

Razvoj programske podpore za web i pokretne uređaje

**- predavanja -
2021./2022.**

10. Dinamički web 2/4

Creative Commons



- slobodno smijete:

- **dijeliti** — umnožavati, distribuirati i javnosti priopćavati djelo
- **prerađivati** djelo



- pod sljedećim uvjetima:

- **imenovanje:** morate priznati i označiti autorstvo djela na način kako je specificirao autor ili davatelj licence (ali ne način koji bi sugerirao da Vi ili Vaše korištenje njegova djela imate njegovu izravnu podršku).
- **nekomercijalno:** ovo djelo ne smijete koristiti u komercijalne svrhe.
- **dijeli pod istim uvjetima:** ako ovo djelo izmijenite, preoblikujete ili stvarate koristeći ga, preradu možete distribuirati samo pod licencom koja je ista ili slična ovoj.



U slučaju daljnjeg korištenja ili distribuiranja morate drugima jasno dati do znanja licencne uvjete ovog djela.

Od svakog od gornjih uvjeta moguće je odstupiti, ako dobijete dopuštenje nositelja autorskog prava.

Ništa u ovoj licenci ne narušava ili ograničava autorova moralna prava.

Tekst licence preuzet je s <http://creativecommons.org/>

■ Nastavak prošlog predavanja...

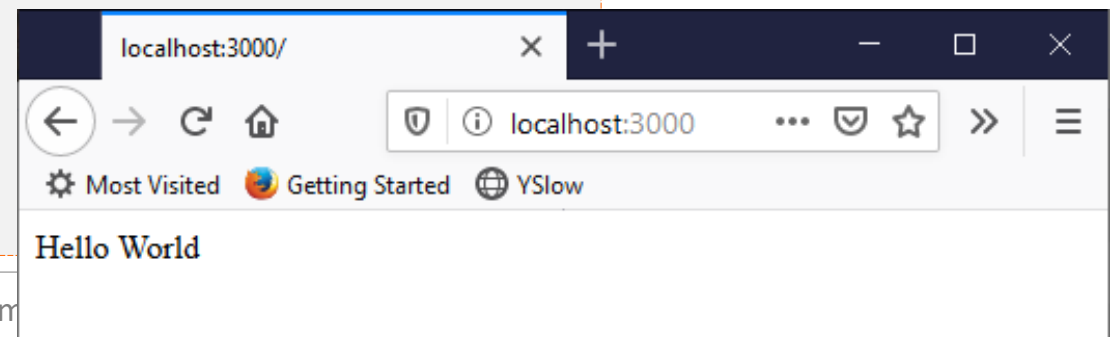
- **ALI:**
 - Radimo samo s korijenskom ('/') putanjom!?
 - Sav kôd je više-manje u jednoj datoteci?
 - Ispreplićemo kôd i HTML i dohvat podataka?
 - Kako poslužujemo statičke datoteke, npr. MB logo?
- **Rješenje:**
 - Ustroj koda: **MVC obrazac**
 - Koristit ćemo popularni **express** radni okvir
 - **Usmjeravanje (*routing*)**

Express radni okvir

express

- "Fast, unopinionated, minimalist web framework for [node](https://nodejs.org/)."
- <https://github.com/expressjs/express>
- Instalacija:
 - \$ npm install express

```
const express = require('express');  
const app = express();  
app.get('/', function(req, res) {  
  res.send('Hello World')  
});  
app.listen(3000);
```



Puno novih stvari!

express

Usmjeravanje (*routing*):

- Mapiranje URI -> kôd koji obrađuje zahtjev

```
app.get('/', function(req, res) {  
  res.send('Hello World')  
});
```

- Request, response objekti
- Middleware obrazac



Usmjeravanje (routing)

express

- "Radimo samo s korijenskom ('/') putanjom!?"
 - -> više ne 😊
- Usmjeravanje definira kako aplikacija odgovara na klijentski zahtjev s obzirom na:
 - Adresu, path (/, /detalji, /student/id/123)
 - Vrstu HTTP zahtjeva (GET, POST, ...)
- Kod expressa ima sljedeću strukturu:

app.Method(Path, Handler)

instanca
express
objekta

HTTP
metoda:
get, post,

Putanja na
poslužitelju,
npr. /about

Funkcija
koja će se
obaviti



Usmjeravanje (routing)

express

- Nekoliko primjera s <https://expressjs.com/en/starter/basic-routing.html>:

```
// Respond with Hello World! on the homepage:
app.get('/', function (req, res) {
  res.send('Hello World!')
})

//Respond to POST request on the root route (/), the application's home page:
app.post('/', function (req, res) {
  res.send('Got a POST request')
})

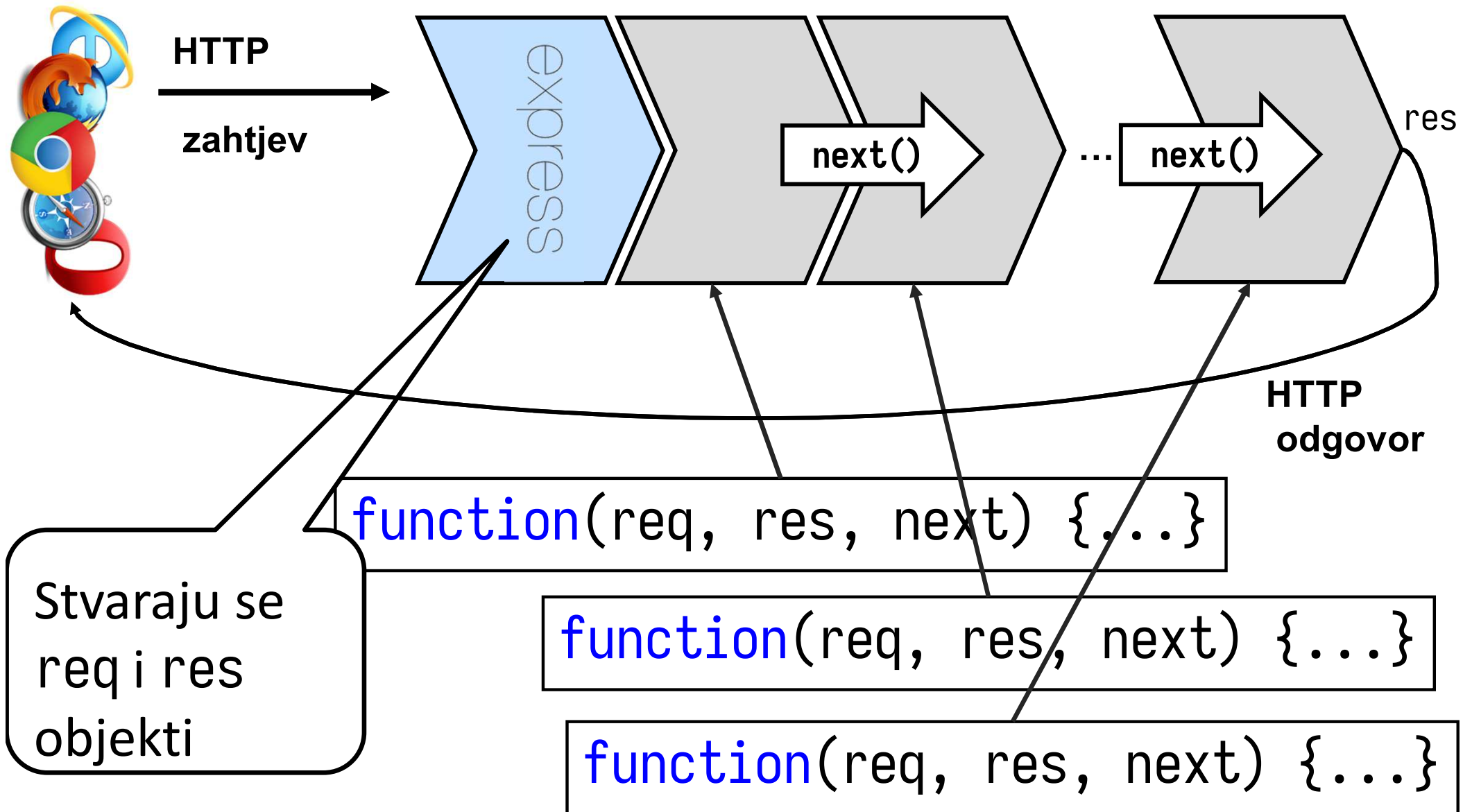
//Respond to a PUT request to the /user route:
app.put('/user', function (req, res) {
  res.send('Got a PUT request at /user')
})

//Respond to a DELETE request to the /user route:
app.delete('/user', function (req, res) {
  res.send('Got a DELETE request at /user')
})
```

- Middleware funkcije su funkcije koje imaju pristup:
 - **request** objektu
 - Odgovara HTTP zahtjevu
 - Stvara ga Express - ima svojstva koja odgovaraju query stringu, parametrima, tijelu (*body*), zaglavljima, itd.
 - **response** objektu
 - Odgovara HTTP odgovoru
 - Pomoću njega odgovaramo na zahtjev
 - **next()** funkciji
 - Prosljeđuje obavljanje sljedećoj middleware funkciji u lancu
- Middleware funkcije mogu:
 - Obavljati proizvoljan kôd
 - Mijenjati request i response objekte
 - Završiti request-response ciklus
 - Pozvati sljedeću funkciju u lancu

Express middleware

express



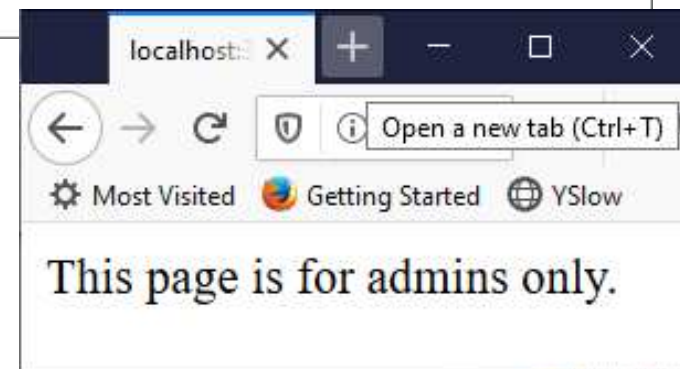
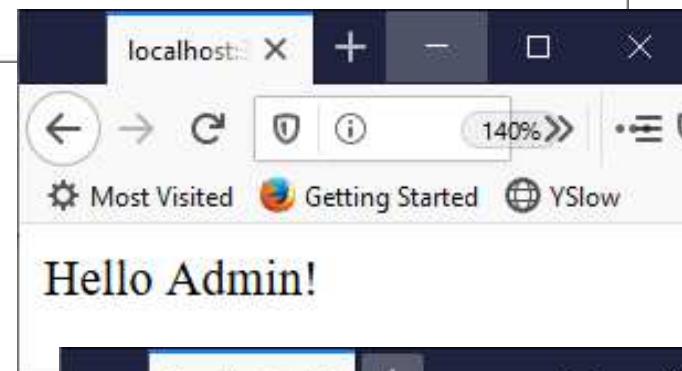
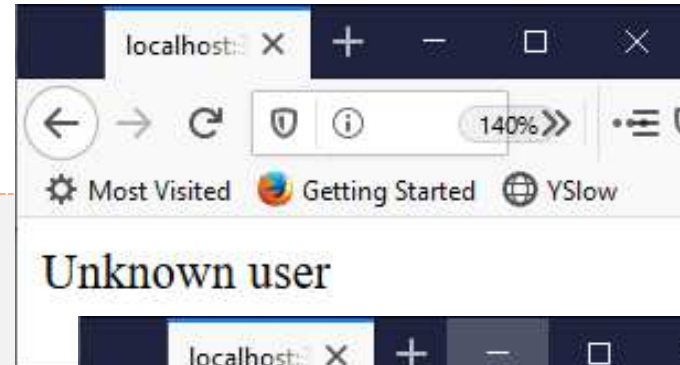
Request i Response objekti "s 500 metara"

- **Request** - predstavlja HTTP zahtjev
 - **req.query** - objekt koji sadrži query parametre
 - Npr. `&foo=9` \Rightarrow `{foo: '9'}`
 - **req.body** - objekt sadrži parsirani body kao ključ-vrijednosti
 - Za parsiranje koristiti middleware, kao `express.json()` ili `express.urlencoded()`
 - **req.get(prop)** - vraća traženo svojstvo iz HTTP zaglavlja
 - ltd. : <https://expressjs.com/en/api.html#req>
- **Response** - predstavlja HTTP odgovor
 - **res.set(p,v)** - postavlja svojstvo p na vrijednost v u zaglavlju
 - Npr. `&foo=9` \Rightarrow `{foo: '9'}`
 - **res.status(code)** - postavlja HTTP kod odgovora
 - **res.send(s)** - vraća sadržaj s
 - Može se ulančavati: **res.status(404).send("Nema!") ;**

Middleware - primjer

```
const express = require('express');
const app = express();
app.use(function(req, res, next) {
  console.log('Incoming request:', req.url);
  next();
});
app.use(function(req, res, next) {
  // zamislamo da ovdje provjeravamo u bazi podataka
  // je li (a) korisnik postojeći.
  //      (b/c) ako je postojeći, je li: admin ili user?
  let i = Math.floor(Math.random() * 3);
  if (i == 2) {
    res.status(403)
      .send('Unknown user');
  } else {
    req.user = ["user", "admin"][i];
    next();
  }
}); //...->
```

Postoji i
app.use() !



```
app.get('/', function(req, res) {
  if (req.user === 'admin') {
    res.send('Hello Admin!');
  } else {
    res.send('This page is for admins only.');
```

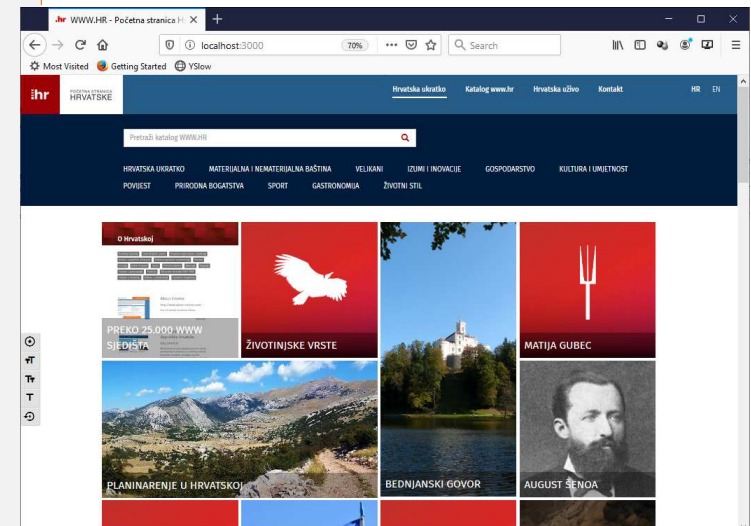
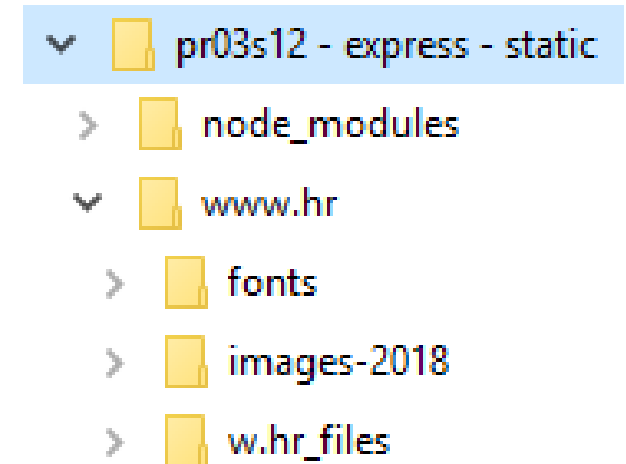
- Postoji pet vrsta:
 - [Application-level middleware](#) (vidjeli na prethodnim slajdovima)
 - [Router-level middleware](#) (vrlo slično)
 - [Error-handling middleware](#) (za one koji žele znati više)
 - [Third-party middleware](#) (npr. express-useragent)
 - [Built-in middleware](#), svega tri:
 - ***express.static***
 - *express.json*
 - *express.urlencoded*
- Riješimo ovaj problem:
"Kako poslužujemo statičke datoteke, npr. MB logo?"
s jednom linijom koda:
 - **`app.use(express.static('neki direktorij'));`**

Rekreirajmo primjer "Posluživanje statičkih datoteka" iz prethodnog predavanja (p08s50)

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

app.use(express.static('www.hr'));
// Tipično:
// app.use(express.static('public'));
// app.use(express.static('assets'));
// (može ih biti više!)

app.listen(3000);
```



MVC motivacija

- Kod u jednoj datoteci?
- Isprepleten kod i HTML?
-> **Spaghetti code**
- **(Anti)primjer** s <https://thisinterestsme.com/mixing-php-html/>:
- Pomiješana:
 - Logika
 - Prezentacija
 - Dohvat podataka

```
1 <?php
2
3 echo '<h1>New Users</h1>';
4
5 $sql = "SELECT * FROM users ORDER BY date_registered";
6 $result = mysql_query($sql) or die(mysql_error());
7
8 echo '<table class="my-table-class">';
9 while($row = mysql_fetch_assoc($result)){
10     echo '<tr><td>' . $row['username'] . '</td><td>' . $row['date_registered'] . '
11 }
12 echo '</table>';
13
14 function random_custom_function($var){
15     $var = $var + 1;
16     return '<span style="font-weight:bold;">' . $var . '</span>';
17 }
18
19 $sql = "SELECT * FROM table WHERE column = 'test'";
20 $result = mysql_query($sql) or die(mysql_error());
21
22 echo '<div id="test">';
23 $i = 0;
24 while($row = mysql_fetch_assoc($result)){
25     if($row['type'] == 3){
26         echo '<div style="margin-bottom:20px;">' . random_custom_function($row['va
27         $i++;
28     }
29     else{
30         echo '<div style="margin-bottom:20px;">' . $row['val'] . '</div>';
31     }
32 }
33
34 if($i == 0){
35     echo '<table>';
36     echo '<tr><td>Found none!</td></tr>';
37     echo '</table>';
38 }
```


ALI:

Radimo samo s korijenskom ('/') putanjom!?

Sav kôd je više-manje u jednoj datoteci?

Ispreplićeemo kôd i HTML-a i dohvat podataka?

The image shows the letters 'MVC' in a very large, bold, and heavily distressed font. The letters are black with a grainy, textured appearance, giving them a stencil-like or hand-painted look. They are centered horizontally and take up a significant portion of the lower half of the slide.

■ MVC obrazac: *Model-View-Controller* (1/2)

- MVC je obrazac za razvoj **korisničkih sučelja**, inicijalno razvio Trygve Reenskaug još 1979. za Smalltalk-80
- Posebno popularan postao za razvoj web-aplikacija
 - Osnovna ideja ista
 - **Implementacije i interpretacije variraju**
- Želimo **razdvojiti nadležnosti**
 - Poznat princip u računarstvu: *separation of concerns*
- Razdvojiti ćemo aplikaciju u labavo povezane (*loosely coupled*) komponente:
 - ***Model***
 - ***View***
 - ***Controller***

MVC obrazac: *Model-View-Controller* (2/2)

■ *Model*

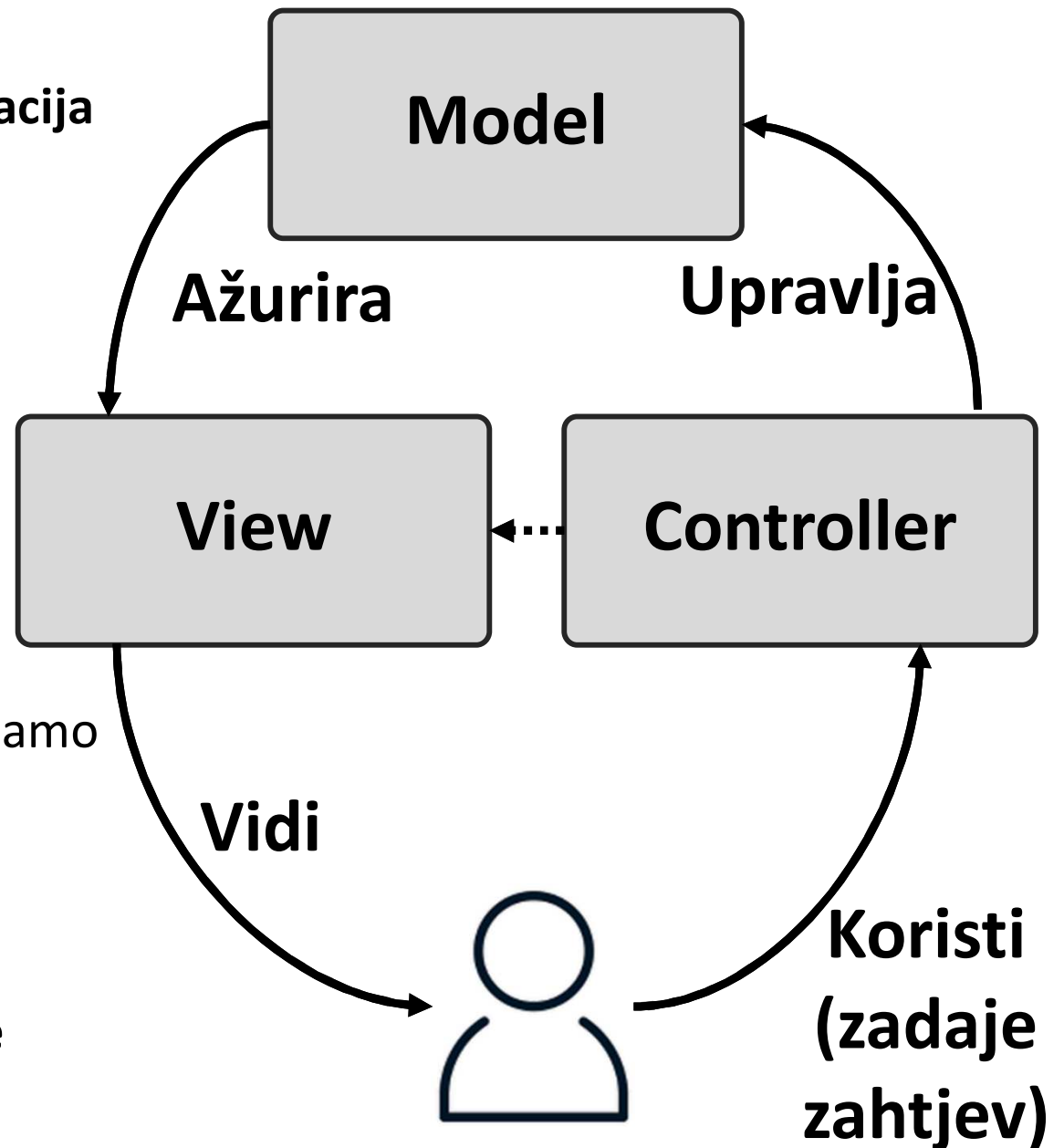
- Zaduženje: dohvat i manipulacija podacima
- Tipično u suradnji s bazom podataka
- Nije svjestan C komponente, nekad ni V

■ *View*

- Zaduženje: prezentacija dostavljenih podataka
- "Glup", ne sadrži logiku, zna samo za M jer prikazuje podatke iz modela

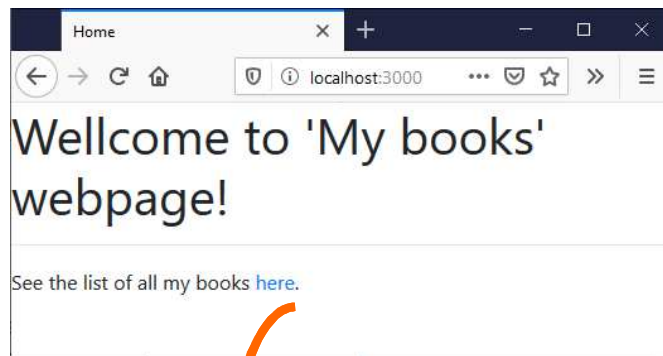
■ *Controller*

- Zaduženje: prima zahtjeve te upravlja s M i V



MVC primjer: katalog knjiga

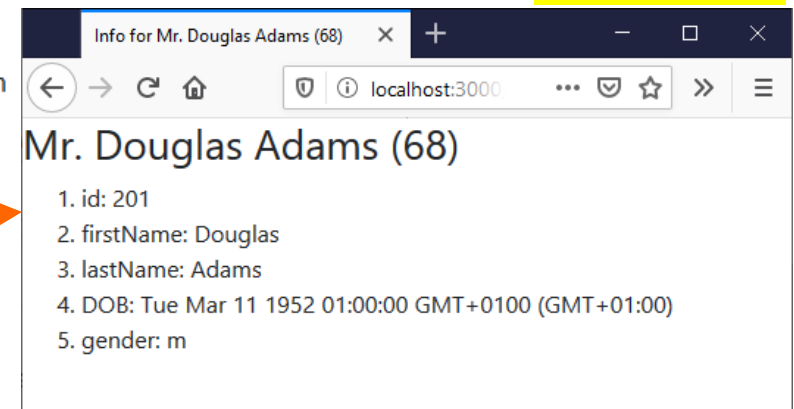
<http://localhost:3000/>



<http://localhost:3000/books>



<http://localhost:3000/author/201>



Što bi tu mogao
biti model?

Primjer: "My books" - Model: Author

```
const AuthorClass = class Author {
  constructor(firstName, lastName, DOB, gender) {
    this.firstName = firstName;
    this.lastName = lastName;
    this.DOB = DOB;
    this.gender = gender;
  }
  get age() {
    return (new Date()).getFullYear() - this.DOB.getFullYear();
  }
  get formattedAuthor() {
    return ((this.gender === 'm') ? 'Mr. ' : 'Mrs.') +
      this.firstName + ' ' + this.lastName + ` (${this.age})`;
  }
};
AuthorClass.prototype.toString = function() {
  return this.formattedAuthor;
}
module.exports = AuthorClass;
```

Primjer: "My books" - Model: Book

```
module.exports =  
  class Book {  
    constructor(title, author, language, publisher, ISBN) {  
      this.title = title;  
      this.author = author;  
      this.language = language;  
      this.publisher = publisher;  
      this.ISBN = ISBN;  
    }  
  };
```

Npr. novu knjigu možemo instancirati s:

```
let nk = new Book('Code Complete',  
  new Author('Steve', 'McConnell', new Date('1965-07-10'), 'm'),  
  'Microsoft Press; 2nd edition',  
  '0735619670'  
);
```

Primjer: "My books" - Repozitorij knjiga

```
const Author = require('../models/author.model');
const Book = require('../models/book.model');
class BookRepository {
  constructor() {
    this.books = [];
    this.seedBooks();
  }
  seedBooks() {
    this.books.push(
      new Book('Code Complete',
        new Author('Steve', 'McConnell', new Date('1965-07-10'), 'm'),
        'Microsoft Press; 2nd edition',
        '0735619670'
      ));
    // (...) this.books.push(...)
  }
}
const repoInstance = new BookRepository();
module.exports = repoInstance;
```

■ Primjer "Moje knjige"

- MVC nacrtan drugačije



GET /
HTML

**Controller
(router)**

```
{  
  title: "Home"  
}
```

HTML

model

**View
home**



GET /books
HTML

**Controller
(router)**

```
{  
  title: "My books",  
  books: [ ... ]  
}
```

HTML

**View
books**



GET
/author/101
HTML

**Controller
(router)**

```
{  
  title: "Info for...",  
  author: { id, ... }  
}
```

HTML

**View
author**

Primjer "My books": server.js

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

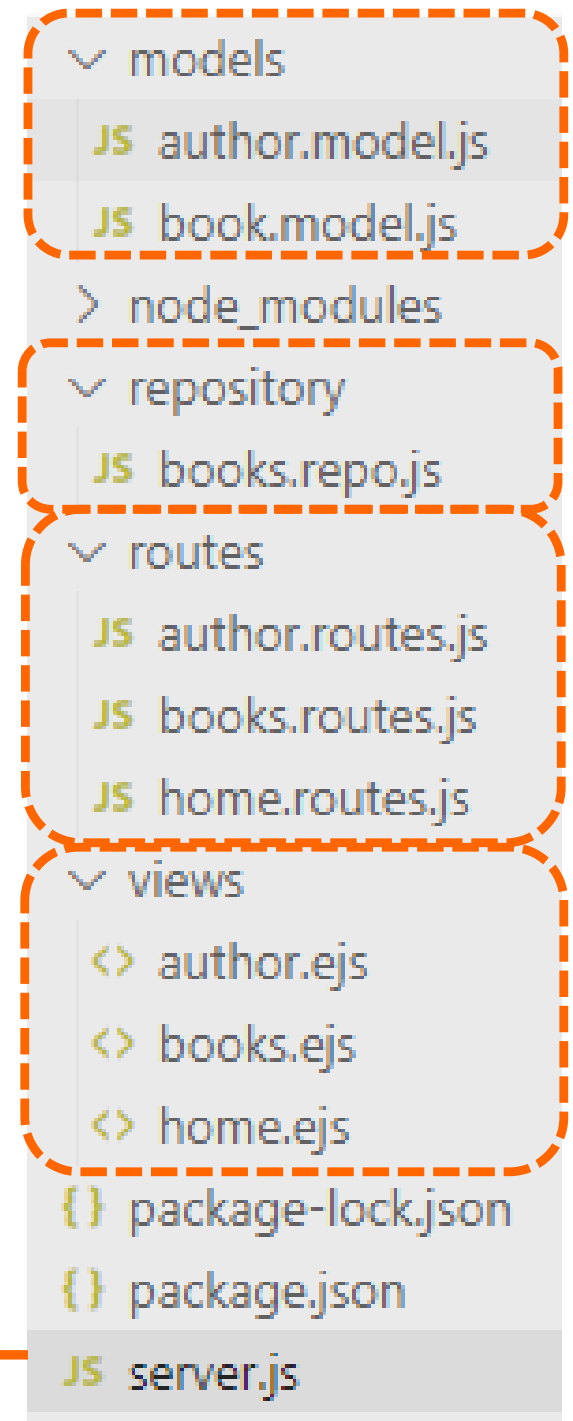
const homeRouter = require('./routes/home.routes');
const booksRouter = require('./routes/books.routes');
const authorRouter = require('./routes/author.routes');

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);
```



■ Kako generirati HTML?

- **Ispreplićemo kôd i HTML i dohvat podataka?**
- Određena razina ispreplitanja se ne može izbjeći, ali:
 - Možemo dohvatiti podatke (prije samog generiranja HTML) i pripremiti podatke (model) te ih predati procesu koji generira view
 - Je li lakše ugrađivati:
 - HTML u podatke, ili
 - podatke u HTML?
 - > ovo drugo ("konkateniramo podatke u HTML")
 - > zato koristimo obrasce (*template*) koji su HTML ili neke izvedenice od HTML-a i unutra ugrađujemo podatke iz modela
- View engines, view templates
 - Softver koji olakšava generiranje HTML-a
 - Koristi obrasce u koji su ugrađene varijable (modela)

Npr. lista template enginea za express (wiki)

Template Engines

Template engines that are Express compliant out of the box.

- [Pug](#) -- Haml inspired template engine (formerly Jade)
- [Haml.js](#) -- Haml implementation
- [EJS](#) -- Embedded JavaScript template engine
- [hbs](#) -- adapter for Handlebars.js, an extension of
- [React](#) -- renders React components on server. It renders
- [h4e](#) -- adapter for Hogan.js, with support for partials and layouts
- [hulk-hogan](#) -- adapter for Twitter's [Hogan.js](#) (Mustache syntax), with support
- [combyne.js](#) -- A template engine that hopefully works the way you'd expect.
- [swig](#) -- fast, Django-like template engine
- [Nunjucks](#) -- inspired by jinja/twig
- [marko](#) -- A fast and lightweight HTML-based templating engine that compiles templates to CommonJS modules and supports streaming, async rendering and custom tags. (render directly to the HTTP response stream)
- [whiskers](#) -- small, fast, mustachioed
- [Blade](#) -- HTML Template Compiler, inspired by Jade & Haml
- [Haml-Coffee](#) -- Haml templates where you can write inline CoffeeScript.
- [Webfiller](#) -- plain-html5 dual-side rendering, self-configuring routes, organized source tree, etc.
- [express-hbs](#) -- Handlebars with layouts, partials and blocks for express 3 from [Bar](#)
- [express-handlebars](#) -- A Handlebars view engine for Express which doesn't suck.
- [express-views-dom](#) -- A DOM view engine for Express.
- [rivets-server](#) -- Render Rivets.js templates on the server.
- [Exbars](#) -- A flexible Handlebars view engine for Express
- [Liquidjs](#) -- A Liquid engine implementation for both Node.js and browsers
- [express-tl](#) -- A template-literal engine implementation for Express.
- [vuexpress](#) -- A Vue.js server side rendering engine for Express.js.

Mi ćemo koristiti EJS jer je vrlo jednostavan i blizak HTML-u. Ukratko, u HTML se ugrađuju:

```
<% code %>  
<%= code %>  
<%- code %>
```

Primjer:

```
<% if (user) { %>  
  <h2><%= user.name %></h2>  
<% } %>
```

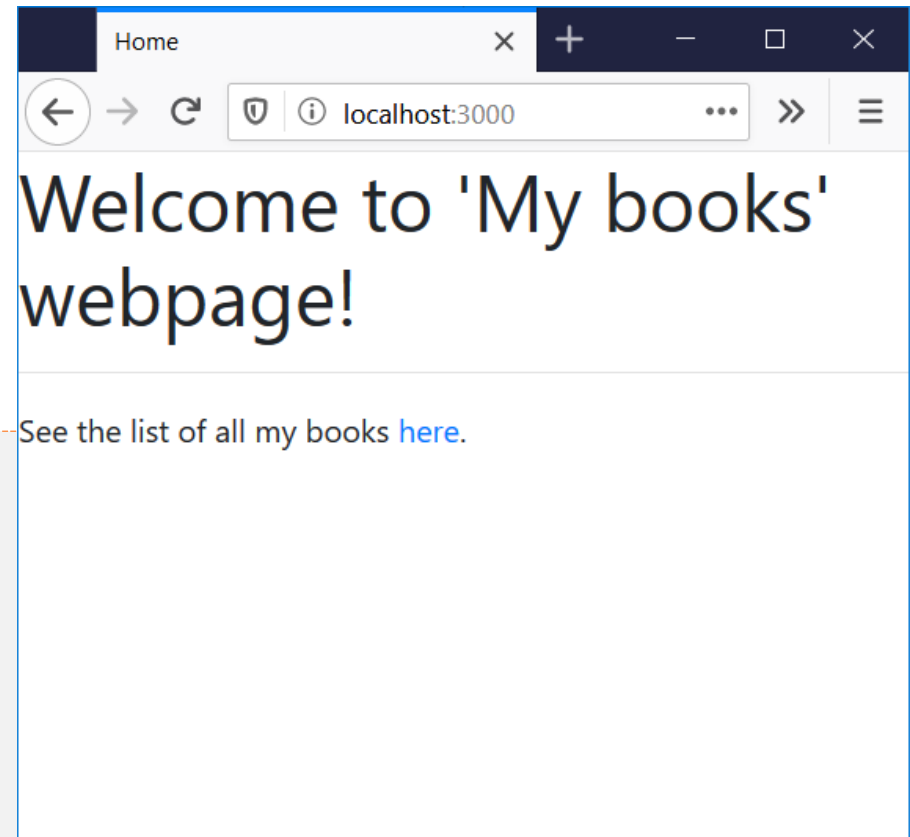
Primjer "My books": view home.ejs

Model:

```
{  
  title: "Home"  
}
```

home.ejs

```
<!DOCTYPE html>  
<html>  
  <head>  
    <title><%= title %></title>  
  </head>  
  <body>  
    <h1>Welcome to 'My books' webpage!</h1>  
    <hr>  
    <p>See the list of all my books <a href="/books">here</a>.</p>  
  </body>  
</html>
```



Primjer "My books": view books.ejs

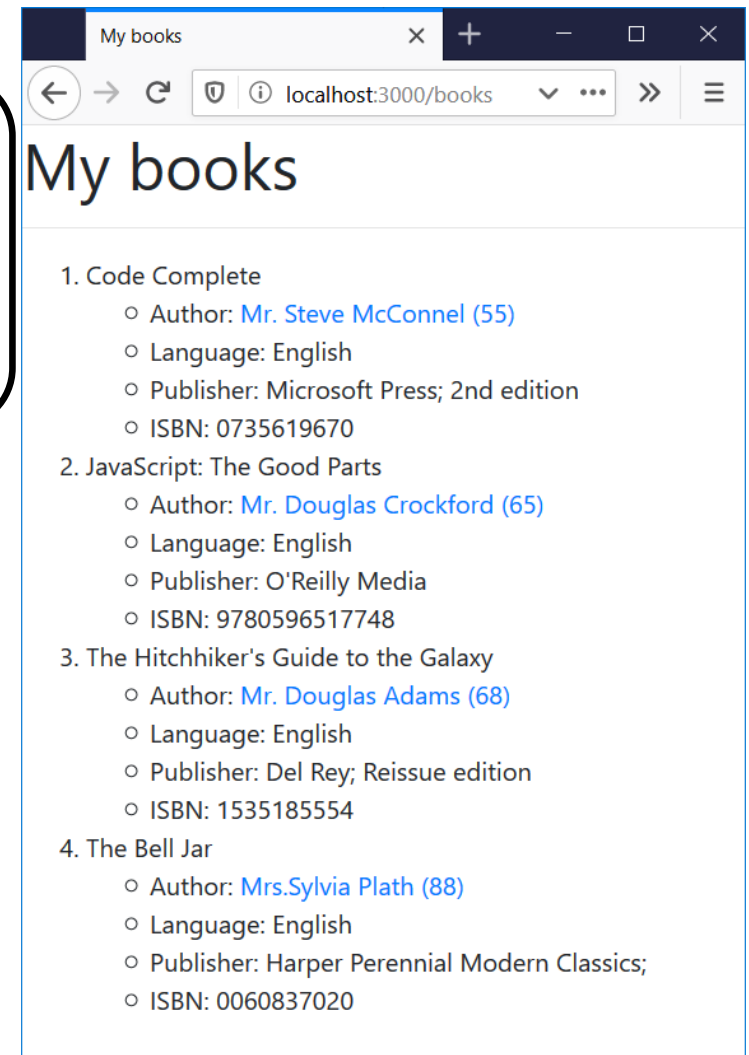
Model:

```
{  
  title: "My books",  
  books: [ {title:"..", author:...}, ... ]  
}
```

books.ejs

```
<body>  
<h1>My books</h1>  
<hr>  
<ol>  
  <% for (let book of books)>  
    <li> <%= book.title %>  
      <ul>  
        <li>Author: <a href="/author/<%= book.author  
r.id %>"><%= book.author %></a></li>  
        <li>Language: <%= book.language %></li>  
        <li>Publisher: <%= book.publisher %></li>  
        <li>ISBN: <%= book.ISBN %></li>  
      </ul>  
    </li>  
  <% } %>  
</ol>  
</body>
```

Prikazan samo body,
ostalo je isto kao na
prethodnom slajdu.



Primjer "My books": view author.ejs

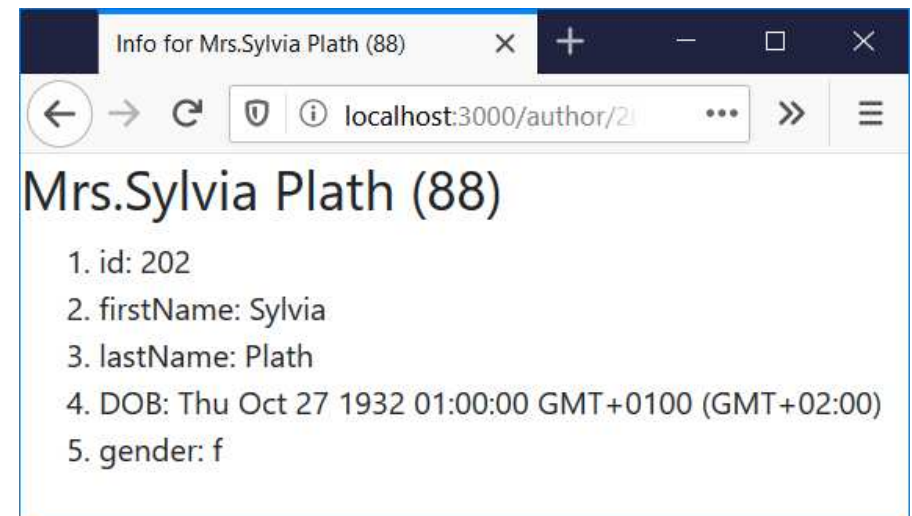
Model:

```
{  
  title: "Info for Mrs. Sylvia Plath",  
  author: {  
    id: 202,  
    firstName: "Sylvia"  
    ...  
  }  
}
```

author.ejs

```
<body>  
  <h3> <%= author %></h3>  
  <ol>  
    <li>id: <%= author.id %>  
    <li>firstName: <%= author.firstName %>  
    <li>lastName: <%= author.lastName %>  
    <li>DOB: <%= author.DOB %>  
    <li>gender: <%= author.gender %>  
  </ol>  
</body>
```

Prikazan samo body, ostalo je isto kao na view home slajdu.



Primjer "My books" - home router

- Konačno, povežimo Model i View:
- *Router (controller)* :
 - prima zahtjev
 - pribavlja model
 - predaje ga u view kako bi se generirao HTML koji se onda vraća

home.routes.ejs

```
var express = require('express');
var router = express.Router();

router.get('/', function(req, res, next) {
  res.render('home', {
    title: 'Home'
  });
});
module.exports = router;
```

Model

Konfigurirani template engine (EJS) će potražiti home.ejs u konfiguriranom direktoriju (views):

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

Primjer "My books" - books router

books.routes.js

```
var express = require('express');
var router = express.Router();
const repo = require('../repository/books.repo');
```

*// Primijetite da je ovo ukupni path "/books" jer je ova ruta
// mapirana na /books u server.js*

```
router.get('/', function(req, res, next) {
  res.render('books', {
    title: 'My books',
    books: repo.books
  });
});
```

```
module.exports = router;
```

Model

Ali odakle knjige?
Što je repo?

Primjer "My books" - author router

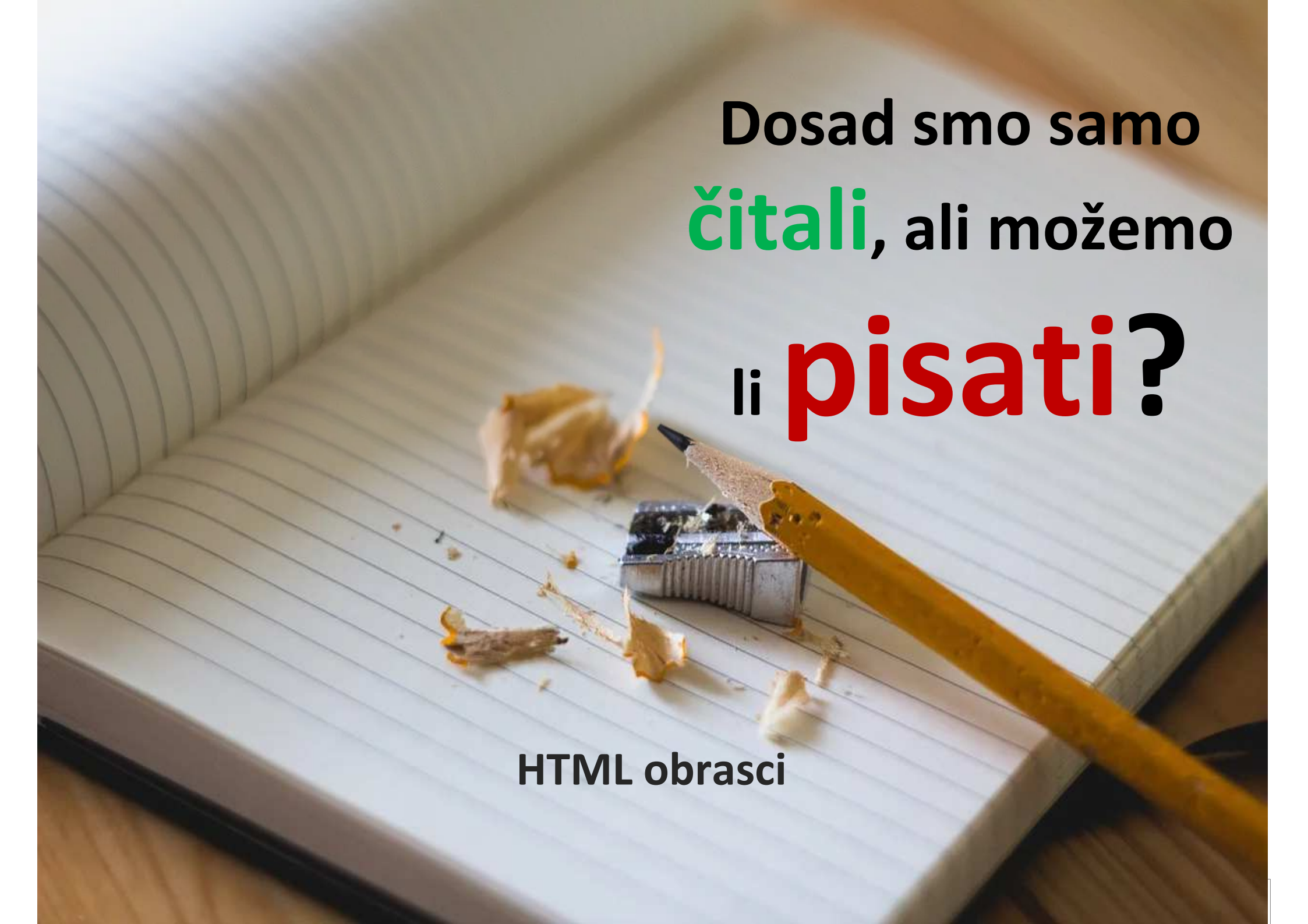
author.routes.js

```
var express = require('express'); var router = express.Router();
const repo = require('../repository/books.repo');
// S ([0-9]{1,10}) ćemo se dodatno osigurati da je id 1-10 znamenkasti broj.
router.get('/:id([0-9]{1,10})', function(req, res, next) {
  let id = parseInt(req.params.id); let author;
  for (let book of repo.books) {
    if (book.author.id === id) {
      author = book.author; break;
    }
  }
  if (author) {
    res.render('author', { title: 'Info for ' + author, author: author });
  } else {
    res.status(404).send("Are you guessing?");
  }
});
module.exports = router;
```

Pogledajte sami još neke *express routing* opcije:

<https://expressjs.com/en/guide/routing.html>

Model

An open notebook with lined pages is shown. A yellow pencil lies diagonally across the right page, with its tip pointing towards the center. Several pencil shavings are scattered on the pages around the pencil. The background is a wooden surface.

Dosad smo samo
čitali, ali možemo
li **pisati**?

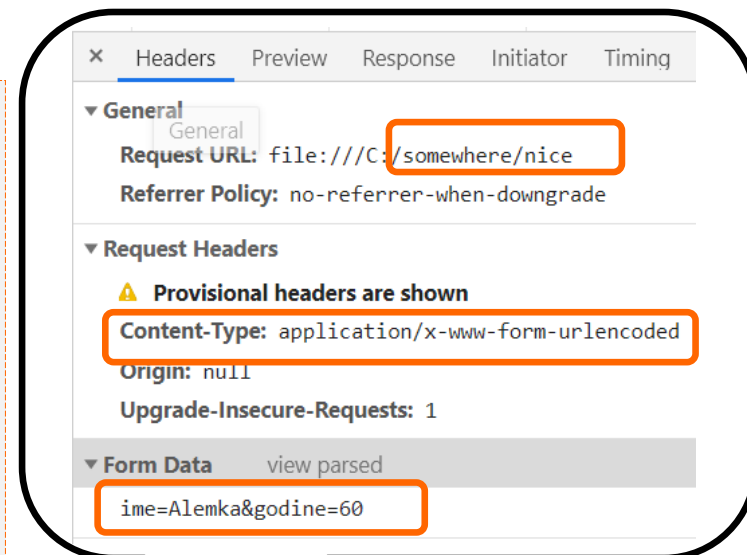
HTML obrasci

Unos podataka putem web-aplikacije

- Koristimo mehanizam **obrazaca (forms)**
 - Vidjeti predavanje 2. HTML, poglavlje Obrasci HTML-a (HTML *forms*)
- Ukratko, definiramo `<form>` element koji:
 - a) Ima postavljene sljedeće attribute:
 - **action**, npr. **action="/authors/add"**, URL na koji će se poslati sadržaj forme
 - **method**, **method="POST"**, HTTP metoda koja će se koristiti kod slanja sadržaja forme. POST metoda kodira sadržaj forme unutar tijela zahtjeva i nema ograničenja na duljinu.
 - b) Sadrži N HTML elemenata koji imaju attribute:
 - **name**, ime elementa, mora biti postavljeno (ili će se element ignorirati)
 - **value**, vrijednost elementa, može i ne mora biti inicijalno postavljeno, tipično korisnik unosi vrijednosti

Primjer – jednostavna forma

```
<form action="/somewhere/nice"
method="POST">
  <input type="text" name="ime"><br/>
  <input type="number" name="godine"><br/>
  <input type="submit" value="Submit">
</form>
```



The screenshot shows a web browser window titled 'Simple-form'. The address bar shows 'file:///C:/Users/igor/Desktop/'. The form has two input fields: 'ime' with the value 'Alemka' and 'godine' with the value '60'. There is a 'Submit' button. Arrows point from the input fields to labels 'ime' and 'godine'.



POST: ime=alemka&godine=60

... somewhere/nice

Primijetiti da nema tipova podataka – sve su stringovi!

■ Proširimo primjer – unos podataka

- Želimo moći unijeti novu **knjigu** i novog **autora**
- Krenimo s **autorom** (koji nam treba za knjigu), potrebno je:
 - Napraviti novu **GET** rutu (URL) koja će vratiti obrazac za autora koji korisnik treba popuniti, npr. **authors/add**
 - Napraviti novu **POST** rutu (URL) na koju će korisnik predati (*submit*) obrazac, npr. **authors/add**
 - U tu svrhu ćemo napraviti i *refactoring** repozitorija tako da:
 - razložimo podatke na tri „tablice”: autori, knjige i jezici (u realnim uvjetima bi i izdavača).
 - Dodamo funkcije za dodavanje zapisa za svakog od njih

**Code refactoring is the process of restructuring existing computer code—changing the factoring—without changing its external behavior. --Wikipedia*

Repository refactoring (1/2)

```
class BookRepository {  
  constructor() { this.seedRepo(); }  
  addBook(title, author, language, publisher, ISBN) {  
    this.books.push(  
      new Book(title, author, language, publisher, ISBN)  
    ); }  
  addAuthor(firstName, lastName, DOB, gender) {  
    let newId = this.authors[this.authors.length - 1].id + 1;  
    let newAuthor = new Author(newId, firstName, lastName, DOB, gender);  
    this.authors.push(newAuthor);  
    return newAuthor;}  
  getLanguage(lang) {  
    let language = this.languages.find(l => l.abbrev === lang);  
    return language && language.langName || 'unknown?'; }  
  getAuthor(id) {  
    for (let a of this.authors) {  
      if (a.id === id) return a;  
    }  
    return null; }  
}
```

Restrukturiranje koda (*code refactoring*) je vrlo uobičajena pojava prilikom razvoja i održavanja softvera – izrazito je teško „pogoditi iz prve”.

Repository refactoring (2/2)

```
... /***** Seeding repo below: *****/
seedRepo() {
  this.seedAuthors();
  this.seedBooks(); this.seedLanguages();
}
seedAuthors() {
  this.authors = [];
  this.authors.push(new Author(100, 'Steve', 'McConnel', new Date('1965-07-10'), 'm'));
  ...
}
seedBooks() {
  this.books = [];
  this.addBook('Code Complete', this.getAuthor(100), 'en', '... edition', '0735619670');
  ...
}
seedLanguages() {
  this.languages = [{ abbrev: "en", langName: "English,, },
                    { abbrev: "hr", langName: "Croatian"}];
}
```

Pogledajte priloženi kod na gitlabu za cjelovitu i ispravno formatiranu verziju

Novi autor: GET /authors/add

authors.routes.js

```
router.get('/add', function(req, res, next) {  
  res.render('addAuthor', { title: "Add author" });  
});
```

addAuthor.ejs

```
<form action="/authors/add" method="POST"><fieldset>  
  <legend>New author</legend>  
  <label for="firstName">First name:</label>  
  <input type="text" name="firstName" id="firstName" size="30"><br />  
  <label for="lastName">Last name:</label>  
  <input type="text" name="lastName" id="lastName" size="30"><br />  
  <label for="DOB">Date of Birth:</label>  
  <input type="date" name="DOB" id="DOB"><br />  
  <label for="gender">Gender:</label>  
  <input type="radio" name="gender" id="gender" value="m" checked>Male  
  <input type="radio" name="gender" value="f">Female <hr>  
  <div class="submitButton">  
    <input type="submit" value="Submit">  
    <input type="reset" value="Reset">  
  </div>  
</fieldset></form>
```

New author

First name:

Last name:

Date of Birth:

Gender: ☒ Male ☐ Female

Novi autor: POST /authors/add

authors.routes.js

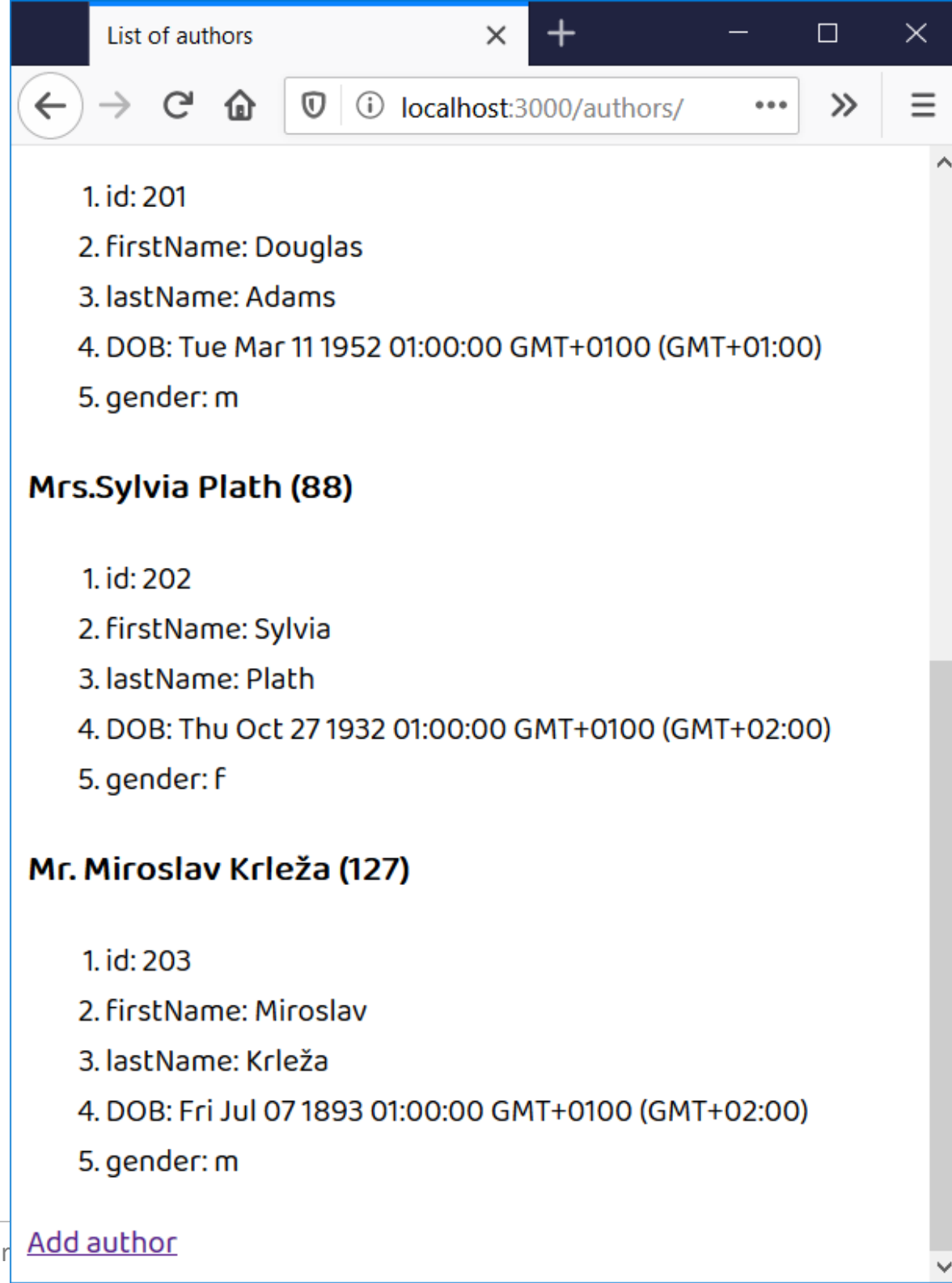
```
router.post('/add', function(req, res, next) {  
  console.log(req.body);  
  try {  
    let newAuthor = repo.addAuthor(  
      req.body.firstName,  
      req.body.lastName,  
      new Date(req.body.DOB),  
      req.body.gender  
    );  
    res.redirect('/authors');  
  } catch (err) {  
    res.render('addAuthor', {  
      title: "Add author",  
      error: JSON.stringify(err)  
    });  
  }  
});
```

```
{  
  firstName: 'Miroslav',  
  lastName: 'Krleža',  
  DOB: '1893-07-07',  
  gender: 'm'  
}
```

Opet, primamo stringove, moramo se sami pobrinuti da ih pretvorimo u odgovarajuće tipove!

Ako je sve u redu, preusmjerimo preglednik na popis svih autora. Inače, vraćamo istu stranicu, s podacima o grešci u modelu.

Uspješan unos, stranica sa svim autorima:



List of authors

localhost:3000/authors/

1. id: 201
2. firstName: Douglas
3. lastName: Adams
4. DOB: Tue Mar 11 1952 01:00:00 GMT+0100 (GMT+01:00)
5. gender: m

Mrs. Sylvia Plath (88)

1. id: 202
2. firstName: Sylvia
3. lastName: Plath
4. DOB: Thu Oct 27 1932 01:00:00 GMT+0100 (GMT+02:00)
5. gender: f

Mr. Miroslav Krleža (127)

1. id: 203
2. firstName: Miroslav
3. lastName: Krleža
4. DOB: Fri Jul 07 1893 01:00:00 GMT+0100 (GMT+02:00)
5. gender: m

[Add author](#)

Nova knjiga: GET

/books/add: router (1/3)

books.routes.js

```
router.get('/add', function (req, res, next) {
  let model = {
    title: 'Add book',
    authorsSelect: {
      name: "author",
      list: repo.authors.map(x => ({
        value: x.id,
        name: x.toString()
      })))
    },
    languagesSelect: {
      name: "language",
      list: repo.languages.map(x => ({
        value: x.abbrev,
        name: x.langName
      })),
      selected: "en"
    }
  };
  console.log(JSON.stringify(model));
  res.render('addBook', model);
});
```

Pripremamo
podatke za
<SELECT>
element

```
{
  "title": "Add book",
  "authorsSelect": {
    "name": "author",
    "list": [
      {
        "value": 100,
        "name": "Mr. Steve McConnell (55)"
      },
      ...
      {
        "value": 203,
        "name": "Mr. Miroslav Krleža (127)"
      }
    ]
  },
  "languagesSelect": {
    "name": "language",
    "list": [
      {
        "value": "en",
        "name": "English"
      },
      {
        "value": "hr",
        "name": "Croatian"
      }
    ],
    "selected": "en"
  }
}
```

Nova knjiga: GET /books/add: view (2/3)

addBook.ejs

```
<form action="/books/add" method="POST">
  <fieldset>
    <legend>New book</legend>
    <div><label for="title" title="">Title:</label> <input type="text" name="title"
id="title" size="30"></div>
    <div><label for="author">Author:</label>
      <% locals.selectData = authorsSelect; %>
      <%- include(`partials/_select`); %></div>
    <div><label for="language">Language:</label>
      <% locals.selectData = languagesSelect; %>
      <%- include(`partials/_select`); %></div>
    <div><label for="" publisher"">Publisher:</label> <input type="text"
name="publisher" id="publisher"></div>
    <div><label for="" ISBN"">ISBN:</label> <input type="text" name="ISBN" id="ISBN"
size="10"></div>
    <hr>
    <div class="submitButton">
      <input type="submit" value="Submit">
      <input type="reset" value="Reset">
    </div>
  </fieldset>
</form>
```

Kako ne bi ponavljali kod, koristimo parcijalni view da iscrtamo <SELECT>

New book

Title:	<input type="text"/>
Author:	Mr. Steve McConnel (55) ▾
Language:	English ▾
Publisher:	<input type="text"/>
ISBN:	<input type="text"/>

Nova knjiga: GET /books/add: parcijalni view (3/3)

partials/_select.ejs

```
<% let selectData = locals.selectData; %>
<select id="<%=selectData.id%>" <%=selectData.name ? 'name="' + selectData.name +
'":"' : ' "' %> <%= selectData.attributes ? selectData.attributes : ' "' %> >
  <% selectData.list.forEach(function(item) { %>
    <option <%= (item.value == selectData.selected) ? "selected" : "" %> value="<%=
item.value%>"><%=item.name%></option>
  <% }); %>
</select>
```

```
{
  "name": "language",
  "list": [
    {
      "value": "en",
      "name": "English"
    },
    {
      "value": "hr",
      "name": "Croatian"
    }
  ],
  "selected": "en"
}
```

```
<select id="" name="language" >
  <option selected value="en">English</option>
  <option value="hr">Croatian</option>
</select>
```

Nova knjiga: POST /books/add

books.routes.js

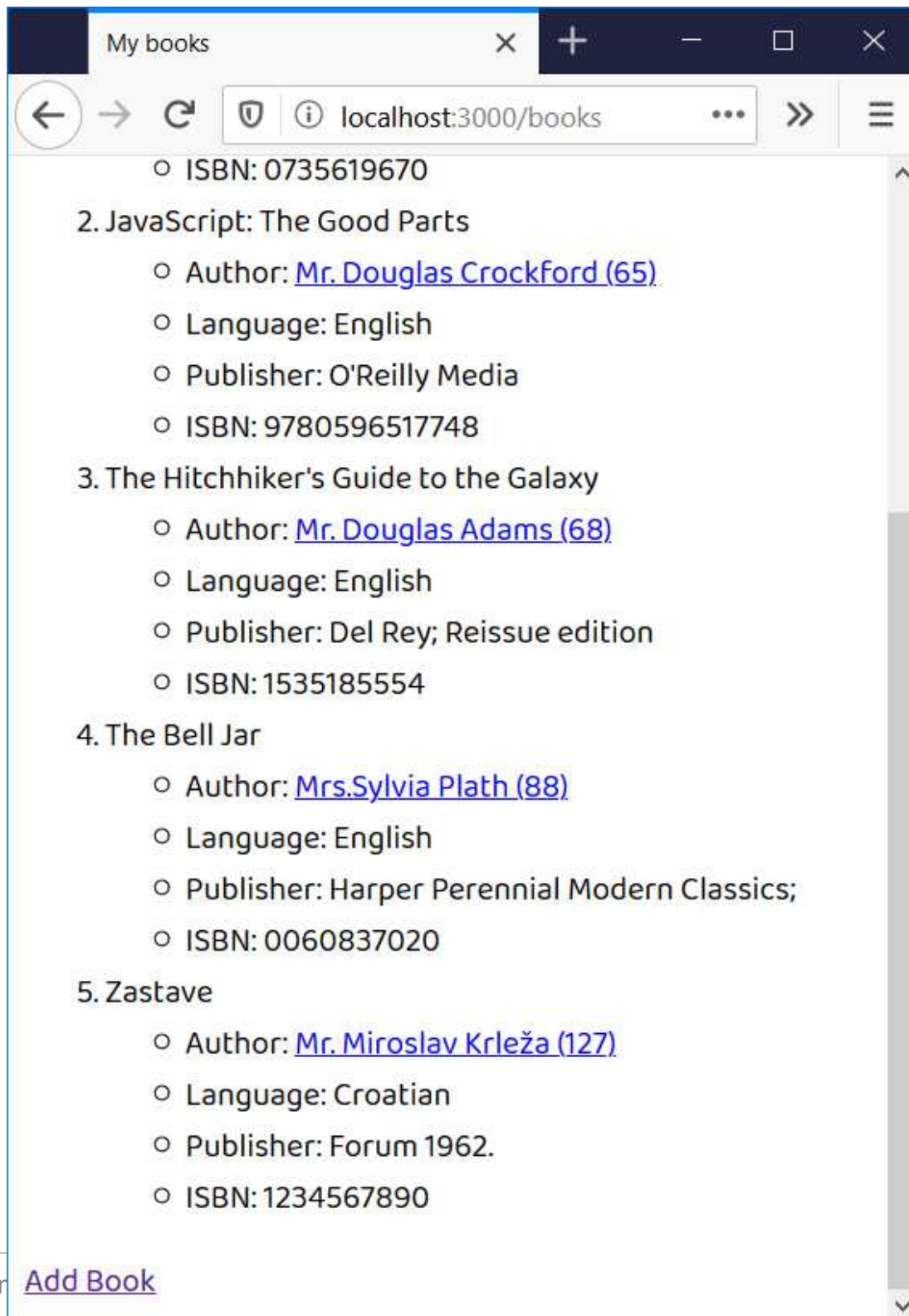
```
router.post('/add', function (req, res, next) {  
  console.log(req.body);  
  try {  
    let newBook = repo.addBook(  
      req.body.title,  
      repo.getAuthor(parseInt(req.body.author)),  
      req.body.language,  
      req.body.publisher,  
      req.body.ISBN  
    );  
    res.redirect('/books');  
  } catch (err) {  
    res.render('addBook', {  
      title: "Add books",  
      error: JSON.stringify(err)  
    });  
  }  
});
```

```
{  
  title: 'Zastave',  
  author: '203',  
  language: 'hr',  
  publisher: 'Forum 1962.',  
  ISBN: '1234567890'  
}
```

Opet, primamo stringove, moramo se sami pobrinuti da ih pretvorimo u odgovarajuće tipove!

Ako je sve u redu, preusmjerimo preglednik na popis svih knjiga. Inače, vraćamo istu stranicu, s podacima o grešci u modelu.

Uspješan unos, stranica sa svim knjigama:



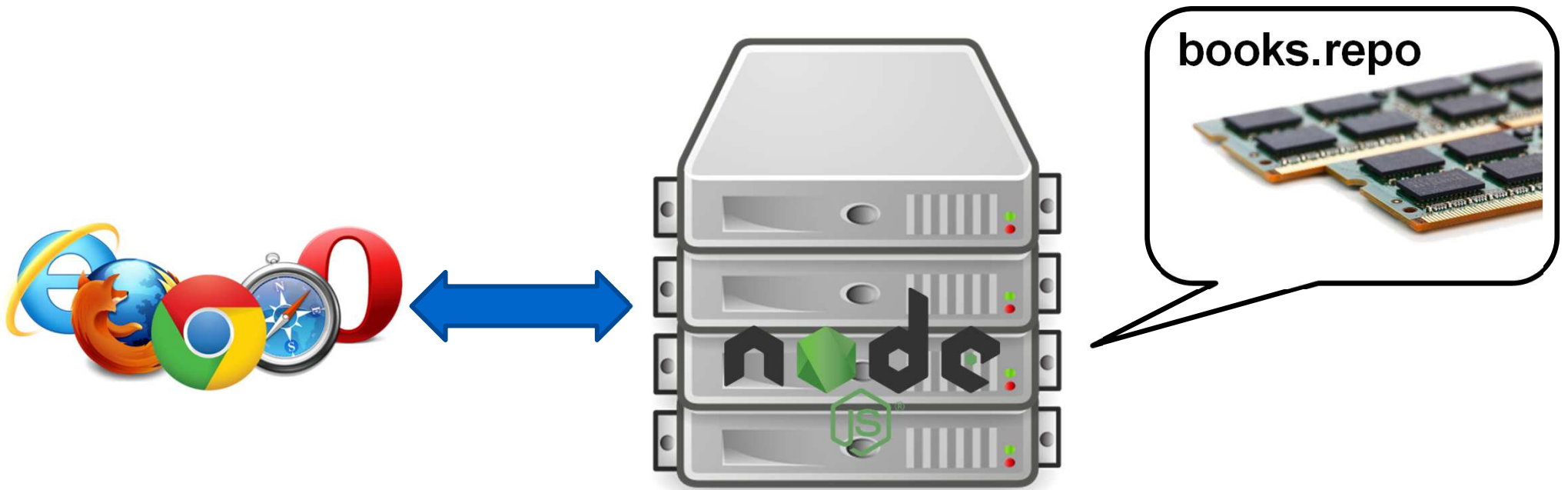
The screenshot shows a web browser window with the title 'My books'. The address bar displays 'localhost:3000/books'. The page content is a list of books, each with a title and a list of details (ISBN, Author, Language, Publisher, and ISBN again). The books listed are:

- ISBN: 0735619670
- 2. JavaScript: The Good Parts
 - Author: [Mr. Douglas Crockford \(65\)](#)
 - Language: English
 - Publisher: O'Reilly Media
 - ISBN: 9780596517748
- 3. The Hitchhiker's Guide to the Galaxy
 - Author: [Mr. Douglas Adams \(68\)](#)
 - Language: English
 - Publisher: Del Rey; Reissue edition
 - ISBN: 1535185554
- 4. The Bell Jar
 - Author: [Mrs. Sylvia Plath \(88\)](#)
 - Language: English
 - Publisher: Harper Perennial Modern Classics;
 - ISBN: 0060837020
- 5. Zastave
 - Author: [Mr. Miroslav Krleža \(127\)](#)
 - Language: Croatian
 - Publisher: Forum 1962.
 - ISBN: 1234567890

At the bottom of the page, there is a link labeled 'Add Book'.

■ Gdje su naši podatci?

- I što se događa kad ponovo pokrenemo poslužitelj?



- U **radnoj memoriji** -> svakim zaustavljanjem poslužitelja ih gubimo
- Prihvatljivo (nekad i poželjno) za vrijeme razvoja i testiranja, ali ne u produkciji

Domaća zadaća

- Zamijenite (tj. napišite dodatni) **books.repo** s **books.db.repo** koji pohranjuje podatke u PostgreSQL bazu podataka
 - Napravite sami bazu podataka i tri tablice – book, author, language, uspostavite integritetska ograničenja
 - Možete ručno unijeti inicijalne zapise, ili putem aplikacije
 - Spajanje na bazu pokazano u prethodnom predavanju
 - Dohvat podataka pokazan u prethodnom predavanju
 - Pogledajte primjer kako se unosi zapis:
<https://node-postgres.com/features/queries>
- Ostatak aplikacije ostaje nepromijenjen!
- Imajući *books.repo* i *books.db.repo*, aplikaciju možete pokretati kako vam odgovara, s trajnom ili privremenom pohranom podataka:
 - -> istražite sami (informativno) pojam ***dependency injection***

Validacija

Provjera ispravnosti podataka



Primijetite trenutno stanje:

New author

First name:

Last name:

Date of Birth:

Gender: ☒ Male ☐ Female


submit

List of authors

localhost:3000/authors

1. id: 201

2. firstName: Douglas

3. lastName: Adams

4. DOB: Tue Mar 11 1952 01:00:00 GMT+0100 (GMT+01:00)

5. gender: m

Mrs.Sylvia Plath (88)

1. id: 202

2. firstName: Sylvia

3. lastName: Plath

4. DOB: Thu Oct 27 1932 01:00:00 GMT+0100 (GMT+02:00)

5. gender: f

Mr. (NaN)

1. id: 203

2. firstName:

3. lastName:

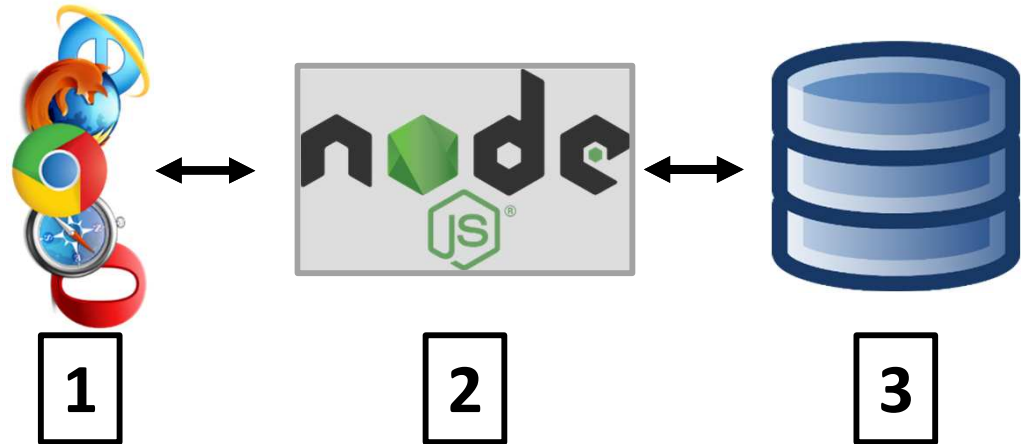
4. DOB: Invalid Date

5. gender: m

[Add author](#)

■ Nedostaje **validacija podataka!**

- **Validacija podatka** je proces čišćenja i **provjere ispravnosti podataka** kako bi podatci bili točni, smisleni i sigurni.
- U općenitoj arhitekturi web-aplikacije, validaciju podataka možemo obaviti na tri mjesta:
 1. Klijent (tipično preglednik)
 2. Poslužitelj (web-aplikacija koja može opet imati N slojeva)
 3. Baza podataka
- Jedno ne isključuje drugo, nerijetko postoji validacija na sva tri mjesta



1. a) Validacija na klijentu (*client-side validation*): HTML

a) Putem HTML-a:

- U novijim verzijama HTML-a postoje **atributi** i **tipovi** input elementa kojima je moguće ograničiti ili zadati tipove podataka čime se sprječava pogrešan unos

Atribut	Objašnjenje	Koristi se za input type=
disabled	Onemogućuje element, vrijednost se ne može promijeniti	bilo koji
maxlength	Definira maksimalni broj znakova	bilo koji
max	Definira maksimalnu vrijednost	Tipično number , ali može i range, date, datetime-local, month, time and week
min	Definira minimalnu vrijednost	
step	Definira dozvoljeni korak, „rezoluciju”	
pattern	Regularni izraz kojim se provjerava ispravnost, moćno i složeno	text, date, search, url, tel, email, password
required	Obavezna vrijednost	bilo koji

1. a) Validacija na klijentu putem HTML-a: primjer

- Prikaz kontrola i mogućnosti ovise o klijentu, prikazana je ista forma u Chromeu i Firefoxu (ima li razlika?):

Novi zapis

Ime: w

Visina: 103 Please lengthen this text to 2 characters or more (you are currently using 1 character).

Visina2:

Datum rođenja: mm/dd/yyyy

Neki trenutak: 01/dd/yyyy --:--:-- x ▾

Submit Reset

March 2020

Sun	Mon	Tue	Wed	Thu	Fri	Sat
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31	1	2	3	4

```
<input type="text" name="ime"
minlength="2" maxlength="30" required>
<input type="number" min="100" max="300"
step="1" name="visina" required>
<input type="range" min="100" max="300"
step="1" name="visina" required>
<input type="date" name="datumRodjenja">
<input type="datetime-local"
name="nekiTrenutak" >
```

Novi zapis

Ime: W

Visina: 112 Please use at least 2 characters (you are currently using 1 characters).

Visina2:

Datum rođenja: mm / dd / yyyy

Neki trenutak:

Submit Reset

■ 1. b) Validacija na klijentu (*client-side validation*): JS

b) Putem Javascripta:

- HTML nije dovoljan, npr. što ako hoćemo provjeriti neko složeno pravilo koje uključuje više od jednog elementa?
- Pišemo Javascript kod!
- Postoje brojne JS knjižnice za validaciju, npr.
<https://www.cssscript.com/best-javascript-form-validator/>

1. b) Validacija na klijentu putem JS-a: primjer (1/2)

- Ovdje koristimo i HTML regex pattern pored JS-a

```
<form action="/into/the/void" method="POST" onsubmit="onSubmit(event)">
<fieldset> (...)
  <input type="text" name="login" id="login" required
    title="Korisničko ime mora počinjati s slovom,
    pattern="(?!^.{3,20}$)^[a-zA-Z][a-zA-Z0-9]*[._-]?[a-zA-Z0-9]+$" >
<!-- regex preuzet s https://regex101.com/r/mF5hM4/1 -->
  <input type="password" id="password1" name="password1" required
    onfocusout="validatePwd()"
    title="Lozinka je dugačka 8-16 znakova, ...broj"
    pattern="^(?=.*\d)(?=.*[A-Z])(?=.*[a-z])(?=.*[^\w\d\s:])([^\s]){8,16}$">
  <input type="password" id="password2" name="password2" required
    onfocusout="validatePwd()"
    pattern="^(?=.*\d)(?=.*[A-Z])(?=.*[a-z])(?=.*[^\w\d\s:])([^\s]){8,16}$">
<!-- regex preuzet s: https://regex101.com/library/0bH043 -->
  (...)
</form>
```

Novi korisnik

Korisničko ime:

Zaporka:

Zaporka ponovo:

1. b) Validacija na klijentu putem JS-a: primjer (2/2)

```
(...)  
</form> </body> <script>  
  function validatePwd() {  
    let pwd1 = document.getElementById("password1");  
    let pwd2 = document.getElementById("password2");  
    if (pwd1.value && pwd2.value && pwd1.value !== pwd2.value) {  
      alert("Zaporke nisu iste!");  
      return false;  
    } else {  
      return true;  
    }  
  }  
  function onSubmit(event) {  
    if (!validatePwd()) {  
      event.preventDefault();  
    }  
  }  
</script>
```

Jednakost lozinki se provjerava i kod gubitka fokusa svake od kontrola, kao i prilikom pokušaja predavanja forme (koji se onda potencijalno programski otkazuje).

Novi korisnik

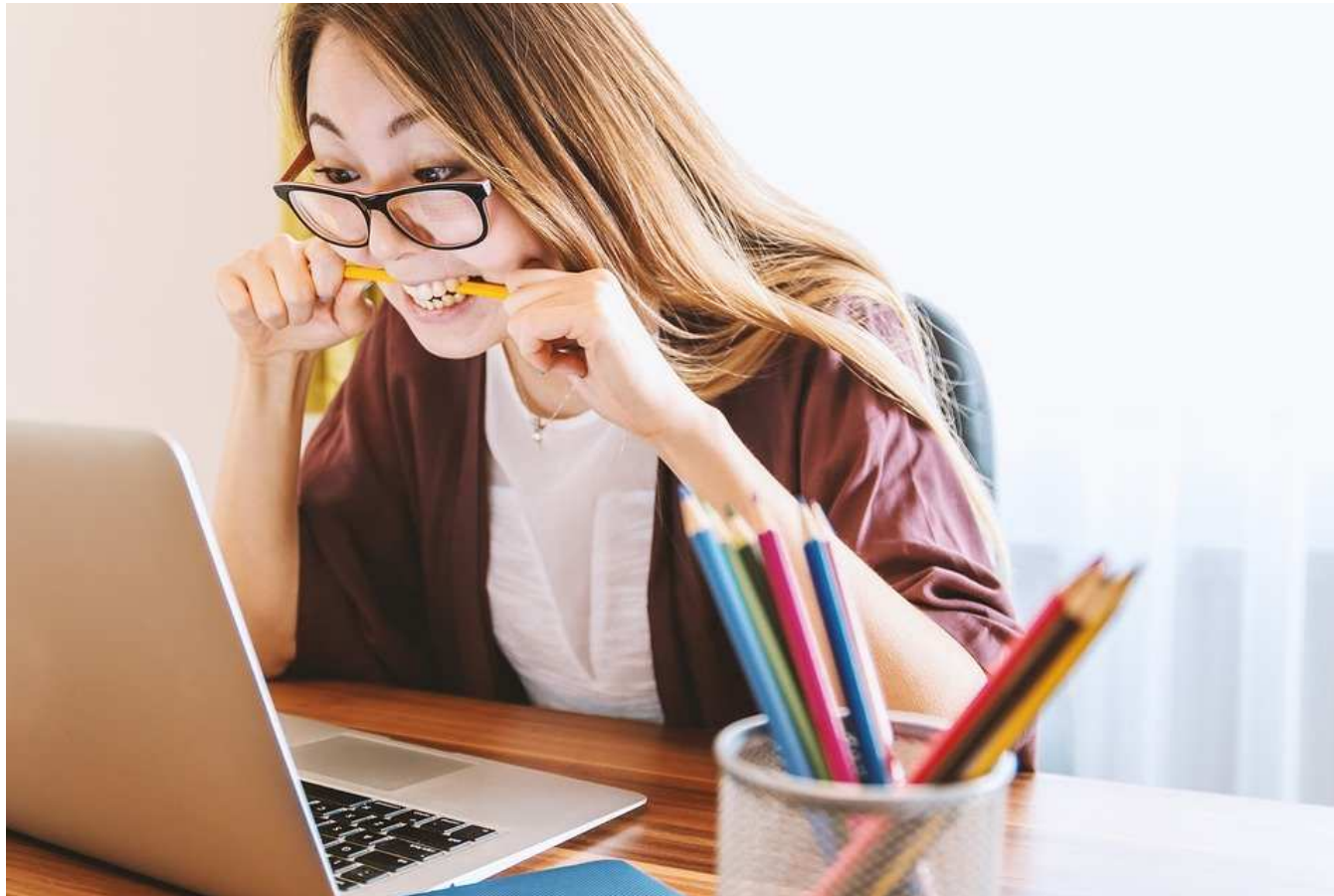
Korisničko ime:

Zaporka:

Zaporka ponovo:

Domaća zadaća

- Promijenite prethodni primjer tako da:
 - Ne koristi `alert()`
 - Poruku da zaporke nisu iste ispisuje pored tih elemenata
 - Kada zaporke nisu iste, staviti im deblji crveni rub



Client vs Server side validation

- Primijetite da je validacija na klijentu:
 - „kozmetička” - nije moguće spriječiti korisnika da pošalje krive vrijednosti
 - ali sa stanovišta **korisničkog iskustva**, *client-side* validacija je **najbolja** jer korisnik ima trenutnu povratnu informaciju, često odmah nakon unosa u neko polje
- Na poslužitelju doista imamo kontrolu i možemo kontrolirano pustiti „dalje” ispravne vrijednosti, odnosno odbaciti neispravne
 - Također, u prilici smo raditi složenije validacije, zapis u kontekstu drugih zapisa i poslovnih pravila, npr. provjera jedinstvenog (postojećeg) korisničkog imena kod registracije
- Problem – dupliciranje koda?

2. Validacija na poslužitelju

- Može se razmatrati kroz dvije faze:
 - *Sanitization*
 - (de)kodiranje, trim, izbacivanje nedozvoljenih znakova, pretvorba u tipove podataka, normalizacija emaila, itd.
 - *Validation*
 - Provjera jednostavnih pravila (atribut popunjen, min/maks duljina, itd.)
 - Provjera složenijih pravila, uključuje više atributa jednog zapisa ili čak u druge zapise
- Mi ćemo koristiti u primjeru express-validator paket:
npm install --save express-validator
- Pogledati primjer na: <https://express-validator.github.io/docs/index.html>

Format:

```
router.post('/add', [  
  //sanit & valid ...  
], function (req, res, next) {  
  // konačno  
});
```

2. Validacija na poslužitelju: primjer – dodavanje knjige

New book

Title:

Author:

Language:

Publisher:

ISBN:

Pogrešan ISBN.
Ali pored toga, želimo
provjeriti i sva ostala polja
da nisu prazna, minimalne i
maksimalne duljine, itd.



New book

Error: [{"value":"1234567890","msg":"ISBN must be a 10 or 13 digit number that confirm to the standard: [https://en.wikipedia.org/wiki/International_Standard_Book_Number](\"https://en.wikipedia.org/wiki/International_Standard_Book_Number\")\";\"param\":\"ISBN\";\"location\":\"body\"}]]

Title:

Author:

Language:

Publisher:

ISBN:

Nema veze što nije
lijepo formatirano – vi
ćete to popraviti za DZ
😊.

2. Validacija na poslužitelju

books.routes.js

```
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 10
or 13 digit number that confirms to
the standard:
https://en.wikipedia.org/wiki/International\_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())
    });
  } else {
    console.log(req.body);
    try {
      let newBook = repo.addBook(
        req.body.title,
        repo.getAuthor(req.body.author),
        req.body.language,
        req.body.publisher,
        req.body.ISBN
      );
      res.redirect('/books
    } catch (err) { ...
```

Zašto više ne
radimo parseInt?

Domaća zadaća

- Promijenite prethodni primjer tako da: prilikom greške:
 - Provjerite još postoji li zadani jezik i autor
 - Testirajte pomoću postmana ili curla ili tako da u devtoolsima promijenite šifru jezika
 - U slučaju pogrešnih vrijednosti:
 - Popravite kod tako da se ne izgube upisane neispravne vrijednosti kao na slici (prije dva slajda)
 - Povežite poruku o pogrešci s odgovarajućim poljem, postavite crveni rub
 - Napravite validaciju kod dodavanja autora



3. Validacija u bazi podataka

- „Zadnja linija obrane”
- I validacija na poslužitelju (web-aplikaciji) se može zaobići
 - Što ako imamo više aplikacija?
 - Što ako netko iz SQL editora napiše "insert into..."?
- Provodi se putem integritetskih ograničenja (primarni i strani ključevi, domenska ograničenja, *check constraints*, okidači)
 - Obradeno na paralelnom predmetu Baze podataka
- Narušeno ograničenje će se manifestirati kao pogreška (iznimka) i potrebno ju je „uloviti” i odgovarajuće obraditi

Domaća zadaća

- Probajte unijeti knjigu nepoznatog jezika ili na neki drugi način narušiti referencijski integritet ili neko drugo ograničenje
 - Uхватite iznimku i prikažite odgovarajuću poruku o pogrešci



Osvrt na pretvorbu podataka

- Primijetimo da u ovom tipičnom lancu imamo problem s pretvorbama **modela i tipova podataka**:

- **1↔2**

- string ↔ string, number, date, itd.
 - string ↔ array, object, ?

- **2↔3**

- objekti ↔ relacije

- Rješenja:

- **1↔2**

- Ručna pretvorba
 - Gotove knjižnice za (de)serijalizaciju (npr. *model binding* u ASP.netu)

- **2↔3**

- Ručno (pisanje SQL naredbi u kodu)
 - ORM (npr. EF u ASP.netu, Sequelize u Node.jsu, itd.)

