

Management de taskuri

Echipa:

Iovu Vali-Cristian

Puscalau Robert-George

Sandu Diana-Elena

Stefan Andrei

Grupa: 1309B

Software Requirements Specification

(IEEE standard)

1.Introducere

1.1.Scop

Scopul acestui document este de a realiza o descriere detaliata in ceea ce priveste proiectul la care am ales sa lucram: "*Management de taskuri*".

1.2. Conventiile documentului

Acest document a fost creat pe baza template-ului *IEEE* pentru *System Requirement Specification*.

1.3. Scopul produsului

Proiectul "Management de taskuri", in linii scurte, este o "unealta" prin intermediul careia mai multe persoane care fac parte din acelasi LAN, pot avea acces la aceeaasi baza de date unde pot prelucra datele. Aceste persoane sunt impartite in doua categorii: user si admin, in consecinta, ei primind drepturi in functie din categoria din care fac parte.

1.4. Audienta

Programul este destinat userilor tipici, precum studenti, care sunt interesati sa afle modul de gestionare a unei baze de date in cadrul mai multor dispozitive asociate acesteia.

1.5. Referinte

<https://www.atlassian.com/software/jira>

<https://www.atlassian.com/agile/kanban>

2. Descriere

2.1. Perspectiva produsului

Pentru a imparti datele intre mai multe dispozitive diferite, am ales sa lucram cu "*SQL Server Express*". In cadrul acestuia am incapsulat mai multe date, precum ar fi: taskurile, userii, cu caracteristicile aferente. Acestea pot fi observate in figurile de mai jos:

Task	
🔑	IdTask
	IdUser
	Tip
	Status
	Continut
	Nota
	TimpEstimat
	LogTime
	Comment

User	
🔑	IdUser
	Username
	Password
	Rights

Userul dispune de urmatoarele proprietati:

- IdUser: un identificator unic
- Username/Password: datele de logare ale contului atribuite in momentul crearii unui cont de user de catre admin
- Rights: poate avea valorile: 0(*ADMIN*), 1(*USER obisnuit*)

Si nu in cele din urma, *task-ul*:

- IdTask: un identificator unic
- IdUser: id-ul userului caruia i-a fost atribuit task-ul in cauza
- Tip: *Bug/Dev...*
- Status: *CODE REVIEW/IN PROGRESS/TO DO/DONE*

- Continut: descrierea task-ului
- Nota: se acorda in functie de dificultatea task-ului
- TimpEstimat: durata aproximativa pana la finalizarea task-ului
- LogTime: timpul pe care user-ul il acorda rezolvarii task-ului
- Comment: comentarii referitoare la ce a reusit ori greutati a intampinat pe parcursul rezolvarii task-ului

2.2. Functionalitatea produsului

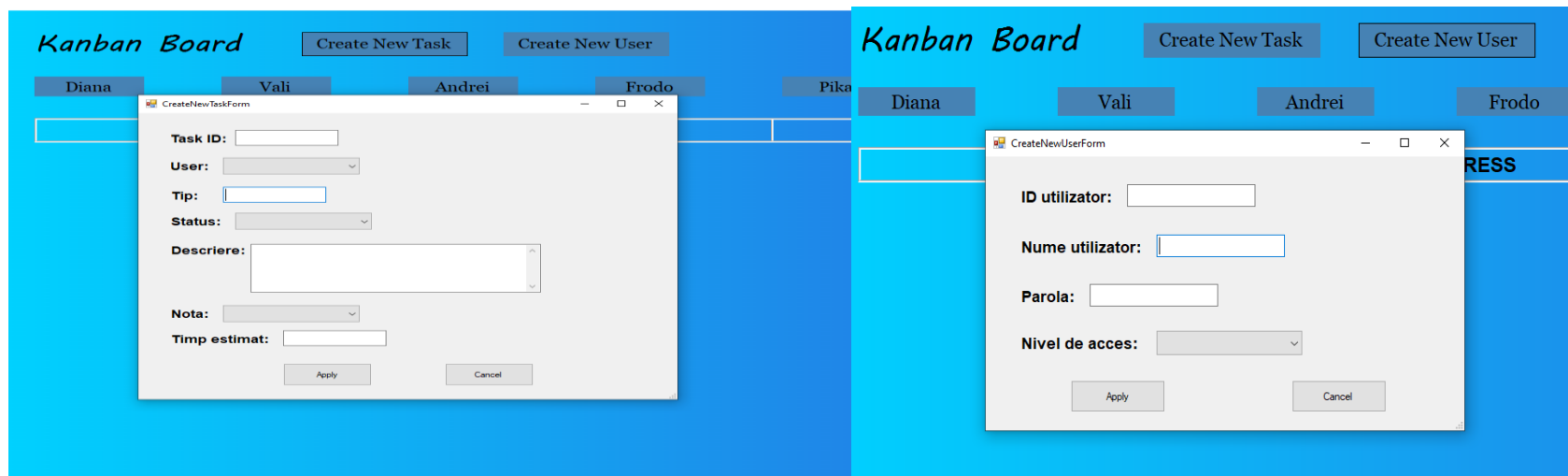
Dupa cum am stabilit si mai sus, exista doua categorii de utilizatori: admin si userii obisnuiti, iar acestia au drepturi in functie de categoria din care fac parte.

Un *user obisnuit* poate dispune urmatoarele functionalitati:

- de a face update oricarui task existent in sectiunea sa de taskuri
- de a inregistra numarul de ore alocate taskurilor la care lucreaza in prezent
- sa vizualizeze taskurile oricarui user
- sa poata adauga comentarii referitor la evolutia taskurilor
- sa poata nota taskul in functie de dificultatea pe care a avut-o

Un *admin*, in plus fata de un user obisnuit, dispune urmatoarele functionalitati:

- sa creeze un nou task si sa-l atribui oricarui user doreste, inclusiv lui
- sa faca update oricarui task existent in baza de date
- sa adauge un nou user



Proiectul nostru este constituit din *doua parti*: cea care se ocupa cu manipularea bazei de date, a cererilor clientilor, mai exact, si cea care opereaza asupra ei. In momentul in care dorim sa facem modificari asupra bazei de date, cum ar fi: adaugare task/user, update task, eliminare task, se va trimite o cerere de la client inspre server utilizand serializare TCP, unde cererea va fi codificata dupa niste reguli bine stabilite, iar in server va fi parsata si tratata corespunzator. Analog, se procedeaza si viceversa. Voi anexa in partea de jos a documentatiei parti din cod unde sunt tratate cele prezentate.

In momentul in care se realizeaza modificari de catre admin in cadrul sectiunii de task al unui user, acesta va primi in scurt timp o notificare. In caz ca se realizeaza aceasta modificare in cadrul sectiunii al altui user, nu va mai primi notificarea, insa asta nu este o problema. E de ajuns doar ca acesta sa dea un refresh si va observa schimbarile facute. Am realizat acest lucru folosind un background worker care verifica periodic, la un timp prestabilit, schimbari in cadrul sectiunii personale de taskuri. La fel cum am precizat si mai sus, voi atasa mai jos parti din cod semnificative.

2.3. Mediul de operare

-Sistem de operare: Windows

- Platforma: Microsoft Visual Studio
- Baza de date locala
- Sistem de tip Client/Server

2.4. Constrangeri

Proiectul a fost dezvoltat în C#(.NET). Acesta a fost construit depinzând de serverul SQL Express. Asadar implementarea a fost centralizată în mod exclusiv pe manipularea bazei de date. În consecință am întâmpinat următoarea situație: cum va manageria serverul să răspundă la cel puțin doi clienți în momentul în care interogările vor fi generate în același timp.

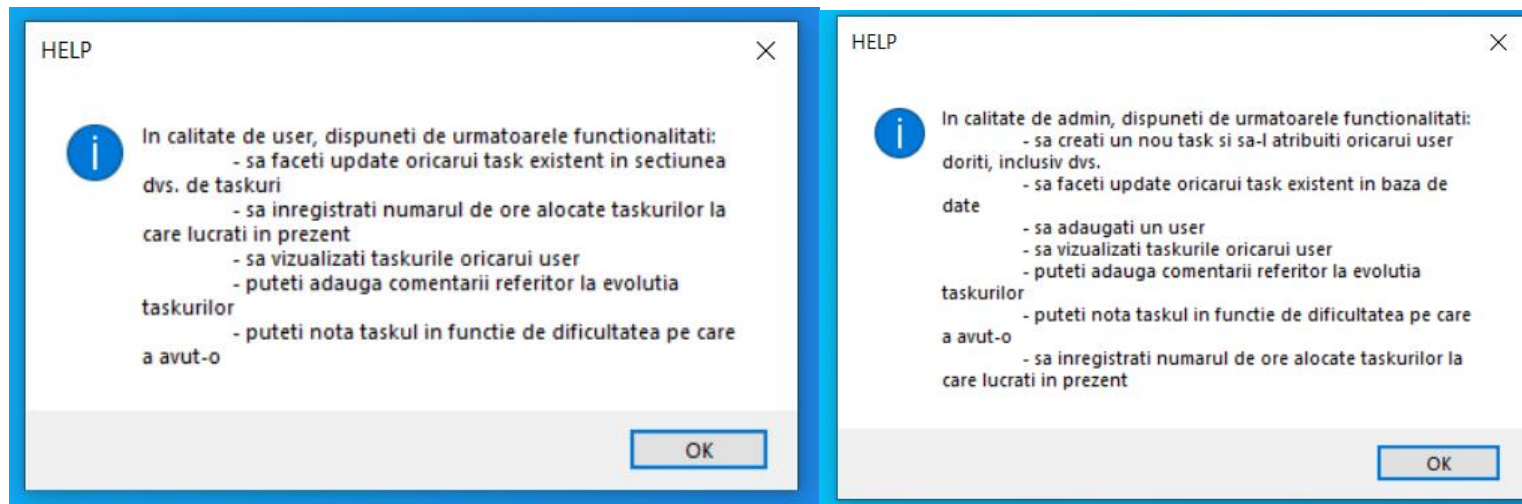
2.5. Ipoteze și dependente

"Management task" este un proiect dezvoltat în C# care utilizează SQL Server Express, prin urmare e necesar ca userul să îl aibă instalat. E necesar ca acesta să ruleze doar de pe un singur dispozitiv. În cadrul proiectului se găsește un txt unde se află comenzile de creare a bazelor de date, tabelor aferente taskurilor și userilor care trebuie rulate în linia de comandă a serverului sql.

3.Cerinte de sistem

3.1. Programul are un help asociat

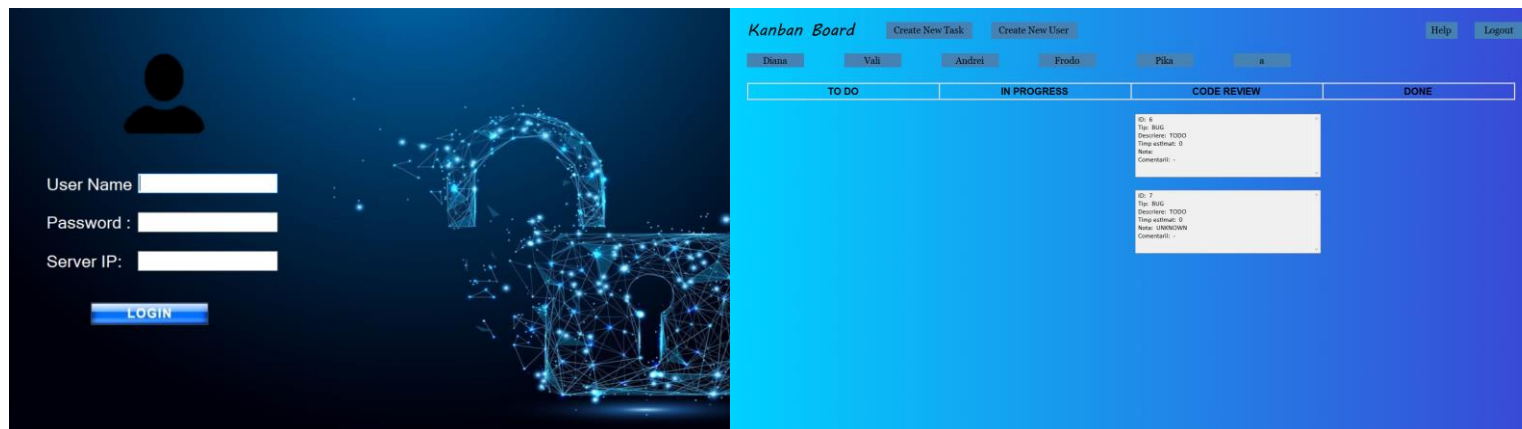
În funcție de drepturile de acces ale utilizatorului, avem două help-uri pe care le putem accesa apăsând butonul aferent de pe interfață.



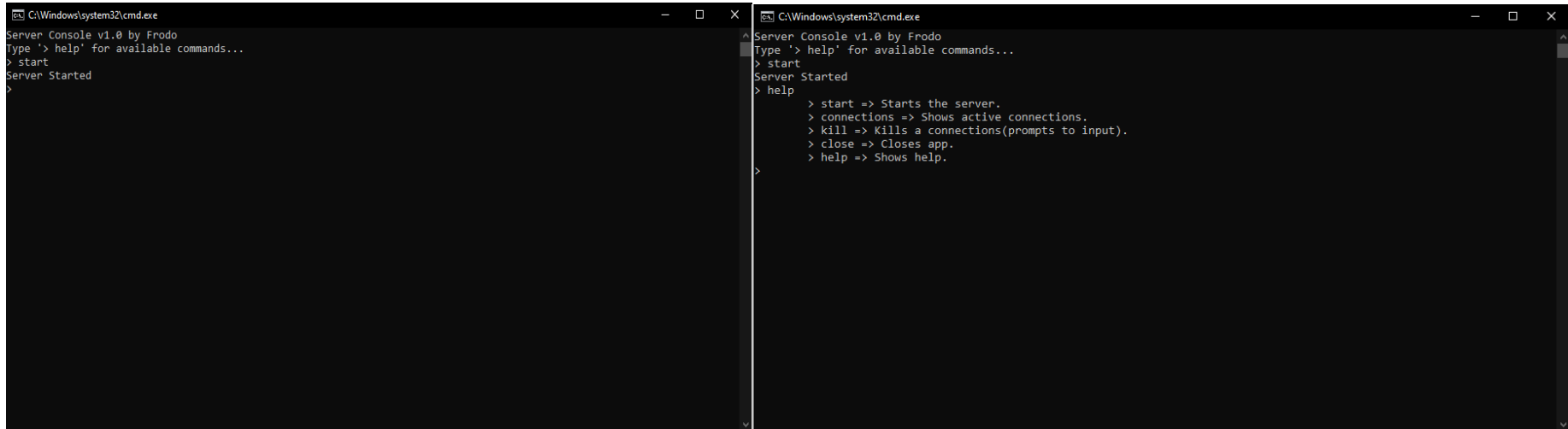
3.2. Proiectul este insotit si de doua diagrame UML aferente celor doua parti

Dat fiind faptul ca diagramele sunt destul de incarcate, am preferat sa le incarc separat in folderul de care apartine documentatia.

3.3. Am realizat o interfata grafica cu utilizatorul dupa cum se poate observa si mai sus



Ne logam cu unul dintre conturile deja inregistrate in baza de date, iar server IP este adresa IP a dispozitivului de pe care ruleaza serverul SQL.



```
C:\Windows\system32\cmd.exe
Server Console v1.0 by Frodo
Type '> help' for available commands...
> start
Server Started
>

C:\Windows\system32\cmd.exe
Server Console v1.0 by Frodo
Type '> help' for available commands...
> start
Server Started
> help
> start => Starts the server.
> connections => Shows active connections.
> kill => Kills a connections(prompts to input).
> close => Closes app.
> help => Shows help.
>
```

3.6. Am tratat exceptiile

```
public void Connect(String serverIP)
{
    try
    {
        Int32 port = 13000;
        client = new TcpClient(serverIP, port);
    }
    catch (Exception e)
    {
        Console.WriteLine("Exception: {0}", e);
        Console.Read();
    }
}
```

3.4. Am implementat fiecare modul intr-un dll separat

3.5. Fiecare fisier contine un antet cu informatii despre autor si functionalitate

```
Command.cs X
Server ServerSQL.Command.Command receivedPacket
1 //Autor: Sandu Diana-Elena
2 //Functionalitate: In aceasta clasa am realizat procesarea cererii clientului. Cand acesta doreste sa opereze cu baza mea de date va trimite
3 // o cerere serverului de forma numeFuncctie|argumente, unde argumentele sunt delimitate de caracterul "|", astfel afland ce
4 // functie trebuie sa apelam din DataController si, implicit, ce argumente va avea functia in cauza.
5 // Prin urmare se va procesa pachetul primit si se va construi pachetul pe care il vom trimite clientului
6
```

3.6. Proiectul cuprinde testarea unitatilor

```
[ClassInitialize] //run just once for all the tests in that class.
0 references | 0 changes | 0 authors, 0 changes
public static void Init(TestContext context)
{
    Console.WriteLine("Test_Step_1: Conectare drept client");
    client.Start("127.0.0.1");
    _Users = new Users(client);
}

[TestMethod]
0 references | 0 changes | 0 authors, 0 changes
public void Test_Nr_1_Login()
{
    //-----//
    //Verificare daca un user se poate conecta cu un username valid si parola valida
    //Type-Positive Test Case
    //-----//

    Console.WriteLine("Test_Step_2: Foloseste un user existent.");
    string userName = "Vali";
    string password = "Vali";

    Console.WriteLine("Test_Step_3: Verifica daca userul exista in baza de date.");

    bool loginStatus = _Users.Login(userName, password);
    Assert.AreEqual(true, loginStatus);
    client.CloseConnection();
}
```

Anexa:

Manipularea pachetelor primite de catre server:

```
public void HandleDevice(Object obj)
{
    TcpClient client = (TcpClient)obj;
    stream = client.GetStream();
    Packet packet = new Packet(); //pachetul primit de la client
    Packet responsePacket = new Packet(); //pachetul pe care il vom trimite in urma cererii clientului

    try
    {
        while (Running())
        {
            packet = SerializeControl.ReadObject(stream, _client.Client.RemoteEndPoint.ToString()); //citesc pachetul primit de la client folosind serializare TCP
            _log.WriteLog(Thread.CurrentThread.ManagedThreadId + ": Received: " + packet._data + "\n");

            Command.Command command = new Command.Command(dataController, packet);
            responsePacket = command.Execute(); // am pregatit raspunsul

            _log.WriteLog(Thread.CurrentThread.ManagedThreadId + ": Sent: " + responsePacket._data + "\n");
            SerializeControl.WriteObject(stream, responsePacket, _client.Client.RemoteEndPoint.ToString()); //trimit raspunsul clientului folosind serializare TCP
        }
    }
    catch (Exception e)
    {
        _log.WriteLog("Exception: " + e.ToString());
        client.Close();
    }
}
```

Exemplu de prelucrare a raspunsului serverului fata de client:

```
public Packet Execute()
{
    String dataPacket = "";
    String input = receivedPacket._data; //data este de forma nume_functie | argumentele functiei

    string[] val = input.Split('|').ToArray(); //numele functiei din DataController
    string[] args = null; //argumentele functiilor din DataController

    if (val.Length > 1) //verific daca functia necesita argumente
        args = val[1].Split(',').ToArray();

    Packet responsePacket = new Packet(); //pregatesc raspunsul pentru cererea clientului
    //Pot avea urmatoarele cereri:

    //1.
    if (CheckSubString(input, "GetTable") && args.Length == 3) //nume_functie = GetTable && arg.Length = 3
    {
        List<List<string>> data = dataController.GetTable(args[0], args[1], args[2]);

        //pregatesc _data raspunsului sub forma: extrag linie cu linie din tabela din BD. Intre elemente liniei extrase introduc "," iar la final de linie introduc ";"
        foreach (List<string> list in data)
        {
            string str = string.Join(",", list);
            dataPacket += str + ";";
        }

        //setez asta pentru retransmitere in cazul in care atunci cand cer tabela primesc un alt mesaj
        if (args[2].Equals("task")) responsePacket._type = "task";
        if (args[2].Equals("user")) responsePacket._type = "user";
    }
}
```

Exemplu de cerere si procesare a clientului:

```

private void GetTable(Client client)
{
    packet._data = "GetTable|TaskDB,Task,task";

    do
    {
        client.WriteObject(packet);
        response = client.ReadObject();
    } while (!response._type.Equals("task"));
    ParseResponse(response);
}

private void ParseResponse(Packet response)
{
    string[] values = response._data.Split(';').ToArray();
    foreach (string str in values)
    {
        string[] toks = str.Split(',').ToArray();

        if (!str.Equals(""))
        {
            Task task = new Task(Convert.ToInt32(toks[0]), Convert.ToInt32(toks[1]), toks[2], toks[3], toks[4], toks[5], Convert.ToInt32(toks[6]), Convert.ToInt32(toks[7]), toks[8]);
            _tasks.Add(task);
        }
    }
}

```

Verificarea periodica utilizand BackgroundWorker pentru a sesiza schimbari in cadrul sectiunii taskurilor userului, unde functia IsUpdateRequired verifica daca taskurile afisate pe interfata corespund cu cele preluate din baza de date periodic:

```

private void DoBackgroundWork(object sender, DoWorkEventArgs e) //workerul verifica periodic daca ar trebui sau nu sa fie actualizate taskurile pe interfata
{
    try
    {
        while (!_worker.CancellationPending)
        {
            if (IsUpdateRequired() == true)
            {
                MessageBox.Show("Refresh is required! Your task section has been updated!", "Notification", MessageBoxButtons.OK, MessageBoxIcon.Information);
            }
            Thread.Sleep(5000);
        }
    }
    catch (Exception ex) { MessageBox.Show(ex.Message); }
}

```