UNIVERZITET U NOVOM SADU
FAKULTET TEHNIČKIH NAUKA

# INDUSTRIJSKI KOMUNIKACIONI PROTOKOLI U EES

PROJEKATNA DOKUMENTACIJA

**Autori:**

Biljana Vukelić, PR20/2016
Milica Medić, PR22/2016

**Asistent:**

Ognjen Jelisavčić

# Sadržaj

# *UVOD*

Sistem se sastoji iz dve osnovne komponente: ***servera*** *i* ***neodređenog broja igrača***, i predstavlja igru pogađanja zamišljenog broja.

Na početku, server igračima omogućava registraciju u vidu unosa njihovog username-a. Kako bismo znali koliko igrača će da učestvuje u igri, prvi klijent koji se registruje, dobija poruku da unese koliko tačno igrača želi. Nakon toga, klijent šalje serveru interval u kom se zamišljeni broj nalazi.

Ukoliko je broj igrača zadovoljen, server obaveštava ostale igrače da je igra počela i u kom opsegu se nalazi zamišljeni broj.

Igrači kreću ta pogađaju broj tako što svoje predloge šalju serveru. Taj predlog server prosleđuje prvom klijentu tj igraču koji je zamislio broj i od njega dobija odgovor „veće“, „manje“ ili „tačno“. Kada server dobije odgovor „tačno“, broj je pogođen i igra je gotova.

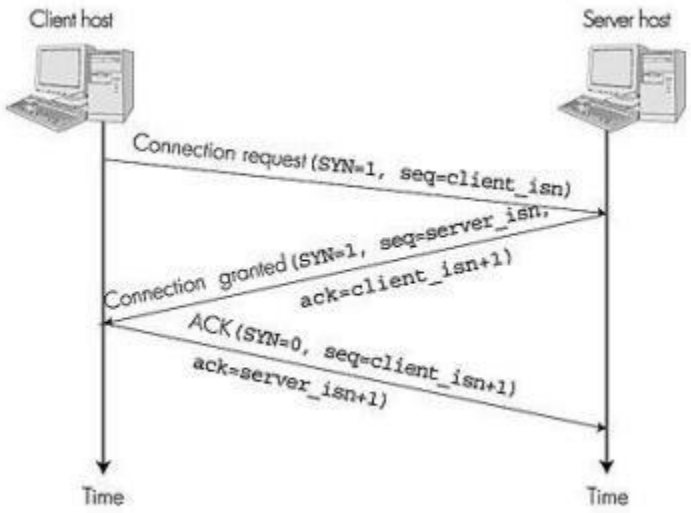
Cilj zadatka je pronaći algoritam pogađanja brojeva, tako da je broj iteracija između igrača i servera što manji.

# *DIZAJN*

U ovom poglavlju ćemo opisati dizajn implementiranog rešenja kao i komponente koje se nalaze u sistemu.

U opisu same aplikacije, gde je zadatak bio raviti rešenje za igru pogađanja zamišljenog broja bilo je potrebno da sistem sačinjavaju dve komponente. Dve osnovne komponente na koje smo se bazirali su server i neograničen broj klijenata.

Između servera i klijenta je korišćen TCP (*Transmision control protocol*) protokol , koji ima ulogu da obezbedi pouzdan i efikasan prenos podataka. .Kao inicijator komunikacije, klijent mora da zna osnovne podatke o serveru kao što je njegova IP adresa, dok server ne mora znati ništa o klijentu pre uspostavljanja komunikacije, već samo čeka da mu se neki klijent prvi obrati. Kako bi se uspostavila komunikacija, klijent i server moraju napraviti sopstveni socket (jedan kraj komunikacionog kanala). Socket je određen IP adresom i brojem porta. Server čeka zahteve klijenata tako da sluša na određenom portu socketa. Za svaku konekciju, server kreira thread, koji opslužuje tog klijenta. Nakon što se uspostavi veza, server i klijent mogu međusobno slati podatke u oba smera.




Slika 1. Klijent-Server komunikacija

Da bi server mogao da komunicira sa više klijenata, potrebno je da čuva sokete od istih. Naša implementacija podrazmeva da server u listi čuva bitne informacije svakog klijenta

i na taj način ostvaruje komunikaciju sa njima.
Thread-ovi  će se vrteti sve dok klijent ne prekine konekciju ili dok server ne signalizira završetak rada programa

**Razlozi navedenog dizajna:**

A. Odvajanje klijenta i server u dva projekta – svaka funkcionalnost u posebnom projektu, uspostava TCP konekcije, mogućnost komunikacije između komoponenti
B. Svakog klijenta osplužuje poseban thread na serveru- konkuretno izvršavanje zadataka na serveru, brže obrađivanje zahteva, usklađen rad klijenata.

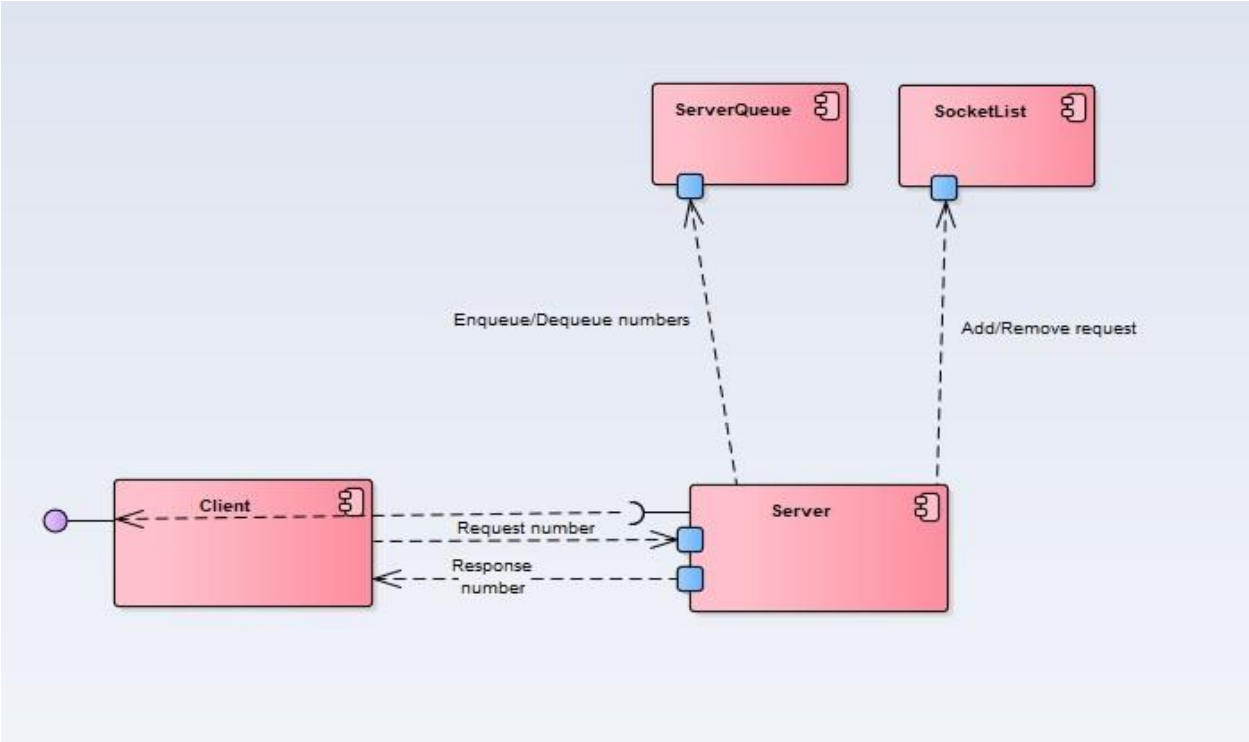
## 2.1 Dijagram komponenti

Dijagram komponenti modeluje strukturu softvera, ukazujući i na zavisnosti između tih softverskih komponenata.  Glavna **namena** dijagrama komponenti je da prikaže **relacije između komponenti sistema.**

Ideja je da se komponenta može lako zameniti, ponovo upotrebiti, jer enkapsulira ponašanja i implementira određeni interfejs**.**

Komponenta može da ima sledeće elemente:

- **Interfejse** - Predstavlja deklaraciju skupa operacija i obaveza
- **Relacije zavisnosti -** relacija gde jedan element zahteva drugi element za svoju potpunu implementaciju
- **Portove** - predstavlja tačku interakcije između komponente i njenog okruženja
- **Konektore** - Povezuje dve komponente, eksterni deo komponente sa internom strukturom




Slika 2.Dijagram komponenti sistema

Na slici2. prikazan je dijagram komponenti sistema, koji se sastoji od sledećih komponenti

- Komponenta *server* – zahteva interefejs komponente klijent, komunicira sa klijentom, a podatke smešta u Listu I Queue
- Komponenta *serverQueue* – omogućava komponenti server da na nju smešta podatke (predloge brojeva od igrača)
- Komponenta *socketList* – omogućava komponenti server da u njoj čuva sokete klijenata
- Komponenta *klijent* – implementira svoj interfejs, komunicira sa serverom tako šalje svoje predloge brojeva

## 2.2 Use-Case Dijagram

Dijagram slučajeva korišćenja (use-case) prikazuje skup slučajeva korišćenja i ušesnika. Tipično se koristi da specificira neku funkcionalnost i ponašanje nekog subjekta.

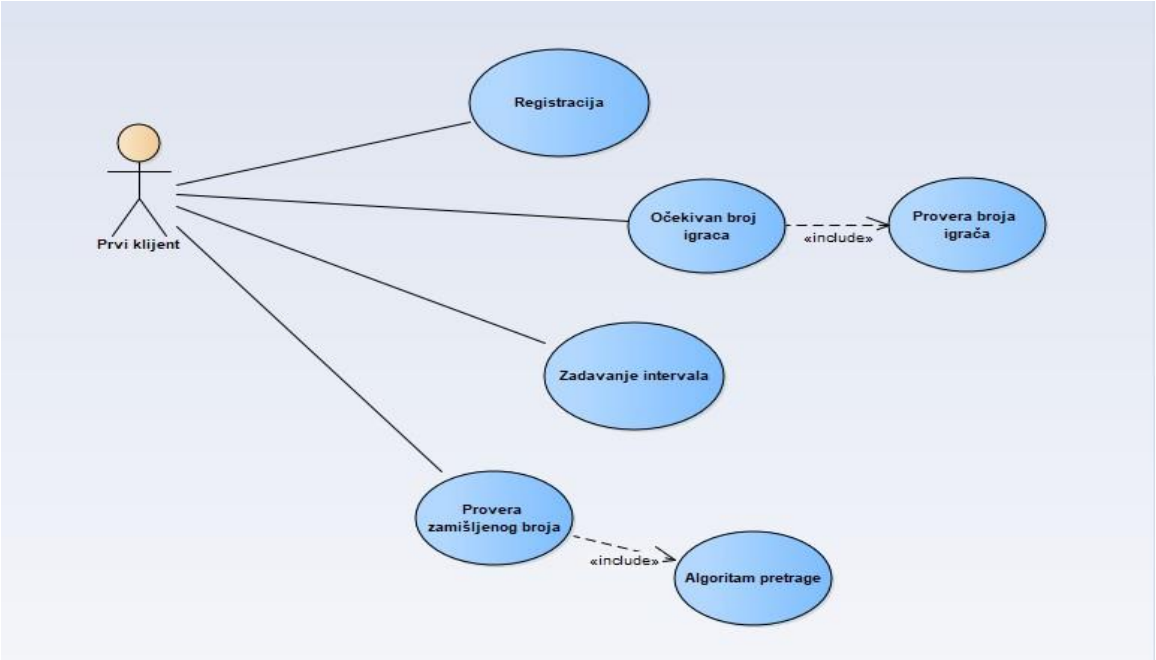
Elementi dijagrama su:

- **Slučajevi korišćenja**
- **Učesnici**
- **Relacije – veze zavisnosti**
- **Paketi**

Napomenuli smo da se sistem sastoji iz dve osnovne komponente: servera i neodređenog broja klijenata.

Sada ćemo klijente podeliti na dve podvrste:

1. ***Glavni klijent*** – u našem slučaju je to klijent koji se prvi registruje. Samo on ima mogućnost zadavanja broja igrača u datoj igri. Pored toga, ima mogućnost zamišljanja broja tj zadavanja intervala u kom se zamišljeni broj nalazi, kao i mogućnost provere dobijenih vrednosti od ostalih igrača, uz odgovarajući algoritam opisan u tekstu koji sledi.




Slika 3. Use Case dijagram za „glavnog klijenta“

### “Podeli pa vladaj” (*divide and conquer*)

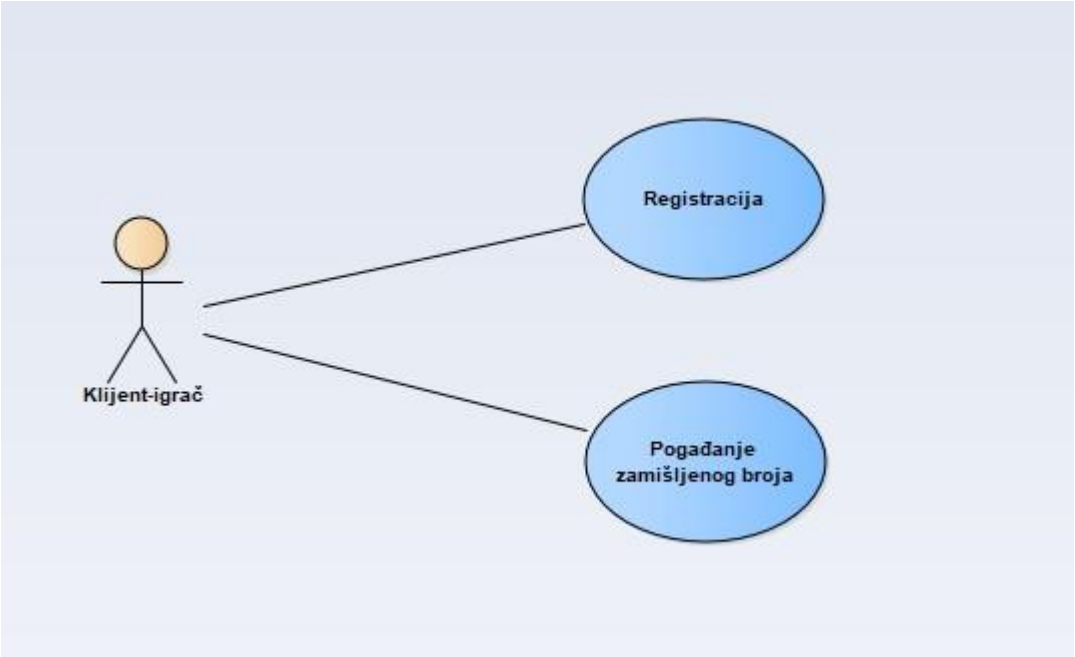
**Binarno pretraživanje je nalaženje zadate vrednosti u sortiranom skupu elemenata. U svakom koraku, dok se ne pronađe tražena vrednost, skup se deli na dva dela i pretraga se nastavlja samo u jednom - odbacuje se deo koji sigurno ne sadrži traženu vrednost.**

Cilj zadatka je bio pronaći algoritam pogađanja brojeva, tako da je broj iteracija između igrača i servera što manji, u diskretnom slučaju, ako skup ima n (konačno mnogo) elemenata, algoritam je vremenski logaritamske složenosti — O(log n), gde je n širina polaznog intervala. Binarno pretraživanje je daleko efikasnije nego linearno, ali zahteva da su podaci koji se pretražuju uređeni, može se implementirati iterativno ili rekurzivno. U ovom slučaju, implementirano je iterativno rešenje binarne pretrage

**Koraci algoritma:**

- **Najveći opseg intervala + najniži opseg intervala -1** = nam daje **broj mogućih rešenja**,
- Nakon toga, taj broj **delimo sa 2** kako bismo pronašli odgovarajuću sredinu, a server postavlja pitanje da li je to on zamišljeni broj?
- Ako je odgovor koji dobije od glavnog klijenta da nije, tačnije da je **niži** od zamišljenog formula po kojoj radimo sledeću iteraciju,ali sada sa promenjenim opsegom intervala: najviši opseg = prethodna mogućnost - 1
- Ako je odgovor koji dobije od glavnog klijenta da nije, tačnije da je **viši** od zamišljenog formula po kojoj radimo sledeću iteraciju,ali sada sa promenjenim opsegom intervala: najniži opseg = prethodna mogućnost + 1
- Postupak se ponavlja sve dok server ne dobije odgovor od glavnog klijenta da je broj tačan. U tom slučaju, igra je gotova.

2. ***Ostali klijenti*** – igrači koji nakon registracije imaju mogućnost pogađanja broja koji je glavni klijent zamislio




Slika 4. Use Case dijagram za „ostale klijente“ – igrače

# *STRUKTURE PODATAKA*

U najopštijem smislu, termin *struktura podataka* koristi se za način organizacije određenih postupaka u programu.

## 3.1 Queue – red

Redove je najjednostavnije objasniti kroz primer čekanja reda u pošti. Prva osoba koja stane u red najmanje će čekati na šalteru, što predstavlja početak reda (*head*), a poslednja koja se nalazi u redu će najduže da čeka što predstavlja kraj reda(*tail*).

Redovima se implementira **FIFO *(first-in first-out*)** princip.

Dve osnovne operacije nad redovima:

- **Enqueue -** dodavanje člana na kraj reda
- **Dequeue** - brisanje elementa sa početka reda

```
struct Queue
{
    int value;
    int socket;
    Queue* next;
};
void Enqueue(Queue** head, Queue** tail, int number, int socket);
int Dequeue(Queue** head);
```

Slika 5. Definicija funkcija Dequeue/Enqueue i Queue

Nakon što klijenti pošalju svoje predloge brojeva serveru, on ih smešta u red pomoću funkcije **Enqueue**. Parametri date funkcije su:

- *Head* – početak reda (*glava*)  *Tail* – kraj reda (*rep*)
- *Number* – broj koji smeštamo
- *Socket* – redni broj klijenta koji je poslao dati broj

Funkcija **Enqueue** se sastoji iz dva slučaja:

- *Red je bio prazan* pre dodavanja novog elementa
- *Red nije bio prazan* pre dodavanja novog elementa

U prvom slučaju, glava (head) reda je pokazivač koji ima vrednost NULL(ne pokazuje ni na jedan element). Nakon dodavanja novog elementaa, *head* i *tail* postaju pokazivači na taj element. U drugom slučaju, prilikom dodavanja novog elementa u red koji nije bio prazan, *head* već pokazuje na prvi element, dok *tail* sada pokazuje na novi element koji je dodat na kraj reda.

Nakon što poslednji klijent pošalje svoj predlog broja, server prvi skida sa reda uz pomoć funkcije **Dequeue**.
Funkcija Dequeue briše element iz reda, takođe ima dve mogućnosti kao i funkcija Enqueue. U prvom slučaju , ako *head* pokazuje na pokazivač koji ima vrednost NULL, ispisaćemo poruku da je red već prazan.
Dok u drugom slučaju kada *head* pokazuje na neki element, brisanje se vrši od početka tj briše se prvi element reda. Nakon uspešnog brisanja elementa, *head* pokazuje na sledećeg u redu.

**Zašto Queue?**

---

***Jednostavan je za implementaciju a pogodan za rešenja gde je neophodno ispratiti redosled slanja podataka.***
**U queue je moguće push-ovati poruke onim redosledom kojim stižu na sam kraj queue-a, a pop-ovati sa njegovog početka. Na taj način glavnom klijentu uvek šaljemo poruku koja je prva stigla od igrača S obzirom da pop vrši skidanje sa queue-a, manje je vremena potrebno za preuzimanje poruka i "brisanja" u odnosu na druge strukture podataka.**

---

## 3.2 LISTA

Liste su najjednostavnije dinamičke strukture i dele se na:

- Jednostruko spregnute liste
- Dvostruko spregnute liste
- Višestruke liste
- Kružne liste

U realizaciji ovog projekta, korišćena je jednostruko spregnuta lista. Server na svojoj strani kreira listu koja se sastoji od soketa i pokazivača na sledećeg u istoj. Na taj način serveru je omogućena komunikacija sa više od jednog klijenta, jer se u listi čuvaju soketi od klijenata koji žele da uspostave konekciju. Za svakog klijenta pokreće se novi thread.

Element liste sastoji se od dva polja:

- podatak (u ovom slučaju socket)
- pokazivač na sledeći element liste

```
struct ListaSocket
{
    SOCKET socket;
    struct ListaSocket* next;
};
void InitList(ListaSocket** head);
void AddToList(ListaSocket** head, SOCKET s);
SOCKET ElementAt(ListaSocket* head, int index);
```

Slika 6. Definicija funkcija initList,AddToList,ElementList i struktura liste

Funkcija ***AddToList*** omogućava dodavanje elementa u našem slučaju socket-a u listu. Prilikom dodavanja moramo obratiti pažnju na dva slučaja:

- ***Lista je bila prazna*** pre dodavanja novog socket-a
- ***Lista nije bila prazna*** pre dodavanja novog socket-a

Pre dodavanja socket-a u praznu listu, glava (head) liste je pokazivač koji ima vrednost NULL(ne pokazuje ni na jedan element). Nakon dodavanja novog socket-a, glava liste postaje pokazivač na taj socket.
U drugom slučaju, lista nije prazna i glava pokazuje na prvi socket. Da bi se dodao novi socket na kraj liste, potrebno je proći kroz sve elemente liste kako bi se pronašao poslednji u istoj, a zatim iza njega dodao novi socket.

Funkcija ***ElementAt*** pronalazi socket od željenog klijenta. Prvi parametar te funkcije predstavlja početak liste, dok drugi predstavlja redni broj željenog klijenta tačnije ID njegovog soketa.

**Zašto LISTA?**

---

**Jednostruko spregnuta lista je veoma fleksibilna struktura. Pre svega, element se može dodati i ukloniti sa bilo kog mesta. Najveća prednost spregnute liste je ta što može da sadrži promenljivi broj čvorova, u našem slučaju broj klijenata koji se očekuju pri započinjanju igre.**
**Jedna lista čuva sve bitne informacije o svim klinetima**
**To nam omogućava da uvek možemo da nađemo klijenta koji nam je potreban i sve njegove bitne informacije. Ukoliko se klijent ugasi, tek tada se njegove informacije brišu iz liste. To je razlog zašto je korišćena baš lista kao struktura.**

---

# *POTENCIJALNA UNAPREĐENJA*

Jedna od mogućih opcija potencijalnog unapređenja opisanog projekta je implementacija stress testova. Stress testiranje je oblik testiranja performansi, gde se namerno testira sistem pod maksimalnim opterećenjem. Obično se stress testovi koriste u sistemima sa prosleđivanjem poruka