

UNIVERSITATEA DIN BUCUREȘTI
FACULTEA DE MATEMATICĂ ȘI INFORMATICĂ

LUCRARE DE LICENȚĂ

COORDONATOR ȘTIINȚIFIC

Lect. Dr. Mihail Cherciu

STUDENT

Alexandru-Răzvan Dumitru

BUCUREȘTI
2014

UNIVERSITATEA DIN BUCUREȘTI
FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ

LUCRARE DE LICENȚĂ

**Șabloane de proiectare a aplicațiilor web
dezvoltate în ASP.NET**

COORDONATOR ȘTIINȚIFIC

Lect. Dr. Mihail Cherciu

STUDENT

Alexandru-Răzvan Dumitru

BUCUREȘTI
2014

CUPRINS

I. NOȚIUNI INTRODUCTIVE	3
II. TEHNOLOGII & RESURSE	10
1. ASP.NET & MVC	10
2. Team Foundation Server	11
3. NuGet.....	14
4. Entity Framework	15
5. LINQ.....	16
6. Bootstrap.....	20
7. jQuery, jQuery Unobtrusive Validation & Select2	22
7.1. jQuery	22
7.2. jQuery Unobtrusive Validation.....	23
7.3. Select2.....	25
8. AdminLTE	27
9. BForms	30

III. DESCRIEREA APLICAȚIEI ȘI DEZVOLTARE ULTERIOARĂ.....	36
1. Inițializare.....	36
1.1. Structură.....	36
1.2. Bază de date.....	38
1.3. Multilingvism.....	41
1.4. Autentificare.....	45
2. Funcționalități.....	47
2.1. Modulul de management al profilului.....	47
2.2. Modulul de management al utilizatorilor.....	50
2.3. Modulul de management al programărilor.....	60
2.4. Modulul de management al rețetelor.....	62
2.5. Modulul de vizualizare a istoricului medical.....	63
3. Dezvoltare ulterioară.....	64
IV. ȘABLOANE DE PROIECTARE ASP.NET & JAVASCRIPT	65
1. RequireJS & șablonul de proiectare JavaScript – AMD.....	65
2. Șablonul de proiectare ASP.NET – Repository.....	68
3. Șablonul de proiectare ASP.NET – Unit of Work.....	70
4. Șablonul de proiectare ASP.NET – Dependency Injection.....	76
V. CONCLUZII.....	81
BIBLIOGRAFIE	82

I. NOȚIUNI INTRODUCTIVE

Societatea actuală se confruntă cu o multitudine de probleme care au la bază aceleași concepte, și deci pot fi rezolvate, din punct de vedere algoritmic, utilizând un anumit șablon, în funcție de contextul aferent.

Fiecare șablon propune o rezolvare general valabilă (*template*) pentru o problemă recurentă, fără a particulariza contextul acesteia. Având în vedere că soluția este propusă pentru un context general, ea poate fi refolosită. Rezolvarea este prezentată ca o suită de componente constitutive, legăturile dintre aceste componente, rolul fiecăruia și felul în care interacționează.

Printre avantajele utilizării șabloanelor de proiectare software identificăm impunerea unui anumit standard de calitate care presupune următoarele însușiri ale produsului final:

- Reutilizabilitate – arhitectul Christopher Alexander inițiatorul șabloanelor de proiectare, afirma în lucrarea sa *A Pattern Language: Towns, Buildings, Construction* (Alexander, 1977) că “*un șablon descrie o problema cu care ne confruntăm frecvent în mediul nostru de lucru, precum și o soluție general valabilă a problemei pe care o putem refolosi de un milion de ori, neutilizând-o în același mod de două ori*”
- Mentenabilitate – ușurința în modificarea produsului (pentru îndeplinirea anumitor specificații sau rezolvarea problemelor apărute în faza de producție)
- Modularitate

Utilizarea șabloanelor de proiectare crește calitatea produsului final și scad efortul programatorului în faza de producție – mentenanță, scopul acestora fiind de a ajuta la dezvoltarea de cod flexibil, ușor de modificat, care să permită adăugarea de noi funcționalități fără modificări majore ale implementării curente.

Dacă acceptăm împărțirea clasică (trivială) a etapelor prin care trece un produs software:

- Analiză – identificarea problemei ce trebuie rezolvată
- Design – modelarea datelor și alegerea șabloanelor
- Implementare
- Test – rezolvarea problemelor apărute
- Documentație – descrierea specificațiilor produsului software către utilizatorul final

vom observa că șabloanele de proiectare au cel mai mare impact în faza de design a produsului și constituie un *guideline* pentru faza de implementare.

Acest *guideline* prezent în faza de implementare este unul foarte solid deoarece șabloanele de proiectare software sunt construite pe baza experienței de lungă durată a programatorilor cu limbajele orientate obiect și cu principiile de design ale acestora.

Popularizarea șabloanelor de proiectare a fost iminentă odată cu publicarea în anul 1994 a lucrării *Design patterns - Elements of Reusable Object Oriented Software* scrisă de E. Gamma, R. Helm, R. Johnson, J. Vlissides care mai apoi au primit titulatura de *Gang of Four (GoF)*, în semn de respect pentru aportul adus domeniului informaticii.

Cei patru autori au definit șabloanele de proiectare astfel: “*Șabloanele de proiectare descriu obiecte și clase care interacționează și care sunt particularizate pentru a rezolva o problemă generală de design într-un context particular*” (Gang of Four, 1994).

În această lucrare au fost documentate și catalogate cele mai folosite șabloane de proiectare, prezente în industria software a vremii respective, fiecare dintre ele fiind prezentat într-un mod logic, intuitiv, și mai apoi exemplificat în limbaje precum C++ și *Smalltalk*.

Cele 23 de șabloane de proiectare prezentate în lucrarea *Design patterns - Elements of Reusable Object Oriented Software* (Gang of Four, 1994) au fost împărțite, în funcție de scopul acestora, în trei mari grupuri:

- *Creational patterns* – utilizate pentru abstractizarea procesului de creare și inițializare a obiectelor
- *Structural Patterns* – utilizate pentru gruparea obiectelor în structuri complexe
- *Behavioral Patterns* – utilizate pentru definirea interacțiunii dintre obiecte; schițează responsabilitățile fiecăruia

Robert C. Martin prezintă în lucrarea sa *Agile Principles, Patterns, and Practices in C#* (Martin, 2002) o colecție a celor mai bune practici folosite în *design-ul* orientat-obiect, denumită în literatura de specialitate *S.O.L.I.D. Design Principles* conform inițialelor următoarelor cinci principii:

Single Responsibility Principle (SRP) - fiecare clasă (obiect) trebuie să aibă o singură responsabilitate pe care să o încapsuleze complet. Aderând la acest principiu, evităm problema claselor (obiectelor) monolitice și scădem semnificativ efortul programatorului pentru eventualele modificări ale produsului software (în faza de implementare, testare sau mentenanță).

Open-Closed Principle (OCP) – acest termen a fost definit pentru prima oară de Bertrand Meyer în lucrarea sa *Object-Oriented Software Construction* (Meyer, 1988) astfel: “*entitățile software (clasele, modulele, funcțiile, etc.) trebuie să fie deschise pentru extinderi ulterioare, dar închise pentru modificări ulterioare*”. Acest principiu se poate exemplifica în două moduri diferite, ambele presupunând utilizarea mecanismului de moștenire. Primul mod presupune derivarea unei clase de bază ori de câte ori dorim să adăugăm funcționalități noi. Astfel, în clasa derivată, vom avea acces la metodele și membrii clasei de bază, fără a modifica produsul în prealabil. Al doilea mod presupune utilizarea interfețelor abstracte și mecanismului de polimorfism pentru implementarea noilor funcționalități. Aderând la acest principiu evităm problemele care se pot genera în cascadă odată cu modificarea claselor existente și, implicit, a claselor care le moștenesc.

Liskov Substitution Principle (LSP) – descrie interoperabilitatea dintre obiectele care aparțin aceluiași arbore de derivare. Acest principiu susține că obiectele de tip subclasă (obiectele care aparțin unei clase derivate) trebuie să poată înlocui obiectele care aparțin clasei de bază. Derick Bailey descrie acest principiu astfel: “*Dacă arată ca o rață, măcăie exact ca o rață, dar are nevoie de baterii – probabil s-a folosit abstractizarea greșită*” (Bailey, 2009). Un exemplu foarte simplu de încălcare a acestui principiu este prezentat în lucrarea *Head First Object-Oriented Analysis and Design* (Brett McLaughlin, 2006). Este descris un scenariu în care un programator face parte din echipa de dezvoltare a unui *framework* pentru jocuri de strategie.

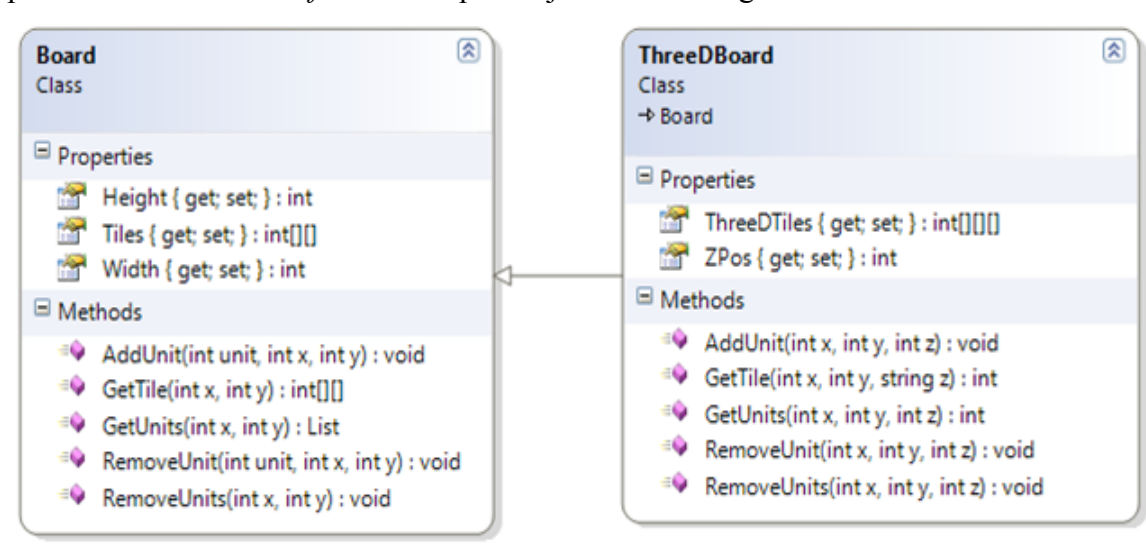


Figura I.1. Exemplu principiul substituției a lui Liskov

Clasa *Board* definită anterior poate controla cu succes unitățile terestre în cadrul unei planșe, deoarece cuprinde un *array (Tiles)* care suportă cele două dimensiuni descrise momentan, precum și metodele caracteristice *framework-ului* pe care programatorul dorește să-l definească. Scenariul continuă cu o modificare a specificațiilor. *Framework-ul* trebuie să suporte planșe de joc 3D pentru angrenarea unităților aeriene.

Pentru început este adăugată o clasă *ThreeDBoard* care extinde clasa de baza *Board* folosită până acum pentru controlul unităților terestre. La prima vedere este o decizie inspirată, dar privind amănunțit structura celor două clase observăm ca odată cu introducerea unei noi dimensiuni, metodele definite în clasa de bază devin inutilizabile pentru clasa derivată, programatorul fiind nevoit să rescrie fiecare dintre aceste metode pentru a primi un parametru în plus, utilizat pentru cea de-a treia dimensiune. Această abordare încalcă principiul menționat. Marco Cecconi, *Software Engineer* la *Stack Exchange* afirma că “*Morala acestei povești este următoarea: întotdeauna ar trebui să modelați clasele valorificând acțiunile care vor fi întreprinse cu ajutorul acestora și făcând abstracție de proprietăți; întotdeauna modelați baza de date valorificând proprietățile.*” (Sklivvz, 2012)

Interface Segregation Principle (ISP) – “*nici un client nu ar trebui să fie forțat să depindă de metode pe care nu le utilizează*” (Martin, 2002) . Acest principiu se referă de fapt la împărțirea interfețelor voluminoase în interfețe specifice care îndeplinesc un singur rol. Astfel, vor trebui implementate doar interfețele care sunt utilizate în totalitate de către clasa curentă. Aceste modificări au ca efect imediat decuplarea claselor (modulelor), precum și creșterea coeziunii produsului *software*.

Dependency Inversion Principle (DIP) – toate clasele (modulele) unui produs software ar trebui să depindă (prin mecanismul de moștenire de exemplu) de interfețe sau de clase abstracte, nu de implementări concrete ale acestora. Promovăm astfel un sistem de dezvoltare de tip *top-down* pornind de la cel mai cuprinzător modul către cel mai facil, asigurând decuplarea acestora.

```
namespace LRazvanDumitru.Service {  
    public interface IBooksRepository {  
  
        IEnumerable<Book> GetEnabledBooksForCategory (int  
        catId);  
    } }  

```


Aplicând principiul descris în paragraful anterior avem o interfață publică *IBooksRepository* care va fi implementată de *repository-ul* următor:

```
public class BooksRepository : IBooksRepository {

    public IEnumerable<Book> GetEnabledBooksForCategory (int
    catId) {

        var books = new List<Book>();

        // utilizând Entity Framework (Documentation) și LINQ
        // (LINQ, 2007) pentru interogare

        books = db.Books.Where(x=>x.Enabled).ToList();
        return books;

    }

}
```

În *repository* populăm lista de cărți cu înregistrările valabile din baza de date. *Book* este modelul generat de *Entity Framework*. Am utilizat LINQ (LINQ, 2007) pentru interogarea bazei de date.

```
public class BookService{

    private IBookRepository _bookRepo;

    public BookService(){
        _bookRepo = new BookRepository();
    }

    public IEnumerable<Book> GetEnabledBooksForCategory (int
    catId){

        // ar trebui să returnăm datele pe care ni le
        generează metoda GetAllBooksForCategory din BookRepository

    }

}
```

Putem observa că serviciul pe care l-am declarat pentru entitatea *Book* este responsabil pentru instanțierea *repository-ului*. Instanțierea se realizează în cadrul constructorului.

În această situație putem aplica principiul injectării dependențelor (*Dependency Injection Principle*) pentru a deresponsabiliza serviciul.

Dependency Injection Principle se poate aplica în trei moduri: constructor, metodă sau proprietate.

```

public class BookService {

    private IBookRepository _bookRepo;

    public BookService (IBookRepository bookRepository) {
        _bookRepo = bookRepository;
    }

    public IEnumerable<Book> GetEnabledBooksForCategory (int
catId) {

        return _bookRepo.GetAllBooksForCategory(catId);
    }

}

```

Pentru exemplul nostru, cea mai naturală abordare a fost injectarea dependenței în cadrul constructorului.

Principiul descris anterior, împreună cu toate dependențele și/sau implementările sale, ajută la scalabilitatea produsului software, crește mentenabilitatea și este un șablon de proiectare de bază pentru implementarea testării automate.

Toate cele 23 de șabloane de proiectare menționate în *Design patterns - Elements of Reusable Object Oriented Software* (Gang of Four, 1994) aderă într-o formă sau alta la aceste principii.

Clasificarea trivială a șabloanelor, în funcție de nivelul la care apar este următoarea:

- *Idioms (low-level design patterns)* - sunt șabloanele dependente de anumite tehnologii (depind de anumite limbaje de programare)
- *Design patterns* – sunt soluții independente de tehnologia folosită de programator (independente de limbajul de programare)
- *Architectural patterns* – sunt șabloane care descriu arhitectura întregului produs software

De exemplu, MVC (*Model-View-Controller*) este un șablon care a trecut de-a lungul timpului prin toate cele trei categorii descrise anterior, fiind la început (1970) prezentat ca un *idiom* dependent de limbajul *Smalltalk*, particularizat și instanțiat *by-default* ca un *design pattern* pentru *framework-ul* Ruby on Rails, ajungând să fie considerat începând cu anul 2009 *architectural pattern* pentru *framework-ul* ASP.NET MVC dezvoltat de către Microsoft.

Scopul lucrării de față este prezentarea generală din punct de vedere teoretic a șabloanelor utilizate în proiectarea și implementarea produselor software dezvoltate în ASP.NET MVC. Fiecare șablon de proiectare prezentat va fi însoțit de un exemplu pertinent, specific unei situații reale.

Produsul software atașat lucrării de față este un produs web cu scop exemplificativ, dezvoltat în întregime folosind tehnologii Microsoft. Șablonul arhitectural MVC (*Model-View-Controller*) este setat *by-default* în *framework-ul* ASP.NET MVC, utilizat pentru dezvoltarea produsului. Am utilizat o bază de date MSSQL, accesul se face prin *Windows Authentication*, limbajul de interogare fiind LINQ (LINQ, 2007). Pentru versionarea produsului software am folosit *Team Foundation Server* (versiunea 2013, via *Team Foundation Version Control*), serverul fiind <https://razvandumitru.visualstudio.com/>. Ca mediu de dezvoltare, am utilizat *Visual Studio 2013*, interfața pentru baza de date fiind *SQL Server Management Studio*.

În partea introductivă am prezentat cele cinci principii care stau la baza proiectării unui produs software însoțite de un exemplu practic sau de scenarii în care pot fi încălcate cu ușurință.

Următoarele capitole abordează o viziune practică, descriind tehnologiile folosite, structura și funcționalitățile produsului software atașat, precum și implementări ale următoarelor șabloane:

- MVC – ca șablon arhitectural – definit teoretic în cadrul subcapitolului II.1 exemplificarea practică fiind prezentată în cadrul structurii produsului (subcapitolul III.1.1)
- *Dependency Injection & Unit of Work* – două șabloane de proiectare care funcționează împreună (subcapitolele IV.3 & IV.4)
- Șablonul de proiectare AMD (*Asynchronous Module Definition*) suportat de *RequireJS*
- *Repository* – prezentat în cadrul subcapitolului IV.2

II. TEHNOLOGII & RESURSE

1. ASP.NET & MVC

ASP.NET este un *framework server-side* conceput pentru dezvoltarea paginilor web dinamice, suportând trei modele arhitecturale: *Web Forms*, *Web Pages* și *Model-View-Controller*, apărut ca o alternativă pentru standardul bazat pe formulare (*Web Forms*).

Design-ul acestui *framework* permite dezvoltarea de aplicații web sau de servicii web. Dezvoltarea de servicii web a fost facilitată odată cu apariția suitei de extensii ASP.NET AJAX, care au introdus caracteristici precum serializare JSON și transmiterea de cereri către serviciul web cu ajutorul *JavaScript*, sau utilizând librării precum *jQuery*.

Model-View-Controller (MVC) este considerat în momentul redactării acestei lucrări un șablon arhitectural. Din punctul meu de vedere, un șablon arhitectural ar trebui să aibă o structură abstractă care să permită dezvoltatorilor să lucreze în paralel, și să permită utilizarea mai multor tehnologii pentru dezvoltare. Șablonul MVC îndeplinește cu succes cerințele definite anterior, un exemplu fiind un sistem software a cărui componente de tip *View* sunt dezvoltate în HTML sau cu ajutorul *template-urilor JavaScript* (asemănătoare cu *Mustache* (Mustache, 2009) – un *parser* pentru ASP.NET regăsindu-se în pachetul *ASP Xtreme Evolution* (negaozen, 2013)), componentele de tip *Controller* sunt dezvoltate în ASP.NET, iar componentele de tip *Model* sunt dezvoltate în Java accesând un serviciu de sine stătător, care întoarce obiecte de tip JSON.

Principalul său scop este partiționarea responsabilităților într-un produs software. Implementarea cu succes a acestui șablon arhitectural ar trebui să aibă ca urmări decuplarea interfeței de logică operațională și creșterea mentenabilității produsului.

Acest șablon arhitectural împarte aplicația în trei componente (*layers*) denumite intuitiv *Model*, *View* și *Controller* în funcție de responsabilitatea principală pe care o definesc, oferind astfel mai mult control asupra implementării.

Cele trei componente ale acestui șablon arhitectural sunt structurate ca entități separate, dar comunică între ele cu ajutorul *Controller-ului*.

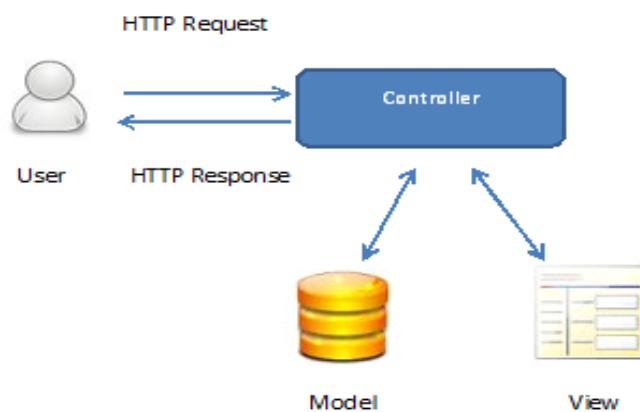


Figura II.1.1. Schemă conceptuală MVC (Bhatia, 2013)

În definiția clasică a acestei structuri, componenta *Model* este responsabilă pentru validitatea datelor și persistența acestora în cadrul aplicației.

Componenta de tip *View* transpune datele într-o formă accesibilă utilizatorului, permițând totodată o interacțiune facilă între utilizator și sistem. Pot exista mai multe componente de tip *View* pentru aceeași componentă de tip *Model* (transpunerea datelor într-o altă manieră pentru scopuri diferite).

Componenta de tip *Controller* reacționează în concordanță cu cerințele utilizatorului, controlând datele (de exemplu filtrarea anumitor înregistrări după țara de origine – România) și generând un răspuns sub forma unui *View*, respectând modificările aduse de utilizator (de exemplu un tabel cu înregistrările care au țara de origine – România).

Produsul software asociat acestei lucrări folosește sintaxa *Razor* pentru definirea componentelor de tip *view*. O descriere amănunțită a acestei tehnologii se găsește în cadrul articolului “*Introducere în programarea ASP.NET utilizând Razor*” realizat de Tom FitzMacken (FitzMacken, 2014).

2. Team Foundation Server

Team Foundation Server este produsul Microsoft pentru controlul codului sursă (suportă dezvoltarea în paralel), raportare, urmărire și testare. Team Foundation Server, în continuare denumit TFS, poate fi utilizat ca o soluție de *back-end* pentru numeroase medii de dezvoltare, dar a fost produs pentru a fi utilizat împreună cu Microsoft Visual Studio, pentru sistemul de operare Windows și împreună cu Eclipse, atât pentru platforma Windows cât și pentru alte platforme care suportă acest mediu de dezvoltare.

TFS suportă două tipuri diferite de *repository* pentru controlul codului sursă: primul fiind cel *build-in* care se numește Team Foundation Version Control (TFVC), al doilea fiind *Git*, începând cu versiunea lansată în anul 2013.

Produsul software atașat acestei lucrări a fost dezvoltat utilizând *Team Foundation Version Control* ca *repository*.

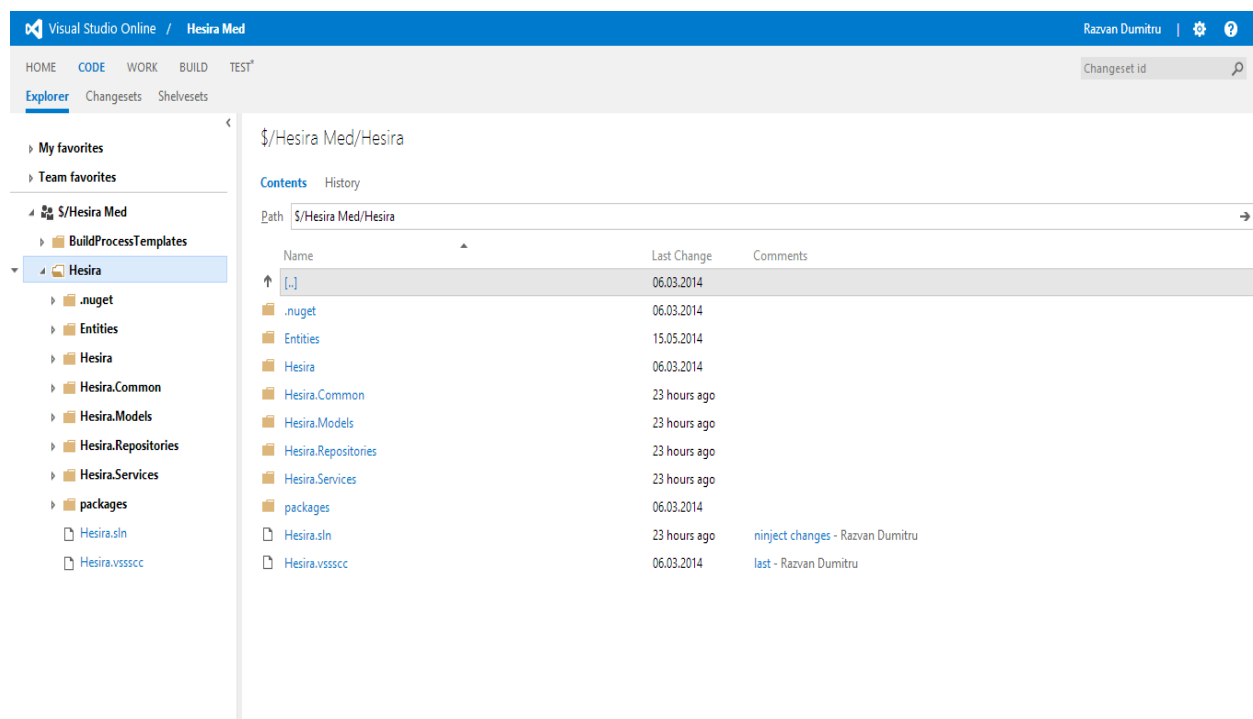
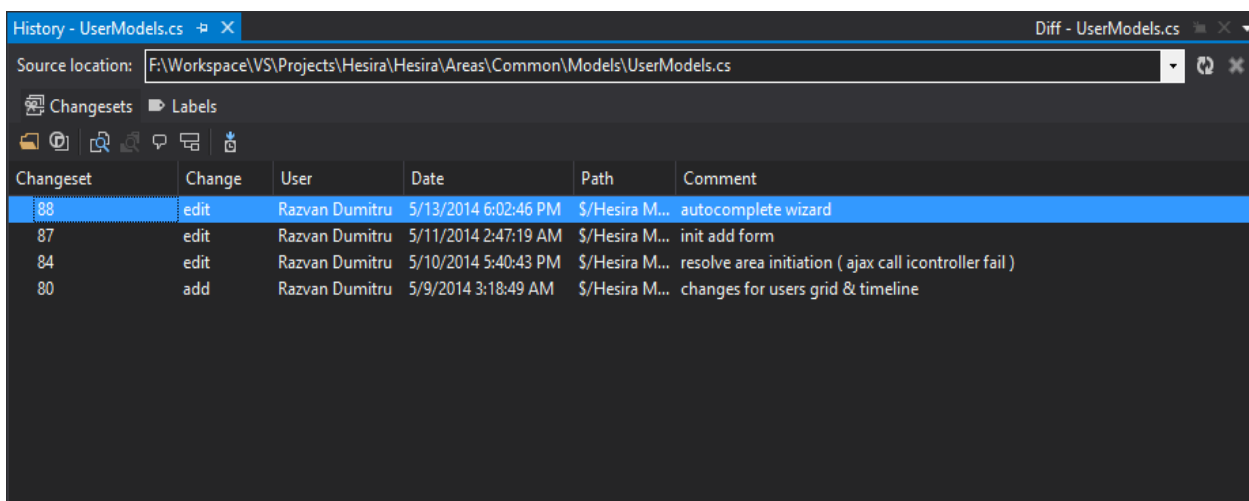


Figura II.2.1. Structura produsului Hesira (echipa Hesira Med)

În figura anterioară putem observa stadiul produsului software atașat în momentul redactării (n.r. 17.05.2014). Ultimele modificări sunt reprezentate de adăugarea unui modul care folosește injectorul de dependențe *Ninject* (Kohari, 2007).

TFVC este util pentru dezvoltarea în echipă a produselor software. Acest produs pentru controlul codului sursă suportă adăugarea modificărilor proprii pe server sub forma unui *changeset*, utilizând sistemul de *check-in*, sau adăugarea acestora într-un *shelveset*, pentru modificările care nu ar trebui să se regăsească pentru moment în *build-ul* produsului final (presupunând ca la fiecare *check-in* există un sistem de integrare continuă precum *TeamCity* (JetBrains, 2006)). Sistemul de *shelveset* este foarte util deoarece există posibilitatea testării codului modificat în cadrul unui *build* privat, sau prezentarea acestui cod unui alt participant la dezvoltare sau echipei de supervizori înainte să ajungă pe server.

Fiecare *check-in* este însoțit de modificările curente (*pending changes*), de un mesaj atașat acestor modificări și de eventualele explicații care pot fi scrise într-o notificare (*check-in note*), concretizându-se pe server într-un singur *changeset*, care reprezintă totalitatea modificărilor aduse proiectului. Aceste *changeset-uri* sunt reprezentative, fiind de fapt pseudo-versiuni ale produsului, deoarece sunt salvate în istoricul fiecărui fișier modificat în parte, existând posibilitatea de *rollback* la pseudo-versiunea precedentă. Un astfel de istoric este cel prezentat în figura II.2.2 unde putem observa *changeset-urile* fișierului *UserModels.cs*, în care sunt definite clasele care modelează entitatea utilizator.



Changeset	Change	User	Date	Path	Comment
88	edit	Razvan Dumitru	5/13/2014 6:02:46 PM	\$/Hesira M...	autocomplete wizard
87	edit	Razvan Dumitru	5/11/2014 2:47:19 AM	\$/Hesira M...	init add form
84	edit	Razvan Dumitru	5/10/2014 5:40:43 PM	\$/Hesira M...	resolve area initiation (ajax call iconcontroller fail)
80	add	Razvan Dumitru	5/9/2014 3:18:49 AM	\$/Hesira M...	changes for users grid & timeline

Figura II.2.2. Exemplu de istoric pentru un fișier

O altă trăsătură foarte importantă a acestui produs pentru controlul codului sursă este posibilitatea comparării pseudo-versiunilor între ele sau a fișierelor modificate cu versiunile anterioare aflate pe server. O comparație a pseudo-versiunilor este cea ilustrată în figura următoare.

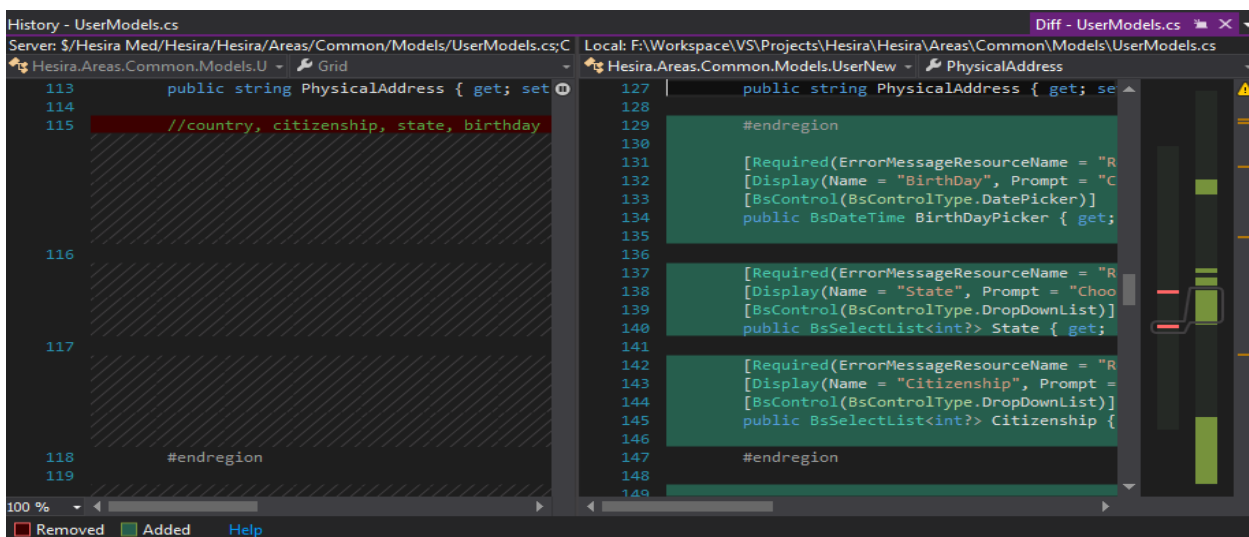


Figura II.2.3. Exemplu de comparație între pseudo-versiuni

3. NuGet

NuGet este un manager pentru pachete externe dezvoltat pentru platforma Microsoft, incluzând limbajul .NET. Principala sursă de pachete este *The NuGet Gallery* (nuget.org). În industria software se utilizează acest manager de pachete pentru componentele comune la nivel de organizație (librării interne, clase de tip ajutor pentru generarea HTML).

Pachetelor *NuGet* li se pot asocia alte pachete ca dependențe. Presupunând ca dorim să adăugăm pachetul X care are ca dependențe Y și Z, *NuGet* va verifica dacă pachetele Y și Z sunt prezente în proiectul curent. Dacă aceste pachete nu există, le va adăuga automat.

Adăugarea unui pachet *NuGet* poate avea ca urmări modificarea soluției în concordanță cu necesitățile pachetului (de exemplu modificarea referințelor).

Ștergerea unui pachet *NuGet* are ca urmare o acțiune de tipul *rollback*, eliminând pachetul precum și dependențele acestuia dacă nu mai sunt utilizate.

Aplicația atașată acestei lucrări folosește acest manager de pachete, cu opțiunea de restaurare a pachetelor *NuGet* în momentul construirii aplicației (în momentul *build-ului*) pentru a nu le integra în sistemul de control al codului sursă. (NuGet, 2013)

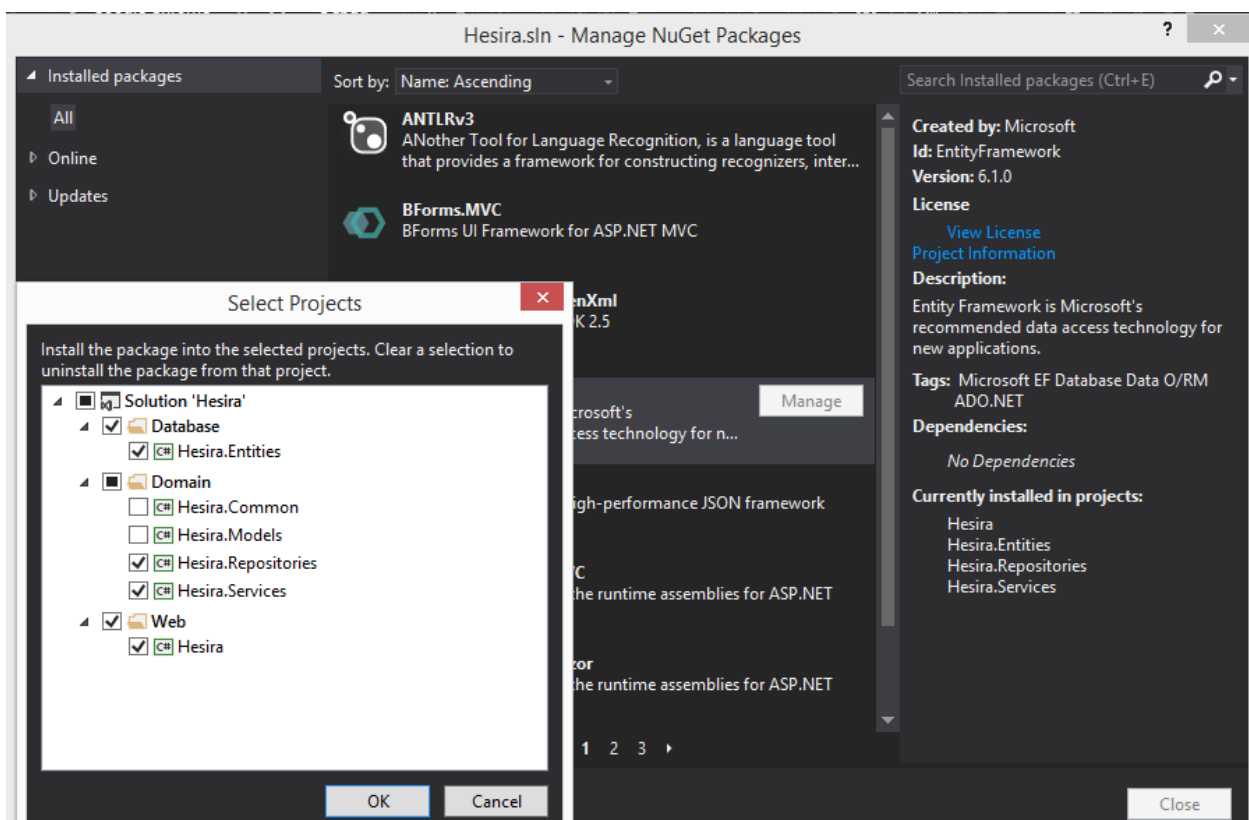


Figura II.3.1. Managementul pachetelor NuGet

În figura II.3.1 putem observa pachetul *Entity Framework* adăugat via *NuGet* pentru o parte din proiectele prezente în cadrul produsului software atașat. *NuGet* nu suportă restaurarea pachetelor pentru proiectele care nu sunt *web applications*, asta fiind una din problemele cu care ne putem confrunta în dezvoltarea produselor.

4. Entity Framework

Entity Framework (EF) este un *framework* ORM(*Object-Relational Mapper*) care facilitează accesul la date, eliminând astfel o mare parte din timpul asociat dezvoltării acestui *layer*.

Acest *framework* se poate utiliza în două moduri total diferite:

- *Database First* – EF va genera modelele pe baza tabelor existente în baza de date
- *Model First* – avem posibilitatea de a crea un model nou folosind *designer-ul* asociat acestui *framework*. Utilizând acest model creat de dezvoltator putem genera baza de date

Având în vedere utilitatea și popularitatea acestui produs, producătorii de la Microsoft au făcut public codul sursă pentru a oferi comunității șansa de a participa la mentenanța și la dezvoltarea acestui produs printr-un sistem de tip *feedback* sau testarea variantelor *nightly*.

Produsul software atașat lucrării de față a fost dezvoltat folosind abordarea *Database First*. Baza de date este construită în Microsoft SQL Server 2012.

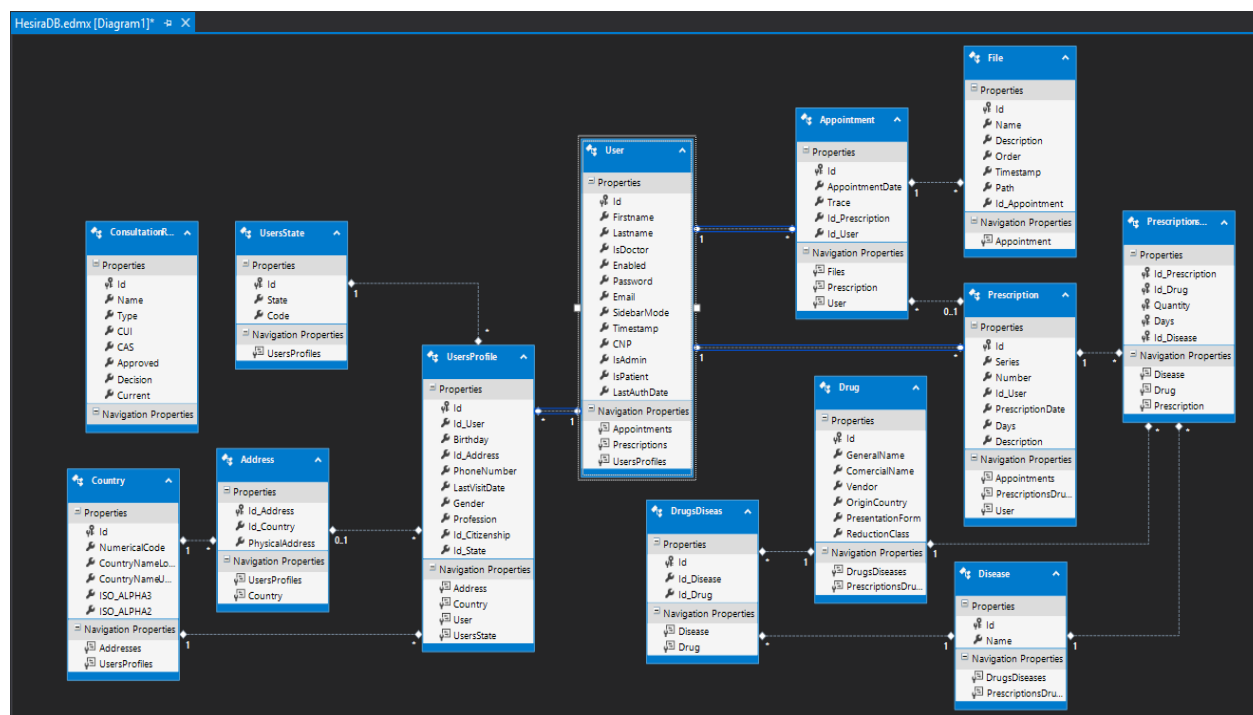


Figura II.4.1. Entity Framework – fișier EDMX generat - HesiraDB

Fișierele de tip EDMX sunt de fapt fișiere XML care cuprind modelul EF asociat compus din model conceptual, model depozitat (*storage model*), precum și informații necesare convertirii între modelul conceptual și cel depozitat. În afară de acest model EF, fișierele de tip EDMX conțin informații care sunt utilizate de *designer-ul* acestui *framework* pentru reprezentarea schematică a modelului, un exemplu fiind prezentat în figura anterioară.

5. LINQ

LINQ (*Microsoft Language Integrated Query*) poate fi considerat un strat (*layer*) care abstractizează accesul și prelucrarea datelor, bazându-se pe un model consistent care e independent de proveniența acestora. LINQ poate fi extins pentru a oferi sprijin oricărui tip de stocare de date.

Datorită structurii puternice pe baza căreia a fost dezvoltat, LINQ oferă o metodă de interogare și procesare a datelor asemănătoare ca sintaxa oricărui limbaj de tip SQL, având de asemenea suport pentru expresii lambda (funcții anonime care nu sunt corelate cu un anumit identificator în momentul definirii). O sursă de inspirație pentru dezvoltarea continuă a acestei componente este limbajul pur funcțional Haskell (Jones & Augustsson, 2010).

În cadrul unui produs datele pot avea surse total diferite. De exemplu, presupunem un scenariu în care avem un document XML care trebuie parcurs pentru extragerea denumirii, prețurilor și producătorilor unor medicamente, avem un API central dezvoltat în Java care ne returnează detalii despre pacienți sub forma unui JSON, o suită de obiecte enumerabile pentru a opera cu aceste entități sau pentru a popula stratul (*layer-ul*) de prezentare și documente în format EXCEL utilizate în modulul de raportare. În acest context, LINQ este absolut esențial deoarece putem interoga și prelucra datele provenite din toate sursele enumerate anterior (pentru XML putem folosi componenta *LINQ to XML*, pentru JSON putem utiliza pachetul *NuGet Newtonsoft.Json* în care este definit un API-ul *LINQ to JSON*, care favorizează utilizarea obiectelor de tip JSON).

ADO.NET este o suită de componente dezvoltată de Microsoft pentru accesul la date și la serviciile acestora având la bază fișiere XML și seturi de date decuplate. Este o componentă esențială, fiind utilizată în practică pentru a interoga și a prelucra datele aflate într-o bază de date relațională, având în același timp suport pentru baze de date precum *MariaDB* care utilizează comenzi și interfețe de tip *NoSQL*.

Componenta *LINQ to ADO.NET* permite interogarea și prelucrarea datelor aflate într-un obiect ADO.NET care poate fi enumerat utilizând acest limbaj de interogare.

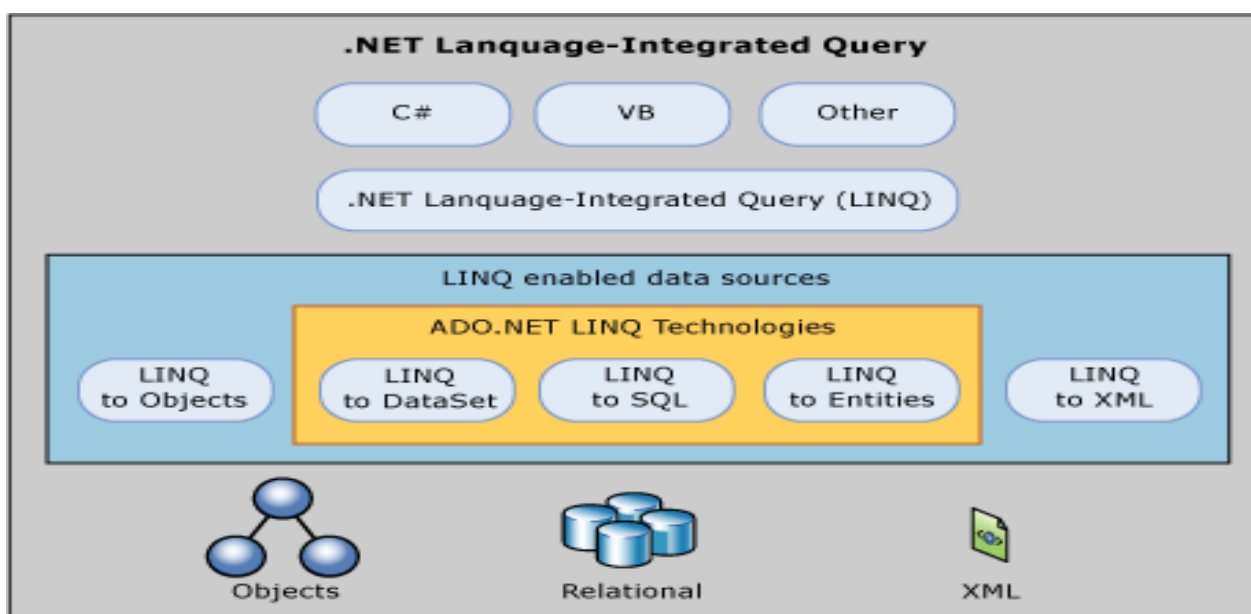


Figura II.5.1. Tehnologii ADO.NET LINQ (Microsoft Corporation, 2012)

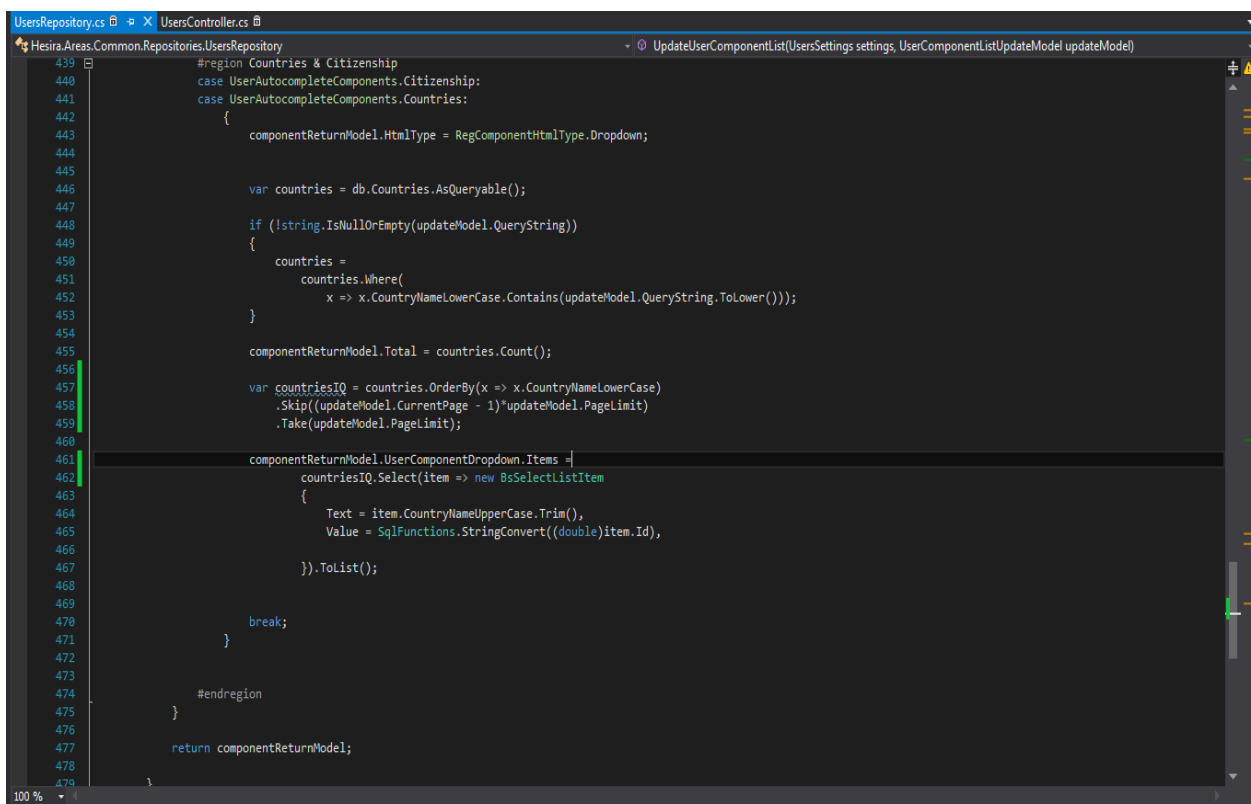
După cum putem observa în figura anterioară *LINQ to ADO.NET* cuprinde mai multe tehnologii specifice fiecărui strat (*layer*) de persistență:

- *LINQ to SQL* – oferă suport pentru convertirea dintre tipurile definite de programator în *framework-ul* .NET și schema tabeli fizice din SQL Server
- *LINQ to Entities* – este un ORM (*Object Relational Mapping*) care folosește un model conceptual de tip EDM(*Entity Data Model*) pentru persistența datelor, deresponsabilizând astfel baza de date. Utilizând această componentă introducem un strat (*layer*) abstract, independent de stratul fizic de persistență a datelor
- *LINQ to DataSet* – oferă suport pentru interogarea unui obiect de tip *DataSet* folosind LINQ

Primele două tehnologii sunt asemănătoare, accesând datele stocate într-o bază de date relațională și utilizând entități separate care sunt reprezentate în memorie ca date externe. Diferența principală dintre cele două tehnologii este faptul că *LINQ to SQL* este corelat direct cu structura bazei de date, în timp ce *LINQ to Entities* folosește un model conceptual abstract (entități operaționale) care este decuplat de structura fizică a bazei de date.

Corelația dintre componenta LINQ și arhitectura proiectului în care este folosită poate fi cercetată amănunțit în *Programming Microsoft® LINQ in Microsoft .NET Framework 4* (Pialorsi & Russo, 2010), lucrare ce constituie o sursă de informare pentru acest subcapitol.

Produsul software atașat acestei lucrări folosește *LINQ to Entities*, modelând datele folosind *Entity Framework* în varianta clasică (*database-first*), pornind de la o bază de date existentă.



```
439 #region Countries & Citizenship
440 case UserAutocompleteComponents.Citizenship:
441 case UserAutocompleteComponents.Countries:
442 {
443     componentReturnModel.HtmlType = RegComponentHtmlType.Dropdown;
444
445     var countries = db.Countries.AsQueryable();
446
447     if (!string.IsNullOrEmpty(updateModel.QueryString))
448     {
449         countries =
450             countries.Where(
451                 x => x.CountryNameLowerCase.Contains(updateModel.QueryString.ToLower()));
452     }
453
454     componentReturnModel.Total = countries.Count();
455
456     var countriesIQ = countries.OrderBy(x => x.CountryNameLowerCase)
457         .Skip((updateModel.CurrentPage - 1)*updateModel.Pagelimit)
458         .Take(updateModel.Pagelimit);
459
460     componentReturnModel.UserComponentDropdown.Items =
461         countriesIQ.Select(item => new BsSelectListItem
462             {
463                 Text = item.CountryNameUpperCase.Trim(),
464                 Value = SqlFunctions.StringConvert((double)item.Id),
465             }).ToList();
466
467     break;
468 }
469
470 #endregion
471
472 return componentReturnModel;
473 }
```

Figura II.5.2. Exemplu interogare folosind LINQ - *LINQ to Entities*

Entity Framework oferă suport nativ pentru încărcarea târzie (*lazy loading*) a entităților, începând cu versiunea .NET Framework 4.

În figura anterioară putem observa un exemplu de interogare a bazei de date folosind tehnologia *LINQ to Entities*. Având în vedere numărul mare de înregistrări ale tabelului *Countries* (239 de înregistrări), popularea și utilizarea unui input de tip *dropdown* care să aibă 239 de opțiuni este o abordare greșită (conceptual, practic și din punct de vedere al interfeței). Metoda prezentată în figura anterioară este utilizată pentru filtrarea *dropdown-ului* asociat țării de origine a unui utilizator (*Select2 dropdown* (Vaynberg, 2012) având funcția de *query* definită pentru a încărca datele de pe server utilizând o cerere de tip AJAX). Opțiunile din *dropdown* se vor modifica atunci când utilizatorul caută anumite caractere, fiind returnate doar țările a căror denumire conține aceste caractere sau când utilizatorul navighează (*scroll*) pentru a încărca următoarele înregistrări, dacă acestea există. Vom reține de fiecare dată numărul maxim de înregistrări (*updateModel.Total*). Când utilizatorul execută una dintre acțiunile definite, se trimite o cerere către server care conține cuvântul introdus (*updateModel.QueryString*) și pagina curentă (*updateModel.CurrentPage*). Popularea acestui *dropdown* se face utilizând un sistem de paginare în funcție de cuvântul căutat de utilizator.

În cadrul interogării LINQ prezentate am utilizat metoda *StringConvert* aparținând clasei *SQLFunctions*. *Entity Framework* pune la dispoziția dezvoltatorilor două clase: *SQLFunctions* și *EntityFunctions* a căror metode pot fi invocate în cadrul unei interogări LINQ având certitudinea că vor fi convertite.

Metodele care aparțin clasei *SQLFunctions* sunt denumite funcții ale bazei de date (*database functions*), iar cele care aparțin clasei *EntityFunctions* sunt denumite funcții canonice (*canonical functions*). Funcțiile bazei de date sunt specifice Microsoft SQL Server.

Codul SQL generat pentru interogarea de tipul *LINQ to Entities* prezentată în figura II.5.2 este următorul (am căutat caracterele 'alb'):

```
{SELECT TOP (10)
[Project1].[Id] AS [Id],
[Project1].[NumericalCode] AS [NumericalCode],
[Project1].[CountryNameLowerCase] AS [CountryNameLowerCase],
[Project1].[CountryNameUpperCase] AS [CountryNameUpperCase],
[Project1].[ISO_ALPHA3] AS [ISO_ALPHA3],
[Project1].[ISO_ALPHA2] AS [ISO_ALPHA2]
FROM ( SELECT [Project1].[Id] AS [Id],
[Project1].[NumericalCode] AS [NumericalCode],
[Project1].[CountryNameLowerCase] AS [CountryNameLowerCase],
[Project1].[CountryNameUpperCase] AS [CountryNameUpperCase],
[Project1].[ISO_ALPHA3] AS [ISO_ALPHA3],
[Project1].[ISO_ALPHA2] AS [ISO_ALPHA2],
row_number() OVER (
ORDER BY [Project1].[CountryNameLowerCase] ASC)
AS[row_number]
FROM ( SELECT [Extent1].[Id] AS [Id],
[Extent1].[NumericalCode] AS [NumericalCode],
[Extent1].[CountryNameLowerCase] AS [CountryNameLowerCase],
[Extent1].[CountryNameUpperCase] AS [CountryNameUpperCase],
[Extent1].[ISO_ALPHA3] AS [ISO_ALPHA3],
[Extent1].[ISO_ALPHA2] AS [ISO_ALPHA2]
FROM [dbo].[Countries] AS [Extent1]
WHERE (CAST(CHARINDEX(LOWER(@p__linq__0),
[Extent1].[CountryNameLowerCase]) AS int)) > 0)
AS [Project1])
AS [Project1] WHERE [Project1].[row_number] > 0
ORDER BY [Project1].[CountryNameLowerCase] ASC}

@p__linq__0=N'alb'
```

Serverul a răspuns populând *dropdown-ul* cu înregistrările Albania și Svalbard & Jan Mayen, după cum putem observa în figura următoare.

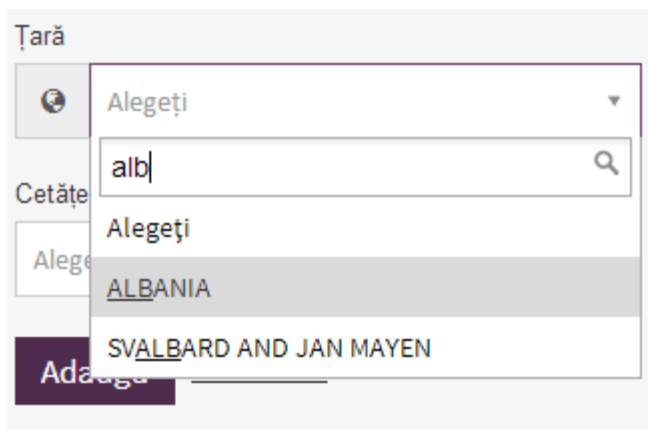


Figura II.5.3. *Dropdown* de tip *select2* (Vaynberg, 2012) având metodă de interogare atașată

6. Bootstrap

În prima variantă, *Bootstrap* a fost un produs intern dezvoltat de către inginerii de la *Twitter* pentru a standardiza obiectele și modul de dezvoltare al interfețelor în cadrul companiei. Pionierii acestui *framework* sunt Mark Otto și Jacob Thornton, ingineri software la *Twitter* în momentul lansării primei versiuni.

Proiectul a fost introdus de Mark Otto afirmând următoarele: “În ultimele zile la *Twitter*, inginerii foloseau orice librărie cu care erau familiari pentru dezvoltarea interfețelor utilizator. Având în vedere această abordare a colectivului, scalabilitatea și mentenabilitatea proiectelor era foarte greu de atins. *Bootstrap* a venit ca un răspuns pentru aceste probleme și a crescut foarte mult în perioada primului eveniment *Twitter Hackweek*. La finalul acestui eveniment aveam deja prima versiune stabilă pe care inginerii noștri o puteau folosi în cadrul companiei.” (Otto, 2011)

Bootstrap oferă suport pentru dezvoltarea formularelor, integrând mai multe *plugin-uri JavaScript*, precum și iconițe (*Glyphicons*).

În industria software, *Bootstrap* este folosit pentru crearea de site-uri *responsive*, adică site-uri a căror interfață răspunde *device-ului* din cadrul căruia este accesat (Smart TV, PC, Tablet Computer, Mobile) precum și rezoluției acestuia. Dezvoltatorii nu mai sunt constrânși să creeze mai multe interfețe pentru același produs, fiind nevoiți astfel, să dezvolte o singură interfață care să fie *responsive*.

Produsul software atașat lucrării de față este dezvoltat în *Bootstrap 3*, interfața fiind *responsive*, după cum se poate observa în figurile următoare, care prezintă panoul de control.



Figura II.6.1. Hesira – Panoul principal – versiunea *Mobile*



Figura II.6.2. Hesira – Panoul principal – versiunea *Tablet Computer*

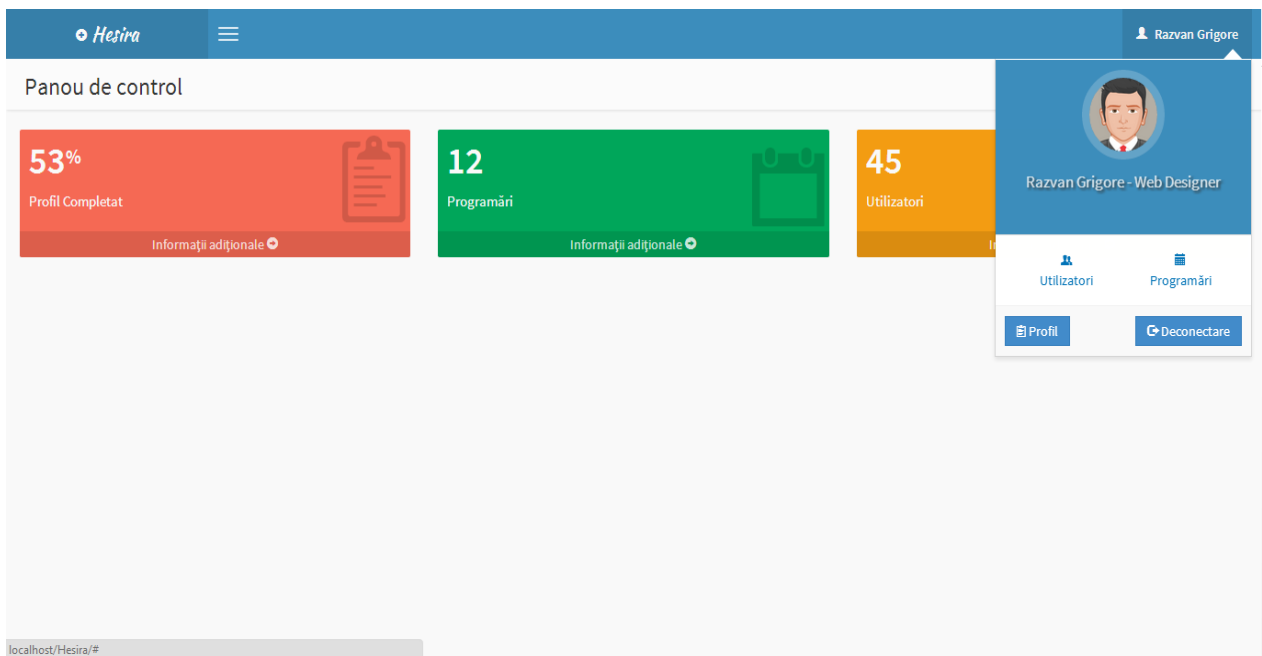


Figura II.6.3. Hesira – Panoul principal – versiunea *Personal Computer*

7. jQuery, jQuery Unobtrusive Validation & Select2

7.1. jQuery

jQuery este o librărie care oferă o putere imensă limbajului *JavaScript*. Prima versiune a fost concepută de John Resig în anul 2006 la BarCamp NYC. Între timp a fost înființată o fundație care se ocupă cu dezvoltarea acestui produs – *jQuery Foundation*, condusă de Dave Methvin. *jQuery* este cea mai populară librărie folosită în zilele noastre, statisticile arătând că dintre cele mai vizitate 10.000 de site-uri web 65% folosesc această librărie.

jQuery este o librărie populară și la nivel *enterprise*. Microsoft o utilizează pentru dezvoltarea *framework-urilor* ASP.NET AJAX și ASP.NET MVC.

Cele mai importante funcționalități *jQuery* sunt următoarele:

- selectarea elementelor DOM – nume, atribut (identificator și clasă), selectori CSS

```
DoctorUsersIndex.prototype._ initComponents = function () {
    this.$grid = $(this.selectors.gridContainer);
    this.$toolbar = $(this.selectors.toolbarContainer);
    this.$searchForm = this.$toolbar.find(this.selectors.searchForm).parent();
    this.$newForm = this.$toolbar.find(this.selectors.newForm).parent();
};
```

Figura II.7.1.1. Exemplu de selectare a elementelor DOM folosind librăria jQuery

- prelucrarea arborelui DOM
- modificarea evenimentelor
- efecte și animații
- AJAX
- informații despre cererea către server (obiectul *location*) sau despre *browser* (proprietatea *\$.browser*)
- poate fi extins cu ajutorul *plugin-urilor*

7.2. jQuery Unobtrusive Validation

jQuery Validation Plugin este realizat de către Jörn Zaefferer, membru al echipei *jQuery* și unul dintre conducătorii proiectului *jQuery UI*. Acest *plugin* a fost dezvoltat și adaptat tuturor modificărilor pe care le-a suferit librăria principală, începând cu prima versiune a acesteia, în anul 2006.

Plecând de la acest *plugin* dezvoltatorii de la Microsoft au introdus, începând cu ASP.NET MVC 3, ceea ce ei au numit *Unobtrusive Client Validation*. În MVC 2 acest sistem de validare *client-side* era decuplat de sistemul de validare *server-side*, comunicarea realizându-se cu ajutorul unui obiect JSON de forma următoare (Wilson, 2010):

```
<script type="text/javascript">
  //
    if (!window.mvcClientValidationMetadata) {
      window.mvcClientValidationMetadata = []; }
    window.mvcClientValidationMetadata.push(
      { "Fields": [{ "FieldName": "FirstName",
        "ReplaceValidationMessageContents": true,
        "ValidationMessageId": "FirstName_validationMessage",
        "ValidationRules": [{
          "ErrorMessage": "The FirstName field is required.",
          "ValidationParameters": {},
          "ValidationType": "required" } ] },
        FormId: "form0",
        "ReplaceValidationSummary": true,
        "ValidationSummaryId": "validationSummary" });
  //]]&gt;
&lt;/script&gt;</pre>
</div>
<div data-bbox="754 951 784 969" data-label="Page-Footer">
<p>23</p>
</div>
```

Acest *plugin* oferă metode pentru validarea *client-side* a datelor, fiind o alegere bună și pentru proiectele noi, dar și pentru proiectele în care vrem să integrăm această facilități pentru un anumit modul. Din punctul meu de vedere, validarea *client-side* nu este o facilități opțională, ci una obligatorie, deoarece trimiterea către server a unui formular care nu îndeplinește constrângerile minimale (de exemplu adresa de email să fie de forma adresa@provider.com) este o abordare greșită în dezvoltarea unui produs software.

Acest *plugin* oferă o varietate de metode pentru validarea *input-urilor* (validare de *url*, *email*, număr etc.), oferind în același timp programatorului posibilitatea de a defini noi reguli de validare.

Produsul software atașat folosește acest sistem de validare *client-side* a datelor introduse de utilizator. Pentru a activa acest *feature*, trebuie să adăugăm în *appSettings* cheile *ClientValidationEnabled* și *UnobtrusiveJavaScriptEnabled* amândouă având valoarea *true*.

```
<appSettings>
  <add key="webpages:Version" value="3.0.0.0" />
  <add key="webpages:Enabled" value="false" />
  <add key="ClientValidationEnabled" value="true" />
  <add key="UnobtrusiveJavaScriptEnabled" value="true" />
  <!-- cultures, languages & default language -->
  <add key="cultures" value="en-US|ro-RO" />
  <add key="languages" value="en|ro" />
  <add key="defaultLanguage" value="ro" />
  <!-- authentication settings -->
  <add key="authSalt" value="Ge6F6u27" />
</appSettings>
```

Figura II.7.2.1. Configurarea *appSettings* pentru *plugin-ul Unobtrusive Validation*

Proprietățile modelului nostru (*viewmodel* folosit pentru stratul de prezentare – *presentation layer*) vor fi decorate cu atributul *Required*. În figura următoare putem observa utilizarea acestui atribut. În cazul în care utilizatorul invalidează acest atribut, mesajul de eroare va fi afișat în funcție de cultura curentă (RO | EN).

```
[Required(ErrorMessageResourceName = "RequiredField", ErrorMessageResourceType = typeof(Resource))]
[BsControl(BsControlType.RadioButtonList)]
[Display(Name = "Role", ResourceType = typeof(Resource))]
public BsSelectList<UserRoles?> RoleBtnList { get; set; }
```

Figura II.7.2.2. Atributul *Required* folosit în cadrul modelului de prezentare

Acest *plugin* înlocuiește obiectul de tip JSON prezentat anterior cu attribute compatibile cu standardul HTML5, care descriu fiecare constrângere în parte, precum și mesajele de eroare asociate. O astfel de structură (extinsă și modificată) este prezentată în figura II.7.2.3.

```

▼<div class="col-sm-6 col-lg-4 form-group has-error">
  ▼<label class="control-label required" for="New_Firstname">
    "Prenume"
    ::after
  </label>
  ▼<div class="input-group">
    ▶<span class="glyphicon glyphicon-user input-group-addon">...</span>
    <input class="form-control bs-text input-validation-error" data-val="true" data-val-required="Acest câmp este obligatoriu"
      id="New_Firstname" name="New.Firstname" type="text" value>
    ▶<span class="input-group-addon glyphicon glyphicon-warning-sign field-validation-error" data-toggle="tooltip" data-valmsg-
      for="New.Firstname" data-valmsg-replace="true" data-original-title="Acest câmp este obligatoriu" title>...</span>
    </div>
  </div>

```

Figura II.7.2.3. Structură *Unobtrusive Validator* – starea de eroare

În afară de posibilitatea de a defini propriile metode de validare a *input-urilor* (de exemplu validarea unor *input-uri* de tip fișier doar pentru fișierele care au extensia *.png* și au cel mult 10mb), acest *plugin* oferă posibilitatea suprascrierii anumitor funcții interne. În figura următoare este prezentată o astfel de abordare în care i-am specificat obiectului de tip validator că aș dori să ignore câmpurile care sunt ascunse și să returneze întotdeauna valoarea de adevăr pentru validitatea acestora.

```

DoctorUsersIndex.prototype._initToolbar = function () {

  this.$toolbar.bsToolbar({
    uniqueName: 'usersToolbar',
    subscribers: [this.$grid],
  });

  this.$searchForm.on('bsformafterinitui', $.proxy(this._initSearchForm, this));

  this.$newForm.on('bsformafterinitui', $.proxy(this._initNewForm, this));
  this.$newForm.on('bsformbeforeformvalidation', $.proxy(this._validationRules, this));

};

DoctorUsersIndex.prototype._validationRules = function (e, data) {

  data.validator.settings.ignore = function (idx, elem) {
    if (!$.elem.parents('.form-group').is(':visible')) {
      return true;
    } else {
      return false;
    }
  };

};

```

Figura II.7.2.4. Exemplu de modificare al obiectului validator

7.3. Select2

Select2 este o bibliotecă care înlocuiește *input-urile* de tip select (de exemplu *list-box* sau *dropdown*), adăugându-le îmbunătățiri precum căutare, navigare infinită a rezultatelor și configurarea unei surse pentru generarea rezultatelor.

Funcționalități select2:

- căutare (configurarea cuvântului căutat – de exemplu lungime minimă)
- cuvânt substituent (*placeholder*)
- utilizarea *template-urilor* pentru afișarea rezultatelor
- încărcarea rezultatelor din mai multe surse (*array* sau *remote data* folosind metoda *select2-query* având atașată o cerere AJAX)
- navigare infinită a rezultatelor
- acces programatic asupra tuturor funcționalităților *input-urilor* de tip select
- evenimente precum schimbarea programatică a valorilor, deschidere, închidere, focusare, defocusare, evidențiere
- pentru *input-urile* de tip *list-box* suportă evenimente precum *drag & drop* care poate fi extins pentru a suporta ideea de sortare, opțiuni persistente
- opțiuni dezactivate
- suport pentru diacritice
- design *responsive* folosindu-se de lățimea predefinită a părintelui (DOM)

Produsul software atașat acestei lucrări integrează această librărie, folosind metoda *query* de încărcare a înregistrărilor de pe server cu ajutorul unei cereri de tip AJAX, după cum se poate observa în figura II.7.3.1.

În figura următoare avem definită metoda *select2* împreună cu opțiunile atașate pentru un câmp de tipul *dropdown* care este folosit pentru selectarea țării natale a utilizatorilor. Funcțiile *formatSelection*, *formatResult* și *id* sunt definite într-un mod generic, sursa datelor putând fi un *array* de obiecte în forma [{ *value* : '32', *text*: 'Italia' }, { *value* : '39', *text*: 'Germania' }] sau o listă provenită de pe server de tipul *SelectList* (clasă predefinită în ASP.NET MVC) a cărei componente au proprietățile *Text* și *Value*. Setarea *placeholder* (cuvânt substituent) este inițializată de către programator folosind o opțiune.

Setarea *minimumInputLength* se referă la lungimea minimă a cuvântului căutat de utilizator (*query*) pentru declanșarea unei cereri AJAX către server care să aibă ca urmare popularea inițială și/sau filtrarea înregistrărilor asociate. Această cerere AJAX este definită folosind obiectul cu același nume.

În momentul trimerii cererii AJAX trebuie să cunoaștem care este cuvântul căutat de utilizator (*queryString*), care este pagina curentă și câte înregistrări dorim să îi afișăm în continuare, aceste informații vor fi regăsite în obiectul *data* asociat. Funcția *results* este cea care intervine în momentul în care cererea AJAX este finalizată cu succes. Această funcție are responsabilitatea de a reface opțiunile din *dropdown* bazându-se pe răspunsul primit de la server.

```
this.$input.select2({
  placeholder: this.options.placeholder,
  formatResult: function (result, container, query, escapeMarkup) {
    var markup = [];
    window.Select2.util.markMatch(result.text || result.Text, query.term, markup, escapeMarkup);
    return markup.join("");
  },
  formatSelection: function (data, container) {
    return data ? typeof data.Text !== "undefined" ? data.Text : data.text : undefined;
  },
  id: function (elem) {
    return elem.Value !== null ? elem.Value.trim() : elem.value.trim();
  },
  minimumInputLength: 0,
  ajax: {
    url: this.options.url,
    dataType: "json",
    data: function (term, page) {
      var jsonData = {
        queryString: term,
        pageLimit: 10,
        currentPage: page,
      };
      return $.extend(true, jsonData, ajaxData);
    },
    results: function (response, page) {
      var leadingResults = page === 1 ? persistentData : [];
      var more = (page * 10) < response.Data.total;
      var finalResults = leadingResults.concat(response.Data.results);
      return { results: finalResults, more: more };
    }
  }
});
```

Figura II.7.3.1. Exemplu de modificare al obiectului validator

8. AdminLTE

AdminLTE este o temă de tip administrare construită la sfârșitul anului trecut (data lansării fiind 25 Decembrie 2013) pe baza temei *SB Admin 2*, care folosește *Bootstrap 3*. A fost dezvoltată de către Abdullah Almsaeed, având un design *full-responsive*, precum și o variantă pentru imprimare. *Repository-ul Github* are aproximativ 1400 de *stargazers* (persoane care au salvat acest *repository* într-o listă de favorite) și peste 300 de *fork-uri*, în jurul acestei teme creându-se o comunitate activă.

53%

Profil Completat

Informații adiționale ● (/Hesira/Doctor/Profile/3)



12

Programări

Informații adiționale ●



45

Utilizatori

Informații adiționale ● (/Hesira/Doctor/Users)



Figura II.8.1. Exemplu de imprimare – Hesira – Panoul de control

Modificarea și/sau adaptarea acestei teme pentru nevoile fiecăruia se poate face utilizând fișierele de tip LESS. În ASP.NET se pot crea *bundle-uri* sau variante minimizate ale fișierelor LESS utilizând pachetul *NuGet dotless* (Hölbling & Foster, 2013).

Funcționalități ale acestei teme (precum și componentele pe care se bazează):

- conține două tipuri de componente de tip *slider* – Bootstrap și Ion
- conține două editoare text – *CKEditor* și *WYSIHTML5*
- oferă suport afișarea coerentă a datelor (înregistrărilor) sub formă de tabele
- conține o componentă de tip calendar pentru afișarea întâlnirilor într-un mod prietenos
- oferă suport pentru validarea și integritatea *input-urilor* (de exemplu validarea numerelor de telefon din USA, utilizând *plugin-ul jQuery Input Mask*)
- oferă suport pentru afișarea datelor sub forma grafică (folosind *Flot*, *Sparkline*, *jQuery Knob* și *Morris.js*)
- oferă suport pentru afișarea progresului încărcării unei pagini sau unui fișier cu ajutorul *plugin-ului JavaScript Pace*
- oferă suport pentru afișarea informațiilor în cadrul unor hărți predefinite sau posibilitatea creării hărților proprii (folosind *jVector Map*)
- oferă suport pentru suitele de iconițe *Glyphicons*, *Fontawesome* & *Ion icons*
- este compatibil cu majoritatea navigatoarelor moderne (*IE 9+*, *Firefox 5+*, *Chrome 14+*, *Safari 5+* & *Opera 11+*)
- oferă suport pentru butoanele sociale

Interfața produsului software atașat acestei lucrări se bazează pe tema descrisă anterior, aceasta fiind integrată cu *framework-ul* pentru dezvoltarea interfețelor *BForms* care va fi prezentat în subcapitolul următor.

```
AdminLTE.prototype._initComponents = function () {  
  
    var currentContext = this;  
  
    //Enable sidebar toggle  
    $('[data-toggle='offcanvas']").click(function (e) {  
        e.preventDefault();  
  
        var active = true;  
  
        //If window is small enough, enable sidebar push menu  
        if ($(window).width() <= 992) {  
  
        } else {  
            active = false;  
        }  
  
        var data = {  
            active: active  
        };  
  
        $.bforms.ajax({  
            name: 'toggleSidebar',  
            url: requireConfig.websiteOptions.toggleSidebarUrl,  
            callbackData: data,  
            success: $.proxy(currentContext._ajaxToggleSidebarSuccess),  
            error: $.proxy(currentContext._ajaxToggleSidebarError),  
        });  
  
    });  
};
```

Figura II.8.2. Cererea de tip AJAX pentru comutarea stării *sidebar-ului*

După cum putem observa în figura anterioară, am extins această temă atașând evenimentului de *toggle* al *sidebar-ului* o cerere AJAX care să înregistreze în baza de date starea acestuia (deschis sau închis) pentru fiecare utilizator în parte. *Sidebar-ul* va rămâne închis dacă lățimea ferestrei este prea mică, comutarea stării acestuia realizându-se odată cu redimensionarea ferestrei (dacă utilizatorul are setat *sidebar-ul* în modul deschis și accesează produsul de pe telefonul mobil, acesta se va închide, adaptându-se la lățimea ferestrei; în momentul când utilizatorul va accesa produsul de pe calculatorul personal sau cu ajutorul unui televizor *smart*, *sidebar-ul* va fi în modul deschis).

Funcția *bforms.ajax* este o componentă de tip ambalaj (*wrapper*) pentru funcția *jQuery ajax*. Utilizarea acestei componente este intuitivă, opțiunile pentru cererea de tip AJAX fiind transmise ca parametru sub forma unui obiect.

Funcția de succes `_ajaxToggleSidebarSuccess` va opera modificările CSS în funcție de starea aleasă de utilizator pentru *sidebar*, după cum putem observa în figura următoare.

```
AdminLTE.prototype._ajaxToggleSidebarSuccess = function (response, callbackData) {

    if (callbackData.active != null && callbackData.active) {

        $('.row-offcanvas').toggleClass('active');
        $('.left-side').removeClass('collapse-left');
        $('.right-side').removeClass('stretch');
        $('.row-offcanvas').toggleClass('relative');

    } else {

        if (response != null &&
            response.SidebarOpen != null && !response.SidebarOpen) {

            if (!$('.right-side').hasClass('stretch')) {
                $('.right-side').addClass('stretch');
            }

            if (!$('.right-side').hasClass('collapse-left')) {
                $('.left-side').addClass('collapse-left');
            }

        }
        else if (response != null &&
            response.SidebarOpen != null && response.SidebarOpen) {

            $('.right-side').removeClass('stretch');
            $('.left-side').removeClass('collapse-left');

        }

    }

};

AdminLTE.prototype._ajaxToggleSidebarError = function(response, callbackData) {

    console.trace();

};
```

Figura II.8.3. Funcțiile de succes și de eroare pentru cererea de tip AJAX

9. BForms

BForms (BForms Team, 2013) este un *framework open source* pentru dezvoltarea rapidă a interfețelor web, fiind realizat începând cu anul 2013 de *BForms Team*, o echipă de programatori români printre care mă număr. Se bazează pe *Twitter Bootstrap* versiunea 3, modificările aduse acestei versiuni fiind realizate în SASS (Weizenbaum, 2008). Este o colecție de clase de tip ajutor adaptate pentru *Razor* (FitzMacken, 2014) și utilizate pentru generarea câmpurilor dinamice.

Această colecție cuprinde și modulele *JavaScript* AMD (Correia, 2014) asociate acestor componente.

SASS este un limbaj care extinde posibilitățile limbajului CSS adăugând funcționalități precum:

- suportă utilizarea variabilelor
- suportă utilizarea claselor de tip *mixins* (clase care pot conține combinații ale metodelor aflate în clase diferite)
- funcții de colorare (*HSL – greyscale, invert | RGB - rgba, mix*)
- funcții de opacitate (*opacity, transparentize*)
- funcții pentru șiruri de caractere (*quote, str-insert*)
- funcții pentru utilizarea numerelor (*percentage, random*)
- funcții pentru utilizarea listelor (*append, join, zip*)
- funcții pentru utilizarea dicționarelor (*map-get, map-merge, map-keys, map-values*)
- funcții de inspectare (*mixin-exists, feature-exists, comparable*)
- poate fi integrat cu *Firebug*

Pentru a putea fi utilizat în mediul de lucru Microsoft trebuie instalat *Ruby*, deoarece limbajul SASS folosește această dependență pentru compilare.

BForms poate fi inclus în proiect via *NuGet*, instalând pachetul asociat *BForms.MVC* care va adăuga directorul *BForms* în cadrul directorului *Scripts* al proiectului, precum și referințe către pachetele *BForms* și *RequireJs.Net*. Componentele JavaScript și CSS ale acestui pachet sunt prezente în două forme diferite, ca fișiere individuale, și ca suite minimizate.

```
3 namespace Hesira.App_Start
4 {
5     public class BundleConfig
6     {
7         // For more information on bundling, visit http://go.microsoft.com/fwlink/?LinkId=301862
8         public static void RegisterBundles(BundleCollection bundles)
9         {
10
11             bundles.Add(new StyleBundle("~/BFormsCSS")
12                 .Include("~/Scripts/BForms/Bundles/css/*.css", new CssRewriteUrlTransform())
13                 .Include("~/Content/Stylesheets/Site/site.css", new CssRewriteUrlTransform())
14             );
15
16             bundles.Add(new StyleBundle("~/LTEAdminCSS")
17                 .Include("~/Content/Stylesheets/LTEComponents/css/AdminLTE.css", new CssRewriteUrlTransform()));
18
19         }
20     }
21 }
```

Figura II.9.1. Definirea suitelor CSS – Hesira – *BundleConfig*

În figura anterioară este prezentată clasa *BundleConfig* în care am definit suitele CSS minimalizate BFormsCSS (format din suitele CSS reprezentative pentru componentele *BForms* și modificările aduse acestora care se regăsesc în *site.css*) și *LTEAdminCSS* format din fișierul CSS asociat temei *AdminLTE*.

Aceste suite le vom “despacheta” cu ajutorul metodei *Render* aparținând clasei de tip ajutor *Styles* din *System.Web.Optimization*. Produsul software atașat acestui lucrări conține un fișier de tip *layout* care este folosit ca machetă de fiecare *view* din proiect. În cadrul acestei machete am ”despachetat” suitele prezentate anterior.

```

1  @using System.Threading
2  @using BForms.Html
3  @using BForms.Models
4  @using Hesira.Helpers.General
5  @using Hesira.Helpers.Globalisation
6  @using Hesira.Helpers.Html
7  @using Hesira.Resources
8  @using RequireJS;
9
10 <html>
11 <head>
12     <meta charset="UTF-8">
13     <meta content="width=device-width, initial-scale=1, maximum-scale=1, user-scalable=no" name="viewport">
14
15     @Styles.Render("~/BFormsCSS")
16
17     @Styles.Render("~/LTEAdminCSS")
18
19
20     @*For IE7-8 support of HTML5 elements and responsive*@
21     <!--[if lt IE 9]>
22         <script src="@Url.Content("~/Scripts/BForms/Bundles/iefix.js")"
23             type="text/javascript">
24         </script>
25     <![endif]>
26     <title>@ViewBag.Title</title>
27     <link href="@~/favicon.ico" rel="shortcut icon" type="image/x-icon">
28
29 </head>
30

```

Figura II.9.2. Despachetarea suitelor CSS – Hesira – *_Layout.cshtml*

BForms folosește un furnizor diferit pentru validarea modelelor aparținând stratului de prezentare (*view models – presentation layer*). Pentru utilizarea acestui furnizor trebuie să-l adăugăm în lista de furnizori în momentul deschiderii aplicației.

```

protected void Application_Start()
{
    ViewEngines.Engines.Clear();
    ViewEngines.Engines.Add(new RazorViewExtension());

    FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);
    AreaRegistration.RegisterAllAreas();
    RouteConfig.RegisterRoutes(RouteTable.Routes);
    BundleConfig.RegisterBundles(BundleTable.Bundles);

    ModelValidatorProviders.Providers.Add(new BsModelValidatorProvider());
    BForms.Utilities.BsResourceManager.Register(Resource.ResourceManager);

    BForms.Utilities.BsUIManager.Theme(BsTheme.Purple);
}

```

Figura II.9.3. Adăugarea furnizorului pentru validarea modelelor

BForms oferă două modalități pentru utilizarea claselor de tip ajutor adaptate pentru *Razor*. Amândouă implică decorarea proprietăților modelului cu un atribut de tipul *BsControlAttribute*. În cadrul componentei de tip *view* putem să folosim ori clasele generice de tip ajutor *BsInputFor*, *BsSelectFor* sau *BsRangeFor* în funcție de categoria în care se încadrează câmpul pe care dorim să-l generăm, ori clasele de tip ajutor specifice câmpului printre care se numără *BsTextBoxFor*, *BsTextAreaFor*, *BsDropDownListFor*, *BsListBoxFor* etc.

În cadrul produsului software atașat am folosit prima dintre cele două abordări prezentate deoarece este o abordare generalistă care facilitează modificarea ulterioară a tipului câmpurilor generate. De exemplu putem utiliza clasa generică de tip ajutor *BsRangeFor* pentru generarea unui câmp de tipul interval în care putem seta o dată de start și una de sfârșit. Presupunem că în faza de mentenanță avem o cerere de modificare acestui câmp pentru schimbarea intervalului dintr-un interval de date într-un interval de marcaje temporale (*timestamp*). Singura sarcină pe care trebuie să o realizeze programatorul este modificarea atributului atașat proprietății, componenta de tip *view* rămânând intactă.

```
[Display(Name = "CNP", ResourceType = typeof(Resource))]
[BsControl(BsControlType.TextBox)]
public string CNP { get; set; }

[Display(Name = "Email", ResourceType = typeof(Resource))]
[BsControl(BsControlType.TextBox)]
public string Email { get; set; }

[BsControl(BsControlType.RadioButtonList)]
[Display(Name = "IsEnabled", ResourceType = typeof(Resource))]
public BsSelectList<YesNoEnum?> EnableBtnList { get; set; }

[BsControl(BsControlType.RadioButtonList)]
[Display(Name = "Role", ResourceType = typeof(Resource))]
public BsSelectList<UserRoles?> RoleBtnList { get; set; }

[Display(Name = "AgeInterval", ResourceType = typeof(Resource))]
[BsControl(BsControlType.NumberRange)]
public BsRange<int?> AgeRange { get; set; }

[Display(Name = "Gender", ResourceType = typeof(Resource))]
[BsControl(BsControlType.RadioButtonList)]
public BsSelectList<GenderEnum?> GenderBtnList { get; set; }

[Display(Name = "Citizenship", Prompt = "Choose", ResourceType = typeof(Resource))]
[BsControl(BsControlType.ListBox)]
public BsSelectList<List<string>> CitizenshipListBox { get; set; }
```

Figura II.9.4. Modelul utilizat pentru filtrarea utilizatorilor asociat stratului de prezentare

În figura anterioară este prezentat modelul asociat stratului de prezentare, folosit pentru filtrarea utilizatorilor. Putem observa că proprietățile *EnableBtnList*, *RoleBtnList* și *GenderBtnList* sunt decorate cu atributul *BsControlType.RadioButtonList* folosit pentru generarea componentei HTML cu același nume.

```

49 <div class="col-sm-6 col-md-6 col-lg-4 form-group">
50   @Html.BsLabelFor(m => m.Email)
51   <div class="input-group">
52     @Html.BsGlyphiconAddon(Glyphicon.Envelope)
53     @Html.BsInputFor(model => model.Email)
54   </div>
55 </div>
56
57 <div class="col-sm-6 col-md-6 col-lg-6 form-group">
58   @Html.BsLabelFor(m => m.RoleBtnList)
59   <div class="input-group">
60     @Html.BsGlyphiconAddon(Glyphicon.Star)
61     @Html.BsSelectFor(model => model.RoleBtnList, new { data_initialvalue = (byte)UserRoles.All },
62     new Dictionary<string, object> { { "preventDeselect", true } })
63   </div>
64 </div>
65
66
67 <div class="col-sm-6 col-md-6 col-lg-6 form-group">
68   @Html.BsLabelFor(m => m.CitizenshipListBox)
69   <div class="input-group">
70     @Html.BsGlyphiconAddon(Glyphicon.MapMarker)
71     @Html.BsSelectFor(model => model.CitizenshipListBox, new { @class = "no-initUI js-citizenshiplist" })
72   </div>
73 </div>
74

```

Figura II.9.5. Folosirea claselor de tip ajutor în cadrul formularului întrebuițat pentru filtrarea utilizatorilor

În figura anterioară este prezentată utilizarea claselor de tip ajutor pentru generarea componentelor HTML în cadrul formularului utilizat pentru filtrarea utilizatorilor. Pentru proprietatea *RoleBtnList* a modelului am folosit metoda *BsSelectFor* având ca parametrii un obiect *HtmlAttributes*, care conține *data_initialvalue* (folosită la inițializarea și la resetarea formularului) și un dicționar *dataOptions*, care conține opțiunea de prevenire a deselectării butoanelor de tip *radio* având asociată valoarea de adevăr.

BForms oferă suport pentru dezvoltarea unei componente de tip *grid*, asemănătoare controlului *GridView* predefinit în cadrul *framework-ului* ASP.NET *WebForms*. Acestei componente i se poate asocia un *toolbar* care poate fi modificat de către programator în concordanță cu cerințele proiectului. Utilizarea acestei componente este prezentată amănunțit în cadrul subcapitolului III.2.2, reprezentând o parte importantă a modului de management al utilizatorilor.

O altă componentă din cadrul *framework-ului* BForms care a fost utilizată în produsul software atașat este cea de tip *group-editor*, convertită în cadrul proiectului pentru crearea rețetelor pacienților. Grupul este de fapt rețeta finală, care poate fi modificată adăugând sau eliminând înregistrări (medicamente) sau modificând proprietățile acestora (zile de administrare / cantitate). Utilizarea acestei componente este prezentată amănunțit în cadrul subcapitolului III.2.4. reprezentând o parte importantă a modulului de management al rețetelor.

Componenta principală a modulului de management al profilului (subcapitolul III.2.1.) este compusă din panouri a căror stare se poate schimba, devenind editabile sau intrând într-un mod care permite doar vizualizarea acestora. În cadrul *framework-ului* este denumită intuitiv: componentă pentru dezvoltarea profilurilor de utilizator.

Concluzionând, *BForms* este un *framework* ușor de folosit și de adaptat în concordanță cu cerințele proiectului, aducând un plus valoare din punct de vedere estetic și eliminând o mare parte din timpul dedicat de programator pentru dezvoltarea interfețelor utilizator.

III. DESCRIEREA APLICAȚIEI ȘI DEZVOLTARE ULTERIOARĂ

1. Inițializare

1.1. Structură

Structura produsului software atașat acestei lucrări este bazată pe șablonul arhitectural MVC. La nivel macro, produsul este structurat pe arii care definesc foarte strict rolurile pe care le poate avea un utilizator, precum și modulele la care acesta are acces.

În figura III.1.1.1. este prezentată structura generală. Cele mai importante zone din punct de vedere structural au fost numerotate.

- 1 – directorul în care sunt definite entitățile cu care operează acest proces. Aici se află fișierul de tip EDMX generat cu ajutorul *framework-ului* EF (*Entity Framework*)
- 2 – directorul asociat șabloanelor de proiectare Dependency Injection & Unit of Work (va fi prezentat amănunțit în cadrul capitolului IV)
- 3 – directorul care cuprinde cele patru arii definite: *Common* (utilizată pentru definirea componentelor de tip *controller*, *view* și *model* care sunt folosite de către toate rolurile), respectiv ariile specifice (*Admin*, *Patient* și *Doctor*) în care se regăsesc componentele specifice acestor roluri
- 4 – directorul care cuprinde elementele statice ale produsului (imagini etc.), precum și fișierele CSS
- 5, 7, 11 – sunt directoarele care definesc cele trei componente (*model*, *view* și *controller*) utilizate la nivelul macro al aplicației (în *AjaxController* este definită metoda de *toggle* a *sidebar-ului*)
- 8 – directorul în care se află clasele de tip *Repository* (acest șablon de proiectare este descris în cadrul capitolului IV)
- 9 – directorul în care se află fișierele de tip XML utilizate pentru multilingvism
- 10 – directorul în care se află fișierele *JavaScript* structurate în funcție de *controller-ul* de care aparțin (structură *RequireJS* – capitolul IV) precum și alte componente utilizate în dezvoltarea acestui produs (directorul reprezentativ pentru *framework-ul* de dezvoltare al interfețelor *BForms*)

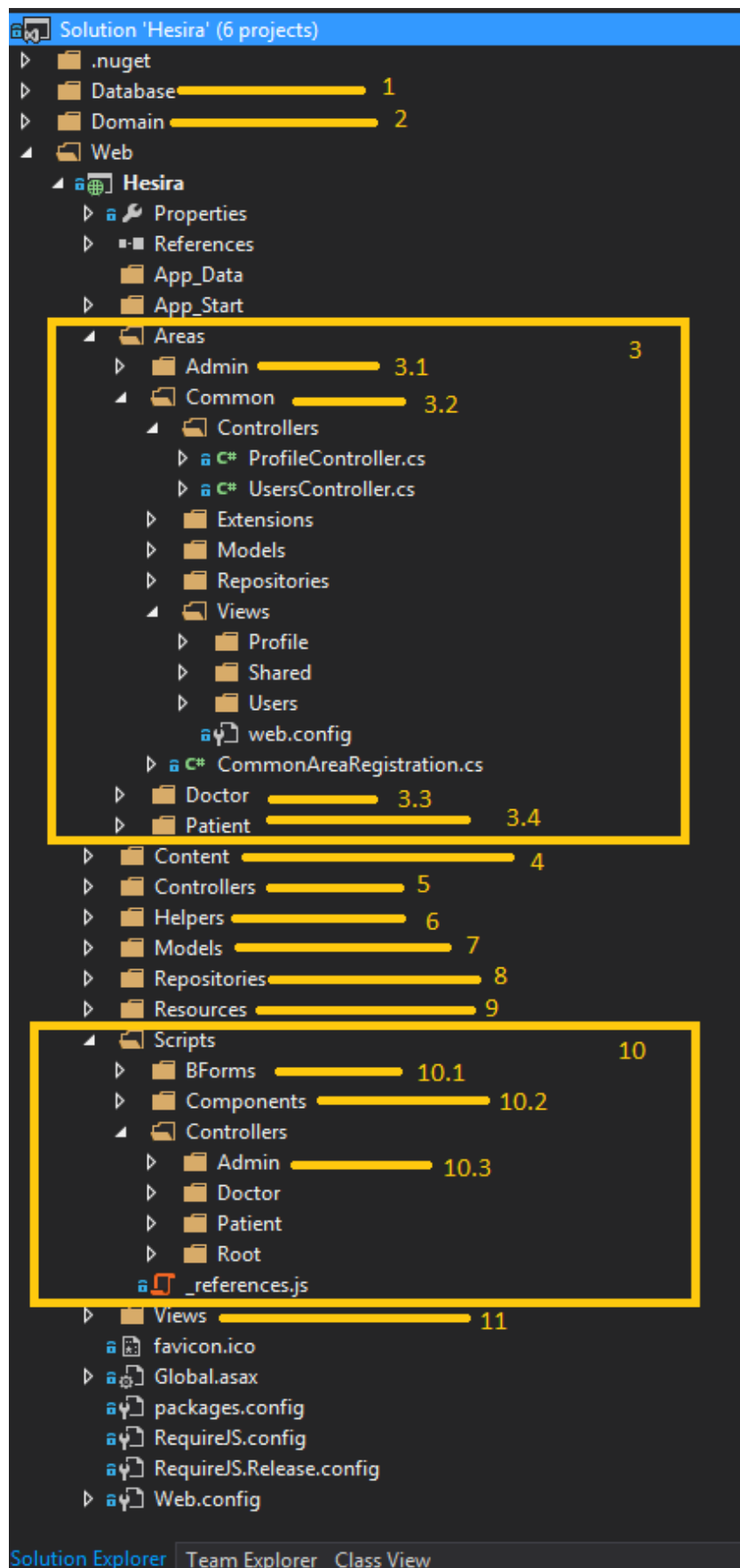


Figura III.1.1.1. Structura produsului software - Hesira

1.2. Bază de date

Produsul software atașat acestei lucrări folosește o bază de date MS SQL, utilizând ca server produsul Microsoft SQL Server 2012.

Baza de date este numită *Hesira* conținând următoarele tabele: *Addresses* (reprezentând adresele utilizatorilor), *Appointments* (reprezentând programările medicale), *ConsultationRooms* (reprezentând date despre centrul medical), *Countries* (pentru definirea adresei, precum și a cetățeniei unui utilizator), *Diseases* (reprezentând gruparea diagnosticelor în clase), *Drugs* (reprezentând tabela în care se mențin informațiile referitoare la medicamente), *DrugsDiseases* (tabela de legătură pentru cele două tabele declarate anterior), *Files* (tabelă pentru menținerea documentelor prezentate în cadrul fiecărei programări), *Prescriptions* (reprezentând rețetele emise), *PrescriptionsDrugs* (reprezentând medicamentele asociate unei rețete precum și specificații de utilizare ale acestora – tabela de legătură dintre *Prescriptions* și *Drugs*), *Users* (reprezentând tabela pentru menținerea utilizatorilor produsului), *UsersProfiles* (reprezentând tabela în care este definit profilul unui utilizator) și *UsersStates* (reprezentând tabela pentru definirea statutului CNAS al unui pacient).

	Column Name	Data Type	Allow Nulls
PK	Id	bigint	<input type="checkbox"/>
	Firstname	nvarchar(MAX)	<input type="checkbox"/>
	Lastname	nvarchar(MAX)	<input type="checkbox"/>
	IsDoctor	bit	<input type="checkbox"/>
	Enabled	bit	<input type="checkbox"/>
	Password	nvarchar(MAX)	<input type="checkbox"/>
	Email	nvarchar(MAX)	<input checked="" type="checkbox"/>
	SidebarMode	bit	<input type="checkbox"/>
	Timestamp	datetime	<input type="checkbox"/>
	CNP	nvarchar(MAX)	<input type="checkbox"/>
	IsAdmin	bit	<input type="checkbox"/>
	IsPatient	bit	<input type="checkbox"/>
	LastAuthDate	datetime	<input checked="" type="checkbox"/>

Figura III.1.2.1. Tabela *Users*


	Column Name	Data Type	Allow Nulls
	Id	bigint	<input type="checkbox"/>
	Series	nvarchar(MAX)	<input type="checkbox"/>
	Number	bigint	<input type="checkbox"/>
	Id_Patient	bigint	<input type="checkbox"/>
	PrescriptionDate	date	<input type="checkbox"/>
	Days	int	<input type="checkbox"/>
	Description	nvarchar(MAX)	<input type="checkbox"/>
	Id_Doctor	bigint	<input type="checkbox"/>

Figura III.1.2.2. Tabela *UsersProfiles*

Id_Citizenship este utilizat pentru definirea cetățeniei unui utilizator, *Id_Address* pentru definirea înregistrării corelate din tabela pentru adrese (existând posibilitatea ca aceasta să fie nulă), iar *Id_State* este utilizat pentru definirea statutului CNAS pentru un utilizator al cărui rol este acela de pacient (nu suntem interesați de statutul unui doctor sau al unui administrator – deci acesta poate fi nul). Toate coloanele precizate anterior sunt chei externe.


	Column Name	Data Type	Allow Nulls
	Id	bigint	<input type="checkbox"/>
	Id_User	bigint	<input type="checkbox"/>
	Birthday	date	<input type="checkbox"/>
	Id_Address	bigint	<input checked="" type="checkbox"/>
	PhoneNumber	nvarchar(MAX)	<input checked="" type="checkbox"/>
	LastVisitDate	date	<input checked="" type="checkbox"/>
	Gender	int	<input type="checkbox"/>
	Profession	nvarchar(MAX)	<input checked="" type="checkbox"/>
	Id_Citizenship	int	<input type="checkbox"/>
	Id_State	int	<input checked="" type="checkbox"/>
	Timestamp	datetime	<input type="checkbox"/>

Figura III.1.2.3. Tabela *Prescriptions*

Id_Doctor este utilizat pentru identificarea doctorului care a emis rețeta, iar *Id_Patient* este utilizat pentru identificarea pacientului. Cele două coloane sunt chei externe – tabela *Users*.

	Column Name	Data Type	Allow Nulls
►	Id_Prescription	bigint	<input type="checkbox"/>
	Id_Drug	bigint	<input type="checkbox"/>
	Quantity	int	<input type="checkbox"/>
	Days	int	<input type="checkbox"/>
	Id_Disease	int	<input type="checkbox"/>
			<input type="checkbox"/>

Figura III.1.2.4. Tabela *PrescriptionsDrugs*

Id_Prescription este utilizat pentru identificarea rețetei emise, *Id_Drug* este folosit pentru identificarea medicamentului al cărui metodă de administrare (cantitate / număr de zile) este definită în cadrul acestei tabele, iar *Id_Disease* reprezintă diagnosticul care se presupune că este tratat de medicamentul asociat. Toate cele trei coloane reprezintă chei externe.

	Column Name	Data Type	Allow Nulls
🔑	Id	bigint	<input type="checkbox"/>
	StartDate	datetime	<input type="checkbox"/>
	Trace	nvarchar(MAX)	<input checked="" type="checkbox"/>
	Id_Prescription	bigint	<input checked="" type="checkbox"/>
	Id_Patient	bigint	<input type="checkbox"/>
	Id_Doctor	bigint	<input type="checkbox"/>
	EndDate	datetime	<input type="checkbox"/>
	State	int	<input type="checkbox"/>

Figura III.1.2.5. Tabela *Appointments*

Id_Prescription este utilizat pentru identificarea rețetei emise (o programare se poate finaliza cu emiterea unei rețete sau nu), *Id_Doctor* este folosit pentru identificarea doctorului care a emis rețeta, iar *Id_Patient* este utilizat pentru identificarea pacientului. Toate cele 3 coloane sunt chei externe – tabelele *Prescriptions* și *Users*. Coloana *State* definește starea în care se află programarea (în desfășurare – după ce a fost generată de sistem, finalizată – în momentul finalizării de către doctor sau respinsă – atunci când pacientul nu se prezintă – stare care se presupune a fi dezvoltată ulterior prezentării lucrării de față).

1.3. Multilingualism

În anul 2005 Comisia Europeană emitea o comunicare legată de strategia acestei organizații cu privire la promovarea multilingvismului în cadrul societății europene și a activităților economice. Obiectivul acestei comunicări era acela de a consolida competențele lingvistice ale cetățenilor europeni, fiecare dintre aceștia fiind susținut pentru a dobândi cunoștințe practice în cel puțin două limbi, excluzând limba maternă. (Europa, 2006)

Având în vedere tendința structurilor administrative de a oferi sprijin cetățenilor pentru învățarea limbilor de circulație internațională, precum și utilitatea acestor decizii în contextul globalizării, produsul software atașat acestei lucrări a fost conceput pentru a suporta limba română precum și o limbă de circulație internațională (limba engleză). Acest comportament poate fi replicat cu ușurință pentru alte culturi.

Un mod foarte simplu pentru îndeplinirea acestui comportament este utilizarea fișierelor de tip resursă (Afana, 2011) și setarea implicită sau odată cu modificările utilizatorului a culturii curente.

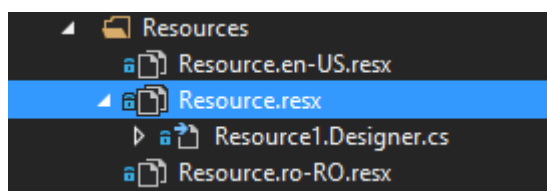


Figura III.1.3.1. Structura directorului de resurse

În cadrul directorului de resurse am definit fișierele de tip RESX utilizate pentru limba română, respectiv limba engleză, precum și fișierul central RESX. Clasa *Resource* este utilizată în cadrul componentelor de tip *view Razor* pentru localizarea mesajelor.

ProfileCompleteness	Profile Completeness	
RememberMe	Remember me	
RequiredField	This input is required	
Reset	Reset	
ResultsPerPage	results per page	
Role	Role	
Romanian	Romanian	

Figura III.1.3.2. Proprietăți definite în cadrul fișierului RESX asociat limbii engleze

În figura următoare putem observa proprietățile din cadrul figurii III.1.3.2. asociate limbii române.

ProfileCompleteness	Profil Completat	
RememberMe	Ține-mă minte	
RequiredField	Acest câmp este obligatoriu	
Reset	Resetează	
ResultsPerPage	înregistrări pe pagină	
Role	Rol	
Romanian	Română	

Figura III.1.3.3. Proprietăți definite în cadrul fișierului RESX asociat limbii române

Clasa *Resource* este definită inițial în cadrul *framework-ului* cu specificatorul de acces *internal*. Pentru a putea accesa resursele din cadrul componentelor *view* de tip *Razor*, suntem nevoiți să schimbăm modificatorul de acces în *public*. Acest lucru se poate realiza accesând fișierul *Resource.resx*.

```

<!-- sidebar menu: : style can be found in sidebar.less -->
<ul class="sidebar-menu">
  <li>
    <a href=@Url.Action("Index", "Home", new { area = "" })>
      @Html.BsGlyphicon(Glyphicon.Home) <span>@Resource.Dashboard</span>
    </a>
  </li>
  <li>
    <a href="#">
      @Html.BsGlyphicon(Glyphicon.Calendar) <span>@Resource.Appointments</span>
    </a>
  </li>
  <li>
    @if (userData.IsAdmin || userData.IsDoctor)
    {
      <a href=@UsersUrl>
        <i class="ion ion-person-stalker"></i> <span>@Resource.Users</span>
      </a>
    }
    else
    {
      <a href="#">
        <i class="ion ion-medkit"></i> <span>@Resource.Prescriptions</span>
      </a>
    }
  </li>

```

Figura III.1.3.4. Accesarea resurselor în cadrul *view-ului*

Generarea rutelor se realizează cu ajutorul metodei *MapGlobalisationRoutes* care are ca parametrii numele, *url-ul*, obiectele implicite, constrângerile, *namespace-ul* asociat, precum și numele ariei asociate. În figura III.1.3.5. este definit un tip de rută de acces pentru aria de administrare. Fiecărei culturi diferită de cea implicită i se va asocia o rută de tipul “*cultură/arie/controller/userId*”, pentru cultura română definindu-se o rută de tipul “*arie/controller/userId*”. Astfel pentru vizualizarea profilului unui utilizator în cadrul ariei de administrare vom avea rutele de acces următoare “*host/Hesira/Admin/Profile/userId*” pentru limba (cultura) română, respectiv “*host/Hesira/en/Admin/Profile/userId*” pentru engleză.

```
context.Routes.MapGlobalisationRoutes(
    "AdminProfile",
    AreaName + "/{controller}/{userId}",
    new {action = "Index", culture = "ro"},
    new RouteValueDictionary() {{"userId", "[0-9]+"}},
    new[] { "Hesira.Areas.Admin.Controllers"},
    AreaName
);
```

Figura III.1.3.5. Accesarea resurselor în cadrul *view-ului*

În următoarele două figuri (III.1.3.6 și III.1.3.7.) putem observa interfața modului de management al utilizatorilor prezentată în cele două culturi definite anterior. În cadrul componentei de tip *sidebar* avem un buton de *toggle* al culturii, care momentan este marcat cu textul *Romanian*, deoarece va schimba cultura asociată *thread-ului* curent în română (figura III.1.3.6).

Name	Role	CNP	Email
Victor Amariei	★★ Doctor	1630820126830	victorAm@gmail.com
Maria Cimpaliuc	★★ Doctor	2930508286610	mar@gmail.com
Ioana Muresan	★★ Doctor	2960508351272	muresanloana@gmail.com
Geo Ion	★★ Doctor	1561015111840	geo@gf.com

Figura III.1.3.6. Interfață în limba engleză

UTILIZATORI + Adaugă Caută

Nume CNP Gen
 Activ Toate Da Nu Interval de vârstă 1 - 2 Email
 Rol Toate Pacient Doctor Admin Cetățean al țării Alegeți

Caută Resetează

Nume	Rol	CNP	Email
Victor Amariei	★★ Doctor	1630820126830	victorAm@gmail.com
Maria Cimpaliuc	★★ Doctor	2930508286610	mar@gmail.com
Ioana Muresan	★★ Doctor	2960508351272	muresanloana@gmail.com
Geo Ion	★★ Doctor	1561015111840	geo@gf.com

Figura III.1.3.7. Interfață în limba română

Acțiunea de *toggle* între culturi are ca efect modificare interfeței utilizator în concordanță cu cultura selectată. Se păstrează toate particularitățile ultimei cereri HTTP (aceleași valori ale dicționarului asociat rutelor, aceleași componente de tip *queryString* etc.). Metoda este prezentată în figura următoare.

```
public static string ChangeLanguage(this UrlHelper instance, string lang)
{
    var culture = lang == DefaultLanguage ? DefaultLanguage : lang;

    var rd = instance.RequestContext.RouteData;
    var request = instance.RequestContext.HttpContext.Request;
    var values = new RouteValueDictionary(rd.Values);
    foreach (string key in request.QueryString.Keys)
    {
        values[key] = request.QueryString[key];
    }
    values["culture"] = culture;

    return instance.RouteUrl(values);
}
```

Figura III.1.3.8. Metoda de modificare a culturii

Modificarea culturii se realizează în cadrul metodei *GetHandler*, care aparține clasei *GlobalisationRouteHandle*. Această metodă suprascrie *handler-ul* implicit, verificând existența valorii *culture* în obiectul data asociat cererii HTTP și setând cultura *thread-ului* curent în funcție de aceasta.

```
public class GlobalisationRouteHandler : MvcRouteHandler
{
    protected override IHttpHandler GetHandler(RequestContext requestContext)
    {
        object culture;

        if (!requestContext.RouteData.Values.TryGetValue("culture", out culture) ||
            culture == null)
        {
            culture = LanguageHelper.DefaultLanguage;
        }

        var ci = new CultureInfo(MvcApplication.CulturesDictionary[culture.ToString()]);
        Thread.CurrentThread.CurrentUICulture = ci;
        Thread.CurrentThread.CurrentCulture = CultureInfo.CreateSpecificCulture(ci.Name);

        return base.GetHandler(requestContext);
    }
}
```

Figura III.1.3.9. Clasa *GlobalisationRouteHandler*

1.4. Autentificare

Produsul software asociat acestei lucrări nu permite înregistrarea utilizatorilor. Aceștia sunt adăugați de către doctor sau administrator cu ajutorul formularului pus la dispoziție în cadrul modului de management al utilizatorilor.

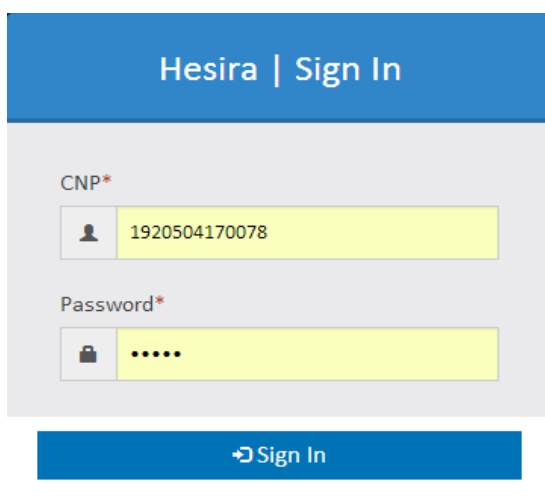


Figura III.1.4.1. Formularul de autentificare – varianta limba engleză

Produsul software asociat lucrării de față folosește sistemul ASP.NET *Membership* (MSDN, 2012) împreună cu sistemul *Forms Authentication* (Wasson, 2012), utilizând *cookie-urile* reținute în cadrul acestui sistem pentru verificarea accesului utilizatorului la un anumit modul.

Sistemul *Forms Authentication* funcționează astfel:

1. Utilizatorul operează o cerere care necesita autentificare
2. Dacă utilizatorul nu este autentificat, serverul returnează codul HTTP 302 și îl redirecționează către pagina de autentificare
3. Utilizatorul încearcă să se autentifice
4. Serverul returnează codul HTTP 302 încă odată redirecționând utilizatorul către resursa cerută inițial, acest răspuns incluzând un *cookie* folosit pentru autentificare
5. Utilizatorul operează din nou cererea inițială, de această dată având asociat *cookie-ul* pentru autentificare, deci serverul va oferi accesul

```
[HttpPost]
public ActionResult Login(LoginViewModel loginModel)
{
    if (ModelState.IsValid)
    {
        if (Membership.ValidateUser(loginModel.CNP, loginModel.Password))
        {
            var userData = new UserData(HesiraDB, loginModel.CNP);
            if (HttpContext.Session != null)
            {
                HttpContext.Session["UserData"] = userData;
            }

            FormsAuthentication.SetAuthCookie(loginModel.CNP, true);

            if (LocalAndSafeUrl(loginModel.ReturnUrl))
            {
                return Redirect(loginModel.ReturnUrl);
            }

            return RedirectToAction("Index", "Home");
        }

        var message = Resource.SignInFailed;

        // login message - provider validation method
        if (System.Web.HttpContext.Current.Items["LoginMessage"] != null &&
            !string.IsNullOrEmpty(System.Web.HttpContext.Current.Items["LoginMessage"].ToString()))
        {
            message = (System.Web.HttpContext.Current.Items["LoginMessage"]).ToString();
        }

        ModelState.AddModelError("", message);
    }
    return View(loginModel);
}
```

Figura III.1.4.2. Acțiunea de autentificare – HTTP POST

Validarea utilizatorilor se face în funcție de codul numeric personal și de parolă. Dacă utilizatorul este validat este adăugat în sesiune obiectul de tip *UserData* și *cookie-ul* asociat sistemului *Forms Authentication*. În momentul deconectării sau redirectionării din cauza restricțiilor de acces, se reține *url-ul* asociat cererii originale în proprietatea *ReturnUrl*. Metoda *LocalAndSafeUrl* verifică dacă acest *ReturnUrl* aparține produsului și dacă este sigur pentru accesare. Dacă validarea eșuează, utilizatorului i se va afișa un mesaj.

Acest sistem a fost extins în cadrul produsului atașat, pe baza lui fiind definite attribute de acces. Dacă utilizatorul este autentificat ca pacient și operează o cerere care necesită rolul de administrator sau de doctor (sunt verificate și cererile de tip AJAX), acesta este redirectionat către formularul de autentificare pentru a se conecta cu rolul asociat resursei cerute. Sistemul este ierarhic în cadrul anumitor module (modulul de management al utilizatorilor) sau specific (modulul de vizualizare a istoricului medical – unde accesul este permis doar pacienților).

2. Funcționalități

2.1. Modulul de management al profilului

În cadrul produsului software atașat am dezvoltat un modul de management al profilului, care permite vizualizarea datelor reprezentative pentru un anumit utilizator precum și editarea acestora. În cadrul acestei suite de date reprezentative am introdus informații privind identificarea utilizatorului, rolul acestuia, precum și datele de contact pentru a favoriza comunicarea dintre cele două comunități prezente în cadrul aplicației (doctori și pacienți).

Profilul unui utilizator poate fi vizualizat de oricine, dreptul de editare aparținând doar posesorului. Pagina profilului este împărțită în trei panouri distincte: Informații de bază, Alte Informații și Modalități de contact. Informațiile de bază nu pot fi modificate deoarece ar declanșa efecte secundare asupra integrității produsului (în cadrul versiunilor viitoare se va permite editarea rolului în cadrul modulului de management al utilizatorilor, extinzând regulile de securitate descrise momentan).

În cadrul panoului Alte Informații regăsim date privind identificarea utilizatorului, data nașterii, precum și statutul acestuia în cadrul Casei Naționale de Asigurări de Sănătate. Modalitățile de contact sunt variate, fiind prezentată adresa de email, numărul de telefon precum și adresa fizică (de reședință) a utilizatorului.

Interfața permite ascunderea, respectiv expandarea unui panou pentru a putea avea un acces dinamic asupra informațiilor prezentate.



Informații de bază	
Rol:	Admin
CNP:	1920504170078
Cetățean al țării:	ROU
Țară:	Romania
▼ Alte Informații 	
▼ Modalități de contact 	

Figura III.2.1.1. Ascunderea panourilor – Profilul utilizatorului

Panourile din care este construită această pagină pot avea două stări diferite (*readonly* și *editable*). În starea *readonly* se permite citirea acestor date, iar în cea *editable* se permite editarea acestora. Starea de *editable* poate fi anulată, panoul revenind în starea inițială (*readonly*), modificările curente fiind anulate.


▲ Alte Informații  1	
2	
Prenume:	Razvan
Nume:	Dumitru
Profesie:	Web Designer
Statut:	Plecat din tara
Zi de naștere:	4 mai 1992

Figura III.2.1.2. Panoul Alte Informații – starea *readonly*

Panoul poate fi expandat sau ascuns cu ajutorul săgeții marcate cu numărul 2 în cadrul figurii de mai sus. Intrarea panoului în starea de editare se face cu ajutorul butonului marcat cu un stilou din partea dreapta-sus a figurii III.2.1.2.

În modul editabil câmpurile asociate fiecărei proprietăți sunt completate cu valorile curente ale acestor proprietăți. În continuare vom demonstra această funcționalitate precum și posibilitatea editării profilului.

Figura III.2.1.3. Panoul Alte Informații – starea *editable*

În figura anterioară este prezentat panoul Alte Informații în starea editabilă. Anularea modificărilor și implicit a stării editabile se poate realiza în două moduri: cu ajutorul butonului de anulare sau apăsând pe iconița X. Salvarea modificărilor se realizează cu ajutorul butonului asociat. Am modificat valoarea câmpului de tip *dropdown* în LIBER, înainte valoare acestuia fiind PDT (plecat din țară).

Figura III.2.1.4. Panoul Alte Informații – starea *readonly* - modificat

În figura anterioară este prezentat panoul Alte Informații după ce a fost editat. Putem observa modificarea statutului utilizatorului în liber profesionist.

Formularului afișat în starea editabilă a panoului i se pot asocia reguli de validare de tipul celor prezentate în cadrul modulului de management al utilizatorilor. Câmpurile obligatorii care nu sunt completate corespunzător vor fi marcate cu roșu, în același timp fiind afișat un mesaj de atenționare, prezentat în figura următoare.

Figura III.2.1.5. Panoul Alte Informații – starea *editable* – mesaj de eroare

În versiunile viitoare ale produsului software atașat ziua de naștere va fi generată având în vedere structura codului numeric personal, această informație migrând în cadrul panoului Informații de bază, evitând astfel conflictele ce pot apărea din cauza neconcordanței celor două informații.

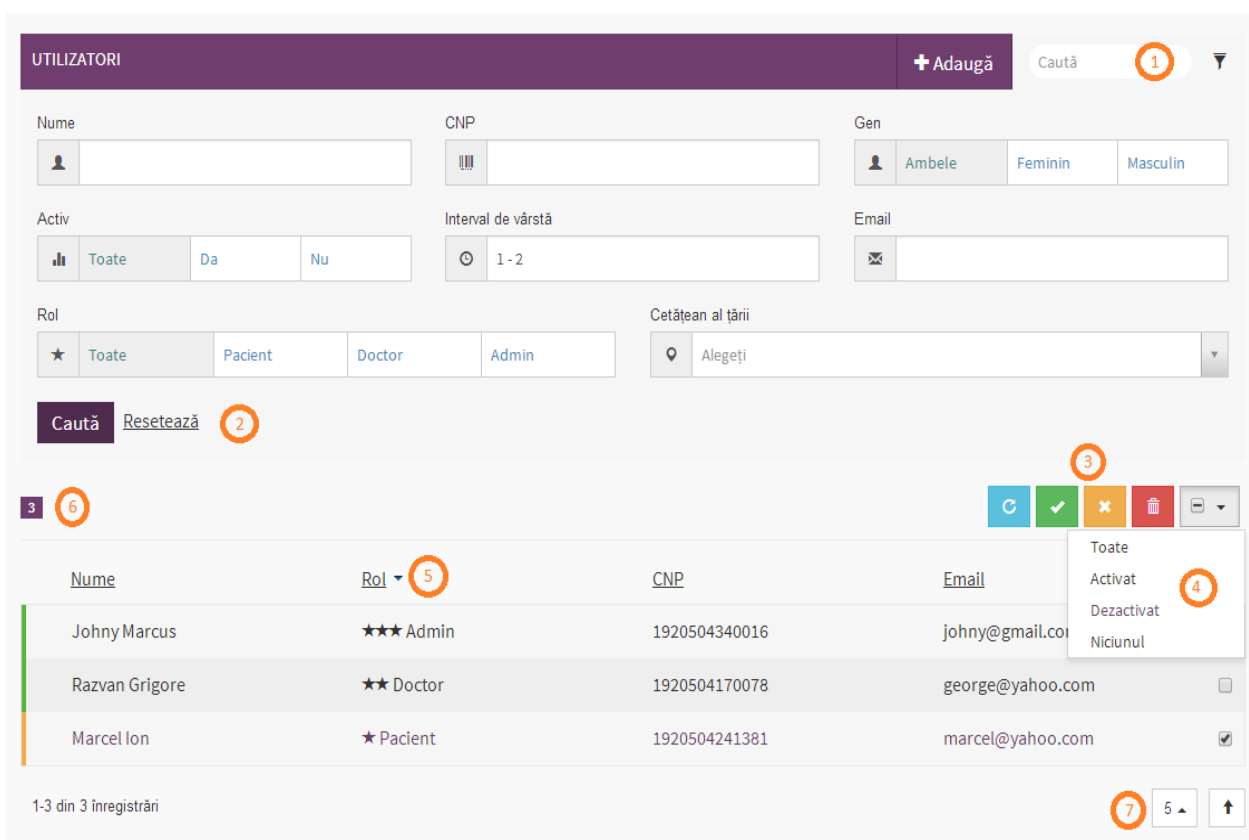
Din punct de vedere tehnic, acest modul a fost dezvoltat folosind șablonul de proiectare ASP.NET *Repository* prezentat în cadrul capitolului IV.

2.2. Modulul de management al utilizatorilor

Unul dintre cele mai frecvente *task-uri* ale unui programator este de a afișa anumite înregistrări sub forma unui *grid* (care facilitează acțiuni precum filtrarea, paginarea sau sortarea înregistrărilor în funcție de anumite criterii). În ASP.NET *WebForms* acest lucru era posibil folosit unul dintre controalele predefinite în *framework* și anume *GridView* care pe lângă acțiunile prezentate anterior oferea posibilitatea de editare și ștergere a înregistrărilor. În ASP.NET MVC acest lucru se poate realiza cu ajutorul clasei *WebGrid*. Componenta de tip *grid* aparținând pachetului *BForms* este o alternativă serioasă la *WebGrid* oferind facilități precum paginare, sortare, filtrare, căutare rapidă, ștergere și editare *in-line* a înregistrărilor. Filtrarea se face pe baza unui formular aflat în cadrul unei componente de tip *toolbar* care este asociată *grid-ului*. *Grid-ul* suportă acțiuni macro (*bulk actions* – ștergere a tuturor înregistrărilor selectate, activarea sau dezactivarea acestora, filtrarea înregistrărilor în funcție de o stare prestabilită de programator sau exportarea înregistrărilor în cadrul unui fișier de tip EXCEL).

Având în vedere că aceste acțiuni pot avea efecte ireversibile, accesul la acest modul este restricționat pentru pacienți. Înregistrările sunt filtrate având în vedere linia ierarhică. Astfel un utilizator al cărui rol este acela de doctor nu va avea acces asupra administratorilor.

Componenta de tip *toolbar* asociată *grid-ului* poate fi modificată în funcție de nevoile programatorului, implicit existând un formular de adăugare, unul pentru filtrarea înregistrărilor și o componentă ce facilitează căutarea rapidă.



UTILIZATORI + Adaugă Caută 1

Nume CNP Gen
 Activ Toate Da Nu Interval de vârstă 1 - 2 Email
 Rol Toate Pacient Doctor Admin Cetățean al țării Alegeți

Caută Resetează 2

3 6

Nume	Rol 5	CNP	Email
Johny Marcus	★★★ Admin	1920504340016	johny@gmail.co
Razvan Grigore	★★ Doctor	1920504170078	george@yahoo.com
Marcel Ion	★ Pacient	1920504241381	marcel@yahoo.com

1-3 din 3 înregistrări 7 5

Figura III.2.2.1. Interfață management utilizatori – rol de administrator

În figura anterioară am prezentat interfața de management a utilizatorilor implementată cu ajutorul componentei de tip *grid* din cadrul pachetului BForms precum și componenta de tip *toolbar* asociată aflată în partea superioară a figurii.

Componenta de tip *toolbar* conține două formulare (unul pentru adăugarea noilor utilizatori, celălalt pentru filtrarea acestora) și o componentă pentru facilitarea căutării rapide (notată în figura III.2.2.1 cu cifra 1). Formularul pentru căutarea utilizatorilor este notat cu cifra 2, oferind posibilitatea filtrării înregistrărilor în funcție de nume, cod numeric personal, gen, email, rol, cetățenie și stare. Resetarea formularului de căutare are ca efect imediat reîntoarcerea componentei de tip *grid* în starea inițială. Acțiunile la nivelul *grid-ului* (resetare, activare, dezactivare, ștergere) sunt numerotate cu 3, aceste *bulk actions* aplicându-se înregistrărilor selectate. Cu ajutorul tabelului numerotat cu 4 putem filtra înregistrările afișate în funcție de starea acestora (de exemplu am selectat doar înregistrările dezactivate). *Grid-ul* se poate sorta în funcție de fiecare coloană definită (în figura anterioară *grid-ul* a fost sortat descrescător în funcție de rol – numărul 5). Numărul total de înregistrări este afișat în pătratul din partea stângă-sus a *grid-ului* (numărul 6), selectarea numărului de înregistrări afișate pe fiecare pagină se realizează cu ajutorul tabelului numerotat cu cifra 7.

Modelul asociat componentei *toolbar* cuprinde modelele celor două formulare (de adăugare și de filtrare a utilizatorilor), iar modelul asociat componentei *grid* se generează pe baza modelului unei înregistrări, definit în figura următoare.

```
public class UserRowModel : BsGridRowModel<UserDetailsModel>
{
    public long Id { get; set; }
    public bool Enabled { get; set; }

    [BsGridColumn(Width = 3)]
    [Display(Name = "Name", ResourceType = typeof(Resource))]
    public string Name { get; set; }

    [BsGridColumn(Width = 3)]
    [Display(Name = "Role", ResourceType = typeof(Resource))]
    public UserRoles Role { get; set; }

    [BsGridColumn(Width = 3)]
    [Display(Name = "CNP", ResourceType = typeof(Resource))]
    public string CNP { get; set; }

    [BsGridColumn(Width = 3)]
    [Display(Name = "Email", ResourceType = typeof(Resource))]
    public string Email { get; set; }

    public override object GetUniqueID()
    {
        return Id;
    }
}
```

Figura III.2.2.2. Modelul asociat stratului de prezentare – înregistrare a componentei *grid*

Proprietățile modelului sunt decorate cu atributul *BsGridColumn* a cărui parametru *width* reprezintă lățimea coloanei, aceasta fiind corelată cu specificațiile *framework-ului Twitter Bootstrap*.

Pentru implementarea acestei construcții avem nevoie de un model central care să conțină modelele asociate componentelor *toolbar* și *grid*.

```
public class UsersPageModel
{
    [BsGrid(HasDetails = true)]
    public BsGridModel<UserRowModel> Grid { get; set; }

    [BsToolbar(Theme = BsTheme.Purple)]
    [Display(Name = "Users", ResourceType = typeof(Resource))]
    public BsToolbarModel<UserSearchModel, UserNewModel> Toolbar { get; set; }
}
```

Figura III.2.2.3. Modelul asociat construcției care definește componentele *grid* și *toolbar*

Cele două componente sunt înglobate cu ajutorul unei metode de tip ajutor *BsGridWrapper*, fiind definite ulterior în cadrul componentelor de tip *view* asociate.

```
@model Hesira.Areas.Common.Models.UsersPageModel

@{
    ViewBag.Title = "Hesira | " + Resource.Users;
    Layout = "~/Areas/Common/Views/Shared/_Layout.cshtml";
}

@using (Html.BsGridWrapper())
{
    @Html.Partial("Toolbar/_Toolbar", Model)

    @Html.Partial("Grid/_Grid", Model)
}
```

Figura III.2.2.4. *Grid Wrapper*

În figura următoare putem observa definiția *builder-ului* pe baza căruia este structurat *grid-ul* utilizatorilor. Principala metodă asociată acestui *builder* este cea pentru definirea coloanelor. Pentru coloana care reprezintă rolul am folosit o metodă de ajutor HTML care generează numărul de stelute asociat celui mai important rol pe care îl deține un utilizator împreună cu denumirea acestui rol. Metoda *Name* setează denumirea coloanei care este selectată în cadrul metodei *For*.

```
@model UsersPageModel

@{
    var builder = new BsGridHtmlBuilder<UsersPageModel, UserRowModel>().ConfigureColumns(cols =>
    {
        cols.For(row => row.Name).Name(Resource.Name)
            .Sortable();

        cols.For(row => row.Role).Name(Resource.Role)
            .Sortable()
            .Text(row => Html.GetRoleIcon(row.Role) + " " + Html.BsEnumDisplayName(row.Role));

        cols.For(row => row.CNP).Name(Resource.CNP)
            .Sortable();

        cols.For(row => row.Email).Name(Resource.Email)
            .Sortable();
    });
}
```

Figura III.2.2.5. *BsGridHtmlBuilder* – utilizat în cadrul componenteii view a *grid-ului*

Variabila de tip *BsGridHtmlBuilder<UsersPageModel, UserRowModel>* declarată anterior este folosită ca parametru pentru generarea componentei de tip *grid*. *UsersPageModel* este modelul care înglobează componenta de tip *grid* și cea de tip *toolbar*. Metodele *NoResultsTemplate* și *NoRecordsTemplate* sunt folosite dacă filtrarea înregistrărilor nu va avea succes, respectiv în momentul în care nu există înregistrări pe care putem să le afișăm. Parametrii acestor metode sunt componente de tip *view* parțiale, care vor fi utilizate în cazurile definite anterior. Cu ajutorul metodei *ConfigureBulkAction* putem defini acțiunile la nivel de *grid* care au fost prezentate în amănunt în cadrul figurii III.2.2.1. Cu ajutorul metodei *ConfigureRows* putem defini setările reprezentative pentru fiecare înregistrare în parte. *HasDetails* este metoda prin care putem seta dacă înregistrarea are asociat un model amănunțit. *HtmlAttributes* este o metoda pentru definirea atributelor de tip HTML asociate fiecărei înregistrări în parte. În cadrul produsul software atașat a fost folosită pentru definirea stării înregistrării (înregistrare activată sau înregistrare dezactivată). Metoda *Highlighter* poate fi folosită pentru colorarea antetului fiecărui rând în parte în funcție de specificațiile acestuia. În cadrul acestui produs, colorarea antetului a fost utilizată pentru a afișa starea curentă a înregistrării. Metoda *PagerSettings* primește ca parametru un obiect de tipul *BsPagerSettings* în care regăsim setările inițiale folosite pentru paginarea *grid-ului*.

```
@(Html.BsGridFor(m => m.Grid, builder)
.NoResultsTemplate("Grid/_NoResults")
.NoRecordsTemplate("Grid/_NoRecords")
.SetTheme(BsTheme.Purple)
.GridResetButton(Resource.Reset)
.ConfigureBulkActions(bulk =>
{
    bulk.AddAction().StyleClass("btn-success js-btn-enable_selected").Title(Resource.Enable).
    GlyphIcon(GlyphIcon.Ok);
    bulk.AddAction().StyleClass("btn-warning js-btn-disable_selected").Title(Resource.Disable).
    GlyphIcon(GlyphIcon.Remove);
    bulk.AddAction(BsBulkActionType.Delete).Title(Resource.Delete);

    bulk.AddSelector(BsBulkSelectorType.All);
    bulk.AddSelector().StyleClass("js-actives").Text(Resource.Enabled);
    bulk.AddSelector().StyleClass("js-inactives").Text(Resource.Disabled);
    bulk.AddSelector(BsBulkSelectorType.None);

    bulk.ForSelector(BsBulkSelectorType.All).Text(Resource.All);
    bulk.ForSelector(BsBulkSelectorType.None).Text(Resource.None);
})
.ConfigureRows(cfg => cfg.HasDetails(row => false)
    .HtmlAttributes(row => new Dictionary<string, object>
    {
        { "data-active", row.Enabled }
    })
    .Highlighter(row => row.Enabled ? "#59b444" : "#f0ad4e")
    .HasCheckbox(row => true))
.PagerSettings(new BsPagerSettings { Size = 5, HasPageSizeSelector = true, HasPagesText = true }));
```

Figura III.2.2.6. Componenta de tip *view* asociată *grid-ului*

În figura următoare avem definită componenta de tip *view* asociată *toolbar-ului*. Definiția este una intuitivă, *toolbar-ul* fiind instanțiat în cadrul metodei *BsToolbarFor*, iar metoda *ConfigureActions* este folosită pentru definirea acțiunilor atașate. Acțiunile definite în figura următoare sunt cele pentru care se oferă suport nativ (filtrare, căutare rapidă și adăugare). Acțiunile de filtrare (căutare avansată) și de adăugare sunt afișate sub forma unor *tab-uri* care au definită o acțiune de *toggle*, după cum am putut observa în figurile anterioare. Metoda *BsPartialPrefixed* este utilizată pentru prefixarea modelului, deci implicit pentru generarea denumirilor câmpurilor create sub forma prefix.denumire (de exemplu *New.FirstName*). Această prefixare este utilizată pentru a profita la maximum de sistemul de *model binding* (Chadwick, 2012).

```
@model Hesira.Areas.Common.Models.UsersPageModel
using BForms.Grid
using BForms.Html
using BForms.Models
using Hesira.Resources

@{(Html.BsToolbarFor(x => x.Toolbar)
    .DisplayName(Resource.Users)
    .SetTheme(BsTheme.Purple)
    .ConfigureActions(ca =>
    {
        ca.Add(BsToolbarActionType.Add)
        .Text(Resource.Add)
        .Tab(x => Html.BsPartialPrefixed(y => y.New, "Toolbar/_New", x));

        ca.Add<BsToolbarQuickSearch>()
        .Placeholder(Resource.Search);

        ca.Add(BsToolbarActionType.AdvancedSearch)
        .Tab(x => Html.BsPartialPrefixed(y => y.Search, "Toolbar/_Search", x));
    })
})
```

Figura III.2.2.7. Componenta de tip *view* asociată *toolbar-ului*

Aceste componente trebuie inițializate în partea *client-side* a produsului, acest proces fiind prezentat în figura următoare. În funcția *_initComponents* am inițializat componentele *grid*, *toolbar* precum și formularele asociate căutării avansate și adăugării utilizatorilor. Metoda *_resetGrid* este utilizată pentru a readuce *grid-ul* în starea inițială eliminând orice tip de filtrare a utilizatorilor sau sortare a acestora.

```

AdminUsersIndex.prototype._ initComponents = function () {
    this.$grid = $(this.selectors.gridContainer);
    this.$toolbar = $(this.selectors.toolbarContainer);
    this.$searchForm = this.$toolbar.find(this.selectors.searchForm).parent();
    this.$newForm = this.$toolbar.find(this.selectors.newForm).parent();
};

AdminUsersIndex.prototype._ initGrid = function () {

    this.$grid.bsGrid(this._bsGridOptions());
    this.$grid.on('click', this.selectors.resetButton, $.proxy(this._resetGrid, this));
};

AdminUsersIndex.prototype._resetGrid = function (ev) {
    ev.preventDefault();
    $(this.selectors.resetSearchBtn).click();
    $(this.selectors.quickSearchBox).val('');
    this.$grid.bsGrid('reset');
};

AdminUsersIndex.prototype._initToolbar = function () {

    this.$toolbar.bsToolbar({
        uniqueName: 'usersToolbar',
        subscribers: [this.$grid],
    });

    this.$searchForm.on('bsformafterinitui', $.proxy(this._initSearchForm, this));

    this.$newForm.on('bsformafterinitui', $.proxy(this._initNewForm, this));

    this.$newForm.on('bsformbeforeformvalidation', $.proxy(this._validationRules, this));
};

```

Figura III.2.2.8. Inițializarea componentelor – *client-side*

Metoda *_initGrid* realizează inițializarea *grid-ului* pe baza opțiunilor definite de programator și adaugă evenimentul de click asociat butonului de resetare. Printre aceste opțiuni se numără componenta de tip *toolbar* asociată, *url-ul* folosit pentru realizarea paginării, sortării sau filtrării, definirea butoanelor de filtrare (prezentate în cadrul figurii III.2.2.1. - Toate, Dezactivat, Activat, Niciunul – numerotate cu cifra 4) și a butoanelor asociate acțiunilor realizate la nivelul *grid-ului* împreună cu evenimentele pe care le declanșează (*bulk actions* – figura III.2.2.1 – numerotate cu cifra 3).

Metoda *_initToolbar* realizează instanțierea *toolbar-ului*, declanșând inițializarea formularelor de adăugare și filtrare.

```

public IQueryable<User> FilterBySearchModel(IQueryable<User> query)
{
    var searchSettings = Settings.Search;

    if (!string.IsNullOrEmpty(Settings.QuickSearch))
    {
        var quickSearch = Settings.QuickSearch.ToLower();

        query =
            query.Where(
                x =>
                x.Email.ToLower().Contains(quickSearch) || x.CNP.ToLower().Contains(quickSearch) ||
                x.Firstname.ToLower().Contains(quickSearch) || x.Lastname.ToLower().Contains(quickSearch) ||
                x.UsersProfiles.FirstOrDefault() != null &&
                x.UsersProfiles.FirstOrDefault().PhoneNumber.ToLower().Contains(quickSearch) ||
                x.UsersProfiles.FirstOrDefault() != null &&
                x.UsersProfiles.FirstOrDefault().Country.CountryNameLowerCase.ToLower()
                    .Contains(quickSearch) ||
                x.UsersProfiles.FirstOrDefault() != null &&
                x.UsersProfiles.FirstOrDefault().Profession.ToLower().Contains(quickSearch) ||
                x.UsersProfiles.FirstOrDefault() != null &&
                x.UsersProfiles.FirstOrDefault().UsersState.State.ToLower().Contains(quickSearch)
            );
    }
}

```

Figura III.2.2.9. Filtrarea înregistrărilor – sistemul căutare rapidă (*quick search*)

Filtrarea utilizatorilor se poate realiza cu ajutorul sistemului de căutare rapidă care va verifica existența cuvântului de căutat în cadrul numelui, adresei de email, numărului de telefon, țării natale, profesiei sau stării utilizatorului în funcție de încadrarea acestuia (pensionar, elev, coasigurat etc.).

UTILIZATORI				+ Adaugă	razvan	▼
2 ▼				C		▼
Nume	Rol	CNP	Email			
Razvan Grigore	★★ Doctor	1841108167599	grigore2000@yahoo.com			
Razvan Dumitru	★★★ Admin	1920504170078	razvan@gmail.com			
1-2 din 2 înregistrări				5 ▲	↑	

Figura III.2.2.10. Filtrarea înregistrărilor – sistemul căutare rapidă (*quick search*) – interfață

O astfel de căutare se realizează introducând cuvântul pe care dorim să-l căutăm în câmpul din dreapta-sus, filtrarea utilizatorilor realizându-se automat în momentul modificării acestui câmp, oferind astfel un acces rapid asupra informațiilor.

Adăugarea utilizatorilor se face cu ajutorul formularului atașat componentei *toolbar*. În concordanță cu rolul pe care îl deținem putem adăuga persoane care să aibă același rol sau un rol cu un nivel ierarhic mai mic (de exemplu: o persoană a cărei rol este acela de doctor, poate adăuga alți doctori parteneri, dar nu poate adăuga o persoană a cărei rol să fie acela de administrator).

Figura III.2.2.11. Adăugarea înregistrărilor noi – interfață

Câmpurile semnalate cu diferențiatorul * sunt câmpuri obligatorii, adăugarea fiind imposibil de realizat fără completarea lor în prealabil. Câmpurile Țară și Cetățean al țării folosesc o componentă de tip select2-query prezentată în subcapitolul dedicat acestei librării. În momentul modificării câmpului Rol putem observa apariția/dispariția câmpului Statut. Acest statut este important doar pentru utilizatorii a căror rol este acela de pacient, definind statutul lor în cadrul CNAS.

Câmpul CNP constituie un identificator uni pentru înregistrările aflate în baza de date, conectarea la această aplicație realizându-se pe baza acestei caracteristici. Astfel, în momentul adăugării, pe lângă mesajele implicite de eroare (câmpuri care sunt obligatorii și nu au fost completate) se mai pot primi mesaje de tipul CNP Invalid sau Codul CNP nu este unic, după cum putem observa în figurile următoare.

Figura III.2.2.12. Unicitatea codului CNP

CNP* CNP Invalid Parola*

1561015111849

Figura III.2.2.13. Validitatea codului CNP

CNP* Acest câmp este obligatoriu Parola*

Figura III.2.2.14. Mesaj atenționare privind câmpurile obligatorii

După adăugarea înregistrării aceasta apare în cadrul *grid-ului*, modificând structura acestuia în cazul în care este nevoie (de exemplu adăugarea înregistrării cu numărul 6 va adăuga o componentă de navigare în partea din stânga jos).

UTILIZATORI				+ Adaugă	Caută	
6						
Nume	RoI	CNP ^	Email			
Geo Ion	★★ Doctor	1561015111840	geo@gf.com			
Victor Amariei	★★ Doctor	1630820126830	victorAm@gmail.com			
Razvan Dumitru	★★★ Admin	1920504170078	razvan@gmail.com			
Marcel Ion	★ Pacient	1920504241381	marcel@yahoo.com			
Maria Cimpaliuc	★★ Doctor	2930508286610	mar@gmail.com			
« ‹ 1 2 › » 1-5 din 6 înregistrări				5 ▲ ↑		

Figura III.2.2.15. Starea *grid-ului* după adăugarea utilizatorului prezentat

Din punct de vedere tehnic acest modul a fost dezvoltat folosind șablonul de proiectare ASP.NET *Dependency Injection* pentru adăugarea utilizatorilor, respectiv *Repository* pentru filtrarea și editarea acestora. Cele două șabloane de proiectare sunt prezentate în cadrul capitolului IV al lucrării de față, împreună cu funcționalitățile pe care le îndeplinesc.

2.3. Modulul de management al programărilor

Pentru a extinde interacțiunea dintre doctori și pacienți, am dezvoltat un modul de management al programărilor. Momentan, fluxul definit în cadrul acestui modul poate fi descris, în linii mari, de următoarele reguli:

- Pacienții au dreptul de a se programa la oricare dintre doctori
- Doctorii nu au dreptul de a refuza o programare
- Data de start a programării este generată automat în funcție de data selectată și de programările curente
- Data de final a programării este generată automat ca fiind data de start plus 30 de minute
- Dacă nu se mai pot face programări pentru o anumită dată, pacientul va fi atenționat
- Doctorii și pacienții își pot observa programările curente în cadrul modulului
- Programările sunt generate în intervalul orar 08:00 – 19:00

În continuare am creat un scenariu pentru a exemplifica funcționalitățile descrise.

Ioana Nitulescu se va programa la medicul Maria Cimpaliuc pentru data de 09 Iunie 2014, ora programării generându-se automat ca fiind ora 08:00, deoarece nu existau alte programări active pentru această dată. În același timp se va programa la medicul Ioana Mureșan, ora programării fiind 08:30. Cele două programări sunt exemplificate în figurile următoare.

The screenshot shows a web application interface for managing appointments. At the top, there is a blue header bar with a menu icon on the left and the user's name 'Ioana Nitulescu' on the right. Below the header, the main content area is titled 'Programări'. Under this title, there is a section labeled 'Informații de bază' (Basic Information). This section contains two form fields: 'Doctor*' (Doctor) with a dropdown menu showing 'Maria Cimpaliuc', and 'Data*' (Date) with a text input showing '09.06.2014'. At the bottom of the form, there are two buttons: a blue 'Salvează' (Save) button on the left and a red 'Resetează' (Reset) button on the right.

Figura III.2.3.1. Programare doctor Maria Cimpaliuc – pacient Ioana Nitulescu

Programări

Informații de bază

Doctor*

Ioana Muresan

Data*

09.06.2014

Salvează

Resetează

Figura III.2.3.2. Programare doctor Ioana Muresan – pacient Ioana Nitulescu

În cadrul acestui modul, pacientul va putea vizualiza programările în cadrul unei componente de tip calendar (momentan se afișează programările pentru ziua curentă în format agendă).

Appointments

Monday 6/9

8am	8:00 - Maria Cimpaliuc
	8:30 - Ioana Muresan
9am	
10am	
11am	
12pm	

Figura III.2.3.3. Ioana Nitulescu – pacient – programări pentru ziua de luni, 9 Iunie 2014

Asemenea pacienților, doctorii pot vizualiza programările pentru ziua curentă. În continuare vom prezenta programările doctorului Ioana Muresan, printre care o vom regăsi și pe Ioana Nitulescu.

Appointments

Monday 6/9

8am	
	8:30 - Ioana Nitulescu
9am	9:00 - Ion Marin
10am	
11am	

Figura III.2.3.4. Ioana Muresan – doctor – programări pentru ziua de luni, 9 Iunie 2014

2.4. Modulul de management al rețetelor

În cadrul acestui modul, doctorii pot seta starea unei programări ca fiind finalizată, completând diagnosticul, starea curentă a pacientului și adăugând, atunci când situația o necesită, metoda de tratament.

În continuare vom prezenta fluxul operațional, care este descris de următoarele reguli:

- Doctorii pot finaliza programările a căror stare este în desfășurare
- Informațiile de bază referă date despre programare, diagnostic și istoric medical
- Rețeta se poate concepe în cadrul componentei prezentate în figura III.2.4.2.
- Doctorii pot selecta medicamentele, împreună cu numărul de unități și numărul de zile de administrare



Figura III.2.4.1. Informații de bază – managementul rețetelor

Legenda interfeței utilizator a modulului curent:

- 1 – câmpul utilizat pentru căutarea rapidă a medicamentelor. Textul va fi căutat în cadrul denumirii comerciale a produsului și în cadrul denumirii importatorului
- 2 – sistemul de paginare al medicamentelor
- 3 – panoul de management al medicamentelor selectate (se poate modifica numărul de zile de administrare și cantitatea)
- 4 – resetarea panoului de management al medicamentelor
- 5 – salvarea formularului
- 6 – resetarea formularului

Figura III.2.4.2. Rețetă – managementul rețetelor

2.5. Modulul de vizualizare a istoricului medical

Continuând scenariul prezentat în cadrul modulului de management al rețetelor, Ion Marin va avea posibilitatea de a-și vedea programarea finalizată, în cadrul unui istoric medical propriu.

Programările finalizate sunt de două tipuri, cu rețetă atașată sau controale de rutină. Pacientul își va putea observa istoricul medical împreună cu toate medicamentele care i-au fost prescrise, aceasta fiind una dintre funcționalitățile cele mai importante a acestui produs.

Figura III.2.5.1. Istoric medical – pacientul Ion Marin

În cadrul figuri anterioră putem observa doctorul care a susținut consultația, în persoana Mariei Cimpaliuc, data de start și data de sfârșit a consultației, păreri medicului în momentul expertizei, precum și rețeta prescrisă pacientului.

3. Dezvoltare ulterioară

Dezvoltarea ulterioară a acestui produs software presupune extinderea funcționalităților curente, precum și modificarea structurii produsului pentru a se adapta la șablonul de proiectare *Dependency Injection* prezentat în cadrul capitolului IV.

Adaptarea presupune crearea unui serviciu pentru fiecare entitate cu care se operează în cadrul produsului (de exemplu: programări sau medicamente) care să funcționeze precum un API căruia îi putem cere informații sau căruia îi putem comanda ștergerea, editarea sau crearea înregistrărilor.

Șablonul de proiectare *Repository* (în forma prezentată în cadrul subcapitolului IV.2) nu favorizează testabilitatea produsului, acesta fiind unul din cele mai importante motive pentru modificarea ulterioară a implementării curente.

Funcționalități ulterioare:

- definirea unui sistem complet de securitate, pentru adăugarea, editarea și ștergerea utilizatorilor
- sporirea interacțiunii dintre doctor și pacient în cadrul modulului de management al programărilor
- implementarea unui modul de management al documentelor
- corelarea modulului de management al documentelor cu celelalte module (atașarea fișierelor în momentul generării rețetei, vizualizarea acestora în cadrul istoricului medical)
- generarea rețetelor în format electronic (semnătură digitală, generare CUID, integrarea datelor din cadrul sistemului informatic al CNAS, etc.)

IV. ȘABLOANE DE PROIECTARE ASP.NET & JAVASCRIPT

1. RequireJS & șablonul de proiectare JavaScript – AMD

Având în vedere ca aplicațiile web devin din ce în ce mai complexe și logica operațională se mută spre partea de client, un manager al resurselor de tip *JavaScript* și/sau *CSS* este vital pentru a susține un produs software excepțional. Momentan pentru limbajul *JavaScript* nu exista un manager complex al pachetelor care să aibă implementate concepte precum *lazy-loading* (încărcarea la nevoie), încărcare asincronă, minimizarea pachetelor, compilarea acestora într-un singur fișier etc.

RequireJS permite încărcarea fișierelor și modulelor *JavaScript*. Poate fi utilizat în browser sau în cadrul unui server *node.js* sau împreună cu *Java/Rhino* (Shepherd, 2014). Resursele, precum și dependențele aferente trebuie definite, încărcarea lor fiind asincronă, utilizând un sistem de tipul *lazy-loading* (încărcare târzie, adică doar dacă este nevoie).

RequireJS folosește un șablon de proiectare *JavaScript* foarte puternic, denumit AMD (*Asynchronous Module Definition*). Șablonul de proiectare Module se referă la utilizarea funcțiilor *JavaScript* pentru abstractizarea noțiunii de incapsulare.

Această nevoie de a modulariza, precum și acest șablon de proiectare, provin din dorința dezvoltatorilor de a elimina managementul manual al *tag-urilor script* utilizate, ceea ce implica și ordonarea dependențelor.

Șablonul de proiectare AMD are următoarele caracteristici:

- utilizează o listă de identificatori unici de tip text pentru a defini dependențele unui modul, eliminând astfel declararea dependențelor și utilizarea globală a *script-urilor*
- acești identificatori unici pot fi corelați cu diferite fișiere, existând posibilitatea de a modifica implementarea curentă a acelui modul cu o versiune anterioară, de exemplu
- incapsulează noțiunea de modul

CommonJS (în trecut *ServerJS*) este utilizat pentru definirea modulelor *JavaScript* atunci când acestea sunt utilizate în afara *browser-ului*. Între *RequireJS*, șablonul de proiectare AMD și modul în care a fost definit *CommonJS* există o strânsă legătură, deși *CommonJS* nu a fost creat utilizând conceptul de browser (Community, 2013).

În cadrul produsului software atașat am folosit pachetul *NuGet* RequireJs.Net care integrează *framework-ul* *RequireJS* cu ASP.NET MVC utilizând fișiere de tip XML pentru definirea dependențelor, clase de ajutor pentru utilizarea obiectelor de tip JSON, precum și o componentă de tip *controller* care trebuie extinsă de fiecare *controller* în parte.

```
public class BaseController : RequireJS.RequireJsController
{
    public HesiraEntities HesiraDB = new HesiraEntities();

    public override void RegisterGlobalOptions()
    {
        RequireJsOptions.Add(
            "homeUrl",
            Url.Action("Index", "Home", new { area = "" })),
            RequireJsOptionsScope.Global);

        RequireJsOptions.Add(
            "toggleSidebarUrl",
            Url.Action("ToggleSidebar", "Ajax", new { area = "" })),
            RequireJsOptionsScope.Global);
    }
}
```

Figura IV.1.1. *BaseController*

În figura anterioară avem prezentat *controller-ul* de bază care moștenește *controller-ul* definit în cadrul pachetului RequireJS.NET. În metoda *RegisterGlobalOptions* am transmis într-un scop global cele două *url-uri* (cel pentru *main-page-ul* produsului și cel utilizat pentru *toggle-ul* componentei *sidebar*), deoarece sunt utilizate în cadrul fiecărei pagini a produsului.

```
<configuration>
  <paths>
    <path key="adminLTE" value="components/AdminLTE/adminLTE" />
    <path key="initLTE" value="components/AdminLTE/initLTE" />
  </paths>
  <shim>
    <dependencies for="initLTE" exports="">
      <add dependency="jquery"/>
      <add dependency="bootstrap" />
      <add dependency="jquery-ui-core" />
      <add dependency="adminLTE" />
    </dependencies>

    <dependencies for="adminLTE" exports="">
      <add dependency="jquery"/>
      <add dependency="bootstrap" />
      <add dependency="jquery-ui-core" />
    </dependencies>
  </shim>
</configuration>
```

Figura IV.1.2. *RequireJs.config*

În figura anterioară este prezentat un fișier XML utilizat pentru definirea modulelor, precum și a dependențelor acestora. Cele două module prezentate sunt utilizate pentru integrarea funcționalităților de bază ale temei *AdminLTE* în cadrul produsului software atașat utilizând *RequireJS*.

Pentru a exemplifica șablonul de proiectare *Module*, vom prezenta modulul responsabil pentru inițializarea funcționalităților native ale temei *AdminLTE*.

```
require([
    'adminLTE',
    'jquery',
    'jquery-ui-core',
    'bootstrap',
    'bforms-ajax'
], function () {

    var AdminLTE = function () {

    };

    // fix slimscroll
    AdminLTE.prototype._fixSidebar = function (...);
    // fix css - check height
    AdminLTE.prototype._fix = function (...);
    // init sidebar & toolbar components
    AdminLTE.prototype._initComponents = function (...);
    // sidebar toggle event success
    AdminLTE.prototype._ajaxToggleSidebarSuccess = function (response, callbackData)...;
    // sidebar toggle event fail
    AdminLTE.prototype._ajaxToggleSidebarError = function(response, callbackData)...;
    // init function
    AdminLTE.prototype._init = function (...);
    // init current module when document is ready
    $(document).ready(function () {
        var ctrl = new AdminLTE();
        ctrl._init();
    });
});
```

Figura IV.1.3. *initLTE Module*

Dependențele sunt declarate în cadrul unui *array* de identificatori unici, acest modul având ca dependențe modulul minimizat *AdminLTE*, librăriile *jQuery* și *jQuery UI*, modulul minimizat *Bootstrap* deoarece produsul este dezvoltat folosind acest *framework*, precum și *bforms-ajax* care este o componentă de tip ambalaj (*wrapper*) pentru metoda *jQuery ajax*.

Modulul este structurat folosind șablonul de proiectare *prototype*, metodele declarate fiind utilizate pentru instanțierea temei *AdminLTE* (*sidebar, box scrolling etc.*).

2. Șablonul de proiectare ASP.NET – Repository

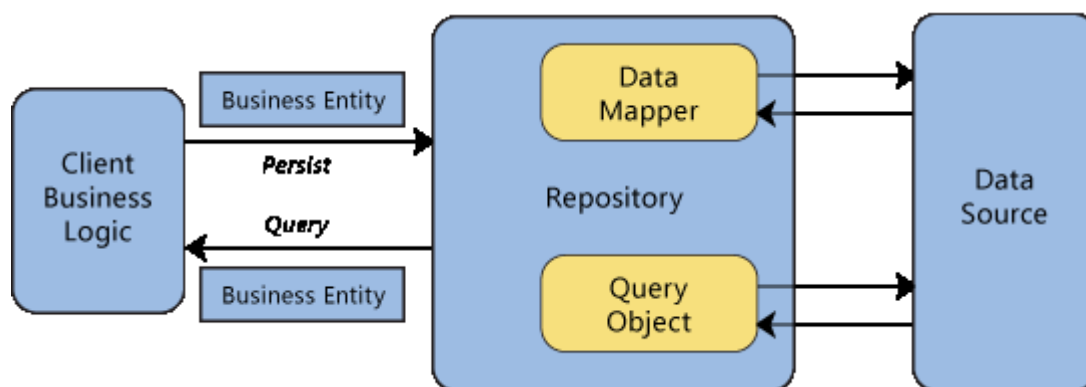


Figura IV.2.1. Șablonul de proiectare Repository (MSDN)

În figura anterioară este prezentată structura unui produs software care implementează șablonul de proiectare *Repository*. Crearea unui strat a cărui responsabilitate principală este accesul la date, este vitală. Componentele constitutive ale acestui strat vor implementa metode de conversie, interogare, fiind în același timp responsabile cu persistența și alterarea datelor.

Componenta de tip controller nu va cuprinde logică operațională, implementând doar șabloane de tipul *request-response* pentru a facilita interacțiunea cu utilizatorul final. De aceste răspunsuri vor fi responsabile toate componentele de tip *repository* folosite în cadrul acelei acțiuni.

Structura produsului software asociat acestei lucrări a fost adaptată acestui șablon, fiind împărțită în trei straturi: modelul de date care este reprezentat de clasele generate cu ajutorul *Entity Framework*, modelul de prezentat utilizat pentru expunerea interfețelor utilizator, stratul reprezentat de componentele de tip *repository* care mediază legătura dintre cele două modele.

În continuare vom exemplifica acest șablon prezentând funcționalitățile *grid-ului* din cadrul modulului de management al utilizatorilor, mai precis, filtrarea și alterarea înregistrărilor asociate acestui *grid*.

Am creat o componenta de tip *repository* de bază care este responsabilă de inițializarea contextului bazei de date. Celelalte componente de acest tip extind *repository-ul* de bază.

Clasa *UserRepository* este responsabilă cu accesul la datele afișate în cadrul *grid-ului* din modulul de management al utilizatorilor, precum și de filtrarea, ordonarea și conversia acestora între modelele de tip *domain* și cele de prezentare utilizate în cadrul interfețelor utilizator.

```

#region Query

public override IQueryable<User> Query()
{
    var query = db.Users.AsQueryable();
    return Filter(query);
}

#endregion

#region Filter

public IQueryable<User> Filter(IQueryable<User> query)
{
    query = FilterByRole(query);
    query = FilterBySearchModel(query);
    return query;
}

public IQueryable<User> FilterByRole(IQueryable<User> query) ...
public IQueryable<User> FilterBySearchModel(IQueryable<User> query) ...

#endregion

#region Order

public override IOrderedQueryable<User> OrderQuery(IQueryable<User> query) ...

#endregion

#region Map

public override IEnumerable<UserRowModel> MapQuery(IQueryable<User> query)
{
    var model = query.Select(Map_User_UserRowModel).ToList();
    return model;
}

```

Figura IV.2.2. Componenta de tip *Repository* asociată entității utilizator

În figura anterioară sunt prezentate etapele creării listei de utilizatori care va fi afișată în cadrul *grid-ului*.

Metoda *Query* inițializează interogarea și o transmite ca parametru către metoda de filtrare. Aici este filtrată în funcție de rolul pe care îl are persoana care interoghează (administrator sau medic) introducând primele reguli de *business* (utilizatorii ai căror rol este acela de medic nu pot acționa asupra administratorilor) . În metoda *FilterBySearchModel* vom filtra utilizatorii pe baza câmpurilor completate în cadrul formularului de căutare prezentat în modulul de management al utilizatorilor. Apoi urmează metoda de ordonare, acest ciclu finalizându-se cu conversia înregistrărilor într-o listă de modele de prezentare și enumerarea acestei interogări. Conversia este realizată cu ajutorul *mapper-ului* prezentat în imaginea următoare.

```

public Expression<Func<User, UserRowModel>> Map_User_UserRowModel = x => new UserRowModel
{
    Id = x.Id,
    Enabled = x.Enabled,
    Name = x.Firstname + " " + x.Lastname,
    Email = x.Email,
    CNP = x.CNP,
    Role = x.IsAdmin ? UserRoles.Admin : (x.IsDoctor ? UserRoles.Doctor : UserRoles.Patient)
};

```

Figura IV.2.3. Componenta de tip *Repository* asociată entității utilizator

3. Șablonul de proiectare ASP.NET – Unit of Work

Unit of Work este un șablon utilizat pentru realizarea accesului la date. Din punct de vedere conceptual este un șablon care menține o listă de obiecte care au fost modificate în cadrul unei tranzacții. Modificarea poate consta din adăugarea, ștergerea sau editarea acestora.

Responsabilitatea acestui șablon este de a asigura integritatea datelor, folosind un sistem de *rollback* în cazul în care tranzacția curentă eșuează.

Șablonul de proiectare descris anterior a fost folosit în cadrul modulului de management al utilizatorilor în cadrul acțiunii de creare.

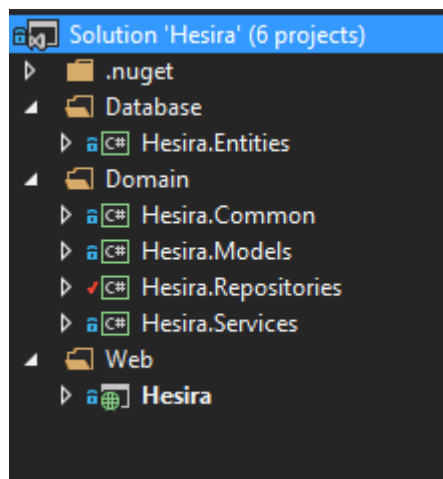


Figura IV.3.1. Structura produsului software

Structura curentă a produsului software atașat este cea prezentată în figura IV.3.1. În directorul *Database* avem proiectul *Hesira.Entities* unde se află fișierul de tip EDMX generat de *Entity Framework*. În directorul *Domain* avem, în ordinea denumirii, următoarele proiecte:

- *Hesira.Common* – definirea claselor comune tuturor serviciilor
- *Hesira.Models* – definirea claselor care constituie modelele de tip *domain*
- *Hesira.Repositories* – definirea claselor în care se realizează convertirea dintre modele *domain* și modele *entity* (generate de *Entity Framework*), precum și a claselor care facilitează accesul la baza de date
- *Hesira.Services* – definirea serviciilor atașate fiecărei entități cu care operăm

În directorul *Web* avem proiectul *Hesira* care este de fapt produsul web (*user interface*).

Structura descrisă anterior facilitează modificarea cu ușurință a interfeței, accesul la date fiind decuplat de modul în care acestea sunt prezentate utilizatorului. Produsul poate fi modificat cu ușurință dintr-un produs web într-un produs desktop.

În continuare vom prezenta implementarea acestui șablon de proiectare software în cadrul produsului asociat lucrării de față.

Pentru început vom defini modelele de tip *domain* care sunt modele conceptuale, asemănătoare modelelor generate de *Entity Framework*, și care sunt folosite pentru a nu expune modelele *entity* altor straturi ale produsului în afară de cel din care fac parte (*data layer*) – *Domain Model Pattern*.

```
public class UserProfileModel
{
    public long Id { get; set; }

    // Id-ul deținătorului acestui profil
    public long Id_User { get; set; }

    // Ziua de naștere
    public DateTime BirthDay { get; set; }

    // Id-ul entității de tip adresă
    public long? Id_Address { get; set; }

    public string PhoneNumber { get; set; }

    public int Gender { get; set; }

    public string Profession { get; set; }

    // Id-ul țării a cărei cetățean este
    public int Id_Citizenship { get; set; }

    // Statusul CNAS
    public int? Id_State { get; set; }

    // Modelul domain pentru Adresă
    public AddressModel Address { get; set; }
}
```

Figura IV.3.2. *UserProfileModel* – model de tip *domain* pentru profilul utilizatorului

Pentru a putea utiliza aceste modele în cadrul produsului software avem nevoie de metode de conversie. În figura următoare este prezentată o astfel de metodă de conversie între modelul *entity* denumit *UsersProfile* și modelul *domain* *UserProfileModel*.

```
public static void ToDomainModel(this UsersProfile dbModel, UserProfileModel domainModel)
{
    domainModel.Id = dbModel.Id;
    domainModel.Birthday = dbModel.Birthday;
    domainModel.Gender = dbModel.Gender;
    domainModel.Id_Address = dbModel.Id_Address;
    domainModel.Id_Citizenship = dbModel.Id_Citizenship;
    if (dbModel.Id_State.HasValue)
    {
        domainModel.Id_State = dbModel.Id_State.Value;
    }
    domainModel.Id_User = dbModel.Id_User;
    domainModel.PhoneNumber = dbModel.PhoneNumber;
    domainModel.Profession = dbModel.Profession;
    if (dbModel.Id_Address.HasValue)
    {
        dbModel.Address.ToDomainModel(domainModel.Address);
    }
}
```

Figura IV.3.3. Metoda conversie model de tip *entity* – model de tip *domain*

În continuare am folosit un șablon de proiectare care se numește *Factory* (Gang of Four, 1994), utilizat pentru abstractizarea stratului corespunzător bazei de date, favorizând astfel adaptabilitatea produsului la schimbarea tipului bazei de date folosite (MySQL, MS SQL).

```
public class DBFactory : IDBFactory, IDisposable
{
    private HesiraEntities db;

    // implement IDisposable
    public void Dispose()
    {
        if (db != null)
            db.Dispose();
    }

    // current context or create another
    public HesiraEntities Create()
    {
        return db ?? (db = new HesiraEntities());
    }
}
```

Figura IV.3.4. DBFactory

Clasa definită anterior este responsabilă pentru crearea contextului bazei de date, implementând metoda *Create* a interfeței *IDBFactory*.

În continuare vom folosi șablonul de proiectare *Repository*, pentru a defini un nou nivel de abstractizare a accesului la date.

Pentru început vom declara o interfață de bază care va fi implementată de interfața asociată fiecărei clase de tip *repository*.

```
public interface IEntityRepository<TDomainModel> where TDomainModel : class
{
}
}
```

Figura IV.3.5. *IEntityRepository* – Interfață de bază pentru componentele de tip *repository*

Având în vedere nivelul de abstractizare al fiecărei clase de tip *repository*, am ales ca această interfață de bază să nu conțină metode de implementat.

Pentru a avea acces la baza de date, fiecare clasă de tip *repository* va trebui să fie derivată din *BaseContextRepository*.

```
public abstract class BaseContextRepository
{
    protected internal HesiraEntities db;

    protected BaseContextRepository(IDBFactory dbFactory)
    {
        db = dbFactory.Create();
    }
}
```

Figura IV.3.6. *BaseContextRepository*

În continuare vom prezenta clasa de tip *repository* asociată modelului de tip *domain UserProfileModel*, precum și interfața pe care o implementează.

```
public interface IUserProfileRepository : IEntityRepository<UserProfileModel>
{
    UserProfileModel GetProfileForUser(long userId);
    long InsertOrUpdate(UserProfileModel model);
}
```

Figura IV.3.7. *IUserProfileRepository*

Metoda *GetProfileForUser* este folosită pentru a obține profilul unui utilizator pe baza identificadorului unic al acestuia, iar metoda *InsertOrUpdate* va fi utilizată pentru salvarea sau editarea profilului unui utilizator.

```

public class UserProfileRepository : BaseContextRepository, IUserProfileRepository
{
    #region Constructor
    public UserProfileRepository(IDBFactory dbFactory)
        : base(dbFactory)
    {
    }
    #endregion

    public UserProfileModel GetProfileForUser(long userId)...

    public long InsertOrUpdate(UserProfileModel model)
    {
        var userProfile = new UsersProfile();

        if (model.Id > 0)
        {
            userProfile = db.UsersProfiles.SingleOrDefault(x => x.Id == model.Id);

            model.ToDatabaseModel(userProfile);
        }
        else
        {
            userProfile.Timestamp = DateTime.Now;
            model.ToDatabaseModel(userProfile);
            db.UsersProfiles.Add(userProfile);
        }

        db.SaveChanges();

        return userProfile.Id;
    }
}

```

Figura IV.3.8. *UserProfileRepository*

În cadrul constructorului vom inițializa contextul bazei de date apelând constructorul clasei *BaseContextRepository* definită în figura IV.3.6. Metoda *InsertOrUpdate* va folosi metoda de conversie către modelul de tip *entity* pentru a salva modelul primit ca parametru, sau pentru a modifica modelul *entity* asociat dacă acesta există deja în baza de date.

În continuare vom prezenta implementarea șablonului de proiectare *Unit of Work*. Clasa responsabilă de integritatea datelor va trebui să implementeze următoarea interfață.

```

public interface IUnitOfWork : IDisposable
{
    void BeginTransaction();

    void Commit();
}

```

Figura IV.3.9. *UserProfileRepository*

```

public class UnitOfWork : IUnitOfWork
{
    private readonly IDBFactory _dbFactory;
    private readonly IsolationLevel _isolationLevel = IsolationLevel.Serializable;
    private DbContext _dbContext;
    private DbContextTransaction _transaction;
    private bool _disposed;

    protected DbContext DBContext...

    public UnitOfWork(IDBFactory dbFactory)
    {
        _dbFactory = dbFactory;
        _isolationLevel = IsolationLevel.ReadCommitted;
    }

    public void BeginTransaction()
    {
        _transaction = DBContext.Database.BeginTransaction(_isolationLevel);
    }

    public void Commit()
    {
        if (_transaction != null)
        {
            _transaction.Commit();
            _transaction.Dispose();
            _transaction = null;
        }
        else
        {
            throw new Exception("You are not in a transaction");
        }
    }

    protected virtual void Dispose(bool disposing)...

    public void Dispose()...
}

```

Figura IV.3.10. *UnitOfWork*

În figura anterioară putem observa implementarea interfeței *IUnitOfWork*. Contextul curent al bazei de date (*DbContext*) este inițializat ca fiind contextul definit (*_dbContext*) sau un nou context creat cu ajutorul metodei *_dbFactory.Create*, în cazul în care contextul definit este nul.

Metoda *BeginTransaction* va crea o tranzacție în contextul definit de noi folosind ca *isolation level* parametrul setat. În cadrul acestui produs am folosit ca *isolation level* *ReadCommitted* pentru a putea citi datele care au fost scrise de alte tranzacții care nu s-au finalizat.

Metoda *Commit* va salva modificările din tranzacția curentă și o va închide, finalizând cu succes tranzacția.

În cadrul produsului software asociat lucrării de față am folosit șablonul de proiectare *Unit of Work* pentru salvarea unui nou utilizator. Metoda *Create* face parte din serviciul atașat entității utilizator pe care îl voi prezenta în cadrul subcapitolului IV.4.

```
public long Create(UserModel userModel, UserProfileModel userProfileModel, AddressModel userAddressModel)
{
    long? userId = null;

    unitOfWork.BeginTransaction();

    if (userAddressModel != null && userProfileModel != null)
    {
        var addressId = _addressRepository.InsertOrUpdate(userAddressModel);
        userProfileModel.Id_Address = addressId;
    }

    userId = _userRepository.InsertOrUpdate(userModel);

    if (userProfileModel != null)
    {
        userProfileModel.Id_User = userId.Value;
        _userRepository.InsertOrUpdate(userProfileModel);
    }

    unitOfWork.Commit();

    return userId.Value;
}
```

Figura IV.3.11. Metoda *Create* – *UserService*

Metodele *InsertOrUpdate* asociate adresei, utilizatorului și profilului acestuia sunt înglobate în cadrul tranzacției definite cu ajutorul obiectului *unitOfWork*. Vom reține la fiecare pas entitățile salvate, sistemul de *rollback* activându-se în cazul în care salvarea uneia din entități eșuează. Presupunem că salvarea entității utilizator eșuează. Tranzacția nu va mai ajunge la metoda de *Commit*, eliminând entitatea adresă pe care tocmai o salvasem (*rollback*), revenind la starea inițială.

4. Șablonul de proiectare ASP.NET – Dependency Injection

În subcapitolul trecut am prezentat modul în care este definit accesul la date precum și persistența acestora în cadrul unor modele de tip *domain* folosit *Domain Model Pattern*. În completarea acestei structuri vom defini modul în care funcționează serviciile declarate.

Situat între stratul de logică operațională și cel de prezentare, stratul de servicii oferă clientului operațiile pe care acesta le poate accesa și definește, în linii mari, structura aplicației, precum și limitele acesteia.

Rolurile stratului de servicii:

- furnizarea entităților participante la fluxul operațional după ce acestea au fost validate și au trecut prin toate etapele fluxului
- furnizarea unor metode pentru alterarea datelor (adăugare, editare, ștergere) care ar trebui să implementeze șabloane de proiectare care să faciliteze interacțiunea dintre serviciu și client (*Request-Response Pattern* - mesaje)

Structura unui serviciu trebuie să fie simplă, să altereze doar datele de care este responsabil și să răspundă la întrebări referitoare la aceste entități (va utiliza componentele de tip *repository* asociate entităților pe care le alterează sau despre care trebuie să știe că există). Metodele unui serviciu ar trebui să fie decuplate și fiecare în parte să reprezinte o tranzacție separată. Această modularitate a structurii serviciului îi furnizează clientului posibilitatea de a realiza tranzacții în orice ordine și/sau mod, acesta nefiind nevoit să acopere implicațiile relaționale (de exemplu: metoda *CreateOrInsert* din serviciul asociat entității utilizator va altera sau persista date referitoare la identitatea acestuia, adresă, profil asociat).

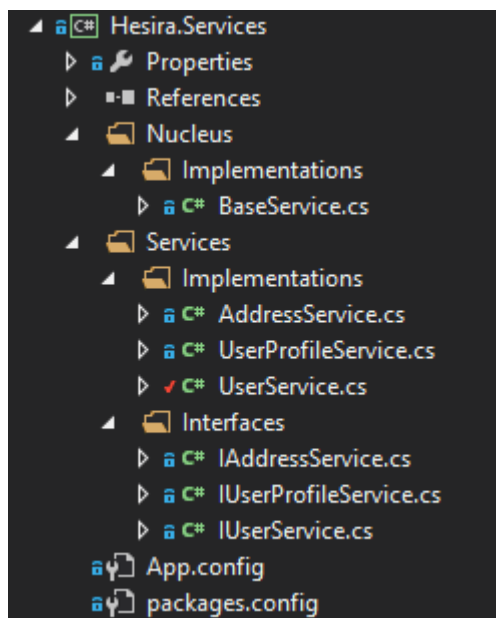


Figura IV.4.1. *Hesira.Services* – Structură

În figura anterioară putem observa structura stratului de servicii din cadrul produsului software asociat acestei lucrări. Fiecare serviciu implementează serviciul de bază și interfața asociată.

Serviciul de bază este responsabil de instanțierea obiectului *unitOfWork*, utilizat pentru crearea de tranzacții SQL, și trebuie să implementeze interfața predefinită *IDisposable*.

În continuare vom prezenta serviciul atașat entității utilizator. Acest serviciu este utilizat în cadrul modulului de management al utilizatorilor pentru salvarea unei noi înregistrări (poate fi utilizat și pentru alterarea unei entități de tip utilizator, acest lucru constituind o etapă din dezvoltarea ulterioară a acestui produs).

```
public interface IUserService
{
    IEnumerable<UserModel> GetAll();
    long Create(UserModel userModel, UserProfileModel userProfileModel, AddressModel userAddressModel);
}
```

Figura IV.4.2. *IUserService*

```
public class UserService : BaseService, IUserService
{
    #region Constructor & Properties

    private readonly IUserRepository _userRepository;
    private readonly IAddressRepository _addressRepository;
    private readonly IUserProfileRepository _userProfileRepository;

    public UserService(IUserRepository userRepository, IAddressRepository addressRepository,
        IUserProfileRepository userProfileRepository, IUnitOfWork unitOfWork)
        : base(unitOfWork)
    {
        this._addressRepository = addressRepository;
        this._userProfileRepository = userProfileRepository;
        this._userRepository = userRepository;
    }

    #endregion

    public IEnumerable<UserModel> GetAll()...
    public long Create(UserModel userModel, UserProfileModel userProfileModel, AddressModel userAddressModel)...
}

class Hesira.Models.User.UserModel
```

Figura IV.4.3. *UserService*

Clasa *UserService* extinde serviciul de bază în cadrul căruia este definit obiectul de tip *unitOfWork*, implementând în același timp interfața asociată. În cadrul stratului de servicii moștenirile sunt interzise pentru a păstra structura modulară, precum și decuplarea acestora.

Fiecare serviciu va ști de existența componentelor de tip *repository* pe care le utilizează în cadrul metodelor definite, precum și de existența componentei *unitOfWork*.

În cadrul acestui produs software am folosit abordarea *constructor injection*. Constructorul serviciului prezentat în figura anterioară primește ca parametru toate componentele de tip *repository* care sunt utilizate de metodele pe care le implementează, precum și componenta de tip *unitOfWork* utilizată pentru crearea tranzacțiilor SQL, înglobând logica descrisă de componentele de tip *repository*. De instanțierea componentei *unitOfWork* este responsabil serviciul de bază.

Pentru rezolvarea dependențelor, în cadrul acestui produs software, am folosit librăria *Ninject* (Kohari, 2007) via *NuGet*. Această librărie cuprinde o serie de extensii utile pentru dezvoltarea aplicațiilor web. Una dintre aceste extensii este prezentată în figura următoare.

```
/// <summary>
/// Load your modules or register your services here!
/// </summary>
/// <param name="kernel">The kernel.</param>
private static void RegisterServices(IKernel kernel)
{
    #region Nucleus

    kernel.Bind<IDBFactory>().To<DBFactory>().InRequestScope();
    kernel.Bind<IUnitOfWork>().To<UnitOfWork>().InRequestScope();

    #endregion

    #region Services

    kernel.Bind<IUserService>().To<UserService>().InRequestScope();
    kernel.Bind<IAddressService>().To<AddressService>().InRequestScope();
    kernel.Bind<IUserProfileService>().To<UserProfileService>().InRequestScope();

    #endregion

    #region Repositories

    kernel.Bind<IUserRepository>().To<UserRepository>().InRequestScope();
    kernel.Bind<IUserProfileRepository>().To<UserProfileRepository>().InRequestScope();
    kernel.Bind<IAddressRepository>().To<AddressRepository>().InRequestScope();

    #endregion

}
```

Figura IV.4.4. *NinjectWebCommon*

Pachetul *NuGet Ninject* va crea în cadrul directorului *App_Start* o clasă utilizată pentru definirea specificațiilor generale ale acestui injector de dependențe. În figura anterioară putem observa metoda principală a acestei clase.

Fiecare regiune prezintă o abstractizare diferită a produsului software atașat. În cadrul regiunii *Nucleus* avem corelarea interfețelor *IDBFactory* și *IUnitOfWork* de clasele care le implementează, oferind astfel un acces unic la date în scopul definit de o cerere HTTP. Având în vedere că toate componentele de tip *repository* referă același *unitOfWork* sensul unei tranzacții de tip SQL este conturat. În cadrul celorlalte două regiuni sunt definite corelațiile dintre servicii și interfețele care le definesc, precum și corelațiile dintre componentele de tip *repository* și interfețele asociate.

```
private readonly IUserService _userService;
private readonly IAddressService _addressService;
private readonly IUserProfileService _userProfileService;
private readonly IUnitOfWork _unitOfWork;

public UsersController(IUserService userService, IAddressService addressService,
    IUserProfileService userProfileService)
{
    this._userService = userService;
    this._addressService = addressService;
    this._userProfileService = (parameter) IAddressService addressService
}
```

Figura IV.4.5. *UsersController*

Componenta de tip controller va conține toate serviciile pe care le utilizează în cadrul acțiunilor definite.

Injectarea dependențelor în cadrul constructorului a fost utilizată în toate straturile produsului software prezentat.

V. CONCLUZII

Șabloanele de proiectare sunt esențiale pentru dezvoltarea produselor software. Pe lângă abstractizarea straturilor aplicației și creșterea calității codului, introduc un limbaj bazat pe cunoștințe solide de proiectare orientată obiect, care poate fi utilizat de toți componenții echipei de dezvoltare, facilitând astfel comunicarea. Șabloanele de proiectare sunt independente de limbajul de programare, facilitând astfel schimbarea ulterioară a platformei de dezvoltare și adaptarea programatorilor.

Șabloanele de proiectare sunt testate de către alți dezvoltatori, implementarea lor aducând cu siguranță valoare produsului final. Având în vedere că șabloanele de proiectare sunt soluții reutilizabile, problemele cu care ne vom confrunta vor însemna, de fapt, alegerea șablonului de proiectare potrivit sau adaptarea unei soluții deja implementate în cadrul altor produse software.

Șabloanele de proiectare nu sunt un “deus ex machina”, programatorul având responsabilitatea de a înțelege structura generală a problemei, precum și toate implicațiile generate de aceasta. Înainte de a decide ce șablon de proiectare se va utiliza, trebuie să decidem dacă problema noastră se pretează spre a fi rezolvată cu ajutorul unui șablon de proiectare, deoarece putem introduce o complexitate care nu e necesară pentru rezolvarea unei probleme simple.

BIBLIOGRAFIE

Afana, N. (2011, January 14). *ASP.NET MVC 5 Internationalization*. Nadeem Afana's Blog:
<http://afana.me/post/aspnet-mvc-internationalization.aspx>

Alexander, C. (1977). *A Pattern Language: Towns, Buildings, Construction*. Oxford University Press.

Bailey, D. (2009, February 11). *SOLID Development Principles – In Motivational Pictures*.
Los Techies: <http://lostechies.com/derickbailey/2009/02/11/solid-development-principles-in-motivational-pictures/>

BForms Team. (2013).
Bootstrap Forms for ASP.NET MVC. <https://github.com/stefanprodan/BForms>

Bhatia, S. (2013). *MVC Introduction*.
Code Project: <http://www.codeproject.com/Tips/669195/MVC-Introduction>

Brett McLaughlin, G. P. (2006). *Head First Object-Oriented Analysis and Design*. O'Reilly Media.

Chadwick, J. (2012). *The Features and Foibles of ASP.NET MVC Model Binding*.
MSDN Magazine: <http://msdn.microsoft.com/en-us/magazine/hh781022.aspx>

Community, S. (2013, May 13). *Relation between CommonJS, AMD and RequireJS?*.
StackOverflow: <http://stackoverflow.com/questions/16521471/relation-between-commonjs-amd-and-requirejs>

Correia, P. (2014). *AMD*. <https://github.com/amdjs/amdjs-api/wiki/AMD>

Documentation, E. F. (n.d.). *Entity Framework*.
Microsoft: <http://msdn.microsoft.com/en-us/data/ef.aspx>

Europa. (2006, January 20). *Cadrul strategic pentru multilingvism*. Europa - Sinteze ale legislației UE:

http://europa.eu/legislation_summaries/education_training_youth/lifelong_learning/c11084_ro.htm

FitzMacken, T. (2014, February 7). *Intro to ASP.NET Web Programming Razor Syntax*.

Microsoft ASP.NET: <http://www.asp.net/web-pages/tutorials/basics/2-introduction-to-asp-net-web-programming-using-the-razor-syntax>

Gang of Four. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. USA: Addison-Wesley.

Hölbling, D., & Foster, J. (2013, December 10). *dotless 1.4.0*.

Nuget Gallery: <https://www.nuget.org/packages/dotless/1.4.0>

JetBrains. (2006). *JetBrains - Team City*.

JetBrains: <http://www.jetbrains.com/teamcity/>

Jones, S. P., & Augustsson, L. (2010). *Haskell Programming Language*.

HaskellWiki: <http://www.haskell.org/haskellwiki/Haskell>

Kohari, N. (2007). *Ninject - Dependency Injector for .Net*.

Ninject: <http://www.ninject.org/>

LINQ. (2007).

Microsoft Developer Network: <http://msdn.microsoft.com/en-us/library/bb397926.aspx>

Martin, R. (2002). *Agile Principles, Patterns, and Practices in C#*. Pearson Education.

Meyer, B. (1988). *Object-Oriented Software Construction*. Prentice Hall.

Microsoft Corporation. (2012). *LINQ and ADO.NET*.

Msdn: <http://msdn.microsoft.com/en-us/library/bb399365.aspx>

MSDN. (2012). *Introduction to Membership*.

MSDN Documentation:

[http://msdn.microsoft.com/en-us/library/vstudio/yh26yfzy\(v=vs.100\).aspx](http://msdn.microsoft.com/en-us/library/vstudio/yh26yfzy(v=vs.100).aspx)

MSDN. (n.d.). *Repository Pattern*.

MSDN: <http://msdn.microsoft.com/en-us/library/ff649690.aspx>

Mustache. (2009). Mustache - GitHub: <http://mustache.github.io/>

negaozen. (2013). *ASP Xtreme Evolution*. ASP Xtreme Evolution -
GitHub:<https://github.com/nagaozen/aspxtremeevolution/blob/master/lib/axe/classes/Parsers/mustache.asp>

NuGet. (2013, August 22). *Using NuGet without committing packages to source control*.

NuGet: <http://docs.nuget.org/docs/workflows/using-nuget-without-committing-packages>

Otto, M. (2011). Twitter Developers: <https://dev.twitter.com/>

Pialorsi, P., & Russo, M. (2010). *Programming Microsoft® LINQ in Microsoft .NET Framework 4*. Sebastopol, California: O'Reilly Media, Inc.

Shepherd, E. (2014, April 14).

<https://developer.mozilla.org/en-US/docs/Mozilla/Projects/Rhino>

Sklivvz, M. C. (2012, May 19). *What is the Liskov Substitution Principle?*

Stack Overflow:

<http://stackoverflow.com/questions/56860/what-is-the-liskov-substitution-principle>

Vaynberg, I. (2012). *Select2*. <http://ivaynberg.github.io/select2/>

Wasson, M. (2012, December 12). *Forms Authentication*.

ASP.NET: <http://www.asp.net/web-api/overview/security/forms-authentication>

Weizenbaum, N. (2008). Sass Language: <http://sass-lang.com/>

Wilson, B. (2010, October 06). *Unobtrusive Client Validation in ASP.NET MVC 3*.

Brad Wilson Technologist:

<http://bradwilson.typepad.com/blog/2010/10/mvc3-unobtrusive-validation.html>