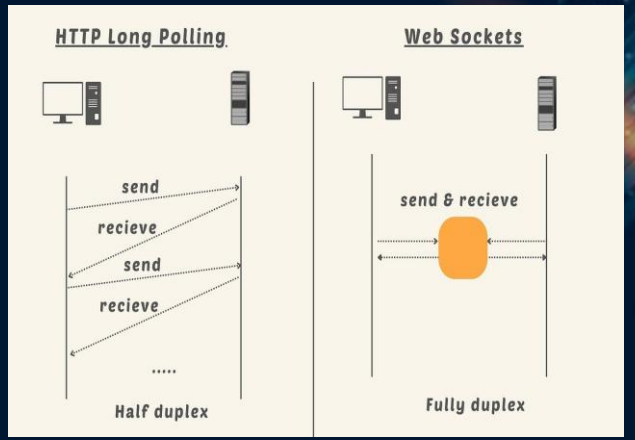
Uvod u Socket.IO

Real-time Multiplayer Tic-Tac-Toe Igra

Anja Cokanović

Šta je Socket.io?

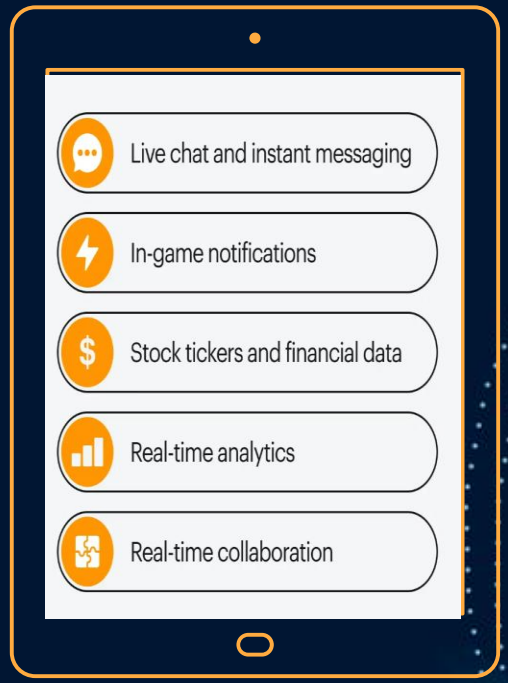
1. Socket.IO predstavlja biblioteku koja omogućava dvosmernu, real-time komunikaciju između web klijenata i servera. Biblioteka se bazira na WebSocket protokolu, ali dodaje dodatne funkcionalnosti poput automatskog fallback-a na alternative (npr. long polling) u slučaju da WebSocket nije podržan.
2. Socket.IO omogućava da se kreira veza između klijenta i servera i da ta veza ostane stalno otvorena.
3. Bez Socket.IO, za svaki pojedinačni zahtev klijent mora ponovo da otvori konekciju ka serveru, server obradi zahtev i zatim pošalje odgovor nazad klijentu. Kada imamo veliki broj zahteva, posebno manjih zahteva, svo to vreme potrebno za stalno otvaranje konekcije se nagomilava i stvara nepotrebno kašnjenje.
4. Sa Socket.IO pristup je drugačiji. Kreira se WebSocket konekcija koja se povezuje sa serverom i ostaje otvorena tokom cele komunikacije. Zahvaljujući tome, moguće je slati više poruka i događaja u realnom vremenu, bez dodatnog troška ponovnog uspostavljanja konekcije.
5. Zbog toga su aplikacije kao što su chat aplikacije ili mini multiplayer igre idealne za korišćenje Socket.IO biblioteke.



Glavne upotrebe socket.io

Socket.IO se široko koristi u real-time aplikacijama kao što su:

- Chat i messaging sistemi
- Multiplayer online igre
- Kolaborativni alati (npr. zajedničko editovanje dokumenata)
- Live notifikacije i dashboard-ovi
- Berzanski tikeri i praćenje podataka u realnom vremenu



Instalacija biblioteka

- Da bismo počeli da koristimo Socket.IO, prvo moramo instalirati potrebne pakete. Ovo radimo u terminalu, u odgovarajućem folderu projekta.
- Server strana (Node.js/Express projekat):

```
npm install socket.io
```

- Opcionalno za debagovanje:

```
npm install @socket.io/admin-ui
```

- Klijentska strana (Angular projekat):

```
npm install socket.io-client
```

POKRETANJE SOCKET.IO SERVERA

- Socket.IO se vezuje za port na kojem server radi, odnosno definiše se na kom portu će socket slušati konekcije.

- Kreira se običan HTTP server pomoću http.createServer().
- Socket.IO se inicijalizuje kao nova instanca klase Server i vezuje direktno za taj HTTP server.
- U CORS podešavanjima navedeni su dozvoljeni origini: Angular aplikacija i admin panel.
- credentials: true omogućava slanje kolačića i autentifikacionih headera ako bude potrebno.
- pingInterval: 10000 i pingTimeout: 5000 definišu koliko često Socket.IO proverava da li je veza živa i koliko dugo čeka odgovor pre nego što proglasi konekciju prekinutom.
- instrument(io, { auth: false }) odmah aktivira admin panel za nadzor i debagovanje (dostupan na http://localhost:8080/admin), više na nekom od narednih slajdova.
- Server sluša na portu 8080 – na ovom portu Socket.IO čeka dolazne konekcije od klijenata.

```
const http = require('http');
const { Server } = require('socket.io');
const { instrument } = require('@socket.io/admin-ui');

const server = http.createServer();

const io = new Server(server, {
  cors: {
    origin: ["https://admin.socket.io", "http://localhost:8080"],
    credentials: true
  },
  pingInterval: 10000,
  pingTimeout: 5000
});

instrument(io, { auth: false });

server.listen(8080, () => console.log('Server radi na http://localhost:8080'));
```

POVEZIVANJE KLIJENTA I CONNECTION EVENT

- Na klijentu se importuje io iz biblioteke socket.io-client i poziva se sa adresom servera.
- Na taj način se dobija individualni socket za tog klijenta.
- Funkcija io.on('connection') se poziva svaki put kada se novi klijent poveže na server. U tom trenutku se za tog klijenta kreira poseban socket objekat. Svaki socket automatski dobija svoj jedinstveni identifikator socket.id.
- Metoda socket.on se koristi za definisanje eventa na koji klijent ili server reaguje. Na klijentu se, na primer, može slušati connect event, koji se aktivira kada se socket uspešno poveže.
- Klijentsa strana:

```
import { io } from 'socket.io-client';

const socket = io('http://localhost:8080');  // podrazumevani namespace "/"
```

- Serverska strana:

```
io.on('connection', (socket) => {
  console.log('Novi klijent povezan:', socket.id);
});
```

Najvažniji default eventi

CONNECT

Aktivira se na klijentu kada je konekcija uspešno uspostavljena

CONNECTION

Aktivira se na serveru za svakog novog klijenta

DISCONNECT

Aktivira se i na klijentu i na serveru kada veza pukne

CONNECT_ERROR

RECONNECT/RECONNECT_ATTEMPT

Aktiviraju se prilikom automatskog ponovnog povezivanja

CUSTOM EVENTS

- Metoda socket.on se koristi za definisanje eventa na koji klijent ili server reaguje.
- Za slanje događaja sa klijenta ka serveru koristi se socket.emit. Event može da ima proizvoljno ime i može da nosi bilo kakve podatke. Na serveru se zatim definiše odgovarajući socket.on koji reaguje na taj custom event i kao parametre prima sve podatke koje je klijent poslao.
- Eventi se mogu slati svaki put kada se na klijentu desi neka akcija. Na primer, kada igrač odigra potez u igri, klijent odmah šalje event serveru sa informacijom o tom potezu. Server zatim dalje prosleđuje te podatke ostalim klijentima.

CUSTOM EVENTS

- Slanje event-a sa klijenta:

```
// Pridruživanje sobi
this.socket.emit('joinRoom', { room, username: name });

// Odlazak iz sobe
this.socket.emit('leaveRoom');

// Odigravanje poteza
this.socket.emit('makeMove', { index });
```

- Primanje I reakcija na serveru:.

```
socket.on('joinRoom', ({ room, username }, ack) => { ... });

socket.on('leaveRoom', (ack) => { ... });

socket.on('makeMove', ({ index }, ack) => { ... });
```

- Imena eventa ('joinRoom', 'leaveRoom', 'makeMove') su proizvoljno izabrana i moraju biti ista na klijentu i serveru.
- Uz event se šalju podaci u obliku objekta (npr. { room, username } ili { index }).
- Na serveru socket.on sluša tačno to ime eventa i prima poslate podatke kao parametre.
- Custom eventi omogućavaju da se precizno definiše šta se dešava pri određenoj akciji korisnika (klik na dugme, odigravanje poteza, itd.).
- Server nakon obrade obično prosledi informacije dalje drugim klijentima ili vrati potvrdu

NAMESPACES

- Socket.IO takođe omogućava kreiranje namespace-ova.
- Na serveru se može kreirati novi namespace, na primer /room, i za njega definisati poseban io objekat.
- Na taj način se može logički odvojiti komunikacija.
- Za namespace-ove se može koristiti i middleware pomoću use, gde se pozivom next() dozvoljava prelazak na sledeći korak, što je korisno za autentifikaciju i autorizaciju.

NAMESPACES

- Server strana:

```
io.of('/game').on('connection', (socket) => {
  console.log('Igrač povezan:', socket.id);

  // Svi listeneri (joinRoom, makeMove, leaveRoom, disconnect...)
  // važe samo za klijente koji su se povezali na /game namespace
});
```

- Klijent strana:

```
this.socket = io('http://localhost:8080/game', { ... });
```

- Podrazumevani namespace je "/" – ako se ne navede ništa posle porta, klijent se povezuje na koren.
- U projektu je kreiran poseban namespace /game kako bi sva komunikacija vezana za igru bila logički odvojena.
- io.of('/game') na serveru vraća poseban io objekat koji obrađuje samo konekcije na putanji /game.
- Klijent se povezuje direktno na http://localhost:8080/game i time automatski ulazi u taj namespace.
- Svi custom eventi (joinRoom, makeMove, notification, gameUpdate...) razmenjuju se isključivo unutar ovog namespace-a.
- Namespaces omogućavaju da se u istoj aplikaciji imaju različiti delovi komunikacije (npr. /game, /chat, /admin) bez mešanja.

SLANJE PORUKA OD SERVERA – UNUTAR NAMESPACE-A I SOBA

- Podrazumevano, emit šalje poruku svim povezanim klijentima, uključujući i onog koji je poslao poruku.
- Ako želimo da se poruka pošalje svim ostalim klijentima osim onog koji je poslao, koristi se socket.broadcast.emit.
- Socket.IO omogućava i rad sa sobama (rooms). Svaki korisnik je inicijalno u sopstvenoj sobi koja ima naziv njegovog socket.id.
- Ako želimo da pošaljemo poruku samo određenom korisniku ili grupi korisnika, možemo koristiti socket.to(room).emit, gde će svi klijenti unutar te sobe dobiti poruku. Ovo već implicitno radi kao broadcast unutar sobe, bez slanja poruke klijentu koji ju je poslao.

NAMESPACES

- Primeri iz projekta (sve unutar /game namespace-a)::

```
// Slanje SVIMA u sobi (uključujući pošiljaoca)
io.of('/game').to(room).emit('gameUpdate', game);

// Slanje SVIMA u sobi OSIM pošiljaocu
socket.broadcast.to(room).emit('notification', `${username} se pridružio sobi ${room}`);

// Slanje SVIMA u namespace-u (bez obzira na sobu)
io.of('/game').emit('neki-globalni-event', data);
```

- Klijent strana- primanje poruka:

```
this.socket.on('gameUpdate', (game: any) => {
  // Ažuriranje table, igrača, poteza...
});

this.socket.on('notification', (msg: string) => {
  this.notifications.push(msg);
});
```

- Svi emit-ovi u projektu koriste io.of('/game') ili socket.broadcast kako bi poruke stizale samo klijentima koji su povezani na /game namespace.
- Kada igrač pridruži sobu (socket.join(room)), sve dalje poruke vezane za igru šalju se samo unutar te sobe pomoću .to(room).emit().
- gameUpdate se šalje svima u sobi da bi se sinhronizovalo stanje table i igrača.
- notification se šalje broadcast-om da pošiljalac ne dobija svoju poruku (npr. “username se pridružio”).
- Na taj način se osigurava da poruke o igri stižu samo relevantnim igračima u istoj partiji

SLANJE ZAHTEVA OD KLIJENTA KA SERVERU I RAD SA SOBAMA

- Na klijentu može postojati dugme, na primer joinRoomButton, koje šalje event join-room.
- Na serveru postoji listener za taj event, a pozivom socket.join(room) klijent se dodaje u određenu sobu.
- Jedan klijent može istovremeno biti član više soba.
- Svaki korisnik je inicijalno u sopstvenoj sobi koja ima naziv njegovog socket.id.

SLANJE ZAHTEVA OD KLIJENTA KA SERVERU I RAD SA SOBAMA

- Slanje zahteva sa klijenta za pridruzivanje sobi:

```
joinRoom(room: string) {
  const name = this.username?.trim();
  if (!name) return alert('Unesite ime');

  this.socket.emit('joinRoom', { room, username: name }, (res: any) => {
    if (res.status === 'joined') {
      this.currentRoom = room;
      // Ažuriranje stanja igre...
    }
  });
}
```

- Obrada zahteva na serveru I dodavanje u sobu:

```
socket.on('joinRoom', ({ room, username }, ack) => {
  // Provere: username obavezan, jedinstven, soba nije puna...

  socket.join(room);                    // Dodavanje klijenta u sobu
  socket.data.username = username;      // Čuvanje podataka o korisniku
  socket.data.room = room;

  game.players.push(username);

  if (ack) ack({ status: 'joined', board: game.board, players: game.players });

  socket.broadcast.to(room).emit('notification', `${username} se pridružio sobi ${room}`);
  io.of('/game').to(room).emit('gameUpdate', game);
});
```

- Klijent šalje custom event 'joinRoom' sa podacima o sobi i korisničkom imenu.
- Server proverava uslove (username nije prazan, nije zauzet, soba nije puna) i zatim poziva socket.join(room) čime se klijent dodaje u naznačenu sobu.
- Nakon pridruživanja, server obaveštava sve u sobi o novom igraču i šalje ažurirano stanje igre.
- Slično radi i za 'makeMove' – klijent šalje zahtev sa indeksom polja, server proverava validnost i ažurira stanje.
- Klijent može napustiti sobu slanjem 'leaveRoom' eventa, nakon čega server čisti stanje

CALLBACK FUNKCIJE (ACKNOWLEDGEMENTS)

- Socket.IO podržava i callback funkcije. Callback se prosleđuje kao poslednji parametar funkciji i uvek mora da bude poslednji argument. On se može koristiti da server obavesti klijenta da je poruka uspešno poslata ili da se klijent uspešno pridružio sobi.
- U projektu se callback koristi prilikom pridruživanja sobi i odigravanja poteza kako bi klijent odmah dobio povratnu informaciju o uspehu ili grešci akcije, bez čekanja na broadcast poruke.

CALLBACK FUNKCIJE (ACKNOWLEDGEMENTS)

- Sa klijenta:

```
this.socket.emit('joinRoom', { room, username: name }, (res: any) => {
  if (res.status === 'joined') {
    this.currentRoom = room;
    this.board = res.board;
    this.players = res.players;
  } else if (res.status === 'full') {
    alert('Soba je puna!');
  } else if (res.status === 'taken') {
    alert('Ovo ime je već zauzeto u sobi');
  }
});
```

- Na serveru:

```
socket.on('joinRoom', ({ room, username }, ack) => {
  // Provere...

  if (/* greška */) return ack?.({ status: 'full' });   // ili 'taken', 'error'...

  socket.join(room);
  // ... dodavanje igrača u game

  ack({ status: 'joined', board: game.board, players: game.players });
});
```

NADZOR I DEBAGOVANJE – ADMIN UI

- Za nadzor i debagovanje može se koristiti @socket.io/admin-ui. Na serveru se instalira paket i uključuje pomoću instrument(io, { auth: false }).
- Zatim je moguće pristupiti admin interfejsu preko /admin na URL-u. U CORS origin listu na serveru mora se dodati https://admin.socket.io kako bi admin UI mogao da se poveže.
- U admin panelu se mogu videti svi aktivni WebSocket-i, uključujući i onaj koji admin UI sam kreira kao poseban socket.

```
const { instrument } = require('@socket.io/admin-ui');

const io = new Server(server, {
  cors: {
    origin: ["https://admin.socket.io", "http://localhost:8080"],
    credentials: true
  },
  // ... ostala podešavanja
});

instrument(io, { auth: false });
```

Alternative

Čisti WebSocket (npr. ws paket u Node.js):
Osnovni protokol bez dodatnih slojeva. Zahteva ručnu implementaciju reconnections, fallback-a i drugih funkcija.

SignalR:
Microsoft-ova biblioteka, optimizovana za .NET ekosistem. Nije idealna za čisti JavaScript/Node.js projekat.

SockJS:
Pruža sličan fallback mehanizam, ali nema ugrađene koncepte poput rooms i namespaces.

Firebase Realtime Database / Firestore:
Cloud-based servisi sa jednostavnom integracijom, ali donose dodatne troškove, zavisnost od treće strane i manju kontrolu nad serverom.

Zašto socket.io?

Socket.IO je izabran jer pruža najbolji balans između:

- **jednostavnosti** (intuitivan event-based API, lako emitovanje i slušanje događaja bez ručnog parsiranja poruka)
- **bogatog seta funkcija** (rooms, namespaces, automatski reconnections, heartbeat mehanizam za detekciju prekida veze)
- **fleksibilnosti u JavaScript/Node.js okruženju**, bez eksternih zavisnosti
- **pouzdanosti i kompatibilnosti** (automatski fallback na HTTP long-polling ili druge transporte u slučaju da WebSocket nije dostupan, npr. iza restriktivnih proxy-ja ili firewall-ova)
- **lakog skaliranja osnovnih scenarija** (built-in podrška za broadcasting, acknowledgements i multiplexing, što štedi vreme u razvoju)

  Ovi dodatni razlozi su posebno važni jer, iako su native WebSockets brži i lakši u 2025. godini (gotovo svi browseri ih podržavaju >99%), Socket.IO i dalje štedi **mnogo vremena** na implementaciji feature-a koje bi inače morao pisati ručno (npr. reconnect logika, detekcija mrtvih konekcija, grupisanje klijenata u rooms).

Zaključak

- Socket.IO predstavlja moćnu i pristupačnu biblioteku za razvoj real-time web aplikacija.
- U ovom projektu demonstrirana je njena primena kroz multiplayer Tic-Tac-Toe igru:
  - Brza i pouzdana dvosmerna komunikacija
  - Jednostavno rukovanje sobama (rooms), konekcijama i događajima
  - Real-time ažuriranje stanja igre za više igrača
- Socket.IO omogućava brzi razvoj kompleksnih interaktivnih aplikacija poput chatova, igara i kolaborativnih alata, čineći ga idealnim izborom za moderne web projekte.

Hvala na pažnji!