Cuprins

1. Introducere

Aplicația își propune să ofere un sistem modern de colectare și gestionare a datelor în cadrul unei grădinițe, utilizând arhitectura de tip MVC Model-View-Controller, sub forma unei aplicații Web ASP.NET. Aplicația va salva informațiile într-o bază de date MySQL, creată în interiorul aplicației și va apela această bază de date pentru generarea statisticilor și a rapoartelor. Pentru o mai bună accesibilitate, informațiile vor fi introduse prin intermediul unei interfețe, în funcție de rolul stabilit pentru fiecare utilizator iar pentru gestionarea datelor se vor implementa funcțiile CRUD - Create, Read, Update, Delete.

Pentru atingerea obiectivului, aplicația trebuie să îndeplinească următoarele cerințe:

- Înregistrarea și gestionarea datelor legate de copii, educatori și grupele grădiniței.
- Alocarea copiilor în grupe și gestionarea fișei zilnice de prezență.
- Gestionarea taxelor aplicate și a plăților înregistrate.
- Asigurarea accesului în aplicație, pe baza rolurilor, atât a părinților cât și a cadrelor didactice.
- Generarea rapoartelor.

1.1 Posibilități de rezolvare a problemei

Datorită multiplelor metode de implementare a acestei soluții dar și a caracterului general al aplicației, am identificat în urma unei căutări, câteva alternative pentru realizarea aplicației:

- Aplicație Desktop ”Stand-Alone”, realizată în C# și conexiune la o bază de date locală de tip SQLite http://www.softgradinita.ro/index.html , http://kido.bizage.ro/
- Aplicație Web realizată în PHP și conexiune la baza de date http://gesoft.ro/evcon/index.php?r=user/login
- http://www.autosoft.ro/gestoc.html

Soluțiile enumerate mai sus nu includ detalii tehnice despre tehnologia folosită sau despre tipul aplicației ci o prezentare a unor funcții și argumente pentru achiziționarea soft-ului respectiv. Accentul este pus pe personalizarea pachetelor și a funcțiilor care se doresc a fi implementate dar lipsesc detalii legate de modul în care este realizată soluția.

1.2 Justificarea metodei alese

Am optat pentru realizarea aplicației de tip ASP.NET MVC deoarece oferă ”scheletul” funcțional pe care se poate construi rapid restul aplicației și multă flexibilitate în dezvoltarea și îmbunătățirea ei. Interfața se poate schimba sau poate implementa diferite module care sunt adăugate doar prin enumerarea referințelor. Având în vedere scopul aplicației, consider că limbajul de programare C# oferă abordarea cea mai bună pentru operațiunile cu clase și obiecte. C# îmi oferă posibilitatea de a scrie cod complex în partea de back-end, menținând o separare a conceptelor și facilitând un sistem de lucru modular. Pe lângă limbajul de programare, un instrument care a ajutat la alegerea

acestei soluții este Visual Studio, un mediu de programare impresionant și o bază de cunoștințe imensă. Practic, nu există multe întrebări legate de C# care să nu aibă cel puțin câteva abordări diferite.

2. Prezentarea detaliată a cerințelor aplicației

Pentru atingerea obiectivului, aplicația trebuie să îndeplinească următoarele cerințe:

1. Să permită înregistrarea și gestionarea datelor de bază legate de copii, educatori și grupele grădiniței.
2. Să permită înregistrarea copiilor în grupe și gestionarea fișei zilnice de prezență.
3. Să gestioneze diversele tipuri de plăți și înregistrarea lor.
4. Să asigure accesului în aplicație, pe baza rolurilor, atât a părinților cât și a cadrelor didactice.
5. Să existe posibilitatea de a genera rapoarte.

În mod evident, înregistrarea și gestionarea datelor implică și conectarea aplicației la o bază de date. Tipul bazei de date alese este MySQL pentru că această componentă software este distribuită în mod gratuit. Clasele implementate în proiect: ”Children”, ”Instructor”, ”Group”, ”Enrollment”, ”DailyAttendance”, ”Payments” și ”FeeTypes” vor permite afișarea, editarea și ștergerea oricărei înregistrări. Pentru o mai bună accesibilitate în afișarea datelor se va implementa un modul care să permită funcțiile specifice pentru o bună paginare a rezultatelor din listele generate. Instrumentul ales este Widgetul Kendo UI Grid care oferă ”un control puternic pentru afișarea datelor într-un format tabelar. Acesta oferă numeroase opțiuni, cum ar fi paginarea, sortarea, filtrarea, gruparea și editarea, care determină modul în care datele sunt prezentate și manipulate. Gridul poate fi legat de date locale sau de la distanță utilizând componenta Kendo UI DataSource.”[1]

2.1 Înregistrarea și gestionarea datelor de bază

Pentru înregistrarea și gestionarea datelor de bază legate de copii, educatori și grupe, se vor inițializa tabelele necesare în baza de date MySQL, proces derulat automat de aplicația noastră.

În continuare, voi prezenta structura claselor care vor alcătui componenta de bază a aplicației:

- Clasa denumită ”Children”, având următoarele proprietăți:
  - Cheia primară ”CID” de tip int.
  - Coloana ”FirstName” de tip string care va conține numele.
  - Coloana ”LastName” de tip string care va conține prenumele.
  - Coloana ”CNP” de tip string care va conține CNP-ul.
  - Coloana ”Address” de tip string care va conține adresa.
  - Coloana ”City” de tip string care va conține orașul.

[1] https://docs.telerik.com/kendo-ui/controls/data-management/grid/overview

- Coloana ”MothersName” de tip string care va conține numele mamei.
- Coloana ”FathersName” de tip string care va conține numele tatălui.
- Coloana ”ContactEmail” de tip string care va conține adresa de EMail a unuia dintre părinți. Această adresă va fi folosită ca și o cheie pentru corelarea conturilor părinților de informațiile legate de copil.

Pe langă datele care sunt introduse de către utilizatorul aplicației, clasa va conține și proprietăți derivate sau calculate din alte proprietăți ale aceleiași clase.

De exemplu:

- Coloana ”FullName” de tip string care va concatena numele și prenumele.

```
public string FullName
        {
            get
            {
               return string.Format("{0} {1}", FirstName, LastName);
            }
        }
```

- Coloana ”Gender” de tip string care va conține genul copilului, informație calculată din CNP-ul introdus.
- Coloana ”DOB” – Date of Birth, de tipul DateTime care va genera data de naștere a copilului în funcție de CNP-ul introdus.

- Clasa denumită ”Instructor”, având următoarele proprietăți:
  - Cheia primară ”InstructorID” de tip int.
  - Coloana ”FirstName” de tip string care va conține numele.
  - Coloana ”LastName” de tip string care va conține prenumele.
  - Coloana ”StartDate” de tip DateTime care va conține data la care instructorul a început activitatea.
  - Coloana ”EndDate” de tip DateTime care va conține data la care instructorul a încetat activitatea.
  - Coloana ”PayRate” de tip decimal care va conține salariul lunar al instructorului.
  - Coloana ”Email” de tip string care va conține adresa de email.
  - Coloana ”FullName” de tip string care va concatena numele și prenumele.

- Clasa denumită ”Group” care va conține următoarele proprietăți:
  - Cheia primară ”GroupID” de tip int.
  - Coloana ”GroupName” de tip string care va conține numele.

2.2 Înregistrarea copiilor în grupe și gestionarea fișei zilnice de prezență

Pentru acest proces vom avea nevoie de informații suplimentare din alte tabele iar din acest motiv, vom folosi ”Foreign Key”-urile atât în tabelele din baza de date cât și în clasele aplicației.

- Clasa denumită ”Enrollments” având următoarele proprietăți:
    - Cheia primară ”EnrollmentID” de tip int.
    - Coloana ”CID” de tip int care va face referire la ID-ul din tabela ”Children” (Foreign Key).
    - Coloana ”InstructorID” de tip int care va face referire la ID-ul din tabela ”Instructor” (Foreign Key).
    - Coloana ”GroupID” de tip int care va face referire la ID-ul din tabela ”Group” (Foreign Key).
    - Coloana ”Date” de tip DateTime care va conține data înregistrării în grupă.

Pentru a accesa datele suplimentare ale tabelelor accesibile prin ”Foreign Key”, se vor include și proprietățile virtuale ale tabelelor menționate:

```
public virtual Group Groups { get; set; }
public virtual Children Children { get; set; }
public virtual Instructor Instructors { get; set; }
```

- Clasa denumită ”DailyAttendance” cu următoarele proprietăți:
    - Cheia primară ”AttendanceID” de tip int.
    - Coloana ”Att_Date” de tip DateTime care va conține data înregistrării.
    - Coloana ”isPresent” de tip bool care va conține valoarea ”true” sau ”false”.
    - Coloana ”Notes” de tip string care va conține note adiționale.

De asemenea, clasa va conține și referința la tabela ”Children”:

```
public virtual Children Children { get; set; }
```

2.2 Gestionarea diverselor tipuri de plăți și înregistrarea lor

Pentru gestionarea plăților, aplicația va folosi două tabele:

- Clasa ”Payments” având următoarele proprietăți:
    - Cheia primară ”PaymentID” de tip int.
    - Coloana ”CID” de tip int care va face referire la ID-ul din tabela ”Children” (ForeignKey).
    - Coloana ”Amount” de tip decimal care va conține suma plătită.
    - Coloana ”FeeID” de tip int care va face referire la ID-ul din tabela ”FeeTypes” (ForeignKey).
    - Coloana Date de tip DateTime care va conține data plății.

De asemenea, vom enumera și tabelele pe care le invocăm:

```
public virtual Children Children { get; set; }
public virtual FeeTypes FeeTypes { get; set; }
```

- Clasa ”FeeTypes” care va conține următoarele proprietăți:
    - Cheia primară ”FeeID” de tip int.

- Coloana ”FeeType” de tip string care va conține numele.

2.3 Asigurarea accesului în aplicație, pe baza rolurilor, atât a părinților cât și a cadrelor didactice

Se va implementa modulul de autorizare și autentificare specific arhitecturii software de tip ”Model-View-Controller”.

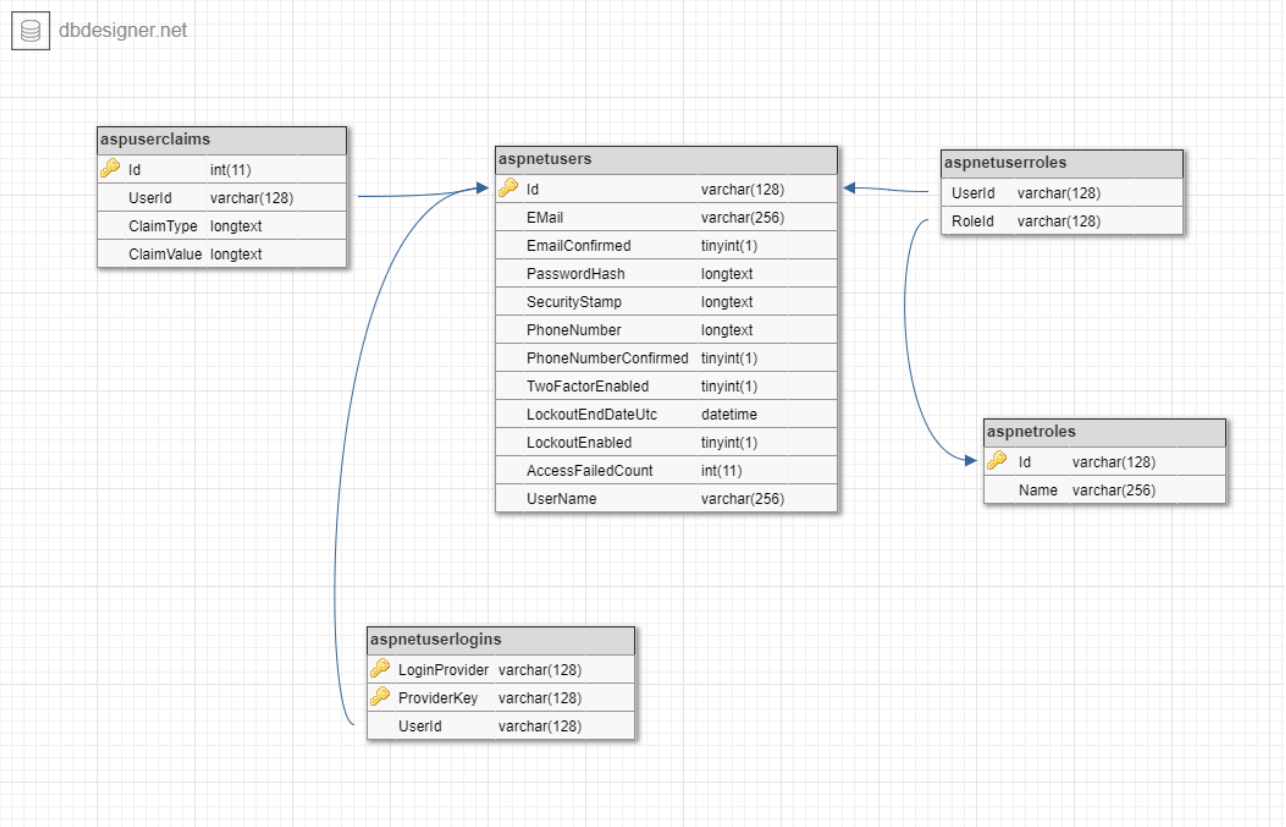


Fig. 1.1 – Reprezentarea grafică a tabelelor de autorizare specifice aplicației ASP.NET MVC

2.4 Generarea rapoartelor

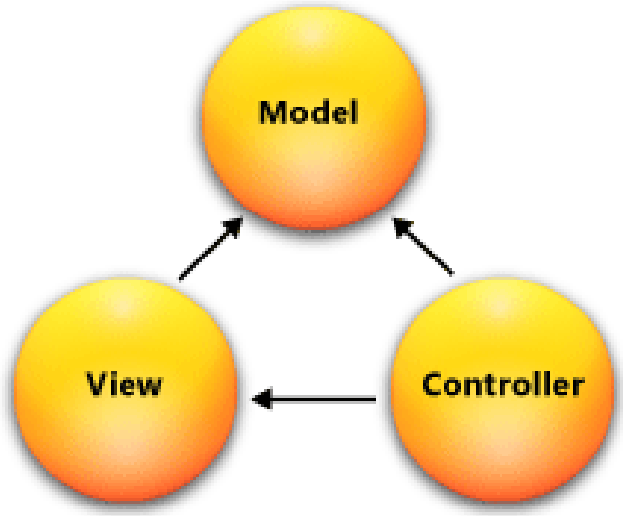
Procesul de generare a rapoartelor se va realiza prin accesarea informațiilor disponibile în baza de date și prin manipularea datelor în funcție de utilizatorul aplicației. De exemplu: un părinte dorește să acceseze fișa de prezență a copilului său sau dorește să acceseze situația plăților

efectuate către grădiniță. În același timp, aplicația va oferi o perspectivă diferită asupra celor două enunțuri de mai sus, pentru un utilizator înregistrat cu drepturi de ”Administrator” sau ”Staff”.

3. Prezentarea arhitecturii software a aplicației

Arhitectura software folosită în acest proiect este de tipul MVC (Model-View-Controller) care împarte aplicația în 3 componente principale: ”modelul”, ”view-ul” și ”controller-ul”. Aceasta reprezintă o alternativă la aplicațiile de tip ASP.NET pentru crearea aplicațiilor web și un instrument ușor de folosit, ușor testabil și integrat cu funcțiile ASP.NET existente, cum ar fi autentificarea bazată pe roluri. Framework-ul MVC este definit în modulul System.Web.Mvc.

Fig. 4.1 – Reprezentarea grafică a elementelor care alcătuiesc arhitectura MVC



Sursa: https://i-msdn.sec.s-msft.com/dynimg/IC263184.png

MVC este un model de arhitectură software pe care mulți programatori îl cunosc. Unele tipuri de aplicații Web vor beneficia de arhitectura MVC iar alții vor continua să utilizeze modelul tradițional de aplicații ASP.NET care se bazează pe Formulare Web și postback-uri. Alte tipuri de aplicații Web vor combina cele două abordări; nici o abordare nu o exclude pe cealaltă. Arhitectura MVC include următoarele componente:

- Modele - Obiectele model sunt părțile aplicației care implementează logica domeniului de date al aplicației. Adesea, obiectele de model recuperează și stochează starea modelului într-o bază de date. De exemplu, un obiect poate prelua informații dintr-o bază de date, poate opera pe acesta și apoi scrie informații actualizate înapoi într-un tabel dintr-o bază de date SQL Server. În aplicațiile mici, modelul este deseori o separare conceptuală în loc de una fizică. De exemplu, dacă aplicația citește doar un set de date și

o trimite în vizualizare, aplicația nu are un strat de model fizic și clase asociate. În acest caz, setul de date preia rolul unui obiect model.[2]

- View-urile - Sunt componentele care afișează interfața cu utilizatorul (UI) a aplicației. În mod obișnuit, această interfață este creată din datele modelului. Un exemplu ar fi o vizualizare de editare a unui tabel care afișează casetele de text, listele derulante și casetele de selectare pe baza stării actuale a unui obiect.[3]

- Controller-urile - sunt componentele care manipulează interacțiunea cu utilizatorul, lucrează cu modelul și, în cele din urmă, selectează un View care afișează interfața. Într-o aplicație MVC, vizualizarea afișează doar informații; controlerul se ocupă și răspunde la intrarea și interacțiunea utilizatorilor. De exemplu, Controller-ul se ocupă de valorile șirului de interogare și transmite aceste valori modelului, care la rândul său ar putea utiliza aceste valori pentru a interoga baza de date. Modelul MVC vă ajută să creați aplicații care separă diferitele aspecte ale aplicației (logica de intrare, logica de afaceri și logica UI), oferind în același timp o legătură slabă între aceste elemente. Modelul specifică unde trebuie localizat fiecare tip de logică în aplicație. Logica UI aparține în View. Logica de intrare aparține Controller-ului. Logica afacerii aparține Modelului. Această separare vă ajută să gestionați complexitatea atunci când construiți o aplicație, deoarece vă permite să vă concentrați asupra unui aspect al implementării la un moment dat. De exemplu, puteți să vă concentrați asupra vizualizării fără să depindeți de logica de afaceri. Cuplajul liber între cele trei componente principale ale unei aplicații MVC promovează de asemenea dezvoltarea paralelă. De exemplu, un programator poate lucra la vizualizare, un al doilea dezvoltator poate lucra pe logica controlerului, iar un al treilea dezvoltator se poate concentra pe logica de afaceri din Model.[4]

3.1 Avantajele unei aplicații Web cu arhitectura MVC

Există o sumă de beneficii atunci când utilizăm arhitectura de tip MVC în cadrul unei aplicații Web dar voi enumera în continuare principalele avantaje ale acestei arhitecturi:

- Facilitează gestionarea complexității prin împărțirea aplicației în Model, View și Controller.[5]
- Nu utilizează formulare bazate pe stare sau pe server. Acest lucru face din arhitectura MVC un instrument ideal pentru programatorii care doresc control absolut asupra comportamentului unei aplicații.[6]

[2] https://msdn.microsoft.com/en-us/library/dd381412(v=vs.108).aspx
[3] *Ibidem*
[4] *Ibidem*
[5] *Ibidem*
[6] *Ibidem*

- Utilizează un model de Controller frontal care procesează cererile de aplicații Web printr-un singur Controller. Acest lucru vă permite să proiectați o aplicație care să susțină o infrastructură bogată de rutare.[7]
- Oferă un sprijin mai bun pentru dezvoltarea bazată pe teste.[8]
- Funcționează bine pentru aplicațiile Web care sunt realizate de echipe mari de dezvoltatori și pentru designerii de web care au nevoie de un grad ridicat de control asupra comportamentului aplicației.[9]

3.2 Caractersticile arhitecturii MVC

Pe lângă beneficiile oferite de utilizarea arhitecturii MVC, caracteristicile includ:

- Separarea sarcinilor aplicației (logica de intrare, logica de afaceri și logica UI), testabilitatea și dezvoltarea bazată pe teste (TDD). Toate contractele de bază din cadrul MVC se bazează pe interfață și pot fi testate folosind obiecte de test, obiecte simulate care imită comportamentul obiectelor reale din aplicație. Puteți testa aplicația fără a fi nevoie să rulați controlorii într-un proces ASP.NET, ceea ce face ca testarea unității să fie rapidă și flexibilă. Se poate utiliza orice cadru de testare a unităților care este compatibil cu .NET Framework. [10]

- Oferirea unui cadru extensibil și conectabil. Componentele aplicației ASP.NET MVC sunt concepute astfel încât să poată fi ușor înlocuite sau personalizate. Puteți conecta propriul motor de vizualizare, politica de rutare a adreselor URL, serializarea parametrilor de acțiune și alte componente. Cadrul ASP.NET MVC sprijină, de asemenea, utilizarea modelelor pentru containere de injecție ” Dependency Injection” (DI) și inversarea controlului ” Inversion of Control” (IOC). DI vă permite să injectați obiecte într-o clasă, în loc să vă bazați pe clasă pentru a crea obiectul în sine. IOC specifică faptul că, dacă un obiect necesită un alt obiect, primele obiecte ar trebui să obțină al doilea obiect dintr-o sursă exterioară, cum ar fi un fișier de configurare. Acest lucru facilitează testarea.[11]

- Sprijin extins pentru rutarea ASP.NET, care este o componentă puternică de cartografiere a adreselor URL care vă permite să construiți aplicații care au adrese URL inteligibile și care pot fi căutate. Adresele URL nu trebuie să includă extensii de nume de fișiere și sunt proiectate să susțină modele de denumire a adreselor URL care funcționează bine pentru adresarea de optimizare a motoarelor de căutare (SEO) și adresă de reprezentare a transferului (REST).[12]

[7] https://msdn.microsoft.com/en-us/library/dd381412(v=vs.108).aspx
[8] *Ibidem*
[9] *Ibidem*
[10] *Ibidem*
[11] *Ibidem*
[12] *Ibidem*

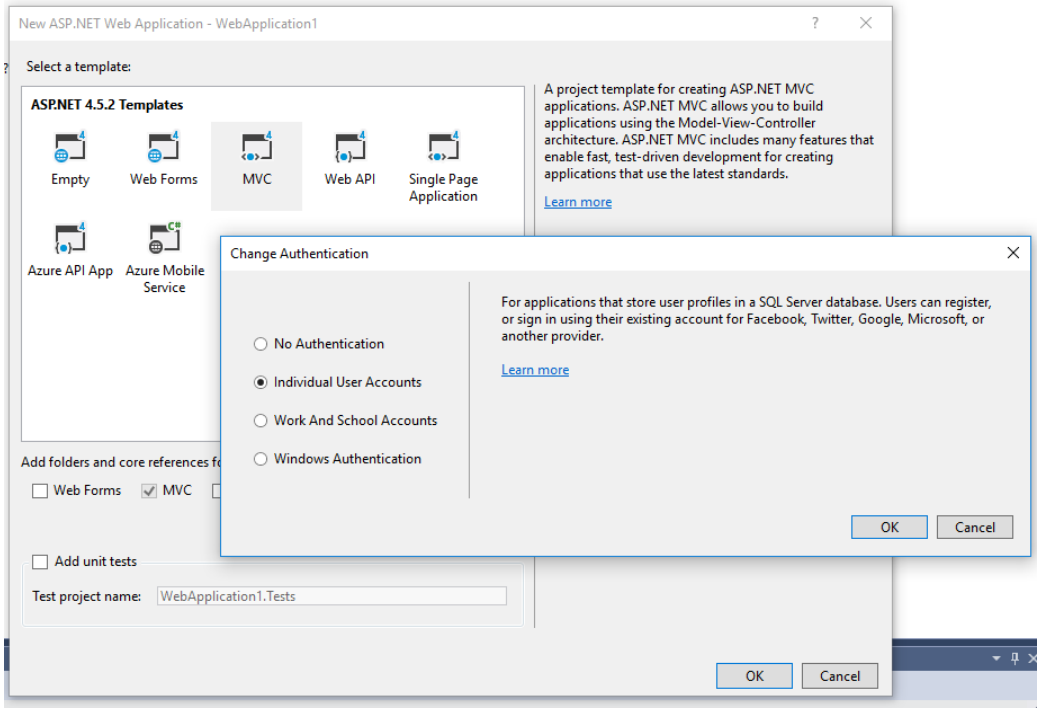
- Suport pentru utilizarea marcajelor în paginile ASP.NET existente (fișiere .aspx), fișierele de control al utilizatorului (fișiere .ascx) și fișierele de marcare a paginilor principale (fișiere .master) ca șabloane de vizualizare. Aveți posibilitatea să utilizați funcțiile ASP.NET existente cu arhitectura MVC ASP.NET, cum ar fi paginile master imbricate, expresiile in-line (<% =%>), controalele declarative ale serverului, șabloanele, legarea datelor, localizarea și așa mai departe.[13]

- Suport pentru caracteristicile ASP.NET existente. ASP.NET MVC vă permite să utilizați caracteristici cum ar fi autentificarea formularelor și autentificarea Windows, autorizația URL-ului, statutul de membru și roluri, cache-ul de ieșire și de date, gestionarea stării de stare și a profilului, monitorizarea sănătății, sistemul de configurare și arhitectura furnizorului.[14]

4. Prezentarea detaliată a modului de rezolvare a temei

Pentru rezolvarea temei am început cu șablonul unei aplicații Web, ASP.NET MVC din Visual Studio, având opțiunea de ”Individual User Accounts” selectată.

Fig. 5.1 – Selectarea tipului aplicației



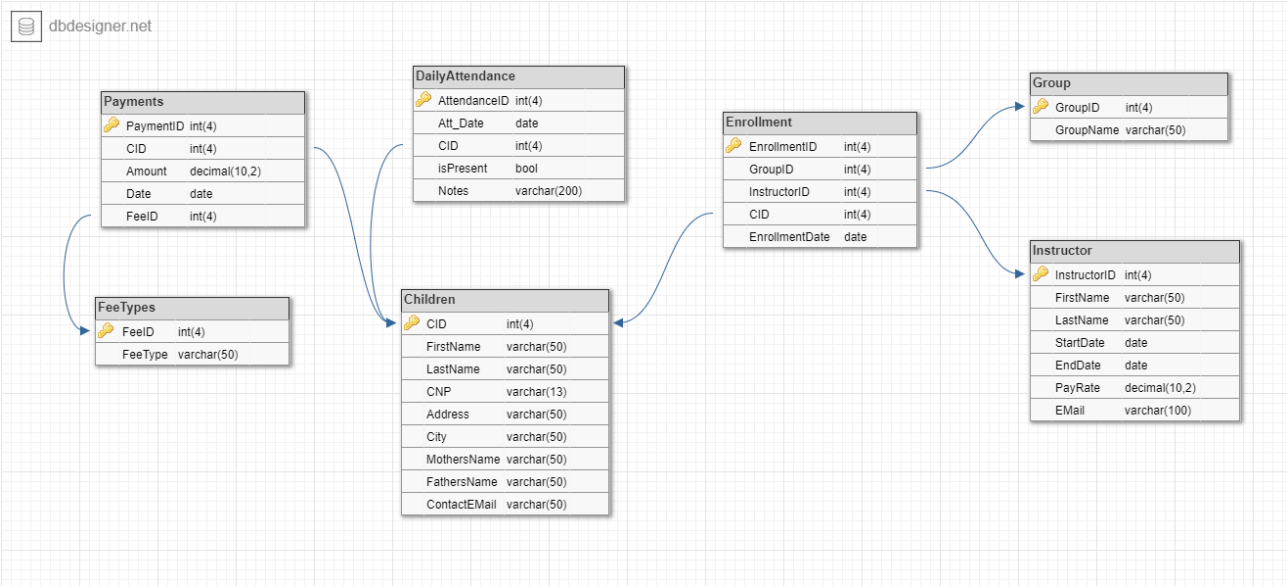
În continuare, am modificat fișierul: Views\Shared_Layout.cshtml, pentru a include informațiile mele:

[13] https://msdn.microsoft.com/en-us/library/dd381412(v=vs.108).aspx
[14] *Ibidem*

```
<nav class="navbar navbar-inverse navbar-fixed-top"
style="background-color:#ffffff; color:#463265">
        <div class="container">
            <div class="navbar-header">
                <button type="button" class="navbar-toggle" data-
toggle="collapse" data-target=".navbar-collapse">
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                </button>
                @Html.ActionLink("KDG Manager", "Index", "Home", new
{ area = "" }, new { @class = "navbar-brand", @style =
"color:#463265" })
            </div>
            <div class="navbar-collapse collapse">

                <div class="nav navbar-nav">

                    <div class="floating-
box">@Html.ActionLink("Home", "Index", "Home", null, htmlAttributes:
new { @class = "btn btn-primary navbar-btn" })</div>
                    <div class="floating-
box">@Html.ActionLink("Contact", "Contact", "Home", null,
htmlAttributes: new { @class = "btn btn-primary navbar-btn" })</div>
                    @if (Request.IsAuthenticated)
                    {
                        <div class="floating-
box">@Html.ActionLink("Dashboard", "Index", "Users", null,
htmlAttributes: new { @class = "btn btn-danger navbar-btn" })</div>
                    }
                </div>

                @Html.Partial("_LoginPartial")

            </div>
        </div>
    </nav>
    <div class="container body-content">
        @RenderBody()
        <hr />
        <footer>
            <p>&copy; @DateTime.Now.Year - KDG Manager (Proiect
disertatie - MIHAI BENEGUI)</p>
        </footer>
    </div>
```

Am adăugat referințele necesare construirii proiectului: Entity Framework 6 și MySQL Connector Net 6.9.10 pentru conectarea la baza de date MySQL.

Am implementat clasele de bază ale proiectului, conform structurii de mai jos:



Clasa principală care coordonează funcționalitatea Entity Framework pentru un model de date cunoscut este clasa de context a bazei de date. Am creat această clasă derivând din clasa System.Data.Entity.DbContext și am denumit-o *AppContext.cs* în directorul *DataAccessLayer:*

```
namespace Kdg_MVC.DataAccessLayer
{
    public class AppContext : DbContext
    {
        public AppContext() : base("DefaultConnection")
        {
        }

        public DbSet<Children>Children { get; set; }
        public DbSet<Enrollment> Enrollments { get; set; }
        public DbSet<Group> Groups { get; set; }
        public DbSet<Instructor> Instructors { get; set; }
        public DbSet<DailyAttendance> DailyAttendances { get; set; }

        protected override void OnModelCreating(DbModelBuilder
modelBuilder)
        {

modelBuilder.Conventions.Remove<PluralizingTableNameConvention>();
        }
    }
}
```

Pentru ca Entity Framework să utilizeze această clasă, am modificat fișierul Web.config al aplicației după cum urmează:

```
<configuration>
  <configSections>
    <section name="entityFramework"
type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection,
EntityFramework, Version=6.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089" requirePermission="false" />
  </configSections>
  <connectionStrings>
    <!-- Modifca aici detaliile conexiunii MySQL!-->
    <add name="DefaultConnection"
connectionString="Server=localhost;Database=kdg;Uid=dbuser;Pwd=dbuse
r;" providerName="MySql.Data.MySqlClient" />
  </connectionStrings>
  <appSettings>

      <provider invariantName="MySql.Data.MySqlClient"
type="MySql.Data.MySqlClient.MySqlProviderServices,
MySql.Data.Entity.EF6, Version=6.9.10.0, Culture=neutral,
PublicKeyToken=c5687fc88969c44d"></provider>
      <provider invariantName="System.Data.SqlClient"
type="System.Data.Entity.SqlServer.SqlProviderServices,
EntityFramework.SqlServer" />
    </providers>
    <defaultConnectionFactory
type="System.Data.Entity.Infrastructure.LocalDbConnectionFactory,
EntityFramework">
      <parameters>
        <parameter value="mssqllocaldb" />
      </parameters>
    </defaultConnectionFactory>
  </entityFramework>
  <system.codedom>
    <compilers>
      …//Removed
    </compilers>
  </system.codedom>
  <system.data>
    <DbProviderFactories>
      <remove invariant="MySql.Data.MySqlClient" />
      <add name="MySQL Data Provider"
invariant="MySql.Data.MySqlClient" description=".Net Framework Data
Provider for MySQL" type="MySql.Data.MySqlClient.MySqlClientFactory,
MySql.Data, Version=6.9.10.0, Culture=neutral,
PublicKeyToken=c5687fc88969c44d" />
    </DbProviderFactories>
  </system.data>
</configuration>
```

Entity Framework folosește mai întâi o tabelă MigrationHistory pentru a urmări modificările modelului și pentru a asigura coerența între schema bazei de date și schema conceptuală. Cu toate

acestea, acest tabel nu funcționează în mod implicit pentru MySQL deoarece cheia primară este prea mare. Pentru a remedia această situație, va trebui să micșorăm dimensiunea cheii pentru acel tabel[15], urmând pașii de mai jos:

- Informațiile despre schemă pentru acest tabel sunt captate într-un text HistoryContext, care poate fi modificat ca orice alt DbContext. Pentru a face acest lucru, am adăugat un director nou aplicației, denumit *MySQLConfig,* în care am adăugat o clasă *MySqlHistoryContext.cs* care conține următorul cod:

```
namespace Kdg_MVC.MySQLConfig
{
    public class MySqlHistoryContext : HistoryContext
    {
        public MySqlHistoryContext(
          DbConnection existingConnection,
          string defaultSchema)
        : base(existingConnection, defaultSchema)
        {
        }

        protected override void OnModelCreating(DbModelBuilder modelBuilder)
        {
            base.OnModelCreating(modelBuilder);
            modelBuilder.Entity<HistoryRow>().Property(h =>
h.MigrationId).HasMaxLength(100).IsRequired();
            modelBuilder.Entity<HistoryRow>().Property(h =>
h.ContextKey).HasMaxLength(200).IsRequired();
        }
    }
}
```

- Apoi va trebui să configurăm Entity Framework pentru a utiliza istoricul modificat HistoryContext, nu unul implicit. Acest lucru se poate realiza prin folosirea funcțiilor de configurare bazate pe coduri. Astfel, am adăugat un nou fișier clasă denumit *MySqlConfiguration.cs* având următorul conținut:

```
namespace Kdg_MVC.MySQLConfig
{
    public class MySqlConfiguration : DbConfiguration
    {
        public MySqlConfiguration()
        {
            SetHistoryContext(
            "MySql.Data.MySqlClient", (conn, schema) => new MySqlHistoryContext(conn,
schema));
        }
    }
}
```

- Furnizorul MySQL nu acceptă în prezent migrări ale Entity Framework, deci va trebui să utilizăm inițializatori de modele pentru a ne conecta la baza de date.

[15] https://docs.microsoft.com/en-us/aspnet/identity/overview/getting-started/aspnet-identity-using-mysql-storage-with-an-entityframework-mysql-provider

Astfel, vom creea un inițializator personalizat pentru Entity Framework prin crearea unei noi clase *MySqlInitializer.cs* care va conține următorul cod:

```
namespace Kdg_MVC.MySQLConfig
{
    public class MySqlInitializer :
IDatabaseInitializer<ApplicationDbContext>
    {
        public void InitializeDatabase(ApplicationDbContext context)
        {
            if (!context.Database.Exists())
            {
                // if database did not exist before - create it
                context.Database.Create();

                //Create Data Tables
                DatabaseRepository.CreateDataTables(context);

                //Create users and roles.
                DatabaseRepository.CreateUsersAndRoles(context);

                //Start Seeding the Database;
                SeedData.Seed();
            }
            else
            {
                // query to check if MigrationHistory table is present in
the database
                var migrationHistoryTableExists =
((IObjectContextAdapter)context).ObjectContext.ExecuteStoreQuery<int>(
                  "SELECT COUNT(*) FROM information_schema.tables WHERE
table_schema = 'IdentityMySQLDatabase' AND table_name =
'__MigrationHistory'");

                // if MigrationHistory table is not there (which is the
case first time we run) - create it
                if (migrationHistoryTableExists.FirstOrDefault() == 0)
                {
                    context.Database.Delete();
                    context.Database.Create();
                    DatabaseRepository.CreateDataTables(context);
                    DatabaseRepository.CreateUsersAndRoles(context);
                    SeedData.Seed();
                }
            }
        }
    }
}
```

- Deschidem fișierul *IdentityModels.cs* din directorul *Models* și înlocuim conținutul acestuia cu următoarele:

```
namespace Kdg_MVC.Models
{
    // You can add profile data for the user by adding more
properties to your ApplicationUser class, please visit
http://go.microsoft.com/fwlink/?LinkID=317594 to learn more.
    public class ApplicationUser : IdentityUser
    {
        public async Task<ClaimsIdentity>
GenerateUserIdentityAsync(UserManager<ApplicationUser> manager)
        {
            // Note the authenticationType must match the one
defined in CookieAuthenticationOptions.AuthenticationType
            var userIdentity = await
manager.CreateIdentityAsync(this,
DefaultAuthenticationTypes.ApplicationCookie);
            // Add custom user claims here
            return userIdentity;
        }
    }
    public class ApplicationDbContext :
IdentityDbContext<ApplicationUser>
    {
        static ApplicationDbContext()
        {
            Database.SetInitializer(new MySqlInitializer());
        }

        public ApplicationDbContext()
          : base("DefaultConnection")
        {
        }

        public static ApplicationDbContext Create()
        {
            return new ApplicationDbContext();
        }
    }
}
```

În acest moment, avem o aplicație funcțională care permite conectarea la baza de date MySQL și implementează funcțiile de roluri ale ASP.NET.

Mai departe, vom integra funcțiile pentru gestionarea rolurilor utilizatorilor. Pentru început vom implementa rolul de Admin și primul utilizator. Pentru a mă asigura că informațiile sunt create la inițializare, am decis să corelez funcția de implementare a rolurilor și popularea bazei de date, de momentul inițializării MySQL. Astfel, am efectuat următoarele modificări:

Am creat un director *Lib,* ca subdirector al *MySQLConfig* în care am adăugat o clasă nouă denumită *SeedData* al cărei scop este de a popula baza de date cu roluri predefinite și a construi tabelele MySQL ale claselor Model:

```
public static void AddUserRoles(ApplicationDbContext context)
        {
            var roleManager = new RoleManager<IdentityRole>(new
RoleStore<IdentityRole>(context));
            var UserManager = new UserManager<ApplicationUser>(new
UserStore<ApplicationUser>(context));

            if (!roleManager.RoleExists("Admin"))
            {

                // first we create Admin role
                var role = new
Microsoft.AspNet.Identity.EntityFramework.IdentityRole();
                role.Name = "Admin";
                roleManager.Create(role);
            }
```

Funcția de atribuire a rolului de Admin către un utilizator:

```
public static void SeedUsersToRoles(ApplicationDbContext context)
        {
            var UserManager = new UserManager<ApplicationUser>(new
UserStore<ApplicationUser>(context));
            string userPWD = "!Pass123";

            //Here we create a Admin super user who will maintain
the website

            var admin = new ApplicationUser();
            admin.UserName = "mbenegui@gmail.com";
            admin.Email = "mbenegui@gmail.com";

            var adminUser = UserManager.Create(admin, userPWD);

            //Add default User to Role Admin
            if (adminUser.Succeeded)
            {
                var result1 = UserManager.AddToRole(admin.Id,
"Admin");

            }
```

Așa cum am amintit mai sus, în clasa *MySqlInitializer* avem următorul cod care face referire la crearea bazei de date MySQL și adăugarea rolului de Admin împreună cu datele utilizatorului:

```
namespace Kdg_MVC.MySQLConfig
{
    public class MySqlInitializer :
IDatabaseInitializer<ApplicationDbContext>
    {
        public void InitializeDatabase(ApplicationDbContext context)
        {
            if (!context.Database.Exists())
            {
                // if database did not exist before - create it
                context.Database.Create();

                //Create Data Tables
                DatabaseRepository.CreateDataTables(context);

                //Create users and roles.
                DatabaseRepository.CreateUsersAndRoles(context);

                //Start Seeding the Database;
                SeedData.Seed();
            }
            else
            {
                // query to check if MigrationHistory table is
present in the database
                var migrationHistoryTableExists =
((IObjectContextAdapter)context).ObjectContext.ExecuteStoreQuery<int
>(
                  "SELECT COUNT(*) FROM information_schema.tables
WHERE table_schema = 'IdentityMySQLDatabase' AND table_name =
'__MigrationHistory'");

                // if MigrationHistory table is not there (which is
the case first time we run) - create it
                if (migrationHistoryTableExists.FirstOrDefault() ==
0)
                {
                    context.Database.Delete();
                    context.Database.Create();
                    DatabaseRepository.CreateDataTables(context);
                    DatabaseRepository.CreateUsersAndRoles(context);
                    SeedData.Seed();
                }
            }
        }
    }
}
```

```
namespace Kdg_MVC.MySQLConfig.lib
{
    public static class DatabaseRepository
    {
        public static void CreateDataTables(ApplicationDbContext
context)
        {

context.Database.ExecuteSqlCommand(SeedData._CreateTbChildren);

context.Database.ExecuteSqlCommand(SeedData._CreateTbEnrollment);

context.Database.ExecuteSqlCommand(SeedData._CreateTbGroup);

context.Database.ExecuteSqlCommand(SeedData._CreateTbInstructor);

context.Database.ExecuteSqlCommand(SeedData._CreateTbDailyAttendance
);

context.Database.ExecuteSqlCommand(SeedData._CreateTbPayments);

context.Database.ExecuteSqlCommand(SeedData._CreateTbFeeTypes);

context.Database.ExecuteSqlCommand(SeedData._Add_Enrollment_fk0);

context.Database.ExecuteSqlCommand(SeedData._Add_Enrollment_fk1);

context.Database.ExecuteSqlCommand(SeedData._Add_Enrollment_fk2);

context.Database.ExecuteSqlCommand(SeedData._Add_DailyAttendance_fk0
);

context.Database.ExecuteSqlCommand(SeedData._Add_Payments_fk0);

context.Database.ExecuteSqlCommand(SeedData._Add_Payments_fk1);
        }
        public static void CreateUsersAndRoles(ApplicationDbContext
context)
        {
            SeedData.AddUserRoles(context);
            SeedData.SeedUsersToRoles(context);
        }
    }
}
```

Pentru adăugarea modulului de gestionare a rolurilor, vom insera în fișierul *Controllers/AccountControllers.cs* următoarea linie:

```
public class AccountController : Controller
    {
        private ApplicationSignInManager _signInManager;
        private ApplicationUserManager _userManager;
        ApplicationDbContext context;

public AccountController()
        {
            context = new ApplicationDbContext();
        }
```

Folosind obiectul ApplicationDBContext vom primi toate rolurile din baza de date. Pentru înregistrarea utilizatorilor vom afișa doar rolul de ”Parent”, restul rolurilor fiind accesibile doar super-utilizatorului.

```
public ActionResult Register()
        {
            var superRoles = new string[] { "Admin", "Staff",
"Manager" };

            if (User.IsInRole("Admin") || User.IsInRole("Manager"))
            {
                ViewBag.Name = new SelectList(context.Roles.Where(u
=> !u.Name.Contains("Admin"))
                                              .ToList(), "Name",
"Name");
            }

            else
            {
                ViewBag.Name = new SelectList(context.Roles.Where(u
=> !superRoles.Contains(u.Name))
                                                .ToList(), "Name",
"Name");
            }


            return View();
        }
```

În continuare vom creea o clasă nouă, denumită *Controller/UsersController.cs* care va gestiona View-urile în funcție de rolul utilizatorului:

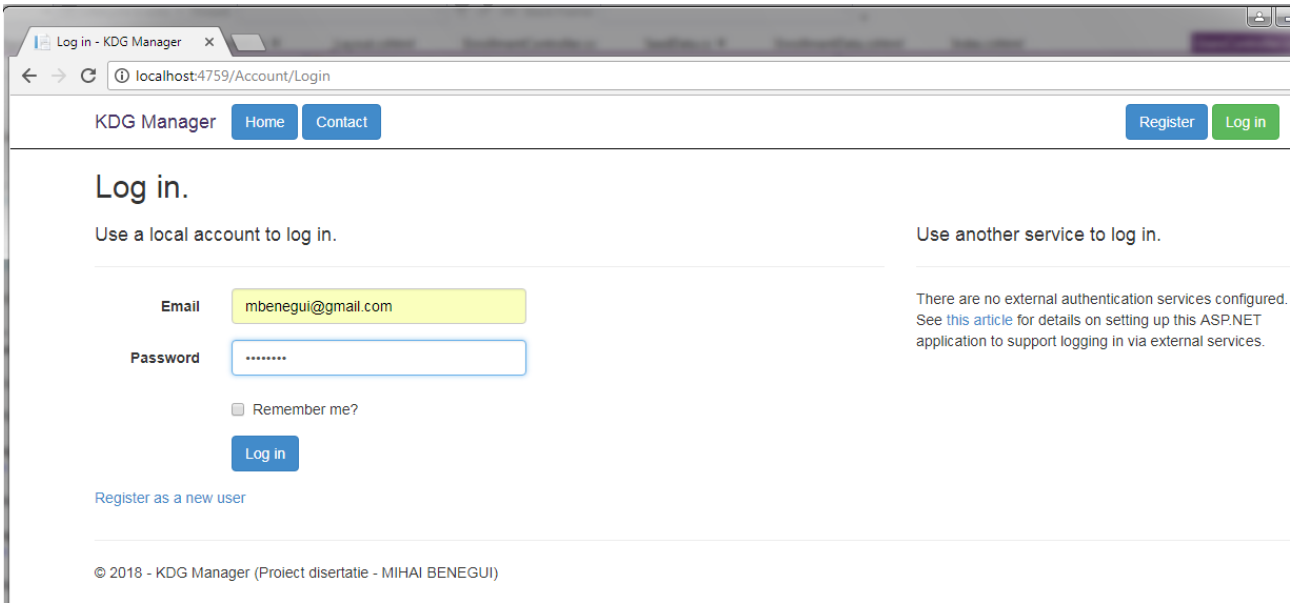
```
namespace Kdg_MVC.Controllers
{
    [Authorize]
    public class UsersController : Controller
    {
        // GET: Users
        public ActionResult Index()
        {
            if (User.Identity.IsAuthenticated)
            {
                var user = User.Identity;
                ViewBag.Name = user.Name;

                ViewBag.displayMenu = "No";

                if (RoleOfUser() == "Admin")
                {
                    ViewBag.displayMenu = "Admin";
                    return View();
                }
                else if (RoleOfUser() == "Parent")
                {
                    ViewBag.displayMenu = "Parent";
                    return View();
                }
            }
            else
            {
                ViewBag.Name = "Not Logged IN";
            }
            return View();
        }

        public string RoleOfUser()
        {
            if (User.Identity.IsAuthenticated)
            {
                var user = User.Identity;
                ApplicationDbContext context = new ApplicationDbContext();
                var UserManager = new UserManager<ApplicationUser>(new
UserStore<ApplicationUser>(context));
                var s = UserManager.GetRoles(user.GetUserId());

                return s[0].ToString();
            }

            else
            {
                return "Not Auth";
            }

        }
    }
}
```

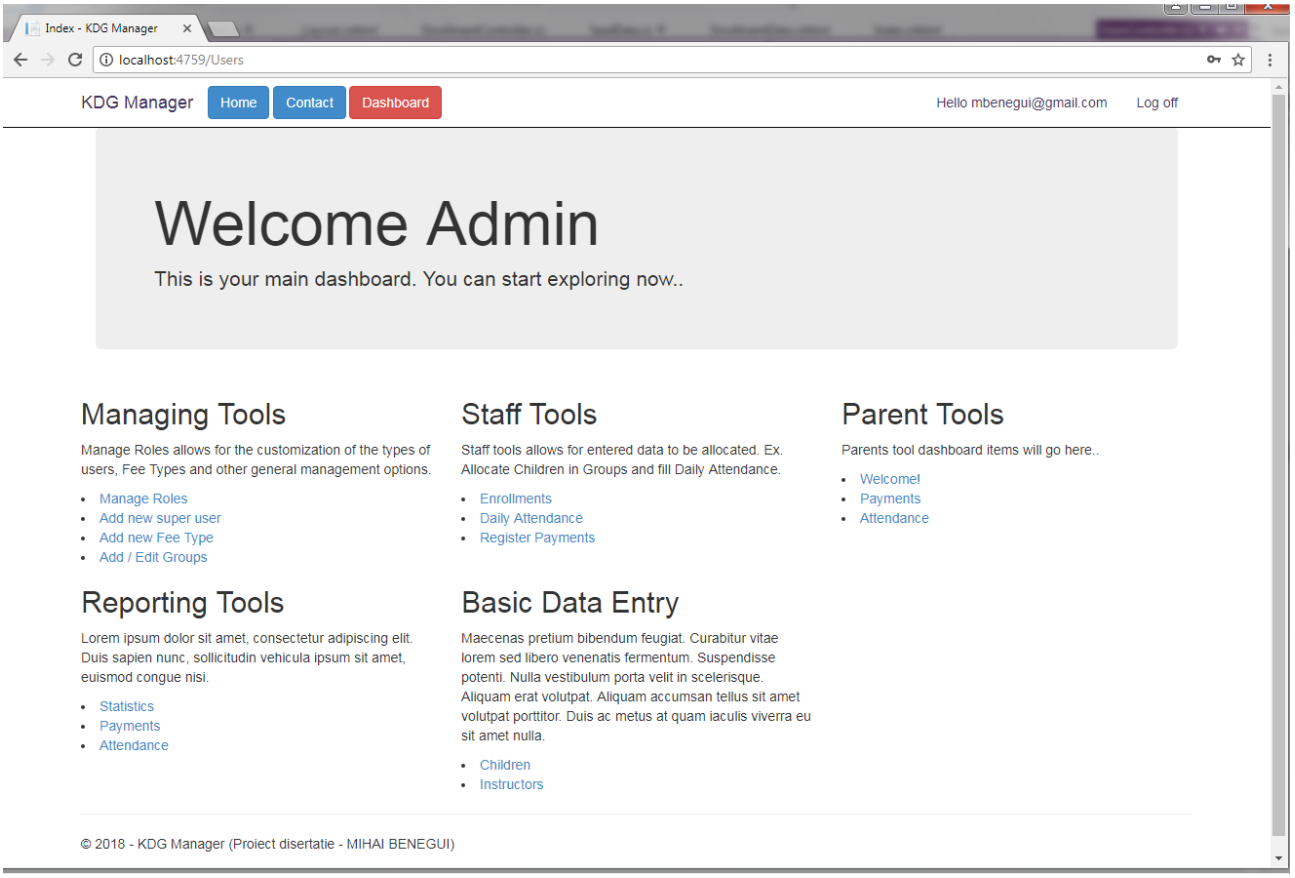
Am creeat în baza de date, în mod automat la inițializare, următoarele roluri generice pentru a exemplifica modul în care aplicația generează View-urile. Astfel, am implementat un rol de ”Admin” pentru user-ul cu adresa de email mbenegui@gmail.com și un rol de ”Parent” pentru user-ul cu adresa de email austen@test.com.

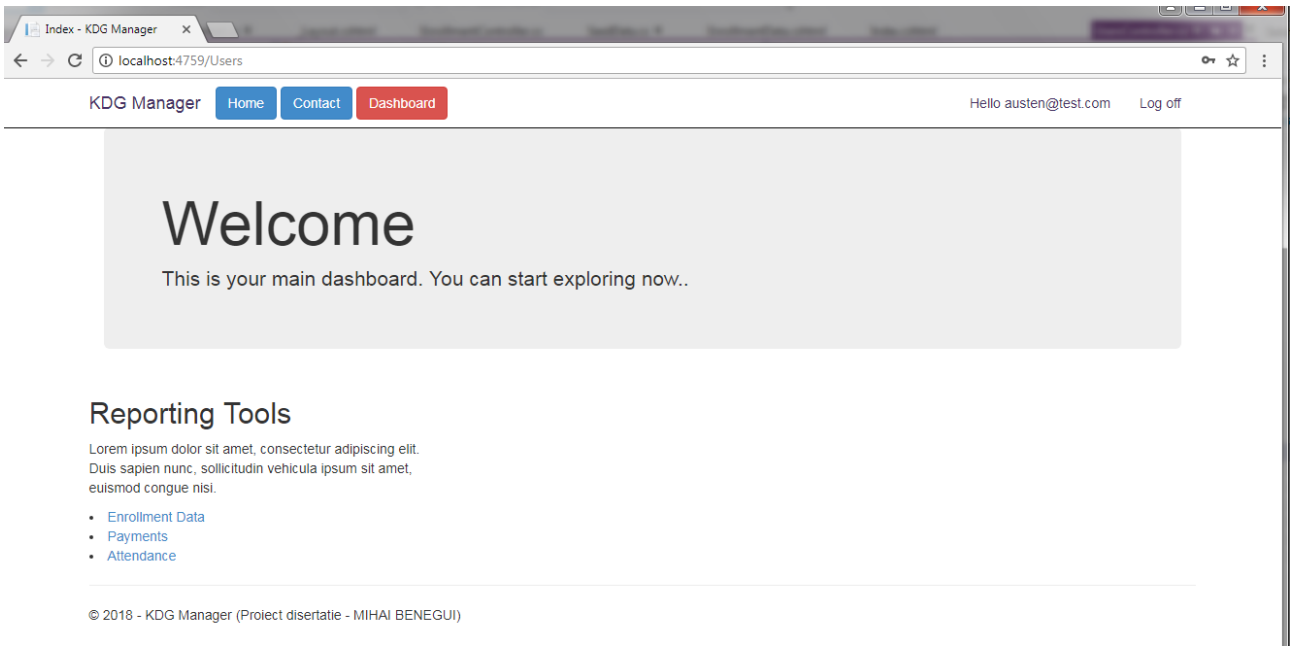
În funcție de rolul determinat de Controller-ul care conține funcția ”Login”, utilizatorul este direcționat către meniul său și View-ul rezultat este afișat.

Astfel, View-ul corespunzător Controller-ului *UsersController.cs* are următoarele informații:

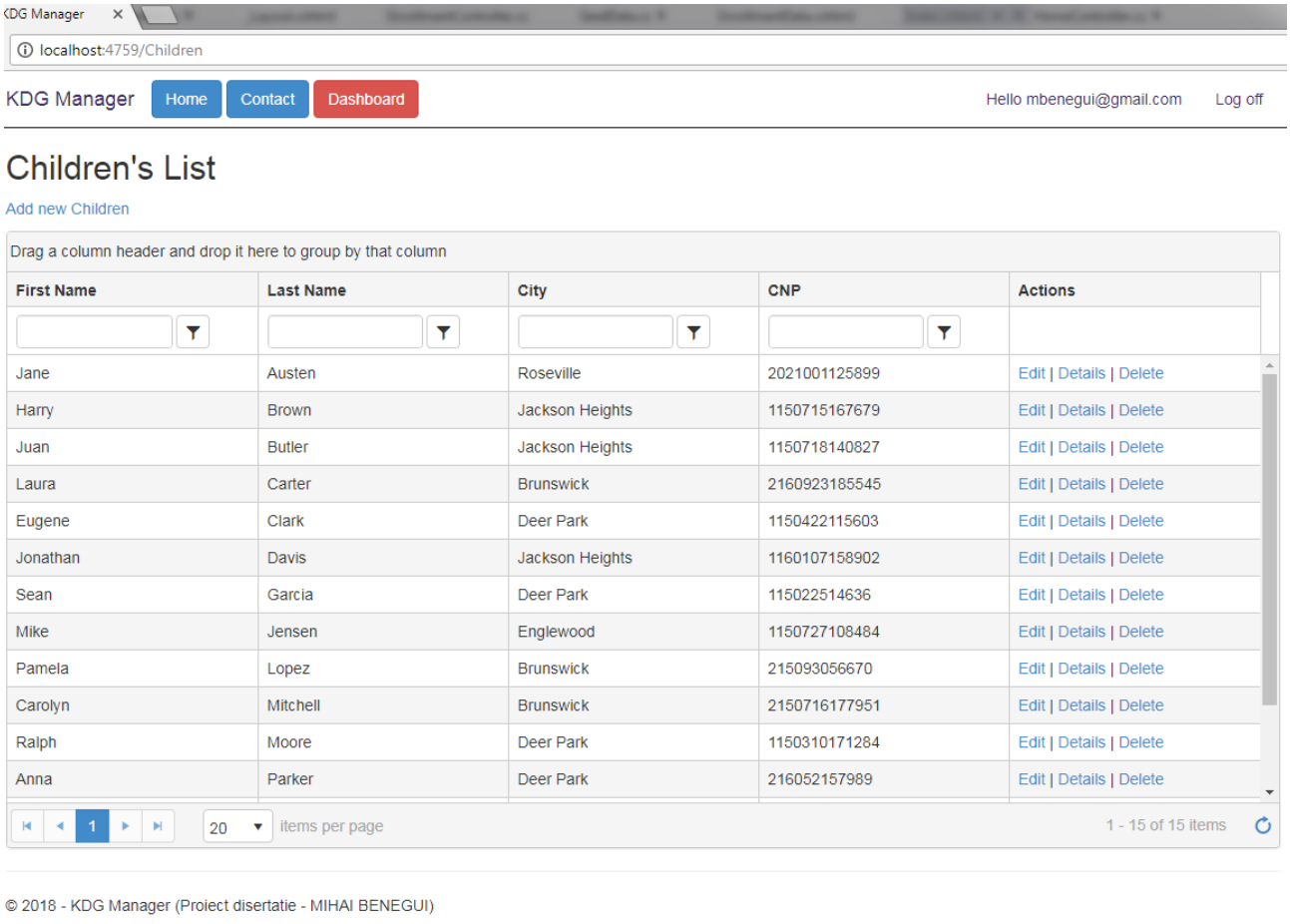
```
@{
    ViewBag.Title = "Index";
}

@if (ViewBag.displayMenu == "Admin")
{
    <div class="container">
        <div class="jumbotron">
            <h1>Welcome Admin</h1>
            <p>
                This is your main dashboard.
                You can start exploring now..
            </p>
        </div>

    </div>
    <div class="row">
        <div class="col-md-4">
            <h2>Managing Tools</h2>
            <p>
                Manage Roles allows for the customization of the types of users, Fee
Types and other general management options.
            </p>
            <li>@Html.ActionLink("Manage Roles", "Index", "Role")</li>
            <li>@Html.ActionLink("Add new super user", "Register", "Account")</li>
            <li>@Html.ActionLink("Add new Fee Type", "Index", "FeeTypes")</li>
            <li>@Html.ActionLink("Add / Edit Groups", "Index", "Group")</li>
        </div>

…//Missing Code

else if (ViewBag.displayMenu == "Parent")
{
    <div class="container">
        <div class="jumbotron">
            <h1>Welcome</h1>
            <p>
                This is your main dashboard.
                You can start exploring now..
            </p>
        </div>

    </div>
    <div class="row">
        <div class="col-md-4">
            <h2>Reporting Tools</h2>
            <p>
                Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis sapien
nunc, sollicitudin vehicula ipsum sit amet, euismod congue nisi.
            </p>
            <li>@Html.ActionLink("Enrollment Data", "EnrollmentData", "Home")</li>
            <li>@Html.ActionLink("Payments", "Payments", "Home")</li>
            <li>@Html.ActionLink("Attendance", "Attendance", "Home")</li>
        </div>
    </div>
}
else
{
    <h2>Welcome! <strong>@ViewBag.Name</strong>This is the main dashboard. Not
implemented yet!</h2>
}
```

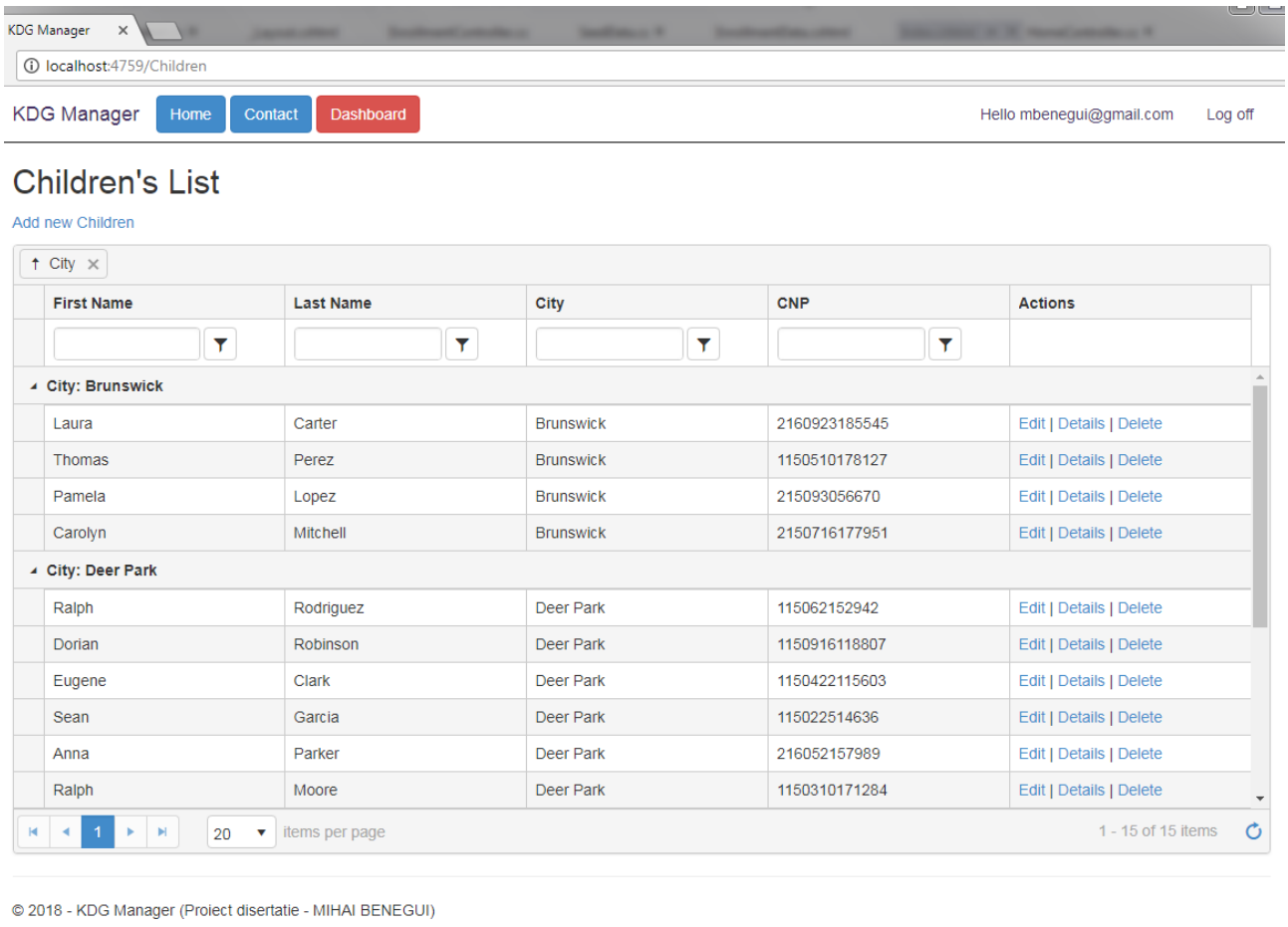

Log in - KDG Manager
localhost:4759/Account/Login
KDG Manager
Home
Contact
Register
Log in
Log in.
Use a local account to log in.
Email
mbenegui@gmail.com
Password
Remember me?
Log in
Register as a new user
Use another service to log in.
There are no external authentication services configured. See this article for details on setting up this ASP.NET application to support logging in via external services.
© 2018 - KDG Manager (Proiect disertatie - MIHAI BENEGUI)

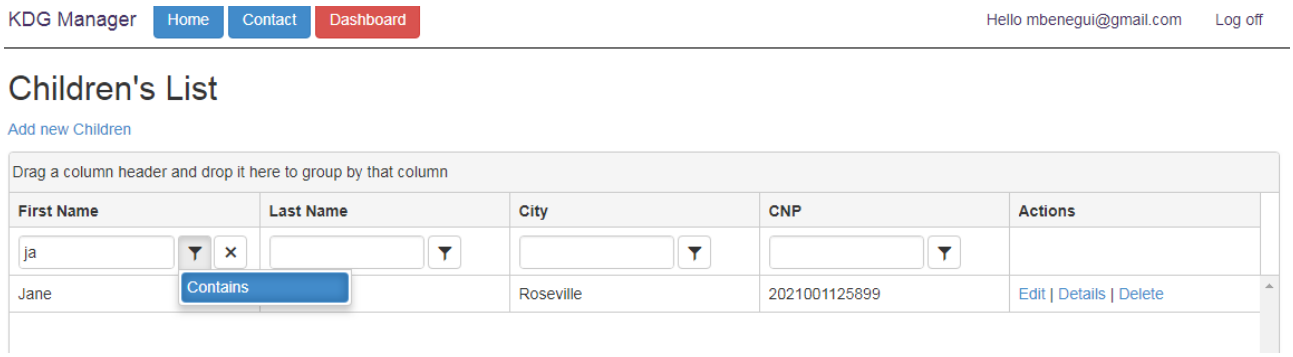

Index - KDG Manager
localhost:4759/Users
KDG Manager
Home
Contact
Dashboard
Hello mbenegui@gmail.com
Log off
Welcome Admin
This is your main dashboard. You can start exploring now..
Managing Tools
Manage Roles allows for the customization of the types of users, Fee Types and other general management options.
Manage Roles
Add new super user
Add new Fee Type
Add / Edit Groups
Staff Tools
Staff tools allows for entered data to be allocated. Ex. Allocate Children in Groups and fill Daily Attendance.
Enrollments
Daily Attendance
Register Payments
Parent Tools
Parents tool dashboard items will go here..
Welcome!
Payments
Attendance
Reporting Tools
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis sapien nunc, sollicitudin vehicula ipsum sit amet, euismod congue nisi.
Statistics
Payments
Attendance
Basic Data Entry
Maecenas pretium bibendum feugiat. Curabitur vitae lorem sed libero venenatis fermentum. Suspendisse potenti. Nulla vestibulum porta velit in scelerisque. Aliquam erat volutpat. Aliquam accumsan tellus sit amet volutpat porttitor. Duis ac metus at quam iaculis viverra eu sit amet nulla.
Children
Instructors
© 2018 - KDG Manager (Proiect disertatie - MIHAI BENEGUI)



Mai departe, am implementat Kendo GRID-ul de la Telerik pentru a obține organizarea listelor de date și a instrumentelor de filtrare și organizare:







Pentru generarea rapoartelor din baza de date, am creat un set de clase în directorul *ViewModels* care conțin proprietățile obiectelor raportate:

Pentru *AttendanceList:*

```
namespace Kdg_MVC.ViewModels
{
    public class AttendanceList
    {
        [Display(Name = "Full Name")]
        public string FullName { get; set; }

        [DataType(DataType.Date)]
        [DisplayFormat(DataFormatString = "{0:dd MMM yyyy}",
ApplyFormatInEditMode = true)]
        [Display(Name = "Attendance Date")]
        public DateTime Date { get; set; }

        [Display(Name = "Attended Class")]
        public string isPresent { get; set; }

        public string Notes { get; set; }

    }
}
```

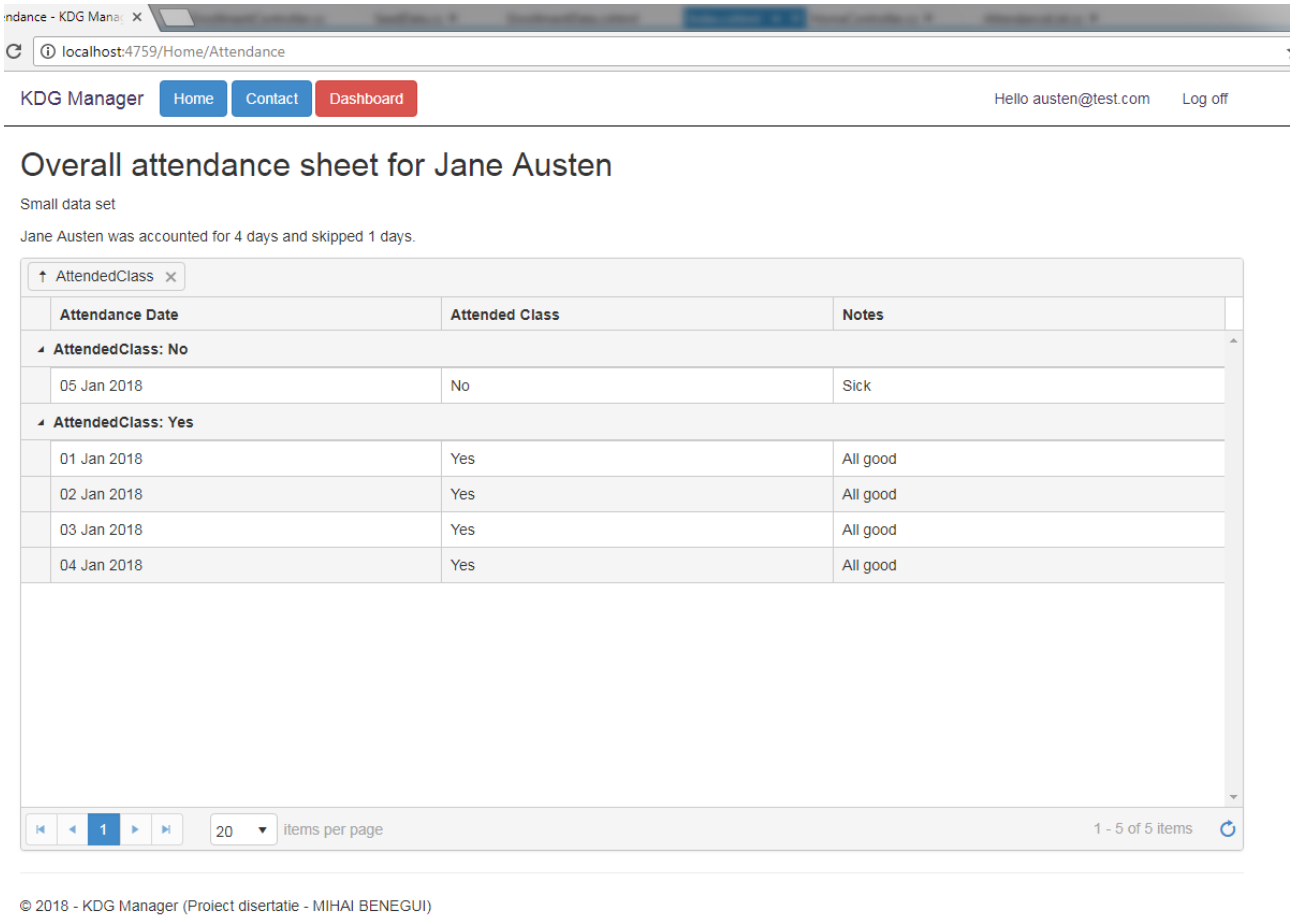
În *Controllers/HomeController* am adăugat:

```
public ActionResult Attendance()
        {
            if (User.IsInRole("Admin"))
            {
               var data = from a in db.DailyAttendances
                          join c in db.Children on a.CID equals c.CID
                          select new AttendanceList
                          {
                              FullName = c.FirstName + " " + c.LastName,
                              Date = a.Att_Date,
                              isPresent = a.isPresent == true ? "Yes" : "No",
                              Notes = a.Notes
                          };
               if (User.Identity.IsAuthenticated)
               {
                   return View(data.ToList());
               }

               else
               {
                   return new HttpUnauthorizedResult("Unauthorized");
               }
            }

            else
            {
               var data = from a in db.DailyAttendances
                          join c in db.Children on a.CID equals c.CID
                          where c.ContactEmail == User.Identity.Name
                          select new AttendanceList
                          {
                              FullName = c.FirstName + " " + c.LastName,
                              Date = a.Att_Date,
                              isPresent = a.isPresent == true ? "Yes" : "No",
                              Notes = a.Notes
                          };
               int DaysYes = data.Count(a => a.isPresent == "Yes");
               int DaysNo = data.Count(a => a.isPresent == "No");
               string FullName = data.Select(d => d.FullName).FirstOrDefault();

               ViewBag.FullName = FullName;
               ViewBag.DaysYes = DaysYes;
               ViewBag.DaysNo = DaysNo;

               if (User.Identity.IsAuthenticated)
               {
                   return View(data.ToList());
               }

               else
               {
                   return new HttpUnauthorizedResult("Unauthorized");
               }
            }
        }
```

View-ul corespunzător conține:

```
else if (ViewBag.displayMenu == "Parent")
{
    <div class="container">
        <div class="jumbotron">
            <h1>Welcome</h1>
            <p>
                This is your main dashboard.
                You can start exploring now..
            </p>
        </div>

    </div>
    <div class="row">
        <div class="col-md-4">
            <h2>Reporting Tools</h2>
            <p>
                Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis sapien
nunc, sollicitudin vehicula ipsum sit amet, euismod congue nisi.
            </p>
            <li>@Html.ActionLink("Enrollment Data", "EnrollmentData", "Home")</li>
            <li>@Html.ActionLink("Payments", "Payments", "Home")</li>
            <li>@Html.ActionLink("Attendance", "Attendance", "Home")</li>
        </div>
    </div>
```

Astfel, elementele de mai sus, vor genera următoarea pagină afișată utilizatorului de tip ”Parent”:



5. Precizarea testelor efectuate și a rezultatelor obținute

Testarea efectuată asupra aplicației a fost de tip manual, prin rularea în modul Debug, introducerea și validarea datelor și verificarea în baza de date a informațiilor introduse.

Pentru a avea date cu care să pot testa aplicația, am creeat o funcție SeedData care la momentul apelării, adaugă înregistrări în baza de date care pot fi folosite pentru a explora și testa aplicația.

```
public static void Seed()
        {
            Kdg_MVC.DataAccessLayer.AppContext ctx = new DataAccessLayer.AppContext();
            var chldren = new List<Children>
            {
                new Children
                {
                    FirstName = "Jane", LastName = "Austen", CNP = "2021001125899",
Address = "591 Myers Lane", City = "Roseville", MothersName = "Amanda", FathersName =
"Mario", ContactEmail = "austen@test.com"
                },
                new Children
                {
                    FirstName = "Mike", LastName = "Jensen", CNP = "1150727108484",
Address = "730 Pawnee Street", City = "Englewood", MothersName = "Lucy", FathersName =
"Elvis", ContactEmail = "nacho@icloud.com"
                },
```

Rezultatul acestei funcții a confirmat faptul că aplicația înregistrează corect informațiile și asigură transferul lor în și dinspre baza de date.

6. Posibilități de continuare și extindere a temei

În soluția prezentată, am abordat doar o mică parte din activitățile și datele care pot să fie gestionate prin intermediul unei aplicații Web ASP.NET și arhitectură MVC. Posibilitățile de continuare și extindere a temei pot să includă:

- Generarea facturilor lunare sub forma unui document electronic.
- Notificarea prin SMS / Email prin intermediul unui WebService a părinților despre diverse activități ale instituției.
- Consolidarea informațiilor prin adăugarea datelor legate de meniul zilnic.
- Accesarea CCTV-ului.
- Ruta autobuzului grădiniței.

Ținând cont de flexibilitatea unei aplicații ASP.NET MVC dar și de cerințele specifice și unice ale fiecărei grădinițe posibilitățile de extindere și continuare a temei pot fi nelimitate, ceea ce este un lucru bun pentru orice programator.

7. Bibliografie

- Getting Started with Entity Framework 6 Code First using MVC 5, https://docs.microsoft.com/en-us/aspnet/mvc/overview/getting-started/getting-started-with-ef-using-mvc/creating-an-entity-framework-data-model-for-an-asp-net-mvc-application
- Cutting Edge - Comparing Web Forms And ASP.NET MVC, https://msdn.microsoft.com/en-us/magazine/dd942833.aspx
- ASP.NET Identity: Using MySQL Storage with an EntityFramework MySQL Provider (C#), https://docs.microsoft.com/en-us/aspnet/identity/overview/getting-started/aspnet-identity-using-mysql-storage-with-an-entityframework-mysql-provider
- ASP.NET MVC 5 Security And Creating User Role, https://code.msdn.microsoft.com/ASPNET-MVC-5-Security-And-44cbdb97
- Kendo UI for jQuery Documentation and API Refference, https://docs.telerik.com/kendo-ui/controls/data-management/grid/overview
- ASP.NET MVC Overview, https://msdn.microsoft.com/en-us/library/dd381412(v=vs.108).aspx
- Introduction to Entity Framework, https://msdn.microsoft.com/en-us/library/aa937723(v=vs.113).aspx
- Curs C6 – WebServices, Prof.dr.ing. Vălean Honoriu