

**Universitatea „Politehnica” Timișoara**

**Facultatea de Automatică și Calculatoare**

**Programul de Licență: #tbd**

**See Now**

Apicație de image storage și identificare de persoane

**Lucrare de licență**

Conducător: Student:

Prof. dr. ing Răzvan-Dorel CIOARGĂ Răzvan-Alexandru BORDUZ

Timișoara,

Iunie 2018

Cuprins

[1 Introducere 4](#_Toc482813935)

[1.1 Scurt Istoric 4](#_Toc482813936)

[1.2 Contextul și domeniul temei 4](#_Toc482813937)

[1.2.1 Tehnologia Android 4](#_Toc482813938)

[1.2.2 Algoritmi detecție facială 6](#_Toc482813939)

[1.2.3 Baze de date 6](#_Toc482813940)

[1.3 Tema propriu-zisă 9](#_Toc482813942)

[2 Medii și tehnologii de programare folosite 10](#_Toc482813943)

[2.1 Android & Android Studio 10](#_Toc482813944)

[2.2 MySQL Server 11](#_Toc482813945)

[2.3 OpenCV 3.0 13](#_Toc482813946)

[2.4 Ubuntu 16. 14](#_Toc482813947)

[2.5 Microsoft Visual Studio 14](#_Toc482813948)

[2.6 Github & Tortoise SVN 14](#_Toc482813948)

[2.6 Adobe Photoshop CS3 14](#_Toc482813948)

[2.7 Notepad++ 14](#_Toc482813948)

[2.8 Descrierea limbajelor de programare 15](#_Toc482813950)

[2.8.1 Java 15](#_Toc482813951)

[2.8.2 Python & OpenCV 16](#_Toc482813952)

[2.8.3 PHP 17](#_Toc482813953)

[2.8.4 MYSQL 17](#_Toc482813954)

[2.9 Apache2 14](#_Toc482813948)

[3 Documentație tehnică a aplicației 20](#_Toc482813955)

[3.1 Design 20](#_Toc482813956)

[3.2 Implementare 24](#_Toc482813957)

[3.2.1 Baze de date 24](#_Toc482813957)

[3.2.2 Server 24](#_Toc482813957)

[3.2.3 Algoritmi detecție facială 24](#_Toc482813957)

[3.2.4 Interfață utilizator 40](#_Toc482813958)

[3.2.4 Interfață utilizator](#_Toc482813958) 44

[4 Manual de utilizare al aplicației 42](#_Toc482813959)

[5 Concluzii 49](#_Toc482813962)

[6 Referințe bibliografice 51](#_Toc482813963)

# Introducere

## Scurt Istoric

În 1972 Theodore George “Ted” Paraskevakos a dezvoltat un sistem de monitorizare cu senzori, care folosea transmisia digitală atât pentru securitate, incendii și alarmă medicală cât și pentru citirea contoarelor de utilități. Această tehnologie a fost una revoluționară pentru sistemul automat de indentificare a liniei telefonice, acum cunoscut sub numele de ID apelant.

În 1974 Dl. Paraskevakos a fost patentat de către U.S. pentru această tehnologie. În 1977 el a lansat  [Metretek, Inc.](http://www.metretekfl.com/) care a produs și a dezvoltat primul contor de citire automat la distanță. [1]

Momentul în care acest sistem a cunoscut o mare dezvoltare a fost abia în anul 1985 când au fost înființate mai multe astfel de proiecte la o scară largă. Hackensack Water Co. and Equitable Gas Go. au fost primii care au realizat astfel de proiecte bazate pe AMR pentru contoare de apă și contoare de gaz. În 1987 Philadelphia Electric Co. confruntându-se cu multe contoare inaccesibile au instalat mii de unități de distribuție AMR pentru a rezolva aceasta problemă. De atunci această tehnologie a devenit din ce în ce mai viabilă zilnic. [1]

Datorită progresului tehnologiei (a microprocesoarelor, a costului tehnologiei pe suprafață si a tehnicii de asamblare) se pot produce sisteme de încredere la un preț accesibil care justifică folosirea acestor sisteme AMR pe o scară largă.

## Contextul și domeniul temei

### Tehnologiei AMR

Automatic meter reading (AMR) este o tehnologie ce citește date automat de la aparatele de măsurat precum apometre, aparate de masură a energiei și transferă aceste date unei baze de date centrală pentru analiză.

Sistemele AMR constă din trei părți principale : [2]

* Contoarele de apă, gaz sau curent sunt echipate cu o interfață electromecanică/electro-optică care convertește semnalele analogice transmise de contoare în informație electronică ce poate fi prelucrată și trimisă spre centru. În multe cazuri interfața de comunicare este bidirecțională pentru a putea permite senzorului să primească informație de la centru. Pentru a putea procesa semnalele inițiale transmise de interfața contoarelor e nevoie de o unitate de control. Aceasta este formată din sursă de alimentare, microcomputer, memorie RAM și circuite electronice necesare operațiilor de convertire.
* Sistemul de comunicații utilizat pentru transmiterea datelor. De obicei datele pot fi trimise de la senzori la centrul de utilități prin telefon, linie de curent, satelit, cablu sau radio frecvență. Aceste date sunt prelevate constant în timp real.
* Echipamentul centrului de utilități include modemuri, receivere, un domeniu de hosting, unul sau mai multe calculatoare care adună informația.

Principalul scop al acestor contoare a fost să faciliteze obținerea datelor. De exemplu, multe contoare sunt instalate în locații care necesită întâlnirea cu proprietarul pentru a putea avea acces la index. Pentru că accesul la contoare era îngreunat factura la utilități de multe ori era calculată pe baza estimărilor celor 12 luni calendaristice.

Astfel unul din avantajele acestei tehnologii este că furnzorii de utilități nu mai sunt nevoiți să se deplaseze la locația fizică a senzorului, ci pot citi datele de la aparatele de măsurat indiferent de dificultățile de acces în spațiile unde sunt montate și au o mare libertate a alegerii momentului citirii. [4]

De asemenea un alt avantaj al acestor sisteme este faptul că asigură corectitudinea indecșilor citiți cât și economia de timp pentru proprietarii de apartamente. Consumatorii pot vedea astfel în orice moment câte utilități au consumat. [5]

Cu ajutorul unui astfel de sistem se poate depista dacă există pierderi în sistem sau posibilitatea de fraudă.

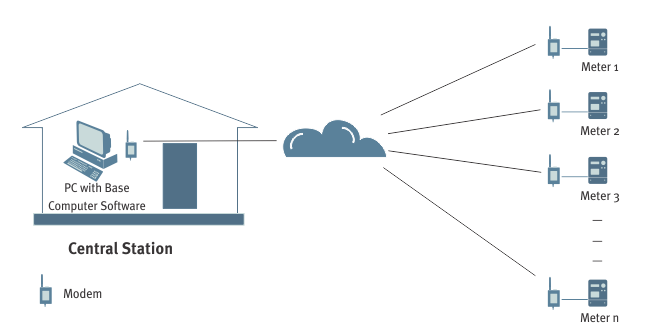


Figure : Structura unui sistem AMR, [7]

### Rețelele de comunicație pentru sisteme AMR

Principala calitate a acestor sisteme este comunicarea bidirecţională de pachete în ambele sensuri: contoare și sistem. Există patru tipuri principale de rețele de comunicații AMR: prin linii electrice purtătoare (PLC), prin rețeaua de telefonie mobilă/internet și prin unde de frecvență radio. [3]

Transmiterea datelor prin linii de curent se face prin aceleași linii prin care este transmis curentul electric. Această tehnică presupune injectarea unui transportator AC de înaltă frecvență și modularea acestuia cu datele originale de la stația centrală. Problemele tehnice ce pot apărea la transmiterea datelor fac ca această metodă sa nu fie foarte eficientă pe distanțe lungi. Această soluție este foarte scumpă iar comunicația este lentă. [4]

Transmiterea datelor prin liniile de telefonie este de dorit din punct de vedere economic deoarece majoritatea utilizatorilor de electricitate au pachete de servicii de telefonie. Sistemul de telefonie oferă infrastructura de comunicare ideală pentru sistemele AMR datorită simplității de funcționare, a calitații datelor, a fiabilitații și a costurilor reduse de implementare.

Dezavantajele acestui tip de comunicație sunt că datele nu pot fi transmise real-time la request și nici centrul de comandă nu poate accesa contoarele în cazul unor dereglări ale sistemului. [5]

Transmiterea datelor prin raze scurte de frecvențe radio se face prin Bluetooth, Wifi, Zigbee în funcție de banda de putere și frecvența semnalului. Probabilitatea ca contoarele de gaz, apă și current să împartă aceleași resurse este foarte mică deoarece companiile de utilități nu partajează infrastructura de rețea. [6]

Un astfel de sistem de transmitere pentru contoarele de gaz a fost implementat cu ajutorul unei unitați mobile (localizată în apropierea contoarelor) care trimite un semnal de trezire contoarelor din zona sa. Contoarele răspund acestui semnal prin transmisia datelor cerute de către sistem. Datele transmise ajung în computer-ul unitații mobile. [2]

### Topologii de rețele de comunicare wireless

Datorită dezvoltării tehnologiei SoC(System on a chip) în ultimii ani a fost posibilă dezvoltarea grupurilor de senzori wireless bazate pe WiFi și utilizarea lor în citirea automată a contoarelor. Această tehnologie permite trimiterea pachetelor de dimensiuni mari, un spectru de acoperire mare, accesarea directă a hotspot-urilor de WiFi fără a fi necesar un echipament adițional. [8]

În rețelele de senzori wireless topologia este elementul cheie ce joacă un rol important în minimizarea diferitelor constrângeri cum ar fi: energia consumată limitată, latența, criza de resurse de calcul și calitatea comunicării. [10]

Consumul de energie în aceste rețele depinde de numărul de pachete transmise și numărul de pachete primite. Consumul de energie de transmisie depinde de distanța dintre nodul transmițător și receptor, însă și dimensiunea pachetului joacă un rol vital în acest sens.

Topologiile frecvent utilizate în nodurile de senzori wireless sunt:

* Topologia Point to Point
* Topologia Star
* Topologia Mesh
* Topologia Hybrid
* Topologia Tree

Topologia Punct-la-Punct este formată dintr-o legătură wireless de capacitate mare între două noduri de senzori. Această topologie a fost utilizată cel mai frecvent în rețelele de senzori wireless. Principalul avantaj dar și dezavantaj este folosirea unui singur canal de comunicare de date. Avantajul este că acest singur canal este sigur însă comunicarea dintre cele două noduri va fi redusă. [24]

Topologia Stea constă dintr-un nod central, la care sunt conectate toate celălalte noduri. Acest nod central oferă un punct de conexiune pentru toate nodurile printr-un hub. Avantajele acestei tipologii sunt: consumul redus de energie și expansiunea ușoară și rapidă a rețelei. [24] Principalul dezavantaj este faptul că toată conexiunea se bazează pe nodul central iar dacă acesta nu mai funcționează nu există nicio altă cale alternativă în structură.

Topologia Mesh se bazează pe cooperarea nodurilor în distribuția datelor din rețea. Avantajele acestei topologii sunt faptul că: rețeaua este una mult mai sigură ca cele prezentate anterior, este o rețea scalabilă iar în sistem există o cale alternativă în momentul în care se pierde un nod. Dezavantajul acestei topologii este faptul că consumul de curent este foarte mare.

În topologia Hybrid senzori sunt aranjați într-o topologie Stea iar routerele sunt aranjate intr-o topologie Mesh. Această topologie preia toate avantajele altor topologii: conexiune sigură, consum redus, mai multe căi de trimitere a informației, însă extinderea acestei topologii este foarte grea. [24]

În topologia arbore senzorii sunt așezați sub forma unui arbore și există două tipuri de noduri: nod părinte și nod frunză. Pachetele sunt transmise de la un nod frunză la nodul părinte. Dezavantajul acestei topologii este timpul necesar pentru raspunderea la un request.

### Sisteme AMR ce folosesc tehnologia WiFi

În figura 2 este prezentat un sistem AMR ce utilizează tehnologia WiFi. Structura sistemului este formată din patru straturi. În cazul aplicației noastre s-a plecat de la premisa că sistemul este format din contoare, surse de WiFi, un server ce preia datele ce vin de la contoare și accesul la internet atât pentru client cât si pentru furnizorii de servicii ce colectează toate datele de la server și le analizează. [9]

Primul nivel este reprezentat de contoare (contoare de apă/gaz/curent) pentru fiecare apartament. Fiecare apartament este un cluster terminal(contor electric) care este responsabil cu colectarea datelor de la contoare prin comunicarea de tip WiFi. Contoarele de apă/gaz/curent sunt noduri WSN (Wireless System Network) care au implementat un modul de comunicare prin WiFi și sunt alimentate de bateri.

Al doilea nivel este nivelul rețea dintre contoarele electrice ale diferitelor apartamentelor și concentratorul (serverul). Acest contor electric poate stoca informații de la contoarele de gaz, apă și curent. El mai este denumit și contor deștept. Contoarele electrice sunt alimentate de la liniile de tensiune, fără a avea limitări legate de consum. Comunicația dintre concentrator și diferitele contoare poate fi văzută ca un arbore. Concentratorul este rădăcina iar contoarele electrice sunt nodurile arborelui.

Nivelul trei este partea dintre concentratorul principal și concentratorul de celule din fiecare clădire. Deoarece nodurile sunt foarte multe și sunt distribuite uniform structura celui ce-al treilea nivel este sub formă de stea.

Al patrulea nivel este nivelul dintre concentratorul principal și centrul de comandă. Comunicarea între cele două se poate face prin 3G, VPN, WiMAX etc.



Figure : Structura unui AMR bazat pe tehnologia WiFi , [8]

## Tema propriu-zisă

Această lucrare își propune să realizeze o platformă online de monitorizare și control al costurilor utilităților domestice. Aplicația permite utilizatorilor domestici vizualizarea consumului pentru utilități (vizualizarea indecșilor pentru apă, gaz, curent, vizualizarea prețurilor acestor utilități dar și vizualizarea totalului de plată pentru fiecare lună a anului calendaristic).

Implementarea a fost facută în contextul unui bloc cu nouă etaje, fiecare etaj având câte trei apartamente pe scară. Fiecare apartament trebuie să fie dotat cu contoare cu citire automată (contoare de tip AMR), fiecare având un cod unic prin care poate fi identificat de către sistem. Citirea contoarelor se face automat la cererea administratorului sistemului. Aplicația a fost implementată pentru un sistem AMR ce funcționează cu noduri de WiFi. Informația de la utilizator la centrul de comandă trece prin patru stagii. Modul concret de funcționare al unui astfel de sistem este descris în capitolul Introducere.

Lucrarea este compusă din două părţi principale: prima parte îşi propune să explice fundamentele teoretice necesare pentru înţelegerea domeniului din care face parte problema tratată, iar a doua conţine detalii de proiectare şi implementare a sistemului dorit, cu accent pe îmbunătăţirile aduse unor sisteme existente şi pe implicaţiile generate de acestea.

Capitolul 2 abordează toate fundamentele teoretice necesare pentru dezvoltarea acestei lucrări de licenţă. Printre acestea se numără descrierea mediilor de programare : T-SQL și C#, HTML, CSS , ASP.NET dar și lucrul cu MVC-uri (Model-View-Controller).

Capitolul 3 prezintă proiectarea propriu zisă a sistemului : proiectarea tabelelor, modul de lucru cu controllere, cu view-uri și lucrul cu modele . În acest capitol sunt prezentate mediile de lucru și limbajele de programare specifice.

Capitolul 4 exemplifică modul de lucru și accesare al platformei. Platforma permite accesul a două grupuri de utilizatori : utilizatorii normali (users) și administratorul sistemului (admin).Vor fi prezentate paginile principale din proiect, panourile de comandă pentru cele două tipuri de utilizatori.

# Medii și tehnologii de programare folosite

De-a lungul timpului, scopul unei aplicații a fost acela de a aduna informații dintr-o anumită locație, de a o prelucra după preferințele utilizatorului și pe urmă, de a o afișa utilizatorului. O metodă simplă de a creea o astfel de aplicație este de a aduna toate informațiile și a le prelucra pe toate ca un tot. Această metodă este bună în sensul că este ușor de implementat.

Ulterior însă când se dorește schimbarea uneia din componentele fluxului de date, apar probleme, spre exemplu atunci cand se dorește schimbarea interfeței. O altă problemă ține de logica de business ce trebuie încorporată, logică care și ea este supusă schimbărilor și care merge dincolo de simpla interschimbare de informație.

În astfel de cazuri apare nevoia de a împărții(modulariza) aplicația pentru a delimita părțile componente ale acesteia dar și pentru a putea fi ușor modificate. De asemenea componentele modificate trebuie să fie compatibile cu celelalte module ce formează aplicația.

O soluție la această problemă este arhitectura *Model-View-Controller (MVC)* care separă partea de stocare a datelor de cea de prezentare și deprelucrare.

Avantajele folosirii ASP.NET MVC sunt :

1. Separarea conceptelor. Reduce complexitatea unei pagini web.
2. Controlul integral asupra HTML-ului redat.
3. Respectarea designului fară stare a web-ului.
4. Include o componentă de mapare a URL-urilor pentru crearea de aplicații cu URL-uri simple, permițând optimizarea pentru motoarele de căutare.
5. Deoarece aplicația este împărțită în 3 părți, testarea se poate face mult mai ușor.
6. Framework extensibil. Framework MVC este proiectat pentru a fi uşor de înlocuit şi personalizat.
7. Nu utilizează conceptele ViewState sau PostBack pe server.
8. Suportă o varietate de librării pentru AJAX : JQuery, Dojo, Prototype etc.

## ASP.NET

ASP.NET este un framework de dezvoltare creat de Microsoft pentru realizarea paginilor web și website-uri folosind HTML, CSS și JavaScript. ASP.NET suportă 3 modele de dezvoltare diferite : ASP.NET Web Forms, ASP.NET MVC, și ASP.NET Web Pages. Acesta beneficiază de puterea platformei de dezvoltare .NET și de setul de instrumente oferite de mediul de dezvoltare Visual Studio .NET. [10]

Calitatea cea mai importantă a ASP .NET este timpul redus necesar dezvoltării aplicației. Tehnologia și uneltele de dezoltare de la Microsoft reduc timpul de dezolvatre a aplicațiilor web prin simplitatea limbajului de programare(c# sau Visual Basic .NET) și prin multitudinea de biblioteci de clase si controale .NET. [10]

ASP .NET a fost gândit inițial pentru dezvoltarea aplicațiilor de dimensiuni mari însă se pretează și aplicațiilor mici de genul site-uri personale sau aplicații front-end pentru o bază de date. Datorită tehnologiilor folosite de ASP .NET o aplicație web se poate crea scriind cantitatea minimă de cod.

Componenta .NET Framework stă la baza tehnologiei .NET, este ultima interfaţă între aplicaţiile .NET şi sistemul de operare şi actualmente conţine:

* Limbajele C#, VB.NET, C++ şi J#. Pentru a fi integrate în platforma .NET toate aceste limbaje respectă nişte specificaţii OOP numite Common Type System (CTS). Ele au ca elemente de bază: clase, interfeţe, delegări, tipuri valoare şi referinţă, iar ca mecanisme: moştenire, polimorfism şi tratarea excepţiilor.
* Platforma comună de executare a programelor numită Common Language Runtime (CLR), utilizată de toate cele 4 limbaje.
* Ansamblul de biblioteci necesare în realizarea aplicaţiilor desktop sau Web numit Framework Class Library (FCL).

## ASP.NET MVC

MVC a fost descris pentru prima oară în 1979 de către Trygve Reenskaug care pe vremea aceea lucra la Smalltalk din cadrul Xerox PARC. Implementarea originală este descrisă în detaliu în lucrarea Applications Programming în Smalltalk-80: How to use Model–View Controller.

MVC este un concept foarte răspândit în programarea Web.  Scopul MVC este de a ține separate logica business-ului și interfața cu utilizatorul, astfel încât cei care întrețin aplicația să schimbe mult mai ușor o parte, fără a afecta alte părți. MVC, sau Model-View-Controller este un șablon arhitectural folosit în industria de software development (inclusiv web development).

Framework-ul ASP.NET MVC prezintă o alternativă pentru WEB form-urile ASP.NET.

În organizarea MVC, modelul reprezintă informația (datele) de care are nevoie aplicația, view-ul corespunde cu elementele de interfață iar controller-ul reprezintă sistemul comunicativ și decizional ce procesează datele informaționale, facând legatura între model și view.

**Modelul** reprezintă partea logică a aplicației. El este responsabil de acțiunile și de operațiile asupra datelor, de autentificarea utilizatorilor, de integrarea diverselor clase ce permit procesarea informațiilor din diverse baze de date. Modelul răspunde interogărilor făcute de View și reacționează la schimbările produse în controller. Cu ASP.NET MVC, acesta este văzut mai ales ca un Data Acces Layer – DAL – de un anumit tip, utilizând de exemplu Entity Framework sau NHibernate combinat cu cod specific logicii aplicației. [12]

Pot fi folosite o varietate de tehnologii pentru a implementa logica de acces la baza de date, cum ar fi Microsoft Entity Framework, NHibernate, Subsonic, LINQ la SQL sau clase ADO.NET.

**View**-ul se ocupă de afișarea datelor (reprezintă interfața cu uitlizatorul UI), practic această parte a programului va avea grijă de cum vede end-userul informația procesată de controller. În view nu se fac prelucrări de date, calcule și nu se adaugă elemente decizionale. O dată ce funcțiile sunt executate de model, viewul-ui îi sunt oferite rezultatele, iar acesta le va trimite către browser. Aceste fișiere pot avea extensia html, asp, aspx, cshtml, vbhtml.

În general view-ul este o mini-aplicație ce ajută la redarea unor informații, având la bază diverse template-uri. Un view corespunde unui singur controller si vice versa. [12]

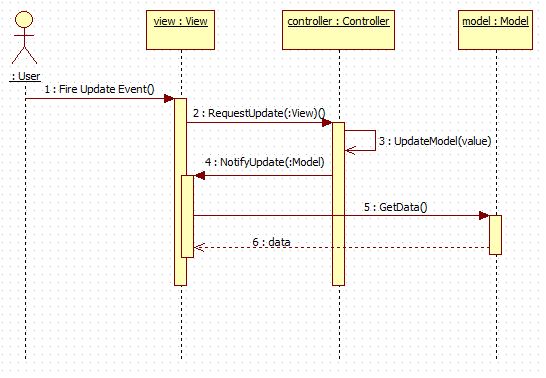
**Controller**-ul reprezintă nucleul aplicației. Acesta este o clasă specială ce face legatura între acțiunile userului și partea decizională a aplicației și între model si view. În funcție de nevoile utilizatorului, controller-ul apelează diverse funcții definite special pentru secțiunea de site în care se află user-ul. Funcția se va folosi de model pentru a prelucra (extrage, actualiză) datele, după care informațiile noi vor fi trimise către view, ce le va afișa apoi prin template-uri. Controller-ul interpretează inputurile userului și comandă modelului si view-ul să realizeze ajustările necesare. [12]

Numele controller-ului trebuie să conțină cuvântul cheie Controller. Un controller este o clasă ce extinde clasa System.Web.Mvc.Controller. Un controller expune acţiuni. O acţiune este o metodă în clasa controller-ului care este apelată atunci când utilizatorul introduce un URL în bara de adrese a browser-ului.

O acțiune a unui controller returnează răspunsul cererii de la browser. Framework-ul ASP.NET MVC suportă saşe tipuri standard de rezultat:

1. ViewResult – Reprezintă HTML.
2. EmptyResult – Reprezintă nici un rezultat.
3. RedirectResult – Reprezintă o redirectare spre un alt URL.
4. RedirectToRouteResult – Reprezintă o redirectare spre o altă acţiune a controlerului.
5. JsonResult – Reprezintă un rezultat JavaScript Object Notation ce poate fi folosit într-o aplicaţie AJAX.
6. ContentResult – Reprezintă un rezultat text.

Toate aceste clase ce pot fi returnate de o acţiune extind clasa de bază ActionResult. În cele mai multe cazuri, acţiunea controller-ului întoarce un ViewResult.

Model-ul, controller-ul și view-urile implicate în triada MVC trebuie să comunice unul cu altul pentru a putea controla interacțiunea cu userul. Comunicarea între view și controller-ul asociat este simplă, pentru că view-ul și controller-ul sunt special concepute pentru a lucra împreună. Modele pe de altă parte comunică într-o manieră mai complicată.

În *Application Programming in Smalltalk-80: How to use Model-View-Controller (MVC)*, Steve Burbeck descrie două variante de MVC: un model pasiv si un model activ.

Figure : Model Pasiv, [13]

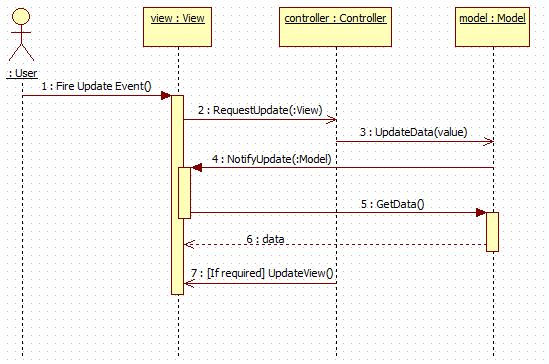
Modelul pasiv este cel mai simplu caz și este folosit când un controller manipulează model-ul exclusiv. În acest caz modelul este inactiv. Controller-ul execută toate operațiile necesare: modifică model-ul și informează view-ul că model-ul a fost schimbat și că trebuie reîmprospatat. În acest caz model-ul este complet independent de view și controller, ceea ce înseamnă că model-ul nu are cum să raporteze schimbările sale de stare.

Figure : Model Activ, [13]

Nu toate modele pot fi pasive deoarece în multe cazuri modelul își schimbă starea. Astfel model-ul trebuie să anunțe viewul aferent. Dacă modelul se schimbă fără implicarea controller-ului acesta se numește model activ. Modelul depinde de view dar de asemenea notifică view-ul despre eventuale modificări.

Pentru a putea dezvolta această aplicație s-au folosit următoarele instrumente de dezvoltare : Microsoft Visual Studio 2014 Express Edition, SQL Server Management Studio 2014 și SQL Server 2014.

## Microsoft Visual Studio 2014 Express Edition

Platforma online de monitorizare și control al costurilor utilităților domestice a fost realizată în întregime în Visual Studio 2014 Express Edition. Acest tool a fost creat de Microsoft și este o aplicație software care oferă facilități complete pentru programatori(IDE). Un IDE constă în mod normal dintr-un cod editor sursă, de un debugger și instrumente pentru automatizarea procesului de build.

Visual Studio se folosește pentru realizarea de aplicații ASP.NET, Servicii Web XML, aplicații desktop și aplicații mobile. Acesta folosește platformele de dezvoltare Microsoft: Windows API, Windows Forms, [Windows Presentation Foundation](http://en.wikipedia.org/wiki/Windows_Presentation_Foundation), [Windows Store](http://en.wikipedia.org/wiki/Windows_Store) și [Microsoft Silverlight](http://en.wikipedia.org/wiki/Microsoft_Silverlight).

Două trasături importante ale acestui tool sunt IntelliSense(feature pentru auto completarea codului) și Code Refactoring(restructurarea codului existent). IntelliSense este foarte utilă deoarece mărește viteza de muncă prin oferirea de sugestii. Restructurarea codului existent oferă lizibilitate și reduce complexitatea ajutând la mentenabilitatea codului sursă.

Alte instrumente incorporate ale acestui tool sunt : un designer pentru Web Forms, web designer , designer de clasă și designer pentru schema bazei de date . Visual Studio suportă următoarele limbaje de programare : C,C++, Visual C++, Visual Basic .NET, C# și F#. De asemenea suportă XML/XSLT, HTML/XHTML, JavaScript și CSS.

Pentru realizarea acestei lucrări s-au folosit urmatoarele limbaje de programare :C#, HTML, JavaScript și CSS.

## Microsoft SQL Server Management Studio 2014

SQL Server Management Studio este un program software creat de Microsoft și care ajută la configurarea, managementul și administrarea tuturor componentelor Microsoft SQL Server. Aplicația ca design seamană foarte mult cu Visual Studio, ceea ce o face foarte familiară.

Acest tool combină un grup larg de instrumente grafice , cu un număr de editori script bogat pentru a oferi acces la SQL Server pentru dezvoltatori și administratori de toate nivelurile de calificare .

Un feature important al acestui tool este Object Explorer care permite userului să navigheze usor prin tot proiectul(șteargă, adauge, să facă operații asupra componentelor serverului). Permite vizualizarea bazelor de date pană în cel mai mic detaliu dar permite și lucrul cu baze de date, cu tabele, cu triggere.

## SQL Server 2014

Microsoft SQL ( Structured Query Language) Server este un sistem pentru managementul bazelor de date relaționale dezvoltat de Microsoft. Acest tool este folosit exclusiv pentru lucrul cu baze de date, pentru stocarea și stocarea informațiilor solicitate de către alte aplicații software, fie de pe același calculator sau de la cele care rulează pe un alt calculator într-o rețea.

Limbajele de programare acceptate de acest tool sunt : T-SQL și ANSI SQL.

Un feature nou adus de către această versiune a tool-ului este tehnologia cunoscută sub numele de Hekaton, care a fost dezvoltat de Microsoft Research. Hekaton este o bază de date, optimizată, stocată în memorie care ajută la procesarea tranzacțiilor online. În esență această tehnologie implementează tabele optimizate în memorie și proceduri stocate pentru a putea crește performanța bazei de date.

## Arduino UNO

**Arduino** este o companie open-source care produce atât plăcuțe de dezvoltare bazate pe microcontrolere, cât și partea de software destinată funcționării și programării acestora. Pe lângă acestea include și o comunitate uriașă care se ocupă cu creația și distribuirea de proiecte care au ca scop crearea de dispozitive care pot sesiza și controla diverse activități sau procese în lumea reală.

Arduino Uno este un microcontroler bazat pe ATmega328. Are 14 intrări / ieșiri digitale (din care 6 pot fi utilizate ca ieșiri PWM), 6 intrări analogice, un rezonator ceramic de 16 MHz, o Conexiune USB, o mufă de alimentare, un antet ICSP și un buton de resetare. Acesta conține tot ce este necesar pentru a sprijini microcontroller-ul; Uno diferă de toate plăcile anterioare prin faptul că nu utilizează FTDI USB-to-serial Driver chip. În schimb, acesta oferă Atmega16U2 (Atmega8U2 până la versiunea R2) programată ca USB-to-serial convertor.[25]

**Specificatii :** Microcontroler: ATmega328, Tensiune de lucru: 5V, Tensiune de intrare (recomandat): 7-12V, Tensiune de intrare (limita):  6-20V, Pini digitali: 14 (6 PWM output), Pini analogici: 6, Curent per pin I/O: 40 mA , Curent 3.3V: 50 mA, Memorie Flash: 32 KB (ATmega328)  0.5 KB pentru bootloader , SRAM: 2 KB (ATmega328), EEPROM: 1 KB (ATmega328), Clock Speed: 16 MHz.

## Descrierea limbajelor de programare

### Arhitectura .Net și C#

C# a fost dezolvtat la sfârșitul anilor ’90 ca parte a ansamblului .NET dezvoltat de Microsoft. C# este direct înrudit cu C,C++ și Java și cuprinde câte puțin din fiecare, ex: cuvintele cheie și operatorii sunt moșteniți de la C.

C# a adus multe facilitați inovatoare, cele mai importante dintre ele fiind suportul încorporat pentru componentele software și posibilitatea de a lucra într-un mediu cu limbaj mixt. Acest limbaj de programare are o legatură directă cu arhitectura .NET deoarece el a fost dezvoltat pentru această arhitectura dar și datorită faptului că toate bibliotecile utilizate de C# sunt bibliotecile arhitecturii .NET.

Arhitectura .NET a definit două entități importante : motorul comun de programare(CLR) și biblioteca de clase .NET.

Motorul comun de programare se ocupă de execuția programelor C#. El este responsabil cu programarea într-un limbaj mixt, cu securitatea și portabilitatea programelor. Din urma compilării programelor scrise într-un limbaj mixt nu rezultă un cod executabil ci rezultă un fișier scris într-un limbaj intermediar IL. Acest fișier este portabil pe toate calculatoarele ce dispun de .NET CLR. Codul intermediar este transformat în cod executabil doar cu un compilator numit JIT(Just in Time).

Limbajul C# fiind un limbaj orientat pe obiecte conţine cele 4 principii esenţiale ale POO : încapsulare, polimorfism, moştenire şi reutilizare.[11]

Încapsularea este mecanismul prin care se combină codul și datele, menținându-le integritatea față de lumea exterioară. Codul și datele încapsulate creează un obiect. Obiectele pot fi publice sau private. Clasa este unitatea fundamentală a încapsularii.

Polimorfismul este calitatea care permite unei interfețe să aibe acces la un grup generic de acțiuni : „O singură interfață mai multe metode”.

Prin moștenire un obiect poate dobândi caracteristicile altui obiect. Un obiect poate fi o instanță a unui caz general.

Pentru a putea reduce timpul de muncă o clasă poate fi reutlizată de o mulțime de obiecte. Astfel deoarece nu mai este nevoie să testăm clasa respectivă ci doar să o utilizăm, timpul de muncă poate fi mult îmbunătățit.

### T-SQL

Transact SQL este o completare adusă limbajului SQL de către Microsoft și Sybase.SQL este acronimul de la Structured Querry Language și se folosește pentru interogarea, modificarea și definirea bazelor de date relaționale, folosind declarații declarative.

T-SQL extinde standardul SQL pentru a include programarea procedurală, variabile locale, diferite funcții de sprijin pentru prelucrarea șirurilor, a datelor etc. Aceste caracteristici suplimentare fac ca acest limbaj procedural să fie complet.

Acest limbaj permite interogarea unei baze de date, inserarea, updatarea, ștergerea, crearea, adăugarea datelor într-o bază de date. Cele mai cunoscute funcții sunt : AVG, COUNT, MAX, MIN, SUM, funcții pentru lucrul cu fișiere, pentru lucrul cu date :DATEADD, DATEPART, GETDATE(), SYSDATETIME, MONTH etc.

Principalele caracteristici ale limbajelor procedural pentru lucrul cu tabele sunt : procedurile(crearea, chemarea, ștergerea, redenumirea), triggerele(creare, ștergere, activarea și dezactivarea), cursori, view-urile și constrângerile(primary key, unique key, foreign key, etc).

### HTML

HTML(HyperText Markup Language) este un limbaj descriptiv, scheletul oricărei pagini Web ce descrie modul în care documentele sunt vizualizate și distribuite pe Internet. Acest limbaj descrie elementele structural ale unei pagini Web: tabele, paragrafe, liste, titluri, legături cu alte pagini etc. HyperText înseamnă ”text cu link-uri în el”. [14]

Un fișier HTML are următoarea structură:

1. zona head(antet) cu etichetele <head> </head>
2. zona body(corp) cu etichetele <body> </body>

Documentele HTML implică o structură de elemente HTML imbricate. Acestea sunt indicate în documentul de tag-uri HTML, închise în paranteze unghiulare < >. În general gradul unui element este indicat de o pereche de taguri un start tag < > și o etichetă de final</ >. HTML poate transforma un text în imagini, tabele, liste link-uri, form-uri etc.

Eticheta de start poate include de asemenea atribute în tag, atribute ce indică alte informații, cum ar fi identificatori de secțiuni din documentul de identificare folosit pentru a lega informațiile de stil la prezentarea documentului(ex: eticheta <img> folosit pentru a încorpora imagini).

Cele mai cunoscute elemente de HTML sunt:

* titlu de pagină <title></title>
* paragrafe <p></p>
* fonturi titluri<h1></h1>, <h2></h2>, <h3></h3>, <h4></h4>, <h5></h5>, <h6></h6>
* liste <ul> <li></li> </ul>
* link-uri <a href=” ”></a>
* imagini <img src=” ”>
* tabele <table> <tr> <td></td> </tr> </table>
* formulare(form, input, textarea, select)

### CSS

CSS sau Cascading Styles Sheets, este o modalitate de a stiliza și prezenta HTML. Întrucat HTML este sensul sau conținutul, foaia de stil, CSS, este prezentarea acestui document. Un stil reprezintă o colecție de valori ale atributelor elementelor unui document, valori ce pot fi aplicate întregului document sau doar unei parți din acesta.Printre attribute amintim: mărimea, grosimea, marginile, tipul și culoarea fontului, paragrafe etc. [15]

Datorită folosirii tilurilor se reduce considerabil efortul depus pentru modificările adiționale aduse aspectului și aranjării elementelor în pagină. Operarea de modificări se face doar asupra foilor de stil ce stilizează elementele și nu asupra fiecărui element.

Într-un document se pot aplica 3 tipuri de stiluri: la nivel de element(stilurile sunt include ca attribute în cadrul etichetelor HTML), încapsulate(stilurile sunt incluse în secțiunea de <head> a documentului prin utilizarea marcajului <style>) și legate(stilurile sunt definite în fișiere separate de documentul HTML).[16]

Stilurile inline fac parte din corpul documentului HTML, spre deosebire de stilurile încapsulate și de foile de stiluri externe. Definițiile de stil inline se aplică prin folosirea atributului style în etichetele HTML standard. Deși aceste stiluri inline sunt scrise de fiecare dată într-un document HTML ele sunt prioritare față de stilurile încapsulate sau externe. Mai jos este prezentat un exemplu de stil inline.

<P **style**="color: red; font-family: Arial; font-weight: bold">

Cele trei tipuri de stiluri pot fi combinate într-un fișier HTML iar relațiile dintre diferite tipuri de stiluri crează efectul de cascadă. Standardul WEB însă indică:

* Foile .css sunt urilizate pentru definirea caracteristicilor care se aplică formatării parții commune a tuturor paginilor unei aplicații
* Stilurile încapsulate sunt utilizate pentru definirea caracteristicilor care se aplică la formatarea unui anumit element.
* Utilizarea stilurilor inline pentru definirea caracteristicilor ce se aplică formatării unui anumit element.

Clasele de stiluri permit definirea unui stil general, care este inserat oriunde este necesar prin intermediul unei singure referiri. Un exemplu de o clasă de stiluri pentru platforma online este prezentat mai jos.

.index-font{

font-size:large;

font-family:'Times New Roman';

text-align:center;

}

Stilul identificator este dat de modul în care acest stil este asociat unui element,prin intermediul atributului id. Definirea acestor stiluri este similară cu definirea claselor de stiluri.

#row-large-maint{

height:200px;

width:90%;

margin-top:10px;

margin-left:60px;

margin-bottom:10px;

padding-left:40px;

padding-top:80px;

padding-bottom:10px;

box-shadow: 2px 2px 5px #999;

background-image:url("../Pictures/Capture (1).jpg");

background-repeat:no-repeat;

background-position-x:top;

background-position-y:bottom;

opacity:0.7;

}

Poziționarea în pagină permite așezarea unui obiect într-un anumit loc, folosind coordonatele sale. Atributul cu care se poate realiza acest lucru este position, cu proprietațile left și top.

h2{

margin-left:10px;

font-size:25px;

font-family:'Times New Roman';

text-align:center;

position:absolute;

}

# Documentație tehnică a aplicației

Aplicația realizată în această lucrare este împarțită în trei parți și anume: Baza de date SQL realizată în Microsoft SQL Server, aplicația ASP.Net de tipul MVC (model-view-controller) si aplicația Arduino realizată în Arduino Genuino.

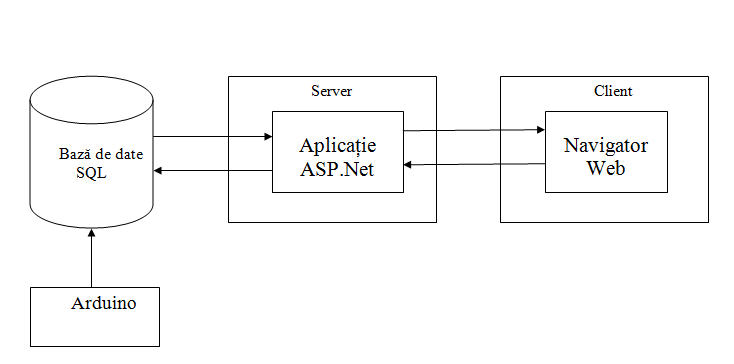


Figure : Arhitectura aplicației

## Arhitectura bazei de date

ADO .NET(Active X Data Objects) reprezintă o parte importantă a nucleului .NET Framework ce permite conectarea la surse de date diverse, extragerea, manipularea și actualizarea datelor. Sursa de date este în general o bază de date, dar ar putea fi un fișier text, un fișier Excel, un fișier Access sau un fișier XML. [17]

În activitațile zilnice este necesară accesarea și actualizarea informației dintr-o bază de date: lansarea unei comenzi, răsfoirea unui catalog de produse, verificarea evidenței plaților etc. Aceste informații sunt descrise cel mai bine folosind un model relațional ce presupune divizarea informației în seturi consistente și definirea relațiilor dintre aceste seturi.

O aplicație web are nevoie de un sistem complet de gestionare a bazelor de date cum ar fi SQL Server, care asigură gestionarea infrastructurii, fiabilitate și performanță.

Accesarea bazelor de date într-o aplicație web este diferită de accesarea unei baze de date prin intermediul unei aplicații desktop client-server. O problemă pentru aplicațiile tradiționale este faptul că odată ce conexiunea cu baza de date este deschisă, ea rămâne așa până la încheierea executării aplicației. Atunci când sunt deschise mai multe conexiuni server-ul de baze de date va răspunde mai lent la comenzile clienților.

O aplicație Web care lucrează cu baze de date trebuie să rezolve problema modului de deconectare de la Internet. Într-o aplicație de tip ASP .NET serverul procesează codul, returnează paginile HTML și închide conexiunea, astfel încât utilizatorul are senzația că lucrează cu o aplicație care rulează continuu deși el beneficiază doar de pagini statice.

ADO .NET permite lucrul în stil conectat dar și lucrul în stil deconectat,aplicațiile conectându-se la server pentru o scurtă perioadă de timp(doar pentru extragerea/actualizarea datelor). Principalele calitați ale ADO .NET sunt:

* Interopabilitate: permite interacționarea cu orice componentă care suportă XML
* Durabilitate: ADO .NET permite dezvoltarea arhitecturii unei aplicații datorită modului de transfer a datelor între nivele arhitecturale.
* Programabilitate: simplifică programarea pentru diferite task-uri cum ar fi comenzile SQL, ceea ce duce la o creștere a productivitații și la o scădere a numărului de erori.
* Performanță: nu mai este necesară conversia explicită a datelor la transferul între aplicații, fapt ce duce la creșterea performanțelor acestora.
* Accesibilitate: utilizarea arhitecturii deconectate permite accesul simultan la același set de date. Reducerea numărului de conexiuni deschise simultan determină utilizarea optimă a resurselor.

Prima etapă și cea mai importantă pentru o aplicație ce utilizează baze de date este analiza datelor și realizarea unui model conceptual corespunzător. Pe baza acestei etape se realizează baza de date. Este mult mai ușor să modificăm un model concceptual decât să modificăm o bază de date în care au fost adăugate deja date.

Crearea unui model conceptual presupune o realizare grafică a datelor și a relațiilor dintre acestea. Reprezentarea datelor și a relațiilor într-o formă convențională se numește diagramă entitați-relații sau ERD(Entity Relashionship Diagram).

Într-un model conceptual principalele concepte folosite sunt: atribut, entitate, relație. O entitate este un obiect pentru care se memorează date și are semnificație pentru problema modelată. Un atribut este o caracteristică a unei entitați. O relație este o asociere între două entitați și exprimă legătura dintre ele. [18]

Relațiile între două entitați pot fi de tipul:

* **1-1(one to one)**: unei instanțe a primei entitați îi corespunde cel mult o instanță a celei de-a doua entitați.
* **1-n(one to many**): o instanță a primei entitați poate fi asociată cu una sau mai multe instanţe ale celeilalte entităţi
* **n-n** (**many-to-many**): o instanţă a primei entităţi poate fi asociată cu una sau mai multe instanţe ale celei de-a doua entităţi şi pot exista mai multe instanţe ale primei entităţi asociate unei instanţe a celeilate entităţi.

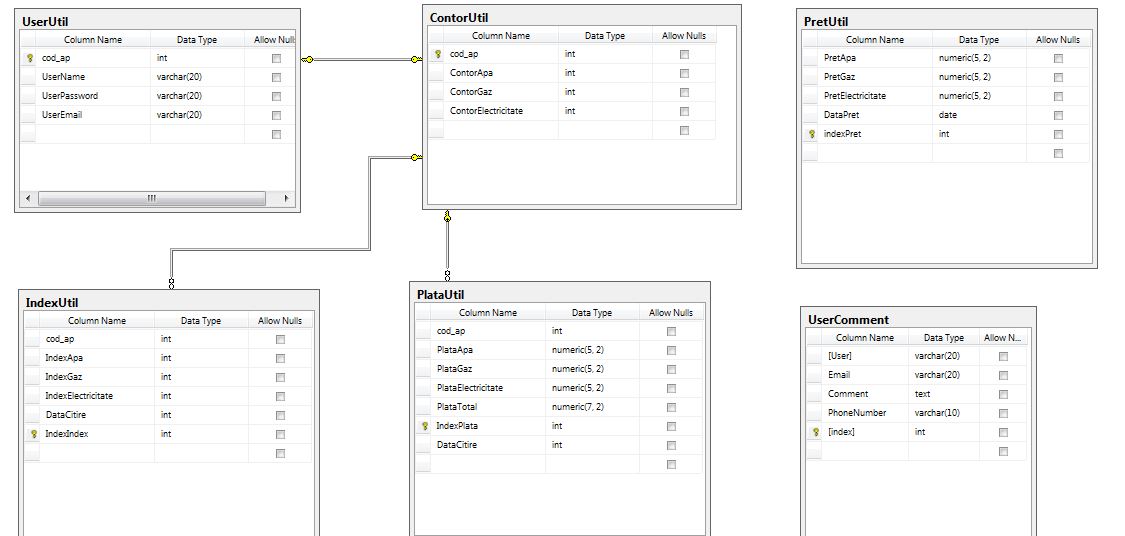
 Astfel în prima etapă am definit nevoile sistemului și am creat o bază de date necesară implementării cerințelor sistemului. Baza de date conține șase tabele fiecare particularizată pentru tipul de operații ce vor fi permise în cadrul ei. În următoarea figură este prezentată baza de date cu legăturile dintre tabele și tipurile de date pentru fiecare câmp al tabelei.

Figure : Baza de date

Pentru a facilita accesul la platforma fiecărui apartament din cadrul acestui bloc i-a fost atribuit un cod unic codificat în aplicație cu numele de cod\_ap.

Tabela principală este ContorUtil. Această tabelă conține toate codurile unice ale fiecărui apartament și toate codurile unice pentru fiecare contor din apartamentul respectiv. În tabelă, codurile contoarelor sunt ContorApa, ContorGaz, ContorElectricitate. Tabela a fost populată înainte ca userii să aibe acces la platformă. Cheia primară din această tabelă este cod\_ap deoarece codul fiecărui apartament este unic. De asemenea codurile contoarelor au fiecare câte o constrângere de unicitate.

Tabela UserUtil este tabela ce reține datele pe care user-ul le adaugă în momentul creeri contului la înregistrarea în platformă. Cheia principală în această tabelă este tot cod\_ap deoarece un cont se poate creea doar pentru un singur apartament. Cheia este în relație de one-to-one cu cod\_ap din tabela ContorUtil adică dacă în tabela ContorUtil nu există codul introdus de user, acesta nu își poate crea cont pe site.

Următoarele câmpuri din tabelă reprezintă datele personale pe care userul le introduce în momentul creeri contului. Câmpurile UserName (numele utilizatorului de pe site), UserPassword(parola cu care userul se v-a loga pe site) și UserEmail(E-mailul pe care userul poate fi contactat) sunt de tipul varchar(20) adică sunt stringuri de dimensiune fixă - 20 caractere. Câmpurile tabelei trebuiesc neapărat completate pentru că nu acceptă null.

Tabela IndexUtil salvează datele ce le primește de la contoare. Cheia primară în această tabelă este IndexIndex, un int cu auto increment de 1. Deși cod\_ap există în această tabelă câmpul cod\_ap nu poate fi cheie primară deoarece în fiecare lună se vor primi date de la contoare. Astfel fiecare cod\_ap apare în această tabelă de 12 ori. În fiecare început de an calendaristic tabela aceasta v-a fi ștearsă pentru a putea primi datele noi de la contoare. Tabela IndexUtil mai conține o constrângere de tipul unique între câmpurile cod\_ap și DataCitire. Cu ajutorul câmpului cod\_ap se iau datele trimise de contoare(pe baza cod\_ap se știe care contoare au trimis indexul pentru ce apartament).

Deoarece fiecare cod\_ap poate apărea de mai multe ori relația dintre tabela de CotorUtil și IndexUtil este one-to-many. Câmpurile din tabelă IndexApa(indexul pentru consumul de apă), IndexGaz(indexul pentru consumul de gaz), IndexElectricitate(indexul pentru consumul de electricitate), DataCitirii(luna calendaristică în care se face citirea) sunt de tipul int.

Tabela PlataUtil păstrează datele despre plată pentru fiecare apartament. Câmpul cod\_ap reține informația pentru un apartament. Câmpul DataCitire(luna calendaristică) se preia din tabela de IndexUtil. Câmpurile PlataApa(calculează totalul de plată pentru apă), PlataGaz(calculează totalul de plată pentru gaz), PlataElectricitate(calculează totalul de plată pentru electricitate), PlataTotal(calculează totalul de plată pentru toate utilitățile) sunt de tipul numeric(5,2) adică acceptă numere cu virgulă(5 cifre înainte de virgulă și 2 după virgulă). Cheia primară în acest tabel este IndexPlata un int cu auto increment de 1. Și aici la fel ca în tabela IndexUtil cod\_ap nu putea să fie cheie primară deoarece se calculează plata pentru fiecare lună a anului calendaristic. De asemenea la începutul unui nou an toate plățile pe anul trecut se șterg.

Tabela PretUtil este tabela care reține prețul pentru fiecare utilitate. Deoarece aceste prețuri se pot modifica a fost necesară introducerea unei astfel de tabele. Cheia primară în acest tabel este IndexPret un int cu auto increment de 1. Câmpurile PretApa(reprezintă prețul pentru 1mc de apă), PretGaz(reprezintă prețul pentru 1kWh/mc), PretElectricitate(reprezintă prețul pentru 1kWh) sunt de tipul numeric(5,2). Câmpul DataPret este de tipul Date(mm/day/year).

Pe site pe tabul de Maintenance există o secțiune de sugestii și reclamații. Pentru a putea aduna aceste sugestii și reclamații din partea userilor a fost necesară introducerea unei tabele UserComment. Câmpurile User(numele utilizatorului), Email(e-mailul utilizatorului), PhoneNumber(număr de telefon al utilizatorului) sunt de tipul varchar(20) respectiv varchar(10) iar câmpul Comment(userul introduce sugestia sau reclamația) e de tipul text. Cheia primară este index de tipul int cu auto increment de 1.

Pentru a putea lucra/ a avea acces la această bază de date a fost necesară crearea unei conexiuni. Această conexiune a fost asigurată de localhost-ul creat de SQL Server 2014. După instalarea acestui program și crearea unui server localhost s-a reușit conectarea bazei noastre de date la SQL Server(SQL1Express).

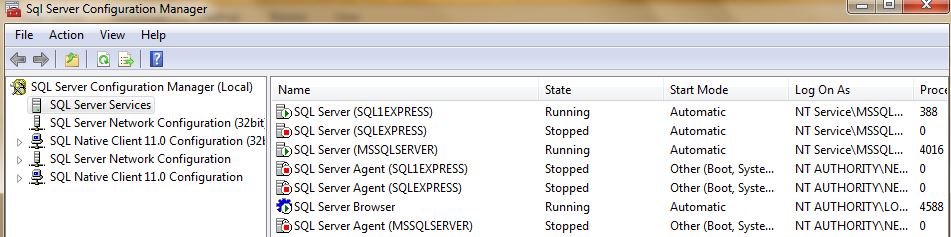


Figure : Serverul SQL1Express



În SQL Management pentru a putea avea acces au fost necesare anumite setări care permit conectarea la localhost-ul creat anterior. Pentru conectarea la server trebuie completate câmpurile din figura 6. Deoarece pentru acestă platformă s-a folosit o versiune de SQL Express numele de server este SQLExpress.

Figure : Autentificare SQL Management Studio

## Structura aplicației web

Aplicația a fost implementată în Visual Studio 2013. Și aici pentru a avea acces la baza de date a fost necesară crearea unui connection string. Un connection string este un șir care conține informații despre o sursă de conectare și mijloacele de conectare la aceasta. Lista de parametri necesară connection string-ului este: Provider(furnizorul de date pentru conectarea la sursa de date), Data Source(reprezintă serverul,în cazul aplicației fiind un server local), Ititial Catalog(numele bazei de date accesate), Integrated Security(logarea se face cu user-ul configurat pentru Windows), User Id(numele unui user care are acces de logare pe server), Password(parola pentru ID-ul specificat).

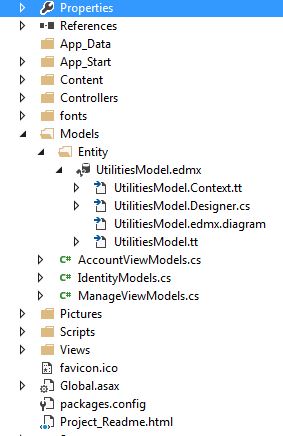
Deoarece ambele produse software, Visual Studio cât și SQL Management Studio sunt create de Microsoft, baza de date a putut fi exportată în Visual Studio. O dată cu acest export a fost creat și connection string-ul, un pas foarte important ce a ajutat la scăderea timpului de execuție al aplicației, deoarece un connection string poate fi destul de complicat de scris. Mai jos se poate observa acest connection string.

<connectionStrings><add name="utilitiesDBConnectionString" connectionString="Data Source=ROXANA-PC\SQL1EXPRESS;Initial Catalog=utilitiesDB;Integrated Security=True" providerName="System.Data.SqlClient" />

<add name="utilitiesDBEntities" connectionString="metadata=res://\*/Models.Entity.UtilityModel.csdl|res://\*/Models.Entity.UtilityModel.ssdl|res://\*/Models.Entity.UtilityModel.msl;provider=System.Data.SqlClient;provider connection string=&quot;data source=ROXANA-PC\SQL1EXPRESS;initial catalog=utilitiesDB;user id=sa;password=roxana;MultipleActiveResultSets=True;App=EntityFramework&quot;" providerName="System.Data.EntityClient" /><add name="utilitiesEntities" connectionString="metadata=res://\*/Models.Entity.UtilitiesModel.csdl|res://\*/Models.Entity.UtilitiesModel.ssdl|res://\*/Models.Entity.UtilitiesModel.msl;provider=System.Data.SqlClient;provider connection string=&quot;data source=ROXANA-PC\SQL1EXPRESS;initial catalog=utilitiesDB;integrated security=True;MultipleActiveResultSets=True;App=EntityFramework&quot;" providerName="System.Data.EntityClient" />

</connectionStrings>

În Visual Studio la crearea unui proiect nou putem vedea structura aplicației în Solution Explorer. Conceptul de MVC se observă datorită foldere-lor Controllers, Models și Views. Pe lângă acestea se poate observa un folder de App\_Data ce conține datele din baza de date, Content ce conține toate elementele de grafică, Scripts ce conține toate scripturile din proiect.

 Pentru a putea folosi Entity Framework trebuie creat un Entity Data Model. În Visual Studio pentru a ușura munca developeri-lor a fost adăugat un feature de Entity Data Model Wizard care generează un model după baza de date. În Solution Explorer pe folderul Models ->Add new item-> ADO .NET Entity Framework. Mai departe se selectează baza de date și conexiunea la baza de date. În figura 8 este prezentată diagrama importată în Visual Studio. [10]

În cazul aplicației noastre în Models în UtilitiesModel e stocată baza de date. În urma exportului clasele bazei de date sunt în folderul UtilitiesModel.tt iar contextul în UtilitiesModel.Context.tt.

Modelul poate fi creat prin două metode: Code First sau Database First.

Figure : Solution Explorer

Conceptul de Code First ne permite să transformăm clasele noastre într-o bază de date, ceea ce înseamnă că utilizatorul poate defini modelul folosing clasa POCO fară a mai fi nevoie să folosim xml- fișiere EDMX. Modelul creat devine modelul pe care Entity Framework se bazează.

Conceptul de Database First se bazează pe crearea bazei de date înaintea transpunerii ei în clase. Informațiile despre structura bazei de date, a modelului datelor precum și maparea lor sunt stocate în XML într-un fișier .edmx. Designer-ul Entity Fraework oferă o interfață fragică pentru afișarea și editarea fișierului .edmx.

În acest caz s-a folosit conceptul Database First. Framework Entity este un obiect de tipul object-relation mapper care preia structura bazei de date și o transformă în obiecte. Dezvoltatorii folosesc aceste obiecte pentru a interacționa cu baza de date în loc să acceseze direct baza de date.

Clasa principală care coordonează funcționalitatea Entity Framework pentru un model de date este clasa Context a bazei de date. Numele connection stringului(care a fost adăugat în Web.config ) este pasat constructorului clasei context.

public partial class utilitiesEntities : DbContext

{

public utilitiesEntities()

: base("name=utilitiesEntities")

{

}

}

DbContext este folosit de obicei cu un tip derivate care conține proprietatea DbSet< > pentru entitățile de bază ale modelului. Aceste seturi sunt inițializate automat când este creată instanța clasei derivate.

DbContext corespunde bazei de date (sau o colecție de tabele și vizualizări în baza de date) în timp ce DbSet corespunde unui tabel sau o vedere în baza de date. Pentru a avea acces la un obiect se va folosi un DbContext reprezentat printr-un DbSet prin care se poate obține acces, se poate șterge, adăuga și modifica datele din clasă.

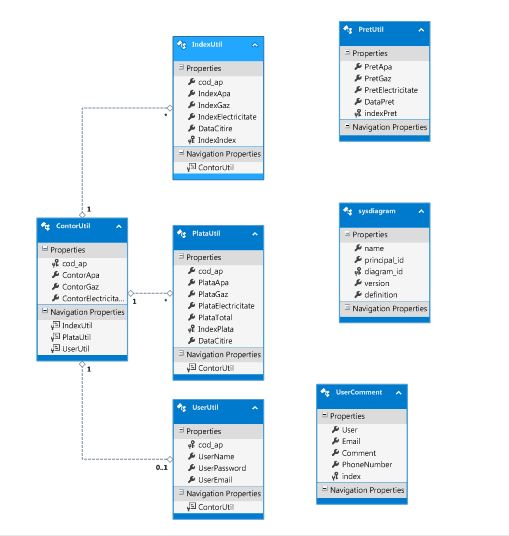


Figure : Diagrama Visual Studio pentru baza de date

Automat la importul bazei de date se crează câte o clasă specific fiecărui tabel. Mai jos este prezentată clasa generată pentru tabelul de ContorUtil. Relațiile cu celalate tabele sunt create cu ajutorul unei clase ICollection cu metodele get/set .

public partial class ContorUtil

{

public ContorUtil()

{

this.IndexUtil = new HashSet<IndexUtil>();

this.PlataUtil = new HashSet<PlataUtil>();

}

public int cod\_ap { get; set; }

public int ContorApa { get; set; }

public int ContorGaz { get; set; }

public int ContorElectricitate { get; set; }

public virtual ICollection<IndexUtil> IndexUtil { get; set; }

public virtual ICollection<PlataUtil> PlataUtil { get; set; }

public virtual UserUtil UserUtil { get; set; }

}.

Validarea datelor (spre exemplu : În momentul înregistrării user-ul nu completează un câmp, în dreptul căsuței ce trebuie completată v-a apărea un mesaj de eroare) se poate face folosind libraria System.ComponentModel.DataAnnotation.[19]

Atributele setate pentru fiecare obiect al clasei definesc pattern-uri comune de validare. Atributele setate cauzează MVC-ului să valideze atât partea de validare a clientului cât și partea de validare a serverului fără ca user-ul să scrie o linie în plus de cod. Un exemplu de astfel de validare se poate observa în liniile de cod de mai jos. [20]

public partial class UserComment

{

[Required(ErrorMessage = "You need to fill in a name")]

[DisplayName("Name")]

public string User { get; set; }

[Required(ErrorMessage = "You need to fill in an email address")]

[RegularExpression(@".\*@.\*", ErrorMessage = "Must be a valid email address")]

[DisplayName("Email address")]

public string Email { get; set; }

[Required(ErrorMessage = "You need to fill in a comment")]

[DisplayName("Your comment")]

public string Comment { get; set; }

[Required(ErrorMessage = "You need to fill in a phone number")]

[DisplayName("Your PhoneNumber")]

public string PhoneNumber { get; set; }

public int index { get; set; }

}

Acțiunile și logica aplicației este realizată de controllere. Din motive de lizibilitate a codului fiecărei tabele îi este asociat un controller(UserUtilsController corespunde tabelei UserUtil ). După generarea realizată cu ajutorul ADO .NET în controllere au fost generate metode specifice oricărei tabele.

Metodele GET și POST sunt metode HTTP (Hypertext Transfer Protocol). Acest protocol a fost creat pentru a permite conexiunea între clienți și server. HTTP funcționează ca un protocol cerere-răspuns între un client și server. Un browser web poate fi clientul iar o aplicație pe un computer care găzduiește un site web poate fi un server.

* afișare tabelă, afișare detalii

public ActionResult Index()

{

var contorUtils = db.ContorUtils.Include(c => c.IndexUtil).Include(c => c.PlataUtil).Include(c => c.UserUtil);

return View(contorUtils.ToList());

}

* ștergere câmpuri (această metodă preia ca parametru un id care în cazul nostru este codul aparamentului pentru care se dorește ștergerea înregistrării)

// GET: ContorUtils/Delete/5

public ActionResult Delete(int? id)

{

if (id == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

ContorUtil contorUtil = db.ContorUtils.Find(id);

if (contorUtil == null)

{

return HttpNotFound();

}

return View(contorUtil);

}

// POST: ContorUtils/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public ActionResult DeleteConfirmed(int id)

{

ContorUtil contorUtil = db.ContorUtils.Find(id); //se caută dupa id

db.ContorUtils.Remove(contorUtil); //se șterge intrarea din baza de date

db.SaveChanges(); //se salvează modificarile în baza de date

return RedirectToAction("Index");

}

* editarea câmpurilor existente

// GET: ContorUtils/Edit/5

public ActionResult Edit(int? id)

{

if (id == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

ContorUtil contorUtil = db.ContorUtils.Find(id);

if (contorUtil == null)

{

return HttpNotFound();

}

ViewBag.cod\_ap = new SelectList(db.IndexUtils, "cod\_ap", "cod\_ap", contorUtil.cod\_ap);

ViewBag.cod\_ap = new SelectList(db.PlataUtils, "cod\_ap", "cod\_ap", contorUtil.cod\_ap);

ViewBag.cod\_ap = new SelectList(db.UserUtils, "cod\_ap", "UserName", contorUtil.cod\_ap);

return View(contorUtil);

}

/ POST: ContorUtils/Edit/5

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult Edit([Bind(Include = "cod\_ap,ContorApa,ContorGaz,ContorElectricitate")] ContorUtil contorUtil)

{

if (ModelState.IsValid)

{

db.Entry(contorUtil).State = EntityState.Modified;

db.SaveChanges();

return RedirectToAction("Index");

}

ViewBag.cod\_ap = new SelectList(db.IndexUtils, "cod\_ap", "cod\_ap", contorUtil.cod\_ap);

ViewBag.cod\_ap = new SelectList(db.PlataUtils, "cod\_ap", "cod\_ap", contorUtil.cod\_ap);

ViewBag.cod\_ap = new SelectList(db.UserUtils, "cod\_ap", "UserName", contorUtil.cod\_ap);

return View(contorUtil);

}

* creare inputuri noi conține două metode : una de Get care se apelează în momentul în care apare pe site o cerere de adăugare input iar metoda de POST se apelează în momentul în care user-ul trimite cererea de înregistrare.

// GET: ContorUtils/Create

public ActionResult Create()

{

ViewBag.cod\_ap = new SelectList(db.IndexUtils, "cod\_ap", "cod\_ap");

ViewBag.cod\_ap = new SelectList(db.PlataUtils, "cod\_ap", "cod\_ap");

ViewBag.cod\_ap = new SelectList(db.UserUtils, "cod\_ap", "UserName");

return View();

}

// POST: ContorUtils/Create

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult Create([Bind(Include = "cod\_ap,ContorApa,ContorGaz,ContorElectricitate")] ContorUtil contorUtil)

{

if (ModelState.IsValid)

{

db.ContorUtils.Add(contorUtil);

db.SaveChanges();

return RedirectToAction("Index");

}

ViewBag.cod\_ap = new SelectList(db.IndexUtils, "cod\_ap", "cod\_ap", contorUtil.cod\_ap);

ViewBag.cod\_ap = new SelectList(db.PlataUtils, "cod\_ap", "cod\_ap", contorUtil.cod\_ap);

ViewBag.cod\_ap = new SelectList(db.UserUtils, "cod\_ap", "UserName", contorUtil.cod\_ap);

return View(contorUtil);

}

Pe lângă metodele generale pentru crearea funcționalității aplicației a fost necesară crearea unor metode adiționale. Aceste metode sunt specifice fiecărei tabele.

Tabela de PlataUtil este completată automat pe baza tabelei de IndexUtil și PretUtil. Pe baza fiecărui cod\_ap se preiau indecși pentru luna curentă din IndexUtil și se calculează prețul într-o metodă. Pentru a citi și a adăuga automat inputuri în tabela de IndexUtil și PlataUtil a fost necesară crearea unei metode ce creează o conexiune la baza de date.

Datele se pot citi automat cu ajutorul unui SqlDataAdaptor executând un querry sql. Pe lângă citire se pot face operații de Select, Insert, Update, Delete toate accesibile în ADO .NET.

Pentru citirea automată a contoarelor, a fost implementată o aplicație Arduino ce transmite date automat către aplicația C#. Indexi vor fi stocați în baza de date și automat la o citire nouă de indexi se va updata tabela de IndexUtil dar și tabela PlataUtil.

În aplicație a fost adăugată crearea unei metode ce citește datele din Excel și le adaugă în tabela de ContorUtil (în cazul în care un contor nu mai transmite datele corect administratorul are puterea sa modifice indexul).

Pe site administratorul sistemului are o pagină specială de unde poate încărca Excel-ul cu datele. La apăsarea butonului de OK aplicația primește informația de la server și cheamă metoda de adaugare în tabel.

Metoda primește ca și parametru un fișier de tip HttpPostedFileBase și verifică dacă fișierul este gol sau nu. În continuare se verifică dacă fișierul este de tip .xls sau .xlsx (ambele tipuri de fișiere sunt acceptate). Fișierul primit ca și parametru este mapat în folderul Content în proiectul ASP .NET. Pentru a putea opera asupra tabelei ContorUtil se crează o conexiune OleDB la baza de date. Informația din excel este păstrată sub formă de tabele.Astfel fiecare linie și coloană din excel este copiată în tabelă. [21]

[HttpPost]

public ActionResult CitesteIndexi(HttpPostedFileBase file)

{

DataSet ds = new DataSet();

if (Request.Files["file"].ContentLength > 0)

{

string fileExtension = System.IO.Path.GetExtension(Request.Files["file"].FileName);

if (fileExtension == ".xls" || fileExtension == ".xlsx")

{

string fileLocation = Server.MapPath("~/Content/") + Request.Files["file"].FileName;

if (System.IO.File.Exists(fileLocation))

{

System.IO.File.Delete(fileLocation);

}

Request.Files["file"].SaveAs(fileLocation);

string excelConnectionString = string.Empty;

excelConnectionString = "Provider=Microsoft.ACE.OLEDB.12.0;Data Source=" +

fileLocation + ";Extended Properties=\"Excel 12.0;HDR=Yes;IMEX=2\"";

//connection String for xls file format.

if (fileExtension == ".xls")

{

excelConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" +

fileLocation + ";Extended Properties=\"Excel 8.0;HDR=Yes;IMEX=2\"";

}

//connection String for xlsx file format.

else if (fileExtension == ".xlsx")

{

excelConnectionString = "Provider=Microsoft.ACE.OLEDB.12.0;Data Source=" +

fileLocation + ";Extended Properties=\"Excel 12.0;HDR=Yes;IMEX=2\"";

}

//Create Connection to Excel work book and add oledb namespace

OleDbConnection excelConnection = new OleDbConnection(excelConnectionString);

excelConnection.Open();

DataTable dt = new DataTable();

dt = excelConnection.GetOleDbSchemaTable(OleDbSchemaGuid.Tables, null);

if (dt == null)

{

return null;

}

String[] excelSheets = new String[dt.Rows.Count];

int t = 0;

//excel data saves in temp file here.

foreach (DataRow row in dt.Rows)

{

excelSheets[t] = row["TABLE\_NAME"].ToString();

t++;

}

OleDbConnection excelConnection1 = new OleDbConnection(excelConnectionString);

string query = string.Format("Select \* from [{0}]", excelSheets[0]);

using (OleDbDataAdapter dataAdapter = new OleDbDataAdapter(query, excelConnection1))

{

dataAdapter.Fill(ds);

}

excelConnection.Close();

}

if (fileExtension.ToString().ToLower().Equals(".xml"))

{

string fileLocation = Server.MapPath("~/Content/") + Request.Files["FileUpload"].FileName;

if (System.IO.File.Exists(fileLocation))

{

System.IO.File.Delete(fileLocation);

}

Request.Files["FileUpload"].SaveAs(fileLocation);

XmlTextReader xmlreader = new XmlTextReader(fileLocation);

// DataSet ds = new DataSet();

ds.ReadXml(xmlreader);

xmlreader.Close();

}

for (int i = 0; i < ds.Tables[0].Rows.Count; i++)

{

string conn = ConfigurationManager.ConnectionStrings["utilitiesDBConnectionString"].ConnectionString;

SqlConnection con = new SqlConnection(conn);

string query = "Insert into IndexUtil(cod\_ap,IndexApa,IndexGaz,IndexElectricitate,DataCitire) Values('" +

ds.Tables[0].Rows[i][0].ToString() + "','" + ds.Tables[0].Rows[i][1].ToString() +

"','" + ds.Tables[0].Rows[i][2].ToString() + "','" + ds.Tables[0].Rows[i][3].ToString() +

"','" + ds.Tables[0].Rows[i][4].ToString() + "')";

con.Open();

SqlCommand cmd = new SqlCommand(query, con);

cmd.ExecuteNonQuery();

con.Close();

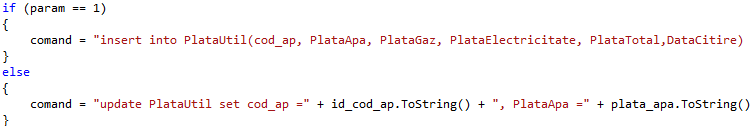
}

return RedirectToAction("AdminLogin", "UserUtils");

}

return View();

}

Pentru calculului totalului de plată al unui apartament pe o lună am creat o metodă ce deschide o conexiune cu baza de date, preia informațiile din tabel într-un SQLDataAdapter, apoi pe rând pentru fiecare entitate din tabel execută operațiile aritmetice necesare(doar dacă luna este cea curentă) iar la final inserează în baza de date și închide conexiunea cu baza de date. 

Clasele DataAdapter generează obiecte ce reprezintă o interfață între baza de date și obiectele DataSet interne aplicației. Ele gestionează conexiunea astfel încât conexiunea să se realizeze doar dacă este necesar. Un obiect DataSet este de fapt un set de tabele relaționale. Datele stocate de un DataSet sunt în format XML.

Administratorul sistemului poate introduce date noi în tabela de IndexUtil din panoul de control al administratorului de pe site. Astfel a fost necesară crearea unor metode noi (asemănătoare cu cea precendentă) pentru calculul totalului de plătit pentru fiecare lună în parte nu doar pentru cea curentă, o metodă pentru ștergerea tuturor înregistrărilor din tabela PlataUtil și o metodă pentru ștergerea unei singure plăți din tabelă. În exemplul de mai jos este prezentat un query sql pentru ștergerea unei singure plăți.

comand = "delete from PlataUtil where cod\_ap = " + id\_cod\_ap.ToString()+ " and DataCitire = " + datacitire.ToString();

da2.InsertCommand = new SqlCommand(comand, con);

da2.InsertCommand.ExecuteNonQuery();

În tabela UserUtil a fost necesară crearea metodelor pentru LogIn-ul și LogOff-ul useri-lor pe site. După cum am precizat înainte în această lucrare pe site se pot loga două tipuri de utilizatori : useri normali și administratorul blocului. Userii normali au fiecare un cod\_ap, iar în aplicație pentru a putea face distincția între cele două tipuri de utilizatori pentru administrator am utilizat cod\_ap = 11111. Am atribuit un singur cod unic pentru un singur administrator deoarece pentru un bloc nu este nevoie de mai mulți administratori.

[HttpPost]

public ActionResult LogIn(Models.Entity.UserUtil utilizator)

{

if (ModelState.IsValid) //se verifică userul si parola

{

var v = db.UserUtils.Where(a => a.UserEmail.Equals(utilizator.UserEmail) && a.UserPassword.Equals(utilizator.UserPassword)).FirstOrDefault();

if (v != null)

{

Session["LoggedUsername"] = v.UserEmail.ToString();

Session["cod\_ap"] = v.cod\_ap.ToString();

//Session["LoggedUserPassword"] = v.UserPassword.ToString();

if (Session["cod\_ap"].ToString() == "11111") //se verifică ce tip de user este

{

return RedirectToAction("AdminLogin");//redirectarea la pagină

}

else

{

return RedirectToAction("AfterLogin");

}

}

}

return View();

}

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult LogOff()

{

Session["LoggedUsername"] = null; //se sterg datele sesiunii

return RedirectToAction("Index", "Home"); //se redirectează spre pagina de start

}

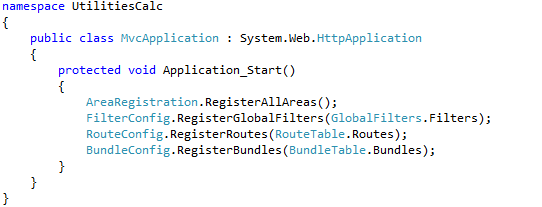
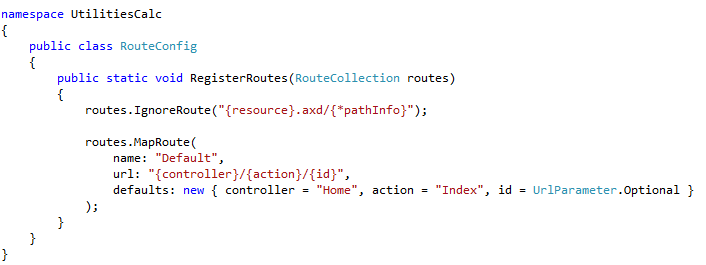
 Routarea ASP .NET permite utilizarea de URL-uri cu un anumit patern pentru a mapa solicitările fișierelor, dar care nu include neapărat numele acestor fișiere în URL. În plus se pot include substituenți într-un model de URL, astfel încât datele variabile pot fi transmise fără a fi necesita un șir de interogare. [22]

Figure : Global.asax

O rută în MVC este un URL care este mapat la o clasă dintr-un controller din aplicație. Pentru a defini o rută se crează o instanță a clasei de rutare definind pattern-ul URL, conducătorul și opțional un nume pentru rută.

În general rutele pentru aplicațiile MVC include {controller}, {action} și {id}. Când o cerere este primită, aceasta este dirijată spre obiectul URLRoutingModule și apoi spre MVCHandler. MVCHandler determină ce controller conține acțiunea respectivă și adaugă sufixul controller. Acțiunea determină ce metodă trebuie chemată din controller.

Spațiul de nume de clasă .NET Framework este o bibliotecă de tipuri de clase și interfețe incluse în SDK-ul .NET Framework. Această bibliotecă oferă acces la funcționalitatea sistemului și este proiectată pentru a fi temelia pe care sunt construite aplicațiile .NET.

Figure : Configurarea rutei

Spațiul de nume System conține clase fundamentale și clase de bază care definesc valori frecvent utilizate, evenimente, atribute și procesări de excepții. Alte clase suportă conversii de tip, manipularea parametrilor metodei, invocarea de programe de la distanță și locale, supravegherea de aplicații gestionate și neadministrate.

Spațiul de nume System.Collections.Generic conține interfețe și clase care definesc colecții generice, care permit utilizatorului să creeze alte colecții specifice aplicației care asigură siguranță și performanță.

Spațiul de nume System.Data oferă acces la clase care reprezintă arhitectura ADO .NET care permite crearea componentelor ce gestioneză eficient date din surse de date multiple. Într-un scenariu deconectat, cum ar fi internetul, ADO .NET oferă instrumentele necesare pentru solicitarea, actualizarea și reconcilierea datelor din multiple sisteme de nivel.

System.Linq este o bibliotecă care oferă clase și interfețe ce susțin interogări sql(Integrated Query - LINQ). System.Linq este prezentă în ansamblul System.Core. Printre clasele cele mai utilizate amintim: clasa Enumerable(conține operatori standard LINQ care operează pe obiecte ce implementează IEnumerable) și clasa Queryable.

System.Net oferă o interfață de programare simplă pentru multe dintre protocoalele utilizate în rețetelele de azi. Clasele WebRequest și WebResponse formează baza a ceea ce se numesc protocoale conectabile, o punere în aplicare a serviciilor de rețea care permit dezvoltatorilor să creeze aplicații care utilizează resurse de pe internet fără griji cu privire la detaliile specifice ale protocoalelor individuale.

System.Web conține tipuri care permit comunicarea browser/server. Derivate ale acestei librării include tipuri care suportă autentificarea pentru Web Forms, servicii pentru aplicații, caching de date pe server, configurarea aplicației ASP .NET, date dinamice .handlere HTTP, serializare JSON, funcționalități AJAX.

Spațiul de nume System.Web.Mvc conține clase și interfețe care susțin cadrul ASP.NET Model View Controller pentru creerea de aplicații web. Acestă bibliotecă conține clase ce reprezintă controller-ele, rezultate ale acțiunilor, view-uri, view-uri parțiale, lianți model etc.

Spațiul de nume System.Web.Security conține clase care sunt utilizate pentru implementarea securității aplicațiilor ASP .NET, în aplicațiile de tip Web server. Printre cele mai cunoscute clase ale acestei biblioteci amintim clasa membership, utilizată pentru a valida acreditările de utilizator și pentru a gestiona setările utilizatorului(parole, adrese de e-mail), clasa Roles care permite gestionarea autorizației pentru grupuri de useri ce au anumite roluri în aplicație.

Cadrul ASP .NET oferă containere la nivel de pagină care pot trece date între controlere și paginile de afișare. Pentru a afișa pagina de view se apelează metoda din controller iar pentru a transmite date paginii de afișare se folosesc proprietatea ViewData din clasa ViewPage. Această proprietate returnează un obiect ViewDataDictionary.

Pe lângă ViewData se pot trimite date spre view și cu ViewBag, PartialView, TempData, ViewModel, Tuple.

Proprietatea ViewBag permite partajarea dinamică a valorilor de la operator spre vizualizare. Este un obiect dinamic ce nu are proprietați predefinite, se gasește în clasa ControllerBase și nu are nevoie de typecasting pentru datatype.

PartialView este utilizat în cazul în care codul (Razor si HTML) necesită o partajare(același cod pentru mai multe view-uri).

TempData este un dicționar de obiecte de tipul TempDataDictionary. Valorile stocate în TempData necesită typecasting la datatype în View. Este similar cu ViewData dar diferența este că TemData permite trimiterea și primirea datelor de la un controller la altul și de la o acțiune a unui controller la altă acțiune.

ViewModel este un model ce permite ca mai multe modele sa fie tratate ca o singură clasă. Acesta nu are mtode proprii ci ar trebuii sa fie o colecție de proprietați necesare unui View.

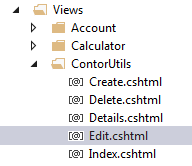
Logica aplicației este în controller însă utilizatorul nu poate vedea nimic în pagina web. Pentru asta este nevoie de creerea unor fișiere HTML care prezintă informația în browser. În solution explorer aceste fișiere pot fi găsite în folderul de Views. În aplicația prezentată în acestă lucrare fiecărei tabele îi sunt asociate mai multe fișiere de vizualizare grupate într-un folder cu același nume ca și tabela.

Figure : Views

În general unei metode din controller îi corespunde un fișier cshtml. Așadar după generarea modelului au fost create fișiere cshtml pentru fiecare metodă din controller(afișare, stergere, editare, creare).

În fiecare view este inclusă declarația modelului în partea de sus a fișierului pentru a putea specifica tipul de obiect pe care view-ul îl așteaptă. De exemplu pentru tabela ContorUtil a fost adăugat următorul include :

@model UtilitiesCalc.Models.Entity.ContorUtil

În continuare fiecărui view îi este precizat numele cu care v-a fi chemat în browser dar și layout-ul. Aplicația prezentată în această lucrare include un singur stil pentru toate fișierele din proiect.

@{

ViewBag.Title = "Edit";

Layout = "~/Views/Shared/\_Layout.cshtml";

}

Un exemplu de View este prezentat în figura 12. Controller-ul returnează spre view o listă cu elementele tabelei ContorUtil.

return View(contorUtils.ToList());

Un formular este alcătuit din câmpuri de editare,butoane radio,butoane de comandă etc. Formularele permit construirea unor pagini Web ce permit user-ilor să introducă informații și să le trimită serverului. Informația de la user trece prin două etape până să ajungă la server. [23]

În prima fază utilizatorul completează formularul și apasă butonul de expediere. Faza a doua este reprezentată de aplicația dedicată de pe server care analizează informațiile primate și raspunde în funcție de cerință.

Exemplul de mai jos prezintă structura unui view(view pentru creerea unui nou contor). La început se creează un formular ce afisează cîmpuri de completat pentru user.

@using (Html.BeginForm())

{

@Html.AntiForgeryToken()

<div class="form-horizontal">

<h4>ContorUtil</h4>

<hr />

@Html.ValidationSummary(true, "", new { @class = "text-danger" })

<div class="form-group">

@Html.LabelFor(model => model.cod\_ap, htmlAttributes: new { @class = "control-label col-md-2" })

<div class="col-md-10">

@Html.EditorFor(model => model.cod\_ap, new { htmlAttributes = new { @class = "form-control" } })

@Html.ValidationMessageFor(model => model.cod\_ap, "", new { @class = "text-danger" })

</div>

</div>

<div class="form-group">

@Html.LabelFor(model => model.ContorApa, htmlAttributes: new { @class = "control-label col-md-2" })

<div class="col-md-10">

@Html.EditorFor(model => model.ContorApa, new { htmlAttributes = new { @class = "form-control" } })

@Html.ValidationMessageFor(model => model.ContorApa, "", new { @class = "text-danger" })

</div>

</div>

<div class="form-group">

<div class="col-md-offset-2 col-md-10">

<input type="submit" value="Create" class="btn btn-default" />

</div>

</div>

</div>

}

Se remarcă stilul inline prezent în acest view: <div class="form-group">, <div class="col-md-10"> .

Eticheta <div> funcționează asemănător cu eticheta <p>, marcând un întreg bloc de conținut, dar fără a genera linii albe între paragrafe.

În figura următoare este prezentat rezultatul codului prezentat anterior.

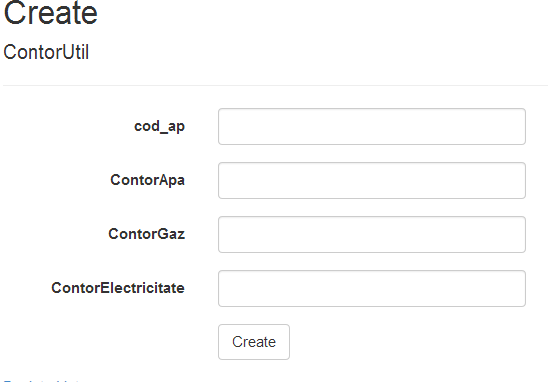


Figure : View creare contor

Pentru partea de statistică a aplicației s-a folosit Canvas JS pentru crearea dinamica de chart-uri și scripturi de Ajax, Javascript și timere (în controller) pentru a putea rula și reactualiza aceste charturi. Mai jos este prezentat modul în care sunt primite date de la tabela IndexUtil.

<script type='text/javascript'>

//setTimeout("location.reload();",10000);

var index\_apa\_citit;

var index\_gaz\_citit;

var index\_electricitate\_citit;

function ReadIndexApa() {

$.ajax({

url: '@Url.Action("get\_indexapa")',

type: 'GET',

dataType: 'json',

success: function (data) {

//do something with data

index\_apa\_citit = JSON.parse(data);

},

error: function (error) {

//log or alert the error

alert("eroare citire index apa");

}

});

}

## Aplicația Arduino

Pentru această aplicație nu s-au folosit contoare reale ci s-a încercat simularea lor cu ajutorul unei placuțe Arduino Uno. Astfel a fost necesară crearea unei aplicații în Arduino Genuino conectat la calculator prin intefață serială ce transmite date către baza de date folosită de aplicația principală. Pentru fiecare apartament din bloc au fost simulate trei contoare (contor de apă, gaz și curent). Datele de la contoarele simulate sunt transmise la fiecare minut (60 sec) și stocate în automat în baza de date în tabela IndexUtil.

Pentru a putea actualize datele primite de la Arduino a fost creat un timer ce va actualize datele în fiecare minut.

public ActionResult Index()

{

System.Timers.Timer timer = new System.Timers.Timer(6000);

timer.Enabled = true;

timer.Elapsed += new ElapsedEventHandler(timer\_elapsed);

return View();

}

private void timer\_elapsed(object sender, ElapsedEventArgs e)

{

SetComPort();

//throw new NotImplementedException();

}

Pentru conexiunea aplicației web cu hardware-ul a fost necesară crearea unei funcții SetComPort().

private void SetComPort()

{

string[] ports = SerialPort.GetPortNames();

foreach (string port in ports)

{

currentPort = new SerialPort(port, 9600);

updateDBfromArduino();

CalcPlataLunaCurenta1(0);

//DetectArduino();

}

După stabilirea conexiunii se citesc datele trasmise de Arduino și se actualizează tabela IndexUtil. Pentru a putea citi datele de la Arduino la fiecare 60 secunde a fost utilizat un timer și s-au folosit librariile:

using System.Timers;

using System.Threading;

using System.IO.Ports;

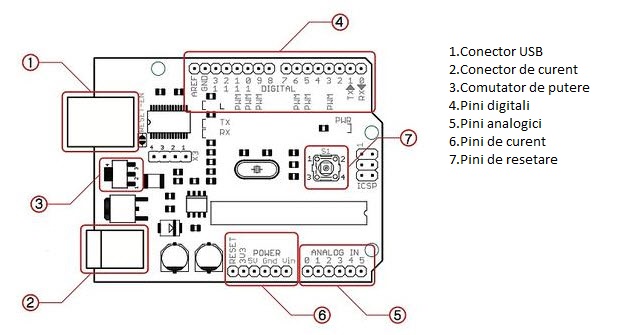
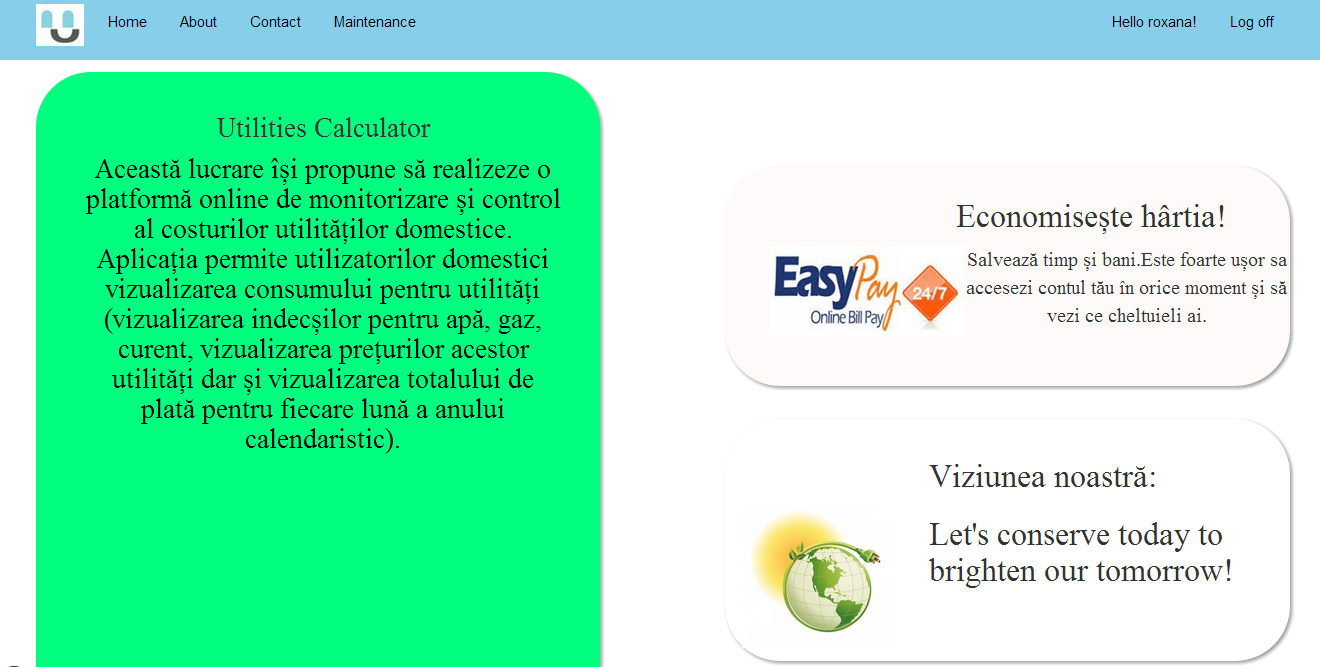


Figure : Schema placuței Arduino

# Manual de utilizare al aplicației

Platforma online de monitorizare și control a costurilor unitaților domestice este o aplicație de tip pagină web ce permite accesul a două grupuri de useri: consumatorii normali și administratorul blocului.

Interfața aplicației este una comună majoritații paginilor web constând din pagina de Home, About, Contact, Maintenance, Register și pagina de LogIn.

 Pagina de Home conține o scurtă prezentare a platformei și sloganul platformei.

Pagina de contact conține informații despre creatorul platformei. Pagina de About conține o prezentare mai amănunțită a platformei (este explicat conceptual AMR).

Figure : pagina Home

În momentul în care un utilizator nou (deasemenea locatar al blocului) intră pe site acesta trebuie să se înregistreze. El se poate loga pe site doar dacă s-a înregistrat în prealabil. La înregistrare acesta este nevoit să completeze câmpurile de User Name, User Password, User E-mail și codul apartamentului. Dacă aceste câmpuri nu sunt completate v-a apărea un mesaj de eroare.

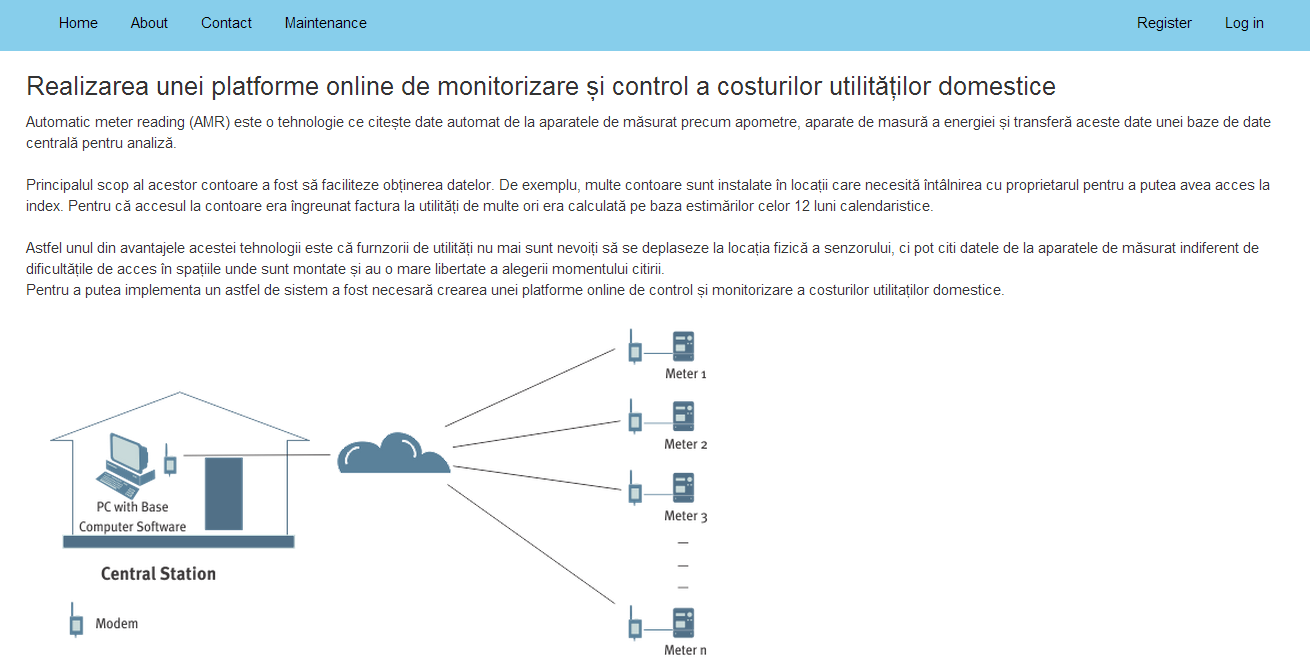
 În urma unei înregistrari reușite se v-a afișa un mesaj de înregistrare cu succes. După caz la conectarea pe site vor apărea panourile de control(pentru user normal sau pentru administrator).

Figure : pagina About

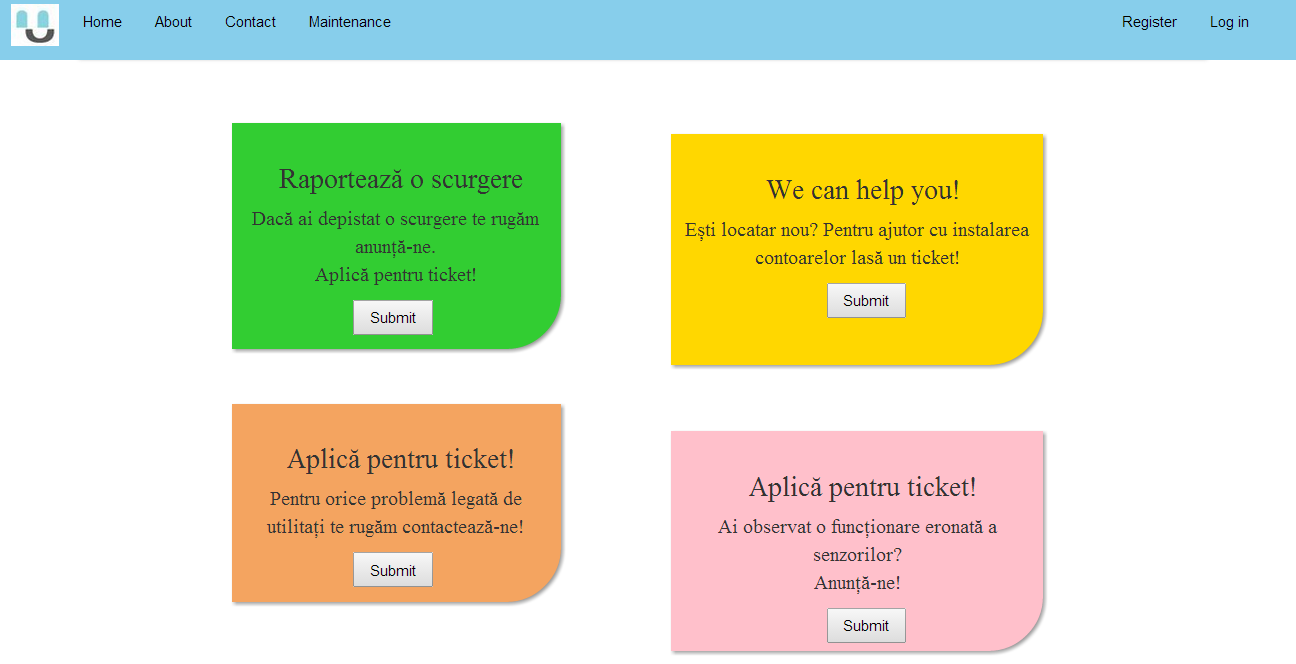
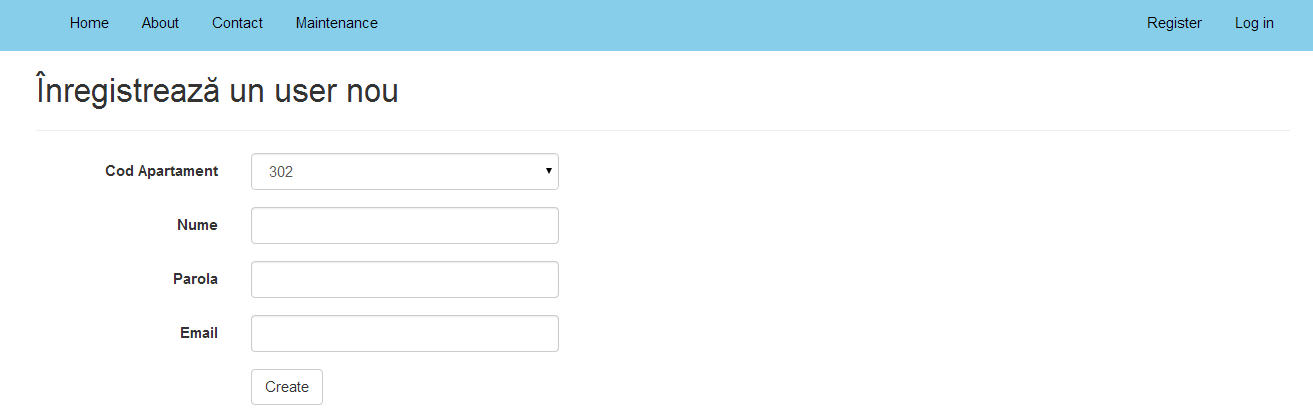


Figure : pagina Maintenance

Pagina de Maintenance este o pagină dedicată locatarilor blocului pentru anumite sugestii și reclamații. Lăsarea unui ticket este foarte simplă, user-ul trebuie să identifice problema și să completeze formularul ce apare la apăsarea butonului de Submit. Aceste reclamații și sugestii vor putea fi vizualizate de către administrator din panoul său de comandă. De asemenea pe măsura rezolvării lor aceste mesaje pot fi șterse de către administrator.



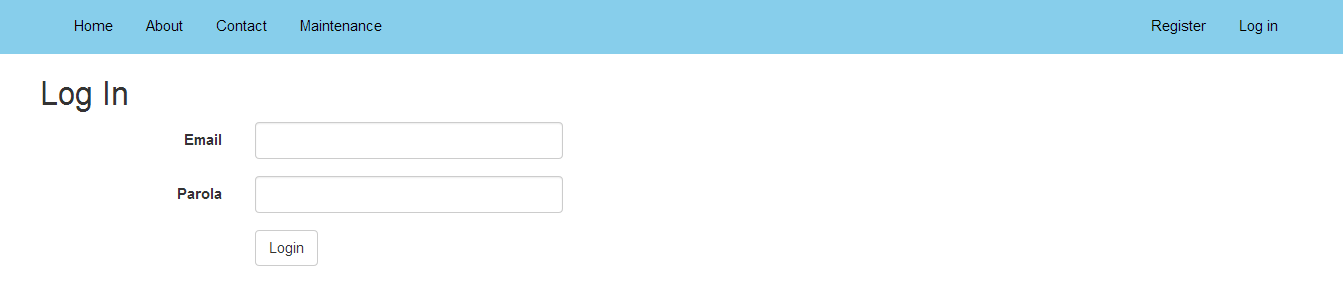
Figure : Înregistrarea unui user

Figure : LogIn pentru utilizatori

## Panou de comandă al utilizatorului

La conectarea pe site un user normal poate executa următoarele acțiuni: editarea contului curent, afișarea totalului de plată pentru luna respectivă, afișarea istoricului, afișarea prețurilor la utilitați.

 Pentru a proba funcționalitatea aplicației am folosit un user de test: Roxana. Pentru acest user am afișat totalul de plată pentru luna curentă, istoricul și afișarea prețurilor.

Figure : Panoul de comandă al user-ului



Figure : Plata pentru luna curentă pentru user

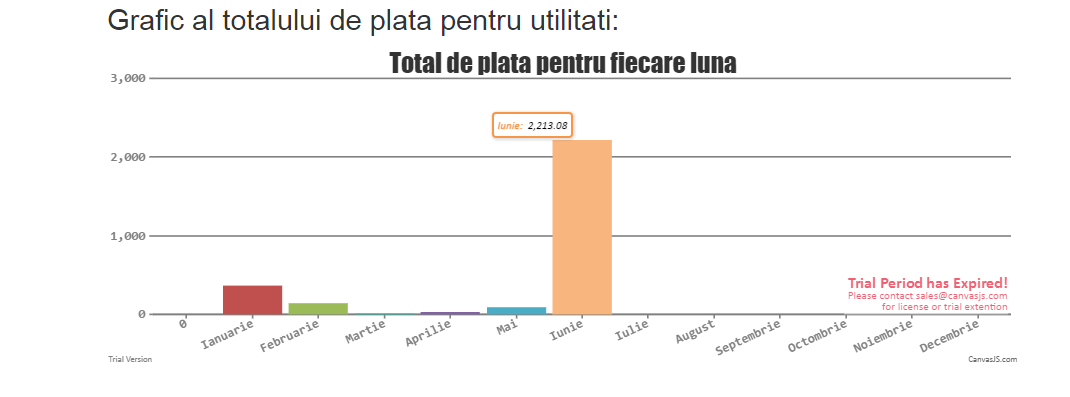


Figure : Istoric plați

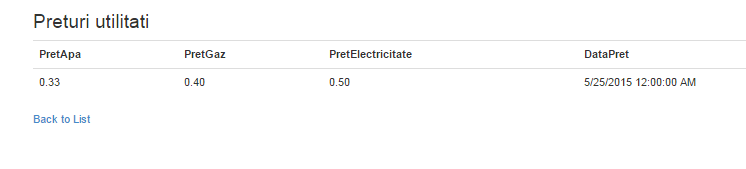


Figure : Tabela de prețuri pentru utilitați

Datorită interfațării cu Arduino, pentru fiecare utilizator se poate vedea consumul în timp real.

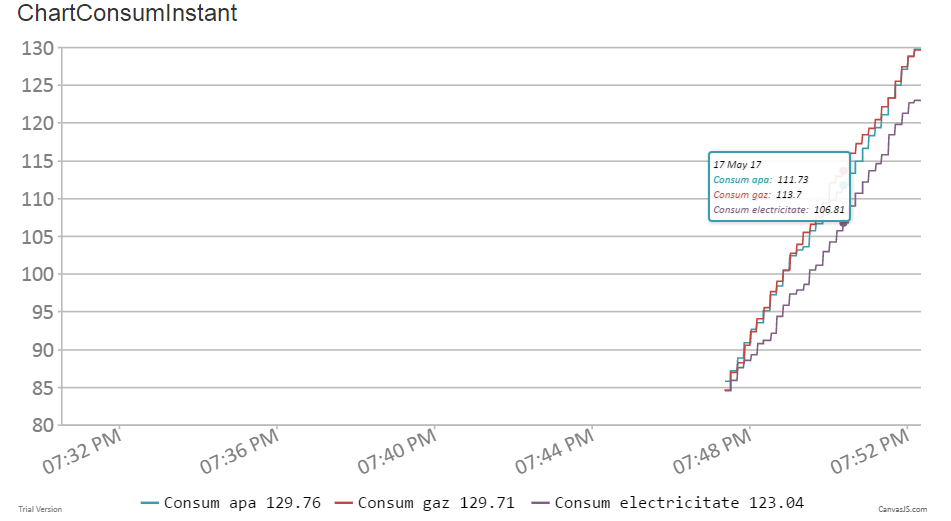


Figure : Consum instant

## Panou de comandă al administratorului

La conectarea administratorului pe site în panoul de comandă acesta poate efectua următoarele acțiuni: modificarea prețurilor la utilitați, editarea tabelei de useri, modificarea unui index în cazul unui contor defect, vizualizarea sugestiilor și reclamațiilor, citirea datelor de la contoare, calculul plații pentru luna curentă și totalul de plată pentru toate lunile în cazul în care se necesită o recalculare.



Figure : Panoul de comandă al administratorului

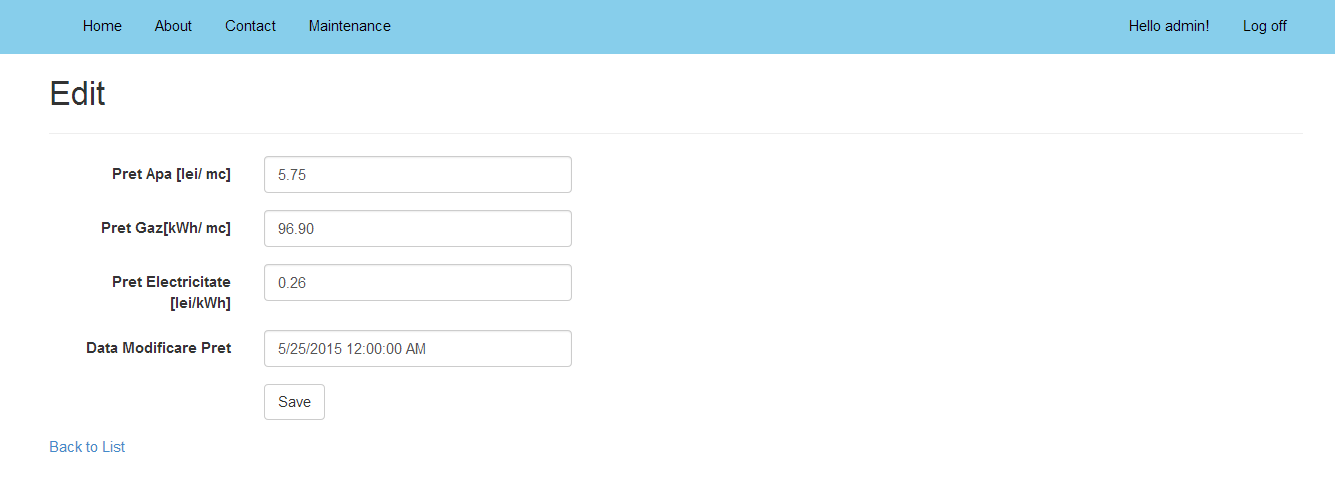


Figure : Modificarea prețurilor

Citirea indexilor pentru utilitați se face de catre admin. Fișierul cu datele de la contoare va fi încărcat pe site iar la apăsarea butonului de OK datele vor fi scrise în baza de date, în tabela de IndexUtil.



Figure : Citirea datelor de la contoare

După calcularea prețurilor indiferent dacă doar pentru o lună sau pentru toate lunile până la data curentă administratorului îi este afișată lista cu totalul de plată pentru fiecare user pentru toate lunile.

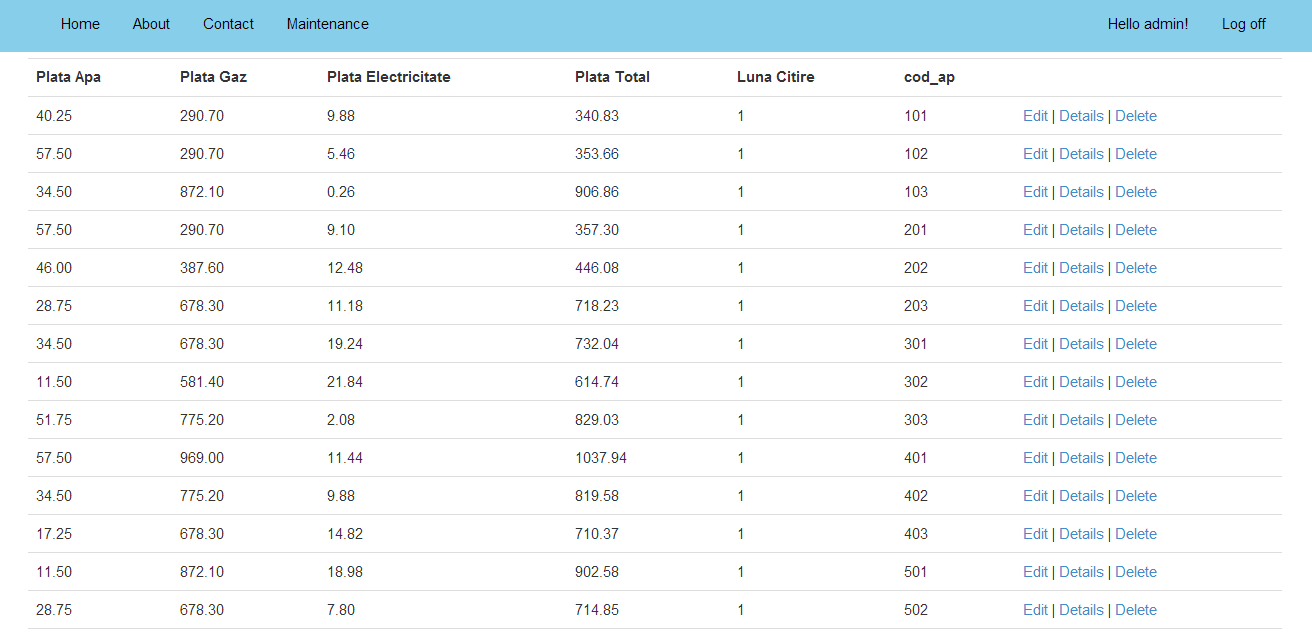


Figure : După calcularea prețului se afișează totalul de plată

# Concluzii

Tema propusă: Platforma online de control și monitorizare a costurilor utilitaților domestice a fost implementată pentru două nivele de utilizatori: utilizatori normali și administratorul sistemului. Aplicația a fost creată după presupunerea că modelul de comunicare AMR este bazat pe WiFi.

Scopul proiectului este facilitarea accesului utilizatorilor la controlul utilitaților domestice. Pentru îndeplinirea acestui obiectiv s-a decis realizarea unei aplicații web care poate fi accesată de către utilizatorii unui bloc. Acest program vine și în ajutorul administratorului de bloc, deoarece cheltuielile lunare sunt calculate automat de către aplicație, astfel administratorul are la dispoziție toate datele necesare pentru închiderea de lună.

Administatorul este singura persoană avizată pentru schimbarea unor informații implicate în procesul de calcul al utilitaților(adăugare/ștergere utilizatori, modificare tarife, modificare index contoare în cazul unor date eronate). Utilizatorii au dreptul de a se conecta pe site astfel având posibilitatea să își verifice situația cheltuielilor on-line fără a fi necesar să se deplaseze la avizierul blocului. Aplicația oferă posibilitatea utilizatorilor la efectuarea unor reclamații(scurgeri / întreruperi curent / întreruperi gaz). În cazul efectuarii unei reclamații administratorul este informat, solicitând societații responsabile sa remedieze situația.

Avantajele aduse de o astfel de aplicație:

* reduce costurile pentru citirea manuală a contoarelor
* monitorizează întregul sistem
* managementul și conservarea energiei
* platforma poate fi accesată în orice moment
* elimină estimările facturilor
* elimină previziunile legate de consum
* optimizarea tarifelor
* salvează bani

Una din direcțiile de dezvoltare presupune valorificarea aplicației prin implementarea unui sistem AMR bazat pe conexiune de WiFi. Structura de comunicație a senzorilor presupune patru nivele de component hardware. Contoarele de apă/gaz/curent din fiecare apartament formează primul nivel. Datele de la aceste contoare vor ajunge la un contor inteligent. La fiecare etaj, pentru cele trei apartamente v-a exista câte un astfel de contor inteligent. Pentru un bloc este necesar un concentrator ce transmite datele mai departe spre centrul de comandă.

Datorită progresului tehnologic aplicațiile software sunt în permanentă schimbare. Pentru a putea face față acestor mari schimbari un sistem trebuie să fie mereu în concordanță cu cerințele tehnologice dar și cu cerințele clienților.

Pentru ca sistemul existent să poată fi competitiv cu celălalte aplicații implementate deja de către alte companii, acesta necesită modificări la nivelul software-ului pentru problema securitații sistemului și adăugarea de funcționalitați.

Din prisma funcționalitații sistemul poate fi extins prin implementarea unei funcționalitați ce asigură o mai bună comunicare cu clienții sistemului. Deoarece sistemul asigură gestiunea și controlul utilitaților domestice, acesta se poate extinde prin implementarea unui modul de plată online. Un astfel de modul este foarte important deoarece oferă clienților posibilitatea economisirii timpului dar în același timp oferă comoditate și simplitate.

Astfel pentru a ușura și munca administratorului este necesară creerea unui modul software pentru determinarea neplații la timp. Utilizatorii care uită să plătească în prima lună vor primi un mesaj de avertizare în primă fază iar la neplata pentru trei luni aceștia vor fi deconectați de la sistem.

Un pas foarte important odată cu crearea unui modul de plată online este asigurarea securitatații aplicației. Sistemul trebuie să protejeze datele clienților, să asigure efectuarea plații în siguranță, etc. De asemenea un aspect critic ce trebuie rezolvat în dezvoltarea ulterioară a sistemului este creșterea securitații la logarea clienților în aplicație.

# Referințe bibliografice

1. Steve Burbeck, "Applications Programming in Smalltalk-80: How to use Model-View-Controller", 1992.
2. Tom D. Tamarkin, “Automatic Meter Reading”, September-Octomber, 1992.
3. Tarek Khalifa, Kshirasagar Naik, Amiya Nayak ”A Survey of Communication Protocols for Automatic Meter Reading Applications”, 2011
4. M. Choi, S. Ju, and Y. Lim, “Design of integrated meter reading system based on power line communication,” in *IEEE International Symp.Power Line Commun. Its Appl.*, Apr. 2008.
5. S. K. Kim, “Automatic meter reading system and method using telephone line,” in *United States Patent 7102533*, Sept. 2006.
6. T. Chandler, “The technology development of automatic metering and monitoring systems,” in *IEEE International Power Eng. Conf.*, Dec.2005.
7. <http://www.securetogether.com/en/solutions/automatic-meter-reading>
8. Li LI, Xiaoguang HU, “The Architecture of WiFi-Based WSN for AMR System and E-HWMP Routing Protocol”, 2014
9. Z. Qiang, “Design and Implementation of WIFI Wireless Sensor Networks”, North University of China, 2012.
10. Ana Întuneric, Nicolae Olãroiu, Cristina Sichim “Programarea Web cu Microsoft .NET”, 2007
11. Adrian Niţã, Maria Niţã, Daniela Tarasã, “Programarea Orientatã pe Obiecte şi Programarea Vizualã”, 2007
12. Ioan Asiminoaei “ASP.NET MVC”, 2014
13. <http://www.codeproject.com/Articles/661878/Implementation-of-MVC-Patterns-in-ASP-NET-Web-form>
14. <http://www.w3schools.com/html/html_intro.asp>
15. <http://www.w3schools.com/css/css_intro.asp>
16. http://www.w3.org/TR/html401/present/styles.html
17. - <https://msdn.microsoft.com/en-us/library/e80y5yhx(v=vs.110).aspx>
18. <http://www.codeproject.com/Tips/869553/CRUD-operation-with-Entity-Framework-Database-Fi>
19. https://msdn.microsoft.com/en-us/library/ee256141(v=vs.100).aspx
20. <http://www.elevenwinds.com/data-validation-in-asp-net-mvc-database-first>
21. <http://www.c-sharpcorner.com/UploadFile/ae35ca/read-and-write-excel-data-using-C-Sharp/>
22. <http://www.asp.net/mvc/overview/older-versions-1/controllers-and-routing/asp-net-mvc-routing-overview-cs>
23. I.Stojmenovic, ”Handbook of Sensor Networks: Algorithms and Arhitectures”, 2005
24. Shamneesh Sharma,Dinesh Kumar,Keshav Kishore, ”Wireless Sensor Networks- A Review on topologies and Node Arhitecture”, 2013
25. Jayanta Kumar Pandey, R.N. Das choudhary,”Embedded Automobile Engine locking System, using GSM technology”, ITER, SOA University Odisha, India.