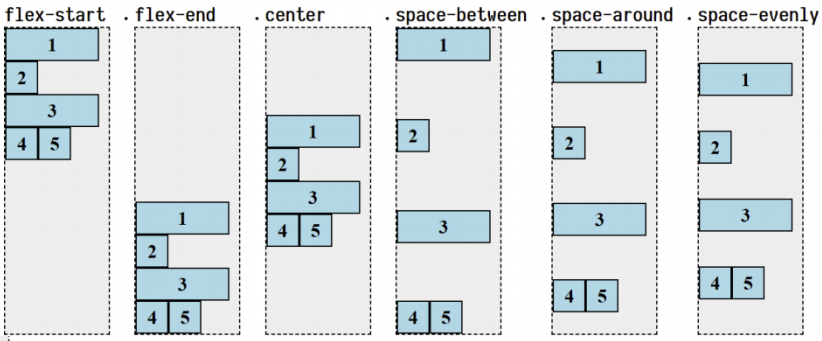
justify-content primjer

Definira poravnanje po glavnoj osi, primjer:

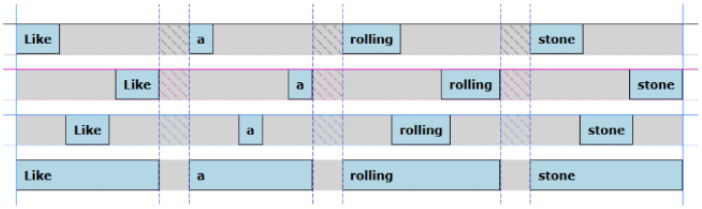


align-content primjer

- Nalik justify-content ali po **sporednoj** osi, primjer:
  - (ako je samo jedna linija onda ovo svojstvo nema učinka)



justify-items: poravnanje po *inline/row* osi




```
/* 4 grida s klasama d1-d4     */
.d1 {
  justify-items: start;
}
.d2 {
  justify-items: end;
}
...
```

```
...
.d3 {
  justify-items: center;
}

.d4 {
  justify-items: stretch;
}
```

align-items: poravnanje po *block* osi

```
/* 4 grida s
klasama d1-d4     */
.d1 {
  align-items: start;
}

.d2 {
  align-items: end;
}

.d3 {
  align-items: center;
}

.d4 {
  align-items: stretch;
}
```


```
let promise = fetch("s12.promise.json");
promise
.then(
 function(response) {
  if (!response.ok) { throw new Error("Cannot load json file"); }
  else { return response.json(); }
 },
 function(error) { throw error; }
)
.then(jsonContents => {
  let spanElement = document.createElement('pre');
  spanElement.innerHTML = JSON.stringify(jsonContents);
  document.body.appendChild(spanElement);
 })
.catch(err => {
  console.log("An error occured while loading json");
});
```

```
promise.catch(
  function(error){
    return new Promise(function(resolve, reject){
      reject(error);
    });
})
.then(
  function(result){ console.log("Resolve:" + result) },
  function(result){ console.log("Reject:" + result) }
);
```

```
let promise = new Promise(function(resolve, reject) {
 // the code executes immediately when the constructor is called
});
promise.then(
 function(result) { /* handle a successful result */ },
 function(error) { /* handle an error */ }
);
```

```
@font-face {
  font-family: 'Dancing Script';
  src: url(...) format("woff2"), url(...) format("woff");
  font-weight: 500;
  font-style: normal;
}
```

```
doSomething()
.then(function(result) {
  return doSomethingElse(result);
})
.then(function(newResult) {
  return doThirdThing(newResult);
})
.then(function(finalResult) {
  console.log('Got the final result: ' + finalResult);
})
.catch(failureCallback);
```

```
router.post('/', async function (req, res, next) {
    //###################### ZADATAK ######################
    //postupak prijave korisnika
    console.log(req.body.username);
    let user = await User.fetchByUsername(req.body.user);
    if (!user.checkPassword(req.body.password)) {
        req.session.err = 'User does not exist or incorrect password';
        req.session.save(() => { res.redirect('/login') });
    } else {
        req.session.user = user;
        req.session.save(() => { res.redirect('/') });
    }
```

```
app.use(session({
    secret: 'random tekst',
    resave: false,
    store: new pgSession({pool: db.pool}),
    saveUninitialized: true
}));
```

```
router.get('/:itemId([0-9]*)/editSupplier/:supplierId([0-9]*)', async (req
    let itemId = parseInt(req.params.itemId);
    let supplierId = parseInt(req.params.supplierId);
```

```
<%- include(`partials/header`); %>
```

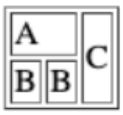
```
async function LoadJSON(){
  let promise = await fetch("s15.asyncAwaitFetch.json");
  if (!promise.ok) { throw new Error("Cannot load json file"); }
  else { var jsonContents = await promise.json();}
  let spanElement = document.createElement('pre');
  spanElement.innerHTML = JSON.stringify(jsonContents);
  document.body.appendChild(spanElement);
}

LoadJSON().catch(err => {
  console.log("An error occured while loading json: " + err);
});
```

Prikaz JSON-a na web-stranici

Greške se hvataju s catch, budući da funkcije obilježene s async uvijek vraćaju obećanja (čak i ako se dogodi greška)

```
router.post('/add', [
  body('title').not().isEmpty()
               .trim().escape(),
  body('author').not().isEmpty()
               .toInt(),
  check('language').trim()
       .isLength({min: 2, max: 2}),
  body('publisher').not().isEmpty()
       .trim().escape(),
  body('ISBN').not().isEmpty()
       .trim().custom(value => {
     return validateISBN(value)
  }).withMessage('ISBN must be a 1
or 13 digit number that confirms t
the standard:
https://en.wikipedia.org/wiki/Inte
ational_Standard_Book_Number')
], function (req, res, next) {
...
```

```
..., function (req, res, next) {
  const errors = validationResult(req);
  if (!errors.isEmpty()) {
    res.render('addBook', {
      title: "Add book (error)",
      error: JSON.stringify(errors.array()
    });
```




```
<table ...>
 <tr><td colspan="2">A</td><td rowspan="2">C</td></tr>
 <tr><td>B</td><td>B</td></tr>
</table>
```

```
let response = await fetch(...);
let product = await response.json()
```

```
const express = require('express');
const app = express();
var path = require('path');

const homeRouter = require('./routes/home.routes');
const booksRouter = require('./routes/books.routes');
const authorRouter = require('./routes/author.routes');

app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'ejs');

app.use(express.static(path.join(__dirname, 'public')));

app.use('/', homeRouter);
app.use('/books', booksRouter);
app.use('/author', authorRouter);


app.listen(3000);
```

```
<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>
```