SOFTWARE FOR MONITORING INTRANET NETWORK

POPESCU MARIUS CONSTANTIN 1, DRIG FLORENTIN-GEORGE 2, NAAJI ANTOANELA 3
1,3"Vasile Goldis" Western University of Arad, Faculty of Economics, Computer Science and Engineering, Arad, Romania
2Atos It Solutions and Services, Timisoara, Romania
mpopescu@uvvg.ro, florentin.drig@outlook.com, anaaji@uvvg.ro3, etc

Abstract:
Nowadays, connectivity is an imperative factor within Intranet networks, especially in terms of speed, functionality, user-friendliness and time of intervention. The software application presented in this paper can be used to monitor network port(s), for one or several computers connected to the network, so that one could see, from a web administration page, which of the configured computers is in the network and which is not. This application can be an important tool for a small or medium business company because it simplifies the monitoring of the entire intranet network. The most important advantages are: high speed of updating the information, it does not slow the network by making useless traffic, it has a full reporting system and it is simple to use.

Keywords: network monitoring; user-friendly interface; connectivity; efficiency; flexibility

JEL classification: Y8.

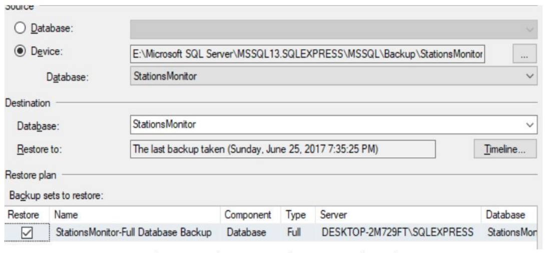
1. Introduction

In this era when information and access to it are very important, connectivity is an imperative factor. Connectivity between computers provides the possibility for information to flow at very high speeds, and monitoring the computers' network port is the first step. This application was thought out strictly for local networks, not for a global level where monitoring is easily made using specialized network equipment [5, 7]. Such an application is beneficial to any professional or company, if there are several computers in the managed network. Since in a company or plant running several computers to manage the production area it is imperative for these computers to be functional and connected to the network at all times, such an application is beneficial, as it can provide the shortest interruption of connectivity possible, and thus keep financial losses at a low level [2,11]. The application we developed can be very helpful for intranet network administrators, being equipped with a simple interface that is user - and administrator-friendly. As novelties brought by the application we can list user-friendliness and the way computers are monitored in a network. Since the whole of administration is easy to manage, the application can be used in a home network by individuals with only basic knowledge of IT. The application functions differently from similar commercial solutions found on the market. The system described is simple, with a set of features similar to the long-market systems or open source solutions (eg Nagios, Zabbix, Sensu) [1,4,8].

2. Software adjacent to the application

This paragraph describes the software that needs to be installed and set up before running the monitoring application. From address [13] one can download SQL Server 2016 Express Edition, and the SQL Server Management Studio R17 kit can be downloaded from address [14]. To configure the database to store data in the application, we will run SQL Server Management Studio. The credentials are the ones set up during installation.

Once connected, it can be seen that there no defined or newly created databases, outside of those used by SQL Server [6, 10]. For a faster configuration the database and the file associated to the work can be restored (on clicking databases and restore database), a window will appear where we will have to choose the file attached to the work, titled stationsMonitor.bak. The file will be selected (click on device and the button […], and the file stationsMonitor.bak will be picked from the location in which it was copied (click on add and Ok) (Fig.1).



Source

Database:

Device: E:\Microsoft SQL Server\MSSQL13.SQLEXPRESS\MSSQL\Backup\StationsMonitor

Database: StationsMonitor

Destination

Database: StationsMonitor

Restore to: The last backup taken (Sunday, June 25, 2017 7:35:25 PM)

Restore plan

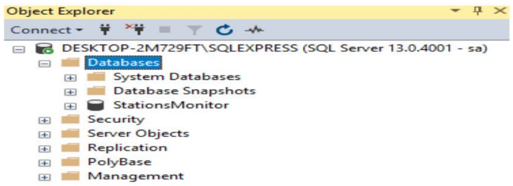
Backup sets to restore:

Restore	Name	Component	Type	Server	Database
✓	StationsMonitor-Full Database Backup	Database	Full	DESKTOP-2M729FT\SQLEXPRESS	StationsMon

Fig.1. Window to choose a database

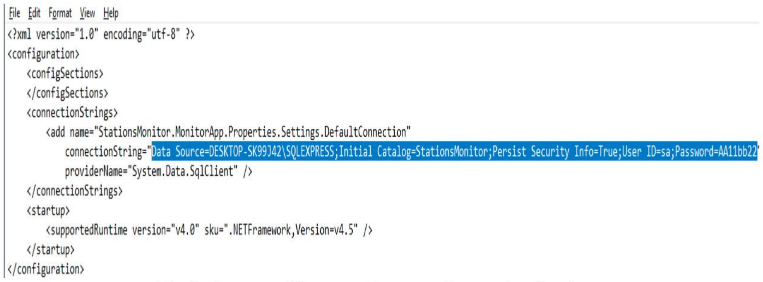
The steps are listed in detail, so as to make sure the application will run as expected and rule out any errors. To run the application it is necessary to fulfill certain requirements, such as a Windows 7/8.1/10 operating system, SQL Server 2014/2016, SQL Server Management Studio (included automatically for SQL Server 2014 versions or older, but not for the 2016 version), .NET Framework 4.6, .NET Core 1.1.

Once the database is restored, a confirmation message will be displayed on the monitor. When the database is restored, it can be used by the application (Fig.2). To use the database, the application must be configured (Fig.3). Thus, the folder Aplicatie monitorizare, associated with the paper, the file StationsMonitor.MonitorApp.exe is edited as follows: data source (the database address must be completed), initial Catalog (name of the previously restored database), user ID (database user) and password (the user's password).



DESKTOP-2M729FT\SQLEXPRESS (SQL Server 13.0.4001 - sa)

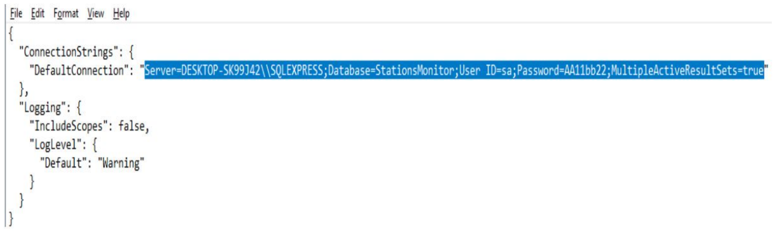
Fig.2. The solution of the server to which we are connected, after restoring the database



```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <configSections>
    </configSections>
    <connectionStrings>
        <add name="StationsMonitor.MonitorApp.Properties.Settings.DefaultConnection"
            connectionString="Data Source=DESKTOP-SK99J42\SQLEXPRESS;Initial Catalog=StationsMonitor;Persist Security Info=True;User ID=sa;Password=AA11bb22"
            providerName="System.Data.SqlClient" />
    </connectionStrings>
    <startup>
        <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />
    </startup>
</configuration>
```

Fig.3. Server address and connection to the database

In the folder Interfata web, associated with the work, the file appsettings.json (Fig.4) is edited, with the data filled out above.



```
{
  "ConnectionStrings": {
    "DefaultConnection": "Server=DESKTOP-SK99J42\\SQLEXPRESS;Database=StationsMonitor;User ID=sa;Password=AA11bb22;MultipleActiveResultSets=true"
  },
  "Logging": {
    "IncludeScopes": false,
    "LogLevel": {
      "Default": "Warning"
    }
  }
}
```

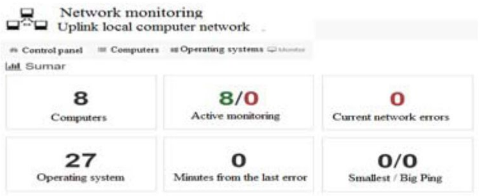
Fig.4. Server address and connection to the database

3. Application description

This paragraph presents each section in the functionality of the application. For better understanding of the functioning of each code sequence, sections from the source code are shown. Each algorithm is presented separately, from starting up the monitoring application and the description of functions, to the description of functions used to test connection with target computers. At the end of the paragraph we describe global parameters utilized, as well as the location where they could be downloaded from the Internet. The application is based on an algorithm created in C# which checks the list of computers added using pings, thus obtaining information on that computer, such as its availability in the network. Implementing technologies such as ASP.NET MVC, C# and Javascript are used. C# was used for the logical part of the application, to generate the algorithm, and ASP.NET Razor Views and Javascript were used for the interface part.

A. Available Menus

A summary of the information stored up until that moment can be viewed in the control panel (Fig.5). The function that returns the entire information is found in the dashboard controller.

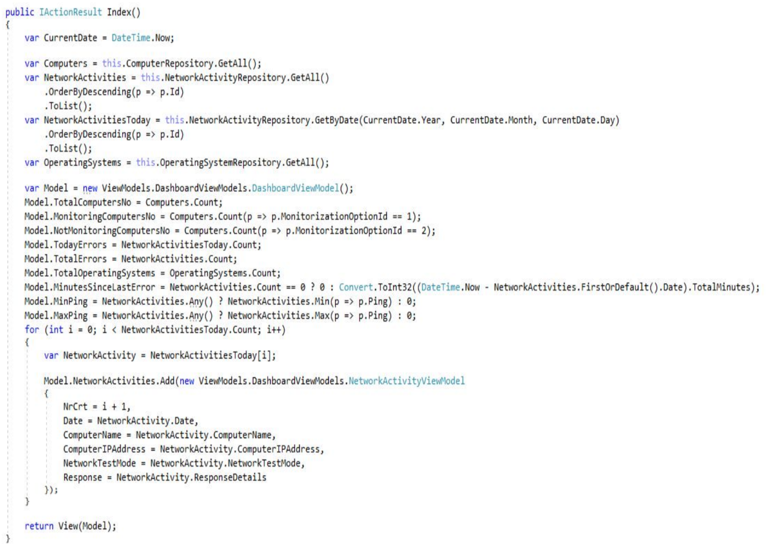


8 Computers	8/0 Active monitoring	0 Current network errors
27 Operating system	0 Minutes from the last error	0/0 Smallest / Big Ping

Fig.5. Summary page with information collected by the application

Steps covered by the algorithm are (Fig.6):

- storing the current date in the currentDate variable;
- extracting the list of computers, operating systems and stored activities from the database;
- the count() function was used to count computers and operating systems, which returns an integer of type int; as the count() function issues an error when the list of elements we wish to count is not initialized, the lists were checked using the any() function, which returns the logical value true or false;
- going through the list of stored activities, adding each element into a new list and displaying it using RazorView pages.

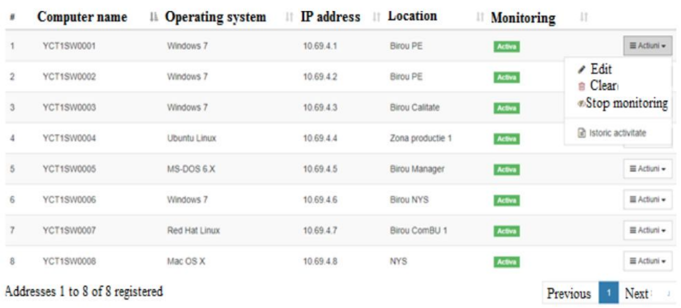


```
public IActionResult Index()
{
    var CurrentDate = DateTime.Now;

    var Computers = this.ComputerRepository.GetAll();
    var NetworkActivities = this.NetworkActivityRepository.GetAll()
        .OrderByDescending(p => p.Id)
        .ToList();
    var NetworkActivitiesToday = this.NetworkActivityRepository.GetByDate(CurrentDate.Year, CurrentDate.Month, CurrentDate.Day)
        .OrderByDescending(p => p.Id)
        .ToList();
    var OperatingSystems = this.OperatingSystemRepository.GetAll();

    var Model = new ViewModels.DashboardViewModels.DashboardViewModel();
    Model.TotalComputersNo = Computers.Count;
    Model.MonitoringComputersNo = Computers.Count(p => p.MonitorizationOptionId == 1);
    Model.NotMonitoringComputersNo = Computers.Count(p => p.MonitorizationOptionId == 2);
    Model.TodayErrors = NetworkActivitiesToday.Count;
    Model.TotalErrors = NetworkActivities.Count;
    Model.TotalOperatingSystems = OperatingSystems.Count;
    Model.MinutesSinceLastError = NetworkActivities.Count == 0 ? 0 : Convert.ToInt32((DateTime.Now - NetworkActivities.FirstOrDefault().Date).TotalMinutes);
    Model.MinPing = NetworkActivities.Any() ? NetworkActivities.Min(p => p.Ping) : 0;
    Model.MaxPing = NetworkActivities.Any() ? NetworkActivities.Max(p => p.Ping) : 0;
    for (int i = 0; i < NetworkActivitiesToday.Count; i++)
    {
        var NetworkActivity = NetworkActivitiesToday[i];

        Model.NetworkActivities.Add(new ViewModels.DashboardViewModels.NetworkActivityViewModel
        {
            NrCrt = i + 1,
            Date = NetworkActivity.Date,
            ComputerName = NetworkActivity.ComputerName,
            ComputerIPAddress = NetworkActivity.ComputerIPAddress,
            NetworkTestMode = NetworkActivity.NetworkTestMode,
            Response = NetworkActivity.ResponseDetails
        });
    }

    return View(Model);
}
```

Fig.6. Algorithm for the Summary page

B. Computers

The menu provides information on the complete list of added computers (Fig.7), which of them are monitored at the current moment, with the possibility of adding, removing or editing a computer.



#	Computer name	Operating system	IP address	Location	Monitoring	
1	YCT1SW0001	Windows 7	10.69.4.1	Birou PE	Activa	≡ Actiuni ▾
2	YCT1SW0002	Windows 7	10.69.4.2	Birou PE	Activa	
3	YCT1SW0003	Windows 7	10.69.4.3	Birou Calitate	Activa	
4	YCT1SW0004	Ubuntu Linux	10.69.4.4	Zona productie 1	Activa	
5	YCT1SW0005	MS-DOS 6.X	10.69.4.5	Birou Manager	Activa	≡ Actiuni ▾
6	YCT1SW0006	Windows 7	10.69.4.6	Birou NYS	Activa	≡ Actiuni ▾
7	YCT1SW0007	Red Hat Linux	10.69.4.7	Birou ComBU 1	Activa	≡ Actiuni ▾
8	YCT1SW0008	Mac OS X	10.69.4.8	NYS	Activa	≡ Actiuni ▾

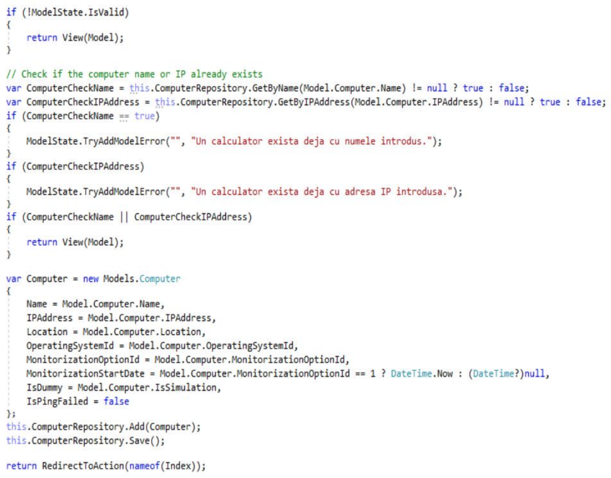
Addresses 1 to 8 of 8 registered

Fig.7. List of computers added for monitoring

Information listed on the page includes: computer name, operating system, IP address through which the activity is monitored, the computer's location, and the status, which concerns current monitoring. The menu of actions present next to each computer hides a set of activities for: editing, removing, starting/stopping monitoring, or viewing the history.

C. Adding a Computer

To add a new computer the blue button "+ New computer" is clicked, which redirects the user to the page where the computer is added, where after filling out the required data the Add button is clicked. If data were not filled out correctly, an error message will be returned to the user. The test sequence if (!ModelState.IsValid) tests if all fields have been filled out correctly (Fig.8). If one of the fields is omitted we will be automatically redirected to the same add computer page and a corresponding error message.



```
if (!ModelState.IsValid)
{
    return View(Model);
}

// Check if the computer name or IP already exists
var ComputerCheckName = this.ComputerRepository.GetByName(Model.Computer.Name) != null ? true : false;
var ComputerCheckIPAddress = this.ComputerRepository.GetByIPAddress(Model.Computer.IPAddress) != null ? true : false;
if (ComputerCheckName == true)
{
    ModelState.TryAddModelError("", "Un calculator exista deja cu numele introdus.");
}
if (ComputerCheckIPAddress)
{
    ModelState.TryAddModelError("", "Un calculator exista deja cu adresa IP introdusa.");
}
if (ComputerCheckName || ComputerCheckIPAddress)
{
    return View(Model);
}

var Computer = new Models.Computer
{
    Name = Model.Computer.Name,
    IPAddress = Model.Computer.IPAddress,
    Location = Model.Computer.Location,
    OperatingSystemId = Model.Computer.OperatingSystemId,
    MonitorizationOptionId = Model.Computer.MonitorizationOptionId,
    MonitorizationStartDate = Model.Computer.MonitorizationOptionId == 1 ? DateTime.Now : (DateTime?)null,
    IsDummy = Model.Computer.IsSimulation,
    IsPingFailed = false
};
this.ComputerRepository.Add(Computer);
this.ComputerRepository.Save();

return RedirectToAction(nameof(Index));
```

Fig.8. Algorithm corresponding to the interface of the form for adding a computer

After going through this step, we check if the name of the computer and its IP address are unique. If not, we will be redirected to the add computer page and a corresponding error message will be displayed. If all data were validated, a new object of the type computer shall be requested, which will be added to the database.

D. Editing a Computer

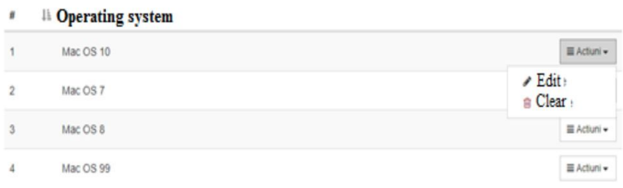
To edit information for a computer we click on the Actions button, found next to each one of them, then on Edit. The edit computer page is automatically filled out with information about it and then we will be able to make any necessary changes. After changing the data, the same validation process is followed as in the case of the adding part. If the name of the computer and its IP address are not unique, then a corresponding error message will be displayed.

E. Removing a Computer

Once redirected to the remove page, it will be filled out automatically with information on that computer. To remove a computer and all information associated to it, the Actions button is clicked, then the Remove button.

F. Operating Systems

The menu illustrated in Fig. 9 shows the current list of operating systems and can add, editor remove an operating system.



#	Operating system	
1	Mac OS 10	Actiuni
2	Mac OS 7	
3	Mac OS 8	Actiuni
4	Mac OS 99	Actiuni

Fig.9. List of added operating systems

To add a new operating system the blue button“+ New operating system” is clicked, which redirects us to the page where an operating system is added. The name of the operating system is filled out, and if omitted, a corresponding error message will be displayed. If no errors were returned after adding the new system, it will show up in the list of operating systems. The same steps are taken to edit or remove an operating system, and if in any of the required actions the data is not filled out properly, the system will alert the user through an error message.

G. Monitor

To follow and monitor the network efficiently, when encountering the first problem, the status of the computers will be changed, which is seen on the page in Fig.10.

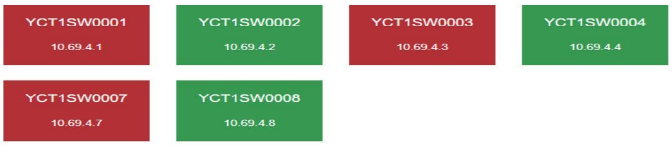


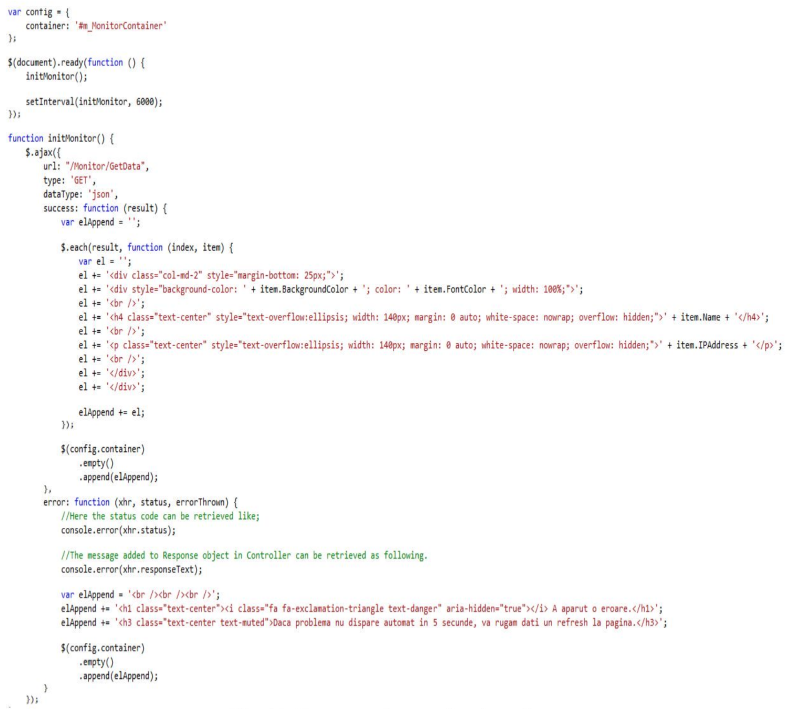
Fig.10. Monitor with the status resulting from the monitoring

The two statuses encountered are shown by two colors: red to highlight a problem and green if no problem was found on that computer. Javascript was used to display the computers dynamically on the monitor. The steps followed by the algorithm are (Fig.11):

- upon loading the full page a timer function will be called, setInerval, which in its turn call the initMonitor function, on a 6-second interval, which changes the status of computers on the page.

- the initMonitor function uses the Ajax principle (information may be requested about computers without having to reload the page), which means asynchronous javascript and xml.

- within the initMonitor function a repetition with a known number of steps is triggered, which goes through the list of computers.



```
var contig = {
    container: '#m_MonitorContainer'
};

$(document).ready(function () {
    initMonitor();

    setInterval(initMonitor, 6000);
});

function initMonitor() {
    $.ajax({
        url: "/Monitor/GetData",
        type: 'GET',
        dataType: 'json',
        success: function (result) {
            var elAppend = '';

            $.each(result, function (index, item) {
                var el = '';
                el += '<div class="col-md-2" style="margin-bottom: 25px;">';
                el += '<div style="background-color: ' + item.BackgroundColor + '; color: ' + item.FontColor + '; width: 100%;">';
                el += '<br />';
                el += '<h4 class="text-center" style="text-overflow:ellipsis; width: 140px; margin: 0 auto; white-space: nowrap; overflow: hidden;">' + item.Name + '</h4>';
                el += '<br />';
                el += '<p class="text-center" style="text-overflow:ellipsis; width: 140px; margin: 0 auto; white-space: nowrap; overflow: hidden;">' + item.IPAddress + '</p>';
                el += '<br />';
                el += '</div>';
                el += '</div>';

                elAppend += el;
            });

            $(config.container)
                .empty()
                .append(elAppend);
        },
        error: function (xhr, status, errorThrown) {
            //Here the status code can be retrieved like;
            console.error(xhr.status);

            //The message added to Response object in Controller can be retrieved as following.
            console.error(xhr.responseText);

            var elAppend = '<br /><br /><br />';
            elAppend += '<h1 class="text-center"><i class="fa fa-exclamation-triangle text-danger" aria-hidden="true"></i> A aparut o eroare.</h1>';
            elAppend += '<h3 class="text-center text-muted">Daca problema nu dispare automat in 5 secunde, va rugam dati un refresh la pagina.</h3>';

            $(config.container)
                .empty()
                .append(elAppend);
        }
    });
```

Fig.11. Program for monitor interface

H. Settings

The settings page allows the user to change certain properties of the application, such as: the number of pings sent (to check the status of the computer), the time – expressed in seconds–when the pings will be sent, the number of pings which will be sent to validate the connection (if the first ping does not return anything), the colors corresponding to the two statuses, and whether or not a virtual computer can be added for simulation.

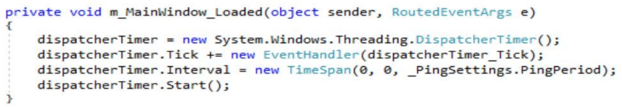
I. Javascript Libraries Used

The application used libraries to add the possibility to sort, search or choose the number of elements displayed in the tables, to validate the correct introduction of the IP address, as well as to make it possible to choose colors for the monitor [3,9,12].

4. Algorithm to monitor the network

A. Starting the Monitoring Application

The application is created using .NET WPF for better compatibility with Microsoft Windows operating system. For this system the WPF is the high end of application development. When starting the application, the algorithm is initialized and goes through the steps successively: an object of the type System.Windows.Threading.DispatcherTime is created, which is configured with a time interval expressed in seconds. The time interval is set in the "Application settings" module, found in the web interface (Fig.12);



```
private void m_MainWindow_Loaded(object sender, RoutedEventArgs e)
{
    dispatcherTimer = new System.Windows.Threading.DispatcherTimer();
    dispatcherTimer.Tick += new EventHandler(dispatcherTimer_Tick);
    dispatcherTimer.Interval = new TimeSpan(0, 0, _PingSettings.PingPeriod);
    dispatcherTimer.Start();
}
```

Fig.12. Full initialization function for the main window

- the object created is assigned a special function of the type eventHandler(nume_functie) by which –on exhausting the xset seconds –the function is called automatically;

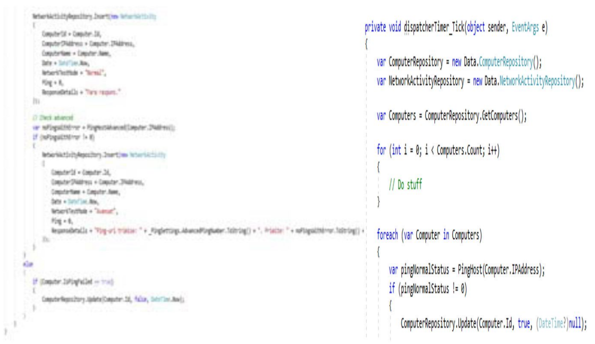
- the timer and monitoring are started on calling the function start();

- the variable pingSettings.pingPeriod contains an int value taken from the database using a sql query.

B. The EventHandler (dispattcherTimer_Tick) Function

For this module the dispatcherTimer_Tick function (Fig.13) has the purpose of taking the list of computers and checking their availability in the network.

Implementing the algorithm entails successively going through the following steps:



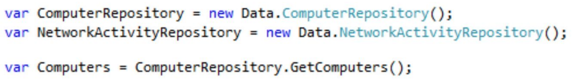
```
private void dispatcherTimer_Tick(object sender, EventArgs e)
{
    var ComputerRepository = new Data.ComputerRepository();
    var NetworkActivityRepository = new Data.NetworkActivityRepository();

    var Computers = ComputerRepository.GetComputers();

    for (int i = 0; i < Computers.Count; i++)
    {
        // Do stuff
    }

    foreach (var Computer in Computers)
    {
        var pingNormalStatus = PingHost(Computer.IPAddress);
        if (pingNormalStatus != 0)
        {
            ComputerRepository.Update(Computer.Id, true, (DateTime?)null);
```

```
            NetworkActivityRepository.Insert(new NetworkActivity
            {
                ComputerId = Computer.Id,
                ComputerIPAddress = Computer.IPAddress,
                ComputerName = Computer.Name,
                Date = DateTime.Now,
                NetworkTestMode = "Normal",
                Ping = 0,
                ResponseDetails = "Fara raspuns."
            });

            // Check advanced
            var noPingsWithError = PingHostAdvanced(Computer.IPAddress);
            if (noPingsWithError != 0)
            {
                NetworkActivityRepository.Insert(new NetworkActivity
                {
                    ComputerId = Computer.Id,
                    ComputerIPAddress = Computer.IPAddress,
                    ComputerName = Computer.Name,
                    Date = DateTime.Now,
                    NetworkTestMode = "Avansat",
                    Ping = 0,
                    ResponseDetails = "Ping-uri trimise: " + _PingSettings.AdvancedPingNumber.ToString() + ". Pierdute: " + noPingsWithError.ToString() +
                });
            }
        }
        else
        {
            if (Computer.IsPingFailed == true)
            {
                ComputerRepository.Update(Computer.Id, false, DateTime.Now);
            }
        }
    }
}
```

Fig.13. Function called automatically on the expiry of seconds in the timer

- initializing the objects that will return or save information about network activity (Fig.14), using the functions new data. ComputerRepository() and new data.NetworkActivityRepository().

- taking the entire list of computers from the database and configuring it for active monitoring, using the function



```
var ComputerRepository = new Data.ComputerRepository();
var NetworkActivityRepository = new Data.NetworkActivityRepository();

var Computers = ComputerRepository.GetComputers();
```

Fig.14. Initialization of objects for working with the database

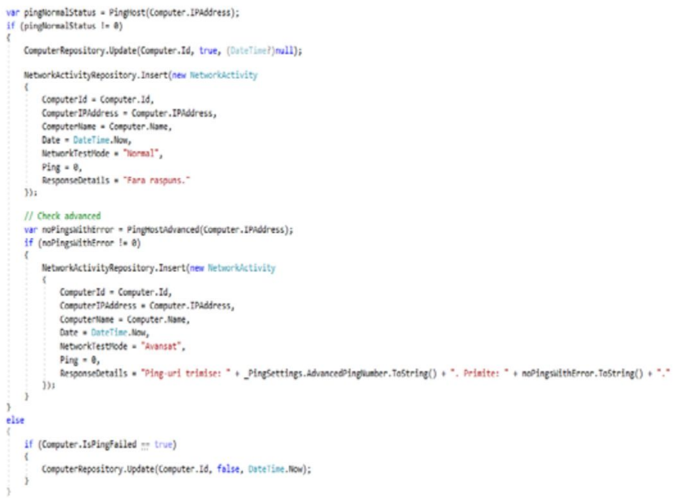
ComputerRepository.GetComputers() .Foreach was chosen to go through the list of computers, which entails a repetition algorithm with a known number of steps. Since the list of computers is initialized and their number is known, for better performance foreach is chosen over the for algorithm, which entails going through and counting the elements in the list.

- we go through the list of computers, computer by computer, and test their availability in the network (Fig.15).

- the normal test entails sending only one ping and checking the number of errors returned. The ping is sent by calling the pingHost(string IPAddress) function, which expects the IP address of a computer as parameter.

- if the number of errors returned equals zero, we check if that computer already has IsPingFailed.

- if the property is active, it should be deactivated, as the computer has answered promptly after the test, and no problems are found. If the number of errors returned by the function is different from zero, the IsPingFailed property is activated on that computer, which will allow us to see the changed status on the monitor, in the web interface. At the same time, the activity of the algorithm up to this moment is saved. This activity is saved into the database by calling the function networkRepository.Insert (new networkActivity{}) which expects a set of parameters, namely: computerId (it is the unique identifier of each computer, which is assigned to the computer automatically upon being added into the database), computerIPAddress (it is the IP address of the computer, set by the administrator), computerName (it is a name given to computer for better identification), Date (it is the current date when the first test is conducted), networkTestMode (this attribute can be "Normal" or "Advanced" depending on the test conducted–in this case the property is set to Normal), responseDetails (this property allows a more detailed saving of the response received following the test).



```
var pingNormalStatus = PingHost(Computer.IPAddress);
if (pingNormalStatus != 0)
{
    ComputerRepository.Update(Computer.Id, true, (DateTime?)null);

    NetworkActivityRepository.Insert(new NetworkActivity
    {
        ComputerId = Computer.Id,
        ComputerIPAddress = Computer.IPAddress,
        ComputerName = Computer.Name,
        Date = DateTime.Now,
        NetworkTestMode = "Normal",
        Ping = 0,
        ResponseDetails = "Fara raspuns."
    });

    // Check advanced
    var noPingsWithError = PingHostAdvanced(Computer.IPAddress);
    if (noPingsWithError != 0)
    {
        NetworkActivityRepository.Insert(new NetworkActivity
        {
            ComputerId = Computer.Id,
            ComputerIPAddress = Computer.IPAddress,
            ComputerName = Computer.Name,
            Date = DateTime.Now,
            NetworkTestMode = "Avansat",
            Ping = 0,
            ResponseDetails = "Ping-uri trimise: " + _PingSettings.AdvancedPingNumber.ToString() + ". Primite: " + noPingsWithError.ToString() + "."
        });
    }
}
else
{
    if (Computer.IsPingFailed == true)
    {
        ComputerRepository.Update(Computer.Id, false, DateTime.Now);
    }
}
```

Fig.15. Algorithm to test the connection of the computer to the network

- the next step is to try a more advanced test by sending several pings to the IP address of that computer. This test is initialized by calling the function pingHostAdvanced (string IPAddress) which in turn expects a string parameter, representing the IP address of the tested computer.

- following the advanced test, the number of errors returned is checked.

- if it is different from zero, the current information on the test is stored in the database by calling the networkActivity.Insert(new NetworkActivity{}) function, which expects the same type of parameters, but with a different value for networkTestMode. Being in the advanced test, this property will take the “Advanced” value.

- after completing the check for the first computer, we will move to the next one on the list, until we have zero computers remaining in the list.

C. Testing Connection using the PingHost(string IPAddress) Function

To test the connection the pingHost function is called, which sends a single ping to the IP address given as parameter. The steps of the algorithm will be (Fig.16):

- the variables countError and countOK are initialized to count the statuses returned following the test, these being ok, or an error;

- an object of type ping is initialized, for the purpose of sending the ping;

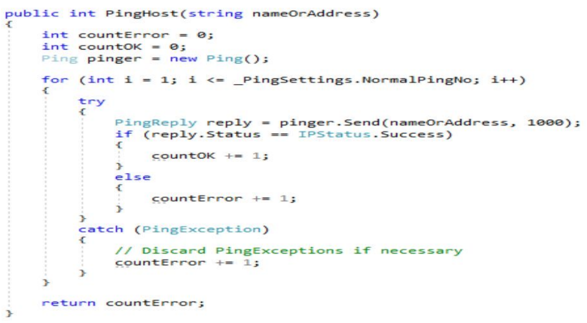
- in the web interface we can configure how many times this ping is send and checked; this number is given by the variable pingSettings.NormalPingNo;

- knowing the number of necessary iterations, an algorithm with a known number of steps was chosen;

- using the for syntax, the is ping is sent and we check if it returned a success status, or not; depending on this status, one of the variables above is incremented;

- to prevent any errors that might occur, this check was embedded into a try...catch structure;

- in the end the number of errors found is returned.



```
public int PingHost(string nameOrAddress)
{
    int countError = 0;
    int countOK = 0;
    Ping pinger = new Ping();

    for (int i = 1; i <= _PingSettings.NormalPingNo; i++)
    {
        try
        {
            PingReply reply = pinger.Send(nameOrAddress, 1000);
            if (reply.Status == IPStatus.Success)
            {
                countOK += 1;
            }
            else
            {
                countError += 1;
            }
        }
        catch (PingException)
        {
            // Discard PingExceptions if necessary
            countError += 1;
        }
    }

    return countError;
}
```

Fig.16. The pingHost function for testing the connection in normal mode

D. Testing the Connection using the PingHostAdvanced (string IPAddress) Function

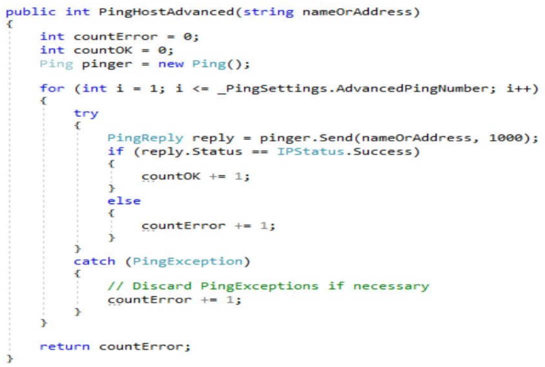
From a structural and algorithmic point of view, the function resembles the previous one, the difference being made by the number of iterations (Fig.17), which is configured by the administrator in the web interface.

The steps of the developed algorithm are:

- the countError and countOK variables are initialized, which memorize the number of pings sent successfully;

- the entire process of sending pings is embedded in a try...catch block to prevent errors that might occur;

- in the end, the number of pings that did not return ok status following the test is returned.

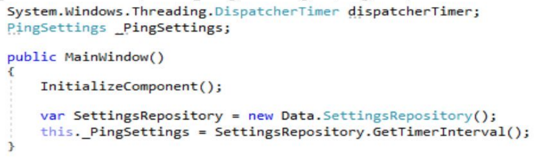


```
public int PingHostAdvanced(string nameOrAddress)
{
    int countError = 0;
    int countOK = 0;
    Ping pinger = new Ping();

    for (int i = 1; i <= _PingSettings.AdvancedPingNumber; i++)
    {
        try
        {
            PingReply reply = pinger.Send(nameOrAddress, 1000);
            if (reply.Status == IPStatus.Success)
            {
                countOK += 1;
            }
            else
            {
                countError += 1;
            }
        }
        catch (PingException)
        {
            // Discard PingExceptions if necessary
            countError += 1;
        }
    }

    return countError;
}
```

Fig.17. The PingHost function for testing the connection in advanced mode

E. Global Parameters

Two global parameters were used in developing the application, system.Threading.DispatcherTimer, and pingSettings (Fig.18).



```
System.Windows.Threading.DispatcherTimer dispatcherTimer;
PingSettings _PingSettings;

public MainWindow()
{
    InitializeComponent();

    var SettingsRepository = new Data.SettingsRepository();
    this._PingSettings = SettingsRepository.GetTimerInterval();
}
```

Fig.18. Defining global parameters and initializing the application interface

The mainWindow() function is called automatically by the application the moment when the application interface is initialized, the interface being initialized on calling the function initializeComponent().

F. The Arhitecture of ConnectingComputers to the Network

The application was tested in a closed network created with a wireless router (Fig.19). Equipment in the infrastructure on which the tests were conducted includes: laptop 1 (on which the application created and all required software were installed and which was connected by wireless network), laptop 2 (client computer which is monitored by the application and connected by wireless network), personal computer 1 (it is a client monitored by the application, and connection is wired), personal computer 2 (it is a client monitored by the application and is connected to the network by wire), wireless router. This application was tested in this closed environment, and the following step would be to test it in a production environment, with a complex topology like ring or star, Layer 2 and 3, network equipment, several VLANs communicating among themselves etc.

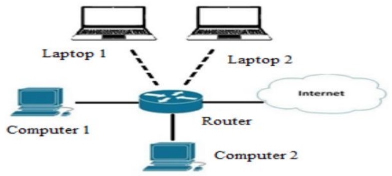


Fig.19.Connecting client computers to the architecture

5. Conclusions

The main advantages of the application lie in the simplicity of use (a minimum of knowledge required to be used), user-friendly interface, high efficiency, fast response times, updating the information in less than 5 seconds. The innovation and performances listed previously raise the value of the application over similar solutions proposed by the IT market. The application requires a few improvements that can be brought to increase the efficiency in a production and utilization environment, such as: automatically detecting all computers that are in the internal network, not using SQL Server 2016 (replacing it with an open source version and reconfiguring the application to use the new database - cost saving), and automating the entire process of configuring the application. This network monitoring application makes it possible to add and remove computers, and as such the monitoring function can be deactivated without removing the computer from the list. The software provides a simple, clean and user-friendly image of all work stations being monitored. At the same time, one can view a simple statistic of the number of computers in the list, how many of them are monitored, or the number of total errors intervening on a particular day. An operating system can be set for each computer, to have a complete image, as well as their activity, as a list, for the latest day. Such an application can be really useful to the IT staff of any company.

References

1. Bicaku, A., Balaban, S., Tauber, M.G., Hudic, A., Mauthe, A., Hutchison, D. (2016). Harmonized Monitoring for High Assurance Clouds. IC2EW, IEEE. Pp.118-123.

2. Drig, Florentin, Popescu, Marius. (2017). Network monitoring. Dissertation paper (in Romanian), Arad, unpublishes.

3. Gajda, V. (2010). JQuery. Poradnik programisty. Heloin, Kosciuszki.

4. Gutierrez-Aguado, J., Alcaraz Calero, J.M., Villanueva, W.D. (2016). IaaSMon: Monitoring Architecture for Public Cloud Computing Data Centers. J. Grid Computing 14. Pp.283–297.

5. Hong, J.W.K., Kwon, S.S., Kim, J.Y. (1999). WebTrafMon: Web-based Internet/Intranet network traffic monitoring and analysis system. Computer Communications 22. Pp.1333–1342.

6. Hotek, M. (2009). Microsoft SQL Server 2008. Computer Press.

7. Kim, H., Feamster, N. (2013). Improving Network Management with Software Defined Networking. Software Defined Networks, IEEE Communications Magazine. Pp.114-119.

8. Neto, A.F., Uchoa, J.Q. (2006). Ferramentas Livres para Monitoracao de Servidores. people.softwarelivre.org. Pp.149-154.

9. Rotmianto, M., Wahyudi, E. (2016). Developing Plugin e-DDC as an Additional Application for Senayan Library Management System with PHP Language Programming and MySQL Database. Record and Library Journal, Vol 2, No 1.

10. Russo, M., Ferrari, A., Webb, C. (2012). Microsoft SQL Server 2012 Analysis Services: The BISM Tabular Model.

11. Spatariu, Edmund Julien, Popescu, Marius Constantin, Naaji, Antoanela. (2015). Intelligent System with Remote Transmission for Monitoring Telecommunication Equipment Parameters. Proceedings of the IEEE 21st International Symposium for Design and Technology in Electronic Packiging, Braşov, Romania. Pp.329-334.

12. Vorobev, A.V., Shakirova, G.R. (2015). Web-Based Information System for Modeling and Analysis of Parameters of Geomagnetic Field. Procedia Computer Science, Vol. 59. Pp.73-82.

13. *** Microsoft SQL Server 2017, https://www.microsoft.com/en-us/sql-server/sql-server-downloads, accessed June 2018.

14. *** Microsoft SQL Management Studio, https://docs.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms, accessed June 2019.