# Monitorizarea traficului

Spilevoi Bogdan-Vasile[1]

Universitatea "Alexandru Ioan Cuza", Facultatea de Informatică
bspilevoi@yahoo.com

**Abstract.** Acest raport descrie implementarea unui sistem distribuit de tip client-server cu o mapă personalizată pentru proceasrea unei retele de informatii pentru participantii la trafic. Lucrarea prezintă arhitectura aplicației, tehnologiile utilizate, implementarea și rezultatele obținute. Sistemul demonstrează o abordare simplă, dar extensibilă, pentru gestionarea datelor într-un mediu distribuit.

**Keywords:** Client-Server · Mapă personalizată · Socket-uri TCP · Limbaj C. · API · JSON/XML · SQLite · GTK4

## 1 Introducere

Proiectul are ca scop dezvoltarea unei aplicații distribuite pentru gestionarea unei retele intre soferi, unde acestia primesc si ofera informatie despre trafic,cat si informatii optionale despre vreme, sport sau preturi la statiile peco, utilizând o mapă personalizată implementată în limbajul C. Soluția este menită să răspundă cerințelor de bază ale unui sistem client-server: fiabilitate, modularitate și extensibilitate.

Problema abordată este implementarea unui protocol de citire al datelor de la client la server cat si de la server la client dar si cea a procesarii acestor date si al extragerii informatiilor din formatarea aleasa.

Metodologia proiectului include utilizarea protocolului TCP pentru fiabilitatea comunicării și o structură simplă de mapă pentru gestionarea eficientă a datelor. Acest raport detaliază toate etapele, de la arhitectură la rezultate.

## 2 Tehnologii Aplicate

Aplicația folosește următoarele tehnologii și concepte:

- **Socket-uri TCP**: Asigură o comunicare fiabilă între client și server. TCP a fost ales datorită mecanismelor sale de retransmisie automată și verificare a ordinii pachetelor.
- **Limbajul C**: Alegerea limbajului C a fost justificată de nevoia unui control detaliat asupra resurselor și performanței.
- **Structuri de date personalizate**: Mapa implementată este concepută pentru a facilita operații rapide pe un set mic de date.

- **API Request**: Programul apeleaza 3 servicii API prin libraria Curl pentru a procura informatii despre Vreme, Meciuri de forbal si preturi la statiile Peco.
- **XML/JSON parse**: Informatiile primite din API sunt de tipul XML sau JSON, acestea fiind parsate fie prin librarii precum cJson sau manual pentru extragere informatii din XML.
- **Non-relational DB**: Baza de date este folosita pentru a stoca preferintele de noutati ale utilizatorului prin intermediul SQLite
- **Client GUI**: Clientul este prezentat cu o interfata grafica cu care interactioneaza, createa cu libraria GTK4.

TCP a fost selectat în detrimentul UDP pentru a elimina pierderile de date într-un sistem critic. Structurile de date personalizate adaugă flexibilitate pentru extinderea funcționalității pe viitor.

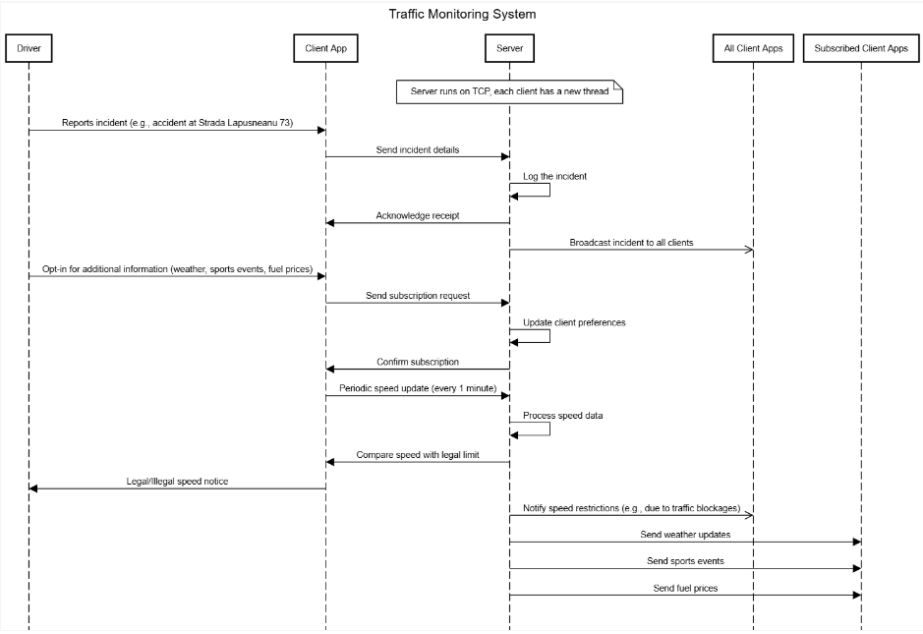
# 3 Structura Aplicației

## 3.1 Arhitectura generală

Aplicația este compusă din trei module principale, folsoind concepte precum servirea concurenta a clientilor, dar si concepte de procesare in protocolul de citire al inofrmatiilor transmise:

- **Client** (`client.c`): Se conectează la server printr-un socket TCP, trimite comenzi ce reprezinta fie actualizari ale starii curente in trafic, fie informatii despre un accident ce a avut loc pe strada curenta.
- **Server** (`server.c`): Gestionează conexiunile multiple, procesează cererile și apelează funcțiile mapei personalizate.
- **Mapă** (`mymap.c`): Oferă metode pentru manipularea perechilor cheie-valoare.

### 3.2 Diagramă UML




## 4 Aspecte de Implementare

### 4.1 Protocolul de comunicare

Protocolul specific client-server include următorii pași:

1. Clientul inițiază o conexiune TCP către server.
2. Serverul acceptă conexiunea.
3. Clientul trimite o cerere formatată (`ID-command-info1-info2-`).
4. Serverul procesează cererea și răspunde cu rezultatul operatiei (daca operatia incercata a avut succes sau nu).

### 4.2 Apelarea API

Apelarea API este realizată prin intermediul librăriei Curl si urmează acesti pași:

1. Realizarea unui cont pe platforma aleasă.
2. Procurarea cheii API.
3. Folosirea parametrilor și al endpoint-ului corect.
4. Parsarea datelor primite și folosirea lor in contextul programului.

### 4.3 Cod reprezentativ

Exemplu de cod pentru manipularea unei conexiuni client:

```
1
2  void *treat(void * arg)
3  {
4     int client;
5     struct sockaddr_in from;
6     bzero (&from, sizeof (from));
7
8     printf ("[thread]- %d - pornit...\n", (int) arg);fflush(
      stdout);
9
10    for(;;)
11    {
12       int length = sizeof (from);
13       if ((client = accept (sd, (struct sockaddr *) &from, &
         length)) < 0)
14       {
15          perror ("[thread]Eroare la accept().\n");
16       }
17
18       while(respond(client,(int)arg) != -1);
19       close (client);
20       threadsPool[(int)arg].thCount++;
21    }
22 }
```

### 4.4 Gestionarea datelor în mapă

Exemplu de cod pentru adaugarea unei chei:

```
1  void insert(HashMap *map, const char *key, void *value) {
2      unsigned int index = hash(key);
3      Entry *current = map->table[index];
4      printf("{HashMap} Inserting at hash [%d].\n", index);
5
6      while (current != NULL) {
7          if (strcmp(current->key, key) == 0) {
8              current->value = value;
9              return;
10         }
11         current = current->next;
12     }
13
14     Entry *new_entry = create_entry(key, value);
15     if (!new_entry) return;
16
```

```
    new_entry->next = map->table[index];
    map->table[index] = new_entry;
}
```

### 4.5 Protocolul de citire din client

Server-ul se asigura ca fiecare informatie trimisa e postfixata de delimitatorul | pentru o citire sigura in client:

```
for(;;)
   {
      bzero(msg, sizeof(msg));
      while(1)
      {
         rd = read(sd, &ch, 1);

         if (rd <= 0)
         {
            perror ("[client]Eroare la read() de la server.\n
            ");
            return errno;
         }
         if(ch != '|')
            strncat(msg, &ch, 1);
         else
            break;

      }
      printf("Message received: [%s]\n", msg);
      fflush(stdout);
   }
```

### 4.6 Apelare API

Cod reprezentativ pentru apelarea unui endpoint API si parsarea JSON ului primit (vreme):

```
void GetWeather(const char * city, char * resTxt)
{
    CURL *curl;
    CURLcode res;

    struct ResponseData response;
    response.data = malloc(1);
    response.size = 0;

    curl_global_init(CURL_GLOBAL_DEFAULT);
    curl = curl_easy_init();

```

```
13      if (curl) {
14          char api[200]  = "\0";
15          sprintf(api, "https://api.tomorrow.io/v4/weather/
            realtime?location=%s&units=metric&apikey=
            fvPZnoEC8Ev8yEl9f4XxwTtZ8FMDTeBi", city);
16          curl_easy_setopt(curl, CURLOPT_URL, api);
17          curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION,
            write_callback);
18          curl_easy_setopt(curl, CURLOPT_WRITEDATA, (void *)&
            response);
19
20          res = curl_easy_perform(curl);
21          if (res != CURLE_OK) {
22              fprintf(stderr, "curl_easy_perform() failed: %s\n
                ", curl_easy_strerror(res));
23          } else {
24              // Parse JSON response
25              cJSON *json = cJSON_Parse(response.data);
26              if (json == NULL) {
27                  fprintf(stderr, "Error parsing JSON\n");
28              } else {
29                  cJSON *data =
                    cJSON_GetObjectItemCaseSensitive(json, "data"
                    );
30                  if (data != NULL) {
31                      cJSON *values =
                        cJSON_GetObjectItemCaseSensitive(data, "
                        values");
32                      if (values != NULL) {
33                          cJSON *precipitationProbability =
                            cJSON_GetObjectItemCaseSensitive(
                            values, "precipitationProbability");
34                          cJSON *temperature =
                            cJSON_GetObjectItemCaseSensitive(
                            values, "temperature");
35                          cJSON *visibility =
                            cJSON_GetObjectItemCaseSensitive(
                            values, "visibility");
36
37                          if (precipitationProbability != NULL
                            && temperature != NULL) {
38                              sprintf(resTxt, "Rain: %.1f%,
                                Temp: %.1fC, Visibility: %.0fkm",
                                 precipitationProbability->
                                valuedouble, temperature->
                                valuedouble, visibility->
                                valuedouble);
39                          }
40                      }
41                  }
```

```
                cJSON_Delete(json); // Clean up
            }
        }

        curl_easy_cleanup(curl);
    }

    free(response.data);
    curl_global_cleanup();
}
```

### 4.7 Crearea GUI pentru client

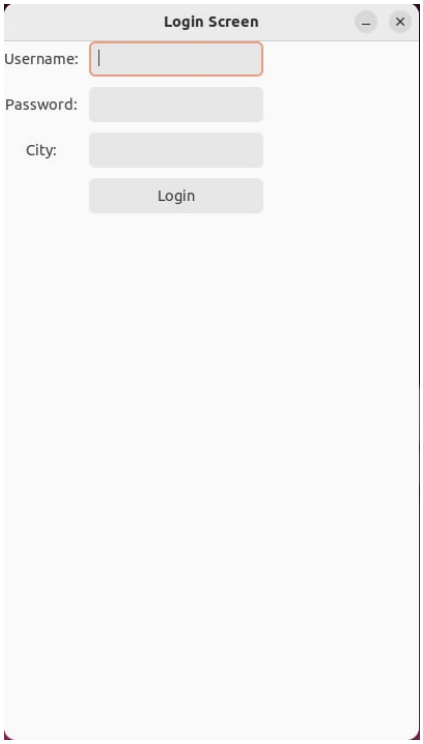
Interfata grafica este realizata cu GTK4, avand o fereastra de LogIn si una pentru aplicatia propriu zisa:

```
void insert(HashMap *map, const char *key, void *value) {
    unsigned int index = hash(key);
    Entry *current = map->table[index];
    printf("{HashMap} Inserting at hash [%d].\n", index);

    while (current != NULL) {
        if (strcmp(current->key, key) == 0) {
            current->value = value;
            return;
        }
        current = current->next;
    }

    Entry *new_entry = create_entry(key, value);
    if (!new_entry) return;

    new_entry->next = map->table[index];
    map->table[index] = new_entry;
}
```





Un exemplu al rularii aplicatiei (ecranul Log In) si aplicatia in sine (cu cele 3 preferinte care vor fi actualizate cu cele ale utilizatorului din baza de date, ele putand fi si oprite dupa preferinte). Orasul este necesar atat pentru informatiile despre vreme dar si pentru cele despre preturile la carburant la statiile peco din jur. Aplicatia totodata recunoaste daca utilizatorul merge cu o viteza regulamentara pe strada curenta, avand optiunea si de a notifica un accident pe strada sa, fapt ce este transmis la toti utilizatorii de pe acea strada.

### 4.8 Utilizare in scenarii reale

Aplicatia poate fi folosita in scenariul real al traficului, intrucat aceasta ajuta participantii la trafic sa fie informati despre legislatiile pe sectiunile de drum pe care circula, dar si despre eventuale accidente, raportate de alti participanti.

## 5 Concluzii

Aplicația a demonstrat o funcționalitate robustă pentru gestionarea datelor și a oferit o bază pentru extinderea ulterioară. Pe viitor, îmbunătățiri precum suportul pentru o baza de date, o interfata sugestiva și optimizarea mapei pentru seturi mari de date ar adăuga valoare suplimentară proiectului.

## 6 Referințe Bibliografice

## References

1. Alexandru Ioan Cuza University, Faculty of Computer Science `https://edu.info.uaic.ro/computer-networks`
2. API informatii vreme `https://app.tomorrow.io/home`
3. API informatii sport `https://sportradar.com/media-tech/data-content/sports-data-api/`
4. API informatii preturi benzina `https://monitorulpreturilor.info/`