

LV2 - Node.js

Web programiranje

2024./2025.

1 Cilj

Cilj ove laboratorijske vježbe je upoznati se s razvojem i korištenjem aplikacija na strani poslužitelja (eng. *server-side*). U nastavku se nalazi kratak teorijski uvod, nakon kojeg su navedeni zadaci koje je potrebno riješiti.

Tijekom ove laboratorijske vježbe će se naučiti:

1. Učitati stranicu kreiranu na LV1 sa serverske strane ([Express](#), [Node.js](#)) - upute za rješavanje zadatka na str. [19](#)
2. Objaviti stranicu na servisu [Railway](#) - upute za rješavanje zadatka na str. [21](#)
3. Dinamički generirati galeriju slika pomoću template-a ([EJS](#) - Embedded JavaScript templating) - upute za rješavanje zadatka na str. [23](#)
4. (dodatno) Naučiti koristiti vanjske datoteke csv ili json za filtriranje sa serverske strane. (express, node.js, csv-parser, ejs)

Izradom [Node.js](#) aplikacije naučit će te na koji način se može manipulirati stranicom i podacima isključivo na serveru.

Dodane dijelove koda dokumentirati (kratko prokomentirati) zbog lakšeg snalaženja u dokumentima.

Nakon što završite dizajn web stranice, **potrebno** je kôd postaviti na dani **github repozitorij** i **objaviti svoju stranicu**. Link stranice postaviti u README.md dokument na za to predviđeno mjesto.

Stranicu će te objaviti pomoću Railway-a, a upute možete pronaći na [LINKU1](#), ali i u nastavku dokumenta.

2 Osnovne Node.js naredbe

Node.js je JavaScript okruženje koje omogućuje izvršavanje JavaScript koda izvan web preglednika, najčešće na poslužitelju. Za rad s Node.js projektima koriste se različite naredbe u terminalu (CLI – Command Line Interface), a najvažnije su vezane uz pokretanje skripti, upravljanje paketima te instalaciju ovisnosti pomoću alata **npm**.

2.1 Pokretanje Node.js skripte

Prvo je potrebno provjeriti je li `Node.js` instaliran na računalu. To se radi u terminalu sljedećom naredbom:

```
node -v
```

Ako je `Node.js` ispravno instaliran, terminal će prikazati broj verzije (npr. `v18.12.1`). U suprotnom, potrebno je [instalirati Node.js](#).

Za pokretanje `Node.js` datoteke (npr. `server.js`), koristi se naredba:

```
node server.js
```

Ova naredba izvršava JavaScript kod unutar datoteke pomoću `Node.js` okruženja.

`npm` (engl. Node Package Manager) je službeni upravitelj paketa za `Node.js`. Omogućuje instalaciju i dijeljenje paketa (biblioteka, modula ili alata) koje drugi programeri mogu koristiti u svojim projektima.

Najčešće korištene naredbe:

- `npm install` – instalira sve pakete navedene u `package.json`
- `npm install <ime-paketa>` – instalira konkretan paket (npr. `express`)
- `npm start` – pokreće aplikaciju ako je definirana `start` skripta
- `npm publish` – objavljuje vlastiti paket na `npm` repozitorij

`npm` koristi datoteku `package.json` za pohranu informacija o projektu i popisu paketa koji su mu potrebni za rad.

2.2 Inicijalizacija projekta (package.json)

```
npm init -y
```

Ova naredba kreira datoteku `package.json` sa zadanim postavkama. Više na stranici: [11](#).

2.3 Instalacija Node modula

Instalacija modula s `npm`:

```
npm install express ejs csv-parser
```

Instalirani moduli spremaju se u direktorij `node_modules` i bilježe u `package.json`.

2.4 Automatsko osvježavanje aplikacije

Koristi se modul `nodemon`:

```
npm install --save-dev nodemon  
npx nodemon server.js
```

Automatski ponovno pokreće server nakon promjena u kodu.

2.5 Uklanjanje modula

Deinstalacija nepotrebnih modula:

```
npm uninstall ejs
```

3 Korisni ugrađeni Node.js moduli

3.1 Modul fs - rad s datotekama

Node.js modul za rad s datotekama u projektu je **File System** (fs) modul. Za uključivanje File System modula koristi se `require()` metoda. [Više o fs modulu](#)

```
var fs = require('fs');
```

Uobičajeno korištenje modula File System su:

- čitanje datoteke (`fs.readFile()`),
- stvaranje nove datoteke (`fs.appendFile()`, `fs.open()`, `fs.writeFile()`),
- ažuriranje datoteka (`fs.appendFile()`, `fs.writeFile()`),
- brisanje datoteka (`fs.unlink()`),
- preimenovanje datoteka (`fs.rename()`).

Prvi redak u primjeru [3.1](#) koda označava učitavanje fs modula (File System) koji omogućuje čitanje i pisanje datoteka u Node.js-u.

Primjer 1: Korištenje Node.js modula za čitanje sadržaja datoteka

```
const fs = require('fs');

fs.readFile('primjer.txt', 'utf8', (err, data) => {
  if (err) throw err;
  console.log('Sadržaj datoteke:', data);
});
```

3.2 Modul path

Olakšava rad s putanjama do datoteka. [Više o path modulu](#)

```
const path = require('path');

const putanja = path.join(__dirname, 'public', 'index.html');
console.log(putanja);
```

Prvi redak gornjeg dijela koda označava učitavanje path modula koji služi za rad s putanjama do datoteka i mapa na različitim operativnim sustavima.

Drugi redak naredbe označava kreiranje apsolutne putanje do datoteke `index.html`.

- `__dirname` - mapa u kojem se nalazi `server.js`, trenutna mapa.
- `path.join(...)` - spaja više dijelova putanje u ispravnu, sigurnu apsolutnu putanju, automatski dodajući odgovarajuće `/` ili ovisno o operativnom sustavu.
- `console.log()` - ispisivanje poruka u terminalu, kada se pokrene dokument `server.js`.

3.3 Modul http

Kreiranje osnovnog HTTP servera. [Više o http modulu](#)

```
const http = require('http');

const server = http.createServer((req, res) => {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('Hello World!');
});

server.listen(3000, () => {
  console.log('Server pokrenut na http://localhost:3000');
});
```

3.4 Varijable i tipovi podataka

JavaScript ima nekoliko osnovnih tipova podataka:

- **String** - tekstualni podatak, npr. "Ivana"
- **Number** - broječni podatak, npr. 25
- **Boolean** - logički podatak (true ili false)
- **Array** - niz podataka, npr. [1, 2, 3]
- **Object** - objekt koji sadrži parove ključ-vrijednost, npr. { ime: "Ivana", godine: 25 }

Primjer deklariranja varijabli:

```
let ime = "Ivana"; // String
let godine = 25; // Number
let student = true; // Boolean
let ocjene = [5, 4, 3]; // Array
let osoba = { // Object
  ime: "Ivana",
  godine: 25
};

console.log(ime, godine, student, ocjene, osoba);
```

Varijable i funkcije:

```
1 let ime = "Ivana";
2 let godine = 25;
3
4 function pozdrav(ime, godine) {
5   console.log('Pozdrav, ${ime}, imas ${godine} godina. ');
6 }
7
8 pozdrav(ime, godine);
```

Petlje i nizovi:

U sljedećem primjeru brojevi iz polja `brojevi` će se prikazivati redom.

```
1 const brojevi = [10, 20, 30];
2
3 for(let broj of brojevi){
4   console.log(broj);
5 }
```

4 Uvod u Node.js & Express

Node.js omogućava izvršavanje JavaScripta na serverskoj strani. Bitno je osnovno znanje JavaScripta.

4.1 Primjer jednostavnog Node.js programa

Datoteka `hello.js`:

```
console.log(`Pozdrav iz Node.js-a!`);
```

Pokretanje programa u terminalu:

```
node hello.js
```

4.2 Jednostavni Node.js web server

Korištenje Node.js-a za kreiranje jednostavnog HTTP servera:

```
1 const http = require('http');
2
3 http.createServer((req, res) => {
4   res.writeHead(200, {'Content-Type': 'text/plain'});
5   res.end("Pozdrav sa Node.js servera!");
6 }).listen(3000);
7 console.log("Server je pokrenut na http://localhost:3000");
```

Objašnjenje gornjeg koda:

- `const http = require('http');`
Učitava ugrađeni Node.js modul za rad s HTTP zahtjevima i odgovorima.
`require('http')` dohvaća funkcionalnost HTTP modula.
`const http` varijabla `http` čija se vrijednost koristi dalje u kodu.
- `http.createServer((req, res) => { ... })`
Stvara novi HTTP server koji reagira na dolazne zahtjeve.
`createServer()` prihvaća funkciju s dva parametra:
 - `req` – predstavlja dolazni zahtjev korisnika (request)
 - `res` – predstavlja odgovor koji ćemo poslati korisniku (response).
- `res.writeHead(200, { 'Content-Type': 'text/plain' });`
Postavlja HTTP zaglavlje: status 200 (OK) i tip sadržaja `text/plain` - znači da ćemo korisniku poslati običan tekst (nije HTML, JSON itd.).

- `res.end(...);`
Zatvara odgovor i šalje sadržaj korisniku.
- `.listen(3000);`
Ova metoda pokreće server i govori mu da „sluša“ zahtjeve na portu 3000.
Kada posjetimo `http://localhost:3000`, server će reagirati i poslati gore definirani odgovor.
- `console.log(...);`
Ispisuje poruku u terminal kako bi korisnik znao da je server pokrenut.

4.3 Funkcionalnosti HTTP modula

HTTP modul u Node.js-u omogućuje rad s web zahtjevima i izradu vlastitog servera bez dodatnih biblioteka.

Funkcija / Klasa	Opis
<code>http.createServer()</code>	Kreira novi HTTP server koji može primiti zahtjeve i slati odgovore.
<code>server.listen(port)</code>	Pokreće server na zadanom portu (npr. 3000).
<code>req</code> (request objekt)	Informacije o zahtjevu: URL, metoda (GET, POST...), zaglavlja itd.
<code>res</code> (response objekt)	Način na koji šaljemo odgovor korisniku (status, zaglavlja, sadržaj).
<code>res.writeHead()</code>	Postavljanje statusnog koda i Content-Type zaglavlja.
<code>res.end()</code>	Završava odgovor i šalje sadržaj korisniku.

Zašto je potreban `require('http')`?

Node.js koristi modularni pristup – funkcionalnosti nisu dostupne automatski, već ih treba eksplicitno uključiti. `http` je ugrađeni modul koji nije automatski dostupan:

```
const http = require('http');
```

Bez toga, funkcije poput `createServer()` nisu poznate i kod će generirati grešku

```
ReferenceError: http is not defined
```

4.4 Riješeni primjer: Hello World

1. Kreirajte novu datoteku naziva `hello.js`.
2. U datoteku upišite sljedeći JavaScript kod:

```
console.log(\"Hello, JavaScript preko Node.js-a!\");
```

3. Otvorite terminal u VS Code-u (Ctrl + ‘ ili View → Terminal).
4. Pokrenite program upisivanjem sljedeće naredbe u terminal:

```
node hello.js
```

5. Provjerite ispis u terminalu. Trebali biste vidjeti:

```
Hello, JavaScript preko Node.js-a!
```

4.5 Express

Express je fleksibilan poslužiteljski okvir za **Node.js**. Omogućuje jednostavno i brzo razvijanje poslužiteljskih (server-side) aplikacija i RESTful¹ API-ja. Njegov osnovni cilj je pojednostaviti rad s HTTP zahtjevima i odgovorima te pružiti intuitivnu strukturu za izgradnju skalabilnih web aplikacija.

Express zbog svoje jednostavnosti i modularnosti, predstavlja jedan od najvažnijih alata za razvoj web aplikacija u Node.js okruženju.

Express djeluje kao "sloj" između osnovnog **Node.js http** modula i stvarne aplikacijske logike. Bez Expressa, rad s HTTP zahtjevima u Node.js-u zahtijevao bi pisanje mnogo koda za upravljanje rutama, zaglavljima, statusima, odgovorima i sl. Express sve to pojednostavljuje pomoću funkcija poput `app.get()`, `app.post()`, `res.send()`, `res.json()` i drugih.

Zašto koristiti Express?

- Pruža jasan način definiranja ruta za GET, POST, PUT i DELETE zahtjeve
- Omogućuje korištenje *middleware* funkcija za obradu zahtjeva
- Lako se integrira s obrascima (šablonama, engl. templates) poput EJS, Pug ili Handlebars
- Podržava posluživanje statičkih datoteka (slike, CSS, JavaScript)
- Jednostavan za učenje, ali dovoljno moćan za profesionalnu upotrebu

Express je srž mnogih poznatih Node.js frameworka i aplikacija, uključujući ME(R)N i MEAN stack (MongoDB, Express, React/Angular, Node.js). Koristi se i u razvoju mobilnih backend servisa, API-ja za frontend aplikacije, upravljanje korisnicima, autentifikaciju, web trgovine i slično.

Instalacija i osnovna struktura aplikacije

Express se instalira putem npm:

```
npm install express
```

Osnovna Express aplikacija može izgledati na sljedeći način:

Primjer 2: Primjer koda jednostavnog Express servera

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

app.get('/', (req, res) => {
  res.send("Pozdrav iz Expressa!");
});
```

¹REST = engl. Representational State Transfer

```
app.listen(3000, () => {
    console.log("Server je pokrenut na http://localhost:3000");
});
```

Ovaj primjer prikazuje kako definirati najjednostavniji web poslužitelj koji odgovara na HTTP GET zahtjev na početnoj ruti /.

Kada koristiti Express?

- Kada se želi razviti složeniju aplikaciju s više ruta
- Kada se želi brže prototipirati API ili web aplikaciju
- Kada je potrebna organizacija projekta kroz middleware i module

Napomena: http modul je koristan za učenje osnova, dok je Express idealan za profesionalni razvoj i ozbiljnije projekte.

4.6 Definiranje ruta u Expressu

Ruta predstavlja putanju unutar web aplikacije koja odgovara na određene HTTP zahtjeve, poput GET, POST, PUT ili DELETE. U Expressu, rute se definiraju pomoću metoda `app.get()`, `app.post()` itd., gdje prvi parametar označava URL, a drugi parametar je funkcija koja opisuje ponašanje pri dolasku na tu rutu.

Primjer 3: Osnovna GET ruta u Expressu

```
app.get('/', (req, res) => {
    res.send('Pocetna stranica');
});
```

Parametri URL-a mogu se dohvaćati pomoću `req.params`, dok se podaci iz tijela zahtjeva pristupaju pomoću `req.body` (uz korištenje `express.urlencoded()` middlewarea).

Primjer 4: Ruta s parametrom

```
app.get('/korisnik/:id', (req, res) => {
    res.send('Korisnicki ID: ${req.params.id}');
});
```

4.7 Middleware u Expressu

Middleware je funkcija koja ima pristup objektima zahtjeva (`req`) i odgovora (`res`), te sljedećoj funkciji u nizu (`next()`). Koristi se za obradu podataka, autentifikaciju, zapisivanje logova, rukovanje greškama i drugo.

Primjer 5: Jednostavni middleware primjer

```
app.use((req, res, next) => {
    console.log('Zahtjev na: ${req.url}');
    next(); // prosljedjuje na sljedecu funkciju
});
```

Middleware funkcije se mogu globalno primijeniti (na sve rute) ili lokalno, samo na pojedine rute.

4.8 Posluživanje statičkih datoteka

Express omogućuje jednostavno posluživanje statičkih datoteka poput HTML, CSS, slika i JavaScript dokumenata pomoću `express.static()` funkcije.

Datoteke se obično nalaze u mapi `public`, a primjer uključivanja izgleda ovako:

Primjer 6: Posluživanje statičkih datoteka

```
const path = require('path');
app.use(express.static(path.join(__dirname, 'public')));
```

Nakon toga, sve datoteke iz mape `public` bit će dostupne izravno putem URL-a. Na primjer, ako u `public` mapi postoji datoteka `style.css`, korisnik je može dohvatiti preko:

```
http://localhost:3000/style.css
```

4.9 Usporedba HTTP modula i Expressa

Express.js je popularan framework za Node.js koji pojednostavljuje izradu web aplikacija. Umjesto da ručno radimo sa svakim zahtjevom kao s `http` modulom, Express nudi jednostavniju i čitljiviju sintaksu.

Node.js HTTP modul	Express.js
<code>http.createServer(...)</code>	<code>const app = express();</code>
Ručna obrada URL-a i metoda	Jednostavne metode: <code>app.get()</code> , <code>app.post()</code>
Ručno postavljanje zaglavlja i slanje odgovora	Automatska obrada i metode poput <code>res.send()</code>
Nema podrške za middleware funkcije	Podrška za middleware, autentifikaciju, logiranje itd.
Ugrađeni modul (nije potrebna instalacija)	Potrebno instalirati s <code>npm install express</code>

4.10 Posluživanje index.html datoteke sa serverske strane

Ako želimo da `Express.js` server korisniku prikaže HTML datoteku (npr. `index.html`), to možemo napraviti na dva najčešća načina:

- korištenjem metode `res.sendFile()` unutar definirane route
- korištenjem funkcije `express.static()` koja automatski poslužuje sve datoteke iz mape

1. način: `res.sendFile()`

Primjer 7: Korištenje `res.sendFile`

```
const path = require('path');

app.get('/', (req, res) => {
  res.sendFile(path.join(__dirname, 'public', 'index.html'));
});
```

Ovaj način koristi se kada želimo precizno specificirati HTML datoteku koju šaljemo klijentu.

2. način: `express.static()`

Primjer 8: Korištenje `express.static`

```
const path = require('path');

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

Ako se u mapi `public` nalazi datoteka `index.html`, ona će se automatski prikazati pri pristupu ruti `/`.

Može se koristiti dva načina za definiranje statičke mape u Express-u:

1. relativna putanja: `express.static('public')`
2. apsolutna putanja: `express.static(path.join(__dirname, 'public'))`

Karakteristika	Relativna putanja	Apsolutna putanja
Značenje	"gleda" relativno na trenutnu radnu mapu (gdje se pokreća aplikacija)	Apsolutna putanja do mape bez obzira na mjesto pokretanja
Ovisnost o pokretanju	prestaje raditi ako se pokrene iz druge mape	Ne ovisi o tome odakle se pokreće aplikacija
Preporučeno za deploy (npr. Railway)	Nije preporučeno	Da, stabilno i sigurno
Jednostavnost pisanja	Kraće za pisanje	Duže, ali pouzdanije
Primjeri postojanje problema	Pokretanje s druge lokacije, testiranje	Gotovo nikada

Table 1: Usporedba načina definiranja statičke mape u Express aplikaciji

Usporedba pristupa

Pristup	<code>res.sendFile()</code>	<code>express.static()</code>
Kontrola prikaza	Ručno određuje koja se HTML stranica prikazuje	Automatski posluživanje svih statičkih datoteka
Pogodno za	Jedan HTML dokument, specifične rute	Veći broj HTML/CSS/JS dokumenata
Potrebna ruta	Da, koristi <code>app.get()</code>	Ne, radi automatski ako postoji <code>index.html</code>
Preporuka za veće aplikacije	Ne	Da

Table 2: Usporedba načina posluživanja `index.html` datoteke u Express.js aplikaciji

5 Node.js & Express: package.json, package-lock.json

`package.json` je jedna od najvažnijih datoteka u svakom Node.js projektu jer:

- opisuje projekt (ime, verzija, opis, autor),
- popisuje sve instalirane pakete i ovisnosti (engl. *dependencies*),
- sadrži korisne skripte za pokretanje, razvoj i testiranje,
- omogućuje jednostavno dijeljenje projekta s drugima.

5.1 Instalacija paketa pomoću npm

Kada u Node.js projektu postoji datoteka `package.json`, ona sadrži popis svih potrebnih paketa (npr. `express`, `ejs`, itd.) koje projekt koristi.

To znači da bilo tko tko preuzme ovaj projekt, može jednostavno pokrenuti sljedeću naredbu u terminalu kako bi instalirao sve ovisnosti:

```
npm install
```

`package.json` je bitan jer:

1. Pamti koji se paketi koriste

Kao na primjer, kada instaliramo paket Express:

```
npm install express
```

U `package.json` se automatski dodaje:

```
"dependencies": {  
  "express": "^4.18.2"  
}
```

2. Sadrži informacije o projektu

Primjer osnovnog `package.json`:

```
{  
  "name": "moj-projekt",  
  "version": "1.0.0",  
  "description": "Testni Node.js projekt",  
  "main": "server.js",  
  "scripts": {  
    "start": "node server.js"  
  },  
  "dependencies": {  
    "express": "^4.18.2"  
  }  
}
```

3. Definira korisne skripte

Primjer skripti:

```
"scripts": {
  "start": "node server.js",
  "dev": "nodemon server.js"
}
```

Sada se u terminalu može upisati sljedeća linija, umjesto pune naredbe `node server.js`:

```
npm start
```

4. Osnova za objavu paketa na npm

Ako se planira objaviti biblioteka na npm (npr. za dijeljenje s drugima), `package.json` je obavezan.

5.2 Kako kreirati `package.json`?

- Interaktivno:

```
npm init
```

- Automatski sa zadanim vrijednostima:

```
npm init -y
```

5.3 Što je `package-lock.json`?

`package-lock.json` je datoteka koju npm automatski generira kada se:

- instalira novi paket,
- prvi put koristi `npm init -y` i dodaje ovisnosti.

5.4 Zašto je `package-lock.json` važan?

1. Zaključava točne verzije paketa i podpaketa
2. Omogućuje svima koji rade na projektu da imaju identičan `node_modules`
3. Ubrzava naredbu `npm install`

Primjer 9: Primjer sadržaja `package-lock.json`

```
"express": {
  "version": "4.18.2",
  "resolved": "https://registry.npmjs.org/express/-/express-4.18.2.
  tgz"
}
```

5.5 Usporedba `package.json` i `package-lock.json`

Značajka	<code>package.json</code>	<code>package-lock.json</code>
Ručno uređivanje	Da	Ne (automatski)
Opis projekta	Da	Ne
Popis glavnih paketa	Da	Da + sve pod-ovisnosti
Zaključava verzije	Ne	Da
Dijeli se u timu (Git)	Da	Da

Napomena

Uvijek zadržati `package-lock.json` u projektu (i Git repozitoriju). On osigurava da svi koriste iste verzije paketa i sprječava greške poput "radilo je kod mene".

6 Embedded JavaScript templating - EJS

EJS (engl. *Embedded JavaScript*) predstavlja jezik za izradu predložaka u JavaScriptu koji omogućuje da se JavaScript kod izravno ugrađuje u HTML. Time se omogućuje generiranje dinamičkog HTML sadržaja na strani poslužitelja. U kombinaciji s Express.js okvirom, EJS omogućuje da se HTML prikazuje na temelju podataka koji se prosljeđuju iz pozadinske logike aplikacije.

Uobičajeno se koristi u Node.js okruženju za aplikacije koje trebaju na jednostavan način generirati HTML s dinamičkim vrijednostima. Za razliku od klijentskih frameworka poput Reacta ili Vue-a, EJS omogućuje renderiranje na strani poslužitelja, čime se eliminira potreba za dodatnim build alatima i složenom konfiguracijom.

EJS predlošci se definiraju kao obične .ejs datoteke koje po strukturi podsjećaju na HTML, ali uključuju posebnu sintaksu za ispis vrijednosti, uvjetne izraze i petlje.

Elementi kao što su `<%= ... %>` koriste se za ispis vrijednosti varijabli, dok se `<% ... %>` koristi za izvršavanje JavaScript logike poput uvjeta i petlji.

Jedna od glavnih prednosti EJS-a jest ta što se omogućuje jasno razdvajanje prikaznog sloja od poslovne logike aplikacije. Prikaz se definira unutar EJS predložaka, dok se logika obrade podataka odvija u Express rutama.

Glavne prednosti EJS-a

1. **Dinamički HTML prikaz** — omogućuje prikaz varijabli, petlji, uvjeta unutar HTML-a.
2. **Prirodno se integrira s Expressom** — radi odmah bez dodatne konfiguracije.
3. **Jasno odvađa prikaz i logiku** — logika ostaje u Express ruti, dok se prikaz generira kroz EJS.
4. **Podrška za uvjete i petlje** — koristi se `if`, `else`, `forEach`, itd.
5. **Jednostavan za početnike** — lakši je za učenje od Reacta/Vue-a jer ne zahtijeva build alate.
6. **Omogućuje ponovno korištenje dijelova stranice** — uključivanjem (`<%- include('header' %>`).

Integracija s Express okvirom ostvaruje se jednostavno, bez dodatne konfiguracije. Dovoljno je postaviti EJS kao predložak pomoću naredbe

```
app.set('view_engine', 'ejs')
```

nakon čega se mogu koristiti funkcije za prikaz stranica, poput

```
res.render()
```

U nastavku se prikazuje primjer korištenja EJS-a kroz prikaz osobnog pozdrava. U EJS datoteci `pozdrav.ejs`, ispisuje se ime osobe prosljeđeno iz Express rute:

```
<h1>Pozdrav, <%= ime %>!/</h1>
```

U odgovarajućoj Express ruti koristi se funkcija `res.render()` koja predložku prosljeđuje vrijednost varijable:

```
res.render('pozdrav', { ime: 'Ivana' });
```

Kao rezultat, generira se HTML dokument u kojem se umjesto varijable prikazuje konkretna vrijednost:

```
<h1>Pozdrav, Ivana!</h1>
```

Osim ispisa pojedinačnih vrijednosti, EJS podržava rad s nizovima podataka, uvjetima i petljama.

Primjer prikaza liste korisnika prikazan je kroz petlju u EJS predlošku:

```
<ul>
  <% lista.forEach(korisnik => { %>
    <li><%= korisnik %></li>
  <% }) %>
</ul>
```

U Express rutu podaci se proslijeđuju kao polje stringova:

```
res.render('korisnici', { lista: ['Ana', 'Ivan', 'Petra'] });
```

Rezultantni HTML sadrži nerealiziranu listu korisnika, generiranu dinamički na temelju prosljeđenih podataka:

```
<ul>
<li>Ana</li>
<li>Ivan</li>
<li>Petra</li>
</ul>
```

EJS se preporučuje koristiti u slučajevima kada je potrebno jednostavno i brzo generirati HTML dokumente na poslužiteljskoj strani. Takvi slučajevi uključuju rad s podacima iz baze, prikaz tablica, kreiranje administracijskih sučelja, formulara ili izvještaja, bez potrebe za potpunim frontend frameworkom.

Iako EJS pruža brojne prednosti u razvoju server-side aplikacija, postoje situacije kada njegovo korištenje nije optimalno. U aplikacijama koje se temelje na bogatoj interakciji na strani klijenta, kao što su SPA (Single Page Application) rješenja razvijena u Reactu, Angularu ili Vue.js-u, EJS se ne preporučuje.

Zaključno, EJS se ističe svojom jednostavnošću, brzinom implementacije te mogućnošću direktnog povezivanja s Express aplikacijama. Kao takav, idealan je za edukacijske projekte, administracijska sučelja i sve aplikacije gdje je server zadužen za pripremu HTML sadržaja.

6.1 Riješeni primjer: To-do lista

Kreirat ćemo to-do listu pomoću node.js, express.js i ejs.

Prvo stvorimo mapu EJS u koju ćemo spremiti sve potrebne dokumente.

U terminalu upisati naredbu:

```
npm init -y
```

koja će stvoriti dokument `package.json`.

Zatim:

```
npm install express ejs
```

kojom ćemo instalirati `express` i `ejs`. Tim naredbama će se u radnoj mapi stvoriti mapa `node_modules` i `package-lock.json`.

Uz to trebamo kreirati mapu `views` gdje će biti sadržani svi `ejs` predlošci. U ovom primjeru ćemo imati samo `index.ejs` predložak.

Također, potreban nam je i server `app.js` dokument u kojem ćemo pozvati sve potrebne metode i funkcije.

Struktura projekta izgleda na sljedeći način:

```
EJS/  
├─ node_modules/  
├─ views/  
│   └─ index.ejs  
├─ app.js  
├─ package.json  
└─ package-lock.json
```

Primjer 10: Express server s podrškom za EJS i dodavanje zadataka

```
1  const express = require('express');  
2  const app = express();  
3  const PORT = process.env.PORT || 3000;  
4  
5  app.set('view engine', 'ejs'); // sets the view engine to ejs, for rendering  
6  //app.use(express.static('project'));  
7  
8  app.use(express.urlencoded({extended: true}))  
9  
10 let tasks = [];  
11  
12 app.get('/', (req, res) => {  
13   //res.render('gallery', { numImages });  
14   res.render('index', {tasks: tasks});  
15 });  
16  
17 app.post('/add', (req, res) => {  
18   const newTask = req.body.task;  
19   tasks.push(newTask);  
20   res.redirect('/');  
21  
22  
23 })  
24 app.listen(PORT, () => {  
25   console.log('Server radi na http://localhost:${PORT}');  
26 });
```

index.ejs je predložak HTML stranice koji se koristi za generiranje sadržaja na poslužiteljskoj strani pomoću EJS-a. Express na serveru učitava `index.ejs`, "ubaci" u nju podatke (npr. `tasks`) i generira HTML koji pošalje korisniku.

To čini sljedećom naredbom:

```
res.render('index', { tasks });
```

a znači:

- uzmi `views/index.ejs`
- ubaci u njega podatke (varijablu `tasks`)

- i pošalji gotov HTML klijentu

Primjer 11: Primjer EJS generiranja HTML-a

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <title>To-Do Lista</title>
5 </head>
6 <body>
7
8 <h1>Lista zadataka</h1>
9
10 <form action="/add" method="POST">
11 <input type="text" name="task" required>
12 <button type="submit">Dodaj zadatak</button>
13 </form>
14
15 <ul>
16 <% tasks.forEach(task => { %>
17   <li><%= task %></li>
18   <% }); %>
19 </ul>
20
21 </body>
22 </html>
23
```

Nakon nekoliko upisanih stavki html stranica izgleda kao na slici 6.1. Stranica se svaki put iznova učita (server generira novi HTML), ali lista zadataka se nadograđuje jer podaci ostaju u memoriji. Kada se server isključi podaci se brišu, zato je bitno spremiti upisane podatke u dokument, na primjer u .json.

Lista zadataka

Dodaj zadatak

- marko
- josip
- šime
- napraviti šumu

Figure 1: Html primjer

7 Korisni linkovi prilikom rješavanja zadatka

Uz gore navedene linkove, slijede neki od korisnik linkova koji mogu pomoći pri izradi rješenja zadatka.

- JavaScript dokumentacija: <https://developer.mozilla.org/en-US/docs/Web/JavaScript>
- Node.js službena stranica: <https://nodejs.org>
- [Introduction to Node.js](#)
- [Express.js](#) - Web framework za Node.js
- [EJS](#) - Templating engine za generiranje HTML-a
- [csv-parser](#) - Modul za rad s CSV datotekama
- [nodemon](#) - Alat za automatsko osvježavanje aplikacije
- Node.js, Express, MongoDB [tutorial](#)
- CodePen za online isprobavanje HTML/CSS/JavaScript: <https://codepen.io>

Zadatak 1

Pokrenuti statičku web stranicu (`index.html`) sa serverske strane pomoću Express.js. Ovaj zadatak je potrebno riješiti na temelju kreirane web stranice na 1. laboratorijskoj vježbi. Sve potrebne dokumente postaviti u repozitorij.

1. Struktura projekta

Projekt treba imati sljedeću strukturu:

```
projekt/  
├── public/  
│   └── index.html  
├── server.js  
└── package.json
```

Figure 2: Primjer strukture projekta

2. Sadržaj `index.html`

Na primjer datoteka `public/index.html` može izgledati ovako:

```
<!DOCTYPE html>  
<html lang="hr">  
<head>  
<meta charset="UTF-8">  
<title>Pozdrav</title>  
</head>  
<body>  
<h1>Ovo je HTML datoteka pokrenuta sa serverske strane</h1>  
</body>  
</html>
```

3. Sadržaj `server.js`

Datoteka `server.js` koja koristi Express za posluživanje statičkog sadržaja:

```
const express = require('express');  
const app = express();  
  
app.use(express.static('public')); // "posluzuje" index.html  
// Automatski koristi sve iz mape public  
app.get('/', (req, res) => {  
    res.send("Ili obican tekst ako nema HTML datoteke.");  
});  
  
app.listen(3000, () => {  
    console.log("Server pokrenut na http://localhost:3000");  
});
```

4. Instalacija Expressa i pokretanje servera

Ako još nemate kreiran `package.json`, prvo u terminalu pokreni:

```
npm init -y
npm install express
```

Morate se nalaziti u mapi: `projekt`. Nakon toga je potrebno pokrenuti server:

```
node server.js
```

5. Prikaz u pregledniku

Nakon pokretanja servera, otvoriti preglednik i ići na stranicu:

`http://localhost:3000`

Ako je sve ispravno postavljeno, prikazat će se sadržaj stranice `index.html`. U suprotnom, bit će prikazan tekst

Ili obican tekst ako nema HTML datoteke.

Napomene

- Datoteka mora biti točno nazvana `index.html` (malim slovima).
- Mora se nalaziti unutar mape `public`, u suprotnom navesti u `express.json` dokumentu naziv mape.
- Express automatski traži `index.html` unutar mape definirane u `express.static(...)`.

Više informacija

- Dokumentacija za Express static: <https://expressjs.com/en/starter/static-files.html>
- Node.js dokumentacija: <https://nodejs.org/en/docs/>

Zadatak 2. - Node.js, Express i Railway

Glavni zadatak je objaviti web stranicu sa serverske strane pomoću Railway-a. Ovaj zadatak je potrebno riješiti na temelju kreirane web stranice na prvoj laboratorijskoj vježbi i prethodnog zadatka (zadatak 1., na stranici [19](#)). Kreirani link u Railway-u postaviti u readme dokument na github-u na zato predviđeno mjesto.

Railway je moderna platforma u oblaku koja omogućuje jednostavan deploy back-end aplikacija poput Express servera. U nastavku predložka su opisani koraci kako pripremiti i postaviti Express aplikaciju na Railway korištenjem GitHub repozitorija. Railway omogućuje brz i jednostavan deploy Node.js aplikacija bez puno konfiguracije. Odličan je izbor za testne projekte i prototipe.

1. Priprema Express aplikacije

Datoteka server.js

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

const PORT = process.env.PORT || 3000;

app.get('/', (req, res) => {
  res.send('Pozdrav sa Railway servera!');
});

app.listen(PORT, () => {
  console.log('Server pokrenut na portu ${PORT}');
});
```

Datoteka package.json

Dodaj "start" skriptu koja će se koristiti za pokretanje aplikacije na Railwayu:

```
{
  "name": "express-railway",
  "version": "1.0.0",
  "main": "server.js",
  "scripts": {
    "start": "node server.js"
  },
  "dependencies": {
    "express": "^4.18.2"
  }
}
```

2. Upload projekta na GitHub

1. Inicijaliziraj git repozitorij:

```
git init
git add .
git commit -m "Inicijalni commit"
```

2. Kreiraj novi repozitorij na GitHubu.
3. Poveži lokalni projekt s GitHub repozitorijem:

```
git remote add origin https://github.com/korisnickoime/ime-  
repozitorija.git  
git push -u origin main
```

3. Deploy na Railway

1. Otvori <https://railway.app>
2. Prijavi se s GitHub računom
3. Klikni "New Project" → "Deploy from GitHub repo"
4. Odaberi repozitorij s Express aplikacijom
5. Railway će automatski:
 - detektirati "start" skriptu
 - instalirati ovisnosti
 - postaviti aplikaciju
6. Nakon nekoliko sekundi dobit ćeš URL (npr. <https://myapp.up.railway.app>)

4. Napomene

- Port `process.env.PORT` je obavezan jer Railway dinamički dodjeljuje port.
- Ako koristiš `public/index.html`, dodaj:

```
app.use(express.static('public'));
```

- Railway automatski redeploya kada primjenite push na GitHubu.

Zadatak 3

Dinamički generiranje galerije slika pomoću predloška (ejs - Embedded JavaScript templating)

Iako su objašnjene osnovne mogućnosti ejs, na stranici 14 i dan je riješen primjer 6.1, u nastavku će biti objašnjeni koraci i dijelovi koda koji su potrebni za rješavanje ovog dijela zadatka. Nakon što implementirate zadatak, objavite ga putem railway-a i link postavite u readme dokument na github-u.

Prijedlog strukture dokumenta unutar projekta dasn je na slici 3.

```
ejs-galerija/  
├── public/  
│   ├── style/  
│   │   └── style.css  
│   └── images/  
│       ├── photo1.jpg  
│       └── photo2.jpg  
├── views/  
│   └── slike.ejs  
├── images.json  
├── server.js  
└── package.json
```

Figure 3: Primjer strukture projekta

Dokument `slike.ejs` mora biti unutar mape `views` jer Express traži EJS datoteke u `views` mapi, osim ako se na serveru (u dokumentu `server.js`) ne navede drugačije.

Paziti da svi statični dokumenti (kao što je `index.html`) budu u mapi `public`. To je iz razlog što Express koristi ovu liniju koda:

```
app.use(express.static('public'));
```

čije je objašnjenje da je sve što se nalazi unutar `public/` mape postaje dostupno putem `/` (kosa crta, slash). Naravno, lokacija se može promijeniti.

1. Lokalne fotografije

Prvo što se treba učiniti u zadatku je spremiti nekoliko fotografija u mapu `images` kako bi ih mogli koristiti u projektu. Za ovaj projekt mogu se koristiti dane fotografije u folderu `Portfolio_slike` na github-u.

Kad se koristi EJS, postojeći HTML (uključujući stilove, strukturu i dizajn) ostaje netaknut, ali mora se paziti na putanje do CSS i JS datoteka – jer Express koristi `public/` mapu za statičke resurse.

Usporedba `index.html` i `index.ejs` Ova linija koda je u `index.html`

```
<link rel="stylesheet" href="style.css">
```

Suprotno, u EJS + Express projektu to poziv u `.ejs` datoteci (npr. `index.ejs`) izgleda ovako:

```
<link rel="stylesheet" href="/style/style.css">
```

To je iz razloga što se CSS dokument nalazi u `public/style/style.css`.

Primjer strukture projekta dan je na Slici 3.

Primjer `index.ejs` koji se nalazi u view mapi za generiranje slika bi izgledala na sljedeći način:

```
<h1>Galerija slika</h1>
<section class="gallery">
  <% images.forEach(image => { %>
    <figure>
      ">
      <figcaption><%= image.title %></figcaption>
    </figure>
  <% }); %>
</section>
```

Prilagoditi vlastiti dokument, paziti na ARIA i SEO značajke.

U dokumentu `server.js` potrebno je naznačiti učitavaju li se fotografije:

1. iz .json datoteke

- kreira se dokument `slike.json`

```
[
  { "url": "/images/photo1.jpg", "title": "Slika 1" },
  { "url": "/images/photo2.jpg", "title": "Slika 2" },
  { "url": "/images/photo3.jpg", "title": "Slika 3" }
]
```

- u dokumentu `server.js` napomenuti da se slike učitavaju iz .json dokumenta

```
const fs = require('fs');
const path = require('path');

app.get('/', (req, res) => {
  const dataPath = path.join(__dirname, 'images.json');
  const images = JSON.parse(fs.readFileSync(dataPath));
  res.render('slike', { images });
});
```

2. direktno iz mape ako želimo da se automatski pronađu slike u `public/images/` mapi, koristi Node `fs` modul:

Primjer 12: Primjer dio koda za `server.js` koji učitava slike s poslužitelja iz mape `/images`

```
1  const fs = require('fs');
2  const path = require('path');
3
4  app.get('/', (req, res) => {
5    res.redirect('/slike');
6  });
7
8  //GALERIJA, ruta prema folderu /slike
```



```

9
10 app.get('/slike', (req, res) => {
11   const folderPath = path.join(__dirname, 'public', 'images');
12   const files = fs.readdirSync(folderPath);
13
14   const images = files
15     .filter(file => file.endsWith('.jpg') || file.endsWith('.svg'))
16     .map((file, index) => ({
17       url: '/images/${file}',
18       id: 'slika${index + 1}',
19       title: 'Slika ${index + 1}'
20     }));
21
22   res.render('slike', { images });
23 });

```

Prilikom definiranja putanja, morate paziti u kojoj se mapi nalazite. Express, odnosno `server.js` dokument pokreće sve iz root mape, a to je u ovom slučaju `public`. Dok, EJS pokreće sve iz mape `view`.

Dio koda	Opis
<code>files</code>	Niz imena svih datoteka iz direktorija (npr. <code>photo1.jpg</code> , <code>file.txt</code> , <code>slika.png</code>)
<code>.filter(file => ...)</code>	Filtrira niz i zadržava samo slike s ekstenzijama <code>.jpg</code> ili <code>.png</code>
<code>.map((file, index) => {...})</code>	Pretvara svaki naziv datoteke u objekt s URL-om i naslovom
<code>url: \images/\${file}</code>	Kreira putanju za sliku (npr. <code>/images/photo1.jpg</code>)
<code>id: slika\${index + 1}`</code>	Generira jedinstveni id za svaku sliku (npr. <code>slika1</code> , <code>slika2</code> , ...) koji se koristi u <code>href</code> i <code>id</code> atributima za lightbox povezivanje. Ovo je ključno da svaka slika ima svoj jedinstveni identifikator koji se može koristiti za otvaranje lightboxa putem <code>#id</code> .
<code>title: Slika \${index + 1}</code>	Dodjeljuje naziv svakoj slici prema rednom broju (npr. <code>Slika 1</code>)

Table 3: Objašnjenje dijela koda za pripremu slika

Kako bi stilovi definirani u CSS dokumentima bili primijenjeni na `.ejs` datoteke, važno je pravilno postaviti putanju do stilskih datoteka u zaglavlju HTML-a. Budući da se `.ejs` predlošci nalaze unutar mape `views`, a statički sadržaji poput CSS-a nalaze se u mapi `public`, Express aplikacija mora imati definiranu početnu (root) statičku mapu.

U `server.js` datoteci to se postiže naredbom:

```
app.use(express.static('public'));
```

Ova naredba govori Express poslužitelju da sve datoteke unutar mape `public` budu dostupne klijentu putem URL-a, pri čemu se `public` izostavlja iz putanje.

Na primjer, ako se stilska datoteka nalazi na putanji `public/src/styles/slike_v2.css`, tada se u `.ejs` datoteci stil uključuje ovako:

```
<link rel="stylesheet" href="/src/styles/slike_v2.css">
```

Važno je ne navoditi `../public` u putanji jer Express automatski mapira `public` kao korijen za sve statičke resurse. Korištenje neispravne putanje poput:

```
<link rel="stylesheet" href="../../public/src/styles/slike_v2.css">

<!-- Pogresno -->
```

rezultirat će pogreškom prilikom učitavanja datoteke (npr. 404 Not Found).

Time se osigurava da svi elementi stranice, poput izgleda galerije, lightbox efekata i responzivnog dizajna, budu ispravno prikazani kada se stranica učita.

Više informacija dostupno je u dokumentaciji Express okvira:

<https://expressjs.com/en/starter/static-files.html>

Ovako definirani dokumente, pokrećete server s naredbom:

```
node server.js
```

a u pregledniku možete provjeriti radili sve kako treba:

```
http://localhost:3000/slike
```

Ovaj gornji dio možete preskočiti, ako u `index.html` stavite poveznicu na file `slike` upisujući samo

```
<a href="slike">Galerija</a></li>
```

U tom slučaju šalje se zahtjev prema ruti `/slike` koja se definira u `server.js` kao server-side stranicu.

Povezivanje s `index.html`

Početna stranica aplikacije je stranica koja se prikazuje kada korisnik posjeti glavnu (korijensku) adresu web aplikacije, odnosno rutu `/` (npr. `http://localhost:3000/`).

Zadatak je povezati ovu početnu rutu s datotekom `index.html`, tako da se korisniku odmah po pokretanju poslužitelja prikaže sadržaj početne stranice.

Istovremeno, slike za galeriju trebaju se generirati dinamički na poslužiteljskoj strani, putem posebne Express rute (npr. `/slike`) i EJS predloška.

Važno je ispravno definirati rute kako bi aplikacija znala što prikazati na kojoj adresi.

Napomena: Prilikom prikaza slika, potrebno je uključiti i **lightbox funkcionalnost**, kao što je prikazano u **laboratorijskoj vježbi 1 (LV1)**. Lightbox omogućuje uvećani prikaz slike kada se na nju klikne, pri čemu se koristi povezivanje putem `href` atributa i identifikatora slike (`#id`). Svaka slika mora imati jedinstveni `id` kako bi otvaranje preko lightboxa ispravno funkcioniralo.

2. Nasumično generirane fotografije

(**Opcionalno**) U ovom dijelu zadatka, umjesto lokalnih slika iz direktorija, koristi se mogućnost dohvaćanja nasumično generiranih fotografija putem vanjskog servisa (primjerice <https://unsplash.it>). Na taj se način galerija popunjava dinamički generiranim URL-ovima koji vraćaju različite slike pri svakom učitavanju stranice.

Ova mogućnost već je implementirana na prvi laboratorijskim vježbama (LV1), a svi ostali elementi zadatka (struktura stranice, prikaz, integracija s EJS-om i Express rutama) ostaju nepromijenjeni.