Verkon laitteiden valvonta teollisuusympäristössä

Antti Taskila
Kevät 2012
Tietojenkäsittely
Oulun seudun ammattikorkeakoulu

TIIVISTELMÄ

Oulun seudun ammattikorkeakoulu
Tradenomi, Tietojenkäsittely

Tekijä: Antti Taskila
Opinnäytetyön nimi: Verkon laitteiden valvonta teollisuusympäristössä
Työn ohjaaja: Leo Ilkko
Työn valmistumislukukausi ja -vuosi: Kevät 2012 Sivumäärä: 35

Tämä on Ruukki Oy:n Raahen tehtaalle toteutettu opinnäytetyö, jonka tavoite oli luoda ohjelma ja valvoa siihen lisättyjen laitteiden saavutettavuutta verkossa. Aiemmin tiedot toimimattomista laitteista kulkivat useamman välikäden kautta esimerkiksi työntekijältä työnjohtajalle, joka taas ottaa yhteyttä parhaakseen katsomaansa henkilöön. Tässä ketjussa saattaa kestää pitkiäkin aikoja. Tästä johtuva tauko saattaa laskea tuotannon laitteiden toimintatehoa suurestikin.

Aluksi mietittiin eri tekniikoita ja niiden sopimista työn toteuttamiseen. Ohjelmapuolen ohjelmointikieliksi mietittiin C++, C# ja JAVA. Verkkopuolella mietittiin, sopiiko tehtävään parhaiten PHP, ASP vai ASP.NET. Lopulta päädyttiin tulokseen C# ja ASP.NET. Kehitysympäristönä toimi Microsoft Visual Studio 2010 ohjelman kehityksessä, Microsoft SQL Server 2008 Management Studio tietokantojen suunnittelussa ja Microsoft WebMatrix verkkosivujen kehityksessä. Ohjelma toimii Ruukin sisäverkossa ja käyttää hyväkseen sisäverkon pingattavuutta.

Ohjelma tulee nostamaan siihen lisättyjen laitteiden käyttöastetta ja tehoa laskemalla niiden korjauksen aloittamiseen tarvittavaa aikaa, koska vika saatetaan huomata, ennen kuin käyttäjä sitä havaitsee ja tiedot tapahtuneesta menevät suoraan oikealle henkilölle.

Lopputulokseksi kehittyi ohjelma, jolla voidaan helposti valvoa tarvittujen laitteiden saavutettavuutta sisäverkossa.

Asiasanat: C#-ohjelmointikieli, SQL, Visual Studio 2010

ABSTRACT

Oulu University of Applied Sciences

Bachelor of Business Administration, Information Technology

Author: Antti Taskila

Title of thesis: Monitoring Devices Connected to a LAN in an Industrial Environment
Supervisor: Leo Ilkko

Term and year when the thesis was submitted: Spring 2012 Number of pages: 35

The goal of this bachelor's thesis was to create a programme for Ruukki's factory in Raahe, capable of monitoring the devices e.g. computers added to the programme. Earlier the information about broken or otherwise disconnected devices or computers went from the users to their supervisor, who then contacted the person who may or may not be the right person to fix the problem. Finding the right person might take very long periods of time and the point of this programme is to reduce this downtime dramatically.

C# was selected for the programming language and the environment for developing the programme is Microsoft Visual Studio 2010. The database used is Microsoft SQL Server 2008 and the programme employed in managing and updating is Microsoft SQL Server Management Studio 2008. The techniques used to create the web user interface of the program are HTML, ASP.NET MVC3, JQuery and CSS.

The program is used in Ruukki's internal network and uses the good "pingability" of internal network to its advantage.

The point of the program is to raise utilisation and efficiency of the various devices added to the program by getting the right information to the right people.

The final product is a program capable of easily monitoring the accessibility of devices connected to the internal network.

Keywords: C# Programming language, SQL, Microsoft Visual Studio, WebMatrix

SISÄLLYS

1 JOHDANTO

Suoritin opintoihin liittyvän harjoittelua kehitysinsinööriharjoittelijan tittelillä Ruukki Oy:llä vuonna 2011 ja sain sieltä syksyllä myös opinnäytetyöksi sopivan aiheen. Ruukilla on sulaton alueellakin satoja verkkoon kytkettyjä laitteita, kuten prosessitietokoneita ja tuotantolaitteita. Osalla laitteista on erillinen valvontaohjelmisto ja osalla ei, kuitenkin olisi aina hyvä tietää jos joku kone menee epäkuntoon, ennen kuin seuraava käyttäjä sille menee ihmettelemään tilannetta ja soittamaan apua, mikä pahimmassa tapauksessa yöllä saattaa kestää tuntejakin. Käyttäjä ei välttämättä saa sinä aikana syötettyä järjestelmään mitään tietoja ja pitkätkin tuotantokatkokset ovat mahdollisia. Muistan kesältä yhden tapauksen, jossa tuotantolaite oli poissa käytöstä 16 tuntia vain sen takia, ettei sähkömiehille ja insinööreille ollut tullut asiasta tietoa.

Työn tarkoituksena oli kehittää ja toteuttaa ongelman ratkaisuun sopiva työkalu, joka olisi helppokäyttöinen eikä vaatisi erillistä asennusta. Ohjelman olisi myös hyvä selvittää ongelman lähdettä etukäteen.

2 MÄÄRITELMÄ

Opinnäytetyön tarkoituksena oli tehdä Ruukki Oy:lle verkkoon kytkettyjen laitteiden (esim. tietokone) valvontaan sopiva työkalu, joka olisi nykyaikaisen näköinen ja tarvittaessa muokattava. Tällä hetkellä valvontaan ei ole mitään järkeviä menetelmiä. Osalla laitteista on oma valvontaohjelmisto ja on vaivalloista katsoa viat joka laitteelle erikseen ja osaa laitteista ei voi valvoa etänä ollenkaan, vaan käyttäjä ilmoittaa viasta.

Koska kaikki valvottavat laitteet sijaitsevat lähiverkossa, on pingaus helppo ja suhteellisen varma tapa testata verkkoyhteyden toimivuus. Ping-komennolla voidaan testata TCP/IP-protokollassa laitteen saavutettavuutta. Komento lähettää annettuun IP-osoitteeseen paketin, johon etäkoneen tulisi vastata vastausviestillä. Jos vastausviestiä ei tule, voidaan tässä opinnäytetyössä vaadittavalla tarkkuudella todeta, että pingattavalla laitteella ei ole verkkoyhteyttä tai se ei jostakin muusta syystä ole tavoitettavissa. /5/

Ohjelman toinen tärkeä ominaisuus on vian alustava selvittäminen ja vastaaminen kysymykseen ”miksi laite ei vastaa?”. Tähän tarkoitukseen luodaan TCP/IP-protokollan mukainen traceroute, joka ilmoittaa ping-paketin liikkuman reitin. Pakettien pitää lähes aina liikkua useiden reitittimien läpi ja reitittimestä toiseen tapahtuvaa liikettä sanotaan hypyksi. Ping-paketille annetaan TTL (Time to Live)-arvo, joka kertoo, kuinka monta hyppyä se elää. Hyppyjen määrää eli TTL:n arvoa lisäämällä saadaan selville reitti, mitä paketti kulkee, tätä paketin kulkemaa reittiä tutkimalla saadaan selville mihin laitteeseen paketti vikatilanteessa pysähtyy. /5/

Tavoitteena oli luoda kolmeosainen järjestelmä joka koostuu pingaamisen hoitavasta ohjelmasta, tietokannasta ja selainkäyttöliittymästä. Käyttöliittymä ja pingaava ohjelma eivät keskustele suoraan keskenään, vaan kaikki tieto

välitetään tietokannan kautta. Tämä tarkoittaa että pinger-ohjelma voi pyöriä millä vain tietokoneella, kunhan sillä on yhteys tietokantaan.

2.1 Vaatimukset

Työkalun tulisi valvoa ja testata siihen syötettyjä laitteita ja tarvittaessa kertoa mahdollisista vioista ja auttaa niiden selvittämisessä. Ping ja Traceroute todettiin tarpeeksi luotettaviksi ja tarkoiksi.

Ohjelman tulisi ongelmatilanteessa lähettää sähköpostia määritettyihin osoitteisiin ja kertoa vian laadusta ja lähettää uuden ja vanhan traceroutauksen vertailu.

Ohjelman käyttöliittymän tulisi olla nykyaikaisen näköinen ja webpohjainen sekä verkkoselaimessa toimiva, niin että ohjelmaa voidaan käyttää miltä vain koneelta, jolta on yhteys lähiverkkoon. Näin työkalua voidaan käyttää ilman, että tietokoneille tarvitsee asentaa erillisiä ohjelmia.

Tarkoituksena oli myös testata verkkokehitystyökaluja ja mikä olisi sulaton tapauksessa jatkossa hyvä tapa tehdä verkko-ohjelmia.

2.2 Kehitysympäristö

Ohjelmointikieltä valittaessa harkittiin C#:n, Javan ja C++:n välillä, joista valittiin C#, koska se ja sen kehityksessä käytettävä Microsoft Visual Studio 2010 oli kesän aikana tullut tutuksi ja C# tuntui tuttuna kielenä loogiselta, nykyaikaisemmalta valinnalta.

C# on yksinkertainen, nykyaikainen, olioperusteinen ja tyyppiturvallinen ohjelmointikieli, joka on C- ja C++- kielien perillinen. C++:aa opiskelleelle C# tuntui heti todella loogiselle, tutulle ja helppokäyttöiselle ja innostuinkin sillä ohjelmoinnista heti kerralla. /1/

Microsoft Visual Studio 2010 on Visual Basic-, C++-, C#- ja J#- ohjelmointiin tarkoitettu ohjelmankehitysympäristö, jolla voidaan tehdä useanlaisia sovelluksia (kuten konsoli-, windows- ja web-ohjelmia). Visual Studio käyttää hyväkseen Microsoftin omaa .NET Framework -ohjelmistokomponenttikirjastoa, joka sisältää useita pinger-ohjelmassa käytettäviä työkaluja, kuten työkalut pingaukseen ja tietokantaoperaatioihin.

Verkkokäyttöliittymän pohjana toimii normaali XHTML-koodi. HTML (HyperText Markup Language) on kieli, jolla verkkosivut koodataan.

ASP.NET on Microsoftin kehittämä kehys (framework), josta löytyy työkalut dynaamisten verkkosivujen kehittämiseen. Uusin ASP.NET MVC3 toi mukanaan uuden Razor-syntaksin, joka mielestäni selkeytti ja helpotti ASP.NETin kirjoittamista. Vakava vaihtoehto ASP.NETin tilalle oli aluksi PHP, jolla ohjelman toteuttaminen olisi myös onnistunut, mutta uusi selkeä Razor-syntaksi innosti minua kokeilemaan ASP.NETiä.

Microsoft WebMatrix on ASP.NET MVC3:n kanssa samaan aikaan julkaistu kehitysympäristö, joka on kokemuksieni mukaan yksinkertainen, mutta tehokas työkalu verkkosivustojen kehittämiseen ja julkaisemiseen. WebMatrixilla voi helposti kehittää verkkosivustoja, joilla on palvelinpuolen toiminnallisuutta, kuten tietokantahaut tässä työssä. WebMatrix tukee css-tyylitiedostoja ja Javascriptiä.

JQuery on Javascript-kirjasto, jonka tarkoitus on helpottaa Javascriptin mielestäni normaalisti hankalahkoa syntaksia ja tuoda mukanaan paljon työkaluja mm. verkkosivujen animoinnin ja AJAX-toimintojen helpottamiseen. JQuery UI on Jqueryn päälle rakennettu Javascript-kirjasto, joka sisältää pääasiassa sivuston ulkoasun parantamiseen käytettäviä työkaluja. JQuerylle on muutamia vaihtoehtoja, kuten Ext ja MooTools, mutta koska olin ennenkin käyttänyt JQuerya päädyin valinnassani siihen.

Tietokannan pohjana toimii palvelimella pyörivä Microsoft SQL Server 2008 -tietokanta, jota hallitaan pääasiassa Microsoft SQL Server Management Studio

2008:lla, mutta joitakin muutoksia on tehty WebMatrixin tietokantatyökalulla. Valmiiksi asennettu tietokanta ja sen lukemiseen tarkoitettu ohjelmisto tekivät tietokantaan liittyvistä valinnoista helppoja, enkä oikeastaan harkinnut edes muuta.

2.3 Palvelin

Palvelin, jolla ohjelmisto pyörittää, on HP PROLIANTA DL380 G7, jonka käyttöjärjestelmänä toimii Windows Server 2008. Palvelimelle on asennettu Microsoft SQL Server 2008-tietokanta, jota käytetään työssä tietokantana. Koneelle on asennettu IIS 6.0- ja IIS 7.0 -palvelinohjelmistot, joita työssä käytetään verkkopohjaisen käyttöliittymän näyttämiseen ja sähköpostien lähettämiseen.

3 TIETONKANTA

Tietokanta on kokoelma tietoa järjestetyssä muodossa. Järjestetyllä tarkoitetaan sitä, että tiedoille on omat paikkansa ja niitä on helppo hakea ja järjestellä tarpeen mukaan. Tietokanta voi koostua yhdestä tai useammasta tietoa kokoavasta taulusta, joiden välillä on yhteyksiä. Näitä kutsutaan relaatiotietokannoiksi. Relaatiotietokannan taulut koostuvat sarakkeista, joilla on omat tietotyyppinsä. Tieto tallennetaan aina valmiisiin sarakkeisiin omille riveilleen. /2/

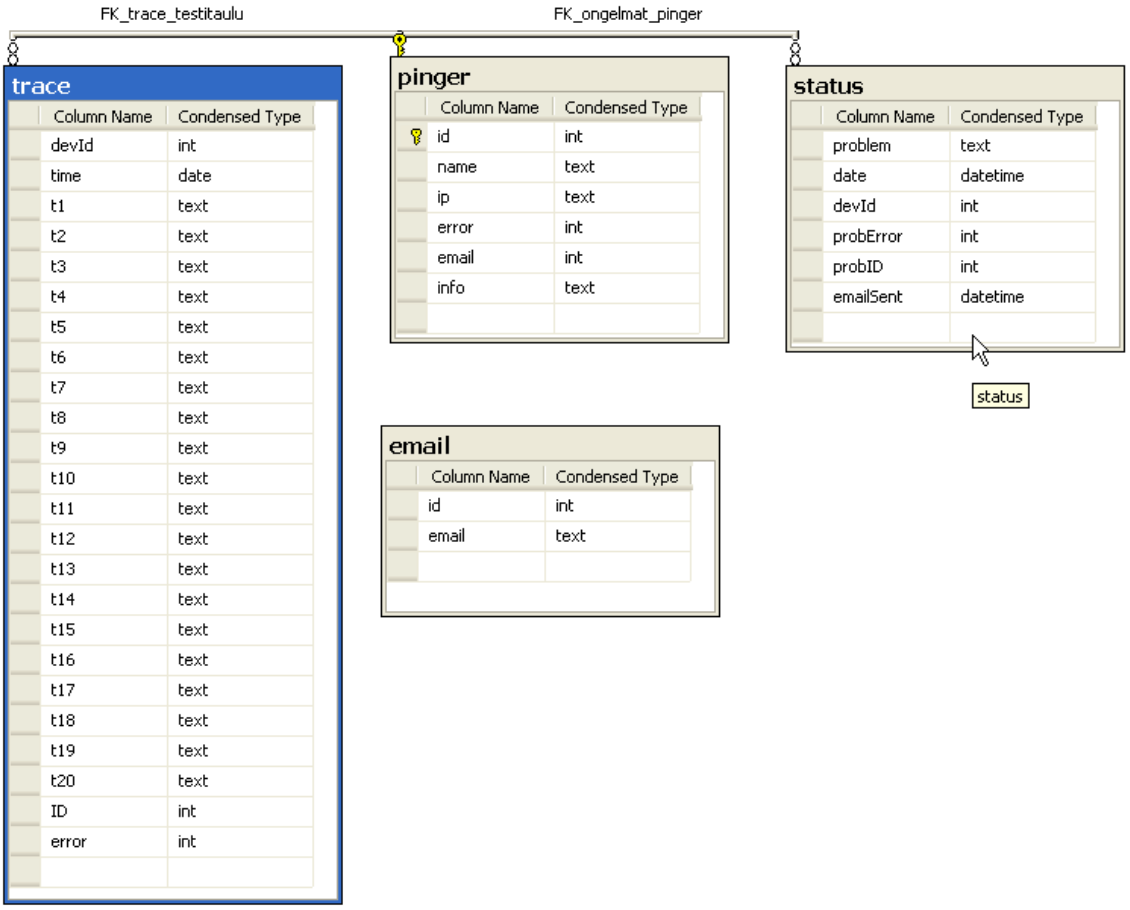
Työssä käytetään Microsoft SQL Server 2008 -tietokantaa joka sijaitsee palvelimella jossa ohjelma pyörii. Tietokantaa hallitaan pääosin Microsoft SQL Server Management Studion avulla, mutta osa muutoksista on tehty myös WebMatrixin tietokannanhallintatyökaluilla. Tietokannasta haetaan ja sinne kirjoitetaan tietoa sekä pinger-ohjelmalla että selainkäyttöliittymällä.

Palvelimelle loin pingerdb -nimisen kannan, joka sisältää 4 taulua: pinger, trace, status ja email. Näitä tauluja lukemaan luotiin käyttäjä jolla on oikeudet lukea ja kirjoittaa vain pingerdb -kantaan.

En kiinnittänyt suurta huomiota tietotyyppeihin, koska tietokantojen koot eivät luultavasti tule olemaan erityisen suuria ja käytetyt TEXT-, INT- ja DATETIME-tyypit käyvät tarpeeksi tässä vaiheessa. Asiasta kerrottu hieman lisää kohdassa 7.4.

Työssä tietokanta on kovalla käytöllä ja se on keskeinen osa ohjelman toimivuutta, koska kaikki keskeinen data tallennetaan siihen. Tietokanta toimii myös välikappaleena käyttöliittymän ja pinger-ohjelman välillä.

3.1 Taulut



KUVA 1. Tietokannan taulut, sarakkeet ja niiden tietotyypit.

Edellisen kuvan 1. taulut ja sarakkeet selitettynä

Pinger-taulu sisältää laitteen perustiedot.

- id, jokaiselle laitteelle automaattisesti annettava arvo, joka periytyy status ja trace -taulujen devId:hen.
- name, laitteelle annettava selkokielinen nimi.
- ip, laitteen IP.
- email, joko 1 tai 0, riippuen lähetetäänkö laitteen tietoja ongelmatilanteessa sähköpostina.
- info, lisää tietoa laitteesta ja sen sijainnista.

Status-taulu, sisältää pingauksen tulokset ja milloin asiasta ilmoitettiin viimeksi.

- devId, arvo joka ilmoittaa, mille laitteelle tiedot kuulut, periytyy pinger-taulun id-arvosta.
- problem, sisältää selkokielisen selityksen pingauksen tilasta esim. Success tai TimeOut.
- date, milloin pingaus tapahtunut.
- probError, luku 1 tai 0, riippuen siitä onnistuiko pingaus.
- emailSent, kertoo, onko ongelmasta lähetetty sähköpostia ja milloin.
- probID, rivin yksilöivä nouseva arvo.

Trace-taulu, sisältää tracerouten tulokset

- devId, arvo joka ilmoittaa, mille laitteelle tiedot kuuluvat, periytyy pinger-taulun id-arvosta-
- time, milloin traceroute suoritettu
- ID, rivin yksilöivä nouseva arvo.
- error, arvo 1 tai 0, kertoo, oliko tracerouten kohde tracerouten hetkelle pingattavissa.
- t1-t20, tracerouten hyppyjen ip-osoitteet

Email-taulu, sisältää listan sähköposteista mihin tieto toimimattomasta laitteesta lähetetään.

- id, rivin yksilöivä nouseva arvo.
- email, lista sähköpostiosoitteista, mihin tieto tarvittaessa lähetetään.

3.2 Triggerit

Triggerit suorittava määritellyn toiminnon, kun tietyssä taulussa tapahtuu jokin tietty tapahtuma, esimeriksi riviä muutetaan tai se poistetaan. Tämän työkalun tietokannan pinger-taulussa on createStatusOnInsert -niminen triggeri, joka suorittaa itsensä aina, kun pinger-tauluun lisätään rivi. Aina kun pinger-tauluun luodaan rivi, luodaan status-tauluun uusi rivi jonka devId:ksi annetaan pinger-

tauluun luodun uuden rivin id. Tämä kiertää ongelman, jonka kohdassa 4.1 esiteltävä, WebGrid aiheuttaa, kun se yrittää näyttää laitteen tietoja, ilman että siitä on mainintaa status-taulussa.

```
CREATE TRIGGER dbo.createStatusOnInsert
   ON  dbo.[pinger]
   AFTER INSERT
AS
BEGIN
    SET NOCOUNT ON;
    INSERT INTO dbo.status (devId) SELECT id FROM dbo.pinger

END
GO
```

KUVA 2. Triggerin luonnin koodi.

3.3 Tietokannan lukeminen

Tietokantaa luetaan ja kirjoitetaan sekä käyttöliittymästä ja että pinger-ohjelmasta. Käyttöliittymästä tietokantaoperaatiot tehdään ASP.NET MVC3 Razor-syntaksilla, jolla tehdyt tietokantaoperaatiot ovat melko yksinkertaisia ja selkeitä, kuten kuvasta 3 näkyy.

```
var db = Database.Open("uTietokanta");
var sql = "SELECT * FROM dbo.pinger WHERE id= @0";
var row = db.QuerySingle(sql, Id);
```

KUVA 3. Yksittäisen rivin hakeminen tietokannasta muuttujaan Razor syntaksilla.

Pinger-ohjelman tietokantahaut ja kirjoitukset tehdään .NET Frameworkista löytyvillä TableAdapter- ja SqlConnection-luokkien olioilla, jotka esitellään tarkemmin kohdassa 5.3.

4 KÄYTTÖLIITTYMÄ

Käyttöliittymän kehitys lähti liikkeelle hakemalla tietokannasta testirivejä ja yrittämällä vain saada ne näkyviin selaimessa. Tästä edettiin pikkuhiljaa ja etsittiin käytännöllistä tapaa esittää laitteita helposti ymmärrettävässä ja muuteltavissa olevassa muodossa. Selainpohjaisia pienehköjä tietokantasovelluksia olin tehnyt php:n avulla aiemmin, mutta php on mielestäni melko vaivalloinen ja epämiellyttävä ohjelmoida, joten päädyin tekemään sovellusta ASP.NETin avulla, josta oli juuri ilmestyny uusi MVC3. MVC tarkoittaa ohjelmistoarkkitehtuuria, missä ohjelmataso on erotettu käyttöliittymästä, joka tässä tapauksessa on erittäin sopiva, koska ohjelma pyörii palvelimella ilman käyttäjää ja sitä hallitaan epäsuorasti tietokannan kautta. Pääasiallinen syy ASP.NET MVC3:n valintaan oli uusi Razor -syntaksi, joka minusta tuntuu PHP:hen verrattuna paljon luonnollisemmalta käyttää.

Uuden version kanssa oli ilmestynyt ilmainen Microsoft WebMatrix -ympäristö, joka on melko aloittelijaystävällinen, mutta josta kumminkin löytyy laajuutta ammattilaiselle saakka. /3/

WebMatrixista löytyy sisäänrakennettuna selainohjelmointiympäristö, tietokannan hallinta ja julkaisujärjestelmä. Julkaisun voisi hoitaa joko siirtämällä tiedostoja käsin, ftp:n ylitse tai Web Deploylla, joka on Microsoftin suunnittelema työkalu verkkosivujen ja verkko-ohjelmistojen siirtämiseen IIS-pohjaisille palvelimille. Koska palvelimella pyöri pohjana IIS, julkaisutyökaluna toimi Web Deploy.

KUVA 4. Käyttöliittymän perusnäkymä

4.1 Webgrid

WebGrid on uuden ASP.NET MVC3:n mukana tullut auttaja, jonka tarkoituksena helpottaa datan näyttämistä taulukkomuodossa helposti muokattavassa muodossa, esimerkin webgridistä näkee kuvasta 4. Webgridiä voi päivittää asynkronisesti eli ilman, että tarvitsee päivittää koko sivua.

```
var sql2 = "SELECT * FROM dbo.status WHERE devId=@0 ORDER BY date DESC";
var data2 = db.Query(sql2, Id);
var grid2 = new WebGrid(data2, ajaxUpdateContainerId: "grid2", rowsPerPage: 5);
```

KUVA 5. Webgridin täyttäminen tietokannasta haetuilla tiedolla.

```
@grid2.GetHtml(
    columns:
    grid2.Columns(
        grid2.Column("problem", "Status"),
        grid2.Column("date", "Ajankohta")
    )
)
```

KUVA 6. WebGridissä voidaan määritellä mitkä kentät halutaan näyttää. Jos halutaan näyttää kaikki, ainoa pakollinen koodi @grid2.GetHtml().

Kuvan 5 koodi voidaan sijoittaa esimerkiksi sen sivun alkuun, missä WebGrid esiintyy. Kuvan 6 koodi sijoitetaan HTML-koodissa siihen kohtaan, johon WebGrid halutaan sijoittaa.

4.2 JQuery

JQuery on tällä hetkellä maailman suosituin avoimen lähdekoodin JavaScript kirjasto jonka tarkoituksena on helpottaa ja muuttaa tapaa jolla JavaScriptiä kirjoitetaan. Se soveltuu hyvin AJAX-pohjaisten verkkosivustojen tekemiseen ja niiden animointiin. AJAX (Asynchronous JavaScript and XML) on tekniikka, jonka tarkoitus on tehdä verkkosivustosta vuorovaikutteinen ja nopeampi, esimerkiksi verkkosivun sisällön muuttuessa ei tarvitse päivittää koko sivua, vaan AJAXin avulla voidaan päivittää vain muuttunut kohta. /4/

JQuery UI on JQueryn päälle rakennettu koodikirjasto, joka mahdollistaa helposti mm. animointien ja efektien teon verkkosivuille.

JQueryllä on työssä toteutettu WebGridin erilaiset toiminnallisuudet kuten napit, WebGridiä klikkaamalla ilmestyvät dialogit, varjostukset ja väritykset. Jquery myös täyttää dialogien sisällön dynaamisesti AJAX:in avulla.

```
//Muokkauksen dialogin aukasu
$('#grid').delegate('.edit-db', 'click', function (e) {
    e.stopPropagation();
    $.getJSON('~/../Pingaus/Methods/dbQuery/' + $(this).attr('id'), function (data) {
        var lista = data;
        $('#edit-id').val(lista.id);
        $('#edit-nimi').val(lista.name);
        $('#edit-info').val(lista.info);
        $('#edit-ip').val(lista.ip);
    });
    $('#action-type').val('edit');
    tyyppi = 'edit';
    $("#edit").show();
    $("#delete").hide();
    $("#emaillistdialog").hide();
    $('#dialog-edit').dialog('open');
});
```

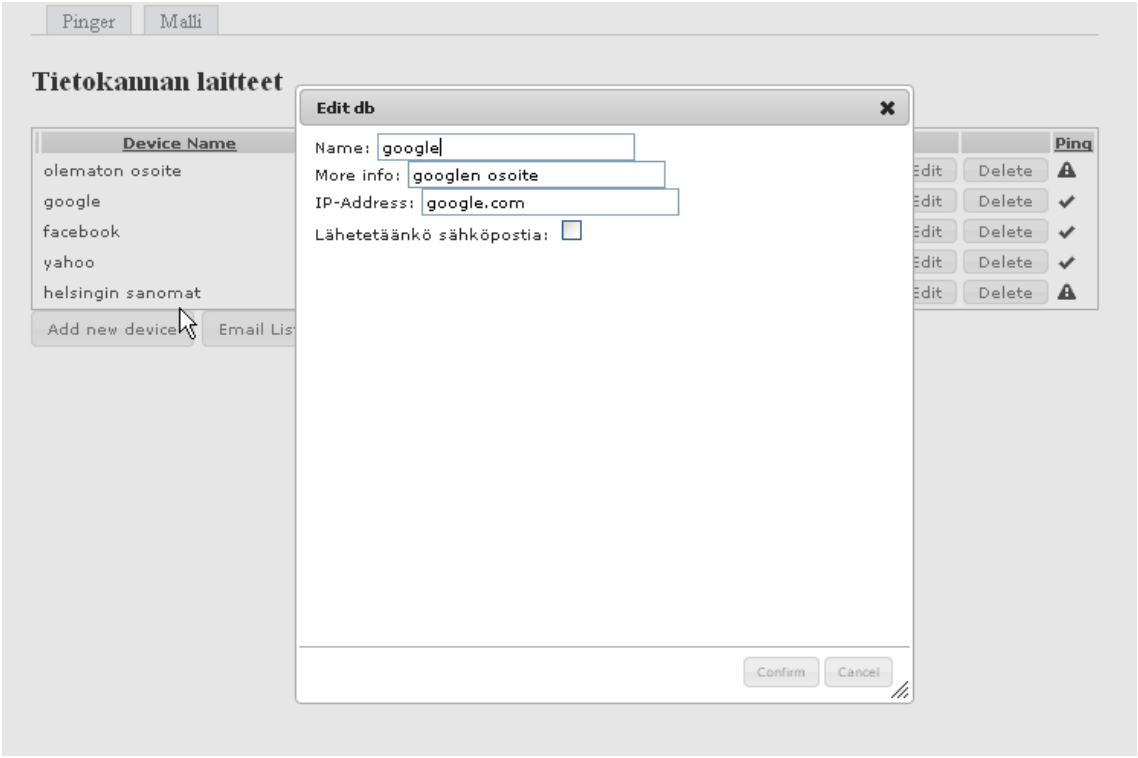
KUVA 7. Muokkausdialogin luominen ja aukaiseminen.

Edellisen kuvan mukainen koodi aukaisee dialog-edit dialogin silloin, kun grid id:llä varustetun div:in sisällä on klikattu edit-db -luokan elementtiä eli käytännössä, kun klikkaat webgridin edit-nappia. Dialogin sisältö haetaan dbQuery:n (katso kuva 8) suorittamasta tietokantahausta JSON -muodossa,

dbQuery.cshtml sisältää razor -syntaksilla tehdyn tietokantahaun joka ottaa vastaan laitteen id:n, tekee sillä haun tietokannasta ja kirjottaa sen JSON-muotoon ja palauttaa sen. Palautettu tieto kirjoitetaan sitten aukaistun muokkausdialogin kenttiin.

```
@{
    if(UrlData[0].IsInt()){
        var db = Database.Open("uTietokanta");
        var sql = "SELECT * FROM pinger WHERE id = @0";
        var lista = db.QuerySingle(sql,UrlData[0]);
        Json.Write(lista, Response.Output);
    }
}
```

KUVA 8. dbQuery ottaa vastaan laitteen id:n ja hakee sillä tiedot tietokannasta.



KUVA 9. Kuva muokkausikkunasta.

```
$('#dialog-edit').dialog({
    autoOpen : false,
    modal: true,
    height: '400',
    width: '400',
    buttons: {
        'Confirm' : function () {
            $.ajax({
                type: "POST",
                url: $("#edit-db-form").attr('action'),
                data: $("#edit-db-form").serialize(),
                dataType: "text",
                success: function () {
                    $('#dialog-edit').dialog('close');
                    $("#grid").load('~/../Pingaus/Grid/ #grid', function () {
                        if (tyyppi === 'edit') {
                            $('#' + id).parent('td').parent('tr').effect("highlight", {}, 2000);
                        } else if (tyyppi === 'add') {
                            $('tbody > tr:first').effect("highlight", {}, 2000);
                        }
                    });
                },
                error: function (response) {
                    alert(response.responseText + "wat");
                    $('#dialog-edit').dialog('close');
                }
            });
        },
        Cancel: function () {
            $('#dialog-edit').dialog('close');
        }
    }// end buttons
});
```

KUVA 10. Muokkausdialogin asetukset.

Muokkausdialogin asetukset, kuten koko ja nappien toiminnot on määritetty aiemmin ja dialogin aukaisu vain täyttää valmiin pohjan. Confirm -napin painaminen lähettää tässä tapauksessa POST -tietona dialogiin täytetyt tiedot sarjamuodossa eteenpäin. Jos POST:in lähetys onnistuu, päivitetään WebGrid uusilla tiedoilla ja tuodaan muutettu tieto esille hienolla efektillä. Cancelin painaminen vain sulkee dialogin. POST on http:n metodi jolla palvelinta pyydetään ottamaan vastaan lähetettävää tietoa.

POST:ina lähetetty tieto otetaan vastaan dbEdit.cshtml -tiedostossa, joka luo tietokantayhteyden ja kirjoittaa uudet tiedot vanhan tilalle tietokantaan.

```
2     if(IsPost){
3     var db = Database.Open("uTietokanta");
4     var actiontype = Request["action-type"];
5
6         if(actiontype=="edit") {
7         var sql = "UPDATE pinger SET name = @0, info = @1, ip = @2, email = @3 WHERE id = @4";
8         var name = Request["nimi"];
9         var info = Request["info"];
10        var ip = Request["ip"];
11        var id = Request["id"];
12        var email = 0;
13        if(Request["Email"] == "on"){
14        email = 1;
15        }
16        db.Execute(sql, name, info, ip,email, id);
17        }
```

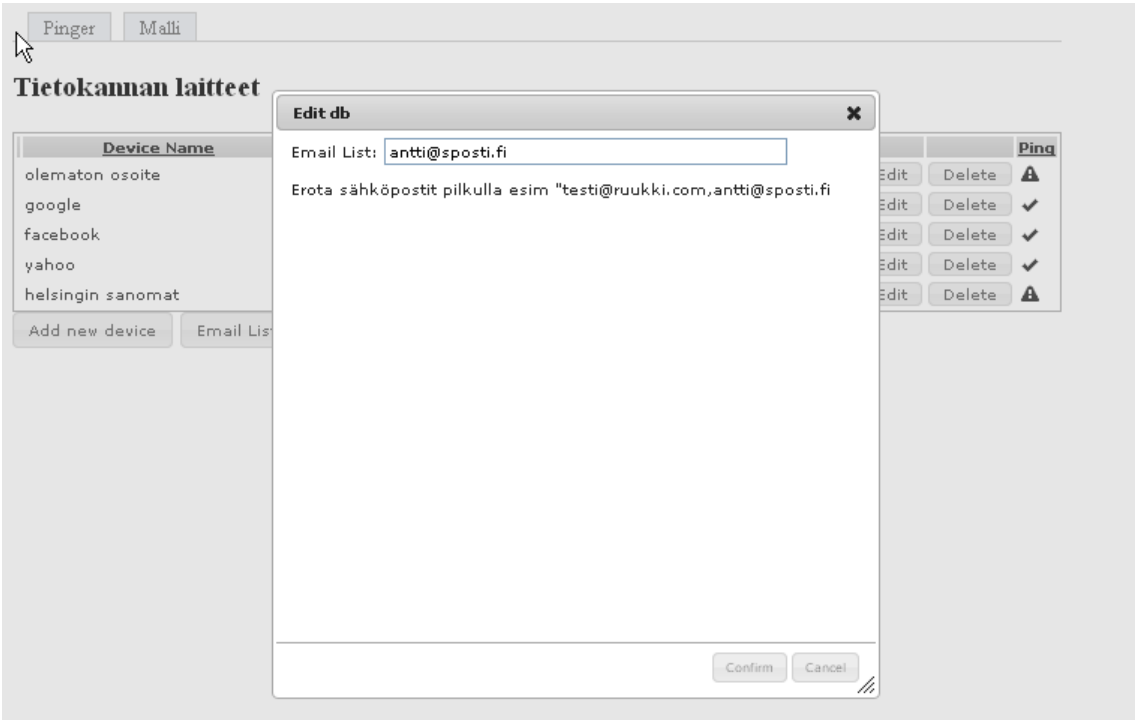
KUVA 11. POST:in vastaanottaminen ja käsittely.

Tässä raportissa on esitelty vain, miten muokkausdialogi toimii, koska muut dialogit ovat pohjiltaan täysin samanlaisia.

4.3 Ulkoasu

Ulkoasu on pitkälti toteutettu JQuery UI:n mukana tulleilla valmiilla tyyleillä, sivun osiolle voi antaa luokkanimiksi JQuery UI:n määrittelemiä nimiä, kuten ui-widget, jota WebGrid käyttää. Valmiiksi määritellyn nimen antaminen tyylittelee sisällön JQuery UI:n verkkosivuilla olevalla verkkotyökalulla määritellyllä tavalla, esimerkiksi nappien ulkonäkö ja väri ovat muokattavissa siellä.

WebGridin ja sen sarakkeiden leveyksiä säädetään gridStyle.css -tiedostosta, jossa on määritelty myös laitteen tiedoista kertovan dialogin asettelu.



Kuva 12. Sähköpostilistan dialogi, jonka ulkoasu tulee JQuery UI:n kautta.

Sivuston tyylin tekemin helposti jatkossa uudelleenkäytettäväksi oli todella helppoa ASP.NET:issä, koska siitä löytyy mahdollisuus käyttää ulkoasusommitelmia (layout). Ensin luodaan pohja (katso kuva 13), mihin voit lisätä esimerkiksi jokaisella sivulla näkyvät linkit, tyylitiedostot tai yhteystiedot, ja ilmoitetaan mihin kohtaan pohjassa uuden sivun tai webohjelman sisältö sijoitetaan. Uutta sivua tehtäessä ilmoitetaan alussa vain layoutin sijainti (katso kuva 14), ja loput kirjoitetusta koodista sijoittuu layoutissa sille ilmoitettuun kohtaan.

```
<!DOCTYPE html>

<html lang="en">
    <head>
        <meta charset="utf-8" />
        <title>Verkko-ohjelmat</title>
        @RenderPage("~/Shared/scriptList.cshtml")
    </head>
    <body>
        <div id="container">
            <div id="linkcontainer"> @RenderPage("~/linkit.cshtml")</div>

            <div>
                @RenderBody()
            </div>
        </div>
    </body>
</html>
```

Kuva 13. Layout-pohja, @RenderBody() ilmoittaa kohdan mihin uusien sivujen tiedot lisätään.

```
@{
Layout="~/Shared/SiteLayout.cshtml";
}
@RenderPage("~/Pingaus/Grid.cshtml")
```

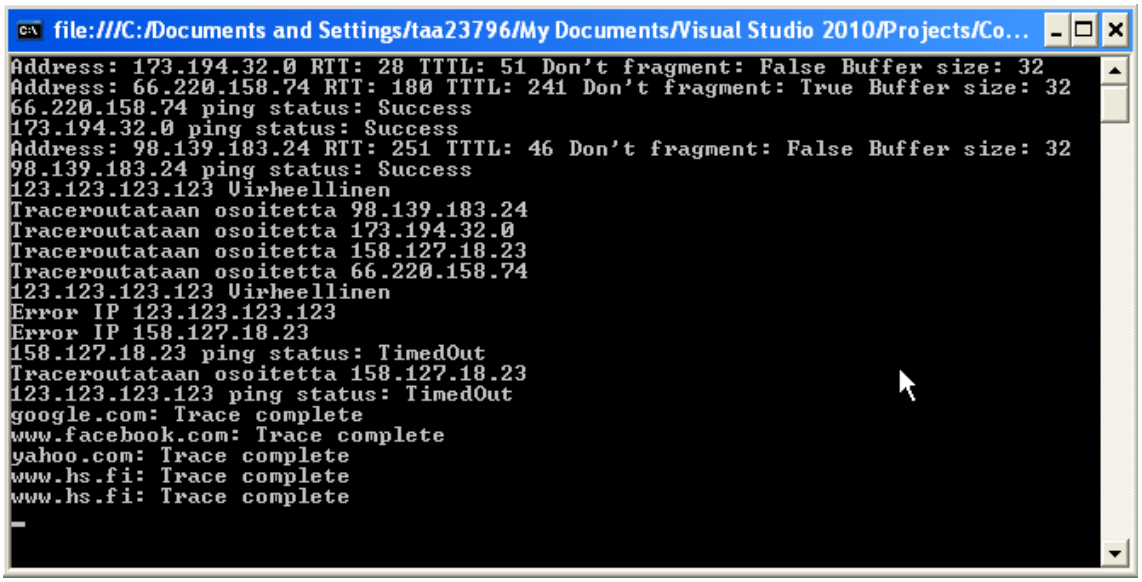
Kuva 14. Layoutin kutsu, @RenderPage(”~/Pingaus/Grid.cshtml”) sijoittaa Grid.csthml:stä löytyvän sivun Layoutiin.

5 PINGER-OHJELMA

Pinger on konsolipohjainen, Microsoft Visual Studio 2010 ympäristössä C# kielellä totetutettu ohjelma, jonka päätarkoitus on pingata ja traceroutata määritellyt laitteet. Pinger lähettää myös käyttäjän määrittelemissä tapauksissa sähköpostia.

Pinger toimii sykleissä, joiden aikavälin voi ohjelmaa käännettäessä määritellä (käytössä olevassa versiossa 5 minuuttia). Ohjelma suorittaa ensimmäisen syklin käynnistyessään ja seuraavan kerran määritetyn ajan kuluttua. Kierroksen aluksi ohjelma tarkistaa milloin viimeisin traceroute on suoritettu ja pitäisikö se suorittaa uudelleen. Tieto tracerouten tarpeellisuudesta saadaan, kun viimeisimmän tracerouten päiväys tallennetaan muuttujaan ja sitä verrataan nykyiseen päiväykseen. Riippumatta tarvitaanko traceroutea, sykli suorittaa aina pingauksen. Traceroutauksesta vastaa aliohjelma nimeltä TraceList ja pingauksesta PingList. Traceroutesta ja pingauksesta enemmän tietoa myöhemmin.

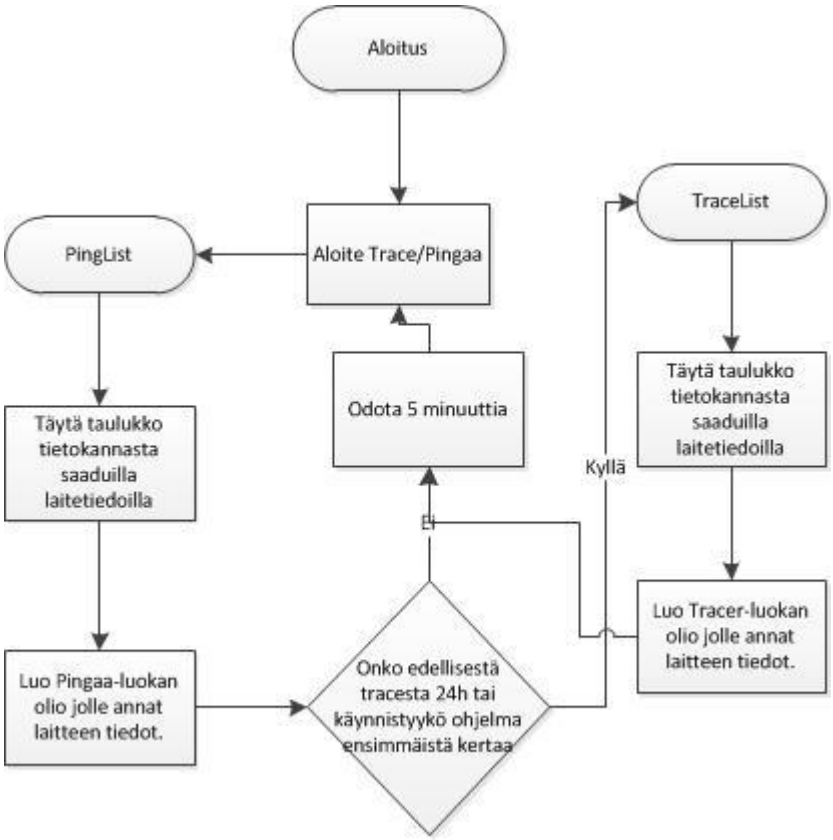
Seuraavissa kappaleissa puhutaan myös säikeistä, jotka tarkoittavat ohjelman itsenäisesti suoritettavia osia. Säiettä suorittaessa ohjelma ei jää odottamaan sen valmistumista, vaan se pyörii taustalla ja valmistuu, kun valmistuu. Tämän ohjelman säikeet ovat melko lyhytikäisiä ja rasittavat konetta suhteellisen vähän. Asiasta enemmän jatkokehitys-kohdassa.



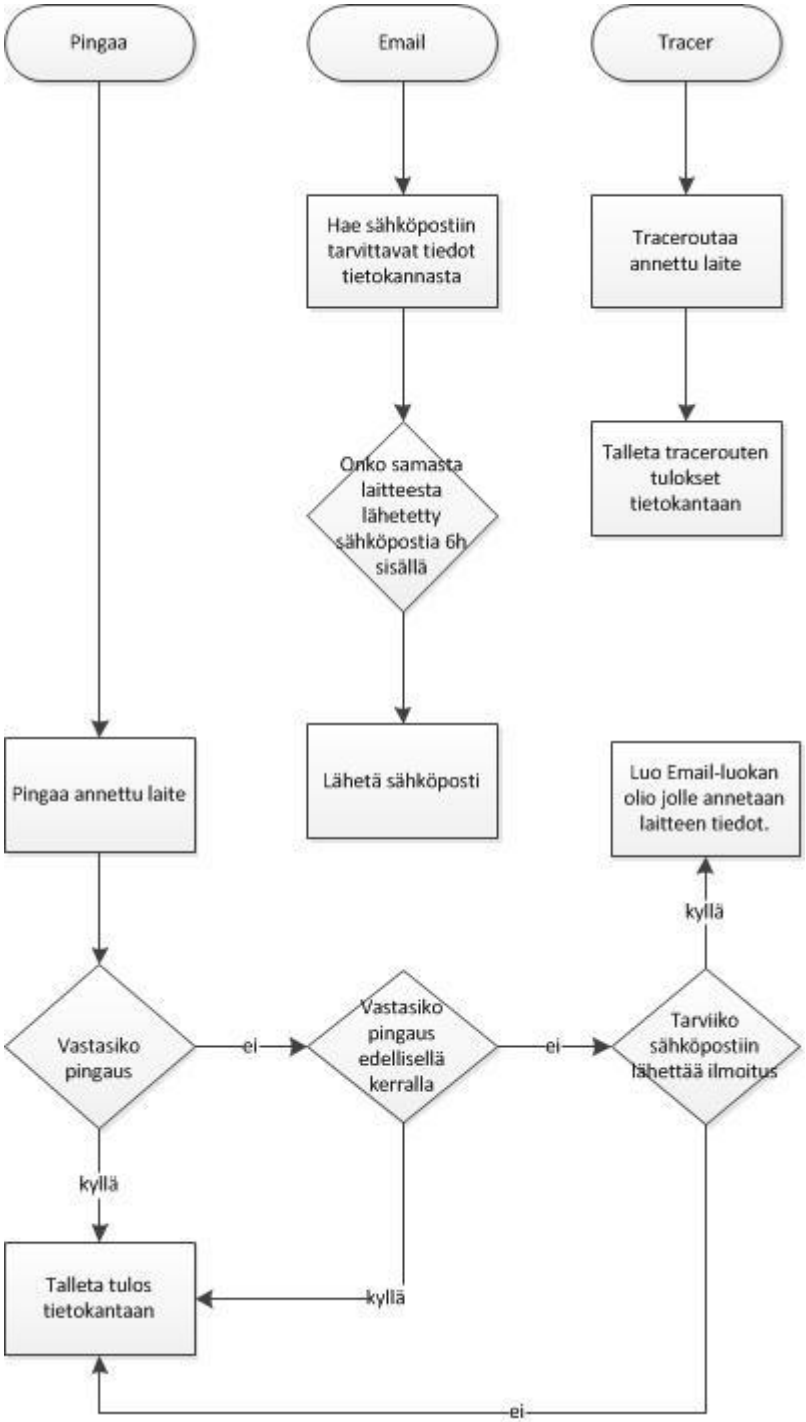
KUVA 15. Pinger toiminnassa.

5.1 Vuokaavio

Vuokaavio pääohjelmasta ja luokista, tarkemmat selitykset seuraavassa kappaleessa.



Kuva 16. Vuokaavio pääohjelmasta



Kuva 17. Vuokaaviot ohjelman luokista.

5.2 Luokat ja aliohjelmat

Ohjelmaa varten on luotu 3 eri luokkaa: Pinger, Tracer ja Email. Luokka olio-ohjelmoinnissa tarkoittaa olion tietotyyppiä, ja se määrittelee sen ominaisuudet, menetelmät ja attribuutit.

Pinger -luokan oliota luotaessa sille annetaan pingattavan kohteen devId, IP-osoite ja tieto siitä, tarvitseeko lähettää sähköpostia. Luokan funktiot ovat:

- Pingaa
    - Suorittaa pingauksen ja kutsuu sen tapahduttua pingSender_PingCompleted -funktion.
- pingSender_PingCompleted
    - Katsoo onko pingauksen vastauksessa virheitä ja jatkaa DisplayReply funktioon, jos niitä ei ole.
- DisplayReply
    - Kirjoittaa tietokantaan ja konsoliin pingauksen tulokset ja epäonnistuneen pingauksen tapauksessa traceroutaa ja lähettää sähköpostia tarvittaessa.

Tracer -luokan oliolle annetaan traceroutattavan laitteen devId, Ip ja tieto mistä aliohjelmasta pingataan. Mistä pingataan kertoo luotiinko olio traceLista aliohjelmassa vai Pinger -luokan instanssin sisältä. Näin voidaan kertoa onko traceroute vajaa vai ei. Luokan ainoa funktio on

- Traceroute
    - Selvittää, mitä reittiä paketti liikkuu kohteeseen ja kirjoittaa sen tietokantaan.

Email -luokan oliolle annetaan sen laitteen devId, jonka tiedot sähköpostilla lähetetään. Luokka sisältää funktion:

- sendEmail
    - Hakee tietokannasta tiedot laitteesta, sen traceista ja viime ongelmasta ja muodostaa niiden pohjalta sähköpostin, jonka sitten lähettää.

Pääohjelmassa on 3 aliohjelmaa:

- ThreadMethod
    - Kutsuu PingList-liohjelmaa ja katsoo pitäisikö TraceList-aliohjelmaa suorittaa.
- PingList
    - Täyttää DataTablen tietokannasta ja luo sen tietueita vastaavat oliot jotka sitten pingaa eri säikeissä.
- TraceList
    - Täyttää DataTablen tietokannasta ja luo tietueita vastaavat oliot, jotka sitten traceroutaa eri säikeissä.

5.3 Tietokantahaut

PingLista- ja TraceLista- aliohjelmat käyttävät hyväkseen .NET Frameworkin DataTable -luokkaa. DataTableen voi säilöä haluamaansa tietoa vapaasti ja tässä tapauksessa se täytetään tietokannasta haetulla tiedolla. DataTable täytetään TableAdapter-luokan objektin avulla, jolla voidaan tehdä erilaisia ennalta Visual Studiossa määriteltyjä tietokantahakuja,

```
pingerdbDataSetTableAdapters.pingerTableAdapter adapter = new pingerdbDataSetTableAdapters.pingerTableAdapter();
pingerdbDataSet.pingerDataTable table = null;
table = adapter.GetData();
```

KUVA 18. DataTablen täyttäminen TableAdapterin avulla

Osa hauista tehdään käyttäen SqlConnection -luokan objektia,

```
using (SqlConnection con = new SqlConnection(Properties.Settings.Default.ConnectionString))
{
    con.Open();
    using (SqlCommand command = new SqlCommand("UPDATE dbo.status SET emailSent = getdate() WHERE ... SORT BY date DESC)", con))
    {
        command.Parameters.Add(new SqlParameter("devId", devId));
        command.ExecuteNonQuery();
    } //using sqlcommandin loppu
}
```

KUVA 19. Tietokannan päivittäminen SqlConnection-objektin avulla.

5.4 Pingaus

PingList -aliohjelma täyttää kutsuttaessa DataTablen tietokannasta haetuilla laitetiedoilla, kuten laitteiden IP:illä ja ID:illä. Tässä tapauksessa on määritelty GetData-metodi, joka hakee valitusta tietokannasta kaiken tiedon.

Tämän jälkeen ohjelma luo jokaista luodun DataTablen tietuetta kohden uuden säikeen, joka taas luo Pinger -luokasta uuden ”pinger” -nimisen olion, jolle annetaan aina kustakin tietueesta laitteen IP, ID ja tieto siitä, lähetetäänkö sähköpostia, ja ryhtyy suorittamaan Pinger-luokan funktiota Pingaa.

```
Pinger pinger = new Pinger(table[i].id, table[i].ip, table[i].email);
new Thread(new ThreadStart(pinger.Pingaa)).Start();
pinger = null;
```

KUVA 20. Olion luonti ja säikeen aloittaminen

Pingaa-funktio pingaa oliolle annetun IP:n funktion sisällä määritellyillä pingausasetuksilla ja pingauksen toteutuessa kutsuu pingSender_PingCompleted -funktiota. Pingausasetuksia ovat mm. paketin sisältönä lähetettävän datan määrittely, hyppyjen maksimimäärä (Time to Live (TTL)) ja kuinka kauan paketti säilyy hengissä ennen kuin tuhoaa itsenä ja pingaus katsostaan epäonnistuneeksi (timeout).

```
pingSender.PingCompleted += new PingCompletedEventHandler(pingSender_PingCompleted);
try
{
    pingSender.SendAsync(Ip, timeOut, buffer, pingOptions, waiter);
}
```

KUVA 21. Pingaus-metodille vietävät asetukset ja pingauksen valmistuessa kutsuttavan funktion kertominen

pingSender_PingCompleted tarkistaa, ettei pingauksesta tulleessa vastauksessa ole mitään virheellistä ja jatkaa sen jälkeen DisplayReply funktioon.

DisplayReply luo tietokantayhteyden ja tarkistaa ensin, onko kyseisessä Ip:ssä ollut viime tarkistuskerralla ongelmaa. Tämän jälkeen, jos uusin pingaus onnistui, kertoo, että ongelmaa ei enää ole. Jos pingaus epäonnistui toistamiseen, ohjelma tarkistaa, tarvitseeko kyseisessä tapauksessa lähettää sähköpostia ongelmasta. Jos sähköpostin lähettäminen on tarpeellista, luodaan Email -luokan email1 -niminen olio ja suoritetaan olion sendEmail funktio uudessa säikeessä. Sähköpostin lähettämisestä lisää kohdassa 5.6. Epäonnistuessa tehdään myös traceroute, että nähtäisiin, mihin kohti paketin matka tyssää.

5.5 Traceroute

TraceList täyttää DataTablen ja tekee jokaista DataTablen tietuetta kohti uuden Tracer-Luokan tracer -nimisen olion, jonka Traceroute funktion se suorittaa aina uudessa säikeessä.

Traceroute funktiossa on Pingaa -funktiosta tuttuja pingausasetuksia, sillä erotuksella, että Time to Liven arvo on alussa aina 1 eli paketti elää vain seuraavaan solmuun (reitittimeen) saakka. Reitittimeltä tullut pingin vastausviesti sisältää vastaajaan IP-osoitteen, joka tallennetaan trace-tauluun tietokannassa TTL:n numeroa vastaavaan kohtaan. Edellinen toistetaan haluttu määrä kertoja (tässä tapauksessa 20) ja joka kerta TTL:n arvoa nostetaan yhdellä, jotta saadaan aina seuraavaan reitittimen IP. Uutta IP:tä verrataan aina viime kierroksen vastaavaan ja, jos ne on ovat samat, funktion suoritus lopetetaan.

Riippuen siitä, suoritettiinko ohjelma TraceList aliohjelmasta vai Pinger-luokan sisältä, kirjoitetaan tietokantaan, että traceroute on ongelmallinen. Pingerin kutsumana tracerouten kohde ei ole vastannut pingiin ja traceroute on suoritettava, että nähtäisiin mahdollinen verkon ongelmakohta.

5.6 Sähköpostin lähettäminen

Sähköposti on helppo tapa ilmoittaa laitteen verkko-ongelmista, koska se on useimmiten kytketty käyttäjien puhelimiin ja ne ilmoittavat mahdollisista ongelmista käyttäjälle paikasta riippumatta.

Email -luokan olio lähettää 6 tunnin välein sähköpostin, jos laitteen pingaus on epäonnistunut 2 kertaa peräkkäin. Oliolle annetaan laitteen devId ja se hakee tietokannasta string -muuttujiin tiedot laitteen nimestä ja ip:stä, viimeisimmästä toimivasta traceroutesta, viimeisimmästä vajaasta traceroutesta, pingauksen antamasta virheestä, osoitteista, mihin sähköposti tarvitsee lähettää, sekä tiedon siitä, milloin sähköpostia on viimeksi lähetetty. Muuttujiksi on myös määritetty SMPT-serverin osoite, sen käyttämä portti, käyttäjänimi ja salasana. SMPT (Simple Mail Transfer Protocol) -protokolla on Tcp-pohjainen ja lähettää viestejä sähköpostipalvelimelta toiselle. Sähköpostipalvelimena tässä tapauksessa toimii palvelin, jolla ohjelma pyörii.

Sähköpostia varten luodaan mail -niminen MailMessage -luokan mukainen olio mail, jolle annetaan tarvittavat arvot, kuten kenelle viesti lähetetään, viestin aihe ja sisältö. Luodaan myös SmtpClient -luokan olio smtp, jolle kerrotaan SMTP-serverin osoite, portti ja tunnukset. Lopuksi smtp -olio lähettää mail -olion sisällön komennolla ”smtp.Send(mail);”

```
MailMessage mail = new MailMessage();
mail.From = new MailAddress(from);
mail.To.Add(@to);
mail.Subject = subject;
mail.Body = body;
SmtpClient smtp = new SmtpClient(smtpaddress, smtpport);
smtp.Credentials = new NetworkCredential(account, password);
ServicePointManager.MaxServicePointIdleTime = 1; //ilman tätä ei lähde samantien
```

KUVA 22. MailMessagen luonti ja SMTP -asetusten antaminen. Nämä lähetettäisiin komennolla ”smtp.Send(mail);”.

6 TESTAUS

Pääosa järjestelmän testauksesta tapahtui ohjelmoinnin ohella jatkuvasti. Aina kun uutta koodia tuli, testasin sen toimivuuden ja testailin erilaisia tapauksia, kuten väärien arvojen antamista. Lopussa testasin valmiilla ohjelmalla erilaisia vääriä arvoja ja miten se niihin reagoi. Tietyissä tapauksissa WebGrid lopetti näyttämästä riviä, jolla väärä arvo oli ja osa tapauksista jopa kaatoi pinger-ohjelman. Karsin ongelmia pois, enkä enää ole saanut pingeriä kaatumaan.

7 JATKOKEHITYS

Ohjelman kehityksen aikana mieleen tuli kymmeniä eri muutosehdotuksia ja parannuksia, joita ei aika- ja resurssisyistä laitettu valmiiseen työhön.

7.1 Selkeämpi käyttöliittymä

Ulkoasun saaminen selkeämmäksi olisi seuraavan iteraatiokierroksen ensimmäisiä muutettavia asioita. En ole täysin tyytyväinen mm. käyttöliittymässä avautuvan tietoikkunan sisällön asemointiin, eikä nyt ollut aikaa perehtyä siihen enemmälti. Muitakin pieniä käyttöliittymäniksejä olisi lisättävä mm. massatoimenpiteiden (esim. usean kohdan poisto) ja sähköpostin lähettämisen määrittäminen WebGridistä esimerkiksi checkboxien avulla.

7.2 WebGridille vaihtoehto

Huomasin työn edetessä, että WebGrid oli todella helppo saada toimimaan suoriltaan, mutta sen edistynyt muokkaus on todella hankalaa ja työlästä. Saadakseni esimerkiksi WebGridissä statuskuvat toimimaan, jouduin käyttämään lähes 10 tuntia löytääkseni oikean ja toimivan koodin. Koodinpätkä ei loppupeleissä ole hankalan näköinen tai pitkä, mutta vaati mm. sql-haun muokkaamista ja kymmenien eri koodivaihtoehtojen testausta, WebGrid ei joustanut ollenkaan, vaan tämä on ainoa tapa, jolla kuvat sai näkymään. Alla kyseinen pätkä koodia.

```
grid.Column("Ping", format: @<text>@if(item.probError==1) { @Html.Raw("<span class=\"ui-icon ui-icon-alert\"></span>") } else { @Html.Raw("<span class=\"ui-icon ui-icon-check\"></span>") }</text>)
```

7.3 Vähemmän tietokantaan kirjoittamista

Pinger-ohjelma tarvitsee tietokantaa paljon ja jatkossa ehkä parempi tapa olisi kirjoittaa enemmän tietoa kerralla, eikä aukaista yhteyttä monta kertaa useassa säikeessä SqlConnectionin avulla. Tiedon voisi esimerkiksi kerätä aiemmin esiteltyyn DataTableen ja kirjoittaa kaikki siihen kerätty tieto tietokantaan aina tietyin väliajoin.

7.4 Tietokannan datatyyppien ja sarakkeiden nimien selkiyttäminen

Tietokannan tietotyypit jäivät testivaiheessa valittuihin, jotka eivät välttämättä ole kaikista optimoiduimpia. TEXT-tietotyyppiä on käytetty monissa kohdissa, joissa VARCHAR-tietotyyppi olisi parempi. VARCHAR olisi parempi lyhyiden tekstinpätkien säilömiseen, kuin työssä käytetty TEXT joka on suunniteltu pitkien tekstien säilömiseen. Näin pienessä tietokannassa tietotyypeillä ei ole suurta vaikutusta, mutta suurissa tietokannoissa haut saattavat hidastua paljonkin väärien tietotyyppien vuoksi.

Tietokannan sarakkeiden nimet ovat hieman epäselviä ja niiden selkeyttäminen olisi suotavaa. Itse kyllä ymmärrän mitä ne tarkoittavat, mutta jos joku muu yrittää niitä katsoa, saattaa ongelmia syntyä.

7.5 Selkeämpi nimeäminen muuttujille ja aliohjelmille.

Osa muuttujien ja aliohjelmien nimistä on melko mitäänsanomattomia ja niiden tehtävä ei tule selväksi kovin helpolla. Selkeät ja yhdenmukaiset nimet helpottaisivat koodin ymmärtämistä.

7.6 Syötettyjen tietojen oikeellisuuden tarkistus

Käyttöliittymään syötetään IP:tä ja sähköpostiosoitteita. Jatkossa olisi hyvä jos esimerkiksi JQuery tarkistaisi, ovatko syötetyt tiedot oikeasti IP:tä ja

sähköposteja vai jotakin ihan muuta. Tällä varmisteltaisiin toimivampia ja todenperäisempiä ilmoituksia ongelmista.

7.7 Helpommin levitettävä ja asennettava paketti

Ohjelmaan voisi liittää tietokantojen luomisen, ja käyttöliittymän tietokantayhteyksistä voisi tehdä helpommin muutettavia, jotta ohjelmaa olisi mahdollista siirtää palvelimelta toiselle helposti ja vaivattomasti. Nyt tietokannan joutuu tekemään käsin ja sen asentamisessa uudelle palvelimelle menisi tunteja.

Ohjelma voisi esimerkiksi käynnistyessään tarkistaa tietokantayhteyden ja siellä olevat taulut. Jos oikeita tauluja ei löydy, se loisi ne ja käyttäisi niitä.

8 YHTEENVETO

Työssä kehitettiin haluttuja verkkoon kytkettyjä laitteita valvoja järjestelmä, jonka tarkoitus on katsoa, että laitteilla on jatkuva yhteys lähiverkkoon. Työ koostuu kolmesta osasta: tietokannasta, käyttöliittymästä ja pinger-ohjelmasta.

Työssä käytettiin ohelmointikielenä C# ja sen ohjelmoimiseen käytettiin Microsoft Visual Studio 2010 ohjelmointiympäristöä. Tietokantaohjelmistona toimii Microsoft SQL Server 2008 ja sitä hallittiin Microsoft SQL Server 2008 Management Studiolla. Verkkokäyttöliittymän ohjelmointiin käytettiin HTML-kieltä, ASP.NET MVC3 -verkko-ohjelmointikehystä, JQuery -javacriptkirjastoa ja CSS -tyylitiedostoja.

LÄHTEET

1: Archer, Tom. 2001. Inside C#. S.xiii

2: Bai, Ying. 2009, Practical Database Programming With Visual C#.NET. S.12

3. Koenig, Chris. 2011, WebMatrix, Razor, MVC3 and Orchard Release Today!, hakupäivä 14.05.12, http://chriskoenig.net/2011/01/13/webmatrix-razor-mvc3-and-orchard-release-today/

4. W3Techs, http://w3techs.com/technologies/overview/javascript_library/all

5. Kaario. Kimmo, 2002, TCP/IP Verkot. S.257-259

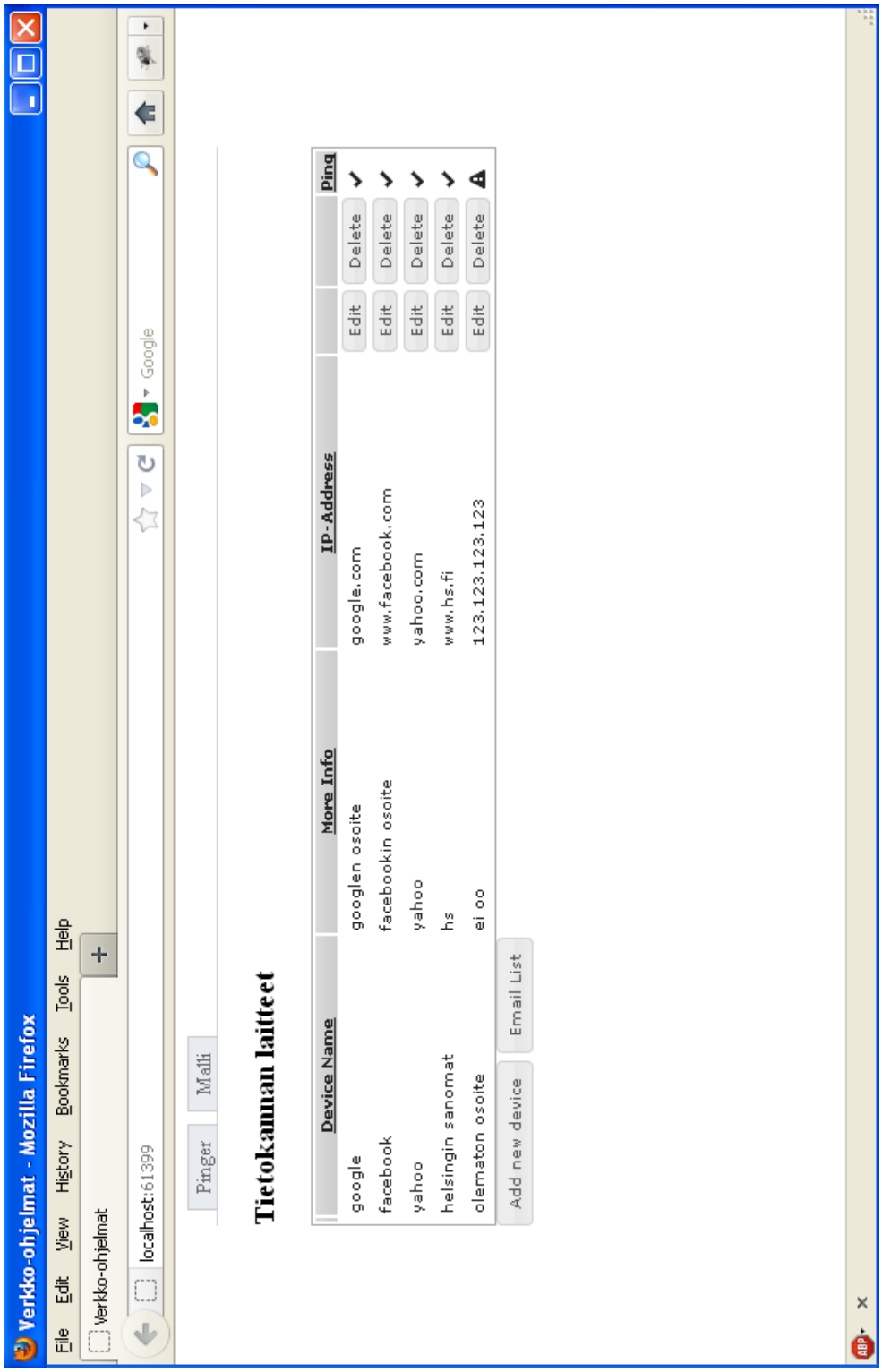
LIITTEET

LIITE 1 Kuvankaappaus käyttöliittymästä selaimessa

LIITE 2 JQuery -skritptien lähdekoodi

LIITE 3 Ohjelman pääluokan lähdekoodi, ilman sen sisältämiä luokkia.

LIITE 1: Kuva käyttöliittymästä



LIITE 2: JQuery -skriptien lähdekoodi

```
$(function () {
    "use strict";
    var tyyppi, id;

    //Muokkauksen dialogin aukasu
    $('#grid').delegate('.edit-db', 'click', function (e) {
        e.stopPropagation();
        $.getJSON('~/../Pingaus/Methods/dbQuery/' + $(this).attr('id'), function
(data) {
            var lista = data;
            $('#edit-id').val(lista.id);
            $('#edit-nimi').val(lista.name);
            $('#edit-info').val(lista.info);
            $('#edit-ip').val(lista.ip);
        });
        $('#action-type').val('edit');
        tyyppi = 'edit';
        $("#edit").show();
        $("#delete").hide();
        $("#emaillistdialog").hide();
        $('#dialog-edit').dialog('open');
    });
    //poistamisen dialogin aukasu
    $('#grid').delegate('.delete-row', 'click', function (e) {
        e.stopPropagation();
        $.getJSON('~/../Pingaus/Methods/dbQuery/' + $(this).attr('id'), function
(data) {
            var lista = data;
            $('#edit-id').val(lista.id);
        });
        $('#action-type').val('delete');
        $("#edit").hide();
        $("#emaillistdialog").hide();
        $("#delete").show();
        $('#dialog-edit').dialog('open');
    });
    // deviceinfon aukasu
    $('#grid').delegate('tbody tr', 'hover', function (e) {
        id = $(this).find('button:last').attr('id'); // ei pystynyt ottaan
inputin valuee, joten otetaan tr-elementin viimesestä buttonista (delete) id
        $(this).toggleClass('clickable');
    });
    $('#grid').delegate('tbody tr', 'click', function (e) {
        e.stopPropagation();
        $('#devInfo').load("~/../Pingaus/deviceInfo/" + id).dialog('open');
        //$('#devInfo').dialog('open');
    });
    //lisää laite  nappi
    $('#Add').button().click(function () {
        $('#edit-id').val('');
        $('#edit-nimi').val('');
        $('#edit-info').val('');
        $('#edit-ip').val('');
        $('#action-type').val('add');
        tyyppi = 'add';
        $("#edit").show();
        $("#delete").hide();
        $("#emaillistdialog").hide();
```

```
        $('#dialog-edit').dialog('open');
    });
    //Email list nappi
    $('#emaillist').button().click(function () {
        $.getJSON('~/../Pingaus/Methods/emailQuery', function (data) {
            var lista = data;
            $('#edit-emaillist').val(lista.email);
        });
        $('#action-type').val('email');
        tyyppi = 'email';
        $("#emaillistdialog").show();
        $("#edit").hide();
        $("#delete").hide();
        $('#dialog-edit').dialog('open');
    });

    $('#devInfo').dialog({
        autoOpen: false,
        modal: true,
        height: '500',
        width: '400',
        title: 'Device Info'
    });

    $('#dialog-edit').dialog({
        autoOpen : false,
        modal: true,
        height: '400',
        width: '400',
        buttons: {
            'Confirm' : function () {
                $.ajax({
                    type: "POST",
                    url: $("#edit-db-form").attr('action'),
                    data: $("#edit-db-form").serialize(),
                    dataType: "text",
                    success: function () {
                        $('#dialog-edit').dialog('close');
                        $("#grid").load('~/../Pingaus/Grid/ #grid', function () {
                            if (tyyppi === 'edit') {
                                $('#' +
id).parent('td').parent('tr').effect("highlight", {}, 2000);
                            } else if (tyyppi === 'add') {
                                $('tbody > tr:first').effect("highlight", {},
2000);
                            }
                        });
                    },
                    error: function (response) {
                        alert(response.responseText + "wat");
                        $('#dialog-edit').dialog('close');
                    }
                });
            },
            Cancel: function () {
                $('#dialog-edit').dialog('close');
            }
        }// end buttons
    });
});
```

LIITE 3 Ohjelman pääluokan lähdekoodi, ilman sen sisältämiä luokkia.

```
// v1.2
// 20.05.2011 Antti Taskila pingausconsole.cs
// Ohjelma lähettää pingin määritettyin koneisiin ja kirjoittaa consoleen vastaukset
// Ohjelma myös traceroutaa halutut koneet ja kirjoittaa tracen consoleen

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
    // sisältää IPAddress ja Dns luokat
using System.Net;
    // Sallii threadingin (useamman samanaikaisen ajon)
using System.Threading;
    // sisältää Ping luokan
using System.Net.NetworkInformation;
//using System.Net;
using System.Net.Mail;
using System.ComponentModel;
    // sisältää stopwatch luokan
using System.Diagnostics;
    // Voidaan kirjoittaa sqlkantaan
using System.Data.SqlClient;

namespace ConsoleApplication1
{
    class pingausconsole
    {
        static int Day = 0;
        static int pingtaajuus = 5*60; //kuinka usein pingaa sekunteina min*sek
        List<string> lista = new List<string>();
        static int Main(string[] args)
        {

            Timer t = new Timer(threadMethod, 1, 0, pingtaajuus * 1000);
            Thread.Sleep(Timeout.Infinite); // Simulating other work (10 seconds)
            t.Dispose(); // Cancel the timer now

            return 0;

        }
        //Metodi joka katsoo tarviiko suorittaa Trace vai pingi
        private static void threadMethod(Object state)
        {
               DateTime day = DateTime.Now;

               if (Day != day.Day)
               {
                   PingList();
                   TraceList();
                   Day = day.Day;
               }
               else
               {
                   PingList();
               }
            Thread.Sleep(1000);
        }

        private static void TraceList()
        {
            DataSet1TableAdapters.pingerTableAdapter adapter = new
DataSet1TableAdapters.pingerTableAdapter();
            DataSet1.pingerDataTable table = null;
            table = adapter.GetData();

            if (table != null)
            {
```

```
                if (table.Rows.Count > 0)
                {
                    for (int i = 0; i < table.Rows.Count; i++)
                    {
                        Tracer tracer = new Tracer(table[i].id, table[i].ip, 0);
                        new Thread(new ThreadStart(tracer.Traceroute)).Start();
                        tracer = null;
                    }
                    adapter = null;
                    GC.Collect();
                    Thread.Sleep(2500);
                }
            }
            else
            {
                Console.WriteLine("Taulu tyhjä");
            }
        }

        private static void PingList()
        {
            try
            {
                DataSet1TableAdapters.pingerTableAdapter adapter = new
DataSet1TableAdapters.pingerTableAdapter();
                DataSet1.pingerDataTable table = null;
                table = adapter.GetData();

                if (table != null)
                {
                    if (table.Rows.Count > 0)
                    {
                        for (int i = 0; i < table.Rows.Count; i++)
                        {
                            Pinger pinger = new Pinger(table[i].id, table[i].ip,
table[i].email);

                            new Thread(new ThreadStart(pinger.Pingaa)).Start();
                            pinger = null;
                        }
                        adapter = null;
                        GC.Collect();
                        Thread.Sleep(2500);
                    }
                }
            }
            catch(Exception e)
                {
                    Console.WriteLine(e.Message);
                }
        }
}
}
```