# Uvod u korišćenje Canvas API-a

Crtanje proizvoljne grafike unutar HTML dokumenata, može se obaviti korišćenjem jednog posebnog elementa. Reč je o canvas elementu koji obezbeđuje prostor unutar koga je moguće crtati proizvoljnu grafiku. Web pregledači poseduju dva skupa funkcionalnosti koja omogućavaju da se unutar canvas elementa crta grafika. Reč je o Canvas i WebGL aplikativnim programskim interfejsima. Modul pred Vama biće posvećen Canvas API-u, pa ćete tako u narednim lekcijama imati prilike da naučite kako se unutar canvas elementa može crtati 2D grafika.

## canvas element

HTML element canvas koristi se za prikaz proizvoljne grafike unutar HTML dokumenta. canvas element se kreira korišćenjem istoimenog otvarajućeg i zatvarajućeg taga:

```
<canvas></canvas>
```

Korišćenje zatvarajućeg taga je obavezno, za razliku od na primer img elementa koji je moguće kreirati i bez takvog taga.

Sadržaj canvas elementa se dinamički kreira korišćenjem JavaScript jezika i nešto ranije spomenutih Web API-a. Ipak, direktno unutar canvas elementa se može postaviti neki tekst, koji će korisniku biti prikazan ukoliko web pregledač ne podržava canvas element:

```
<canvas>
    Your web browser does not support canvas element.
</canvas>
```

Tekst definisan između <canvas> i </canvas> tagova ostaće nevidljiv za korisnika ukoliko web pregledač podržava canvas element. Svi moderni web pregledači podržavaju canvas element (canvas nije podržan samo unutar Internet Explorer 8 i starijih verzija ovog pregledača).

## Veličina canvas elementa

canvas element podrazumevano ne poseduje vizuelnu reprezentaciju unutar web pregledača. On samo obezbeđuje pravougaoni prostor unutar koga je moguće crtati proizvoljnu grafiku. Većina web pregledača canvas elementu dodeljuje podrazumevanu veličinu koja iznosi 300 x 150px. Veličina canvas-a se zapravo odnosi na dva pojma, pa tako postoji:

- veličina prostora za crtanje, koja se definiše width i height HTML atributima
- veličina canvas elementa unutar HTML dokumenta, koja se utvrđuje na osnovu veličine prostora za crtanje, ali i na osnovu CSS stilizacije koja se može definisati nad canvas elementom

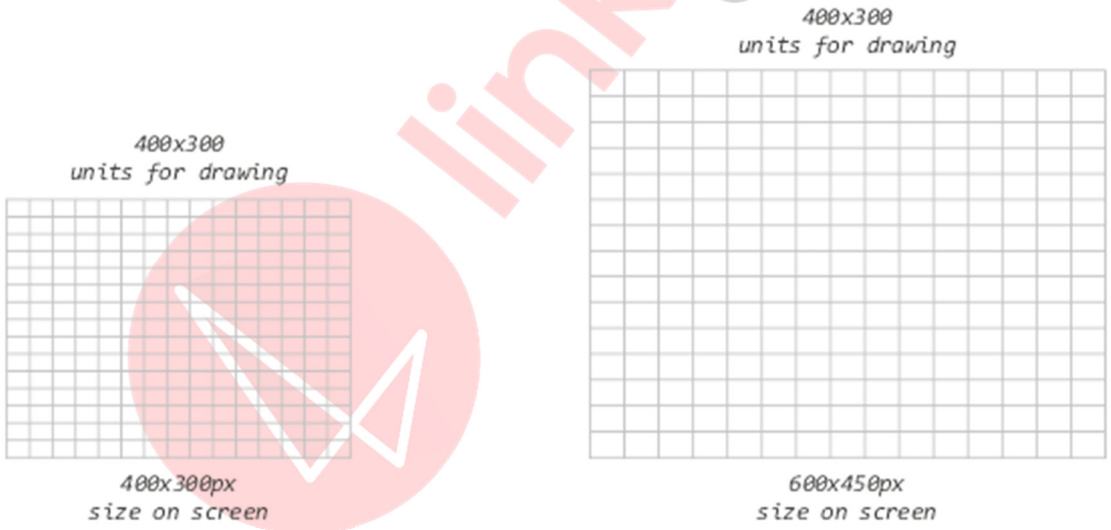
Veličina prostora za crtanje definiše se direktno na canvas elementu korišćenjem width i height atributa:

```
<canvas width="400" height="300">
    Your web browser does not support canvas element.
</canvas>
```

Na ovaj način, obavlja se postavljanje veličine prostora za crtanje unutar canvas elementa. Ukoliko veličina canvas elementa nije definisana korišćenjem CSS jezika, dimenzije postavljene korišćenjem width i height atributa će biti one koje će element imati i unutar HTML dokumenta. Ipak, na veličinu koju će canvas element imati unutar HTML dokumenta, moguće je uticati CSS stilizacijom, pa je tako moguće napisati nešto ovako:

```
canvas {
    width: 600px;
}
```

Sada će canvas element unutar HTML dokumenta imati širinu od 600px i visinu od 450px. Drugim rečima, visina će srazmerno pratiti promenu širine. Ipak, vrlo je bitno razumeti, da i pored većih dimenzija koje će canvas element imati unutar HTML dokumenta, prostor za crtanje će ostati veličine 400x300px. Takav prostor za crtanje će biti uvećan kako bi popunio nešto veći prostor definisan CSS stilizacijom (slika 4.1).




*Slika 4.1 - Razlika između fizičke veličine canvas elementa i prostora za crtanje*

Na slici 4.1 možete videti razliku između fizičke veličine canvas elementa i veličine prostora za crtanje. Analogija se može napraviti sa prikazom slika unutar HTML dokumenata. Veličina prostora za crtanje može se uporediti sa fizičkom veličinom slike, a veličina canvas elementa sa veličinom koju slika ima unutar HTML dokumenta. Sa slike 4.1 možete da vidite da kada je canvas element fizički veći od prostora za crtanje, mreža piksela za crtanje je krupnija nego što bi to bilo bez eksplicitnog definisanja veličine canvas elementa korišćenjem CSS-a.

Prilikom definisanja veličine canvas elementa korišćenjem CSS-a, potrebno je voditi računa. Naime, definisanje veličine canvas elementa korišćenjem CSS-a, može da poremeti proporcije oblika koji se crtaju, pa grafika može biti deformisana. To se može dogoditi kada se pored `width` svojstva, definiše vrednost i svojstva `height`. Tada u slučaju nepodudaranja odnosa stranica, dolazi do deformacije grafike koja se crta unutar canvas elementa.

**Sadržaj canvas elementa je raster**

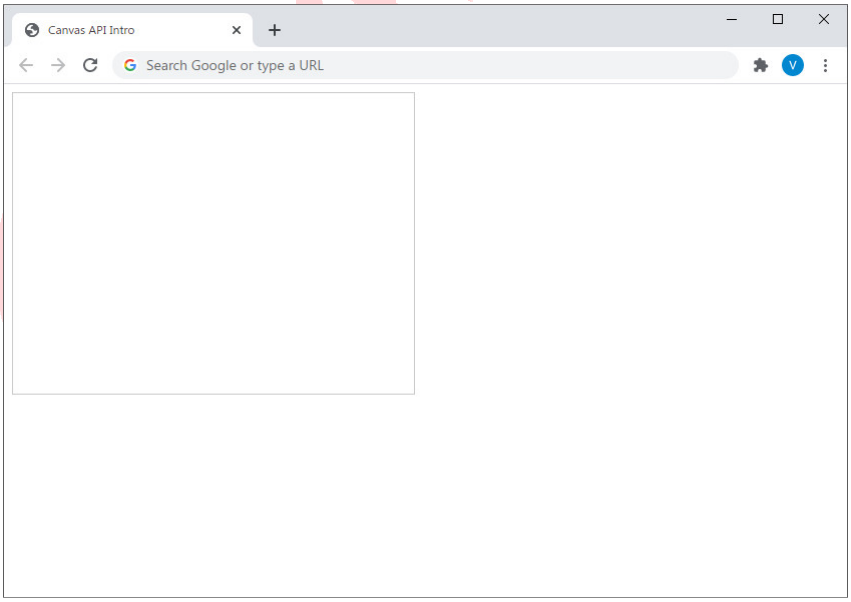
Sadržaj canvas elementa je slika, odnosno rasterska grafika, ali grafika čiji je svaki piksel moguće programabilno kontrolisati korišćenjem JavaScript jezika.

Za sada mi nećemo eksplicitno definisati veličinu canvas elementa korišćenjem CSS-a, već samo veličinu prostora za crtanje upotrebom atributa `width` i `height`. Nešto kasnije, mogućnost definisanja prostora koji će canvas element da zauzme unutar HTML dokumenta, mi ćemo iskoristiti za lako postizanje responsive ponašanja.

S obzirom da naš canvas još uvek ne poseduje nikakav programabilno kreiran sadržaj (grafiku), kako bismo lakše mogli da vidimo njegove granice i poziciju unutar HTML dokumenta, na njega ćemo postaviti okvire:

```
#my-canvas {
   border: #cacaca 1px solid;
}
```

Na ovaj način, unutar pregledača biće dobijen prikaz kao na slici 4.2.




*Slika 4.2 - canvas element unutar HTML dokumenta*

Pravougaonik koji možete videti u gornjem, levom delu prozora web pregledača jeste canvas element koji je trenutno prazan.

## Prostor za crtanje (rendering context)

Nakon kreiranja `canvas` elementa i definisanja njegove veličine, može se preći na crtanje grafike. Ipak, pre nego što budemo u mogućnosti da nacrtamo neku grafiku, neophodno je doći do prostora za crtanje koji se drugačije naziva *rendering context*. Takav posao se može obaviti na sledeći način:

```
<canvas id="my-canvas" width="400" height="300">
    Your web browser does not support canvas element.
</canvas>

<script>
    let myCanvas = document.getElementById("my-canvas");
    var ctx = myCanvas.getContext('2d');
</script>
```

U prikazanom primeru prvo je obavljeno kreiranje promenljive (`myCanvas`) sa referencom na DOM objekt koji predstavlja naš `canvas` element. Nad DOM objektom koji predstavlja `canvas`, zatim je obavljeno pozivanje metode `getContext()`, koja isporučuje traženi kontekst za crtanje.

**Metoda getContext()**

Metoda `getContext()` isporučuje objekat koji predstavlja prostor za crtanje unutar `canvas` elementa. Ova metoda prihvata jedan parametar:

```
getContext(contextType)
```

Parametar koji se prosleđuje metodi `getContext()` određuje tip prostora za crtanje koji će da bude isporučen. Naime, `canvas` element je moguće koristiti za crtanje grafike različitih osobina, o čemu je već bilo reči u dosadašnjem toku ovoga kursa. Mi se na samom početku moramo odlučiti za tip grafike koju ćemo crtati, a u zavisnosti od takve odluke formira sa i vrednost parametra koji se prosleđuje metodi `getContext()`:

- `'2d'` - parametar kojim se dobija podloga za crtanje dvodimenzionalne grafike
- `'webgl'` - parametar kojim se dobija podloga za crtanje trodimenzionalne grafike, hardverski pogonjene od strane OpenGL ES 2.0 biblioteke
- `'webgl2'` - parametar kojim se dobija podloga za crtanje trodimenzionalne grafike, hardverski pogonjene od strane OpenGL ES 3.0 biblioteke
- `'bitmaprenderer'` - parametar kojim se dobija podloga čiji sadržaj će biti moguće zameniti isključivo bitmap grafikom

Mi ćemo se u nastavku baviti crtanjem dvodimenzionalne grafike, te je stoga u primeru, metodi `getContext()` prosleđena vrednost `'2d'`.

## Provera podrške za canvas

Pre crtanja grafike unutar canvas elementa, dobro je obaviti i proveru podrške za canvas element i Canvas API skup funkcionalnosti od strane web pregledača. Nešto ranije je rečeno da svi moderni web pregledači podržavaju canvas element i različite skupove funkcionalnosti za crtanje. Ipak, neki stariji web pregledači tako nešto ne podržavaju, pa je pre crtanja dobro proveriti da li je tako nešto uopšte moguće obaviti. Najjednostavniji način za obavljanje takvog posla je provera raspoloživosti getContext() metode:

```
if (myCanvas.getContext) {
    var ctx = myCanvas.getContext('2d');
    // drawing code here
} else {
    alert("Canvas is not supported.");
}
```

Ukoliko canvas element i pripadajući aplikativni programski interfejsi nisu podržani od strane web pregledača, svojstvo koje ukazuje na metodu getContext() imaće vrednost undefined, pa će samim tim biti izvršen kod unutar else uslovnog bloka. U suprotnom će biti aktiviran uslovni blok if, pa će se izvršiti naredba za dolazak do reference na objekat koji predstavlja kontekst.

**Pitanje**

Dobijanje objekta koji predstavlja kontekst za crtanje, postiže se korišćenjem metode:

**a) getContext()**
b) getRaster()
c) getCanvas()
d) getVector()

**Objašnjenje**

*Metoda getContext() isporučuje objekat koji predstavlja prostor (kontekst) za crtanje unutar canvas elementa.*

## Crtanje prve grafike unutar canvas elementa

Za crtanje grafike unutar canvas elementa važi identično pravilo kao i za programabilnu intervenciju nad bilo kojim drugim HTML elementom - neophodno je da se struktura stranice u potpunosti učita i da web pregledač izgradi objekti model dokumenta (DOM). Stoga dobra praksa nalaže da se logika za crtanje unutar canvas elementa pokrene tek nakon što je DOM učitan. Mi ćemo to obaviti tako što ćemo kompletnu logiku koju smo do sada kreirali upakovati u jednu zasebnu metodu, koja će se aktivirati prilikom emitovanja load događaja, window objekta:

```
window.onload = draw;
function draw() {
    let myCanvas = document.getElementById("my-canvas");

    if (myCanvas.getContext) {

        var ctx = myCanvas.getContext('2d');
        // drawing code here

    } else {
        alert("Canvas is not supported.");
    }
}
```

Sada je kompletna logika za crtanje upakovana unutar funkcije `draw()`. Referenca na nju je dodeljena `onload` svojstvu `window` objekta, pa će na taj način funkcija `draw()` da bude pozvana kada web pregledač kreira objektni model dokumenta i učita sve resurse.

Prvi primer grafike unutar `canvas` elementa podrazumevaće crtanje pravougaonika. Evo kako može da izgleda kod za obavljanje takvog posla:

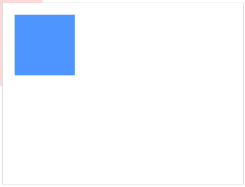
```
function draw() {
    let myCanvas = document.getElementById("my-canvas");

    if (myCanvas.getContext) {
        var ctx = myCanvas.getContext('2d');

        ctx.fillStyle = 'rgb(79, 149, 255)';
        ctx.fillRect(20, 20, 100, 100);

    } else {
        alert("Canvas is not supported.");
    }
}
```

Unutar if uslovnog bloka, nakon naredbe za dobijanje prostora za crtanje, komentar koji je do sada tu postojao, zamenjen je kodom za crtanje jednog kvadrata. Tako je unutar `canvas` elementa dobijen prikaz kao na slici 4.3.



*Slika 4.3 - Kvadrat unutar canvas elementa*

Kvadrat unutar `canvas` elementa koji možete videti na slici 4.3 dobijen je upotrebom dve naredbe, pri čemu obe podrazumevaju upotrebu objekta koji predstavlja kontekst za crtanje:

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

ctx.fillStyle = 'rgb(79, 149, 255)';
ctx.fillRect(20, 20, 100, 100);
```

Prvom naredbom obavljeno je postavljanje boje kojom će biti obojen kvadrat koji želimo da nacrtamo. Možete videti da je boja definisana kao vrednost jednog posebnog svojstva objekta koji predstavlja kontekst crtanja - `fillStyle`.

Drugom naredbom obavljeno je konkretno crtanje kvadrata, metodom `fillRect()`, koja je namenjena crtanju pravougaonika koji imaju ispunu. U primeru smo mi metodu za crtanje pravougaonika, iskoristili za crtanje kvadrata, tako što smo sve stranice učinili jednakim. Naime, prva dva parametra metode `fillRect()`, odnose se na koordinate gornje, leve tačke oblika koji se crta, dok se druga dva parametra odnose na širinu i visinu pravougaonika.

**Kompletan kod primera crtanja prve grafike unutar canvas-a**

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Canvas API Intro</title>
    <style>
        #my-canvas {
            border: #cacaca 1px solid;
        }
    </style>
</head>
<body>
    <canvas id="my-canvas" width="400" height="300">
        Your web browser does not support canvas element.
    </canvas>
    <script>
        window.onload = draw;
        function draw() {
            let myCanvas = document.getElementById("my-canvas");
            if (myCanvas.getContext) {
                var ctx = myCanvas.getContext('2d');
                ctx.fillStyle = 'rgb(79, 149, 255)';
                ctx.fillRect(20, 20, 100, 100);
            } else {
                alert("Canvas is not supported.");
            }
        }
    </script>
</body>
</html>
```

Prikazani kod možete da preuzmete sa sledećeg linka:

`first-canvas-drawing.rar`

## Rezime

- HTML element `canvas` koristi se za prikaz proizvoljne grafike unutar HTML dokumenta
- direktno unutar `canvas` elementa se može postaviti neki tekst, koji će korisniku biti prikazan ukoliko web pregledač ne podržava `canvas` element
- većina web pregledača `canvas` elementu dodeljuje podrazumevanu veličinu koja iznosi 300 x 150px
- veličina prostora za crtanje definiše se direktno na `canvas` elementu korišćenjem `width` i `height` atributa
- ukoliko veličina `canvas` elementa nije definisana korišćenjem CSS jezika, dimenzije postavljene korišćenjem `width` i `height` atributa će biti one koje će element imati i unutar HTML dokumenta
- prostor koji će `canvas` element zauzimati unutar HTML dokumenta, može se definisati korišćenjem CSS svojstava `width` i `height`
- objekat kojim se predstavlja prostor za crtanje `canvas` elementa drugačije se naziva *rendering context*, a dobija se pozivanjem metode `getContext()`
- programabilnu proveru raspoloživosti Canvas API-a, moguće je obaviti proverom dostupnosti metode `getContext()`
- logiku za crtanje unutar `canvas` elementa potrebno je aktivirati tek nakon što se učita kompletan DOM