|  |  |  |
| --- | --- | --- |
| sigla MI cu text | **Universitatea TRANSILVANIA din Braşov**  **Facultatea de Matematicǎ şi Informatică**  **Programul de studiu Informatică aplicată** | sigla unitbv transparenta |

Lucrare de licenţă

Autor: **Gherghelău Andrei**

Coordonator ştiinţific: **Prof. univ. dr. Dorin Bocu**

Braşov

Iulie, 2016

**FIŞA LUCRĂRII DE LICENŢĂ**

|  |  |
| --- | --- |
| Universitatea *Transilvania* din Braşov | Lucrare de absolvire/ lucrare de licenţă/ proiect de diplomă/ disertaţie nr. .......... |
| Facultatea de Matematică şi Informatică |
| Departamentul de Matematică şi Informatică | Viza facultăţii |
| Programul de studii Informatica aplicata | Anul universitar 2015-2016 |
| Candidat: Gherghelău Andrei | Promoţia 2016 |
| Cadrul didactic îndrumător | Prof. dr. Dorin Bocu |

|  |
| --- |
| **LUCRARE DE LICENŢĂ** |
| Titlul lucrării:  O soluție flexibilă la problema orarului în cadrul Universității Transilvania din Brașov |
| Problemele principale tratate:  Motivarea alegerii implementării unei aplicații web  Tehnologii folosite: Web API 2, T-SQL, Angular JS, Bootstrap JS  Unelte folosite: IIS, Postman, VS 2012  Sisteme de versionare: Git, SVN  Soluție tehnică: Web API 2 (server), Angular JS + Bootstrap (client), Sql Server (baza de date)  Manualul de utilizare a aplicației  Ghidul de instalare a aplicației |
| Locul şi durata practicii: - |
| Bibliografie:  Patterns of enterprise application (M.Fowler), DoFactory – Design Patterns,  Ingineria sistemelor soft reale (D.Bocu), Medii vizuale de programare (L.M.Sasu)  Pro ASP.NET – Web API (A.Zeitler), wikipedia.ro, codeproject.com |
| Aspecte particulare: - |

|  |
| --- |
| Primit tema la data de: 1 octombrie 2015 |

|  |
| --- |
| Data predării lucrării: 27.06.2016 |

|  |  |  |
| --- | --- | --- |
| Director departament, Cadru didactic îndrumător,  *Conf.univ.dr. MARIUS PĂUN, Prof.univ dr. Dorin Bocu* | | |
| Candidat,  *Gherghelău Andrei* | | |
| **LUCRARE DE LICENŢĂ** | | | |
| Data  vizei | | Capitole/ problemele analizate | Semnătura cadrului didactic îndrumător |
|  | |  |  |
|  | |  |  |
|  | |  |  |
|  | |  |  |
|  | |  |  |

|  |  |  |
| --- | --- | --- |
| **APRECIEREA ŞI AVIZUL CADRULUI DIDACTIC ÎNDRUMĂTOR** | | |
|  | | |
| Data: | ADMIS pentru susţinere/ RESPINS | CADRU DIDACTIC ÎNDRUMĂTOR  Prof. univ. dr. Dorin Bocu |

|  |  |  |
| --- | --- | --- |
| **AVIZUL DIRECTORULUI DE DEPARTAMENT** | | |
| Data: | ADMIS pentru susţinere | Director departament  Conf.univ. dr. Marius Păun, |

|  |  |
| --- | --- |
| **SUSŢINEREA LUCRĂRII DE ABSOLVIRE/ LUCRĂRII DE LICENŢĂ/**  **PROIECTULUI DE DIPLOMĂ/ DISERTAŢIEI** | |
| Sesiunea iunie-iulie 2016 | |
| Rezultatul  susţinerii | PROMOVAT cu media: |
| RESPINS **cu** refacerea lucrării |
| RESPINS **fără** refacerea lucrării |
| PREŞEDINTE COMISIE | |

O soluţie flexibilă la problema orarului în cadrul Universităţii Transilvania din Braşov

Cuprins

[1. Introducere 7](#_Toc454712572)

[1.1. Motivarea alegerii temei 7](#_Toc454712573)

[1.2. De ce aplicaţie Web? 7](#_Toc454712574)

[2. Tehnologiile folosite 9](#_Toc454712575)

[2.1. C# - Web API 2 9](#_Toc454712576)

[2.2. SQL – Microsoft Sql Server 2012 (MSSQL) 13](#_Toc454712577)

[2.3. Angular JS 1.3 (Javascript) 19](#_Toc454712578)

[2.4. Bootstrap CSS (CSS3) 23](#_Toc454712579)

[3. Unelte folosite 25](#_Toc454712580)

[3.1. IIS (Internet Information Services) 25](#_Toc454712581)

[3.2. POSTMAN 26](#_Toc454712582)

[3.3. Web Essentials 29](#_Toc454712583)

[3.4. ReSharper 9 30](#_Toc454712584)

[3.5. Visual Studio 2012 30](#_Toc454712585)

[3.6. Versionare 32](#_Toc454712586)

[3.6.1. SVN 34](#_Toc454712587)

[3.6.2. Git 37](#_Toc454712588)

[4. Soluție Tehnică a proiectului 41](#_Toc454712589)

[4.1. Soluție Tehnică pe parte de server 41](#_Toc454712590)

[4.2. Soluție Tehnică pe parte de client 51](#_Toc454712591)

[4.3. Soluție Tehnică pe partea de păstrare a datelor 56](#_Toc454712592)

[5. Manual de utilizare a aplicației 57](#_Toc454712593)

[5.1. Ghid de utilizare – Administrator 58](#_Toc454712594)

[5.2. Ghid de utilizare – Profesor 63](#_Toc454712595)

[5.3. Ghid de utilizare – Elev 64](#_Toc454712596)

[6. Ghidul de instalare a aplicației 67](#_Toc454712597)

# Introducere

## Motivarea alegerii temei

Orarul este unealta cea mai folosită în timpul facultăţii. De la intrarea mea în facultate şi până la finalul acesteia m-am lovit aproape zilnic de necesitatea informatizării orarului şi a evenimentelor de zi cu zi. Din această cauză m-am gândit la implementarea unei platforme web cu care să putem noi, ca studenţi, să relaţionăm oricând şi prin care să fim informaţi non-stop de orice schimbare a programului.

În trecut, profesorii şi studentii beneficiau de orarul scris la intrarea în facultate cât şi de orarul prezent pe site-ul facultăţii in versiunea pentru telefoane/tablete care nu era chiar lizibil din pricina soft-ului cu care a fost realizat.

Din cauza acestor inconveniente m-am hotărât să construiesc o platformă care să înglobeze toate aceste aspecte legate de orar cât şi posibilitatea de extindere.

Această platformă nu este dedicată numai studenţilor ci şi profesorilor. Toţi utilizatorii vor putea să-şi vadă orele din cursul unei săptămâni şi ultimele anunţuri care vor fi legate strict de orar.

Din punctul de vedere al organizării conţinutului, platforma vine cu un modul separat de administrare. Pentru ca platforma să funcţioneze corect administratorul va trebui să adauge în diferitele rubrici datele tuturor studenţilor şi a profesorilor.

Aplicaţia a fost construită pentru a uşura munca funcţionarilor facultăţii dar şi a profesorului ce întocmeşte orarul şi totodata viaţa studentilor care vor afla in timp util modificările aduse orarului.

## De ce aplicaţie Web?

În zilele noastre asistăm la o schimbare tehnologică importantă. Smartphone-urile şi tabletele sunt din ce în ce mai prezente în viaţa noastră de zi cu zi, lucru care m-a determinat să creez o aplicaţie cross-device. Aceste aplicaţii cross-device / cross-platform, rulează indiferent de platforma, sistemul de operare de pe care sunt executate.

Avantajele unei aplicaţii web sunt foarte numeroase fapt ce m-a determinat să merg către această latură a informaticii. Aplicaţiile web rulează prin intermediul unui browser web (Google Chrome, Mozilla Firefox, Internet Explorer, etc).

Programul se află pe un server web (Tomcat, JBoss, Spring, IIS, etc) sau pe un server local pentru aplicaţiile tradiţionale. În aplicaţiile clasice vom vedea pagini statice sau care nu interacţionează prin intermediul request-urilor XHR(Xml Http Request). În schimb aplicaţia pe care o vom exemplifica, vom vedea că s-au folosit tehnologii pentru realizarea unei SPA(Single Page Application).

Aplicaţiile web folosesc de obicei o bază de date pentru a stoca permanent informaţii şi pentru a avea actualizări rapide si usoare. Alt avantaj ar fi accesibilitatea. O aplicaţie web este accesibila de oriunde în lume prin intermediul host-ului ([www.google.ro](http://www.google.ro)). Disponibilitatea este un alt factor care a fost decisiv în implementarea sistemului cât şi în alegerea tehnologiilor.

Tot în sfera avantajelor încadrăm şi faptul că smartphone-urile pot fi folosite pentru a vedea conţinutul paginilor web. Dacă aplicaţia web are un templating engine în cadrul ei care ştie să randeze în funcţie de device-ul client atunci putem deduce incă un avantaj(Landscape / Portrait – Bootstrap css / js).

Aplicațiile web folosesc de obicei o combinație de server-side script (ASP, PHP, etc) și client-side script (HTML, Javascript, etc), pentru a dezvolta aplicația.Scriptul pe partea de client se ocupă cu prezentarea informațiilor , în timp ce scriptul server-side se ocupă cu toate lucrurile mai dificile cum ar fi stocarea si regăsirea de informații. Uneori, unele aplicații pot avea și logică de prezentare. Acest lucru este făcut cu ajutorul scripturilor Javascript.

Avanjate:

* Disponibilitate 24/24.
* Soluție mai ieftină.
* Multi-platform.
* Nu ocupă spațiul de pe calculatorul client-ului.
* Sisteme simplificate.
* Se rulează dintr-un browser free.

Aplicațiile web au parcurs un drum lung și oferă acum avantaje competitive la sistemele software tradiționale. Un an alt avantaj îl oferă costul mult mai mic de întreținere a platformei.

# Tehnologiile folosite

## C# - Web API 2

C# este un limbaj de programare elegant, obiect-orientat şi strong-typed care permite dezvoltatorulului să construiască o varietate mare de aplicaţii securizate şi robuste, care bineînteles rulează pe platforma .NET. C# se foloseşte pentru a crea aplicaţii Windows cât şi aplicaţii Web, client-server şi multe altele.

De ce C# ? Acest limbaj de programare de câteva versiuni a fost portat pe orice sistem de operare. Ca şi limbaj de programare a apărut în anii 2000 fiind un limbaj de programare C-like obiect-orientat. Denumirea de C# vine de la C++ ++, de unde suprapunând cele două plusuri ar rezulta #.

Limbajul C# este foarte expresiv şi foarte uşor de invăţat. El se aseamănă foarte mult cu C++, Java sau C. Dezvoltatorii recunosc instantaneu sintaxa. Sintaxa limbajului este menită să ajute programatorul prin tot felul de “sintactic sugars”(dulcegării sintactice) şi incearcă totodata să păstreze inteligibilitatea codului scris.

Platforma .NET este responsabilă cu rularea oricărui program C#. Codul scris în C# se rulează cu ajutorul CLR (Common Language Runtime) care reprezintă un mediu virtual de execuţie unde se unesc și librariile de sistem. CLR este o implementare comercială scrisă de cei de la Microsoft a CLI-ului (Common Language Infrastructure), un standard internaţional pe care se construiesc limbajele de programare.

Codul C# este compilat într-un limbaj intermediar numit (IL / CIL / MSIL – Common Intermediate Language) care confirmă că infrastructura este respectată. Codul IL şi resursele adiacente sunt stocate pe disk sub forma unor fişiere (.exe / .dll), ele fiind numite si assembly files. Aceste fişiere conţin un manifest care ne facilitează informaţii în legătură cu versiunea, tipul şi cultura fişierelor, totodata ajutând la securitate.

În imaginea de mai jos sunt prezentate etapele prin care trece un program scris în C#. Diagrama ilustreaza chiar şi partea în care JIT-ul acţionează. JIT = Just in Time Compiler este componenta care converteşte codul MSIL/IL/CIL în cod nativ la run-time.

Garbage collector. Este unealta prin care limbajul C# se remarca foarte puternic faţă de C++. Această unealtă colectează memoria care nu mai este folosită. Totul se bazează pe reference counting. Dacă o zonă de memorie nu mai este referenţiată printr-un pointer, atunci GC-ul o să golească acea zonă de memorie. Acesta are 3 faze de lucru(3 generaţii de colectare).

Mai jos avem o diagramă ce exemplifica explicaţiile anterioare.

Practic, platforma .NET este liantul dintre programator şi sistemul de operare. Absolut orice cerere a dezvoltatorului (alocare memorie, modificare date, etc) se petrec sub vederea protectoare a platformei .NET. Această platformă a luat naştere deoarece problema managementului memoriei era o adevărată corvoadă in trecut. Acum datorită GC-ului memoria este administrată de către platformă.



Figura 1. C# Flow

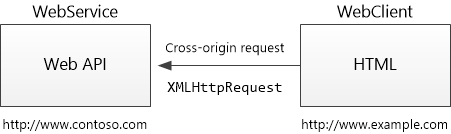


Figura 2. Web API 2 Client Request

Web API 2. Acest framework este o unealtă cu care se pot construi servicii HTTP de mare amplitudine. Folosind Web API 2 cu serializare JSON avem portabilitatea asigurată deoarece decuplăm randarea view-ului de partea de server. În zilele noastre se folosesc din ce în ce mai mult templating engine-urile care au rol de a separa codul UI de logica server-ului respectând principiul Low Coupling (cuplare slabă).

Tot în cadrul acestui framework putem folosi server-ul C# ca fiind o aplicaţie REST. Ce este o aplicaţie REST ? O aplicaţie REST (Representational State Transfer) este o aplicaţie care are 6 constrângeri. Aceste constrângeri se aplică arhitecturii.

Cele 6 constrângeri REST sunt:

* Uniform Interface
* Stateless
* Cacheable
* Client-Server
* Layered System
* Code on Demand(opţional)

**Uniform Interface** - defineşte interfaţa între clienţi şi servere. Simplifică şi decuplează toată arhitectura şi conferă independenţă componentelor.

**Stateless** – o aplicaţie este Stateless, dacă starea ei nu este reţinută. Pentru a ajunge stateless request-urile de tip client-server trebuie să fie independente, fiecare recreându-şi conexiunea şi având payload diferit.

**Cacheable** – pentru ca arhitectura să fie respectată, trebuie să avem posibilitatea să cache-uim date. Acest termen de caching reprezintă încărcarea inteligentă a datelor. Încărcarea diferitelor cereri se poate cache-ui pentru a uşura şi a creşte viteza de transmisie a datelor. Prin caching înțelegem fenomenul de reținere pe client a unor date pentru a nu repeta requesturile către server și în același timp eliminarea redundanțelor.

**Client-Server** – acest concept stă la baza arhitecturii REST, deoarece fără el aplicaţia nu ar avea sens. Acest concept separă grijile. Grija server-ului este de a procesa datele primite de către client şi de a trimite datele împachetate într-un mod inteligibil pentru client. Clientul are în grijă să prezinte conţinutul într-o formă inteligibilă pentru utilizator şi să transmită datele mai departe pentru procesare server-ului.

**Layered System** – tot sistemul trebuie să fie stratificat pentru o mai bună segregare a responsabilităţilor. Un sistem care este bine modularizat / împarţit pe straturi reuşeşte să fie reutilizabil.

**Code on demand** – serverele sunt în stare să se extindă uşor aşadar şi clientul trebuie să ţină pasul, însă aici este vorba de transfer de logică între server şi client.

În cadrul lucrării de licenţă vom observa folosirea acestei tehnologii împreună cu alte TP-uri(Third Party-uri). De cele mai multe ori vom observa necesitatea folosirii TP-urilor în problema Autentificării / Autorizării.

Exemple.

**OAuth, OAuth2.** Sunt TP-uri care folosesc la autentificarea utilizatorilor dintr-un anumit context.

**OWIN**(Open Web Interface for .NET) este interfaţa web serverele .NET şi clienţi folosit de obicei pentru Autorizări / Claims. (Este un middleware framwork).

**OWIN** a fost dezvoltat peste serverul IIS și urmărește să decupleze relația dintre ASP.NET și IIS prin definirea unei interfețe standard. Dezvoltatorii serviciilor web pot fi siguri că dacă se implementează corect OWIN, aplicațiile ASP.NET vor rula corespunzător pe acele servere. Similar, noile framework-uri pot fi dezvoltate ca o alternativă la ASP.NET. O comparație între tehnologiile Java și tehnologiile Microsoft ar fi că OWIN își dorește să facă pentru .NET ce face **Java Servletul și Servlet** containerul pentru **JVM**.

În figura 3 vom vedea un flow minimal descris într-o diagramă prin care se exemplifică mersul lucrurilor în cadrul folosirii acestei tehnologii.

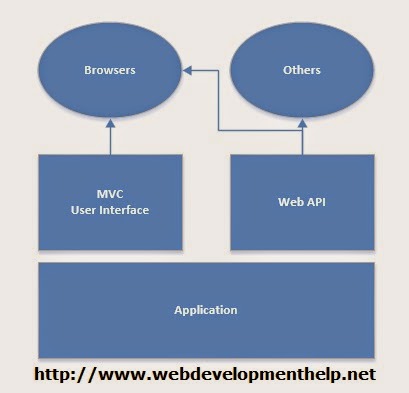


Figura 3. Arhitectura MVC

În general toate serviciile REST se cuplează cu un templating engine şi folosesc cereri de tip AJAX pentru a cere / livra date.

Ce este AJAX-ul ? … Ajax-ul este abrevierea de la Asynchronous JavaScript and XML. Este de fapt un request XHR către server.

De obicei este implementat de către librariile de JavaScript pentru comunicarea cu server-ul.

Exemple de librarii care implementează ajax.

* JQuery
* Angular JS
* Angular 2 TS
* Backbone
* Prototype
* ASP.NET AJAX
* SPRY framework
* DOJO Toolkit
* Ext JS
* Unified JS
* MooTools
* IntercoolerJS



Figura 4. Ciclu XHR

În figura 4 este ilustrat cum funcţionează un request de tip Ajax.

Observăm că totul pleacă de la un eveniment pe care utilizatorul îl produce, iar mai apoi se transmit date via Internet către server.

Server-ul procesează datele şi întoarce datele necesare către browser. Request-ul ajax interceptează semnalul serverului prin intermediul unui promise şi mai departe, clientul face update la conţinut.

În afară de faptul că s-a ales Web Api pentru a crea un RESTFull Application, o altă motivaţie ar fi că se îmbină perfect cu serviciile de baze de date oferite de Microsoft. MSSQL este următorul capitol pe care îl vom aborda.

## SQL – Microsoft Sql Server 2012 (MSSQL)

MSSQL sau Microsoft SQL, este unealta folosită pentru stocarea datelor. Sql Server este de fapt server-ul care administrează fiecare cerere pe baza de date. Am folosit versiunea 2012 deoarece această versiune este cea mai stabilă la momentul actual.

MSSQL este de fapt un RDBMS (Relational Database Management System) produs de compania americană Microsoft Corporation. Limbajul de interogare este SQL, iar extensia procedurală este T-SQL.

Acesta suportă versiunea Microsoft de SQL (Structure Query Language), cel mai răspândit limbaj pentru bazele de date. Este un sistem pentru întreprinderi – se poate aplica şi bazelor de date foarte mari.

Codul de bază pentru MSSQL îşi are originile în Sybase SQL Server şi a reprezentat intrarea Microsoft pe piaţa bazelor de date relaţionale, concurând cu Oracle, IBM şi Sybase. Microsoft, Sybase şi Ashton-Tate s-au unit pentru a crea şi scoate pe piaţă prima versiune numită SQL Server 4.2 pentru Win OS/2. Mai tărziu Microsoft a negociat pentru drepturi de exclusivitate la toate versiunile de SQL Server scrise pentru sistemele de operare Microsoft, Sybase şi-a schimbat ulterior numele în Adaptive Server Enterprise, pentru a evita confuzia cu Microsoft SQL Server.

SQL Server 7.0 a fost primul server de baze de date bazat pe GUI. O variantă de SQL Server 2000 a fost prima variantă comercială pentru arhitectura Intel. În anii urmatori s-au lansat şi alte versiuni, ce au adus îmbunătăţiri de performanţă, tool-uri IDE şi ETL, suport pentru date XML, geometrice sau spaţiale, căutare în text, self-tuning, clustering şi HADR.

Versiuni de MSSQL. SQL Server 2005, SQL Server 2008, SQL Server 2012, SQL Server 2014, SQL Server 2016.

Mod de lucru al Server-ului Sql Microsoft. Acesta foloseşte o variantă de SQL numită T-SQL sau Transact-SQL, o implementare de SQL-92 (standard ISO) cu unele extensii. T-SQL în principal adaugă sintaxă adiţională pentru procedurile stocate şi pentru tranzacţii.

Standardele SQL necesită implementare principiilor ACID. Ce este ACID? ACID este acronimul pentru cele 4 condiţii pe care orice tranzacţie trebuie să o respecte. Şi anume, atomicitate, consistenţă, izolare, durabilitate.

Stocarea datelor se face pe baza modelelor relaţionale. Modele relaţionale bazate pe tabele(entităţi) având fiecare coloane(proprietăţi). Fiecare coloană trebuie să aibă un tip şi anumite constrângeri de mărime.

Tipuri de date în SQL Server. MSSQL suportă tipurile primare, precum întreg, zecimal, caracter, dată, dar şi tipuri complexe, date binare, date geometrice, date spaţiale, XML, etc.

Altă funcţionalitate a MSSQL-ului este crearea de diagrame relaţionale, din care putem observa structura bazei de date, tipul relaţiilor cât şi interacţiunea entităţilor.

Mod de lucru. După spusele anterioare, MSSQL are posibilitatea de a fi folosit prin intermediul unei interfeţe. Această interfaţă este furnizată de Microsoft Management Studio 2012 (MSMS 2012), acesta fiind un soft care este un wrapper peste SQL Server 2012. MSMS ajută foarte mult utilizatorul cât şi admin-ul bazei de date prin interfaţa prietenoasa. În cele de mai jos, vom putea observa câteva imagini cu 2 funcţionalităţi.

În figura 5 putem observa că în partea stângă avem o unealtă numită Object Explorer, de unde ne putem selecta baza de date pe care dorim să lucrăm ca mai apoi să observăm tabelele cât şi alte lucruri importante din ea.

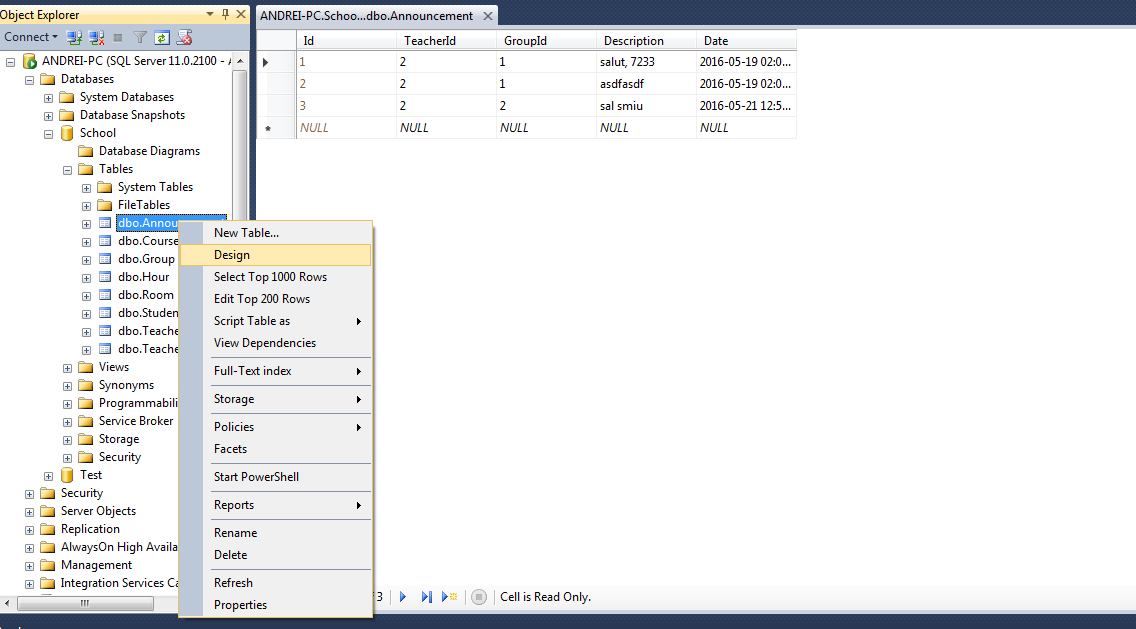


Figura 5. Object explorer & Data

În figura 5 am dat click drepta pe o tabelă dintr-o bază de date şi am selectat „Edit Top 200 Rows” din meniul contextual. După selectarea acelei opţiuni, motorul de baze de date ne-a afişat conţinutul tabelei prin enumerarea coloanelor cu intrările aferente.

Tot prin meniul contextual putem alege opţiunea Design, de unde putem altera / modifica structura unei tabele. (vezi urmatoarea poză).

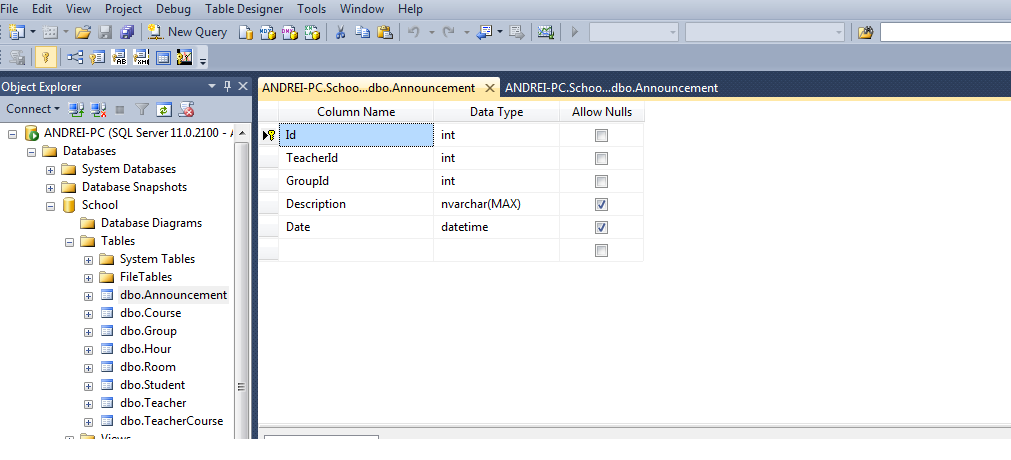


Figura 6. Date din tabela

Creearea unei baze de date se face foarte simplu, doar dând click pe opţiunea „Create Database” din meniul contextual oferit de folder-ul Databases.

MSSQL oferă posibilitatea salvării unei baze de date. Salvarea se poate efectua într-un fişier de tip .bak, fie într-o altă bază de date aflată la distanţă.

Această funcţionalitate de salvare a conţinutului cât şi a structurii se face prin crearea unui fişier cu comenzi SQL, serializat şi comprimat, prin care se poate reface baza de date oricand. Practic acest back-up realizează un snapshot al bazei de date la momentul T0.

Creerea unui back-up. Back-up-ul unei baze de date se face prin interfaţă, dându-se click drepta pe baza de date pe care dorim să efectuăm back-up-ul, şi din meniul contextual alegem opţiunea Tasks -> Back Up …

Prin alegerea acestei opţiuni ni se va deschide o nouă fereastră prin care efectuăm back-up-ul prin intermediul unui fişier .bak. Selectăm calea ca în figura 7 şi apăsăm Ok pentru finalizarea procesului.

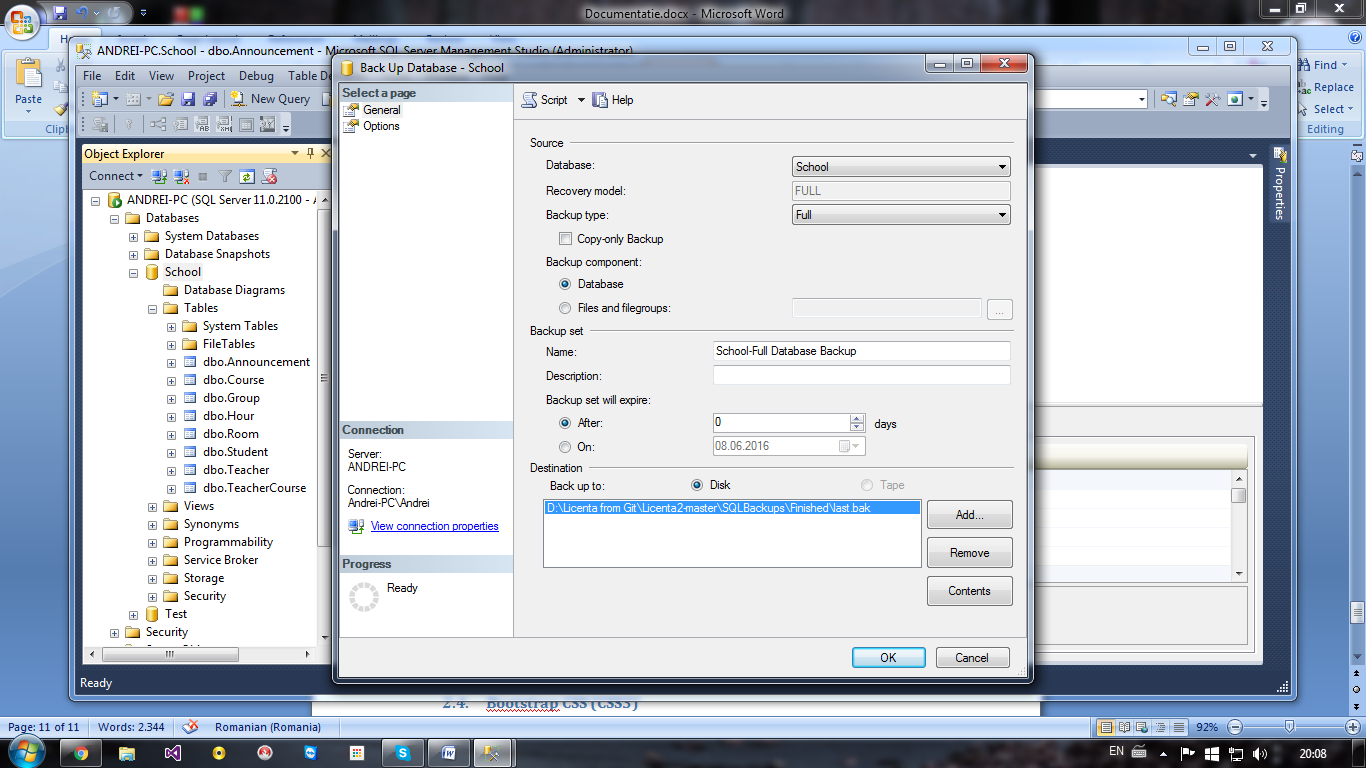


Figura 7. Back-up baza de date

În figura 7 putem observa fereastra care ne va apărea dacă vom dori să facem back-up unei baze de date.

Pentru a selecta calea către fişierul bak, vom da click pe butonul Add, de unde vom selecta folder-ul de apartenenţă.

După setup se va apasa OK, pentru a începe operaţiunea de back-up. La finalul operaţiunii vom primi un mesaj de finalizare cu succes. Pentru a putea folosi bak-ul, îl vom depozita corespunzător şi îl vom denumi cu numele bazei de date şi data creării sale.

Acum, că am înţeles cum trebuie să facem în cazul în care dorim să salvăm datele, vom trece la capitolul unde le vom restaura.

Pentru restaurarea unei baze de date, vom începe prin conectarea la un MSSQL Server, cu ajutorul Management Studio-ului. După conectarea propriu-zisă, vom da click dreapta pe folder-ul Databases, unde vom selecta din meniul contextual, opţiunea Restore Database. După selectarea opţiunii menţionate, ne va apărea urmatoarea fereastră.

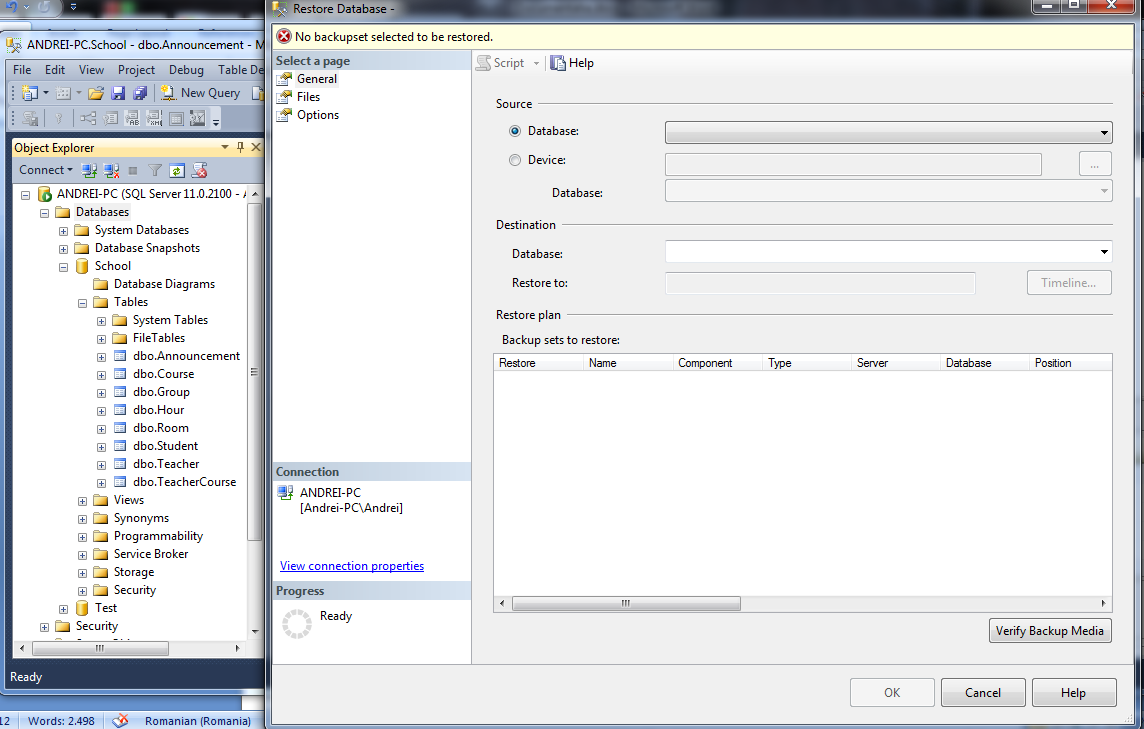


Figura 8. Fereastra de selectare fisier BAK

În figura 8 avem fereastra pe care trebuie să o vedem după selectarea opţiunii de restaurare. Pentru a restaura o bază de date prin intermediul unui fişier de tip bak, va trebui selectat radio buton-ul, Device. După selectarea opţiunii, va trebui să dăm calea absolută către fişierul bak. Selectarea fişierului bazei de date va fi următorul pas. Baza de date poate fi una existentă pe acel MSSQL sau poate fi una nouă (în cazul unui install).

Executarea script-ului din fişierul bak va restaura baza de date urmând apoi folosirea ei după nevoi. Putem vedea în poză şi existenţa butonului Verify Backup Media. Acel buton este folosit pentru verificarea fişierului. Verificarea se va face prin checksum-uri. De obicei se verifică fişierele pentru a nu se corupe.

Tema securităţii în cadrul bazelor de date este vitală, aşadar vom verifica de fiecare dată fişierul pe care îl încărcăm.

Securitatea în cadrul MSSQL. Vom face o scurtă prezentare despre securitatea în cadrul uneltei SQL Server. SQLS facilitează o arhitectură a securităţii care permite dezvoltatorilor şi administratorilor de baze de date să creeze medii securizate şi prin care să prevină atacurile asupra sistemului.

De la versiune la versiune cei de la Microsoft au reuşit să îmbunătăţească partea de securitate prin introducerea unor noi mecanisme de securitate şi funcţionalităţi noi care să faciliteze bunul mers al aplicaţiilor.

Tema securităţii în cadrul SQL Serverului a fost abordată din două perspective. Prima perspectivă este aceea a autentificării şi bineînţeles cea de-a doua este autorizarea.

Autentificarea este procesul sau ansamblul de procese prin care se verifică identitatea unui individ care accesează o resursa. În cadrul SQL Serverului este un proces prin care pe baza unor credenţiale se stabileşte identitatea şi permiterea accesării sistemului.

Autorizarea este procesul prin care se determină dacă utilizatorul autentificat are permisiunea de a accesa o anumită resursă şi ce operaţii se pot face pe acea resursă.

În cazul în care dorim ca un utilizator sau vreun proces să se poată autentifica şi să aibă acces la resurse, vom folosi Object Explorer-ul şi în cadrul său vom vedea folder-ul Security.

Expandăm folder-ul Security unde vom găsi folder-ul Logins. In acest folder dăm click drepta şi vom putea să adăugăm un nou Login. Acest Login se referă la conectarea unor alte resurse în afară de localhost. Aici în ghidul de instalare al aplicaţiei am prezentat cum trebuie să procedăm pentru autentificarea server-ului IIS.

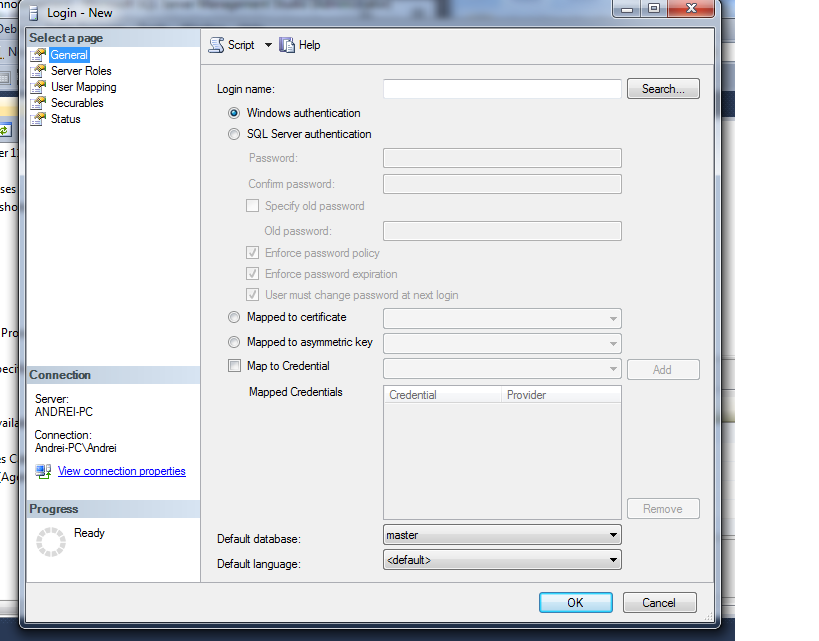


Figura 9. Securitate, Logins

În figura 9 vom vedea cum arată fereastra de adăugare de nou login. În cadrul login name-ului vom trece numele procesului sau numele utilizatorului care doreşte să se conecteze.

Pentru a-l autoriza vom merge ulterior pe utilizatorul în cauză şi îi vom acorda drepturile pe baza de date pe care o dorim. Aici trebuie avut grijă ce drepturi dăm anumitor utilizatori.

Putem crea chiar noi tipul nostru de utilizator unde putem seta ce permisiuni vrem să aibă acel utilizator. În ghidul de instalare al aplicaţiei vom putea observa ce drepturi am lăsat.

## Angular JS 1.3 (Javascript)

Angular JS 1 este o librărie de Javascript fondată de Google. Este o librărie care are la bază principiile MVVM(Model View ViewModel). Totul se bazează pe ideea de binding a unui model pentru view.

Angular a fost creat pentru partea de client ca şi templating engine pentru aplicaţiile web. Codul de la Angular JS este open-source, aşadar oricine poate să preia şi să folosească acest cod.

Principala funcţionalitate a acestei librării este de a crea SPA-uri. Single page application-ul este o aplicaţie web, în care se schimbă doar conţinutul paginii pe baza unor request-uri XHR. Ideea de SPA este de a separa partea de view de partea logică (server from client).

Ce este pattern-ul MVVM ? Model-View-ViewModel este o arhitectură care înglobează 3 componenete. Modelul, care este entitatea logică sau obiectul domeniu.

Model. Reprezintă datele clare sau informaţia pe care o dorim. De cele mai multe ori în şablonul MVVM, modelul este constituit din obiectele POCO folosite pentru modelarea entităţilor de baze de date. Modelul trebuie separat total de logica de bussiness a aplicaţiei, deoarece face parte din alt strat.

Modelul în Angular JS, este orice obiect cu care decorăm serviciul Scope. „$scope”, este un serviciu furnizat de Angular prin care partea de View este notificată asupra modificărilor oricărui model. Decorarea scope-ului se face prin adăugarea proprietăţilor necesare obiectului în cauză.

Un exemplu ar fi: „ $scope.model = { name: „Andrei” }; ”. Prin sintaxa prezentată anterior noi am decorat scope-ul cu o variabilă pe nume model, în care avem o proprietate de tip string în membrul name.

Pentru ca modelul să se vadă pe UI, trebuie ca dezvoltatorul să scrie şi partea de view. În cazul nostru view-ul va fi un template HTML.

View-ul reprezintă stratul de prezentare a datelor. Sarcina şablonului MVVM este de a separa codul de UI de codul de domeniu cât şi separarea de logică de bussiness a aplicaţiei. Aici View-ul este reprezentat de un HTML. Acest HTML trebuie să conţină anumite tag-uri specifice Angular JS astfel încât să putem template-iza cât mai bine datele primite.

View-ul are rol de prezentare. În cadrul view-ului facem marea parte din prezentare, aşadar aici nu ar trebui să ţinem logică sau orice manipulare de date consistentă.

Un exemplu pentru a demonstra cele spuse mai sus ar fi.

*<html>*

*<body ng-app=”myApp” ng-controller=”mainController”>*

*<p> {{model.name}} </p>*

*</body>*

*</html>*

Prin exemplul de mai sus am creat o simplă aplicaţie Angular JS numită myApp, iar prin declararea controller-ului mainController şi prin decorarea serviciului scope din exemplul din model ni se va afişa în browser într-un paragraf următorul text: Andrei.

Am observat introducerea unei noi noţiuni, şi anume, controller-ul. În cazul Angular-ului ViewModel-ul este redenumit Controller. De ce a fost denumit ViewModel-ul, Controller ? Înaintea conceptului de MVVM, a fost inventat conceptual de MVC sau Model View Controller.

Ce este MVC? Ca şi MVVM respectă primele 2 litere, numai că în loc de ViewModel avem un controller. Ce este un Controller din sfera MVC ? Un controller de acest tip este o entitate care se ocupă cu manipularea datelor şi cu toate evenimentele şi input-ul utilizator. Până aici acelaşi lucru face şi un ViewModel.

Care este totuşi diferenţa între un ViewModel şi un Controller ? Diferenţa o face principiul de funcţionare. Un controller va avea mereu o legătură strânsă între el şi View, în schimb un ViewModel prin mecanismele librăriei şi prin implementarea şablonului Observer va şti mereu când orice membru din ViewModel se va modifica.

Această conexiune între View şi ViewModel se face prin intermediul binding-ului. Ce este binding-ul? Binding-ul este legarea printr-o sintaxă specifică librariei a unui obiect din viewModel cu un token din View. Observăm în exemplul de mai sus cum prin încadrarea în acolade duble reuşim să realizăm primul bind. Aşadar prin acel bind, orice modificare se va vedea direct în View, respectiv pe ecranul utilizatorului. Anterior am pomenit de şablonul Observer ca şi idee de bază a ideii de binding.

**Ce este şablonul Observer ?**

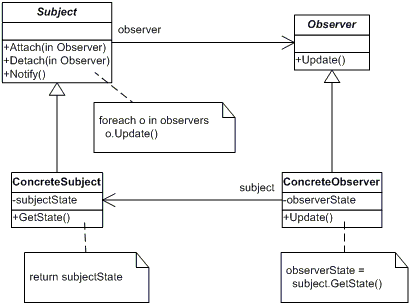


Figura 10. Observer UML

Observer-ul este un şablon de proiectare prin care o listă de abonaţi sunt notificaţi de orice acţiune a unui subiect. În cazul nostru subiectul îl constituie orice element decorat pe scope, iar observatorii sunt elementele de DOM din html. Mai sus avem o diagramă UML prin care observăm cum funcţionează şablonul. Un anumit input determină o schimbare a subiectului. După verificarea stării în care se determină dacă obiectul a fost updatat, se vor notifica toţi observatorii (abonaţii).

Observer este un şablon foarte folosit în programare, deoarece el marşează pe idea de provider – consumer. El stă la baza evenimentelor şi a tuturor acţiunilor care au la bază un input uman.

Datorită anumitor cazuri speciale, bindurile s-au împărţit în mai multe tipuri. Tipuri de binding-uri: - One Way, - Two Way

Tipul de binding Two Way: Începem cu el deoarece este tipul de binding default.

Ce reprezinta “way-ul” ? way-ul reprezintă calea de update a unui model. Two way reprezintă calea bidirecţională de update a modelului. Adică orice input care vine către un model este automat redat în view şi orice modificare în view se va vedea în model. Two way bind se realizează prin intermediul directive ng-model.

One way binding-ul se face prin intermediul acoladelor duble. Acesta va realiza doar o parte de update, şi anume partea din model în view. Aşadar orice input extern nu va lansa o modificare în modelul propriu-zis.

În figura 11 este prezentată o comparaţie între MVC şi MVVM.

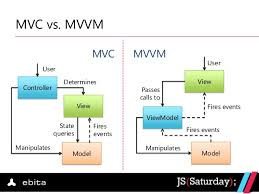


Figura 11. MVC vs MVVM

De ce am alege MVVM în detrimentul MVC ? Alegem MVVM deoarece ne scapă de codul de UI pe care trebuie să-l scriem pentru a updata View-ul de fiecare dată când schimbăm o proprietate.

Comparaţie între MVVM | MVC.

MVVM:

1. ViewModelul încapsulează logica de prezentare si statusul
2. ViewModelul este un şablon opţional
3. Utilizatorul va “lovi” prima dată UI-ul
4. View-ul nu poate “vedea” Modelul
5. View-ul are o instanţă proprie de ViewModel

MVC:

1. Controllerul determină flow-ul aplicaţiei
2. Controller-ul e obligatoriu
3. Utilizatorul va lovi întotdeuna controller-ul
4. View-ul are cunoştinţe despre model
5. View-ul va avea instanţe de tip model, lucru care nu este bun.

După lamurirea dedesubturilor Angular-ului, ar trebui să înţelegem ce se petrece şi pe partea de server.

Mai jos avem o diagramă prin care putem observa cum se comportă un Single Page Application faţă de o aplicaţie tradiţională de tip multi-page sau request based.

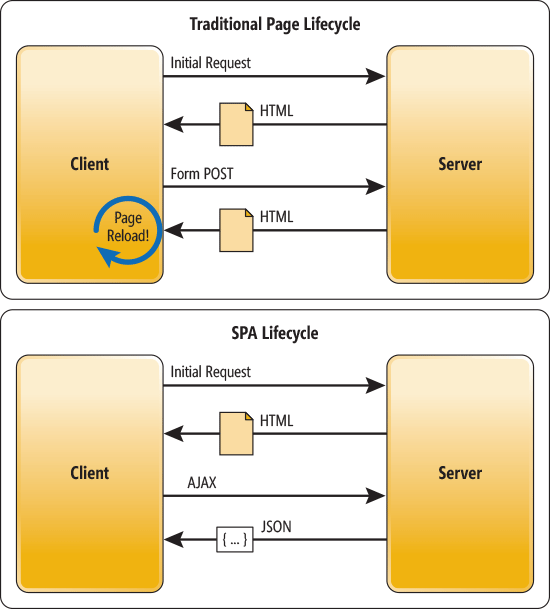


Figura 12. SPA vs MPA

Putem observa că în primul scenariu fiecare request aduce după sine şi o reîncărcare a paginii, lucru care din punct de vedere al performanţei nu este chiar optim. În schimb în cadrul unui SPA se va face un request iniţial pentru HTML(View), mai apoi tot conţinutul va fi updatat după noile date introduse în View.

Observăm AJAX-ul prezent cât şi JSON-ul care se va întoarce. Acel JSON (Java script object notation) este obiectul pe care îl vom primi de la serviciul web şi prin care vom updata modelul.

SPA cu Angular JS funcţionează pe baza unui motor de rute pentru aplicaţia în cauză. Rutarea semnifică procesul de a ajunge la o anumită resursă într-un mod controlat de către aplicaţia web.

În Angular JS rutarea se poate face cu engine-ul standard (ngRoute) sau se poate folosi o extensie Angular JS numită Angular UI-Router.

În cadrul lucrării s-a folosit engine-ul default (ngRoute). Mai jos vom vedea cum funcţionează acest motor de rutare.

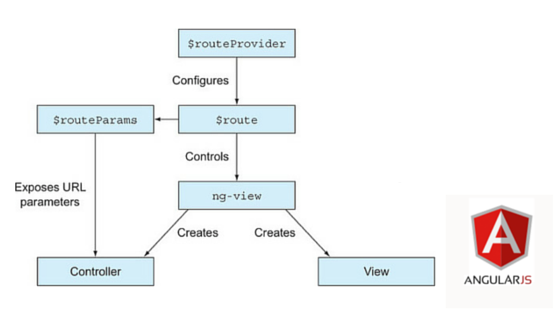


Figura 13. Angular Routing System

După cum putem vedea rutarea se face pe baza unui provider care se va configura odată cu adăugarea modulelor. Practic, ngRoute defineşte un arbore cu N noduri pe fiecare nivel. Rădăcina de obicei este index-ul paginii web.

Tranziţiile între pagini se vor face prin injectarea serviciului $location. Acest serviciu permite dezvoltatorului să treacă pe baza configurărilor dintr-o stare în alta.

## Bootstrap CSS (CSS3)

Bootstrap este un framework open-source pentru partea de front-end a aplicaţiilor web. Este folosit ca o unealtă de customizare şi înfrumuseţare a conţinutului.

Acest framework conţine template-uri HTML / CSS gata concepute şi care pot fi integrate. Aceste template-uri sunt out of the box (de tipul plug and play).

Bootstrap CSS a fost dezvoltat de Mark Otto şi Jacob Thornton în cadrul companiei Twitter ca şi tool intern.

Primul nume al librăriei a fost Twitter Blueprint. Această librarie a luat naştere după câteva luni de la lansare prin contribuţia unui grup de dezvoltatori care au făcut o saptămână de tip hackathon.

Avantajul folosirii Bootstrap este că toate browserele suportă operaţiile definite de Bootstrap. Exemple de browsere, Google Chrome, Firefox, IE, Opera, Safari, etc.

Începând cu versiunea 2.0, librăria suportă şi design responsive.

Ce este design-ul responsive? Este capabilitatea aplicaţiei de a se adapta în funcţie de mărimea Viewport-ului(ecran). Spre exemplu există clase speciale pentru adaptarea unui anumit conţinut pe parte mobilă (xs = extra small, sm = small device, lg = large, etc).

Versiunea 3.0 a adaptat de la început mobile-first design, aşadar by default aplicaţiile suportă şi versiunea mobilă.

Versiunea 4.0 alpha, s-a adăugat suport pentru Sass şi Flexbox. Acest framework este open-source şi disponibil pe GitHub sau pe site-ul oficial.

Una din cele mai folosite funcţionalităţi Bootstrap este layout-ingul. Şi anume folosirea sistemului bazat pe grid. Grid-ul bootstrap constă în creerea unei structuri prin intermediul claselor css.

Ideea de bază este aceea că, se creează un container la care se adaugă linii. În fiecare linie se adaugă coloane. Aceste lucruri se fac cu ajutorul claselor css bootstrap(container, row, col-\*-\*).

În figura14 avem o dispunere prin care putem vedea cum a fost gândit sistemul Grid pentru telefoane mobile.

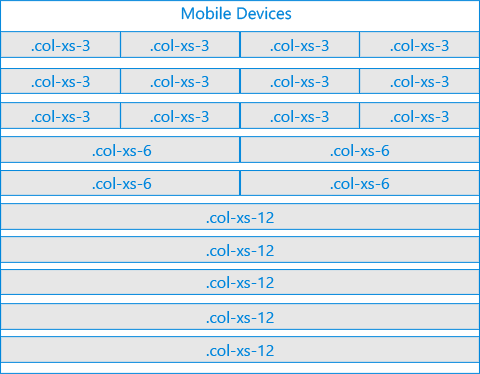


Figura 14. Sistem Column Grid Bootstrap CSS

# Unelte folosite

În cadrul proiectelor, pentru dezvoltarea aplicaţiei web, am folosit un ansamblu de unelte auxiliare. Am folosit pentru partea de hosting un server de tip IIS 7. Pentru partea de testare back-end şi partea de teste de penetrare unealta POSTMAN pentru a simula request-uri asincrone.

Pentru partea de cod de view şi InteliSense în cadrul IDE-ului am folosit Web Essentials 4. ReSharper 9 a fost folosit pentru o mai bună calitate a codului scris pe partea de server şi Code Analysis.

Ca şi IDE s-a folosit Visual Studio 2012 cu tool-urile auxiliare menţionate anterior.

Partea de versionare a aplicaţiei a fost făcută cu ajutorul a două servicii externe. Primul SVN cu server-ul de stocare pe assembla.org, iar cel de-al doilea a fost un repository public pentru deployment care este ţinut pe GitHub.

## IIS (Internet Information Services)

Internet Information Services (IIS) – reprezintă un set de servicii de Internet destinate serverelor care folosesc sistemul de operare Windows. Acest server web ocupă locul 2 în topul celor mai cunoscute servere. Este devansat doar de liderul Apache HTTP Server. În prezent serverele includ următoarele protocoale. FTP, FTPS, SMTP, NNTP, HTTP/HTTPS.

Această unealtă a fost folosită pentru instalarea codului C# pe o instanţă IIS. IIS 7.0 are ca şi funcţionalitate principală comunicarea folosind HTTP / HTTPS a unui anumit server.

Aplicaţia orarului foloseşte un astfel de server pentru a administra cererile făcute cu ajutorul request-urilor AJAX.

IIS-ul a fost scris în C++ şi acum este la cel de-al 10-lea release.

Versiuni ale IIS:

* + IIS 1.0 – A fost la început ca un add-on free pentru Windows NT 3.5
  + IIS 2.0 – Ataşat Windows NT 4.0
  + IIS 3.0 – A fost inclus in Service pack 2 pentru Windows NT 4.0.
  + IIS 4.0
  + IIS 5.0 – Microsoft adaugă pentru Windows 2000 această variantă de IIS şi de acum IIS suportă WebDAV protocol şi alte mecanisme pentru ASP. De asemenea această versiune renunţă la protocolul Gopher.
  + IIS 5.1 – Pus pe Windows XP, este aproape la fel ca şi 5.0
  + IIS 6.0 – Se adaugă suport pentru IPv6. Vine cu Windows XP x64.
  + IIS 7.0 – Vine pe windows server 2008 cât şi pe Vista
  + IIS 7.5 – Adaugat pentru Windows 7 şi necesită pornirea din side panel-ul din Programs. Vine cu îmbunătăţiri asupra protocolului WebDAV si FTP. Aici s-a introdus suport pt TLS 1.1 şi TLS 1.2.
  + IIS 8.0 – Adăugat pentru Windows 8. Apare suportul pentru semnarea certificatelor SSL.
  + IIS 8.5 – Adăugat pentru Windows 8.1. Adaugă sistem de logging mult mai performant cât cel vechi.
  + IIS 10 – Este ultima versiune şi apare pe Windows 10 şi Windows Server 2016 oferind suport şi pentru HTTP/2.

Pentru a putea avea IIS pe calculatorul nostru, trebuie să activăm acest feature din tab-ul de Features al Microsoft.

Instalare IIS. Primul pas îl reprezintă ajungerea în panoul de control(Control Panel). Următorul pas este să intrăm în colecţia de programe( Programs and Features). După executarea acestor paşi, vom vedea în partea de sus următorul link(Turn on/off features). Ajunşi acolo, va trebui să intrăm pe link-ul oferit şi să ajungem la Internet Information Services. Aici trebuie să bifăm tot ceea ce ţine de web service şi mai apoi la final apăsăm Ok.Pentru a verifica dacă IIS s-a instalat corect, va trebui să scriem într-un browser <http://localhost/> . În imaginea de mai jos vom vedea imaginea ce trebuie să ne apară.



Figura 15. IIS pagina pornire

## POSTMAN

Testarea unui API este o parte din testarea software care implică programarea anumitor teste pe care trebuie să le rulăm cu un program fie scris de noi fie un program stand-alone. În cazul meu am ales să-mi testez web api-ul cu Postman. Postman este un program stand-alone prin care simulăm anumite request-uri XHR ale clienţilor pentru a testa stabilitatea şi anumite funcţionalităţi ale server-ului nostru.

În figura 16 putem observa cum arată un request.

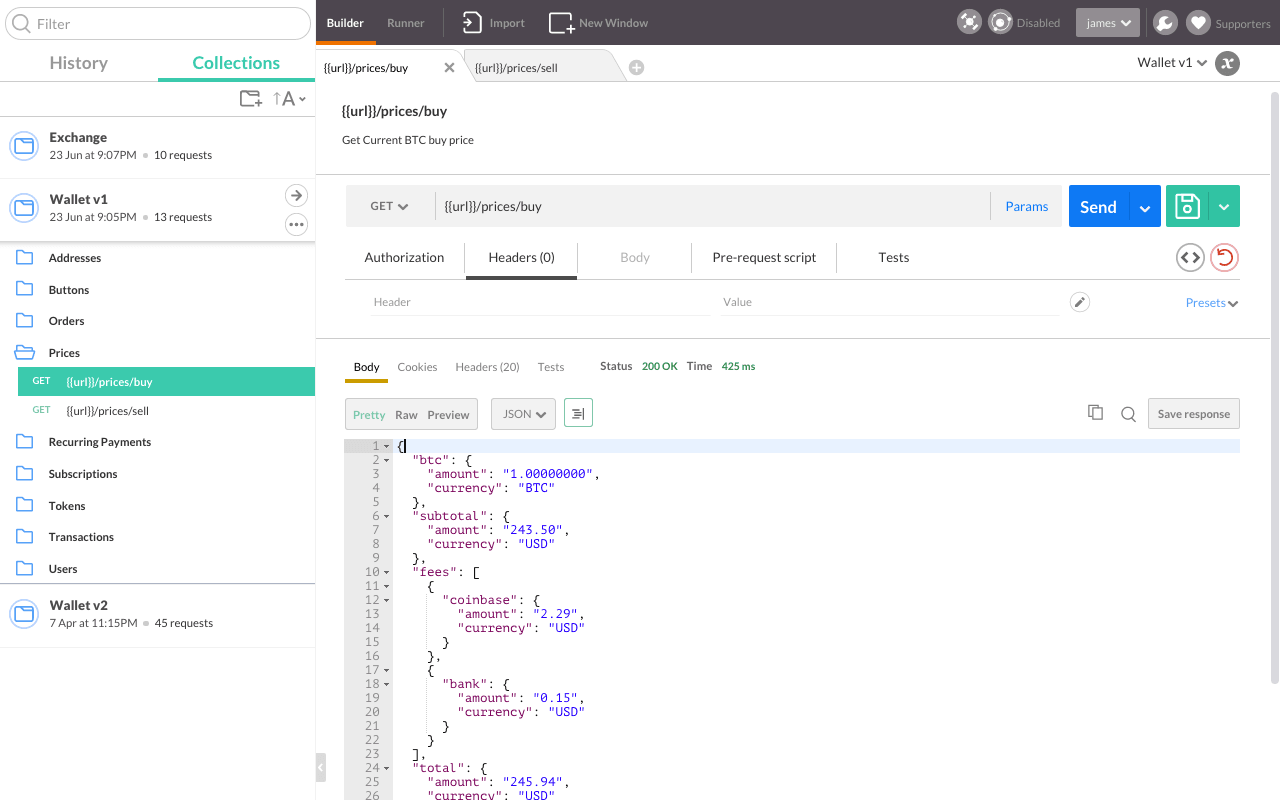


Figura 16. Postman request

În partea de sus avem URL-ul pe care server-ul trebuie să- routeze. Mai jos de URL avem un dropdown, prin care alegem tipul request-ului. Tipul request-ului poate fi: GET, POST, PUT, DELETE. În secţiunile aferente se setează anumiţi parametrii ce intră în header-ul request-ului HTTP.

Pentru ca simularea să fie reală putem trimite chiar payload direct serializat. Trimitem ca şi RAW payload un json către un anumit endpoint iar server-ul va trebui să ne răspundă.

Un alt avantaj al folosirii Postman-ului este că nu trebuie ca dezvoltatorii de pe partea de front-end să facă partea lor prima dată ca mai apoi să vină dezvoltatorii de back-end să pună în funcţiune site-ul. Tot ceea ce se doreşte a se simula se poate face cu această unealtă.

În poză se observă un răspuns de tip json de la server. Cazul de mai sus este cazul fericit, unde totul merge bine pe partea de server, însă există şi cazul nefavorabil în care server-ul nu răspunde cum ne-am aştepta.

În cazul în care server-ul nu răspunde conform aşteptărilor noastre, această unealtă poate fi folosită pentru depanare. Se compară rezultatele cu inputul şi cu ce ar fi trebuit să primim şi ne putem da seama unde este eroarea.

Dacă este vorba de o problemă în structura codului sau în implementare Postman va şti să ne notifice dacă este vreo eroare prin acel Status Code. Aici programatorul trebuie să intervină şi pe baza codului şi a endpoint-ului să-şi dea seama care este problema.

Statusuri HTTP. 1xx sunt codurile http informaţionale. Aceste coduri le primim dacă se cer informaţii în timpul unui request. Lista codurilor 1xx :

* 100 Continue
* 101 Switching Protocols
* 102 Processing(WebDAV;RFC 2518)

Statusuri Http 2xx. Aceste coduri sunt codurile de succes. Dacă primim un cod de acest tip, este clar că request-ul a atins ţinta şi că s-a procesat. Lista codurilor 2xx:

* 200 OK
* 201 Created
* 202 Accepted
* 203 Non-Authoritative Information
* 204 No Content
* 205 Reset Content
* 206 Partial Content
* 207 Multi-Status(WebDAV; RFC)
* 208 Already Reported
* 226 IM USED

Statusuri Http 3xx. Sunt statusuri de redirectare. Clientul va primi un status de redirectare dacă aplicaţia va dori acest lucru. Lista codurilor de redirectare 3xx:

* 300 Multiple Choices
* 301 Moved Permanently
* 302 Found
* 303 See Other
* 304 Not Modified
* 305 Use Proxy
* 306 Switch Proxy
* 307 Temporary Redirect
* 308 Permanent Redirect

Statusuri de eroare client. Sunt statusurile de tip 4xx. Acestea indică o eroare la nivelul client-ului. De cele mai multe ori aplicaţia se aşteaptă la un anumit răspuns din partea server-ului şi din cauze necunoscute nu primeşte răspunsul dorit, semnalând acest lucru printr-un cod de eroare. Lista codurilor de eroare 4xx:

* 400 Bad Request
* 401 Unauthorized
* 402 Payment Required
* 403 Forbidden
* 404 Not Found
* 405 Method Not Allowed
* 406 Not Acceptable
* 407 Proxy Authentication Required
* 408 Request Timeout
* 409 Conflict
* 410 Gone
* 411 Length Required
* 412 Precondition Failed
* 413 Payload Too Large
* 414 URI Too Long
* 415 Unsupported Media Type
* 416 Range Not Satisfiable
* 417 Expectation Failed
* 418 I’m a teapot ☺
* 421 Misdirected Request
* 422 Unprocessable Entity
* 423 Locked
* 424 Failed Dependency
* 426 Upgrade Required
* 428 Precondition Required
* 429 Too Many Requests
* 431 Request Header Fields Too Large
* 451 Unavailable for Legal Reasons

Statusuri de tip eroare server 5xx. Aceste statusuri vor să ilustreze dezvoltatorului că server-ul a întâmpinat nişte probleme la procesarea unui anumit request.

Printr-un status de tip 5xx, atenţia ne va fi îndreptată către server mai mult pentru a vedea de ce request-ul nu s-a finalizat cu succes. Uneori poate fi şi o problemă client-side, dar de cele mai multe ori este o validare omisă pe partea de server.

Lista statusurilor 5xx:

* 500 Internal Server Error
* 501 Not Implemented
* 502 Bad Gateway
* 503 Service Unavailable
* 504 Gateway Timeout
* 505 Http Version Not supported
* 506 Variant Also Negotiates
* 510 Not Extended
* 511 Network Authentication Required

Bineînţeles că aceste liste sunt şi pot fi extinse în funcţie de Api-ul cu care lucrăm. Statusurile sunt destul de self-explanatory, însă pentru o anumită problemă pe care o vom avea la un moment dat întotdeuna este bine să consultăm o resursă avizată pentru a şti ce s-a întâmplat.

## Web Essentials

Web Essentials este o extensie de Visual Studio cu o multitudine de funcţionalităţi pe care web developerii nu le-au avut. De obicei această extensie este folosită în cadrul design-ului unui site.

Dacă dezvoltatorul va dori să scrie CSS, HTML, Javascript, TypeScript, CofeeScrit sau LESS atunci multe din funcţionalităţile WE îi vor fi de folos. Viaţa sa se va simplifica deoarece acest tool are integrat InteliSense pentru JS, TS, Html, CSS.

Pe lângă tool-urile de code-writting, are şi posibilitatea de code-analysis. La capitolul code-writting are code-snippet-uri cu ajutorul cărora codul se va scrie foarte rapid.

## ReSharper 9

Resharper 9 este un utilitar folosit pentru scrierea codului de server în cazul nostru. ReSharper este un tool dedicat dezvoltatorilor care folosesc .NET framework. El este o extensie pentru Visual Studio. Se instalează de pe site-ul oficial al JetBrains, firma producătoare.

Acest tool este free pentru studenţi. El vine cu anumite code-snippet-uri şi template-uri de clase şi un InteliSense mai bun decât cel de bază.

Am ales tool pentru code-writting pe partea de server deoarece ajută foarte mult productivitatea.

## Visual Studio 2012

Visual Studio este un IDE. Ce este un IDE ? Este un mediu de dezvoltare software sau direct din abreviere, mediu integrat de dezvoltare(Integrated development envinroment).

Acesta conţine o suită de programe care ajută programatorul în scrierea programelor. Un mediu de dezvoltare combină toţi paşii necesari creării unui program. Pornind de la editarea surselor, compilarea lor, depanare, testare si generare de documentaţie. Toate acestea sunt cuprinse într-un singur soft, care de regulă are o interfaţă grafică cât mai prietenoasă.

Principalele componente ale unui mediu de dezvoltare sunt editorul de cod şi depanatorul(debugger). Mediile de dezvoltare apelează compilatoare sau interpretoare, care pot veni în acelaşi pachet cu mediul însuşi, sau pot fi instalate separat. Printre facilităţile prezente în mediile de dezvoltare mai sofisticate se numără. Exploratoare de cod, analizoare, sisteme de control al versiunii, designere, etc…

De obicei mediile de dezvoltare sunt specifice unui limbaj anume de programare însă aici Visual Studio face rabat de la regulă, el permiţând scrierea multor limbaje, cum ar fi( Javascript, Html, CSS, C# , C++, C, Python, F#, VB, etc).

VisualStudio include un set complet de instrumente de dezvoltare pentru generarea de aplicații ASP.NET, Servicii Web XML, aplicații desktop și aplicații mobile. Visual Basic, Visual C++, Visual C# și Visual J# toate folosesc același mediu de dezvoltare integrat (IDE) care le permite partajarea instrumentelor și facilitează crearea de soluții folosind mai multe limbaje de programare. Aceste limbaje permit să beneficieze de caracteristicile .NET Framework care oferă acces la tehnologii cheie care simplifica dezvoltarea de aplicații web ASP și XML Web Services cu Visual Web.

**Istoric al versiunilor Visual Studio**.

**Visual Studio 4.0**

Înainte de lansarea versiunii 4 produsele Visual Basic, Visual C + +, Visual FoxPro și SourceSafe s-au vândut separat. Visual Basic a fost disponibil în acel moment deja la versiunea 3. Scopul a fost de introducere pentru Windows 95 îmbinarea în mod corespunzător a produselor.

**Visual Studio 97**

Visual Studio 97 poartă numele de cod Boston. Visual Studio 97 era disponibilă în edițiile Professional Edition și Enterprise Edition. Visual Basic 5.0, Visual C++ 5.0 pentru programarea Windows; Visual J++ 1.1 pentru programarea Java; Visual FoxPro 5.0 pentru dezvoltarea de date de baze. Visual InterDev pentru crearea paginilor web utilizând Active Server Pages.

**Visual Studio 6.0 (1998)**

Visual Studio 6.0 are numele de cod Aspen și este ultima versiune care rulează pe platforma 9x. Include suport pentru Internet Explorer 4.0, control HTML dinamic, Windows NT 5.0 și componente de infrastructură precum Microsoft Message Queue server, serviciul Active Directory și tehnologia Microsoft's Zero Administration.

Era comercializat în două ediții Professional și Enterprises. Versiunea Enterprises include caracteristici care nu regăsim în versiunea Professional precum:

* Application Performance Explorer
* Automation Manager
* Microsoft Visual Modeler
* RemAuto Connection Manager
* Visual Studio Analyzer

**Visual Studio .NET 2002**

A fost lansat în februarie 2002 și poartă numele de cod Rainier. Este disponibilă în 4 ediții: Academic, Professional, Enterprise Developer și Enterprise Architect. Microsoft a introdus un limbaj de programare nou denumit Microsoft C # care se bazează pe .Net Framework. Este prima versiune de Visual Studio care se bazează pe platforma Windows NT.

**Visual Studio .NET 2003**

Lansat în aprilie 2003 este un upgrade minor pentru Visual Studio .Net 2002 cu numele de cod Everett. Acesta include un upgrade la .Net Framework versiunea 1.1 și este prima versiune care sprijină dispozitivele mobile utilizând ASP.NET sau .NET Compact Framework. În afară C++, C# și Visual Basic include o versiune .NET a Java numit J # (cu care nu se pot ținti serverele J2EE). Prin intermediul NET CLR (Common Language Runtime) se pot încorpora codul din limbajele cum ar fi COBOL, Fortran și mai puțin cunoscutele Curriculum Mondrian, Oberon, și Oz. Este disponibilă în 4 ediții: Academic, Professional, Enterprise Developer și Enterprise Architect.Noul instrument include conectivitatea cu bazele de date sistemele Oracle cu caracteristici îmbunătățite de securitate.

**Visual Studio 2005**

Visual Studio 2005 are numele de cod Whidbey. Versiunea suportă Windows 2000 sau versiuni mai noi. Pe data de 14 decembrie 2006 Microsoft a lansat primul Service Pack pentru Visual Studio 2005. Visual C + + 2005 susține compilarea pentru x86-64 (AMD64 și Intel 64) și IA-64 (Itanium).

**Visual Studio 2008**

Visual Studio 2008 are numele de cod Orcas și este prima versiune care permite dezvoltatorilor să folosească versiuni de .NET Framework. Visual Studio este bazat pe platforma de design .NET Framework 3.5, setul de instrumente .NET 3.0 a introdus versiuni actualizate ale ASP.NET, ADO.NET, Visual Basic, C# și CLR.Pentru interoperabilitatea și gestionarea codului mașină, Visual C++ introduce STL/CLR, care este un port de C++ Standard Template Library (STL) conține containere și algoritmi pentru gestionarea codului. Microsoft a lansat primul Service Pack pentru Visual Studio 2008 pe data de 11 august 2008.

**Visual Studio 2010**

Visual Studio 2010 are numele de cod Dev10și a fost lansat pe 12 aprilie 2010 alături de .NET Framework 4. Visual Studio 2010 are un editor nou care utilizează WPF (Windows Presentation Foundation), sprină interfața de tip Ribbon, suportă monitoare multiple, Windows 7 multitouch, funcționalitatea SharePoint, instrumente de Windows Azure și IntelliTrace, un nou produs care ajută la eradicarea bug-urilor irepetabile. Va veni furnizată împreună cu Expression Studio, Business & Enterprise Servers și Microsoft Office și în versiunile Ultimate și Premium.

**Visual Studio 2012**

Visual Studio 2012, dezvoltat de Microsoft şi scos pe piaţă la data de 30.03.2012. A fost scris în C++ şi C# şi a avut ca şi sistem de operare Windows 7 şi succesorii săi. Este disponibil în chineză, engleză, franceză, italiană, japoneză, koreană, spaniolă şi rusă.

Acesta vine cu support mult mai bogat pe partea de UI Desktop pentru WPF, dar şi cu un suport consistent pentru partea de aplicaţii web. I se adaugă tool-uri de design mult mai client friendly. Class Designer, Data designer, mapping designer, WPF Designer, Web Designer.

Am ales această versiune de Visual Studio deoarece este cea mai stabilă la momentul actual.

**Visual Studio 2015**

Este ultima versiune de Visual Studio apărută pe piaţă. Vine la pachet şi cu funcţionalitatea nouă de creare de proiecte XAMARIN, Python Django, F#, etc.

## Versionare

Controlul versiunilor (din engleză: *version control* sau *revision control*) este un domeniu software care se ocupă cu gestionarea mai multor versiuni (numite și revizii) ale unor fișiere. Este aplicată cu predilecție în programare, cu scopul de a păstra versiuni succesive ale codului sursă al unui program de calculator. O soluție ar fi arhivarea separată și completă a fiecărei versiuni a programului într-o bază de date (pe un purtător de date extern), dar această metodă ar necesita în general prea mult spațiu de memorie. În locul ei se utilizează tehnici speciale, care reduc memoria totală necesară și care facilitează reconstrucția „în zbor”, la cerere, a oricărei versiuni din istoria programului.

Istoria uneltelor de Control al versiunilor poate fi impărţita in trei generaţii:

Prima generaţie

Prima generaţie de unelte pentru controlul versiunilor foloseau/versionau cate un singur fişier si nu aveau o corespundere între diferite fişiere din *repository*. Acestea nu aveau suport pentru reţea.

Exemple de astfel de unelte: Source Code Control System (SCCS), Revision Control System (RCS).

A doua generaţie

A doua generaţie de unelte pentru controlul versiunii stochează mai multe fişiere și menține un istoric al versiunilor prin nodurile ramurii.

Exemple de astfel de unelte: Concurrent Versions System (CVS), Subversion (SVN), TFS, Perforce, SVK, VSS.

A treia generaţie

A treia generaţie de unelete pentru controlul versiunilor stochează fișierele în ramuri separate și permit interclasarea acestora cu ajutorul uneltelor de diferențiere și interclasare.

Exemple de astfel de unelte: git, BitKeeper (BK), Bazaar.

Terminologie

**repository**

„depozitul“ în care sunt păstrate fișierele curente și versiunile anterioare. Deseori acest depozit este o bază de date găzduită pe un server.

**working copy (copie de lucru)**

copie a fișierelor din *repository* pe calculatorul de lucru al unui dezvoltator (de unde și numele). Acestea sunt fișierele pe care lucrează un dezvoltator în mod obișnuit.

**check-out**

operația de creare a unei copii de lucru luate din *repository*

**commit sau check-in**

operația de introducere în *repository* a schimbărilor din copia de lucru

**update (actualizare)**

introducerea în copia de lucru a schimbărilor făcute de alte persoane (colegi la același proiect) la *repository*

**branch (ramificare)**

bifurcarea unui set de fișiere în două căi de dezvoltare distincte

**merge (integrare)**

unirea a două versiuni diferite ale unui aceluiași fișier într-o singură versiune

**tag**

o „etichetă“ aplicată fișierelor din *repository* la un anumit moment important din "viața" programului, de exemplu la lansarea unui produs

Pentru proiectul acesta, s-au folosit 2 sisteme de versionare. Primul a fost SVN pentru management-ul privat al versiunii, iar pentru release version am folosit GIT. Ca și server pentru versionare am folosit assembla pe partea de SVN împreună cu TortoiseSVN ca şi soft pentru administrarea surselor.

Pentru partea de release version s-a folosit GIT împreună cu GitHub Desktop version pentru administrarea versiunii de release.

### SVN

Apache Subversion(cunoscut în trecut sub numele de Subversion) este un sistem de revision control fondat și sponsorizat în anul 2000 de firma ColabNet Inc. Este folosit pentru menținerea versiunilor curente și istorice ale fișierelor cod sursă, paginilor web și a documentației în proiectele software. A fost creat ca un înlocuitor modern al sistemului Concurrent Versioning Sytem.

Subversion, sau pe scurt svn, este unul din principalele sisteme revision control folosite în dezvoltarea software liber. Este folosit de proiecte sau organizații precum Apache Software Foundation, Free Fascal, FreeBSD, GCC, Django, Ruby, Mono, Source Forge, Tigris, PHP, Python și Media Wiki.

Subversion este folosit pe larg și în dezvoltarea de software proprietar. Conform unui raport din 2007 al organizației Forrester Research, Subversion este leaderul absolut al categoriei Standalone Software Configuration Management (SCM) și unul leaderii categoriei Software Configuration și Change Management (SCCM).

Subversion este publicat sub licența Apache și este considerat software liber. Pentru utilizarea sistemului SVN s-a folosit server-ul Assembla şi tool-ul TortoiseSVN.

Ce este TortoiseSVN? Este un tool auxiliar pentru sistemul de versionare bazat pe SVN. Autorul acestui tool este Tim Kemp. TortoiseSVN a fost lansat în anul 2002 şi este scris în C++. Este făcut pentru sistemul de operare Windows. El este o extensie Shell care ajută programatorul să execute operaţiile uzuale de subversioning pe un anumit proiect.

Eu am folosit acest tool prin interfaţa pe care o pune la dispozitie. După instalarea lui calculatorul necesită un restart ca programul să se ataşeze explorer-ului şi prin simpla apăsare a click-ului dreapta, să avem funcţionalităţile repository-ului la îndemână(în folder-ul de check-out).

În figura 17 avem prezentate câteva funcţionalităţi ale Tortoise-ului (SVN).

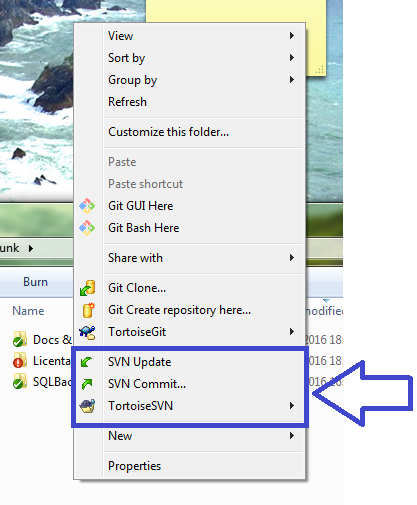


Figura 17. Tortoise SVN funcţionalităţi de bază

După cum spuneam în alineatele de mai sus, observăm funcţionalităţile de bază ale uneltei, şi anume (Update, Commit). Cele două funcţionalităţi reprezintă cererile de pe server, respectiv request-urile către server cu cod sursă.

Pentru ca noi să luăm codul pe care alţi dezvoltatori l-au scris şi să dezvoltăm peste el, va trebui în folder-ul ROOT al repository-ului să dăm click dreapta şi să selectăm SVN Update. Odată făcută cererea se va încerca să se primească ultima versiune încărcată pe server. Dacă în cadrul acestei operaţiuni apar conflicte între fişierele noastre modificate şi fişierele de pe server, atunci va trebui să facem operaţiunea de conflict solving(rezolvare de conflicte între fişierele locale cu cele de pe server).

În ce constă operaţiunea de rezolvare de conflicte ? Această funcţionalitate constă în păstrarea fişierelor sau alterarea lor cu partea de pe server în scopul de a nu pierde funcţionalitate scrisă local în momentul update-ului. În cadrul operaţiunii de conflict solving există 3 opţiuni. Prima opţiune ar fi să se păstreze fişierul local. Cea de-a doua funcţionalitate ar fi păstrarea fişierului de pe server. Ultima opţiune ar fi alterarea fişierului local cu cel de pe server. Operaţiune care presupune un tool secundar de diff. Acest tool de diff există integrat în TortoiseSVN şi se numeşte TortoiseDiff şi face diferenţa între cele două fişiere. Acolo există posibilitatea alterării fişierului local.

Cealaltă funcţionalitate de bază (commit-ul) este operaţiunea prin care încărcăm pe un server aflat la distanţă versiunea noastră. Această funcţionalitate creează un nod nou în lanţul branch-ului în care ne aflăm.

Ce presupune commit-ul ? Constă în selectarea fişierelor pe care dorim să le încărcăm într-un nou nod. Această operaţiune nu se poate face dacă nu ne aflăm pe ultimul nod ca şi nivel cronologic.

Ultimul nod reprezintă ultima versiune a aplicaţiei. Dacă nu ne vom afla cu fişierele pe care dorim să le încărcăm în ultima versiune, atunci SVN-ul ne va răspunde cu un mesaj de eroare. Acest mesaj de eroare ne va spune să updatăm codul sursă astfel încât să putem încărca modificările noastre.

În figura 18 vom vedea un exemplu de commit. Există 3 secţiuni în cadrul commit-ului. Prima secţiune este mesajul pe care îl vom lăsa în log pentru dezvoltatorii care doresc să preia ultima versiune. Cea de-a doua secţiune este cea în care vedem fişierele modificate. Aici putem vedea modificările pe care le-am făcut utilizând unealta de diff. Modificările vor apărea cu + iar cu - vor apărea liniile de cod şterse.

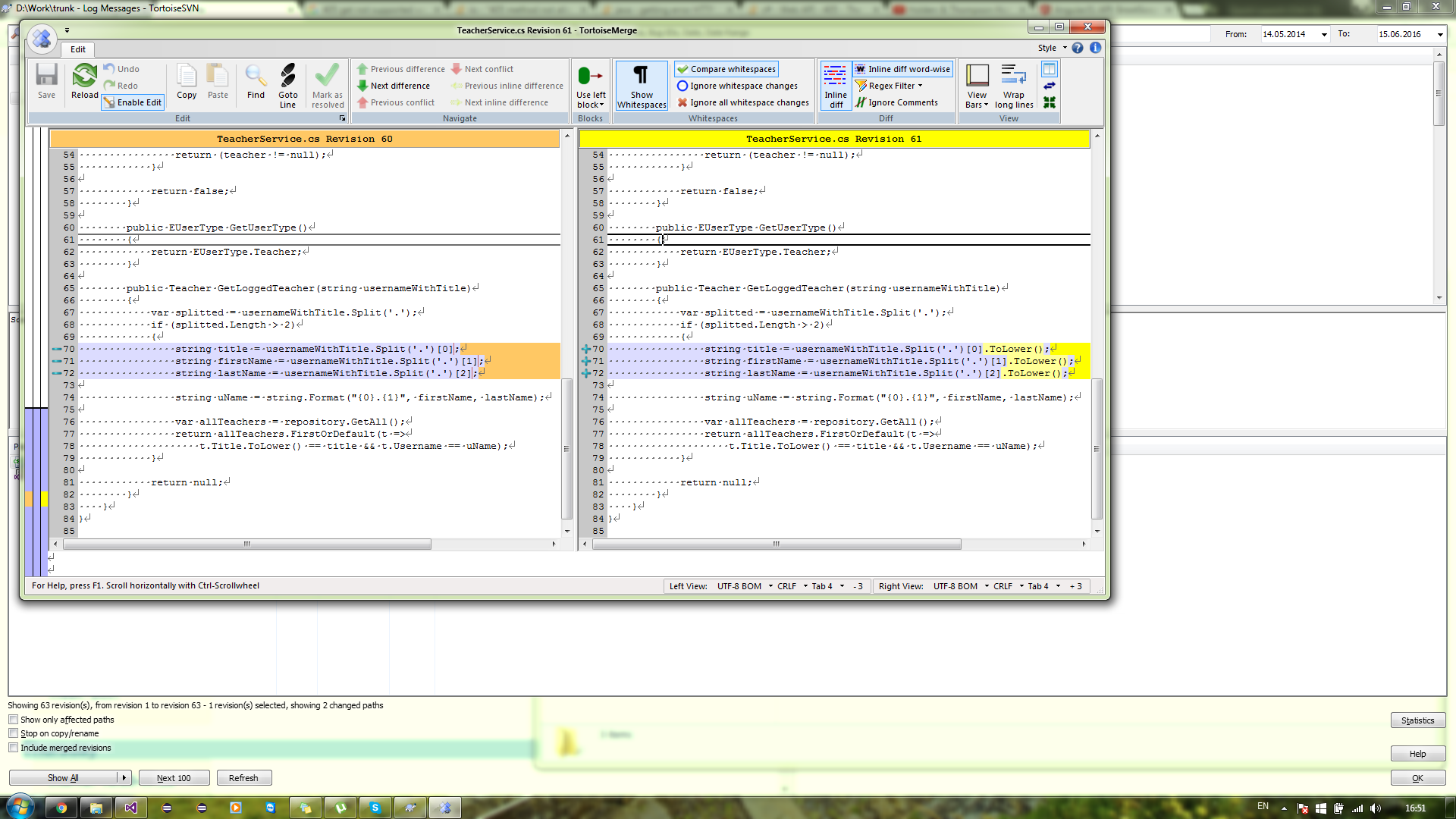
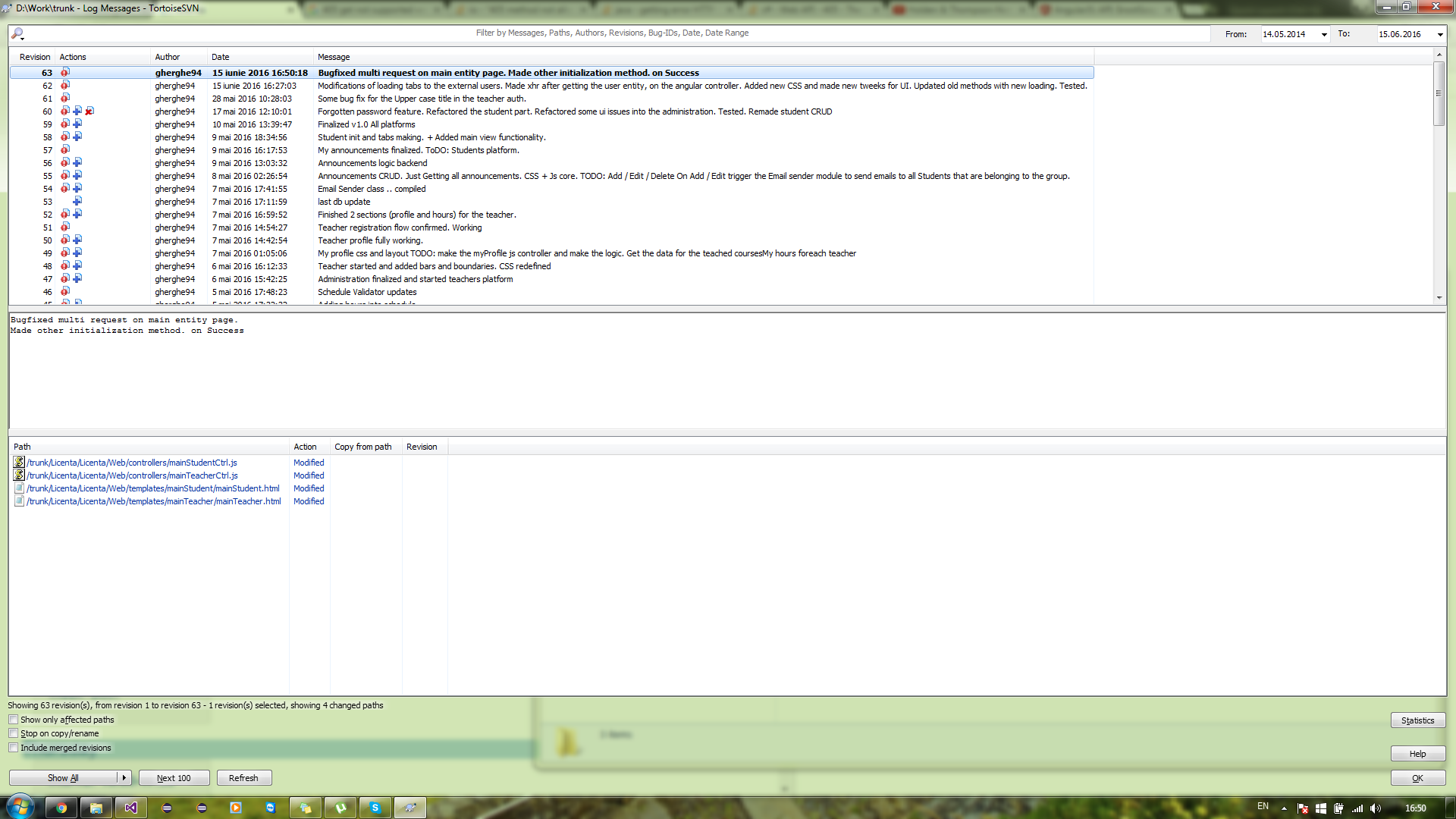


Figura 18. Diff Tool Tortoise SVN

O altă funcţionalitate impresionantă a SVN-ului este Log-ul. Log-ul indică ultimele activităţi asupra soluţiei aflate în dezvoltare. Aceste informaţii apar într-o fereastră din cadrul TortoiseSVN-ului. Dând click pe unul din changeset-uri/commit-uri vom putea vedea fişierele adăugate sau modificate.

În figura următoare avem un exemplu de Log din proiectul nostru. Vom vedea fiecare commit împreună cu modificările aduse soluţiei.



### Git

**Git** este un sistem revision control care rulează pe majoritatea platformelor, inclusiv Linux, POSIX, Windows și OS X. Ca și Mercurial, Git este un sistem distribuit și nu întreține o bază de date comună. Este folosit în echipe de dezvoltare mari, în care membrii echipei acționează oarecum independent și sunt răspândiți pe o arie geografică mare.

Git este dezvoltat și întreținut de Junio Hamano şi Linus Torvalds(fondator Linux), fiind publicat sub licență GPL și este considerat software liber.

**Istoric GiT.**

Dezvoltarea Git a început după ce mai mulți developeri ai nucleului Linux au ales să renunțe la sistemul de revision control proprietar BitKeeper. Posibilitatea de a utiliza BitKeeper gratuit a fost retrasă după ce titularul drepturilor de autor a afirmat că Andrew Tridgell a încălcat licența BitKeeper prin acțiunile sale de inginerie inversă. La conferința Linux.Conf.Au 2005, Tridgell a demonstrat în timpul discursului său că procesul de inginerie inversă pe care l-a folosit a fost pur și simplu o sesiune telnet pe portul corespunzător al serverului BitKeeper și rularea comenzii *help* pe server.

Controversa a dus la o renunțarea rapidă la sistemul BitKeeper care a fost înlocuit cu un nou sistem intitulat Git construit special pentru scopul de revision control în cadrul proiectului Linux kernel. Dezvoltarea noului sistem a fost începută de Linus Torvalds în 3 aprilie 2005pentru a fi anunțat câteva zile mai târziu (aprilie 6) pe lista de email a proiectului Linux kernel. O zi mai târziu, noul sistem a început să fie folosit pentru dezvoltarea actuală de cod pentru proiectul Git.Primele operații merge a avut loc pe data de 18 aprilie. În data de 16 iunie, versiunea 2.6.12 Linux kernel a fost pusă în Gitcare continuă și în ziua de azi să fie sistemul revision control folosit de proiectul Linux kernel.

Tot în această perioadă, și tot cu scopul de a înlocui BitKeeper, a fost creat sistemul Mercurial.

Principala diferență dintre Git și oricare alte sisteme de versionare (Subversion și prietenii săi inclusiv) este modul în care Git își gestionează datele. Conceptual, majoritatea celorlalte sisteme își stochează informațiile ca o listă de schimbări asupra fișierelor. Aceste sisteme (CVS, Subversion, Perforce, Bazaar și altele) văd informațiile ca o mulțime de fișiere și schimbările asupra fișierelor în timp, după cum este ilustrat în Figura 19.



Figura 19. Git check-in system

Git nu vede și nici nu stochează datele în acest mod. În schimb, Git consideră datele sale mai mult ca o mulțime de instantanee (snapshots [en]) ale unu mini sistem de fișiere. De fiecare dată când faceți commit, sau salvați starea proiectului dumneavoastră în Git, acesta practic salvează o poză a stării curente a tuturor fișierelor din acel moment și stochează o referință la acel instantaneu. Pentru a fi eficient, dacă există fișiere care nu s-au schimbat, Git nu stochează fișierul iarași ci doar o legătură către fișierul anterior stocat identic cu cel din prezent. Git vede datele stocate similar cu Figura 20.



Figura 20. Check-in in time

Aceasta este o distincție importantă dintre Git și aproape toate celelalte VCS. Aceasta face ca Git să reconsidere fiecare aspect al controlului versiunilor pe care majoritatea sistemelor le-au copiat de la generația anterioară. Acest aspect face ca Git să fie mult asemănător cu un mini sistem de fișiere cu niște unelte incredibil de utile adăugate peste el, comparativ cu un simplu VCS, Vom analiza unele dintre beneficiile câștigate prin a vedea datele voastre în acest fel atunci când ne vom ocupa de crearea ramurilor (branches [en]).

O altă principală diferenţă între GIT şi SVN este că Stagingul este făcut diferit. Există o diferenţă în cadrul încărcării pe server. Încărcarea este diferită din cauza procesului de commit local. Acest commit local nu încarcă un nou nod pe ramura în lucru ci, creează un snapshot local.

Pentru ca datele să fie persistate, se va dori extragerea de pe server a ultimei versiuni de pe ramura în lucru şi rezolvarea conflictelor ca mai apoi să se facă acţiunea de push.

Diferența principilă în cadrul acțiunii de commit la GIT este că la commit, GIT-ul va face un snapshot local, față de SVN unde commit-ul încărca codul sursă direct pe server. GIT-ul la push va lua ultima versiune locală și va încerca încărcarea dacă aceasta este posibilă.

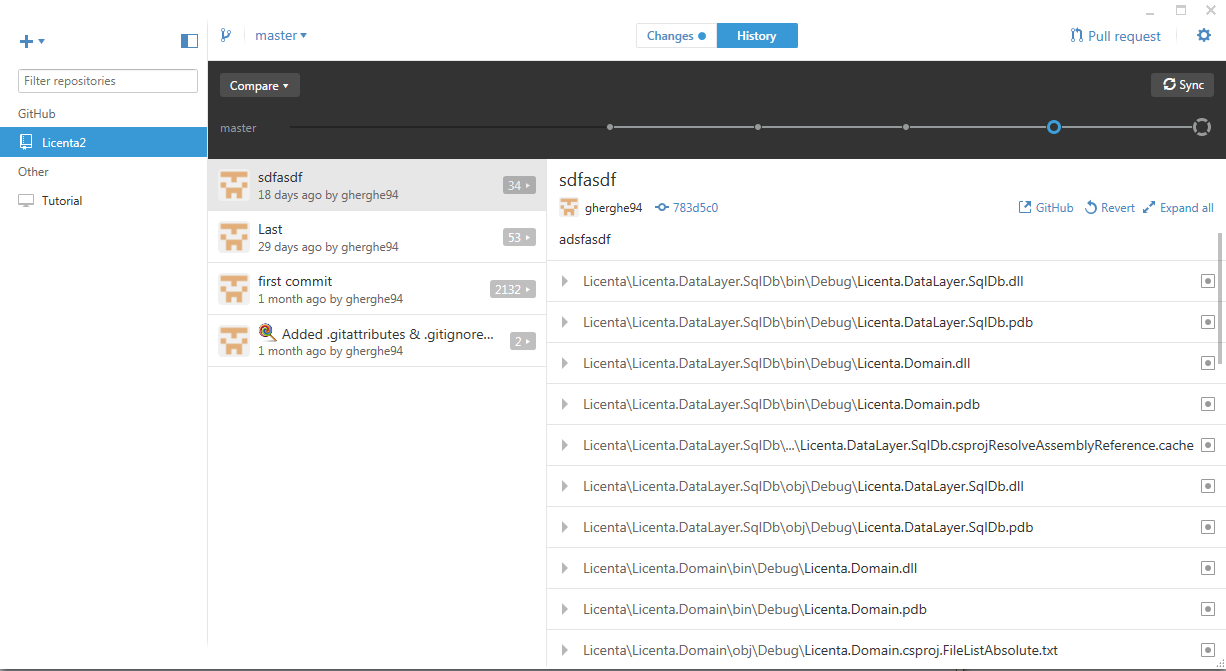
În cadrul proiectului am lucrat cu GitHub ca și client desktop pentru operarea cu GIT.

Figura 21. GitHub aplicație

În figura 21 putem vedea client-ul numit GitHub. În partea stângă a aplicației, vedem repository-urile user-ului autentificat. Prin selectarea unui repository în partea stângă ni se vor încărca datele despre ultimele push-uri. În partea de sus reprezentat cu gri putem observa un timeline pe care sunt reprezentate nodurile din branch-ul curent.

Tot ca și la SVN, Git-ul are uneltele in-place pentru comparare și rezolvare de conflicte. GIT-ul este de preferat dacă proiectul va fi de anvergură. Dacă proiectul este mare sau foarte mare se preferă GIT-ul pentru că oferă un suport special pentru MERGE-ul branch-urilor. Merge-ul în cadrul GIT-ului se face fie prin interfața grafică, fie prin GIT Shell.

Spre exemplu în cadrul unui merge vom vedea necesitatea tool-urilor internet a GitHub(desktop app).

În figura 22 vom putea vedea tool-ul de diff în acțiune. Tool-ul funcționează exact ca și tool-ul prezent în TortoiseSVN. Alte Tool-uri care pot fi integrate ca și plug-in. WinDiff, SurfDiff, etc.

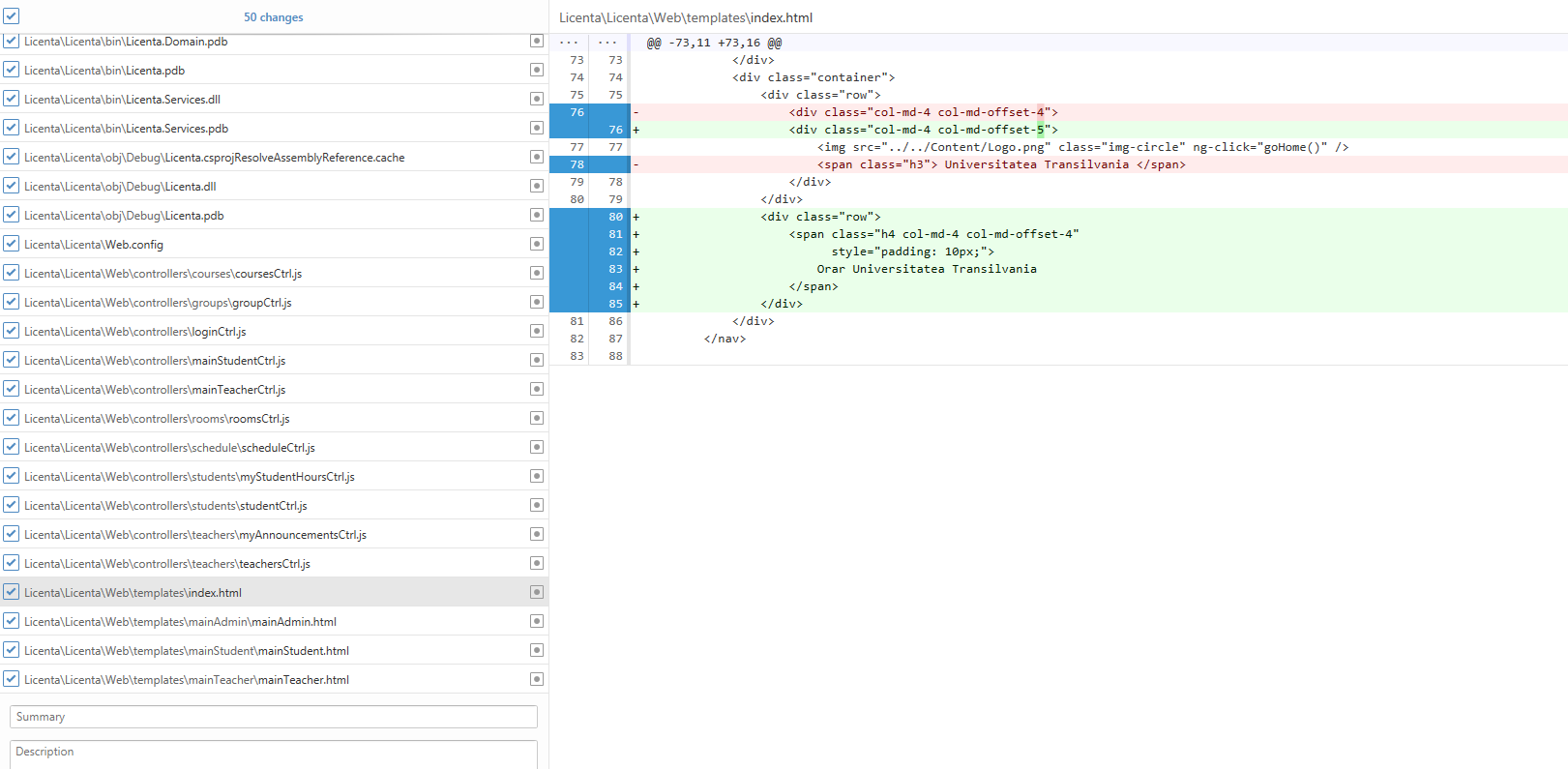


Figura 22. Git Diff

# Soluție Tehnică a proiectului

În cadrul proiectului, ca și soluții tehnice am utilizat numai tehnologii aflate în plin apogeu în anul dezvoltării aplicației. Aplicația este o aplicație web, cross browser, cu extensie pentru partea de device-uri mobile. Această multitudine de tehnologii au permis implementarea unei soluții cross-platform și cross-browser.

Pentru realizarea view-ului pe partea de front-end am folosit framework-ul Bootstrap. Pentru ca aplicația să poată rula și nativ, partea de server, este realizată ca și un serviciu REST, astfel încât dezvoltarea ulterioară a unei aplicații native să fie la îndemâna oricui.

Partea de client este un SPA(single page application) care poate fi foarte ușor integrat prin realizarea unui Webview în cadrul aplicațiilor native. Tehnologia SPA este foarte folosită la momentul actual deoarece eficientizează lucrul cu partea de front-end, ușurând server-ul de redundanța multitudinilor request-urilor GET pentru partea de HTML/CSS/JS.

Pentru data warehousing s-a folosit o bază de date relațională din cadrul tehnologiilor Microsoft. Am ales modelul relațional deoarece este cel mai cunoscut model de stocare al datelor cât și cel mai ușor de înțeles.

La fel ca și partea de stocare a datelor, partea de server este realizată cu ajutorul tehnologiilor Microsoft. Alegerea celor două tehnologii s-a făcut pe baza performanțelor extraordinare de care au dat dovadă de-a lungul timpului.

Așadar în capitolele ce urmează vom vorbi despre un serviciu REST Web Api 2 susținut pe un server IIS care are o bază de date SQL, ținută pe un server Microsoft Sql Server 2012, iar pe partea de front-end vom discuta despre un SPA realizat cu ajutorul librăriei Angular Javascript 1.5 și librăriei Bootstrap CSS.

## Soluție Tehnică pe parte de server

Cum spuneam, pentru partea de server aplicația rulează pe un IIS care desfășoară o aplicație WEB API 2. Ce este un Web api? Un web API este prin definiție o aplicație interfațată de programare. Un Web API se folosește atunci când aplicația se dorește a se extinde și pe alte platforme/device-uri.

Așadar prin crearea unui serviciu web avem parte de o segregare extraordinară a logicii de păstrare a datelor și a logicii de display al datelor.

De obicei pentru a se putea comunica între client și server trebuie stabilit un limbaj. Acest limbaj are la bază o limbă. Această limbă de cele mai multe ori în cadrul serverelor este fie JSON, fie XML. Aceste două lucruri sunt defapt protocoale de serializare a datelor / obiectelor trimise de către server sau primite de către server.

Aceste mesaje bineînțeles trebuie trimise pe un canal de comunicare securizat. Pentru acest lucru avem nevoie de un protocol numit HTTPS. Web API-urile sunt bazate pe protocolul HTTP. HTTPS este un protocol de comunicare ce extinde HTTP, S-ul venind de la HTTP Secure.

Toate apelurile făcute către un server web, trebuiesc făcute prin intermediul unui URI (Uniform Resource Identifier). Acest URI / URL ne va conduce prin intermediul unui motor de rutare la resursa cerută. Acest mecanism de rutare se numește Routing, iar elementele la care ajungem se numesc resurse și cheia de identificare a resursei se numește Endpoint.

Ce este un Endpoint? Un endpoint este o metodă de cele mai multe ori în cadrul unui Web Api 2 Microsoft. Această metodă trebuie sa facă parte din următoarele tipuri de metode:

(GET, POST, PUT, DELETE). Aceste tipuri de metode reprezintă tipurile pe care o metodă dintr-un serviciu REST o poate lua.

Protocoalele de transport stau la baza oricarui serviciu web( inclusive REST), cel mai folosit protocol de transport fiind HTTP. Ultima versiune a acestui protocol este HTTP/1.1 – versiune de îmbunătăţire şi reparare a neajunsurilor versiunilor anterioare.

La HTTP se pierd informaţiile cererilor vechi (deci este un protocol fără reţinerea stării).

Toate serviciile web folosesc HTTP dar il folosesc in moduri diferite.

Metodele disponibile sunt :

* GET: este cea mai folosită metodă, fiind utilizată atunci când serverului i se cere o resursă.
* HEAD: se comportă exact ca metoda GET, dar serverul returnează doar antetul resursei, ceea ce permite clientului să inspecteze antetul resursei, fără a fi nevoit să obţină şi corpul resursei.
* PUT: metoda este folosită pentru a depune documente pe server, fiind inversul metodei GET.
* POST: a fost proiectată pentru a trimite date de intrare către server.
* DELETE: este opusul metodei PUT.
* TRACE: este o metodă folosită de obicei pentru diagnosticare, putând da mai multe informaţii despre traseul urmat de legătura HTTP, fiecare server proxy adăugându-şi semnătura în antetul Via.
* OPTIONS: este folosită pentru identificarea capacităţilor serverului Web, înainte de a face o cerere.
* CONNECT: este o metodă folosită în general de serverele intermediare.

În cele ce urmează vom povesti despre motorul de routing al aplicației cât și despre autentificarea unui REQUEST.

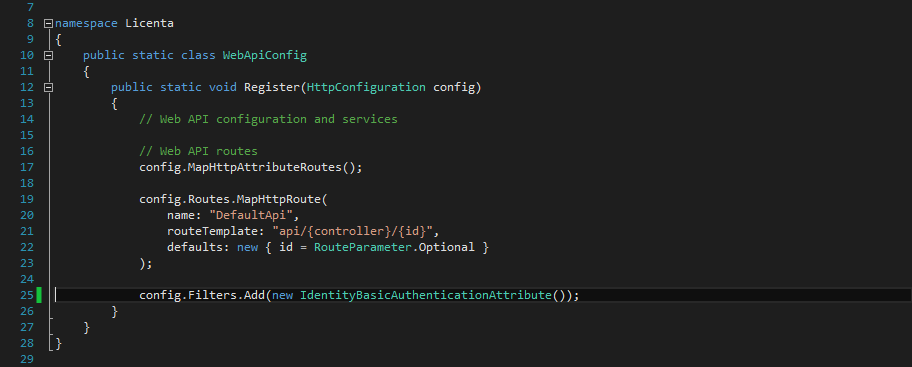


Figura 23. Web Api Routing

După cum putem observa în codul prezentat în Figura 23, rutarea request-urilor se va face pe baza numelui controller-ului apoi pe baza id-ului. Așadar orice metodă va trebui să satisfacă următorul șablon pentru ca server-ul să răspundă. Un exemplu ar fi /api/Groups/Save. Acest request va trebui să lanseze în execuție codul de pe partea de GroupsController și să execute metoda Save care este un POST. În Figura 24 putem observa metoda Save din Groups Controller.

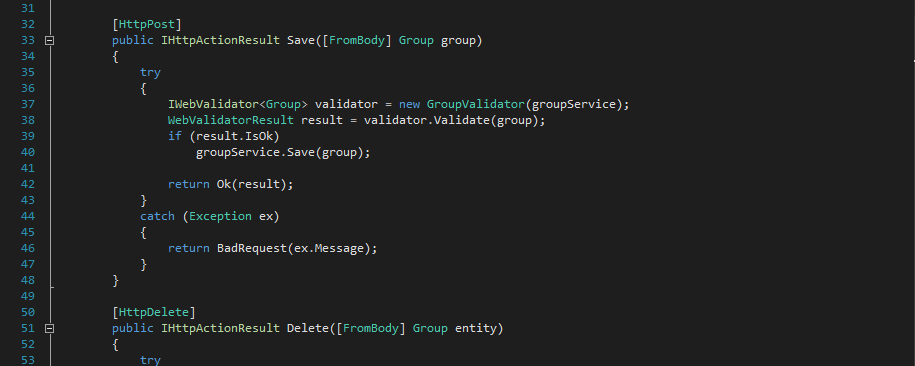


Figura 24. Save (POST)

Observăm cum creăm o metodă de tip POST. Prin atributul specific [HttpPost]. Prin acest atribut notificăm server-ul că în controller-ul nostru am atașat o metodă de tip POST.

Pentru a prelua datele din parametrii vedem acel atribut cu care decorăm Endpoint-ul și anume, FromBody. Acest atribut spune server-ului că acest endpoint va trebui să conțină un JSON care să respecte proprietățile unei grupe. În acest caz server-ul va incerca o deserializare a obiectului primit, iar dacă acea deserializare nu este posibilă atunci obiectul returnat va fi null.

Observăm în figura 23 linia de cod în care se adaugă un filtru pe partea de WebApi. Acel filtru are ca și utilitate interceptarea oricărui REQUEST și trecerea prin acel filtru. Acel filtru pe care îl observăm în Figura 23 este un filtru menit să prevină atacurile din partea hackerilor. IdentityBasicAuthenticationAttribute este un filtru folosit pe fiecare controller pentru autorizarea unui anumit user care dorește access la un anumit endpoint.

Tot în cadrul autorizării și securității pomenim faptul că aplicația are în spate o arhitectură bazată pe Basic Authentication. Această autentificare basic este un token-based authentication care menține un token serializat într-un string base 64 care are datele de autentificare ale utilizatorilor.

Ce este autentificarea de bază?

Http Basic authentication sau BA este o implementare simplistă pentru a controla accesul la anumite resurse în cadrul unei aplicații web. Este de multe ori preferată de către dezvoltatori deoarece nu necesită cookies / session IDs sau orice altceva. Autentificarea de bază se folosește de header-ul HTTP pentru a trimite acest token.

Din punct de vedere al securității acest mecanism nu oferă confidențialitate deoarece credențialele sunt stocate ca un string Base64. Pentru o securitate mai solidă se încurajează implementarea protocolului HTTPS.

Deoarece BA se trimite în header, acest token va fi prezent în fiecare REQUEST care va avea nevoie de autorizare. Pentru implementarea acestei arhitecturi este nevoie de un handshake inițial care se produce la login. Acest Handshake (strângere de mână) se face după ce utilizatorul a fost autentificat cu succes, acesta primind odată cu răspunsul de succes și token-ul serializat.

Avantaje. Dacă un REQUEST anonim nu conține în partea de header acest token serverul va da un mesaj de eroare clientului, și anume clientul va primi codul 401(Unauthorized). În figura 25 vom putea observa header-ul cu proprietatea Authorization.

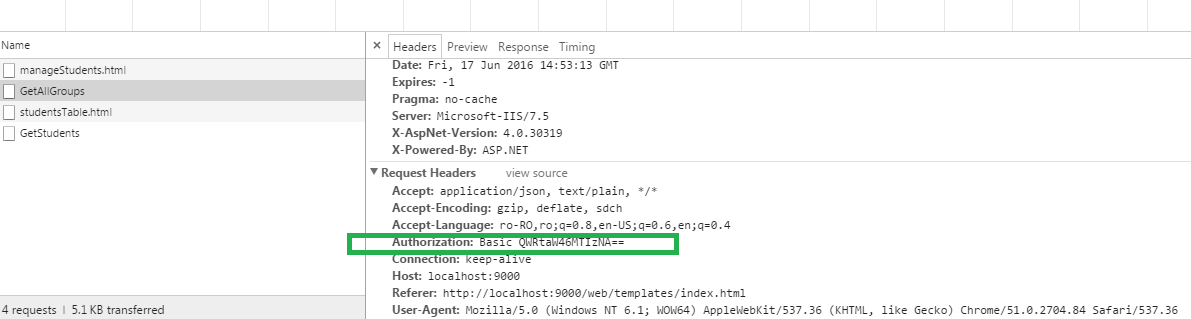


Figura 25. Basic Authorization

Despre partea de implementare a Basic Authentication.

În figura 26 avem prezentată partea de autorizare din punct de vedere al implementării.

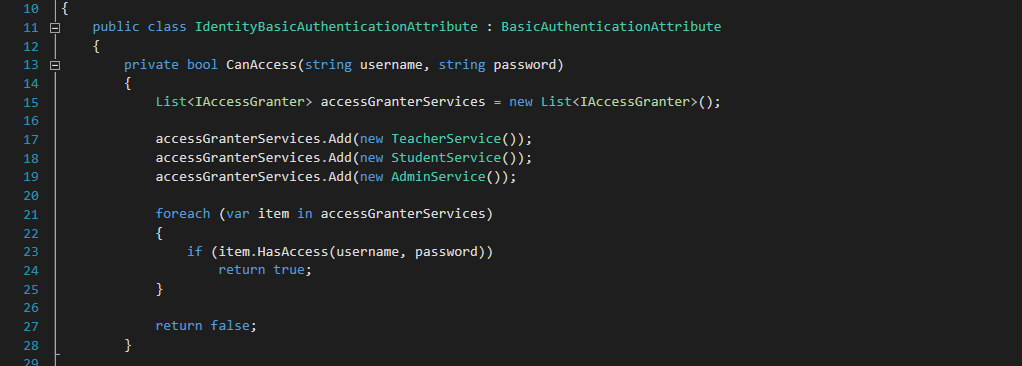


Figura 26. Implementare filtru basic authentication

Observăm că avem trei tipuri de utilizatori. Pentru fiecare dintre utilizatori avem implementată interfața IAccessGranter. Această interfață cuprinde 2 metode. Prima metodă este HasAccess(string username, string password), iar cea de-a doua este GetUserType care întoarce tipul userului. Această implementare a metodei CanAccess va trece prin toți utilizatorii concreți și va stabili dacă vreun utilizator are acele credențiale.

De ce am abordat soluția aceasta? Deoarece fiecare utilizator are tipul său de autentificare. Adminul este unul singur per toată aplicația. Studenții se vor autentifica pe baza prenumelui și a numelui iar profesorii se vor autentifica pe baza titlului, a numelui și al prenumelui. Așadar am interfațat toate serviciile de autentificare astfel încât să putem distinge corect tipul de utilizator și dacă utilizatorul are acces la acea resursă.

Un exemplu de implementare al metodei HasAccess putem observa în figura 27.

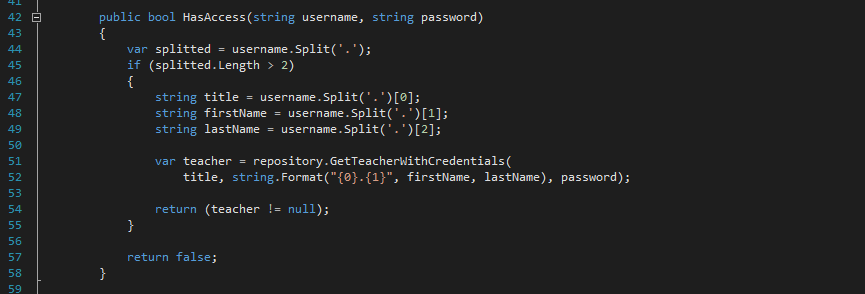


Figura 27. Teacher implementare IAccessGranter

În implementarea din Figura 27 se observă că se dorește ca datele să păstreze un anumit format. Acel format este de tipul [<titlu>.<prenume>.<nume>]. Dacă formatul nu este respectat atunci metoda HasAccess va returna fals.

În cele ce urmează vom discuta despre arhitectura aplicației server. Arhitectura aplicației este bazată pe trei straturi(layere). Stratul server-endpoint, în care se primește REQUEST-ul de la Client. Stratul de servicii în care se face validarea entității primite ca și parametru pentru procesare. Stratul de administrare a datelor.

C:\Users\Andrei\Desktop\Untitled Diagram.png

Figura 28. Request Flow

În figura 28 avem reprezentată o schemă a unui Request care ar necesita o procesare completă. Având client-ul browser, vom spune că un Request pleacă din Angular JS sub forma unui obiect JSON, obiect care este primit pe partea de server IIS de un endpoint Web Api 2, obiect ce este deserializat într-o entitate C# care va fi validată de către stratul de bussiness, ca mai apoi înregistrată în baza de date prin intermediul stratului de administrare de date folosind Micro-ORM-ul NPOCO.

Stratul de bussiness. Acest strat este stratul care se ocupă cu validarea și este un wrapper peste stratul de administrare de date. Aplicația este construită pe baza șablonului de proiectare numit Generic Repository. Acest șablon trebuie regăsit și în partea de bussiness deoarece toate clasele de administrare de date sunt wrapped de către alte clase de tip bussiness (services).

Pentru a putea înțelege mai bine cum funcționează acest concept avem în următoarea figură o reprezentare grafică a claselor de bussiness din proiectul nostru.

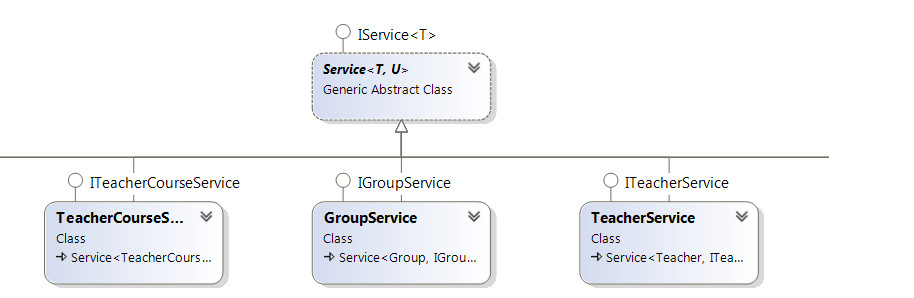


Figura 29. Abstract Service UML

Practic avem o clasă de bază care înglobează marea parte a operațiilor și din care se moștenește fiecare serviciu. Fiecare Entitate va avea un serviciu și fiecare serviciu va avea un repository al său.

Absolut toate clasele au interfețe așadar se pot extinde și nu uită de Open Closed Principle din principiile SOLID. Implementarea clasei Service este implementarea unei clase care permite operații CRUD pe o entitate anume, T fiind modelul pe care se operează iar U repository-ul concret al modelului T.

Clasa Service implementează interfața IService<T> care are metodele:

-GetAll() : List<T>

-Save(T entity) : bool

-Delete(T entity) : bool

-GetById(int id) : T

Așadar orice clasă ce va moșteni această clasă pentru POCO-ul său va putea efectua aceste operații fără a le mai implementa specific.

În figura 30 avem implementarea clasei de bază a proiectului de Bussiness.

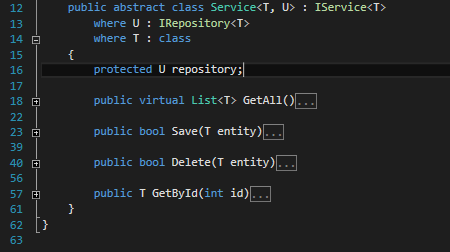


Figura 30. Service class

După cum spuneam anterior, în aceste clase se vor face proceduri de validare, cât și proceduri de salvare prin intermediul repository-ului. În Figura 31 vom observa implementarea Save-ului generic. Observăm validarea aferentă prin aruncarea excepțiilor dacă acestea sunt aruncate.

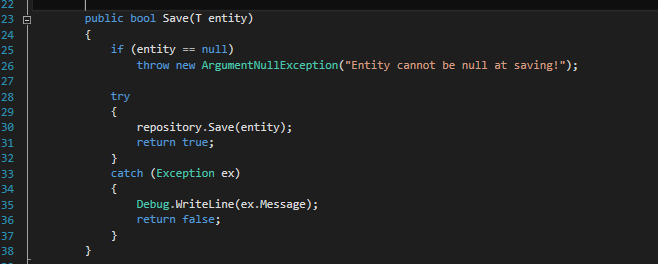


Figura 31. Save Generic

În această metodă putem observa că primim o entitate T despre care nu știm nimic, dar pe care o vom interoga astfel încât să vedem că nu este null și apoi încercăm să trimitem spre baza de date această entitate. Dacă stratul de administrare de date nu va arunca nici o excepție atunci înseamnă ca totul s-a petrecut perfect normal și că entitatea noastră T a ajuns cu success în baza de date.

Stratul de administrare al datelor. Acest strat este stratul care învelește micro-ORM-ul folosit pe proiect. DAL(Data Access Layer) este creat pentru ca dezvoltatorul să se asigure că entitățile ajung în tabelele corespunzătoare iar integritatea datelor să fie păstrată.

Așadar după cum spuneam în capitolul bussiness layer-ului, aplicația se ghidează după renumitul Generic Repository Pattern. Ce este generic Repository? Este un șablon de proiectare ce permite scrierea funcționalităților comune de tip CRUD(Create Read Update Delete) într-o clasă de bază, care mai apoi să fie părinte pentru restul claselor de administrare al datelor. Avem un exemplu în următoarea figură, o diagramă UML prin care observăm cum au fost implementate clasele respective.

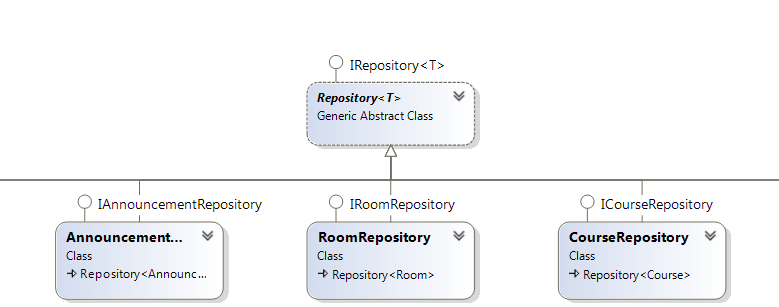


Figura 32. Repository Pattern UML

Pentru implementarea concretă a claselor de tip Repository, am folosit un ORM(Object Relational Mapper).

Ce este un ORM? Este o descriere în cod a bazei de date care pe baza unor atribute puse pe clase va ști cum să administreze entitățile astfel încât să ajungă în tabela corespunzătoare. În cazul ORM-ului nostru s-a folosit NPOCO. NPOCO este un micro ORM care se folosește de atribute specifice pentru a administra entitățile. Implementarea POCO-urilor s-a făcut în proiectul Domain. Ce este un POCO(Plain Old CLR Object)? Este o clasă normală care specifică proprietățile unei entități. În următoarea figură vom observa implementarea unui POCO.

În figura 33 vom observa cum s-a definit o clasă POCO pentru modelarea unei materii.

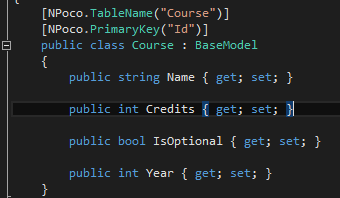


Figura 33. POCO Course - Materia ca obiect

În Figura 33 observăm cum se definește o clasă care să se mapeze pe o tabelă. Deasupra clasei POCO se inserează cele 2 atribute care identifică tabela și cheia primară. Responsabilitatea dezvoltatorului este de a mapa 1 la 1 proprietățile din tabelă cu cele din clasă. Dacă vreo proprietate nu se dorește a se mapa, i se va aplica un atribut specific NPOCO numit Ignore [NPoco.Ignore]. În figura 34 observăm corelarea între obiectul POCO Course și tabela Course. Corelare între Figura 33 și 34.

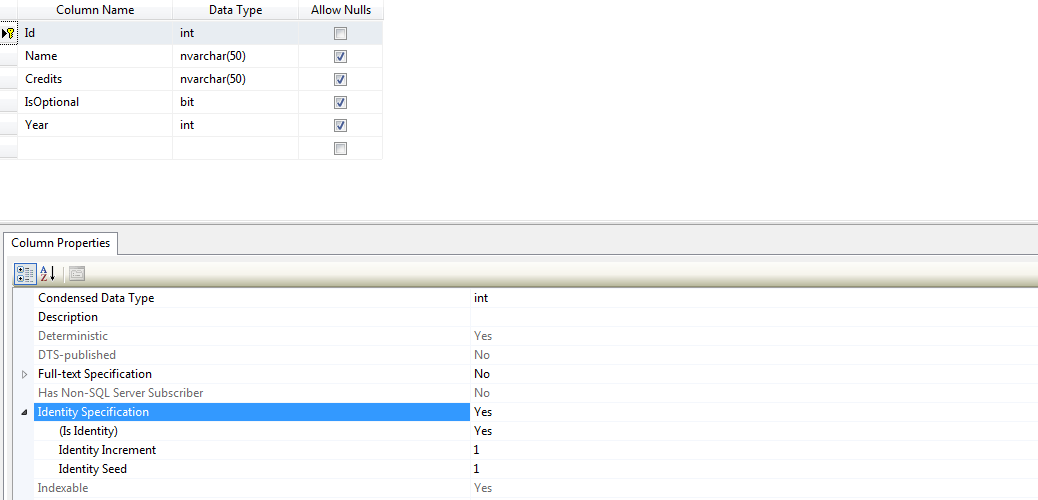


Figura 34. Tabela Course

Un lucru pe care trebuie să-l menționăm este faptul că această conexiune se realizează manual între baza de date și aplicație. Această conexiune se realizează printr-un ConnectionString pus în fișierul de configurări a aplicației.

C:\Users\Andrei\Desktop\Untitled.png

Figura 35. String de conexiune

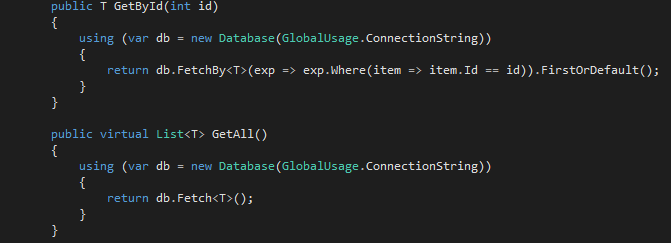


Figura 36. Folosire NPOCO

În figura 36 observăm folosirea efectivă a ORM-ului cu metodele aferente.

## Soluție Tehnică pe parte de client

Pe parte de client am abordat soluția web. Această soluție web a fost abordată din cauza portabilității. Un website dacă este corect conceput poate fi accesat de pe orice device.

Pentru partea tehnică am ales un templating engine pentru aplicații SPA(single page application). Acest engine este Angular JS versiunea 1.5. Angular este un framework foarte puternic de templating în care au fost integrate mecanisme pentru trimiterea de mesaje HTTP prin intermediul request-urilor AJAX. În cele ce urmează vom vedea cum au fost configurate modulele Angular și cum s-a creat aplicația client.

Angular-ul după cum am menționat în capitolele precedente este unul din puținele tool-uri care a fost creeat având la bază ideea de binding. Prin acest șablon (MVVM) dezvoltatorul este ajutat extrem de mult deoarece scapă de orice update de view.

După cum am spus Angular creează aplicații SPA, așadar vom observa în următoarea diagramă cum funcționează engine-ul de routing.

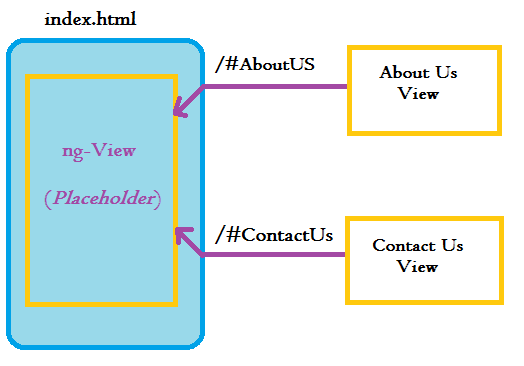


Figura 37. Routing

În aplicația noastră avem un ng-view care funcționează ca un container pentru view-ul care va urma să fie decorat. Din diagramă observăm acel #. Acel # este un marker specific pentru site-urile Angular pentru a marca un entry point al unui ng-view. Practic ce este după # este numele state-ului care trebuie să fie înlocui în părintele care are ng-view-ul. Toate aceste ng-\* sunt directive. Aceste directive sunt cod nativ javascript. Aceste directive sunt scrise în cadrul Angular-ului.

Pentru ca toate acestea să funcționeze Angular are nevoie de un modul specific injectat. De ce trebuie injectat? Angular-ul funcționează pe principiu DI (Dependency Injection) principiu foarte sănătos, deoarece se decuplează foarte mult implementarea de abstractizare. Angular funcționează cu ajutorul modulelor. Ce sunt modulele Angular? Modulele Angular sunt ca niște plug-in-uri pentru aplicația noastră. Sunt ca un fel de librării peste Angular, menite să ajute developer-ul ce dorește să rezolve o anumită problemă.

În cele ce urmează vom observa cum s-a realizat routarea efectivă a ecranelor. Practic Ng-Route este un modul ce trebuie inclus în configurația inițială. Practic acest ng-route este o mașină de stări sau un automat finit. În acest automat finit, se definesc stări în care se poate ajunge pe baza URL-ului.

Definirea stărilor se face cu ajutorul $stateProvider-ului. Exemplul din aplicație este ilustrat în figura 38.



Figura 38. Rute Angular

Observăm și existența altor Provideri. $httpProvider este un serviciu Angular prin care putem seta anumite opțiuni REQUEST-urilor pe care le vom face. Putem observa că la linia 2 se adaugă un interceptor http iar linia 3 și 4 se adaugă pentru Header-ul HTTP un atribut Content-Type. Acel Content Type este specificat astfel încât server-ul să știe ce trimitem. În cazul nostru vom trimite mesaje serializate JSON, iar charset-ul va fi UTF-8(Unicode).

Pentru partea de interceptori vom discuta în cele ce va urma. Interceptorii Angular, sunt niște servicii prin care se va trece fiecare REQUEST. Acest serviciu în cazul nostru se numește authInterceptorService. Serviciul authInterceptorService interoghează fiecare request și îi adaugă pe headerul HTTP acel atribut numit Authorization pentru ca server-ul să autorizeze utilizatorul nostru. Tot în acest serviciu pe partea de răspuns din partea serverului, dacă primit un mesaj cu un cod de eroare 400 aplicația ne va redirecta către login.

Acest mecanism de redirectare se folosește deoarece dacă se vor pierde datele de sesiune cache-uite atunci aplicația nu mai este integră. Așadar dacă ne vom uita într-un REQUEST făcut de aplicația noastră, vom observa acel atribut după ce se face login-ul.

Dacă utilizatorul va face un refresh al browser-ului, atunci aplicația va pierde datele de autentificare stocate în serviciile Angular și utilizatorul va fi redirectat către pagina de login.

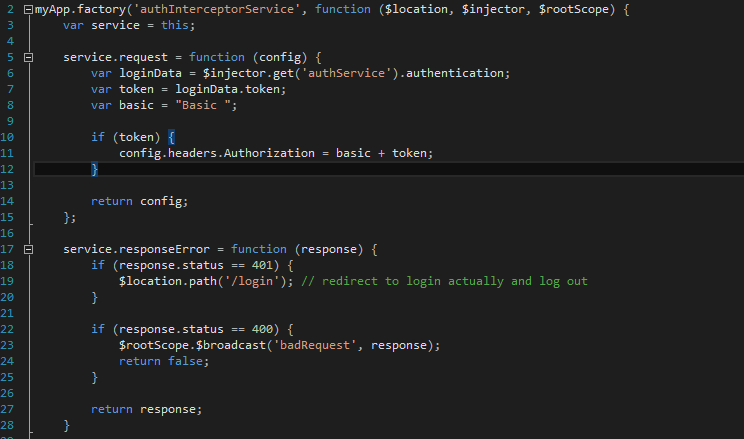


Figura 39. Auth interceptor service

În figura 39 observăm implementarea concretă a serviciului de intercepție a mesajelor Http. Practic acest serviciu se interpune între server și request-ul XHR și fie inserează date în header, fie redirectează utilizatorul în cazul în care server-ul nu mai recunoaște datele de autentificare.

Pe parte de template-ing Angular-ul stă extraordinar. Putem observa câteva directive din librărie care ne ușurează extrordinar munca. Câteva exemple de directive Angular.

ng-app – folosită pentru a inițializa o aplicație AngularJS.

ng-init – folosită pentru inițializarea datelor pe un input.

ng-model – folosită pentru a bind-ui o valoare pe un control HTML (input, select, textarea).

ng-repeat – folosită pentru a itera peste un array și pentru a controla view-ul fiecărui element.

ș.a.md.

Pe partea de view, angular se remarcă prin sintaxa sa specifică. Sintaxa de one way bind care este reprezentată de {{model}}. Acel “model” este defapt o variabilă cu care $scope-ul este decorat. Ce este $scope? Este un serviciu AngularJS prin care view-ul va fi notificat de orice schimbare a valorilor cu care este decorat. Spre exemplu dacă într-un controller injectăm serviciu $scope, valorile cu care vom decora $scope-ul dacă vor fi binduite pe UI ele vor apărea. Orice schimbare se va vedea pe UI.

Pe partea de view (HTML) nu trebuie referențiate ca fiind $scope.view.name, ci doar prin simpla scriere a acoladelor duble. În figura 40 și 41 vom observa diferența dintre View și Controller. Partea de two way binding se realizează prin intermediul directivei ng-model.

Ng-model este directiva Angular care reflectă atât în View cât și în controller modificările.

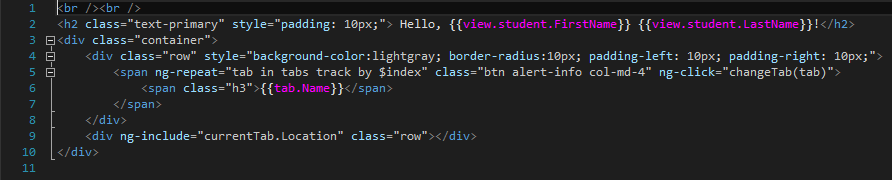


Figura 40. View HTML-Angular

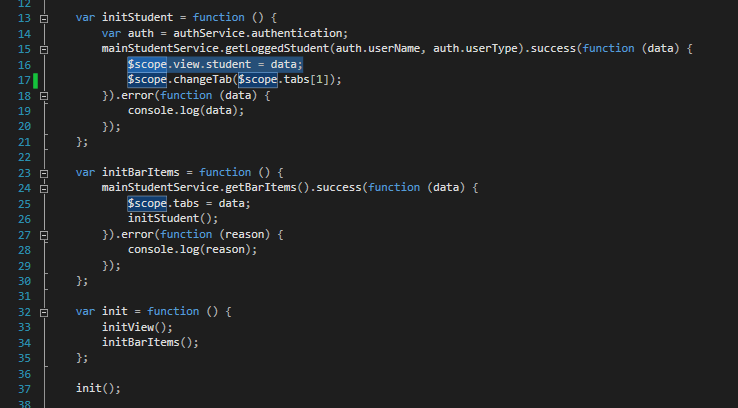


Figura 41. Controller Angular

Se poate observa în partea de controller este highlighted partea în care setez pe serviciul $scope variabila student. Variabila student nu este decorată direct pe $scope ci pe un obiect intermediar pe care setăm toate datele legate de view. Observăm în partea de controller serviciile create custom pentru lucrul cu server-ul. Cum observăm serviciile custom? Serviciile care au funcții Javascript ce întorc promise-uri. Aceste promise-uri sunt Angular specifice și au callback functions. Cele 2 callback functions folosite de mine sunt success și error. Observăm că în controller chemăm o funcție inline pentru succesul request-ului “GetBarItems”.

Pentru a putea discuta în continuare trebuie să știm ce face serviciul $http din AngularJS. Ce este $http ? este un serviciu din nucleul AngularJS care facilitează comunicarea cu serverele HTTP prin intermediul request-urilor XHR(XMLHttpRequest) sau JSONP(Javascript Object Notation with Padding). Noi în cadrul proiectului am folosit partea de serializare JSON.

API-ul $http este bazat pe promise-uri expuse de către alt serviciu din nucleul Angular și anume $q(Kriskowal Q). Un exemplu simplu de request ar fi.

*$http( {*

*method: ‘GET’,*

*url: ‘/url’*

*}).then(function successCallback(response) {*

*// cod*

*}, function errorCallback(response) {*

*// tratare eroare.*

*});*

Un exemplu concret din aplicație vom observa în figura 42.



Figura 42. Course Service HTTP ANGULAR JS

În cadrul serviciului (coursesService) observăm creerea unei fabrici care are 3 funcții care ne vor returna un promise $http prin care se execută request-ul către server pe baza parametrilor trimiși.

Observăm diferența dintre REQUEST-uri. Pentru GET-uri vom pune toți parametrii în URL, iar pentru POST/DELETE/PUT se vor pune în Http Body Payload prin intermediul proprietății data.

## Soluție Tehnică pe partea de păstrare a datelor

Pentru partea de stocare a datelor (Data warehousing) am folosit un server SQL de la Microsoft. Acest server folosește limbajul T-SQL, care este un dialect de SQL scris de Microsoft pentru lucrul cu bazele de date relaționale.

Datele aplicației sunt stocate într-o bază de date relațională cu numele School. Pentru partea tehnică a acestei secțiuni remarcăm faptul că tabelele sunt în forma normală 3.

În figura 43 vom observa relațiile între entități cât și faptul că baza de date a fost construită bine din punct de vedere al reducerii redundanțelor. Tot din diagramă realizăm și relațiile și constrângerile dintre entități.

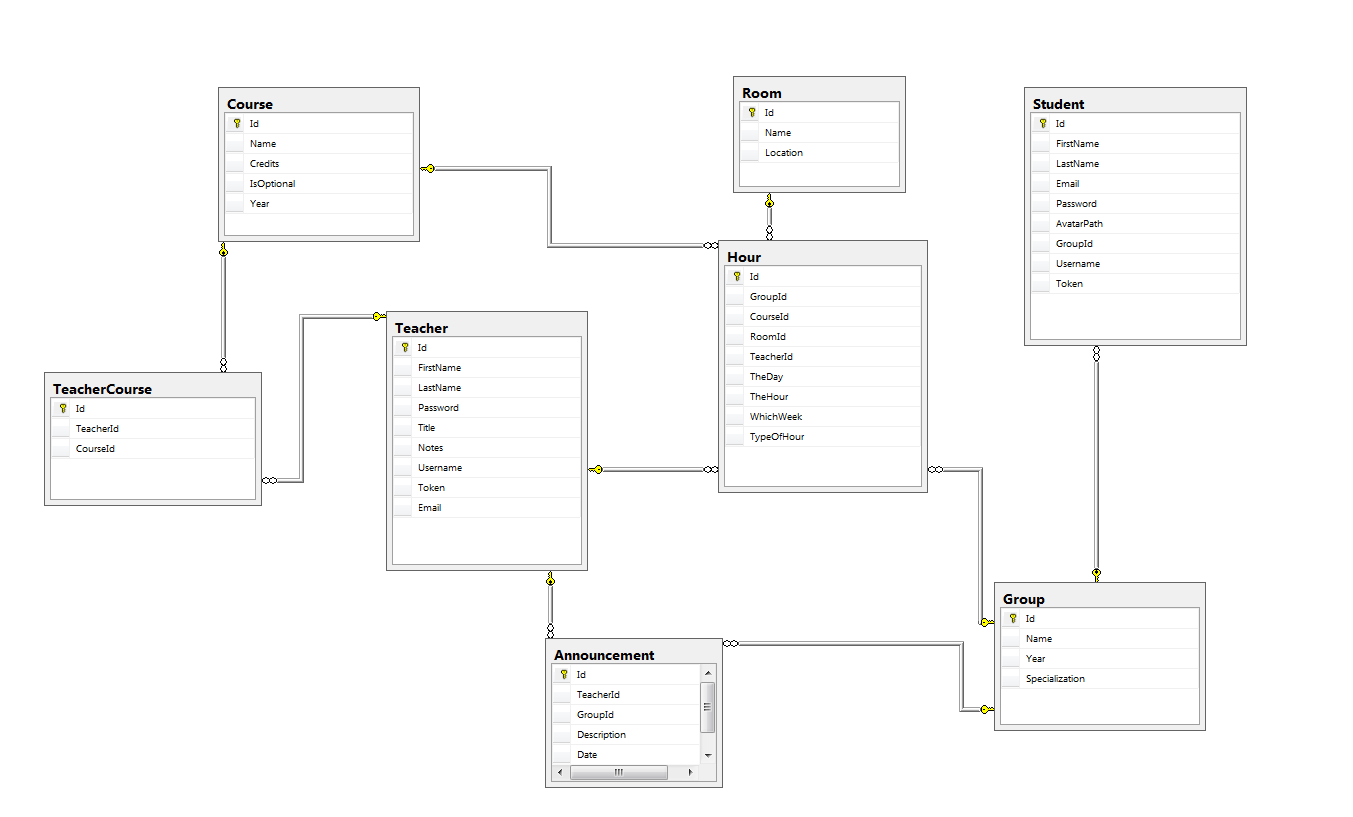


Figura 43. Diagramă bază de date – School

Din diagramă putem observa și faptul că fiecare entitate are un ID. Acel ID este și cheie primară și identitate pentru tabela în cauză. Observăm și relațiile catre au fiecare regulă de cascadare a ștergerii cât și a update-ului. Așadar când avem o relație de tip 1 – N și părintele este șters, atunci server-ul de baze de date va lansa un trigger și va șterge toți copii săi. Un alt lucru foarte important ce privește aspectul tehnic al acestui capitol este faptul că am pomenit de ideea de identitate. Ce este această identitate? Este un atribut prin care identificăm oricând o tuplă. Știind ID-ul și tabela din care dorim să luăm informația vom avea întotdeauna acces la aceasta. Identitatea asigură și unicitatea prin incrementare.

Pentru ca un atribut să fie identitate trebuie ca în momentul creării tabelei trebuie să specificăm că este identitate prin setarea flag-ului IsIdentity pe true.

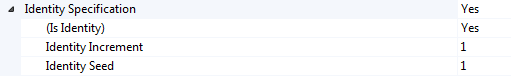


Figura 44. Identitate BD

# Manual de utilizare a aplicației

Care a fost problema ce a declanșat implementarea acestei soluții. Problema a fost dorința informatizării anunțurilor și a orarului. De cele mai multe ori, un elev dorește să-și vadă doar orele sale. În trecut exista un raport excel făcut pentru ilustrarea orelor. Acest lucru era un overhead în întocmirea orarului lucru ce m-a motivat să creez o platformă care să se dedice fie studentului, fie profesorului.

Din punct de vedere al utilității aplicației pot spune că aplicația este o unealtă de organizare a orelor foarte versatilă. Softul reușește să îmbine organizarea orelor studenților cât și a profesorilor. Aplicația are un portal pentru administrare.

Aplicația este structurată astfel încât să mențină 3 platforme. Prima platformă este dedicată administrării ei. Cea de-a doua platformă este dedicată profesorului. Ultima dar nu cea din urmă este platforma dedicată elevului.

Platforma administratorului este construită ca un CMS(Content Management System). Ea înglobează toate entitățile care pot apărea în cadrul gestionării unui orar. Entitățile prezente în aplicația noastră sunt: studenții, profesorii, sălile, grupele, materiile și orarul în sine. Până să intrăm în detaliile aplicației va trebui să discutăm despre problema autentificării. Aplicația se lansează prin ecranul ilustrat în figura 45. După cum observăm în infobox-ul gri ni se prezintă metodele de autentificare. Fie cea de elev, fie cea de profesor.

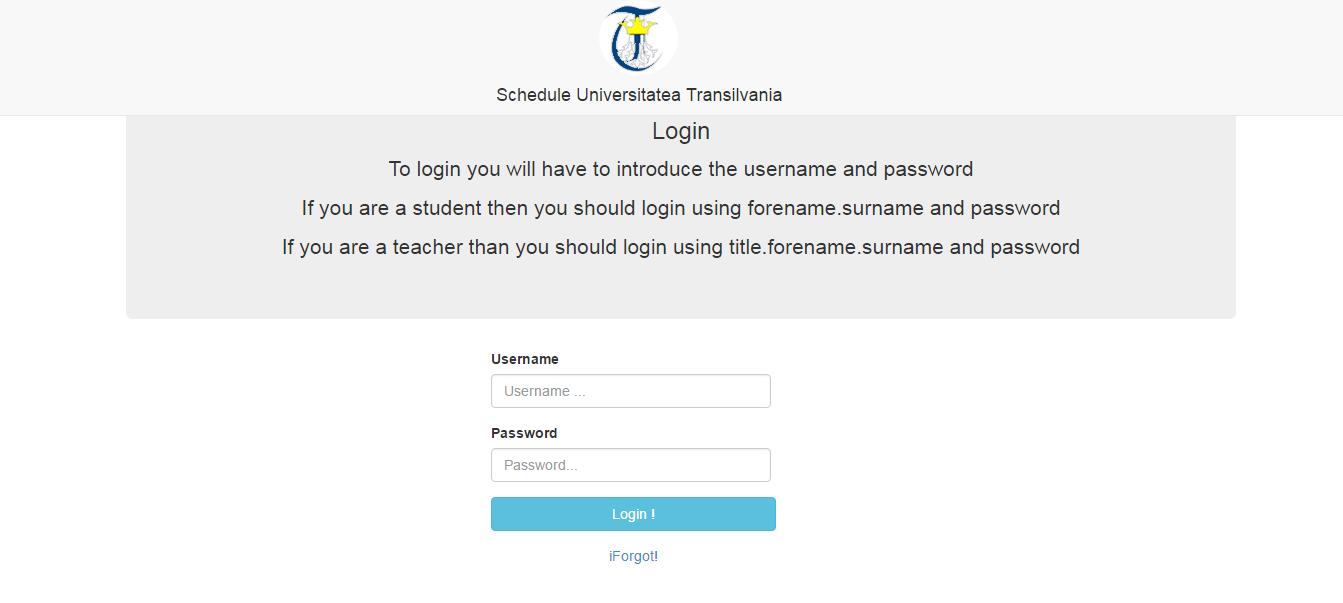


Figura 45. Login Page

După autentificarea corectă aplicația poate duce în 3 locuri diferite. Primul loc ar fi partea de administrare a conținutului. Această platformă de administrare este dedicată unei persoane desemnate ca și administrator. Acest administrator are alt sistem de autentificare. Dacă autentificarea se face pe baza titlului și a numelui atunci platforma va căuta în tabela profesorilor și dacă parola este corect furnizată atunci aplicația ne va redirecta către platforma profesorului. Bineînțeles dacă utilizatorului se autentifică doar cu numele, atunci aplicația va căuta un elev și dacă datele de autentificare sunt corecte atunci platforma elevului va fi afișată.

## Ghid de utilizare – Administrator

Despre partea de administrare a conținutul avem prezentate funcționalitățile în figura 46. Pentru fiecare entitate s-a creat un tab special în care se pot adăuga, modifica sau șterge entități din acel tip.

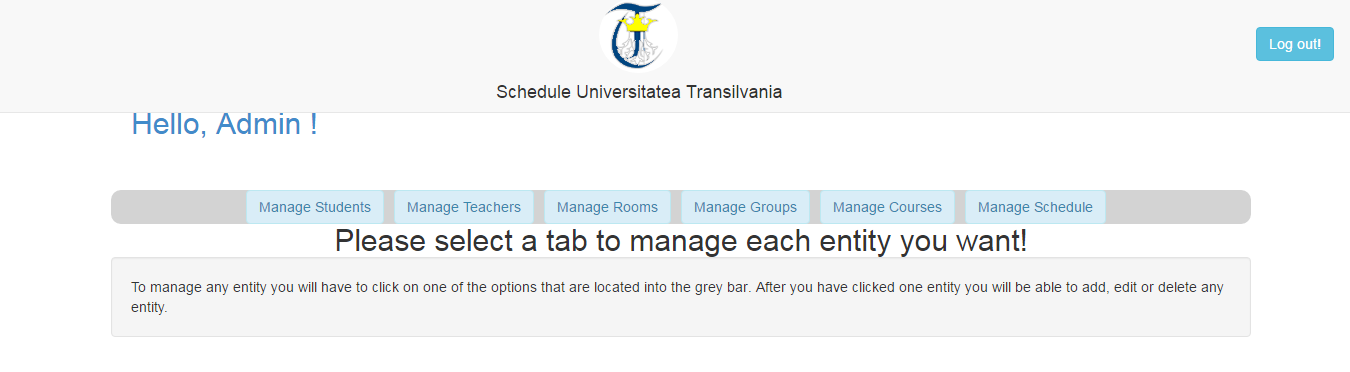


Figura 46. Prima pagină administrator

Tot în figura 46 observăm un infobox prin care suntem informați despre posibilitățile pe care le avem dacă suntem autentificați ca și administrator.

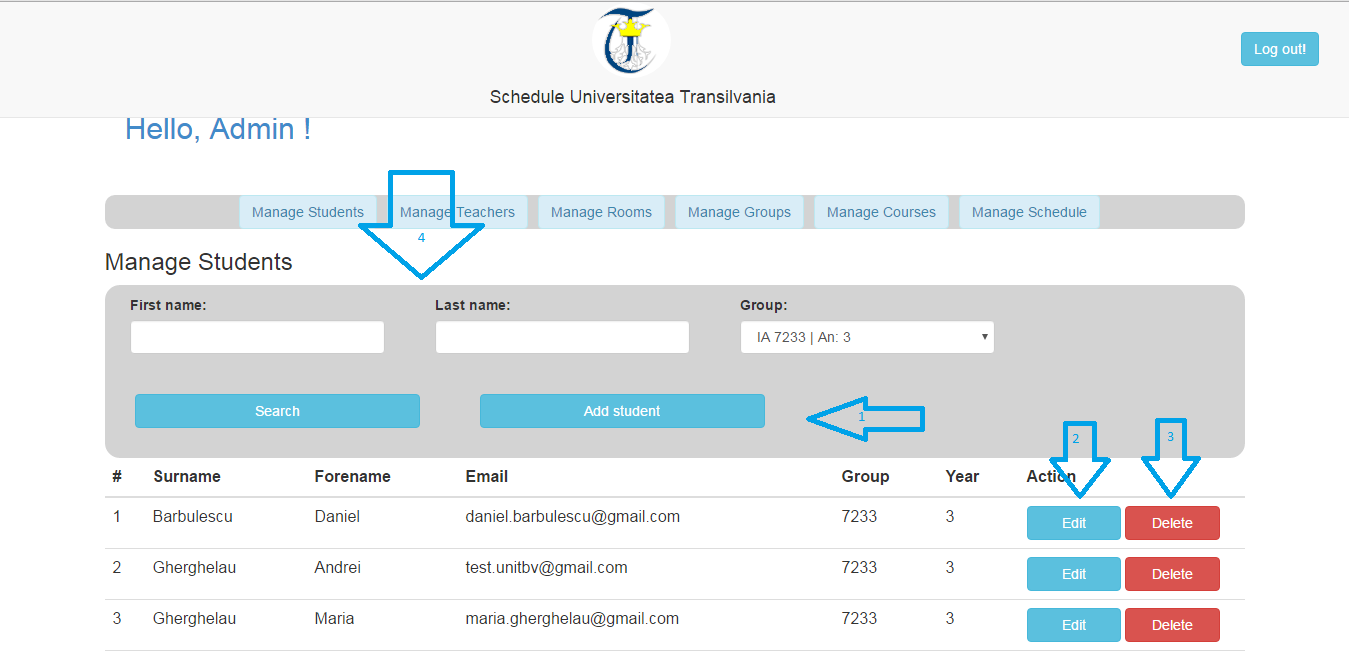


Figura 47. Model administrare entitate – Student

În figura 47 observăm tab-ul de studenți. Aici avem posibilitatea să căutăm un elev prin intermediul criteriilor prezente în spațiul marcat cu gri (4).

O altă funcționalitate ar putea fi să adăugăm un student. Pentru a adăuga un student trebuie ca administratorul să dea click pe butonul marcat cu (1) (Add student). După acel click aplicația va lansa o fereastră prin care putem completa câmpurile în vederea adăugării unui student nou. Tot aceeași fereastră se va deschide în cazul în care dorim să modificăm datele unui student. Pentru modificarea datelor unui student, administratorul va trebui să dea click pe butonul marcat cu label-ul (Edit) sau (2 – din figura 47). În figura 48 se poate observa cum arată fereastra de modificare sau adăugare a unui student.

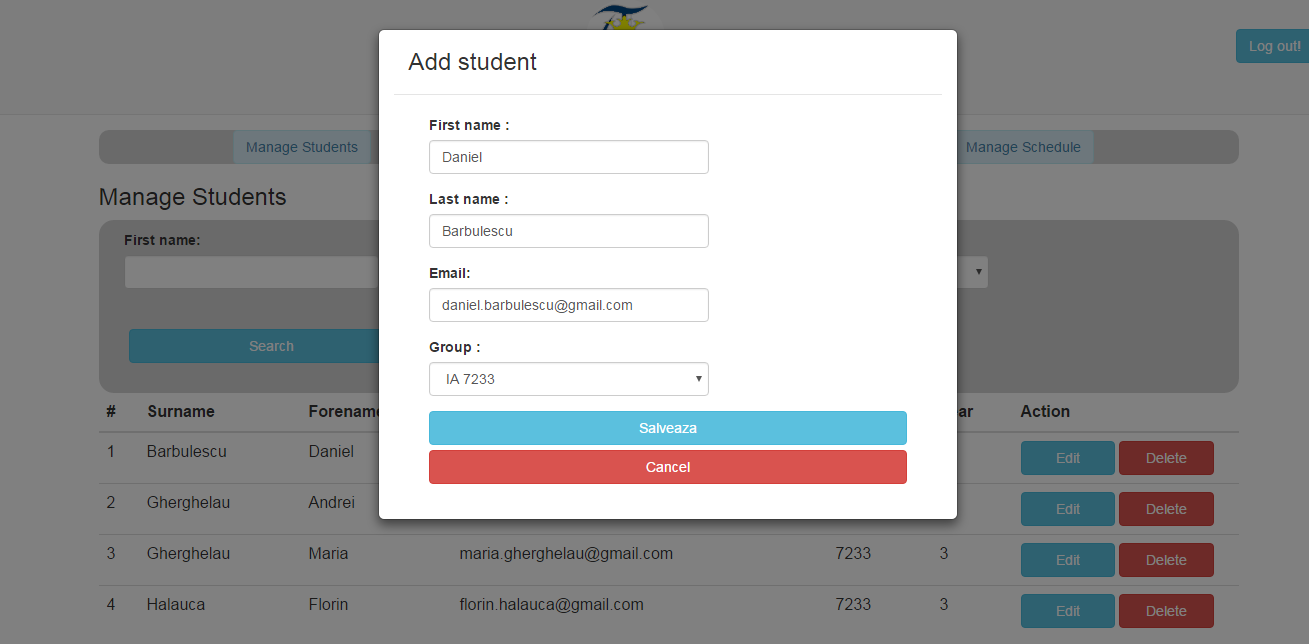


Figura 48. Model Adăugare și Editare entitate

Dacă administratorul dorește să șteargă un student din aplicație, atunci acesta va trebui să apese pe butonul Delete marcat cu (3) în figura 47. După apăsarea butonului respectiv, o fereastră de tip modal va apărea exact ca în figura 49.

Această fereastră are rolul de a recontrola decizia pe care o dorim. Dacă administratorul este ferm convins că dorește să șteargă un student, atunci el va trebui să apese butonul Delete astfel aplicația va șterge permanent acel cont.

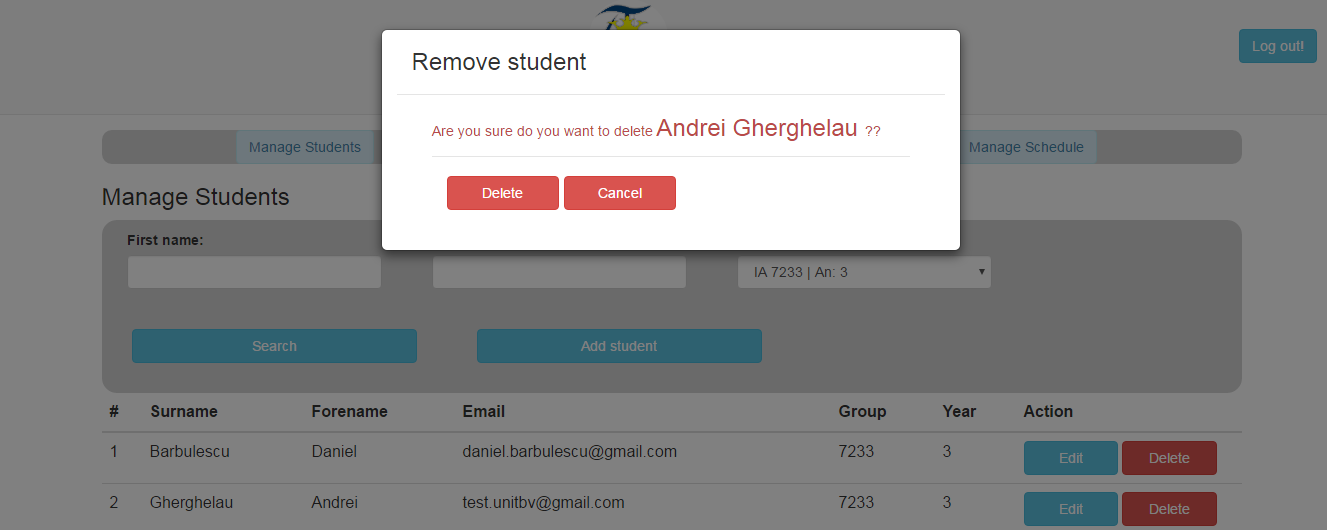


Figura 49. Model ștergere entitate – fereastră

Dacă utilizatorul (administratorul) realizează că a fost o greșeală și nu dorește ștergerea studentului, atunci acesta poate abandona decizia prin apăsarea butonului Cancel.

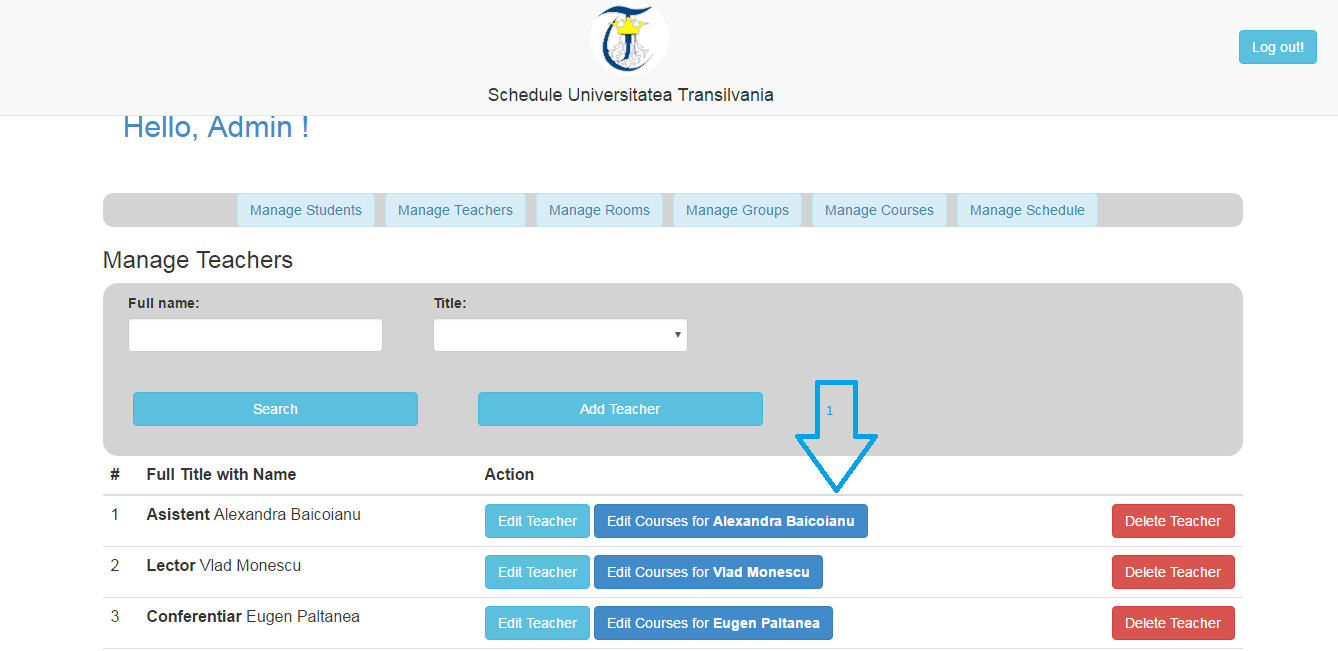
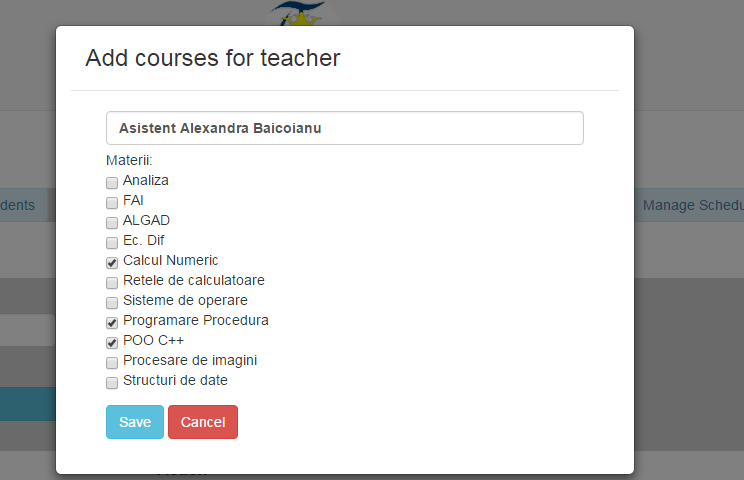


Figura 50. Administrare profesori

Pentru partea de administrare a profesorului putem observa aceleași funcționalități ca pentru studenți, la care s-a mai adăugat încă o funcționalitate și anume adăugarea materiilor predate.

Această funcționalitate poate fi exercitată dacă administratorul va da click pe butonul de editare materii din dreptul unui profesor. Odată apăsat acest buton, aplicația va lansa o fereastră de tip modal prin care utilizatorul va putea alege ce materii poate preda profesorul selectat. Această fereastră are o multitudine de checkbox-uri prin care putem selecta ce poate preda profesorul în cauză. Ilustrarea din aplicație este în figura următoare.



Toate celelalte funcționalități și anume administrarea de grupe, săli, materii au exact aceleași comportament. Prin comportament ne referim la faptul că se pot adăuga, șterge și modifica aceste entități. Bineînțeles pe lângă operațiile CRUD, putem căuta entități specifice prin criteriile din search bar.

Singura entitate diferită din punct de vedere al administrării este administrarea orarului. Acesată entitate se va administra pe grupă. Se va selecta o grupă din multitudinea de grupe prezente. În figura 51 se pot observa grupele ilustrate prin dreptunghiurile albastre. După selectarea unei grupe anume, aplicația va cere serverului orarul pentru grupa aceea.

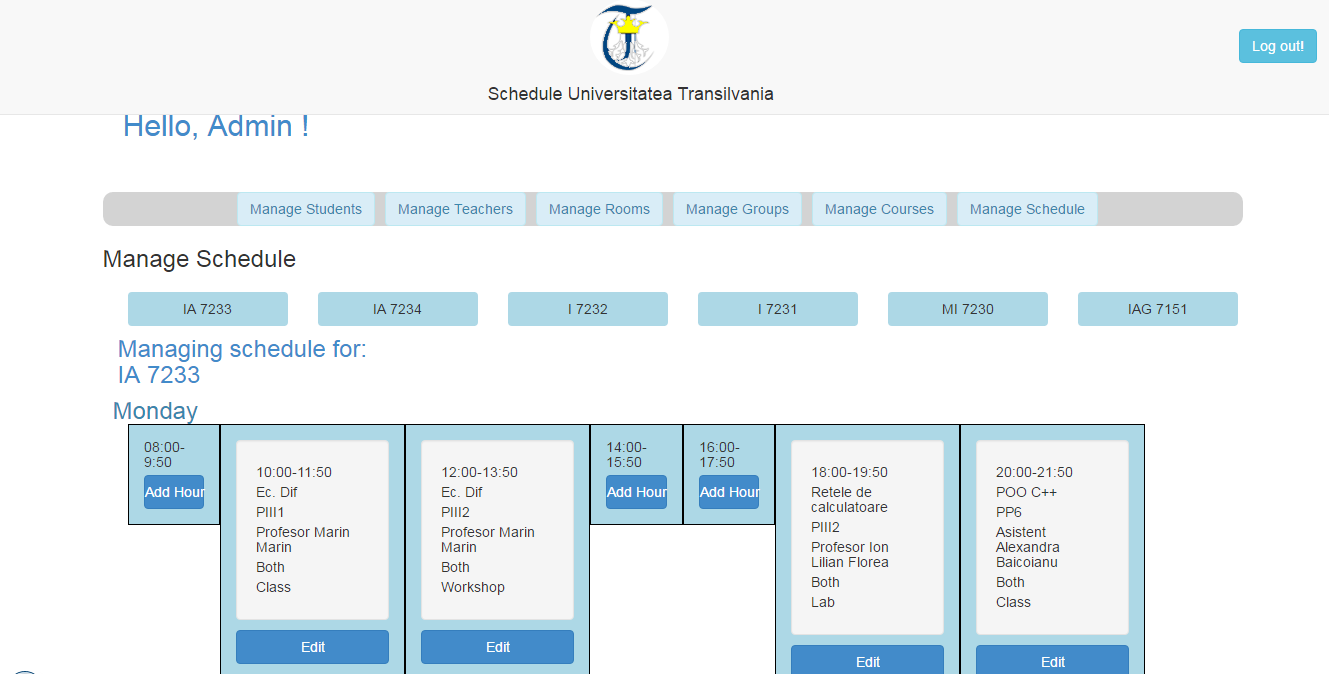
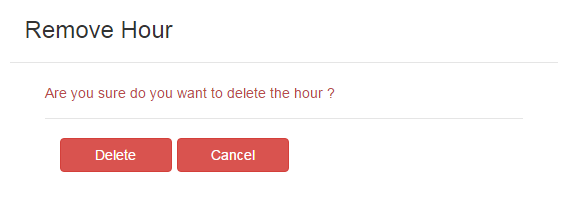


Figura 51. Administrare Orar

Putem observa în figura 51 că putem adăuga o oră prin intermediul butonului (Add Hour) prezent într-un anumit interval sau putem edita sau șterge o anumită oră. Dacă se constată ca s-a terminat administrarea orelor pentru o anumită grupă, se poate schimba grupa prin selectarea din panoul de sus a unei alte grupe. Dacă se dorește ștergerea unei anumite ore administratorul trebuie să apese butonul de Delete(3) prezent în figura 52. Cum am obișnuit și în celelalte cazuri, o ștergere a unei anumite entități va atrage după sine și necesitatea reconfirmării acestei acțiuni. Putem observa această fereastră în imaginea de mai jos.



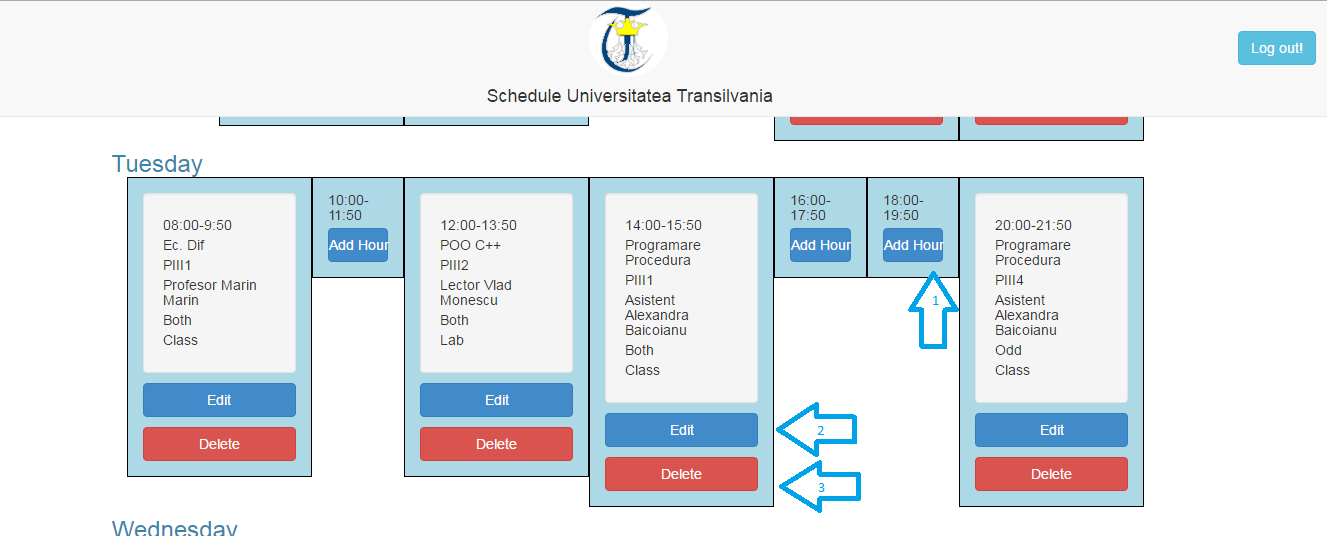


Figura 52. Funcționalități administrare orar

Pentru editarea unei anumite ore administratorul va trebui să apese butonul de edit (2) a unei ore deja create și mai apoi aplicația va lansa o fereastră de tip modal. Această fereastră este ilustrată în figura 53. Pentru adăugarea unei ore administratorul va trebui să dea click pe butonul de Add Hour și îi va apărea aceeași fereastră. Validările sunt făcute întocmai pentru a păstra integritatea unui orar și sensul său.

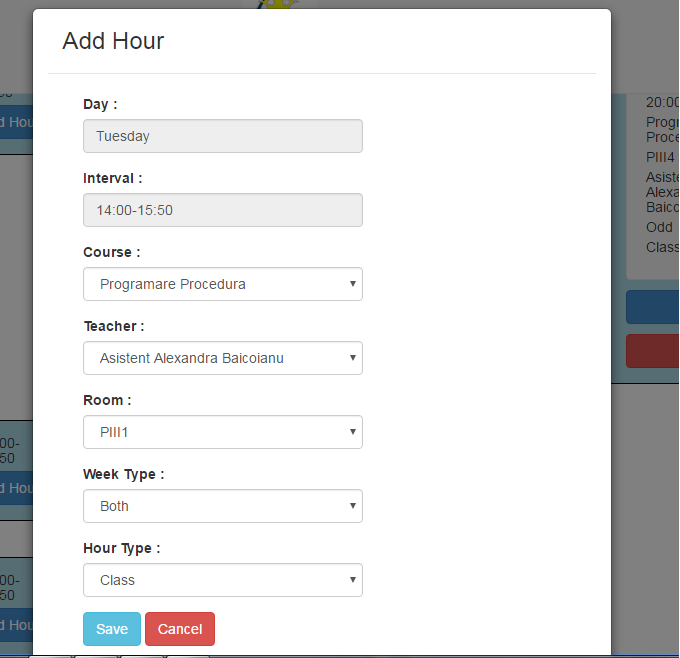


Figura 53. Adăugare sau modificare oră

## Ghid de utilizare – Profesor

Din punct de vedere al profesorului, platforma se prezintă exact ca în figura 54. Principalele use case-uri ale unui profesor ar fi să-și vizualizeze profilul, unde poate să-și schimbe datele de contact. Să vizualizeze orele la care trebuie să ajungă. Să facă sau sa vadă ultimele sale anunțuri. Toate aceste funcționalități sunt prezentate în cele ce urmează. Ele pot fi accesate pe baza tab-urilor prezentate în figura 54.

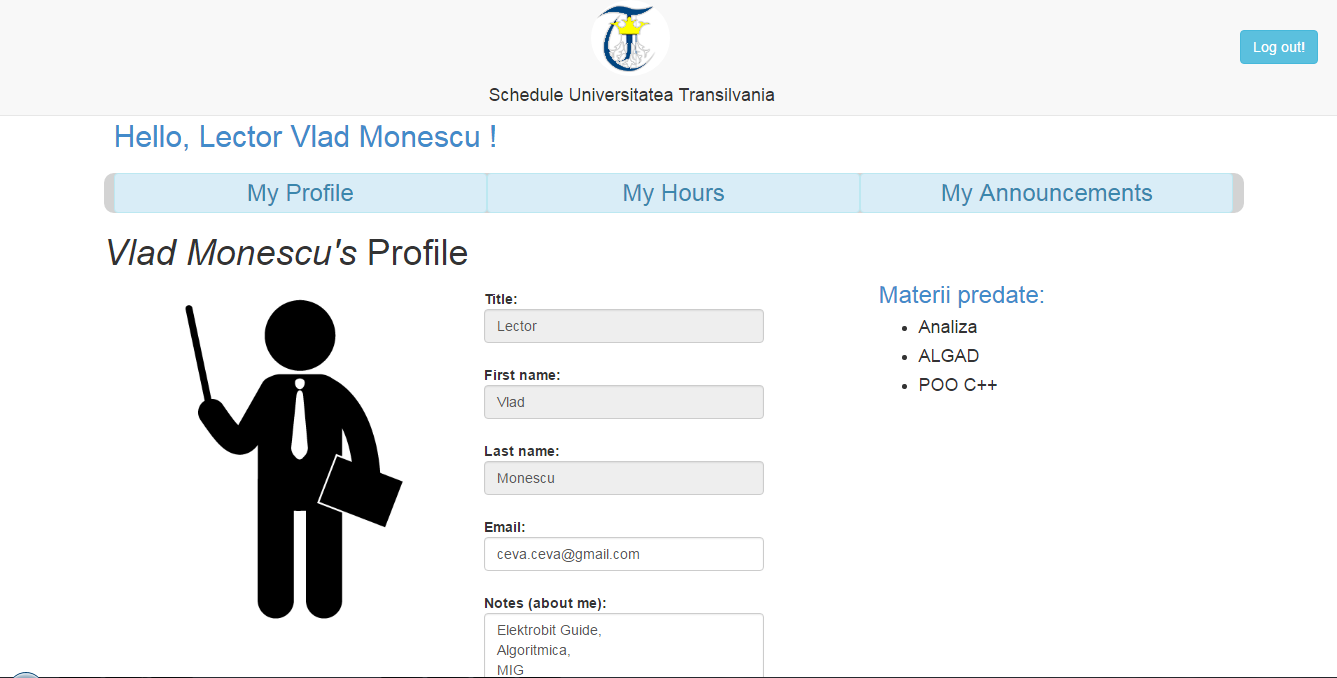


Figura 54. Pagină profil – Profesor

În momentul autentificării corect a unui profesor, aplicația va prezenta tab-ul de ore ale profesorului autentificat. În figura 55 putem observa un exemplu de orar pentru profesor.

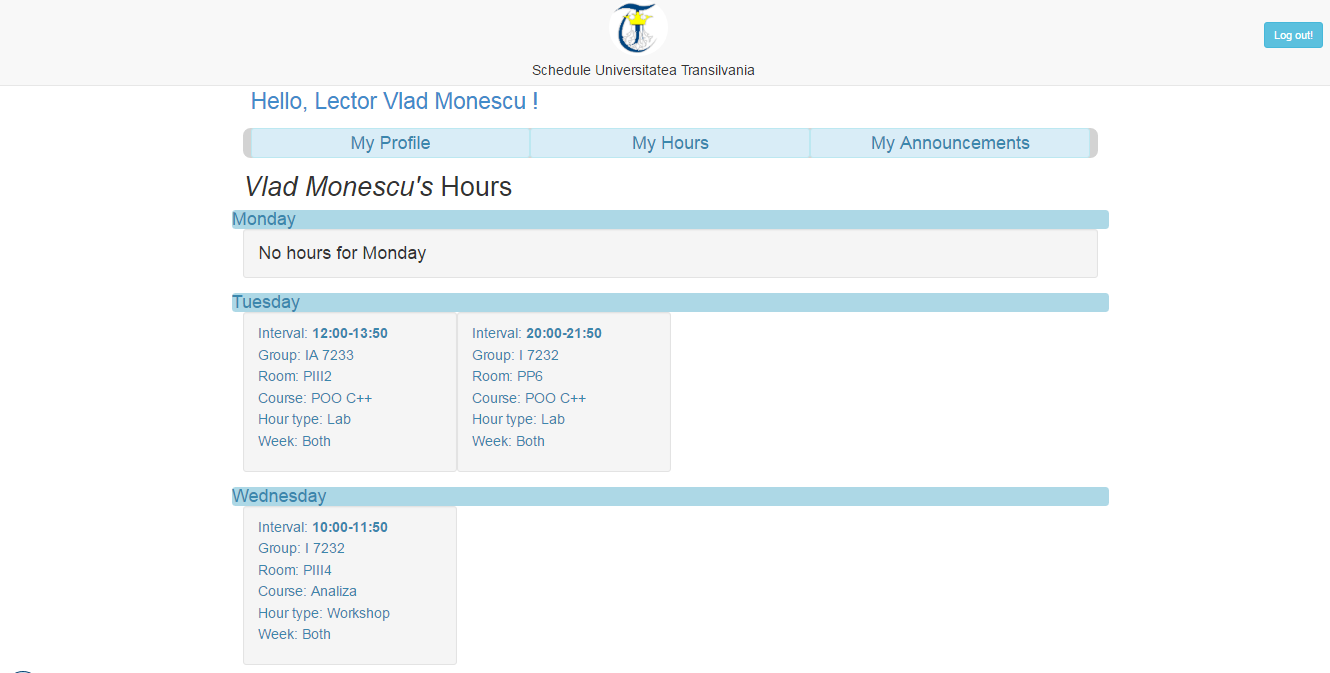


Figura 55. Pagină ore - Profesor (pagină de pornire)

Cea de-a treia funcționalitate pe care o putem observa în platforma profesorului este cea de a crea un anunț pentru o grupă. Această funcție o poate exercita dacă profesorul apasă pe butonul de Add Announcement. Din această apăsare aplicația va lansa o fereastră pe care profesorul trebuie să o completeze și să marcheze o grupă ca și țintă. La apăsarea butonului de salvare aplicația va trimite mail-uri către elevii acelei grupe.

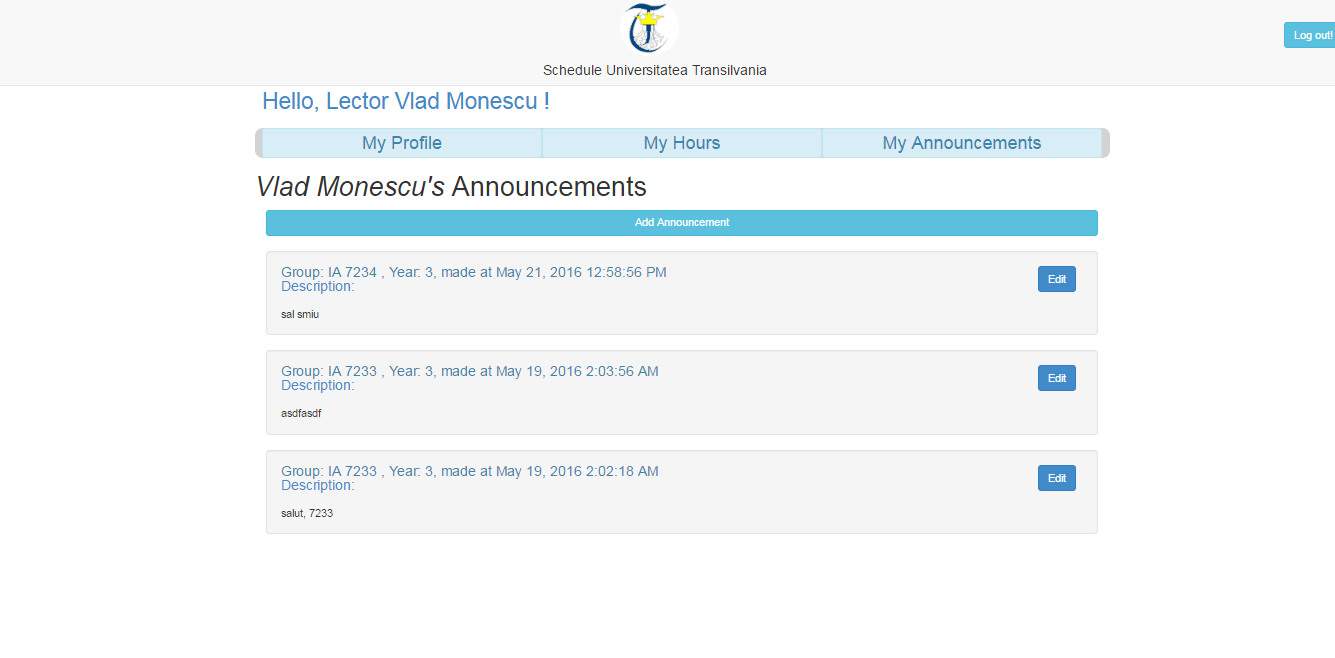


Figura 56. Pagină anunțuri – Profesor

Bineînțeles că profesorul poate edita un anunț dar acesta nu va mai retrimite o notificare către toți studenții. Pe tabul de anunțuri vor fi actualizate întotdeuna ultimele 10 anunțuri ale profesorului.

## Ghid de utilizare – Elev

Prin primsa unui elev platforma se prezintă asemănător cu cea a profesorului numai că aici elevul nu are posibilitatea de a modifica nimic. Elevul este doar consumator de orar, cât și consumator de anunțuri. Elevul poate fi notificat și atât. Singura funcționalitate de modificare este prezentă în figura 58 unde elevul își poate modifica datele de contact.

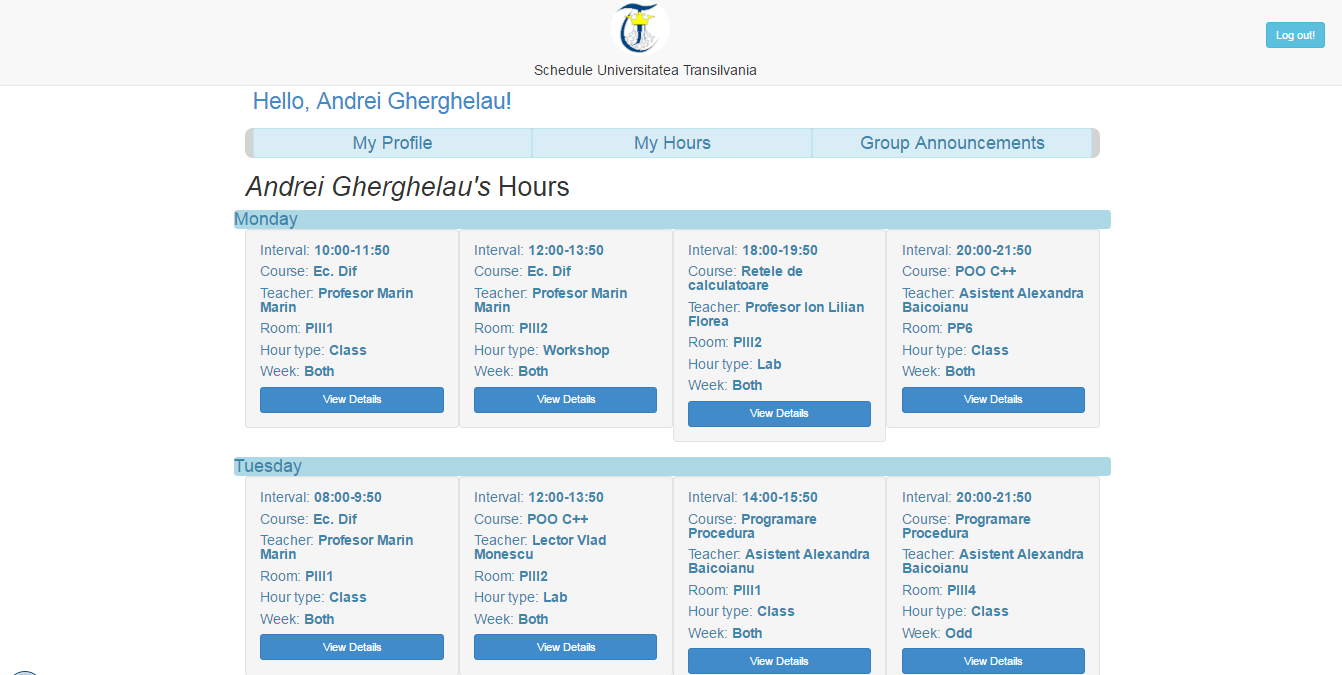


Figura 57. Pagină ore - elev (pagină pornire)

În figura 58 putem observa pagina de profil a unui elev, unde acesta își poate modifica email-ul de contact.

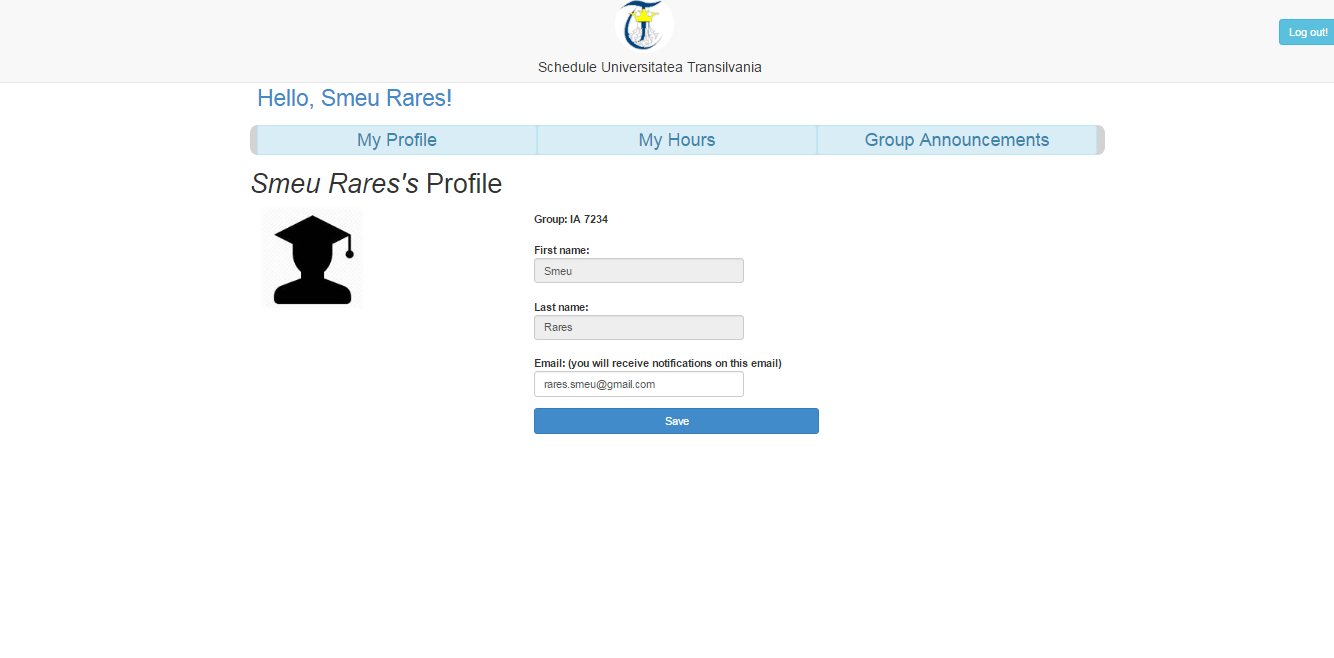


Figura 58. Pagină profile – Elev

Tot în cadrul platformei elevul va putea vizualiza ultimele anunțuri în ordine cronologică ale grupei din care face parte. Aici va putea observa cine a pus anunțul cât și data emiterii acestuia. Anunțul după sine atrage și un email pe care fiecare elev îl va primi. În figura 60 vom putea observa un exemplu de mail.



Figura 59. Pagină anunțuri – Elev

În figura 59 avem prezentată fereastra de anunțuri. În această secțiune vom vedea ultimele anunțuri pe care le-au făcut profesorii. Aceste anunțuri sunt legate de grupa în care se află studentul.

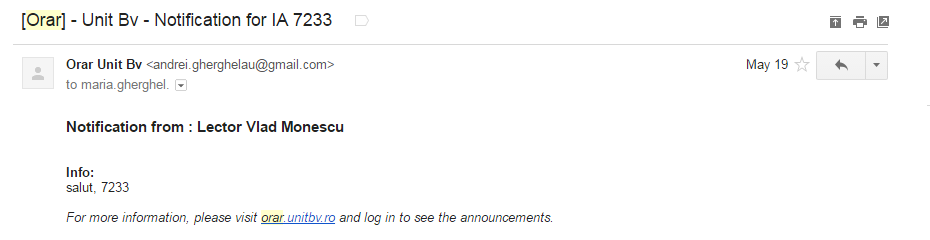


Figura 60. Exemplu mail - Elev

În figura 60 avem un email pe care îl primim de la platformă în momentul în care un profesor va face un anunț. Mai jos vom avea un link care ne va conduce către site-ul nostru.

# Ghidul de instalare a aplicației

Instalarea acestei aplicații se va face într-o succesiune de 5 pași. Până să putem instala aplicația primul lucru care trebuie făcut este să luăm soluția server-ului de pe **Git.** <https://github.com/gherghe94/Licenta2>

**Pasul 1**:

Se instalează SQL Server 2012. Dacă softul este deja instalat atunci se trece la **pasul 1.1.**

**Pasul 1.1** Se va face back-up la baza de date prin funcționalitatea de restore Database a Sql server-ului.

**Pasul 1.2** Pentru restaurare se va folosi fișierul bak din folder-ul SQLBackups

**Pasul 1.3** Se va crea un nou Login pentru procesul de IIS. Acest pas se va face prin intrarea în secțiunea Security din SQL server și apoi se va adăuga un nou Login cu numele „IIS APPPOOL\school”. School fiind numele application pool-ului din IIS

**Pasul 2**:

Se va instala IIS din programe și funcționalități Windows.

**Pasul 2.1** Se vor activa toate funcționalitățile IIS.

**Pasul 2.2** Se va merge în folderul framework-ului de .NET în ultima versiune și acolo se va deschide o fereastră cmd unde se va rula următorul script. aspnet\_regiis –i.

**Pasul 2.3** După instalarea IIS. Se va verifica application pool-ul astfel încât el să ruleze pe versiunea 4.x.

**Pasul 2.4** Se creează web site-ul și se va pune Physical Path-ul în folder-ul Licenta.

**Pasul 2.5** Se creează un application pool cu numele school, se va înlocui pe website-ul nostru application pool-ul default și numele va fi school.

**Pasul 3**:

Conectarea la baza de date.

**Pasul 3.1** Se vor verifica stringurile de conexiune aflate în Web.config astfel încât să pointeze către baza noastră restaurată.

**Pasul 4:**

Se va face un rebuild la soluție și se vor downloada ultimele librării dacă este cazul.

**Pasul 4.1** Se va verifica apoi dacă aplicația rulează cu succes prin deschiderea unui browser la urmatorul link:

<http://localhost:9000/web/templates/index.html#/login>

**Pasul 5**:

Se va încerca autentificarea cu contul de admin. Dacă totul a decurs bine atunci aplicația s-a instalat cu succes.

Index de figuri

[Figura 1. C# Flow 10](#_Toc454712346)

[Figura 2. Web API 2 Client Request 10](#_Toc454712347)

[Figura 3. Arhitectura MVC 12](#_Toc454712348)

[Figura 4. Ciclu XHR 13](#_Toc454712349)

[Figura 5. Object explorer & Data 15](#_Toc454712350)

[Figura 6. Date din tabela 15](#_Toc454712351)

[Figura 7. Back-up baza de date 16](#_Toc454712352)

[Figura 8. Fereastra de selectare fisier BAK 17](#_Toc454712353)

[Figura 9. Securitate, Logins 18](#_Toc454712354)

[Figura 10. Observer UML 20](#_Toc454712355)

[Figura 11. MVC vs MVVM 21](#_Toc454712356)

[Figura 12. SPA vs MPA 22](#_Toc454712357)

[Figura 13. Angular Routing System 23](#_Toc454712358)

[Figura 14. Sistem Column Grid Bootstrap CSS 24](#_Toc454712359)

[Figura 15. IIS pagina pornire 26](#_Toc454712360)

[Figura 16. Postman request 27](#_Toc454712361)

[Figura 17. Tortoise SVN funcţionalităţi de bază 35](#_Toc454712362)

[Figura 18. Diff Tool Tortoise SVN 36](#_Toc454712363)

[Figura 19. Git check-in system 38](#_Toc454712364)

[Figura 20. Check-in in time 38](#_Toc454712365)

[Figura 21. GitHub aplicație 39](#_Toc454712366)

[Figura 22. Git Diff 40](#_Toc454712367)

[Figura 23. Web Api Routing 42](#_Toc454712368)

[Figura 24. Save (POST) 43](#_Toc454712369)

[Figura 25. Basic Authorization 44](#_Toc454712370)

[Figura 26. Implementare filtru basic authentication 44](#_Toc454712371)

[Figura 27. Teacher implementare IAccessGranter 45](#_Toc454712372)

[Figura 28. Request Flow 46](#_Toc454712373)

[Figura 29. Abstract Service UML 47](#_Toc454712374)

[Figura 30. Service class 48](#_Toc454712375)

[Figura 31. Save Generic 48](#_Toc454712376)

[Figura 32. Repository Pattern UML 49](#_Toc454712377)

[Figura 33. POCO Course - Materia ca obiect 49](#_Toc454712378)

[Figura 34. Tabela Course 50](#_Toc454712379)

[Figura 35. String de conexiune 50](#_Toc454712380)

[Figura 36. Folosire NPOCO 50](#_Toc454712381)

[Figura 37. Routing 51](#_Toc454712382)

[Figura 38. Rute Angular 52](#_Toc454712383)

[Figura 39. Auth interceptor service 53](#_Toc454712384)

[Figura 40. View HTML-Angular 54](#_Toc454712385)

[Figura 41. Controller Angular 54](#_Toc454712386)

[Figura 42. Course Service HTTP ANGULAR JS 55](#_Toc454712387)

[Figura 43. Diagramă bază de date – School 56](#_Toc454712388)

[Figura 44. Identitate BD 56](#_Toc454712389)

[Figura 45. Login Page 57](#_Toc454712390)

[Figura 46. Prima pagină administrator 58](#_Toc454712391)

[Figura 47. Model administrare entitate – Student 58](#_Toc454712392)

[Figura 48. Model Adăugare și Editare entitate 59](#_Toc454712393)

[Figura 49. Model ștergere entitate – fereastră 59](#_Toc454712394)

[Figura 50. Administrare profesori 60](#_Toc454712395)

[Figura 51. Administrare Orar 61](#_Toc454712396)

[Figura 52. Funcționalități administrare orar 62](#_Toc454712397)

[Figura 53. Adăugare sau modificare oră 62](#_Toc454712398)

[Figura 54. Pagină profil – Profesor 63](#_Toc454712399)

[Figura 55. Pagină ore - Profesor (pagină de pornire) 63](#_Toc454712400)

[Figura 56. Pagină anunțuri – Profesor 64](#_Toc454712401)

[Figura 57. Pagină ore - elev (pagină pornire) 65](#_Toc454712402)

[Figura 58. Pagină profile – Elev 65](#_Toc454712403)

[Figura 59. Pagină anunțuri – Elev 66](#_Toc454712404)

[Figura 60. Exemplu mail - Elev 66](#_Toc454712405)

BIBLIOGRAFIE & WEBOGRAFIE

1. [www.wikipedia.ro](http://www.wikipedia.ro)
2. [www.codeproject.com](http://www.codeproject.com)
3. [www.w3schools.com](http://www.w3schools.com)
4. Google IT definitions
5. Sasu, M., Lucian, ***Medii vizuale de programare***
6. Bocu, D., ***Ingineria sistemelor soft reale***, Editura Albastra, Cluj-Napoca, 2002
7. Bocu, D., ***Inițiere în modelarea obiect orientată a sistemelor soft******utilizând UML,***  Editura Albastra, Cluj-Napoca, 2006
8. Fowler, M., ***Patterns of enterprise application architecture,*** Addison Wesley, New York
9. Tugberk Ugurlu, Alexander Zeitler and Ali Kheyrollahi**,*PRO ASP.NET Web API,*** Apress, Septembrie 2013