**UNIVERSITATEA DIN PITEȘTI FACULTATEA DE ELECTRONICĂ, COMUNICAȚII ȘI CALCULATOARE DEPARTAMENTUL ELECTRONICĂ, CALCULATOARE ȘI INGINERIE ELECTRICĂ PROGRAMUL DE STUDII CALCULATOARE**

**PROIECT DE DIPLOMĂ**

**Absolvent**

**Militaru Bogdan Alexandru**

**Conducător Științific**

**Șef Lucrări dr.ing. Enescu Florentina**

**Pitești**

**Sesiunea 2017**

**UNIVERSITATEA DIN PITEȘTI FACULTATEA DE ELECTRONICĂ, COMUNICAȚII ȘI CALCULATOARE DEPARTAMENTUL ELECTRONICĂ, CALCULATOARE ȘI INGINERIE ELECTRICĂ PROGRAMUL DE STUDII CALCULATOARE**

**PROIECT DE DIPLOMĂ**

**Sistem Software pentru prelucrarea și**

**consultarea datelor de identificare ale studenților**

**Absolvent**

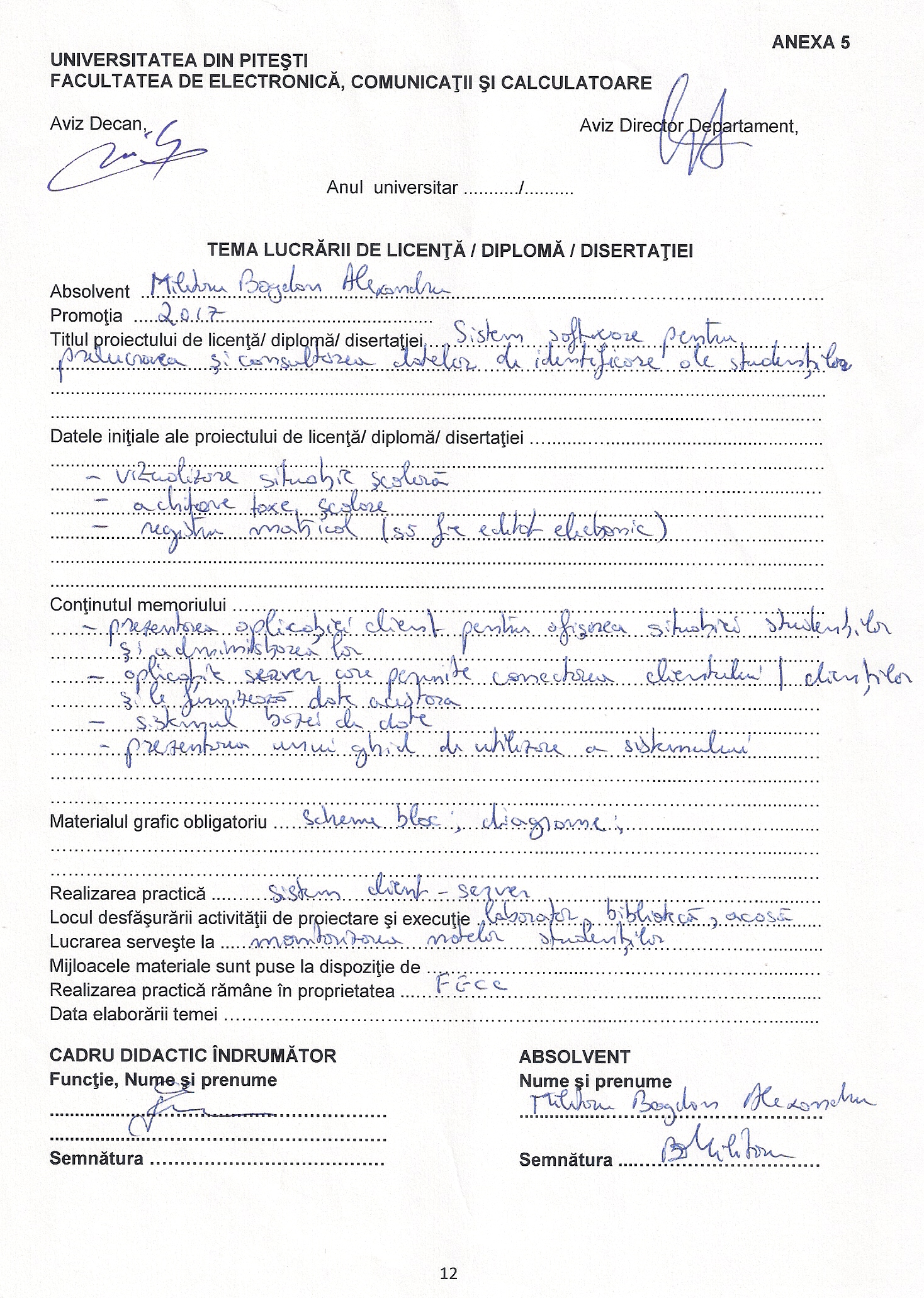
**Militaru Bogdan Alexandru**

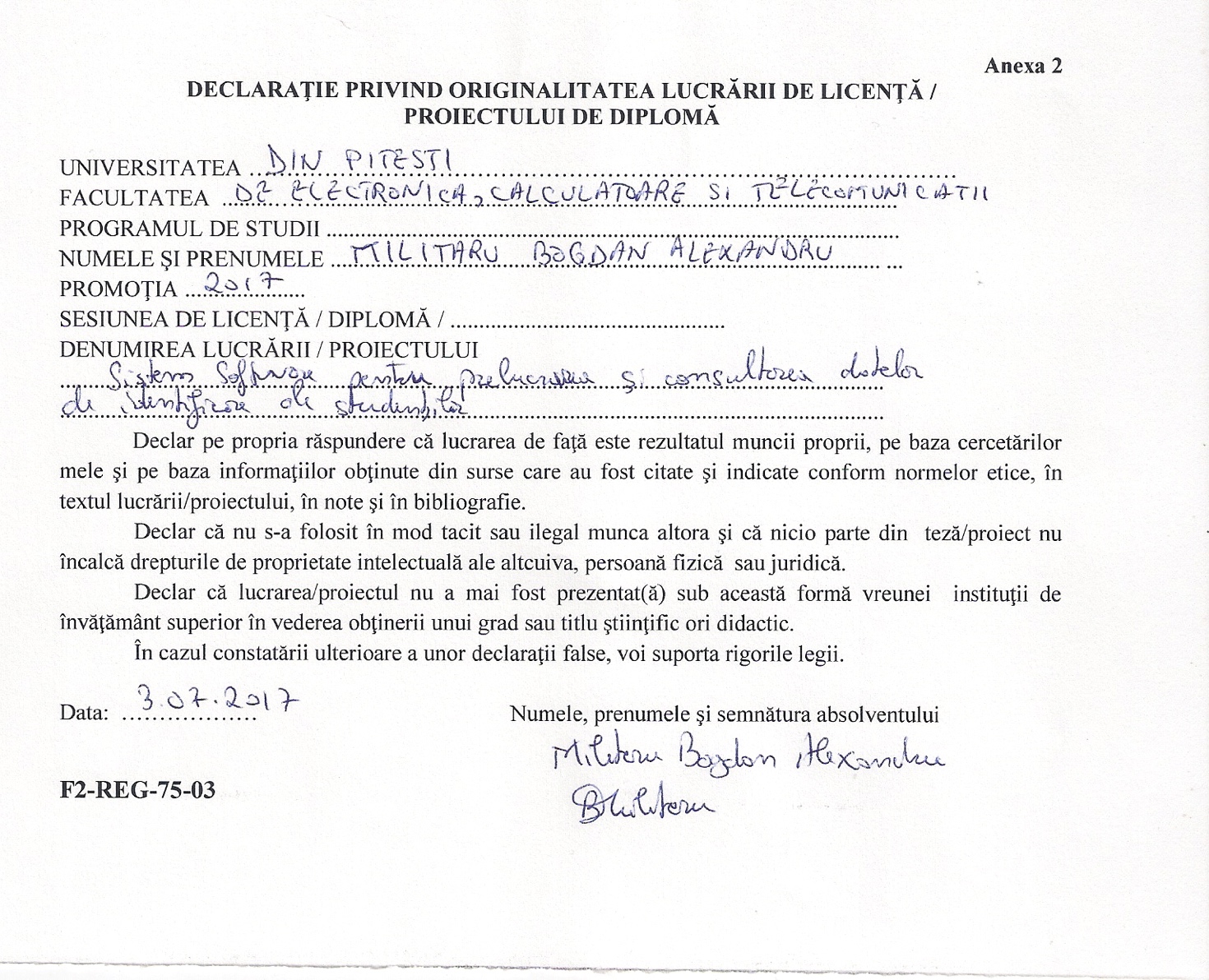
**Conducător Științific**

**Șef Lucrări dr.ing. Enescu Florentina**

**Pitești**

**Sesiunea 2017**





# Cuprins

Table of Contents

[Cuprins 12](#_Toc487008527)

[Lista Figurilor 14](#_Toc487008528)

[Lista tabelelor 16](#_Toc487008529)

[Lista acronimelor din proiect 17](#_Toc487008530)

[Introducere 10](#_Toc487008531)

[Stadiul Actual 12](#_Toc487008532)

[1. Web Service 14](#_Toc487008533)

[1.1. Introducere 14](#_Toc487008534)

[1.2. Faza de proiectare a webservice-ului 14](#_Toc487008535)

[1.3. Introducere în NodeJS 17](#_Toc487008536)

[1.4. Sistemul de gestiune al bazei de date (SGBD) 19](#_Toc487008537)

[1.5. Arhitectura 21](#_Toc487008538)

[1.6. Arhitectura bazei de date 28](#_Toc487008539)

[1.7. Securitate 29](#_Toc487008540)

[1.8. Conectarea la webservice 31](#_Toc487008541)

[1.9. Rularea webservice-ului 32](#_Toc487008542)

[1.10. Concluzii 32](#_Toc487008543)

[2. Client WEB 33](#_Toc487008544)

[2.1. Introducere 33](#_Toc487008545)

[2.2. Angular 2 - Noțiuni Teoretice 33](#_Toc487008546)

[2.3. Arhitectura 37](#_Toc487008547)

[2.4. Nucleu 39](#_Toc487008548)

[2.5. Autentificare și Securitate 41](#_Toc487008549)

[2.6. Module de extensie 45](#_Toc487008550)

[2.7. Stiluri și design 48](#_Toc487008551)

[2.8. Concluzii 50](#_Toc487008552)

[3. Interacțiunea cu factorul uman 51](#_Toc487008553)

[3.1. Introducere 51](#_Toc487008554)

[3.2. Experiența Utilizatorului 51](#_Toc487008555)

[3.3. Utilizarea aplicației 54](#_Toc487008556)

[3.3.1. Înregistrarea studentului pe site 55](#_Toc487008557)

[3.3.2. Logare Student 58](#_Toc487008558)

[3.3.3. Administrare 60](#_Toc487008559)

[3.3.4. Administrare Discipline 61](#_Toc487008560)

[3.3.5. Administrare Specializări 62](#_Toc487008561)

[3.3.6. Administrare Studenți 64](#_Toc487008562)

[3.3.7. Administrarea notelor 67](#_Toc487008563)

[3.3.8. Administrare Utilizatori 69](#_Toc487008564)

[3.4. Mod de instalare 73](#_Toc487008565)

[3.4.1. Instalare baza de date 73](#_Toc487008566)

[3.4.2. Instalare server Node JS 74](#_Toc487008567)

[3.4.3. Rulare Sistem 74](#_Toc487008568)

[3.5. Versionarea codului 75](#_Toc487008569)

[Concluzii 77](#_Toc487008570)

[Bibliografie 79](#_Toc487008571)

[Anexe 80](#_Toc487008572)

[CV 83](#_Toc487008573)

# 

# Lista Figurilor

Figura 1.1 Timp de Execuție Bubble Sort 15

Figura 1.2 Timp de Execuție Bubble Sort 16

Figura 1.3 Popularitate PHP,Node JS 16

Figura 1.4 Componente importante Node JS 18

Figura 1.5 Structura arborescenă a serverului 22

Figura 1.6 Structura blocurilor de cod în raport cu Sequelize 23

Figura 1.7 Structura bazei de date 28

Figura 1.8 JWT semnare token 29

Figura 1.9 Verificare token 30

Figura 2.1 Concept Model View Controller 34

Figura 2.2 Arhitectura Unei Aplicații cu Angular 2 35

Figura 2.3 Arhitectura Unei Componente Angular 2 35

Figura 2.4 Arhitectura Modulelor în Angular 2 36

Figura 2.5 Încapsularea ES5 și ES6 în TypeScript 37

Figura 2.6 Conținutul folderului /public 38

Figura 2.7 Schema de funcționare a aplicației client 39

Figura 2.8 Mod de operare JWT 41

Figura 2.9 Verificare nivel de securitate înainte de cerere către server 42

Figura 2.10 Structura componentelor în aplicația Client 45

Figura 2.11 Principii Material Design bară laterală 48

Figura 2.12 Principii Material Design meniu 49

Figura 2.13 Principii Material Design butoane 49

Figura 3.1 Bară de navigare în modul desktop 51

Figura 3.2 Bară de navigare în modul mobile (Hamburger Menu) 52

Figura 3.3 Afișare notificare Toast pentru mesaj de success 52

Figura 3.4 Exemplu de poziționare a butonului sub formular 53

Figura 3.5 Buton pentru înregistrare valori din formulare 53

Figura 3.6 Buton pentru adăugare 53

Figura 3.7 Buton pentru editare 53

Figura 3.8 Buton pentru ștergere 54

Figura 3.9 Buton pentru note 54

Figura 3.10 Notificare pentru atenționare acces nepermis 54

Figura 3.11 Pagina de logare 55

Figura 3.12 Înregistrare utilizator, pasul 1 56

Figura 3.13 Înregistrare utilizator, eroare număr matricol incorect 56

Figura 3.14 Înregistrare utilizator, pasul 2 57

Figura 3.15 Înregistrare utilizator, username-ul există 57

Figura 3.16 Înregistrare utilizator cu success 58

Figura 3.17 Logare site, utilizatorul nu există 58

Figura 3.18 Logare site, parola greșită 58

Figura 3.19 Pagina home, cont student 59

Figura 3.20 Pagina de note a studentului 59

Figura 3.21 Suprafața extinsă cu informații despre student 60

Figura 3.22 Selecția notelor ce se doresc afișate 60

Figura 3.23 Pagina home, profil administrator 61

Figura 3.24 Administrare Discipline 61

Figura 3.25 Pagina Adăugare/Editare Discipline 62

Figura 3.26 Pagină de vizualizare Specializari 63

Figura 3.27 Pagină Adăugare/Editare Specializare 63

Figura 3.28 Pagină afișare studenti 64

Figura 3.29 Pagină adăugare student 65

Figura 3.30 Pagină editare student 65

Figura 3.31 Editare detalii grupa pentru studentul administrat curent 66

Figura 3.32 Înscrierea studentului într-o grupă 66

Figura 3.33 Pagină administrare note student 67

Figura 3.34 Pagină adăugare/editare notă student 68

Figura 3.35 Selector dată 68

Figura 3.36 Selector discipliăa pentru o notă 69

Figura 3.37 Pagină adăugare/modificare utilizator 69

Figura 3.38 Formular pentru utilizator de tip administrator 70

Figura 3.39 Afișare meniu în modul mobil 70

Figura 3.40 Pagină pentru discipline în modul mobil 71

Figura 3.41 Pagină pentru specializări în modul mobil 71

Figura 3.42 Pagină Studenți în modul mobil 72

Figura 3.43Pagină Note în modul mobil 72

Figura 3.44 Pagină Utilizatori în modul mobil 73

Figura 3.45 Mod de operare GIT 76

Figura 4.1 - Lansare în execuție web-service 80

Figura 4.2 - Compilare cod SASS 80

Figura 4.3 - Lansare în execuție aplicație client 81

Figura 4.4 - Rezultate consolă pentru logare utilizator 81

Figura 4.5 - Consolă de execuție aplicație client 82

# Lista tabelelor

Table 1.1 Limitări PostgreSQL 20

# Lista acronimelor din proiect

API - Application Programming Interface

CLI - Command Line Interface

CMS - Content Management System

CROS - Cross-origin Resource Sharing

CSS - Cascading Style Sheets

DNS - Domain Name Server

DOM - Document Object Model

HTML - Hypertext Markup Language

HTTP - Hypertext Transfer Protocol

JSON - Javascript Object Notation

JWT- Json Web Token

MIT - Massachusetts Institute of Technology

MVC - Model-View-Controller

MVVM - Model-View-Viewmodel

NPM - Node Package Manager

ORM - Object Relational Mapping

SASS - Syntactically Awesome Style Sheets

SGBD - Sistem de gestiune al bazei de date

SQL - Structured Query Language

UX - User Experience

VCS - Version Control System

XML - Extensible Markup Language

# Introducere

Lucrarea ce urmează a fi prezentată își propune realizarea unei aplicații web pentru gestiunea situației școlare și financiare a studenților din cadrul Facultății de Electronică, Calculatoare și Telecomunicații. Scenariul pe care aplicația ce urmează a fi realizată și descrisă în capitolele următoare trebuie să îl soluționeze este acesta: un student înmatriculat în cadrul universității trebuie să aibe posibilitatea de a accesa o pagină web în interiorul căreia își poate vedea notele, poate afla informații cu privire la situația financiară, ceea ce constă în posibilitatea de a verifica dacă este înmatriculat în anul curent într-un plan cu subvenție sau este necesară plata unei taxe universitare. În acest context voi discuta despre o aplicație web ce va fi capabilă să rezolve scenariul propus, dar totodată să ofere siguranță și încredere utilizatorilor ce îl vor accesa. Voi ține cont de asemenea și de timpul de acces, având ca principal scop îmbunătățirea vieții studențești prin accesul rapid la informație, în acest context informația reprezentând notele. Nu voi neglija nici aspecte precum structurarea și aranjarea acestor informații în pagină, deoarece sunt de părere că unui utilizator nu îi trebuie un ghid de utilizare în prealabil, ci interfața grafică trebuie sa vorbească de la sine.

Motivele pentru care această temă mi-a atras interesul în mod deosebit, față de celelalte afișate în lista de proiecte disponibile, constau în soluționarea unei necesități pe care facultatea din care fac parte o gestionează, dorința de a pune în practică informațiile acumulate pe parcursul celor patru ani de studiu, atât din cadrul facultății, cât și ca rezultat al studiului personal în direcția programării web,

din dorința de a pune la dispoziție generațiilor următoare, un serviciu, pe care l-am dorit și eu în anii de studenție și totodată consider că această temă constituie o bază solidă pentru îmbogațirea cunoștințelor mele despre actualele tehnologii utilizate în producție.

Am hotărât ca aplicația pe care o voi dezvolta să fie alcatuită din 3 subsisteme, reprezentând o aplicație client-server, fiecare subsistem reprezintă o componentă a aplicației finale. Creierul sistemului este definit în aplicația server și are ca principal scop interacțiunea dintre utilizator și informații, acesta trebuie să fie apt să răspundă cererilor venite de la aplicația client, să ofere informații stocate în baza de date și sa înregistreze noi date. Subsistemul al doilea, reprezintă componenta ce va interacționa în mod direct cu utilizatorii, acesta constă în aplicația client și va conține metode de interacțiune a utilizatorilor cu serviciul web, din primul subsistem. Cel de-al treilea subsistem, constă într-un server ce va rula un sistem de gestiune al bazei de date alese.

Lucrarea va fi structurată în 3 capitole, ce vor prezenta detaliat câte o temă. Fiecare capitol își propune să redea pe larg aspecte ce țin de tehnologiile utilizate, modalitatea și motivele pentru care au fost alese, dar și beneficiile pe care le aduc proiectului în comparație cu alte variante propuse. Am ales această structură a capitolelor deoarece mi s-a părut o abordare potrivită, de a oferi informațiile cititorului, astfel încât acesta să abordeze toate temele sistemului, într-o ordine care să îi permită înțelegerea modului în care aplicația operează.

Capitolul 1 – Web Service, reprezintă primul, dar totodată cel mai amplu capitol, care aduce în prim plan tehnologia utilizată pentru realizarea subsistemului ce ruleaza web-service-ul. Alegerea nu a fost una simplă, dar am conchis asupra tehnologiei Node JS, deoarece este un limbaj modern, iar aplicația necesită rapiditate, stabilitate și totodată economie de resurse. Am analizat trei limbaje de programare pe server PHP, NodeJS și Golang(GO) din mai multe perspective, iar rezultatele obținute, cât și dorința de a învața o nouă tehnologie, m-au determinat să aleg acest limbaj de programare pe server. Deși reprezintă un serviciu independent, baza de date, va fi detaliată în interiorul acestui capitol fiind în stransă legatură cu web-service-ul. Am ales tehnologia PostgreSQL, deoarece aceasta este o bază de date relaționară bazată pe obiecte, capabilă să opereze cu date de dimensiuni mari și să ofere în același timp, un rezultat potrivit, din punct de vedere al timpului de execuție. În cadrul acestui capitol vor fi prezentate de asemenea și metode de prevenție, implementate în cadrul web-service-ului care să elimine potențiale atacuri, în încercarea unei persoane neavizate de a obține datele personale ale studenților.

Capitolul 2 – Client Web: cel de-al doilea capitol al lucrării, va oferi detalii despre tehnologiile utilizate în aplicația client, printre care pot remarca Angular 2, Typescript, Javascript și alte module și librării. În ceea ce privește subsistemul, vor fi detaliate aspecte ce țin de modul în care acesta a fost gândit și organizat pentru a oferi cele mai bune servicii, utilizatorilor. Dacă în perioada de început a comerțului, ambalajul avea rol pur protector, astăzi putem observa cu ușurință faptul că ambalajul vinde produsul, afirmație care susține următoare idee: o interfață placută va reprezenta un plus în fața utilizatorilor.

Capitolul 3 – Interacțiunea cu factorul uman: ultimul captiol, dar cu rol decisiv în susținerea îndeplinirii obiectivului stabilit inițial. Acest capitol are ca obiectiv prezentarea modului în care elementele vor fi afișate în pagină, logica din spatele acestui aranjament și impactul pe care îl are asupra utilizatorilor. Pentru ca aplicația să aibe efectul dorit, aceasta necesită înainte de toate o analiză asupra dorințelor publicului, iar această analiză am încercat să o simulez prin transpunerea personală în pielea utilizatorului și prin chestionarea colegilor. Am încercat de asemenea în cadrul acestui capitol să mă bazez pe psihologia umană, pe efectul cromatic asupra preferințelor umane, dar și pe experiența personală ca utilizator al unei aplicații. Pe de altă parte rolul acestui capitol este și acela de a oferi informații despre modul în care sistemul creat poate fi utilizat.

Acest proiect va reprezenta o provocare personală de a gândi un sistem atât de amplu, pentru a gestiona un număr mare de utilizatori și o interacțiune crescută în raport cu aceștia. Platforma va integra armonios, printr-o interfață grafică simplă și o experiență ergonomică în raport cu utilizatorul, tehnologii web moderne: HTML5, CSS3, JavaScript, NodeJS, Angular 2, TypeScriptPostgreSQL etc., având ca obiectiv satisfacția utilizatorului, dar și cea personală de a crea o aplicație integral, pornind de la zero, utilizând noile tehnologii utilizate pe piață.

# Stadiul Actual

Aplicația pe care am realizat-o este proiectată și implementată integral de mine, pornind de la 0 ca o reacție în urma unei analize de piață. Înainte de definirea unor probleme pe care aplicația trebuie să le rezolve, în afară de obiectivul central, afișarea notelor pentru studenți, am început să caut platforme disponibile gratuit (open-source) sau plătite. Problema majoră a sistemelor pe care le voi prezenta în rândurile următoare constă în faptul că sunt învechite și nu oferă avantaje precum o experiență placută a utilizatorilor, design ce conține elemente și tehnologii noi, acces rapid la informație, actualizări de securitate sau arhitecturi scalabile ce pot îmbunătății modul de dezvoltare.

Dintre sistemele pe care le-am verificat am ales să prezint doar o parte dintre acestea. Le-am ales deoarece erau cel mai aproape de îndeplinirea obiectivului. Primul sistem este unul care își propune ca principal obiectiv să rezolve problema carnetului digital în rândul elevilor de liceu. Poate fi găsit la următoarea adresă web http://www.catalogulmeu.ro/ și conține de asemenea și o secțiune demonstrativă pentru cele 3 nivele de acces pe care le pune la dispoziție: elev, părinte și profesor. Însă dezavantajele acestui sistem, pe care le-am observat încă de la primul contact sunt: interfața grafică, total neintuitivă pentru utilizator în găsirea informațiilor necesare. Am încercat minute în șir să găsesc secțiunea din cadrul căreia pot adăuga/modifica note, însă fără succes. Deși la prima vedere, taburile afișate în partea de sus a paginii promit să ofere esențialul, lucrurile se schimbă în momentul în care acestea sunt accesate, oferă informații suplimentare și mult prea aglomerate, astfel oferind impresia de haos. O parte dintre aceste secțiuni sunt goale și nu oferă absolut nici-o informație cu privire la motivul pentru care conținutul este inexistent. Ca o concluzie pentru acest prim sistem, am decis să continui căutarea, deoarece existau suficiente motive pentru care să nu fie ales. Un al doilea sistem pe care l-am eliminat în primele 30 de secunde poate fi găsit la adresa web https://noteincatalog.ro/, însă proprietarii nu oferă platforma spre uz personal, ci doar pe baza unui abonament lunar în valoare de 5 Ron/elev.

Un sistem demn de luat în calcul, reprezintă Rosariosis, care poate fi obținut gratuit, fiind distribuit sub o licență open-source, de pe github, la adresa web https://github.com/francoisjacquet/rosariosis, deoarece pune la dispozitie o serie de avantaje și îndeplinește fără probleme o serie de condiții inițiale ale proiectului pe care trebuie să îl realizez. Este un sistem dezvoltat în mai multe limbi, cu acces la setări despre Universitate, utilizatori, studenți, note ale studenților, afișarea situației calendaristice a examenelor, afișarea unei secțiuni de notificări, diferite rapoarte pe baza notelor, studenților și alte informații. Însă, deși la o primă vedere, pare proiectul perfect, ce poate fi îmbunătățit să respecte condițiile FECC, acesta are aceiași problemă pe care o au și celelalte, este dezvoltat pe tehnologii vechi, însă arhitectura este destul de bine proiectată, fiind bazată pe module, astfel extinderea proiectului pare la prima vedere mult mai simplă. Este dezvoltat în PHP, ca o aplicație de sine stătătoare, ceea ce nu permite operarea în timp real. Am ales totuși să continui căutarea, deși aceasta a reprezentat cea mai bună opțiune.

În afară de sisteme prezentate anterior, care au ca principal obiectiv afișarea notelor pentru elevii de liceu sau studenți, am găsit o serie de alte platforme ce oferă o diversitate a serviciilor oferite. Acestea conțin și o platforma e-learning, platformă de blog, evenimente, rețele sociale, calendare și multe altele. Însă nu îndeplinesc cerințele de bază.

Am luat în calcul și varianta de a utiliza una dintre platformele existente ca o fundație pentru sistemul pe care urmează să îl prezint, însă avantajele pe care această abordare le oferă constau în simplu fapt ca startul nu se facea de la 0, ci de la un procent de aproximativ 40-50%. Consider acest procent de start, deoarece sistemele disponibile, pe care le-am verificat oferă servicii generale, ce pot fi aplicate în cadrul unor facultăți sau licee, însă acestea trebuiau modificate pentru a îndeplini condițiile necesare în cadrul FECC. Pe lângă acest aspect era nevoie de asemenea și de îmbunătățirea arhitecturii, a design-ului, a timpului de access la informație, dar foarte probabil și a codului. Totodată am analizat și dificultatea dezvoltării și implementării tuturor celor enumerate anterior și am concluzionat faptul că timpul de dezvoltare, dar și proiectul final ar fi afectate, deoarece este de preferat proiectarea unui sistem care sa îndeplinească de la început cerințele și să ofere avantaje față de competiție. Astfel am decis că cea mai bună opțiune o reprezină dezvoltarea unui sistem pornind de la 0.

# Web Service

## Introducere

Pentru acest capitol mi-am propus să explic care este rolul unui webservice și de ce este de preferat utilizarea lui în defavoarea unei aplicații unitare, informații generale despre tehnologiile utilizate și arhitectura sistemului. Vor fi prezentate de asemenea aspecte ce țin de  analiza realizată în timpul proiectării sistemului, factorii care mi-au influențat alegerea precum și dorințele personale care au stat la bază, în găsirea tehnologiilor optime pentru acest proiect.

Utilizarea web-service-urilor în aplicațiile moderne a devenit din ce în ce mai frecventă, deoarece permit conectarea mai multor aplicații client. Web Service-ul este un serviciu oferit pentru a transfera date prin HTTP de la un dispozitiv către altul prin formate text, precum XML sau JSON.

În cadrul aplicației ce urmează a fi realizată, am considerat importantă utilizarea unui webservice deoarece:

* pot fii utilizate mai multe aplicații client, în afară de cea inițială, web;
* sistemele sunt separate, ceea ce permite o mai bună dezvoltare și mentenanță sporită; bază de date, web-service-ul și aplicația client putând fii separate și utilizate de pe diverse servere aflate în diverse locații;
* scalabilitate: modificarea webservice-ului curent sau a unui modul apartenent, pentru a putea fi utilizat în aplicații viitoare;
* structurarea codului într-o formă mult mai plăcută, decât în cazul aplicațiilor dedicate;
* rapiditate, având resursele partajate doar pentru un singur tip de sarcină;
* securitate sporită, având în componență 3 subsisteme care sunt dezvoltate independent;

Datele furnizate de webservice sunt în format JSON, putând fi utilizate cu ușurintă în limbajul Javascript.

## Faza de proiectare a webservice-ului

În această etapă de proiectare, obiectivul pe care mi l-am propus a fost găsirea unor tehnologii fezabile care să îmi permită realizarea unui serviciu web rapid, în timp real. Prin aplicație web în timp real se întelege că website-ul folosește tehnologii și practici ce permit utilizatorilor să vadă conținutul, actualizat, fără ca ei să facă o nouă cerere către server pentru a verifica dacă există noi date (să reîmprospăteze pagina) [7]. Această noțiune de real-time în aplicațiile web se diferențiază de cea din tehnica de calcul, unde nu se știe când sau dacă se va primi un răspuns [7].

Performanța, însă, nu reprezintă singurul și cel mai important criteru. Pe a doua poziție, pentru găsirea tehnologiei ce o voi folosi în realizarea aplicației am avut în vedere utilizarea unei tehnologii open-source. Prin model open-source se întelege un model de lucru descentralizat, în cadrul căruia mai mulți dezvoltatori colaborează din același loc sau din locuri geografice diferite, pentru realizarea unui produs sau serviciu, oferit gratuit către public sub o licență specifică [8]. Astfel utilizând o tehnologie open-source costurile de producție pentru webservice mențin o linie scazută.

Dacă pentru cerințele de mai sus m-am transpus în postura unui potențial client, în continuare ca dezvoltator al acestei aplicații mi-am propus să urmăresc câteva direcții, în strânsă legatură cu procesul de dezvoltare, precum: comunitatea de dezvoltatori ce folosesc această tehnologie, documentația și efortul depus utilizând o anumită tehnologie. Deși aparent cele trei elemente enumerate mai sus nu au o legatură directă, în fapt acestea sunt absolut dependente una față de cealaltă. Indiferent de ce sistem se dorește dezvoltat, documentația este foarte importantă, deoarece producătorul trebuie să își exprime în mod redactat, video sau audio ideile folosite în realizarea produsului final, domeniul de utilizare și limitările pe care le are, de asemenea descrierea tuturor funcțiilor ce pot fii folosite și a parametrilor necesari pentru configurare. În cazul în care documentația are mici lipsuri, comunitatea de dezvoltatori poate fi extrem de utilă, deoarece aceștia pot veni în rezolvarea unei soluții rapide, care poate reduce astfel timpul necesar pentru rezolvarea unei probleme cauzată de minusurile acelui produs. Produsul oferit poate fi extrem de util și de asemenea extrem de customizabil, reușind astfel să reducă minusurile spre 0, însă este la fel de important ca dezvoltarea propriu-zisă utilizând acel produs să fie încetinită de modul în care acesta trebuie customizat, astfel un produs fezabil este acela care îndeplinește nevoile de proiectare și nu este neapărat nevoie să fie capabil de mai mult.

Menținând ideea de fezabilitate a sistemului și din dorința de a crea un sistem care să îmi permită o maximă manevrabilitate a resurselor, mi-am propus încă de la început ca sistemul să fie modular. Consider ca tehnologia a avansat atât de mult în zilele noastre încât este o mare pierdere de timp ca un sistem să fie construit de la 0, astfel putând fi folosite părți ale unui alt proiect în crearea unui nou proiect. Tocmai din aceste motive, aplicația pe care o doresc va trebui construită sub forma unor blocuri de lego, care conectate în locul potrivit ajută la realizarea unui întreg proiect.

Având în vedere lista cu toate cerințele menționate mai sus, am început cercetarea prin a compara trei tehnologii cunoscute și utilizate în mediul web: PHP (Hypertext Preprocessor), NodeJS și Goglang. Astfel au apărut următoarele rezultate:

1. **Performanțe**
   1. **PHP** - este gratuit, un limbaj intepretat cu ajutorul motorului Zend și executat; este single-thread cu posibilitatea de a fi utilizat ca multi-thread prin intermediul unei extensii [1].
   2. **NodeJS** - este gratuit, are la baza mediul de lucru Javascript, fiind un limbaj intepretat, cu ajutorul motorului Google V8 [2]. Este un limbaj de programare  asincron, single-thread și non-blocant [2].
   3. **GO** - este cunoscut de asemenea sub denumirea de Golang, este gratuit, a apărut în 2007, fiind dezvoltat de Google. Este un limbaj de programare multi-thread, se compilează și își propune să reprezinte o variantă îmbunatațită a C-ului clasic [3].
2. **Timp de executie**

Pentru a compara timpii de execuție am ales un algoritm clasic, Bubble Sort, iar rezultate au fost următoarele :

* PHP - 3.835s
* NodeJS - 0.071s
* Golang - 0.038s

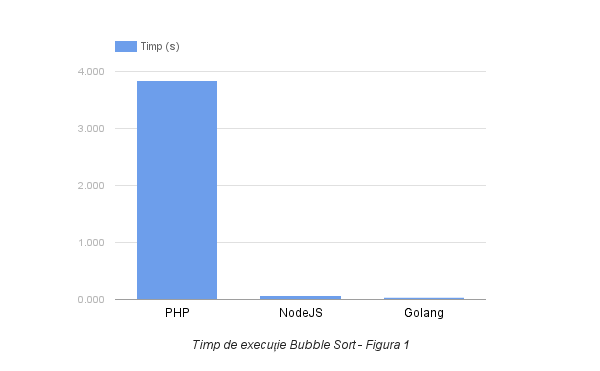


Figura 1.1 Timp de Execuție Bubble Sort

Sursa: [4]

În urma acestui test am concluzionat faptul că Go este cel mai rapid dintre cele trei, iar PHP-ul este un limbaj bătrân și totodată lent.

1. **Timpul de răspuns al serverului HTTP**

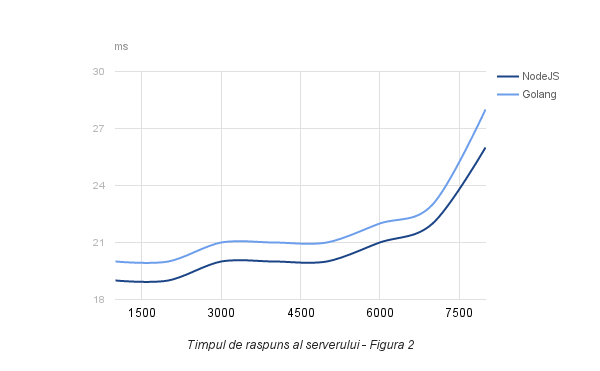


Figura 1.2 Timp de Execuție Bubble Sort

În urma acestui grafic am ajuns la concluzia că un timp de răspuns mai mic, înseamnă totodată și un avantaj, astfel în cadrul acestui test NodeJS a învins, deși o milisecundă nu reprezintă o diferență atât de mare [4].

1. **Costuri**

În ceea ce privește costurile, toate cele trei limbaje sunt gratuite și sunt susținute de comunitatea de dezvoltatori, astfel un alt criteriu de analiză fiind numărul de dezvoltatori activi.

1. **Comunitatea de dezvoltatori**

Dacă privim rezultate unui sondaj realizat de StackOverflow pe parcursul anilor 2013, 2014, 2015, 2016 cu privire la popularitatea în materie de limbaje de programare web, numărul dezvoltatorilor care utilizează limbajul PHP este în scădere, spre deosebire de numărul dezvoltatorilor de NodeJS, care au o creștere semnificativă; în ceea ce privește limbajul Golang, acesta nu se află în topul preferințelor dezvoltatorilor.

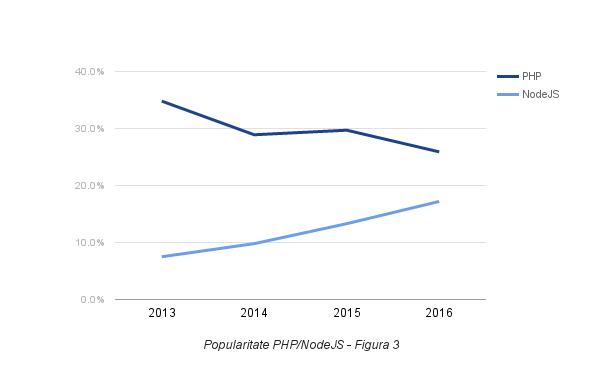


Figura 1.3 Popularitate PHP,Node JS

Sursa: [5]

Acest criteriu a fost foarte important pentru mine în alegerea tehnologiei, deoarece o comunitate solidă ajută la realizarea rapidă a proiectului. Fiind vorba despre limbaje de programare gratuite, documentația oficială poate fii de multe ori sumară sau poate să nu acopere toată plaja soluțiilor care pot fi rezolvate cu ajutorul acelui limbaj, astfel dezvoltatorii sunt cei care pot oferi sprijinul necesar.

1. **Documentația**

Conform celor spuse la punctul anterior, documentația reprezintă un alt criteriu important de luat în considerare, în alegerea unui limbaj ce urmează a fi utilizat în cadrul unui proiect.

În căutarea materialelor care îmi vor oferi sprijin în dezovoltarea aplicației, vârsta limbajelor a fost direct preponderentă cu rezultatele găsite. Astfel, cele mai multe rezultate au fost în favoarea PHP-ului, urmat în clasament de NodeJS, iar în final situându-se GO.

1. **Ușurința în dezvoltare**

Cu siguranță acesta este aspectul cel mai important în realizarea oricărei aplicații, deoarece un limbaj de programare greoi poate duce de cele mai multe ori la eșecul proiectului, nepopularitatea acestuia, durata mare de realizare și mai ales dificultate în proiectare.

Pe baza celor menționate mai sus, cât și din dorința de a învăța o nouă tehnologie am hotărât că cea mai potrivită tehnologie dintre cele trei: NodeJS, PHP, GoLang pentru dezvoltarea web-service-ului este NodeJS.

## Introducere în NodeJS

Prima lansare a avut loc pe 27 mai 2009, fiind conceput de dezvoltatorul Ryan Dahl și scris în C, C++ și Javascript. NodeJS este un mediu de lucru open-source, ce poate fi rulat pe majoritatea sistemelor de operare și permite execuția codului Javascript pe server [2]. La început limbajul Javascript a fost conceput pentru a fi utilizat, strict, în partea de client a unei aplicații, fiind executat de un motor de execuție ce permite compilarea codului, de către browser, însă conceptul oferit de NodeJS vine în totală antagonie cu ceea ce presupune de fapt limbajul Javascript, deoarece acesta permite rularea codului Javascript pe un server care compilează codul înainte de a fi utilizat [2].

Pe parcursul anilor NodeJS a devenit unul dintre cele mai folosite servere datorită timpului mic de răspuns, a faptului că operează asincron, fără a pune în pericol blocarea cererilor și a mediului de dezvoltare Javascript, care este un limbaj simplu, ușor de înțeles, întreținut și utilizat, dar totodată și a faptului că este utilizat din ce în ce mai mult de aplicațiile client [2].

Serverul gestionează toate cererile într-un loop, operând asincron pe un singur thread.

Procedeul asincron este o forma de procesare a intrarilor/ieșirilor care permite altor procese să continue, fără ca cele de dinainte să fie întrerupte [9]. Acest lucru îi permite serverului să opereze cu zeci de mii de conexiuni concurente fără a crește costul pentru thread-ul folosit [9]. De asemenea poate fi folosit și cu fire de execuție, principalul thread fiind capabil să creeze sau gestioneze un astfel de mod de lucru [2].

Motorul de execuție folosit poartă numele V8 și a fost construit inițial pentru Google Chrome. Acesta a fost scris în C++ cu scopul de a compila codul Javascript în cod mașină în loc să fie interpretat de browser în timp real. Codul compilat este optimizat și recompilat la fiecare rulare [2].

NPM este managerul de pachete implicit folosit de NodeJS și vine preinstalat [10]. Este scris complet în Javascript și a fost dezvoltat de Isaac Schlueter cu dorința de a crea un manager suficient de bun, care să satisfacă dorințele dezvoltatorilor, să le permită accesul rapid la multitudinea de module existente [10]. Un modul reprezintă un serviciu sau o parte a unei aplicații, dezvoltate de un programator și oferit pentru a fii folosit de toți ceilalți dezvoltatori, un exemplu de modul este bcrypt care permite criptarea unui string în funcție de configurația aleasă. Modulele au un rol extraordinar de important în dezvoltarea unei aplicații deoarece acestea reduc considerabil timpul de dezvoltare cât și probleme ce pot interveni în timpul dezvoltării. Interacțiunea cu NPM se face prin folosirea unor comenzi în linie de comandă și permit instalarea, actualizarea sau ștergerea unor module, cât și comenzi mult mai complexe. Modulele care vin preinstalate cu NodeJS se găsesc în directorul CommonJS [10]. Avantajul major pe care îl oferă, constă în faptul că orice proiect nodejs are în componență un fișier .json ce conține toate dependențele locale sau globale, iar printr-o simplă linie de comandă toate aceste module dependente pentru buna funcționare a proiectului vor fii instalate [10].

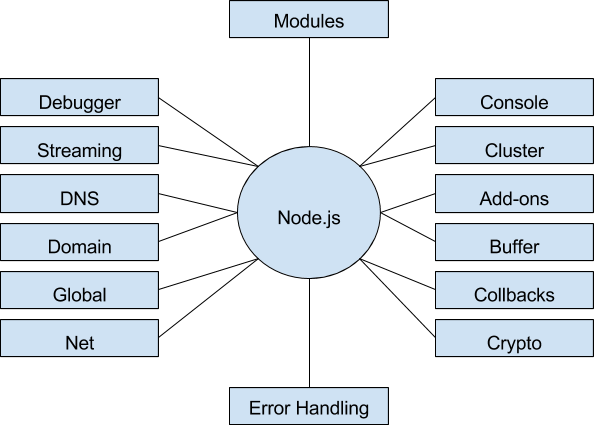


Figura 1.4 Componente importante Node JS

Printre beneficiile pe care le oferă NPM, trebuie menționat și faptul ca dezvoltatorii creează module și le încarcă în aceasta platformă pentru a fi folosite de ceilalți dezvoltatori, practic roata este șlefuită, pentru a nu fi creată de fiecare dată când se începe dezvoltarea unui proiect nou. Tocmai din aceste motive au apărut framework-urile, care au rolul de a ușura munca dezvoltatorilor, punând la dispoziție printr-un singur apel o funcționalitate care poate avea în spate multe linii de cod. Utilizarea unui framework în dezavantajul unei tehnologii native constă în rapiditate, consecvență și securitate. Faza de dezvoltare pornește astfel cu un avans considerabil, deoarece o mare parte din funcționalitățile necesare pentru implementare există, rămâne doar ca acestea să fie utilizate în modul dorit, codul este mult mai bine structurat, având un template pe care se ghidează, iar securitatea este cu mult ridicată, deoarece având în spate un număr mare de dezvoltatori șansele ca breșele de securitate să existe sunt tot mai mici, iar când acestea sunt descoperite ele sunt oferite printr-o actualizare.

De asemenea trebuie menționat faptul că rolul acestui manager de pachete nu este doar de a căuta și a instala module, el pune la dispoziție o sumedenie de comenzi în linie de comandă, menite să crească eficiența și rapiditatea în dezvoltare, a programatorului. Prima și cea mai importantă comandă pe care un programator trebuie să o cunoască în inițierea sa ca programator node js este *npm init*, această comandă permite setarea directorului curent ca unui repository node js. În urma execuției acestei comenzi, în consolă vor fi enunțate mai multe întrebări, precum: numele proiectului ce se dorește creat, versiunea, scurtă descriere a proiectului ce se dorește creat, fișierul de start ce se va executa (implicit index.js), sursa unui repository git (sistem de versionare a codului), cuvinte cheie, numele autorilor și tipul licenței sub care va fi distribuit proiectul. În cazul în care se apasă tasta enter, sistemul va oferi răspunsuri implicite, iar la final toate răspunsurile vor fi salvate în fișierul package.json, și de asemenea va fi creat un folder node\_modules unde vor fi salvate module necesare funcționării serverului node.

Pentru acest proiect am considerat că cel mai potrivit framework este Express. Express.js este un framework NodeJS lansat ca software gratuit sub o licență MIT în noiembrie 2010. Pe site-ul producătorului este descris ca un framework minimalist și flexibil înzestrat cu o serie de funcționalități menite să ajute în realizarea aplicațiilor web și mobile [11]. Este de asemenea perfect pentru crearea API-urilor, punând la dispoziție nenumărate metode pentru apeluri HTTP și middleware, simplu și rapid [11]. Comunitatea este în creștere, iar numărul aplicațiilor care au la bază acest framework este foarte mare [11].

Limbajul Javascript a trecut de-a lungul timpului prin mai multe schimbări și actualizări, cel mai recent standard utilizat este ES2015 sau ES6, apărut în iunie 2015, care aduce îmbunătățiri considerabile și transforma limbajul Javascript într-o armă în mâinile dezvoltatorilor. Printre îmbunătățirile considerabile, pe departe cea mai importantă îmbunătățire o constituie apariția claselor și a conceptului de programare orientată pe obiecte. Marele dezavantaj alături de care vine la pachet îl constituie faptul că browserele web nu sunt pregătite pentru o astfel de schimbare, standardul actual înțeles de acestea fiind ES5. În cadrul acestui web-service voi folosi standardul ES6, pentru a beneficia însă de această oportunitate este nevoie ca serverul să poată interpreta noul format al codului, fapt pentru care voi utiliza un modul din NPM, numit Babel CLI.

Lucrul cu clase, în Javascript, la fel ca și în celelalte limbaje de programare se poate face sub mai multe forme de confidențialitate. Astfel, pentru a face disponibilă o clasă, pentru a fii utilizată în interiorul altei clase se folosește cuvântul cheie *export*, înaintea declarării clasei ( exemplu: *export class test { … }* ) sau de asemenea prin plasarea după definirea clasei cu ajutorul cuvântului cheie class a structurii *export default nume\_clasa;*. Fără utilizarea uneia dintre aceste doua metode, o clasă nu poate fii importată. Importul unei clase se face cu ajutorul cuvântului *import* (exemplu: *import nume\_variabila from ‘calea\_catre\_fisier’* ). O clasă poate avea un constructor și metode ce pot fii statice sau non-statice, pot returna o valoare sau pot fii de tip void. Este de menționat faptul ca pentru declararea metodelor nu este necesară utilizarea unor cuvinte cheie ce definesc tipul rezultatului ce va fii returnat, limbajul javascript având grijă singur de acest aspect.

## Sistemul de gestiune al bazei de date (SGBD)

În rândurile următoare voi prezenta aspecte legate de baza de date și alegerea SGBD-ului. Printre aspectele pe care le-am avut de luat în calcul s-au înumărat la fel ca și în alegerea web-service-ului: performanța, costurile, timpul de execuție, rapiditate în dezvoltare, documentația, comunitatea, dar totodată și aspecte ce țin strict de SGBD:

* ușor de folosit atât indepedent cu ajutorul unui soft oferit de producător, cât și conectat la un alt sistem (în cazul de față web-service-ul creat pentru aplicația de gestionare a studenților);
* capacitatea de a afișa rapid și structurat datele și rezultatele interogarilor, deoarece în cazul în care datele returnate în urma interogărilor sunt oferite într-un format greu de interpretat, acest aspect poate produce incoerență și greșeli de afișare corecta a datelor;
* securitate sporită: asistăm la o perioadă în care tehnologia se schimbă continuu și războaiele nu mai au aceiași formă ca acum sute de ani, un razboi reprezintă de fapt o luptă cibernetică pentru a desconspira datele unei organizații sau companii, deci securitatea reprezintă un oponent în calea unui cal troian;
* existența unor funcții predefinite, susțin munca rapidă și eficientă, deoarece este mult mai ușor și sigur să folosești o funcție predefinită care a fost concepută și rafinată pentru a oferi cele mai bune rezultate pe sistemul respectiv, în defavoarea unei funcții customizate de dezvoltator;

Am luat în calcul 5 sisteme de gestiune: Neo4J, PostgreSQL, MySQL, MariaDB și MongoDB, toate dintre acestea îndeplinind condițiile enumerate mai sus. Primele 4 sunt sisteme de gestiune SQL, iar MongoDB este de tip Nonsql, ceea ce m-a făcut să renunț la cea din urmă este dată de faptul că în sistemul ce se dorește construit, avem nevoie de relații între tabele, lucru ce poate fii realizat și într-o bază de date nonsql, dar neavând ca principal obiectiv acest aspect, precum celelalte. Neo4j este un sistem de gestiune de tip graf, foarte rapid, capabil să lucreze cu date extrem de mari (big-data) și foarte sigur. Însă deși îndeplinește toate condițiile de mai sus, am renunțat la acesta deoarece utilizarea și implementarea acestui sistem era mult mai dificilă decât a celorlalte, iar beneficiile pe care le oferă în plus, nu erau necesare sistemului ce se dorește creat. Astfel au rămas de analizat 3 sisteme, MySql și MariaDB, neavând prea mari diferențe, nucleul acestora fiind aproape identic, iar utilizarea lor se axează în mare parte spre sistemele realizate în PHP, fiind achiziționată recent de Oracle, MySQL se înclină cu pași repezi către segmentul premium, deși oferă și funcționalități open-source, iar apariția sistemului MariaDB reprezintă de fapt un răspuns la această decizie. Dintre cele 3 variante rămase în calcul, PostgreSQL este cea mai folosită și totodată cea mai veche dintre toate și singura pe care nu am utilizat-o în trecut, fapt care m-a și determinat alături de celelalte aspecte să o aleg ca tehnologie pentru dezvoltarea acestui sistem.

**PostgreSQL** - reprezintă un sistem de gestiune al bazelor de date bazat pe relații între obiecte, gratuit și extrem de puternic, ce a apărut în urma cu mai bine de 20 de ani, pe data de 9 iulie 1996 și poate fi rulat pe majoritatea sistemelor de operare importante (Windows, Mac OS,  distribuții Linux) [6]. Este scris în C, are compatibilitate cu o varietate de limbaje de programare și suportă de asemenea: operații cu chei străine, triggere, vederi, indexi, join-uri, proceduri stocate și tranzacții. Începând cu anul 2008, au fost introduse un număr semnificativ de tipuri de date suportate: boolean, numeric cu precizie a virgulei, caractere (text, string, char), binar, data/timp, monedă, array, json și altele [6].

Ca orice sistem construit, are anumite limitări ce pot fi verificate în tabelul următor:

|  |  |
| --- | --- |
| **Limita** | **Valoare** |
| Dimensiunea maximă a bazei de date | nelimitat |
| Dimensiunea maximă a unei tabele | 32TB |
| Dimensiunea maximă a unei linii | 1.6TB |
| Dimensiunea maximă a unei celule | 1GB |
| Număr maxim de coloane pe tabelă | 250 - 1600 (în funcție de tipul coloanei) |
| Număr maxim de indexi pe tabelă | Nelimitat |

Table 1.1 Limitări PostgreSQL

**Sequelize** este o librărie și un modul NodeJS, scris în totalitate în Javascript ce oferă o legatură relaționară între obiecte (ORM - Object Relational Mapping), făcând astfel posibilă extragerea informațiilor dintr-o bază de date sub forma unor obiecte javascript, ce pot fi manipulate cu usurință [12]. Apărut în iunie 2010, oferă ca dialecte SQL suportate: PostgreSQL, MySQL, MariaDB, SQLite și SQL Server. Motivul principal pentru care am decis să utilizez această librărie este unul evident, iar concluzia poate fi extrasă din fraza anterioară, deoarece datele sunt manipulate sub forma unor obiecte javascript, însă acesta nu este singurul motiv. Sequelize este capabilă să opereze cu baze de date de dimensiuni mari, iar printre beneficiile pe care le oferă se enumeră: capacitatea de a lucra cu tranzacții, oferă suport pentru utilizarea relațiilor între obiecte de tip 1:1, 1:n și n:n, definiția schemelor și sincronizarea lor, lucrul cu promisiuni, cicluri de viață, CLI (Command Line interface) și migrări [12].

Definirea schemelor presupune pentru fiecare tabelă să existe un fișier javascript în care să fie descris comportamentul și structura acesteia, astfel vor fi declarate numele, câmpurile și tipurile acestora, cât și tipul de relații folosit între tabele, toate acestea într-un format OOP. Marele avantaj însă este constituit din faptul că aceste scheme sunt capabile să fie sincronizate, ceea ce înseamnă că nu este necesar ca baza de date să fie creată în cod SQL, deoarece o singură comandă de sincronizare este responsabilă de generarea codului SQL pe baza modelelor scrise în javascript.

Noțiunea de promisiune în javascript presupune așteptarea unui rezultat care nu poate fi obținut instant. Se folosește spre exemplu în cererile către un server, când rezultatul poate apărea în secunda următoare sau în perioada următoare, astfel promisiunile au un rol foarte bine stabilit de a permite rularea codului asincron, fără a bloca astfel tot procesul, restul codului fiind rulat fără întrerupere, iar în momentul în care rezultatul așteptat de promisiune a apărut, acesta va fi procesat.

Ciclurile de viață, reprezintă stagiile prin care o aplicație trece din momentul în care este pornită, pe toată durata de funcționare. Astfel o aplicație poate avea mai multe cicluri: pornire, rulare, în lucru, în așteptare, în proces de oprire sau diverse astfel de stări. Aceste cicluri de viață sunt vitale interacțiunii cu aplicația, deoarece asupra ei pot fi aplicate diferite procese, în funcție de stadiul de rulare al acesteia.

Lucrul cu linia de comandă (CLI) este vital în viața oricărui administrator de sistem, faptul că Sequelize beneficiază de un astfel de serviciu este îmbucurător, mai ales pentru momentele în care sistemul nu răspunde în aplicația client și se dorește efectuarea unor operații precum restartul bazei de date sau eliminarea unor date care blochează rularea aplicației.

Migrarea presupune unul dintre cele mai importante aspecte când vine vorba de mutarea bazei de date de pe un server pe altul, cel mai întâlnit caz fiind în momentul în care se dorește migrarea aplicației din modul de dezvoltare în producție.

## Arhitectura

Arhitectura sistemului reprezintă modul în care codul a fost structurat în fișiere javascript și împarțită în mai multe directoare, dar totodată și modul în care codul a fost structurat din punct de vedere al funcțiilor ce trebuiesc îndeplinite. În rădăcina web-service-ului pot fi regăsite următoarele fișiere:

* **node\_modules** - se regăsesc modulele necesare pentru funcționarea serverului. Așa cum am prezentat anterior, node js este un server ce are implicit instalat NPM, locul de unde pot fi descărcate module puse la dispoziție de comunitate. În acest fișier sunt descărcate și ulterior instalate modulele. Este un folder creat implicit de NPM în momentul în care se utilizează o comandă de creare a serverului *npm init* sau de instalare a dependențelor din fișierul package.json;
* **server** - directorul în care se află implementarea propriu-zisă. În conținutul acestuia se află fișiere de configurare, lucru cu baza de date, implementarea pentru conectarea aplicațiilor client și alte configurări necesare rulării aplicației;
  + **config** - se regăsesc surse pentru configurarea web-service-ului
    - server-config.json - această filă are un format json, așa cum reiese și din extensie;
  + **database** - fișier ce conține surse pentru interacțiunea cu baza de date
    - **builder** - surse ce conțin metode de selecție, modificare sau ștergere în raport cu baza de date;
    - **config** - fișiere de configurare pentru conexiunea cu baza de date;
    - **handlers** - surse ce conțin metode care fac legătura dintre builder și rute;
    - **model** - surse în care se află schema tabelelor în format Sequelize;
  + **middlewares** - surse folosite pe post de middleware;
  + **routes** - surse pentru configurarea rutelor HTTP;
  + **server.js** - filă principală, ce va fi rulată de serverul node și cea care conține configurarea acestuia;
* **.babelrc** - configurarea modului babel. Babel CLI este un modul responsabil pentru conversia codului Javascript din ES6 în ES5. Această conversie este necesară pentru ca serverul node js să poată interpreta codul scris;
* **.gitignore** - fișier de configurare pentru sursele ce se doresc excluse din repository-ul de git;
* **package.json** - fișier de configurare în care se află prescurtări pentru comenzi node și dependențe ale modulelor necesare;
* **README.md** - fișier de citit înainte de instalarea serverului. Aici este prezentat modul în care serverul trebuie rulat, modul de instalare și tot ce trebuie știut înainte de instalare;

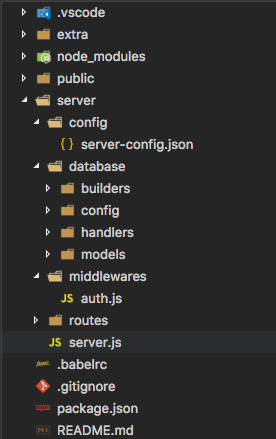


Figura 1.5 Structura arborescenă a serverului

În continuare voi prezenta aspecte ce țin de structura codului și descrierea componentelor ce se află în interiorul directorului **server**:

**config > server-config.json**

Am ales acest tip de fișier deoarece așa cum am explicat și în prezentarea javascript, este un format special creat pentru acest limbaj. Rolul acestui fișier este acela de a permite dezvoltarea sau execuția serverului pe mai multe computere, fără a fi necesară căutarea acestor setări în interiorul codului propriu-zis. Astfel aplicația este scalabilă și prin simpla adăugare a unui nou obiect js, poate fi utilizată și rulată pe un alt computer. Formatul pe care îl are acest fișier este următorul:

{  
 "environment":"local",  
 "local":{  
 "hostname":"localhost",  
 "port":8080,  
 "secret":"testkey"  
 },  
 "production":{  
 "hostname":"live",  
 "port":3000,  
 "secret":""  
 }  
}

Astfel proprietatea *environment* conține numele obiectului ce se dorește utilizat. În cazul de față folosesc ca environment obiectul local, ce are în componența lui numele hostului (hostname), portul pe care operează și o cheie secretă utilizată pentru criptarea între web service și o aplicație client.

**database**

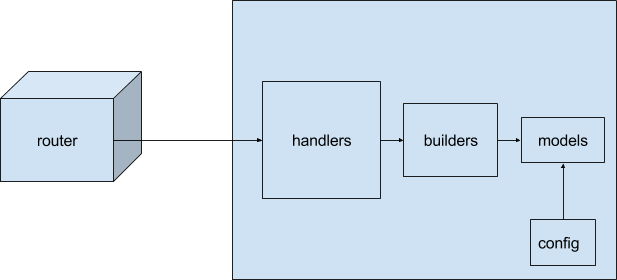


Figura 1.6 Structura blocurilor de cod în raport cu Sequelize

**database > models**

În conținutul directorului *models* se află 6 fișiere javascript, având un nume sugestiv, corespondent fiecărei tabele din baza de date (discipline.js, group.js, note.js, specialization.js, student.js și user.js) plus un al șaptelea fișier cu numele index.js.

Așa cum am explicat mai sus în descrierea librăriei Sequelize, aceasta operează cu ajutorul unor fișiere în care este descrisă arhitectura bazei de date sub forma unui obiect javascript ce definește numele tabelei și a componentelor acestora. Fiecare coloană reprezintă o proprietate a obiectului, care la rândul ei are mai multe proprietăți ce definesc tipul datei ce va fi stocat în acel câmp, în cazul în care este necesară definirea unor constrângeri precum chei primare, chei secundare, dimensiunile minime sau maxime ale câmpului, posibilitatea de a fii null sau nu, abilitatea de a fi incrementat automat sau nu (utilizată în special în cazul identificatorilor). De asemenea se va defini un alt obiect în care vor fi precizate relațiile pe care le are tabela curentă cu alte tabele existente în baza de date, cheile străine utilizate pentru legăturile dintre acestea și constrâgeri privind modul de actualizare sau ștergere a datelor.

Astfel în cadrul filei index.js se instanțiază o conexiune la baza de date PostgreSQL, prin crearea unui obiect de tip Sequelize, utilizând datele de conectare la baza de date aflate în fișierul config/database.json. În continuare se citește conținutul tuturor celorlalte file javascript din folderul curent, models, acestea reprezentând de fapt obiecte json. Ele vor fi asociate unui vector, cu ajutorul metodei *associate()* din clasa Sequelize. În final se va exporta un obiect ce are în componență obiectul Sequelize creat la început și modelele sequlize din cele 6 fișiere. Obiectul exportat este folosit în cadrul filei server.js, pentru sincronizarea bazei de date; însă mai multe detalii despre modul în care este structurat acesta vor fi prezentate ulterior.

În ceea ce privește modelele, fiecare din cele 6 fișiere reprezintă câte unul. Fiecare model este de fapt o funcție javascript ce are în antet 2 parametrii: *sequelize* - un obiect de tip sequelize ce va fii apelat din fișierul index.js și cel de-al doilea parametru *Datatypes* - obiect din clasa Sequelize ce ne va ajuta să alegem tipul datelor pentru fiecare coloană. În cadrul funcției se crează un obiect apelând funcția *define()* asupra obiectului sequelize din antet. Această funcție *define()* necesită minim 2 parametrii, primul constând în numele dorit pentru tabela curentă, iar celelalte începând cu al doilea, reprezentând obiecte JSON ce definesc structura și tipul datelor pentru tabela curentă. Am folosit doua obiecte JSON pentru a configura tabelele, astfel: primul dintre acestea având rolul de a defini coloanele tabelei, iar cel de-al doilea pentru a stabili relațiile tabelei curente cu celelalte. Fiecare proprietate a primului obiect reprezintă la randul ei un obiect, numele proprietății reprezentând de fapt numele coloanei în tabelă, iar în obiectul ce reprezintă valoarea vor fii definite proprietății precum:

* type - tipul datei, valorile cele mai des folosite fiind String, Integer, Dateonly. Acestea fiind apelate cu ajutorul celui de-al doilea parametru utilizat în funcția Datatypes (exemplu: DataTypes.String);
* primaryKey - ce poate avea o valoare booleana, în cazul în care aceasta este specificată și are valoarea true, coloana respectivă va avea funcția de cheie primară. Implicit această proprietate este tratată de Sequelize ca false, fără a trebui să fie definită explicit;
* autoincrement - de tip boolean, în cazul în care se setează true, valoarea coloanei curente va fi incremenetată automat. Am folosit acest tip pentru coloanele identificatorilor unici, pentru a mă asigura ca o coloană este cu siguranță unică;
* allowNull - de tip boolean, folosită în special în cazul în care nu se dorește ca valoarea curentă să fie null, astfel pentru acel caz va fi setată ca true;
* unique - de tip boolean, în cazul în care primește valoarea true, conținutul acestei celule va fi unic, iar informația ce se dorește introdusă sau actualizată va fii verificată în prealabil. Am folosit această constrângere în cazul studenților pe coloana registration\_number, care corespunde numărului matricol al fiecarui student;

Cel de-al doilea obiect de configurare care corespunde relației cu alte tabele se bazează pe același tip cheie - valoare. Din toate acele setări am folosit proprietatea *classMethods*, care are la randul ei ca valoare un obiect cu numele *associate,* însă de această dată vom folosi pe post de valoare o funcție care are un singur parametru de intrare *models*, acesta fiind de fapt un obiect ce conține toate modelele existente, pentru a putea stabili legături între tabele.  Pentru a stabili legături între tabela curentă și altele vom aplica asupra tabelei curente (care este de fapt un obiect) diferite funcții Sequelize, apelate cu două valori, primul parametru reprezentând obiectul tabelei dorite pentru a crea o relație, iar cel de-al doilea parametru fiind un obiect JSON pentru configurare. Printre proprietățile utilizate în cadrul proiectului se pot evidenția:

* foreignKey - această proprietate reprezintă numele chei primare din tabela curentă;
* as - cu ajutorul acestei proprietăți relația de legatură dintre cele doua tabele poate primi un alias, care în dezvoltare va reprezenta de fapt numele obiectului returnat după o operație de join;
* allowNull - în cazul în care această proprietate este setată false, nu se acceptă ca una dintre valorile din cele doua tabele să fie null, returnând astfel o eroare;
* onDelete - cu valoare “Cascade” permite ștergerea în cascadă a entităților. În cazul în care o cheie primară este ștearsă, atunci toate cheile străine din tabela 2 vor fi șterse;
* timestamp - de tip boolean. Implicit sequelize creează 2 coloane de log, pentru a înregistra data la care a fost inserată și data ultimei actualizări pentru interogări. Prin setarea acestei proprietăți ca *false*, cele doua coloane vor fi omise;
* through - reprezintă numele tabelei secundare ce va fi utilizată pentru a crea o relație de tipul n:n. Astfel o nouă tabelă va fi creată pentru a ține legătura între cheile străine;
* otherKey - folosită împreună cu proprietatea de mai sus, aceasta specifică numele coloanei din cealaltă tabelă;

Am precizat mai sus, că lista proprietăților este folosită într-un obiect, ce reprezintă cel de-al doilea parametrul al unor funcții ce stabilesc tipul relației dintre două tabele. Printre funcțiile Sequelize utilizate am folosit:

* belongsToMany - pentru a realiza relația n:n. Aceasta are nevoie de numele tabelei ce va fi creată pentru a ține legătura între cheile celor două tabele, numele coloanei cheii din tabela curentă și numele cheii din tabela 2, ce se doresc utilizate pentru a crea relația;
* belongsTo - utilizată pentru a realiza  relația de tip 1:1;

În cazul în care nu se dorește stabilirea unei relații între doua tabele, cel de-al doilea obiect de configurare poate fi omis.

**database > builders**

Cel de-al doilea nivel pentru utilizarea acestui webservice, după definirea arhitecturii modelelor este acela de a defini metodele cu ajutorul cărora se va realiza lucrul cu baza de date, acestea fiind regăsite în folderul builders. În componența acestui folder se află 6 fișiere javascript, cu un nume sugestiv, corespondent fiecărei tabele din baza de date. Fiecare fișier javascript reprezintă o clasă în componența căreia se află mai multe metode utilizate pentru a extrage, insera, actualiza sau șterge date, pe scurt pentru a opera asupra modelelor create anterior, tocmai de accea în antetul fiecărei clase vom importa un obiect ce va conține toate modelele Sequelize create la pasul anterior, iar pentru tabela curentă vom crea o variabilă globală în care vom stoca obiectul corespunzător, deoarece pe baza lui vom putea apela diferite metode Sequelize.

În funcție de cerințele care s-au impus pentru fiecare tabelă în parte am creat diferite metode, care să îmi permită o manipulare rapidă a datelor, însă toate cele 6 fișiere conțin metode esențiale pentru a opera cu baza de date și au nume sugestive. Acestea returnează un obiect JSON prin apelul unor funcții Sequelize asupra modelului curent. Fiecare funcție Sequelize utilizată are ca parametru de intrare un obiect ce permite configurarea acestuia pentru a returna doar datele dorite. Deoarece există șansa ca serverul bazei de date să nu poată răspunde instant, în urma mai multor cereri înregistrate, așa cum am enunțat și în descrierea Sequelize, acesta pune la dispoziție lucrul cu promisiuni. Lucrul cu acestea fiind foarte simplu, după denumirea funcției se folosește construcția Javascript .then( function(result) { return result }), pentru a permite returnarea rezultatului în momentul în care acesta este disponibil. Printre metodele regăsite în toate dintre cele 6 fișiere se pot evidenția:

* findAll() - metoda ce returnează toate înregistrările din baza de date. Este creată având ca suport funcția Sequelize cu aceiași denumire *findAll().* Înobiectul de configurare introdus ca parametru al acestei funcții pot fii definite proprietății precum:
  + include: un array de obiecte ce permit prezentarea tabelelor care au o relație cu tabela curentă. Astfel în obiectele din array pot fi prezente proprietăți precum:
    - model: obiectul modelului creat în etapa precedentă, aflat în relație cu modelul curent;
    - attributes: un array în care vor fi introduse numele coloanelor ce se doresc extrase din modelul 2;
    - as: poate fi setat ca alias pentru numele obiectului ce va fi returnat pentru relația dintre cele doua modele;
    - where: este la rândul sau un obiect în care pot fi prezentate condiții exact ca în sintaxa sql;
  + order: reprezintă un vector, de vectori. Fiecare vector component conține numele coloanei și criteriul de ordonare ascendent sau descendent;
  + where: condiție ce trebuie îndeplinită pentru a fi returnată;
* findById(id) - pentru această metodă se utilizează funcția Sequelize findOne() ce acceptă la fel ca findAll un obiect ca parametru în care pot fii folosite aceleași proprietăți;
* add(data) - utilizează metoda Sequelize cu numele .create() ce acceptă ca parametru de intrare un obiect, de această dată, pe langă cel de configurare, încă unul ce conține datele ce se doresc introduse: numele corespunzător coloanei și valoarea ce se dorește introdusă. De asemenea pot fi utilizate din proprietățile menționate la metoda .findAll();
* update(data) - se folosește de metoda cu același nume, .update() cu aceleași caracteristici precum funcția add();
* delete(id) - utilizează metoda Sequelize .destroy(), care are nevoie de cel puțin clauza where; dacă aceasta nu este precizată, vor fi șterse toate valorile din tabela curentă;

**database > handlers**

Ultimul nivel de prelucrare al webservice-ului în raport cu baza de date se află în folderul handlers. Așa cum îi sugerează și numele (handlers = manipulant), în cadrul acestui folder se află file javascript cu rol de antemergător între metodele create pe baza modelelor și router, despre care voi vorbi într-un subcapitol următor. Utilizarea acestor file putea fi evitată, codul scris aici putând fi scris în filele corespunzatoare din folderul builder, însă am preferat crearea lor, deoarece este mult mai ușor de întreținut un sistem, în momentul în care metodele sunt scurte, erorile apărute pot fi depistate mai ușor și chiar updatarea sistemului se poate face mai ușor în viitor.

Sunt prezente 7 file javascript, dintre care 6 corespunzătoare celor 6 modele create, una pentru modulul de autentificare pe site și o a opta filă json în care sunt salvate mesaje și coduri de eroare sau succes, generate de webservice. Fiecare filă javascript este de fapt o clasă, în componența căreia se află mai multe metode. Fiecare metodă este corespondentă unei metode din clasa cu același nume din folderul builder, însă rolul acestora este de a procesa operația serverului în funcție de ruta accesată. Toate metodele au în antet cb, care de fapt este o funcție callback ce are 2 parametrii. O functie callback este o funcție declarată ca parametru în antetul altei funcții și este apelată la un moment dat în interiorul celei din urmă. Primul parametru al funcției callback este null în cazul în care cererea poate fi procesată fără eroare, și primește eroarea generată de server în momentul în care aceasta apare și trebuie transmisă către client. Cel de-al doilea parametru este corpul mesajului ce va fii transmis către client, acest parametru este de fapt un obiect json ce conține de obicei 3 proprietăți: statusCode - codul returnat pentru operația executată, message - mesajul corespunzător codului returnat și data - reprezintă valorile returnate din baza de date, primele două proprietăți sunt extrase din componența fișierului message.json. Fiecare metodă returnează rezultate obținute prin apelul funcției corespunzătoare din clasa din builder într-un promise, iar mai departe router-ul trimite răspunsul către client. În cazul în care o eroare apare în returnarea unui rezultat, este tratat de o funcție .catch() în componența căreia se returnează funcția callback cb() cu parametrul de eroare setat cu eroarea generată de server.

Fila auth.js a fost creată utilizând tot modelul User, însă rolul acestuia este de a putea accesa aceste metode fără a avea accesul restricționat sau a lăsa vre-o breșă în sistem, deoarece anumite rute nu pot fi accesate decât cu ajutorul unui cont. Pentru crearea unui cont sau pentru logare, apare problema că utilizatorul nu este conectat. În cazul în care aș utiliza aceleași metode ca în handler-ul pentru User, aș fi nevoit să las deschise o serie de portițe, care nu ar fi tocmai sigure. Din acest motiv, prin aceste metode special create, serverul accesează doar o anumită parte a sistemului. Funcția de înregistrare, signUp() are doi parametrii de intrare, primul, reqBody conține datele primite de la client: username, password, level ( nivel de administrare) și registration\_number (număr matricol, în cazul în care level == 3, adică este student), toate aceste date sunt salvate în baza de date, iar cel de-al doilea este funcția de callback. Funcția de login are de asemenea doi parametrii, la fel și signUp(), însă primul conține username-ul și parola necriptată, provenită de la client. Verific pentru început dacă numele de utilizator există, în cazul în care nu există returnez un mesaj corespunzător, dacă acesta există continui procesul de logare. Compar cu ajutorul unui modul Node JS, bcrypt, parola introdusă de utilizator cu cea din baza de date, iar dacă acestea corespund generez un nou token și transmit un mesaj corespunzător. În cazul în care cele doua parole nu se potrivesc trimit un mesaj pentru parolă greșită. Despre sistemul de criptare și token voi detalia mai multe într-un subcapitol special.

Fila message.json, așa cum se prezintă încă din extensie conține un obiect json, în componența căruia se află mai multe obiecte. Numele fiecărei proprietăți este sugestiv pentru fiecare mesaj în parte, iar fiecare dintre aceste proprietăți au ca valoare un alt obiect: statusCode - codul setat pentru fiecare mesaj în parte, message - un mesaj descriptiv, iar în cazul în care o eroare există, adaug și error cu valoarea “error”. Aceste mesaje pot fi configurate foarte ușor și există posibilitatea ca acestea să fie setate pe mai multe limbi, fără a fi necesar un proces anevoios.

**database>config**

În cadrul acestui director se află o filă database.json, în componența căruia se află setările necesare pentru conectarea librăriei Sequelize cu serverul pe care se află baza de date PostgreSQL. Acesta filă conține mai multe obiecte, în funcție de serverul ce se dorește conectat. Fiecare obiect de configurare conține:

* user - numele utilizatorului prin care se face conectarea la baza de date;
* password - parola corespondentă utilizatorului;
* database - numele bazei de date la care se dorește conectarea;
* host - ip-ul sau dns-ul serverului pe care se află setată baza de date;
* port - portul pe care este disponbilă baza de date;
* dialect - tipul bazei de date utilizate, în cazul de față PostgreSQL;

Am ales această structură pentru relația web-service-ului cu baza de date, deoarece consider ca orice sistem trebuie să fie scalabil, iar mentenanța și îmbunătățirea unei aplicații împarțită în mai multe clase, fiecare cu metode specializate pentru un anumit obiectiv și cu o mărime cât mai mică se poate realiza mult mai ușor.

## Arhitectura bazei de date

Baza de date este compusă din 6 tabele principale: Students, Users, Specializations, Groups, Disciplines, Notes. A fost concepută astfel încât să satisfacă nevoile unei universități/facultăți cu mai mulți studenți.

Un cont de utilizator, stocat în tabela Users, reprezintă un utilizator al aplicației, acesta poate fi administrator, având ca valoare pe coloana level: 1 - admin, 2 - profesor. În cazul în care utilizatorul are level-ul 3, acesta este un student, iar câmpul registration\_number va fi completat. Aceasta este o cheie străină în tabela Users și face posibilă conectarea acestei tabele cu tabela Students. Tabela Students conține date despre studenți și în raport cu toate celelalte tabele, crează identitatea unui student în cadrul Universității/Facultății. Un student poate face parte dintr-o grupă, într-un an și poate aparține uneia sau mai multor specializări. Legătura între tabela Groups și Students se realizează prin cheia primară a tabelei Students, student\_id, iar legatura dintre Groups și Specializations se realizează prin cheia primară a tabelei Specializations, specialization\_id.

Un student poate avea una sau mai multe note, existând o relație de 1:n între tabela Students și tabela Notes, relație realizată prin intermediul chei primare din tabela Students, student\_id. Acea notă aparține de asemenea unei specializări, relație stabilită între Specializations și Note prin cheia primară a tabelei Specializations, specialization\_id. și de asemenea o notă aparține și unei discipline, relația fiind stabilită prin cheia primară a tabelei Disciplines, discipline\_id.

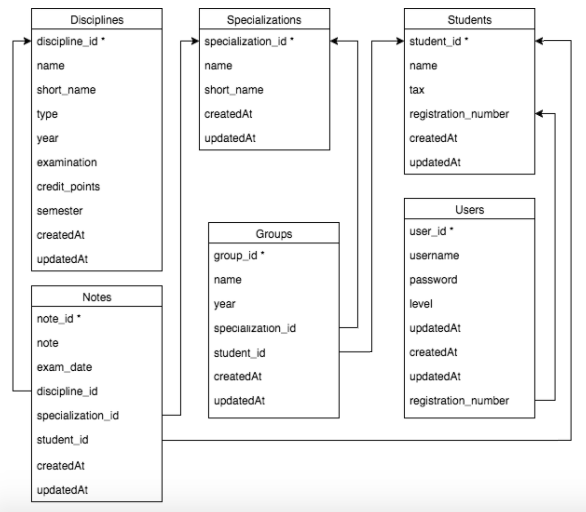


Figura 1.7 Structura bazei de date

## Securitate

Securitatea unui sistem reprezintă cu siguranță unul dintre cele mai importante aspecte ce trebuiesc luate în calcul în momentul în care se realizează proiectarea acestuia, deoarece o breșă de securitate poate pune la pământ întreg sistemul și chiar mai rău de atât poate compromite datele a mii de studenți în cazul de față. Astfel am încercat să combat aceste breșe prin implementarea unor tehnici care permit sistemului să nu ofere mai mult decât este necesar la un moment de timp.

În cadrul sistemului am proiectat o metodă de siguranță, ce privește accesul la date în funcție de nivelul de permisiuni pe care un utilizator le are și se bazează pe necesitatea unui cont pentru a putea accesa datele prin intermediul webservice-ului. Metoda prin care am integrat această funcționalitate se bazează pe un principiu pus la dispoziție de framework-ul node js, express js, utilizat și poartă denumirea de middleware. Această funcționalitate face parte din modulul router, ce va fi prezentat în subcapitolul următor, însă voi face o scurtă prezentare a modului în care îl utilizez legat de securitate. Un middleware este o funcție executată în timpul unei cereri către server, astfel în momentul în care aceasta este executată se rulează codul aflat în componența sa, iar totul decurge fără întreruperi. Atunci router-ul își va continua sarcina, altfel acesta va trimite un răspunsul oferit în interiorul middleware-ului către client. În cadrul folderului middleware din radacina cu numele server, poate fi identificată o filă javascript ce poartă numele auth.js. În cadrul acestei file este definită o clasă cu două metode, ce vor fi folosite ca middleware pentru diferite rute definite în modulul Router, însă înainte de a prezenta cele doua clase, voi oferi câteva informații despre un modul node js utilizat în cadrul celor doua metode, jsonwebtoken.

JsonWebToken(JWT) este un standard liber ce definește o metodă independentă și compactă de a transmite o informație între două entități ca un obiect JSON [13]. Acesta poate fi semnat utilizând o cheie publică sau privată [13]. Necesită să fie compactă, deoarece se transmite în header-ul cererilor HTTP și este independentă pe motiv că payload-ul conține toate datele necesare despre utilizator, fără a mai fi nevoie de o cerere suplimentară către server [13]. Așa cum am prezentat mai sus JWT este compus din trei părți separate prin punct: header, payload și Signature [13]. Headerul este de obicei constituit din două proprietăți type care de obicei sunt jwt și alg, care este de fapt numele algoritmului utilizat pentru criptare [13]. Payload-ul este a doua componentă ce conține în general date importante ce țin de configurare precum: iss - problema, exp - dată de expirare, sub - subiect, aud - audiență, name - nume sau alte valori importante pentru aplicație ce se doresc transferate [13]. Iar cea de-a treia componentă, Signature (semnatură), pentru a realiza asta este nevoie de criptarea header-ului, a payload-ului a cheii secrete cu algoritmul specificat în header [13].

Modul în care modulul JWT este utilizat în cadrul sistemului pentru generarea unui token este următorul:

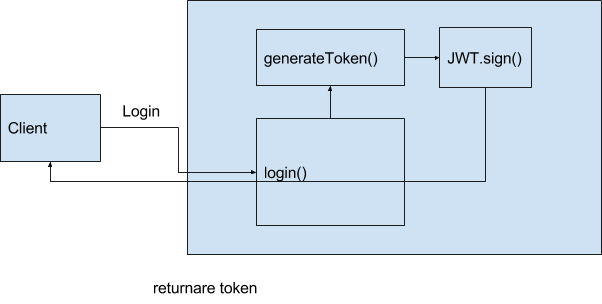


Figura 1.8 JWT semnare token

Clientul inițiaza o cerere de login către server, acesta procesează cererea, iar dacă toate condițiile de logare au fost îndeplinite, se apelează metoda *generateToken()* din clasa Auth(), din folderul handlers. Aceasta metodă are doi parametrii de intrare, username-ul și level-ul utilizatorului și returnează un token. Acest token se va genera cu ajutorul unei metode din clasa jwt, care poartă numele sign(). În funcția sign vor fi introduși 3 parametrii: *payload*, primul parametru este constituit dintr-un obiect json în care va fi utilizată proprietatea sub(subiect) în care se va introduce username-ul, al doilea parametru este cheia de criptare, stocată în fișierul de configurare *server-config.json*, iar cel de-al treilea parametru este constituit de obiectul pentru alte opțiuni, aici vom seta durata de viață a unui token de 1 zi și în header vom trimite și nivelul utilizatorului. Vom proceda în acest fel pentru a nu face un nou request către baza de date de fiecare dată când se trimite o cerere. În momentul curent având la dispoziție în acest token tot ce este nevoie pentru a putea face un request către server. Aceasta este metodă de semnare a token-ului, iar în continuare vor fii prezentate aspecte ce țin de verificarea acestuia și raspunsul server-ului.

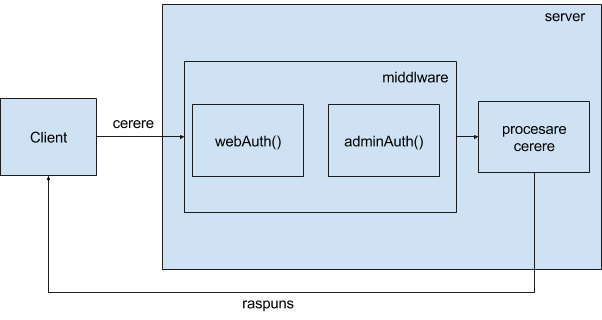


Figura 1.9 Verificare token

De fiecare dată când o nouă cerere este inițiată către web-service, aceasta va fi procesată de modulul Router și va alege o rută, iar dacă rutei respective îi este atașată una sau mai multe metode pe post de middleware, acea metodă va fii verificată înaintea procesării cererii și returnării oricărui răspuns către client. Metoda webAuth(req, res, cb) este folosită pentru a verifica autenticitatea unui client pentru ca cererea inițiată de acesta să fie procesată. Este definită cu 3 parametrii de intrare, primul reprezintă obiectul în care se află toate datele venite ca cerere, res reprezintă datele ce vor fi trimise ca răspuns, iar cb reprezintă o funcție callback. În corpul acesteia este utilizat un block try - catch. Se încearcă extragerea token-ului din req.header, aflat sub proprietatea “x-auth-token”, dacă acesta există va fi găsit sub forma ‘Barear <token>’, motiv pentru care se va face o procesare asupra stringului și va fi extras doar tokenul. Tokenul extras va fi verificat cu ajutorul metodei verify(), din clasa jwt și a chei de securitate utilizate la generarea sa. Dacă metoda de verificare returnează o eroare, către client va fi trimis un răspuns cu statusul 200, atașat obiectului res, pentru a-l informa asupra faptului ca token-ul nu poate fi utilizat și o eroare aferentă, pentru ca acesta să încerce să se reconecteze (logheze). În caz contrar, se apelează funcția callback, fără nici-un parametru pentru a permite serverului să își reia activitatea și a procesa efectiv cererea, și a trimite ulterior răspunsul așteptat. În cazul în care pe parcursul încercării de extragere a tokenului și de verificare a acestuia apar erori, precum valoarea setată în header pentru token nu respectă formatul așteptat, acestea vor fi procesate de blocul catch, care va trimite un răspuns cu statusul 401, pentru a informa utilizatorul ca există probleme de autentificare și este necesară relogarea. Cea de-a doua metodă din clasa middleware, adminAuth(req, res, cb) este creată cu scopul de a verifica dacă un utilizator, logat, cu un token valid are acces pentru cererea inițiată. Deoarece vorbim de un sistem pe mai multe nivele de administrare, această metodă se poate defini ușor și poate fi utilizată mai târziu în cazul în care se dorește actualizarea sistemului. Metoda este construită sub același format ca și cea de dinainte și are ca obiectiv extragerea token-ului din header și decriptarea lui utilizând metoda decode() din clasa jwt, se va compara level-ul stocat în token și dacă acesta este mai mic decât 3, ceea ce înseamnă ca este administrator sau profesor, atunci funcția callback va fi apelată pentru a continua procesarea cererii inițiate de client, în caz contrar va fi returnat un răspuns cu status 403, pentru a informa utilizatorul că nu are acces pentru acea rută. În cazul în care apar erori, acestea vor fi returnate către client în blocul catch atașat raspunsului cu statusul 401.

Astfel aceste metode integrate în cadrul sistemului vor permite o bună funcționare a acestuia fără a pune în percol datele existente sau a compromite informații vitale despre un student, unor persoane neavizate fără acordul acestuia.

## Conectarea la webservice

Modulul Router utilizat în cadrul proiectului face parte din miezul framework-ului express js. Acesta este responsabil pentru gestionarea cererilor venite de la client și selecția metodelor ce trebuiesc apelate pentru a oferi răspunsul dorit de utilizatori. În cadrul acestui sistem avem definite mai multe rute, fiecare rută reprezentând o clasă ce conține la rândul ei mai multe metode. Acest router va fi instanțiat în fila generală server.js, aflată în rădăcina, iar toate fișierele de configurare se află în folderul routes. Pentru o cale mai simplă de mentenanță și dezvoltare am creat o clasă Router și am salvat fișierul cu numele index.router.js. În constructorul acestei clase se instanțiază un nou obiect, din clasa auth pentru a putea folosi metodele din middleware. De asemenea în componența acestei clase mai exista o metodă, denumită sugestiv configure(app), cu un parametru app, obiect creat cu ajutorul framework-ului express js și cel care deține toate configurările necesare pentru rularea serverului. În corpul acestei clase sunt instanțiate mai multe obiecte, pentru a defini rutele generale, apoi cu ajutorul metodei app.use(“numele\_rutei”, obiectul\_instanțiat), se definesc rutele generale, practic se crează câte o rută pentru fișierele existente în acest folder routes: admin.js, auth.api.js, client.api.js, discipline.api.js, group.api.js, note.api.js, specialization.api.js, student.api.js, user.api.js. În orice moment în care una din rute se dorește schimbată poate fi înlocuită foarte simplu aici.

Fiecare fișier javascript din acest folder, reprezintă o clasă în constructorul cărora vor fi definite mai multe rute, aceste rute vor reprezentă nivelul doi de selecție. Se crează un obiect router cu ajutorul metodei Router din clasa express. Pentru acest obiect router se vor defini metodele și numele nivelului 2. Metodele HTTP utilizate de acest sistem sunt GET, POST, PUT, DELETE, utilizate după cum urmează: extragere date, introducere date, actualizare date, ștergere date. Fiecare din aceste metode are un corespondent, o metodă cu același nume în clasa router, având următorii parametrii:

* primul parametru este un string și reprezintă numele sub care va fi apelată acea metodă;
* următorii parametrii reprezintă funcțiile utilizate pe post de middleware, în funcție de specificul rutei. aici vor fi apelate una sau chiar amandouă metodele definite în middleware-ul prezentat în subcapitolul de securitate;
* ultimul parametru reprezintă o altă funcție cu doi parametrii req și res în conținutul căreia se vor apela metodele corespunzătoare din clasele Handlers. În cazul în care apar erori, asupra obiectului res se va atașa statusul 500 (eroare server) cât și enunțul erorii;

## Rularea webservice-ului

În finalul acestui capitol am lăsat partea cea mai importantă, dar totodată constrâns de explicațiile necesare oferite pe parcurs. Toate clasele și metodele definite pe parcursul implementării sunt apelate într-un singur loc, în fila javascript server.js. În componența acestei file se importă modulele necesare și fișierul de configurare server-config.json și se crează obiectul app cu ajutorul funcției express() din modulul express. Importul obiectului de configurare reprezintă de fapt stocarea obiectului json din fila server-config.json într-un obiect, însă deoarece aceasta este de fapt un obiect de obiecte, iar obiectul ce se dorește utilizat este setat în proprietatea “environment” este suficient să apelăm această proprietate, iar singurul loc în care rămâne de modificat în cazul în care se dorește modificarea obiectului de configurare este în proprietatea environment din fila json.

Astfel în acest moment serverul a fost creat, pentru definirea oricărei configurări se va aplica metoda use() asupra obiectului app (exemplu app.use() ). Am activat ca modalitate de returnare a datelor formatul json și de asemenea am activat codificarea url-urilor, pentru a asigura funcționarea webservice-ului în cazul în care se introduce din greșeală un caracter nedorit în numele unui url. De asemenea am oferit suport CORS (Cross Origin Resource Sharing), procedeul prin care se permite accesul la anumite resurse precum fonturi, imagini de pe un alt domeniu decât cel hostat [15]. Următorul pas a fost crearea unui obiect pentru utilizarea router-ului, prin instanțierea sa din clasa special concepută cu acest scop în fila routes/index.router.js. Atașarea router-ului la aplicația app am realizat-o cu ajutorul metodei oferite de clasa Router, configure(app). În funcția initDB() am citit din nou într-un obiect toate modelele Sequelize utilizate și asupra lor am aplicat o funcție oferită de modulul precizat, .sync(), care permite sincronizarea modelelor în baza de date, generând astfel codul sql pentru crearea datelor. Apelul acestei funcții se face doar în momentul în care se dorește inițierea bazei de date. Ultimul pas pentru rularea serverului constă în utilizarea unei metode a modulului express js, listen(port,host,function() ), apelată cu 3 parametrii, primul reprezintă portul, extras din fișierul de configurare inclus la începutul fișierului, al doilea reprezintă adresa curentă a hostului, iar cel de-al treilea parametrul, funcția, am folosit-o pe post de log pentru a oferi un mesaj administratorului de sistem, informând-ul asupra rulării corecte a webservice-ului la adresa prezentă în mesaj.

## Concluzii

În închierea acestui capitol aș dori să menționez faptul ca am încercat să tratez cele mai importante aspecte și să prezint detalii despre conceptul de funcționare și necesitatea unui web-service în realizarea sistemului.

Informațiile prezentate în acest capitol au rezultat în urma unor conspecte și înainte de realizarea acestui proiect am avut în vedere principiile menționate în cartea Clean Code [15], care mi-a oferit o perspectivă despre ce ar trebui și nu ar trebui să conțină o aplicație, modul în care să structurez codul astfel încât dezvoltarea și mentenanța să poată fi efectuate cu succes și eliminarea pe cât de mult a posibilelor probleme.

# Client WEB

## Introducere

Acest capitol reprezintă cel de-al doilea subsistem, componentă a proiectului final și constituie partea care va interacționa cu utilizatorii, mai precis interfața web și modul în care sistemul operează. Acest sistem a fost realizat utilizând Angular 2 cu Typescript, tehnologii ce vor fii prezentate în subcapitolele următoare.

Așa cum am prezentat anterior, în capitolul 1, un sistem client-server are marele avantaj de a permite conectarea mai multor clienți la același server, lucru ce oferă diversitate în dorința utilizatorilor de a utiliza aplicații într-un anumit format, web, mobile sau desktop. Decizia de a construi un client web nu a fost o alegere dificilă, fiind bazată pe o analiză în rândul colegilor și studenților din universitate, pe care i-am interogat. S-au evidențiat în mod decisiv, studenții care au ales o aplicație web pentru afișarea notelor, iar printre motivele invocate se pot remarca aspecte precum: lejeritatea accesării unui site web, deoarece indiferent de platforma utilizată există un browser la îndemana oricui; eliberarea unui spațiu de memorie ce ar trebui alocat în telefon sau computer, de către fiecare student pentru a putea accesa notele; posibilitatea de mentenanță și îmbunătățire a sistemului cu o lejeritate sporită, actualizarea putând fii realizată într-un singur loc, pe serverul de client, în defavoarea implementării unei logici care să permită actualizarea aplicațiilor.

Modul în care am ales ca tehnologie web principală, Angular 2, a avut la baza aspecte evidențiate și în capitolul 1 precum:

* costurile de achiziție, Angular JS fiind o librărie open-source;
* comunitatea dezvoltatorilor;
* maturitatea proiectului, care este la versiunea 2 și care s-a bucurat de un real succes în versiunea 1, în rândul dezvoltatorilor;
* capacitatea de a putea interpreta formatul datelor, JSON, oferit de web-service-ul creat;
* dorința personală de a utiliza o tehnologie actuala;

În rândurile ce urmează vor fi prezentate aspecte teoretice, legate de limbajele de programare și modulele utilizate, conceptul de funcționare al aplicației client, interacțiunea cu serverul, aspecte de securitate ce au fost implementate, proiectarea modului în care elementele vor fi afișate în pagină, conectarea utilizatorilor și nivelele de permisiune, cât și arhitectura sistemului din punct de vedere al organizării codului, dar și al organizării fizice a folderelor.

## Angular 2 - Noțiuni Teoretice

Angular este un framework javascript open-source, lansat de Google și menținut împreună cu o comunitate de dezvoltatori [16]. A fost conceput pentru a permite crearea aplicațiilor client, web sau hybrid (aplicații mobile utilizând tehnologii web, compilate pentru a putea fi interpretate de sistemele de operare) și reprezintă o mare atracție pentru dezvoltatori, datorită arhitecturilor MVC (model-view-controller) și MVVM(model-view-viewmodel) [16]. Într-un studiu realizat în octombrie 2016, 12000 de website-uri din 1000000 analizate sunt bazate pe aceasta tehnologie și de asemenea este pe locul 6 între depozitele de proiecte stocate pe Github [16].

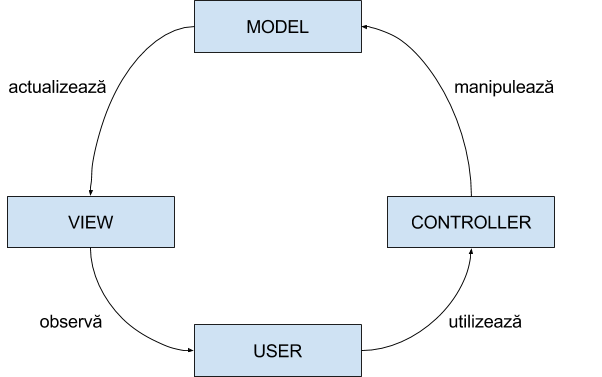


Figura 2.1 Concept Model View Controller

Conceptul de funcționare se bazează pe definirea unor atribute html proprii [16]. În prima fază arborele DOM (Document Object Model) este parcurs și aceste atribute sunt identificate [16]. Sunt interpretate apoi de angular ca directive și sunt stocate în variabile, cărora le sunt atribuite valori în urma unor prelucrari javascript sau direct din obiecte JSON [16].

Printre beneficiile pe care le oferă Angular 2, spre deosebire de alte framework-uri javascript se remarcă:

* face din HTML un limbaj mult mai expresiv, deoarece permite utilizarea unor instrucțiuni (if, loop) și a unor funcții predefinite (numărare elemente, mărirea textului, micșorarea textului, formatarea textului în diferite forme și altele), utilizarea variabilelor;
* data binding - permite actualizarea informațiilor de pe interfața grafică în timp real, odată ce datele sunt modificate în urma unor prelucrări sau sunt disponibile de la server, fără a fi necesară reîncarcarea paginii;
* modularitate - permite utilizarea unor clase javascript pe post de blocuri, astfel un bloc creat poate fi reutilizat în mai multe locuri fără a fi necesară modificarea acestuia;
* suport inclus pentru comunicarea cu web-service-urile, având incluse metode de comunicații HTTP (POST, GET, PUT, OPTION, DELETE);
* viteză considerabilă în raport cu web-service-ul sau încarcarea informațiilor în pagină;
* este o librărie modernă, ceea ce permite utilizarea noilor beneficii pe care le oferă javascript, utilizarea ES6, lucrul cu OOP, clase, decoratori și suport pentru noile browsere;
* îmbunătățirea productivității, față de Angular 1;

O aplicație realizată în Angular 2 este compusă din mai multe componente. O componentă are ca principal scop definirea unei funcții. Datele sunt extrase de la web-service și eventual procesate, dacă este cazul, în așa numitele Servicii, care sunt apelate ulterior de componente. Arhitectura unei aplicații în Angular 2 este reprezentată în figura următoarea:

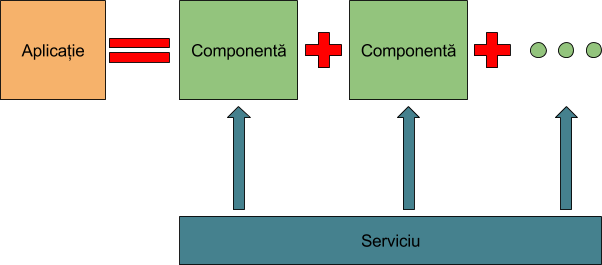


Figura 2.2 Arhitectura Unei Aplicații cu Angular 2

Dupa cum se poate observa în figura anterioară, rolul unei componente este esențial pentru o aplicație, astfel voi detalia atribute ce alcătuiesc o componentă. O componentă este alcatuită dintr-un template HTML ce constituie structura paginii, o clasă javascript ce conține proprietăți și metode, capabile să definească funcționalitatea elementelor din cadrul template-ului și de metadate ce conțin informații suplimentare despre componentă, care informează framework-ul că aceasta este o componentă Angular.

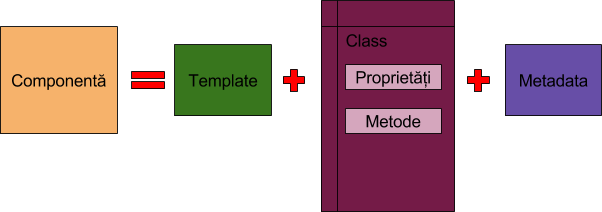


Figura 2.3 Arhitectura Unei Componente Angular 2

Faptul ca o componentă este structurată în acest mod, are însă și o bază practică, deoarece astfel există o interacțiune solidă între utilizator și date, modificările propagându-se în ambele sensuri, atât de la obiecte ( clasa componentei ) către utilzator, cât și de la utilizator către clasă. Acest procedeu poartă denumirea de data binding și este inclus în clasa ngModel. Pentru a face o referire către o variabilă vom folosi atributul HTML ngModel urmat de egal și între ghilimele numele variabilei din clasa curentă. Așa cum am prezentat anterior interacțiunea se poate face de la clasă către model și în acest caz vom utiliza paranteze drepte pentru a cuprinde atributul ngModel, iar pentru a face posibilă interacțiunea în sens invers de la model către clasă, vom încapsula ngModel utilizând paranteze rotunde. În cazul în care se dorește o legatură în sens dublu se va utiliza următorul format [(ngModel)] = ’’numevariabila’’.

În figurile de mai sus, ce prezintă arhitectura unei aplicații în Angular 2 am prezentat concepte despre componenta unei aplicații, însă toate aceste componente și servicii sunt interconectate în module. O aplicație Angular 2 are cel puțin un modul, modulul principal sau root module. Toate modulele secundare trebuiesc declarate și în modulul principal pentru ca Angular să știe că acestea vor fi folosite în cadrul aplicației.

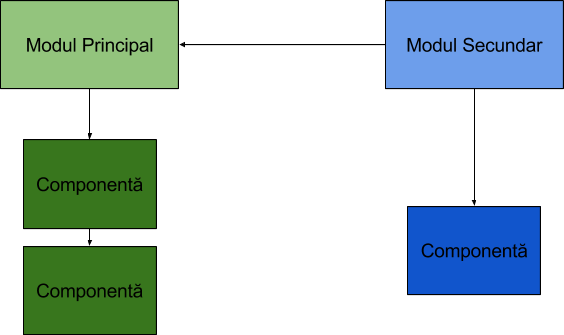


Figura 2.4 Arhitectura Modulelor în Angular 2

Codul poate fi scris în Angular 2 în doua moduri:

* în modul clasic, utilizând nativ limbajul Javascript;
* al doilea mod sugerat de Google, utilizând limbajul Typescript;

Typscript este un limbaj de programare open-source, dezvoltat și întreținut de Microsoft [17]. Este un superset de sintaxe stricte javascript, ceea ce înseamnă că fiecare variabilă, clasă sau metodă necesită să îi fie declarat tipul, acest lucru reprezintă o diferență față de modul clasic de a scrie cod în Javascript, deoarece javascript este interpretat și nu compilat de către browser [17]. Permite definirea variabilelor sub anumite tipuri, iar ceea ce este cu adevărat o noutate, față de limbajele tradiționale este tipul *any*, ce poate fi utilizat atunci când nu se cunoaște tipul datelor ce vor fi stocate într-o variabilă [17]. Typescript a fost gândit ca un limbaj de programare bazat pe obiecte, proiectat pentru dezvoltarea de aplicații mari, atât pentru client cât și pentru server, iar nucleul este scris chiar în Typescript [17]. Se transcompilează în fișiere Javascript și permite includerea fișierelor în antetul filelor, precum includerea headerelor în C-ul clasic sau a librăriilor java, utilizând extensia fișierelor .ts [17].

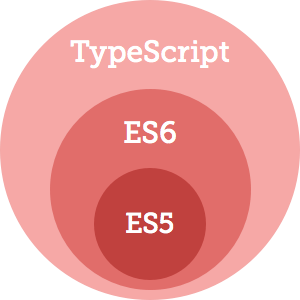


Figura 2.5 Încapsularea ES5 și ES6 în TypeScript

Sursa: [24]

## Arhitectura

Din punct de vedere arhitectural aplicația client este structurată modular, respectând în același timp arhitectura la nivel de concept, modul în care blocurile (clasele) sunt construite cât și cea la nivel fizic, în care folderele sunt organizate. Conținutul aplicației client poate fi regăsit în folderul *public* din rădăcina proiectului. Primul nivel de fișiere conține:

* directorul *app* în care este implementată întreaga funcționalitate a aplicației;
* directorul *config* în care pot fi găsite fișiere de configurare ale aplicației și în care pot fi introduse în continuare pentru o ulterioară dezvoltare a aplicației. Este de preferat conceperea aplicației cu aceste aspecte luate în calcul de la început, deoarece permite o mai bună administrare a aplicației ca fișierele să fie într-un singur loc și nu dezorganizate în mai multe fișiere, pentru ca o problemă precum schimbarea webservice-ului pe un alt computer să devină o întreaga aventură;
* directorul *node\_modules*, prezent și în cadrul aplicației client, așa cum am explicat în capitolul anterior, acest folder conține modulele necesare bunei funcționări a sistemului, de această dată aparținând aplicației client. Acesta este creat automat la rularea comenzii *npm init*, prima dată când aplicația este instalată;
* *.gitignore* este o filă de configurare pentru sistemul de versionare utilizat, care îmi permite excluderea unor fișiere precum folderul *node\_modules*, din urmărirea acestora. Despre sistemul de versionare voi discuta în capitolul următor;
* *gulpfile.js* este o filă de configurare a modulului gulp utilizat în cadrul acestui proiect. Gulp este un utilitar de automatizare a unor taskuri, iar motivul pentru care l-am adăugat în acest proiect a fost pentru compilarea codului SASS (Syntactically Awesome Style Sheets) în cod CSS (Cascading Style Sheets), de asemenea pentru generarea unui fișier minifiat css. Fișierele minifiate sunt o practică de optimizare a paginilor web, prin descărcarea conținutului mult mai rapid. Ceea ce se întamplă de fapt este că fișierului inițial îi sunt eliminate toate spațiile și comentariile, de asemenea în unele cazuri se ajunge și la înlocuirea integrală sau parțială a unor denumiri de id-uri sau clase foarte lungi, ceea ce nu este cazul în proiectul de față. Aceste fișiere minifiate vor fi folosite doar pentru interpretarea lor de browser, menținând în continuare o formulă completă a codului în fișierele sursă SASS;
* *index.html* reprezintă fișierul de inițiere a aplicației. Acesta va fi primul interpretat de browser și pe baza lui va fi generat întregul arbore DOM. Conținutul acestui fișier este foarte restrâns, deoarece el are rolul de a încarca dependențele pentru compatibilitatea cu browserele mai vechi, fontul necesar librăriei Materialize, filele CSS pentru Materialize (librăria de css utilizată pentru design-ul responsive) și stilurile custom, generate în fila ns.min.css. De asemenea vor mai fi încărcate ca dependențe librăria javascript jQuery și librăria javascript Materialize. În interiorul câmpului body există tagul <notes-students> </notes-students>, creat în componenta angular *app.component.js*, despre care voi discuta în rândurile următoare;
* *package.json* reprezintă fila de configurare a serverului node, aici se află așa cum am explicat și în capitolul 1, numele proiectului, dependențele modulelor node js și scurtături pentru comenzi, precum cea de pornire a aplicației *start*, care va fi rulată cu ajutorul managerului de pachete NPM astfel: *npm start*;
* tsconfig.json este fila de configurare pentru compilatorul Typescript și este necesară pentru compilarea codului Typescript în ES5, pentru ca acesta să fie înțeles de browser. Codul este compilat în timp real, astfel fiecare filă cu extensia .ts va fi compilată într-o fila javascript, având extensia .js. În conținutul acestei file, care este de fapt un obiect JSON, am configurat modulul utilizat pentru compilare *commonjs*, standardul utilizat pentru compilare, ES5 și alte setări necesare.

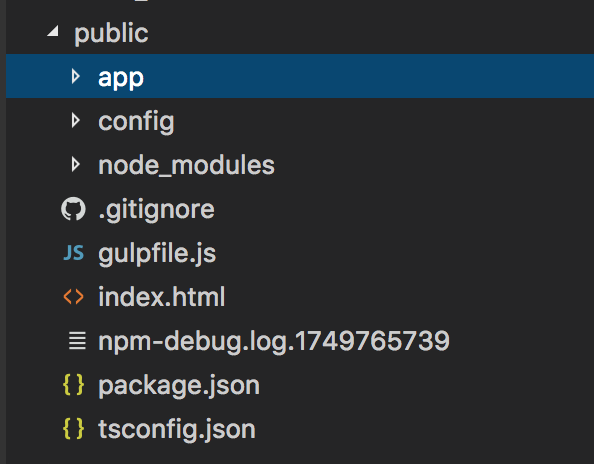


Figura 2.6 Conținutul folderului /public

În continuare voi prezenta modul în care aplicația client este concepută și blocurile componente.

Schema de principiu este următoarea:

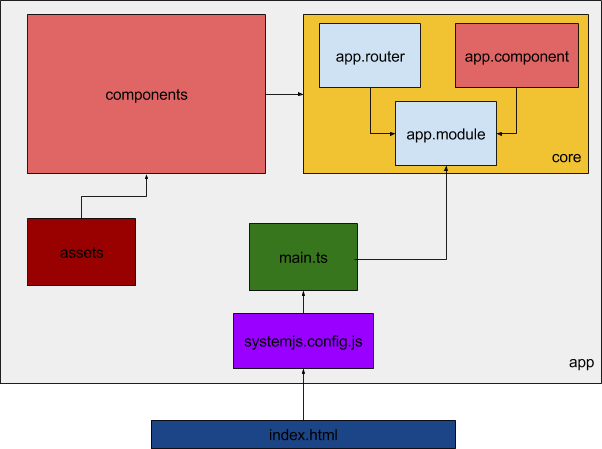


Figura 2.7 Schema de funcționare a aplicației client

Astfel locul în care aplicația este inițiată, îl reprezintă așa cum am precizat și mai sus fila index.html, care încarcă modulele necesare cu ajutorul filei systemjs.config.js. De asemenea aceasta este responsabilă pentru a iniția aplicația Angular, prin mențiunea locului în care fișierul principal se află, *main.ts*, de fapt browserul nu va reda această filă, ci varianta cu extensia .js. În fila *main.ts* este importat modulul *app.module.ts* din folderul */core*, responsabil pentru rularea aplicației. Acesta solicită în continuare rularea routerului și a componentei principale și încarcă toate celelalte module ale aplicației și a componentelor, routere-lor și serviciilor aflate în folderul *components*.

Pe parcursul celor prezentate vor putea fi observate denumirile fișierelor sub forma nume.tip\_bloc.extensie, aceasta fiind o opțiune proprie pentru a ușura dezvoltarea, dar și mentenanța, sporind productivitatea prin definirea unui model de urmat. Numele este întotdeauna unul sugestiv și definește clasa sau fila curentă. Tipul blocului constă în definirea informațiilor: poate fii config, module, components, router, iar extensia este în principal .ts sau .json, iar după compilare aceasta devine .js.

## Nucleu

Nucleul aplicației constă în codul responsabil pentru buna funcționare a aplicației și totodată pentru securitatea și conexiunea cu webservice-ul. Toate fișierele se află în folderul */app/core/* și vor fi prezentate pe rând în acest subcapitol. Structura fișierelor typescript respectă normele OOP și ES6. În partea de început a filelor există importurile altor module Node JS sau clase Typescript necesare, apoi declararea metadatelor, despre care am vorbit în subcapitolul de introducere și nu în ultimul rând declararea claselor.

Am considerat necesară definirea unui subcapitol special pentru acest modul, deoarece este foarte important pentru aplicație și în cele ce urmează voi explica în detaliu fiecare filă/clasă. Conform figurii 2.4, există 3 blocuri importante în componența nucleului aplicației: *app.module.ts*, *app.component.ts* și *app.router.ts*. Despre modulul principal, cel care interconectează toate celelalte module existente în aplicație putem spune că respectă un format clasic, importarea tuturor modulelor definite, a celor npm, dar și a celor 4 componente principale *home.component.ts*, *header.component.ts*, *navbar.component.ts* și *footer.component.ts*, importul acestora fiind necesar, deoarece acestea vor fi afișate prima dată, când se intră în aplicație. În continuare, în conținutul fișierului este definită metadata *@NgModule()*, ce are ca parametru un obiect cu următoarele proprietăți: *imports* - aici vor fi încărcate într-un array toate clasele modulelor utilizate de aplicație, *declaration* - într-un array vor fi încărcate componentele utilizate de acest modul, *providers* - vor fi încărcate clasele ce oferă conținut, în cazul de față *urls.config.ts*, în componența căruia se află rutele oferite de web-service și utilizate de aplicație, *bootstrap* - este un array ce conține clasa componentei principale, *app.component.ts* care este considerată părintele tuturor celorlalte, această clasă fiind definită, conform figurii 2.3. Până acum nu am specificat rutele ce vor fi atribuite fiecărei componente a modulelor; cu acest scop am extins în interiorul filei *app.router* clasa *NgRouter()* oferită de Angular 2. Pentru acest lucru a fost nevoie de utilizarea funcției .forRoot() aplicate asupra clasei *RouterModule*, definită de Angular și a include ca parametru un obiect, de tipul *Router*, acest lucru fiind permis de Typescript. Așa cum am explicat în definiția acestuia, are un mod de operare strict, ceea ce înseamnă că fiecare variabilă are un tip. Pentru început am definit 2 rute */home* căruia îi va răspunde componenta *HomeComponent*. Deoarece am dorit ca această rută să poată fi accesată doar în cazul în care utilizatorul este logat, în conținutul proprietății *canActivate* am apelat clasa *AuthGuard* care implementează clasa *CanActivate* din modulul *Router*, definit de Angular, despre această clasă vom vorbi în subcapitolul următor. Cea de-a doua rută definită este pentru a redirecționa traficul ce vine pe ruta /, adică <http://host:port/>, și redirecționează tot traficul către ruta */home*. Fila componentei principale, *app.component* conține importul metadatei @Component(), în parametrul căreia va fi introdus, așa cum am precizat și mai sus un obiect JSON, acesta conține două proprietăți: *selector*, cu valoarea notes-students, care este de fapt chiar tagul apelat în index.html, practic această proprietate definește un nou tag html, așa cum am precizat în introducere la detaliile despre Angular, iar cea de-a doua proprietate este *template,* în componența căreia poate fi scris cod html. Această utilizare fiind posibilă prin introducerea unor ghilimele speciale `exemplu cod html`, utilizate pentru definirea codului html în javascript, apărut odată cu introducerea standardului javascript ES6. În încheierea filei este definită o clasă goală, fără proprietăți.

În componența fișierului */core* se află de asemenea și alte file Typescript, după cum am precizat anterior *urls.config*. Această filă are în componență metadata @Injectable, care așa cum îi spune și numele are rolul de a permite injectarea proprietăților în alte clase. Fără specificarea acestei metadate, nu ar putea fi posibilă utilizarea acestei clase în proprietatea providers din *app.module*. Clasa curentă conține mai multe proprietăți, prima dintre acestea fiind baseUrl, ce primește ca valoare url-ul de bază al webservice-ului, practic primul nivel al routerului. În cazul în care o nouă funcție va fi dezvoltată pentru sistem, noua rută va trebui specificată în conținutul acestui fișier. O altă filă este *materialize.module*, această clasă este definită pentru a putea utiliza toate proprietățile modulului npm, materialize, despre care voi oferi detalii într-un subcapitol următor. Ultima filă din conținutul acestui folder este *message.config*, în componența căruia este definită o clasă ce conține un obiect msg și o proprietate. Obiectul msg, are o primă proprietate language, care reprezintă numele proprietății ale cărei mesaje vor fi utilizate și pentru moment o a doua, english, definită la rândul ei ca un obiect cu multe proprietăți ce definesc diferite mesaje. Practic este foarte simplu, în acest moment scalarea sistemului pe mai multe limbi, deoarece nu trebuie decât să se adauge un nou obiect cu limba și mesajele dorite, iar prima proprietate language poate fi setată prin crearea unui buton în interfața grafică, pentru ca utilizatorului să i se permită modificarea sa. Există însă un impediment, deoarece în prezent, doar mesajele de succes, erori sau alte notificări sunt implementate în template-urile html din această filă, însă cu toate acestea, implementarea nu ar dura foarte mult.

De asemenea în cadrul folderului despre care discutăm, */core* există un alt director */auth*, despre care voi detalia în capitolul următor, făcând obiectul părții de securitate. Astfel toate aspectele ce țin de nucleul aplicației au fost prezentate, iar aplicația poate rula fără probleme. În acest moment, singurul aspect ce trebuie realizat este definirea modulelor care să extindă această funcționalitate primară.

## Autentificare și Securitate

În capitolul 1, am discutat despre securitatea implementată la nivel de web-service și modul în care aceasta era concepută pentru a oferi un nivel ridicat, însă asta nu este suficient, deoarece aplicația de client este conectată direct la web-service, iar asta poate pune mari probleme sistemului, dacă anumite aspecte nu sunt luate în calcul. Am precizat anterior despre utilizarea unei librării JWT, pe care o voi utiliza și în aplicația client, deoarece am nevoie ca token-ul generat acolo să poată fi interpretat și verificat și de aplicația client, practic având același concept de criptare/decriptare.

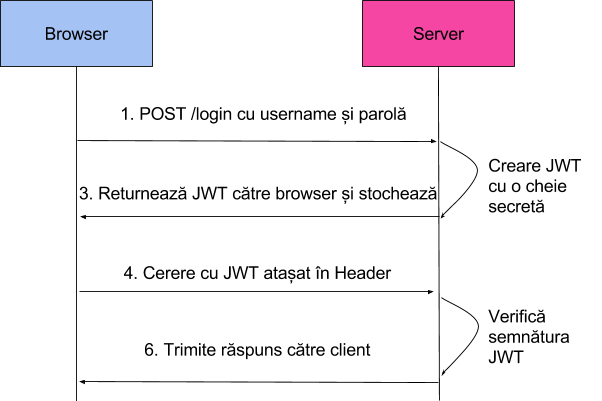


Figura 2.8 Mod de operare JWT

Sursa: [13]

Este necesară verificarea token-ului în aplicația client deoarece avem nevoie să știm dacă token-ul este valid, adică cele 24 de ore au expirat, pentru a cere o nouă logare și de asemenea pentru a verifica nivelul de permisiuni al utilizatorului logat. Motivul pentru care dorim să cunoaștem nivelul de permisiuni în aplicația client se datorează unor restricții asupra rutelor, de exemplu un student nu trebuie să aibă acces pe o rută de administrator, ci trebuie sa îi fie afișat un mesaj corespunzător sau să fie redirecționat pe o altă pagină. Desigur că în prezentarea securității implementate la nivelul web-service-ului am restricționat accesul utilizatorilor pe anumite nivele de permisiuni, deci utilizatorii nu vor putea să afișeze, editeze sau să șteargă informații, însă este de preferat ca utilizatorul să fie informat atunci când încearcă să acceseze o rută fără permisiuni, deoarece în caz contrar se provoacă o dezinformare afișând o pagină goală, pe care nu poate face nimic și în conținutul căreia nu vor fi afișate informațiile așteptate, deoarece web-service-ul nu le va trimite din cauza nivelului de permisiuni.



Figura 2.9 Verificare nivel de securitate înainte de cerere către server

În rândurile ce urmează voi prezenta conținutul fișierului */core/auth*, la care am trimis câteva referințe în rândurile de mai sus. În acest folder se află toate informațiile ce țin de securitatea și logica de restricții asupra rutelor, sistemul de logare și înregistrare a utilizatorilor în platformă. Voi începe prin a detalia fila *auth.guard.ts,* la care am făcut referire în cadrul definirii Router-ului, care reprezintă de fapt o clasă, *AuthGuard*, ce implementează clasa *CanActivate* din modulul *Router*. În interiorul său se află o metodă, numită sugestiv *canActivate*, metodă ce returnează true dacă metoda *loggedIn()* din clasa *AuthService* are valoarea true, adică utilizatorul are un token valid, ceea ce înseamnă că este logat și false, în caz contrar. De asemenea dacă metoda *loggedIn()* returnează false, înseamnă că utilizatorul nu este logat, deci îl voi redirecționa automat pe pagina de logare. În sens larg, această metodă este utilizată în toate sub-routere-le, aflate în diferite blocuri, ce definesc diferite componente, pentru a redirecționa utilizatorul către pagina de login, dacă aceștia nu sunt logați sau token-ul a expirat.

Așa cum am expus și mai sus, în cadrul folderului curent se află și fișierele ce țin de blocul de logare și înregistrare, adică: un modul - auth.module, un serviciu - auth.service, un router - auth.router.module și componente - login.component, signup.component. Voi începe prin descrierea fișierului auth.module, cu rol de a îmbina toate elementele ce alcătuiesc blocul de autentificare. În componența sa vor fi declarate componentele, serviciul și router-ul, cât și alte module Node  JS, folosite pentru rularea acestui bloc, toate utilizând metadata NgModule(). În lista de provideri pot fi regăsite următoarele obiecte aferente claselor AuthService, AuthGuard, ToastService, Messages, iar un element cu totul nou față de celelalte blocuri ce vor fi descrise este metoda *provideAuth()*, ce face parte din fișierele modulului angular-jwt, aceasta are ca parametru de intrare un obiect cu următoarele proprietăți:

* **headerName**: acesta reprezintă numele header-ului asupra căruia este atașat token-ul, în cazul de față a primit valoarea *X-Auth-Token*, deoarece aceasta a fost setată și în webservice[18]. Un header HTTP reprezintă un partea de început a mesajului ce se transmite înainte de fiecare cerere sau răspuns [18];
* **headerPrefix**: reprezintă prefixul valorii headerului [18], acest text va fi pus în fața token-ului și are valoarea *‘Bearer’* asemenea celei setate în webservice;
* **tokenName**: reprezintă numele token-ului curent [18], deoarece pot fi utilizate mai multe token-uri în funcție de necesități în cadrul unui modul, acesta are valoarea ‘*userToken*’;
* **tokenGetter**: reprezintă locul în care token-ul a fost salvat [18], în cazul de față în localStorage, aspect pe care îl vom dezvolta într-un paragraf următor. În localStorage a fost stocat cu numele *‘usetToken’*;
* **globalHeader**: conține alte headere utilizate pentru transmiterea mesajelor, iar valoarea corespunzătoare constă într-o listă de obiecte cu numele headerelor și valoarea acestora [18]. Am utilizat un singur token 'Content-Type': 'application/json';

Fișierul utilizat ca router pentru modulul curent, *auth.router.module*, este reprezentat de o clasă ce are atașată metadata @NgModule, având ca parametru un obiect cu două proprietăți:

* import: conține ca valoare funcția RouterModule.forChild(), care are la rândul ei ca parametru un vector de obiecte, care definesc de fapt rutele. În cazul de față avem două rute, prima pentru logare ce are următoarele proprietăți:
  + path: acesta reprezintă numele rutei, ce va apărea în casuța de url-uri având valoarea setată sugestiv *login*;
  + component: obiectul corespunzător componentei ce va răspunde rutei curente, *LoginComponent*; iar cea de-a doua rută constând în ruta de înregistrare, sub același format ca prima;
* path: cu valoarea *signup*;
* component: cu valoarea obiectului corespunzător componentei *SignUpComponent*;
* export: această proprietate reprezintă modulul sau modulele ce vor fi exportate pentru a fi utilizate în cadrul modulului principal app.module și are setată valoarea obiectului RouterModule, pentru a cuprinde pe langă cele două rute definite ințial pentru homepage și pe acestea.

Se poate remarca faptul că nu am utilizat în nici-una din cele două rute proprietatea *canActivate*, deoarece aceste două rute pot fi accesate de oricine și pentru că nu au restricții.

Fila de servicii, concepută pentru acest model are un format asemănător cu toate celelalte, conține mai multe metode, utilizate în principal pentru a descrie funcționalitatea aplicației în raport cu webservice-ul. Fila are numele *auth.service* și reprezintă o clasă, cu numele *AuthService*, în constructorul căruia sunt definiți ca parametrii privați 3 obiecte, de tipul Http, ApiUrls și Router, obiecte ce vor fi utilizate în conținutul metodelor pentru a accesa unul din cele 3 module pe care le extind. În conținutul clasei există următoarele metode:

* **login**: este o metodă utilizată pentru a trimite către webservice username-ul și parola introduse de utilizator și de a primi răspunsul, această metodă este apelată în interiorul funcției cu același nume, din componenta *login.component*, în conținutul căreia răspunsul este procesat. Modul în care webservice-ul lucrează, a fost prezentat în capitolul anterior și tot acolo am prezentat motivele pentru care fiecare răspuns pe care acesta îl returnează este însoțit de un *statusCode*. În cazul de față dacă răspunsul are atașat codul 0, înseamnă că username-ul și parola sunt corecte, caz în care trimit un mesaj de informare că logarea s-a efectuat cu succes, salvez în localStorage token-ul, pentru a putea fi folosit mai târziu și redirecționez utilizatorul către pagina de */home*. Dacă statusul este 4, parola este greșiă și afișez un mesaj corespunzător, iar dacă este 1, atunci username-ul nu există. Am optat pentru acest tip de verificare pentru a oferi informații exacte asupra erori, unde este cazul, deoarece nu de multe ori am întalnit site-uri în care suntem notificați că username-ul sau parola sunt greșite, însă nu exista o referire stricta asupra erori.
* **logout**: această funcție este una foarte simplă, principalul obiectiv este acela de a șterge din localStorage, variabila *usetToken,* iar apoi face un redirect către pagina de login;
* **loggedIn**: este o metodă ce returnează o valoare booleană, ce utilizează la rândul ei o metodă din clasa JWT, *tokenNotExpired*(), care verifică dacă token-ul curent este expirat sau nu;
* **signUp**: folosită pentru a trimite prin metoda POST către server, date de înregistrare. Este apelată în componenta *signup.component*, în metoda cu același nume;
* **getRegistrationNumber**: este o metodă ce trimite o cerere de tip GET către server pentru a primi informații referitoare la numărul matricol introdus de student în formularul de înregistrare;
* **getUserWithRegistrationNumber**: metodă folosită pentru a returna datele utilizatorului cu numărul matricol specificat ca parametru al funcției;
* **getUserWithUsername**: returnează utilizatorul cu username-ul specificat ca parametru al funcției;

Despre componente pot spune, așa cum am precizat și în introducere faptul că acestea au rolul de afișa template-ul și de a prelucra în interiorul clasei aferente datele ce sunt introduse în template. Atât în clasa *LoginComponent*, cât și în *SignUpComponent*, utilizez o tehnică pusă la dispoziție de Angular, care se numește Data Binding și presupune că valorile introduse în formulare să poată fi procesate în timp real de clasa căreia aparține, astfel orice modificare adusă în formular apare și în variabila corespunzătoare, cât și invers. Dacă asupra unei variabile se aplică un proces de transformare, aceasta se va propaga și în formular. Despre modul în care se realizează funcția de login, am precizat mai sus în cadrul descrieri serviciului. Cât despre signup, clasa conține două metode, în corelație cu faptul că înregistrarea se face în 2 pași. În primul pas se utilizează funcția *checkRegistrationNumber()*, iar în template este afișat un singur câmp, pentru introducerea numărului matricol de către student. Prin procesul de înregistrare, unui student îi este asignat un cont, pentru a-și putea vedea notele și situația financiară, dacă este la subvenție sau trebuie să achite taxa. Contul de administrator nu se poate înregistra, ci poate fi creat din panoul de administrator de un alt administrator, tocmai pentru a nu crea breșe de securitate, în contextul în care nu este foarte frecventă înscrierea unui nou administrator în platformă. În cadrul funcției specificate, se verifică în primul rând dacă un număr matricol a fost inserat, înainte ca butonul de verificare să fie apăsat, apoi cu ajutorul funcției din serviciu, *getRegistrationNumber()*, se verifică dacă numărul matricol specificat de student există. Dacă serverul returnează un cod de status 0, înseamnă că a găsit cel puțin un rezultat și că există un student cu acel număr matricol, iar în cazul în care codul este 1, înseamnă că numărul matricol nu a fost găsit, deci voi afișa un mesaj de eroare. Dacă numărul matricol există, se verifică în continuare, făcând o nouă cerere către server prin metoda *getUserWithRegistrationNumber()*, din serviciu, iar dacă serverul returnează un statusCode 0 se afișează un mesaj, deoarece numărul matricol are un cod înregistrat deja, iar dacă este în schimb este 1, înseamnă că nu există un utilizator conectat la acel număr matricol, deci salvez într-un obiect numărul matricol și nivelul de administrare 3, corespunzător studentului, pentru a pregăti pasul următor. În acest moment totul este pregătit și vor fi afișate alte două câmpuri pentru introducerea username-ului și a parolei. Metoda folosită pentru acest formular este *signup()*, în conținutul căreia se verifică dacă numărul matricol există și nu cumva accidental a fost șters. Ca o metodă de prevenție am dezactivat editarea acestuia în interfața grafică, pentru pasul curent, apoi se face o cerere către server pentru a verifica disponibilitatea username-ului, dacă serverul răspunde cu statusul 0, înseamnă că numele este deja folosit, în acest caz afișez un mesaj corespunzător, de notificare a studentului că numele este deja folosit, altfel dacă *statusCode* este 1, folosind funcția *signup()* din serviciu înregistrez utilizatorul, afișez un mesaj de confirmare că noul cont a fost creat și îl redirecționez către pagina de pornire.

De asemenea trebuie sa nu omit faptul ca formularele de logare și înregistrare au un sistem de validare a datelor, înainte ca cererile să fie trimise către server. Se verifică tipurile așteptate, dacă este un e-mail valid, dacă un câmp obligatoriu este completat sau nu, pentru a evita situațiile neplăcute de a trimite date incorecte către server.

## Module de extensie

În subcapitolele prezentate anterior am discutat despre blocurile elementare ce intră în componența nucleului și modulul de securitate, însă pentru a dezvolta întreaga funcționalitate propusă, aplicația de client are nevoie de mult mai multe blocuri, astfel în acest subcapitol voi prezenta blocurile ce definesc aplicația client, aflate în folderul *app/components*. Fiecare bloc din acest folder are o denumire intuitivă ce descrie rolul său în cadrul aplicației, iar majoritatea acestor foldere conțin așa cum am precizat și în rândurile de mai sus conceptul de modul: un modul, un serviciu, un router și o componentă. Excepție fac blocurile ce intră în componența modulului principal, *app.module* și voi începe cu descrierea acestora. Doresc să subliniez diferența dintre cuvântul utilizat pentru a defini o parte din funcționalitatea aplicației - modul și cea de modul Angular. În toate folderele următoare se pot remarca dubluri ale filelor, deoarece așa cum am descris anterior fișierele Typescript, ce au extensia .ts sunt compilate în timp real și sunt stocate în fișiere cu extensia .js.

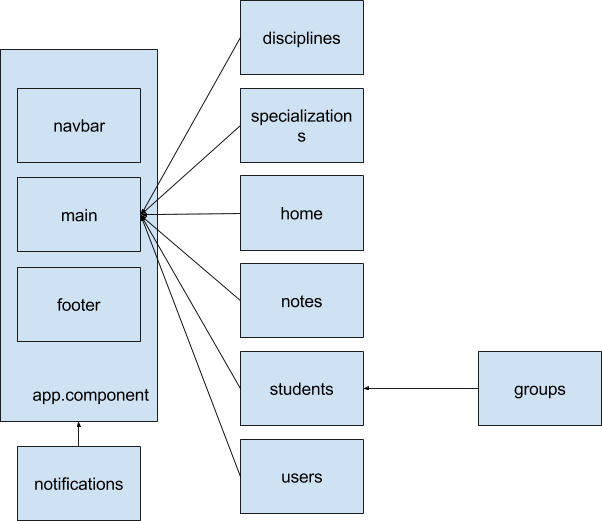


Figura 2.10 Structura componentelor în aplicația Client

Voi începe prin a descrie pentru început blocurile ce sunt folosite în majoritatea modulelor, iar printre acestea se pot remarca:

* *footer*: în componența acestui folder se află o singură componentă, *footer.component*. Este o componentă foarte simplă în interiorul căreia în proprietatea *template* sunt adăugate câteva taguri html ce prezintă date copyright, despre mine și pot fi adăugate de asemenea și altele. Această componentă reprezintă o parte din șablonul inițial și este încarcată pe toate url-urile;
* *home*: în componența acestui folder se află o singură componentă. Reprezintă componenta ce va fi afișată pe ruta */home*, iar momentan în cadrul proprietății template poate fi remarcat un text ce prezintă idealurile pe care și le propune aplicația în viitor, de integrare a unor mesaje pe aceasta pagină, aceasta fiind prima pe care utilizatorul este redirecționat după logare;
* navbar: în componența acestui folder poate fi remarcată componenta *NavbarComponent*, responsabilă pentru funcționalitatea barei de navigare. Folosește ca template navigația implicită a librăriei Materialize. În cadrul clasei corespondente putem remarca declararea unei proprietăți *isAdmin*. Aceasta este folosită prin metoda data binding pentru afișarea sau ascunderea unor elemente din bara de navigare în funcție de nivelul de administrare. În constructor sunt apelate două metode *checkIfAdmin()*, funcție care preia variabila stocată la logare în localStorage ce conține numele utilizatorului curent și verifică dacă utilizatorul este administrator sau nu, modificând valoarea proprietății *isAdmin*. De asemenea avem nevoie de implementarea unei logici care verifică dacă pe parcursul navigării nivelul de administrare al utilizatorului s-a schimbat, și este apelată din nou metoda *checkIfAdmin()*. Pentru a urmări aceste schimbări am folosit o funcționalitate oferită de modulul Router, prin intermediul metodei *subscribe()* a clasei *Events*. De asemenea pentru a putea utiliza o bară de navigare pentru dispozitivele mobile exact ca în aplicațiile native, am setat butonul de hamburger menu ca buton responsabil pentru lansarea barei de meniu. Acest eveniment este apelat în cadrul unei metode ce face parte din ciclurile Angular, *ngAfterViewInit()*, evenimentul fiind acționat imediat după ce pagina a fost încarcată în browser;
* notification: în componența acestui folder este definit un serviciu responsabil pentru notificări. Pe parcursul prezentării aplicației am specificat faptul ca în cazul unor erori sau succese vor fi afișate mesaje de informare pentru utilizator. Aceasta extinde funcționalitatea librăriei Materialize apelând funcția *toast()* cu afișarea mesajului pentru 0.5s.

Toate celelalte module au o componentă, un modul, un serviciu și un router. Rolul serviciului este acela de a interacționa cu web-service-ul, dar totodată în majoritatea modulelor am introdus o logică, ce îmi permite jonglarea cu date mult mai simplă și totodată îmbunătățește timpul de răspuns și susține astfel ideea de timp real. Ceea ce am constat în calitate de utilizator este că de foarte multe ori apar cereri către server doar pentru a solicita în mod repetat aceleași date, ceea ce nu este benefic din nici-un punct de vedere, deoarece serverul este solicitat fără motiv, iar utilizatorul trebuie sa aștepte nejustificat. Modul de lucru cu obiecte (OOP) mi-a permis ca odată solicitate datele să fie stocate într-un obiect și folosite pe parcursul interacțiunii. Acest mod de lucru este disponibil cât timp utilizatorul menține ruta curentă, spre exemplu */admin/disciplines*. Obiectul se inițializează cu valoarea obiectului pe care s-a dat click, fiind extras dintre toate rezultatele afișate. De îndată ce ruta părinte se schimbă și devine spre exemplu */admin/specializations*, obiectul anterior este distrus și unul nou este creat pentru stocarea datelor extrase pentru specializări. Acestea fiind menționate în partea de început a claselor cu rol de serviciu se remarcă declararea unui obiect cu nume sugestiv fiecărei clase de tip *any*. Apoi sunt declarate metode pentru setarea acestui obiect cu denumirea *setNumeSugestivClasei()*, pentru extragerea obiectului din această clasă *getNumeSugestivClasei()*. În constructor se poate observa faptul că obiectul este declarat cu valoarea *null*. Am făcut acest lucru, deoarece există șansa ca în cazul unei conexiuni mai slabe de internet sau în cazul în care serverul este aglomerat și răspunsul serverului pentru a seta conținutul acestui obiect este întarziat, metodele care vor încerca să acceseze conținutul să nu se lovească de o eroare de nedefinire a acestuia. De asemenea un alt motiv este acela de a ne asigura că de fiecare dată când încercăm să stocăm datele provenite de la server, obiectul este gol și în conținutul său nu au apărut alte valori nedorite. În componența serviciului sunt declarate clase pentru extragerea tuturor informațiilor de pe ruta respectivă sau a unor informații parțiale în funcție de id-uri sau denumiri și nu în ultimul rând pentru inserare, actualizare și ștergere *insert(disc)*, *update(disc)*, *delete(id)*. Ceea ce se poate remarca în cadrul fiecărei metode este ca în interiorul ei se apelează un url din clasa *ApiUrls*, specific fiecărei rute a webservice-ului, în funcție de datele ce se doresc solicitate. Aceste funcții returnează un Promise(), deoarece datele pot veni mai devreme sau mai târziu de la server. Când răspunsul de la server a sosit, acesta este returnat în format json într-o funcție .then(). În cazul în care se primește o eroare, aceasta este tratată și pe lângă afișarea mesajului atașat, verific dacă statusul este 403, ceea ce îmi sugerează faptul că utilizatorul nu este autorizat să acceseze ruta curentă,  îi returnez un mesaj pentru acces neautorizat și redirecționez utilizatorul către pagina de homepage.

În componența fiecărui modul sunt incluse dependențele de modulele npm sau de cele create în aplicație, declararea componentelor și a serviciilor utilizate. În metadata ngModule, se introduce ca parametru un obiect ce conține în majoritatea modulelor 3 proprietăți:

* *imports*: pe langă importarea routerului specific fiecărui modul în parte aici sunt incluse module precum *BrowserModule*, *HttpModule*, *FormsModule*, dar și *MaterializeModule*;
* *declarations*: aici sunt importate componentele atașate fiecărui modul în parte;
* *providers*: aici se importă obiectele serviciilor aferente fiecărui modul dar și clasa *AuthGuard;*

Despre filele router trebuie menționat faptul că respectă modelul router-ului primar și folosește de asemenea modulul implicit din Express pentru Router, fiecare rută fiind declarată prin specificarea proprietăților: path, component și canActivate, așa cum au fost prezentate anterior în cadrul paragraful dedicat router-ului default. În majoritatea cazurilor se utilizează rute părinte /admin/numecomponenta pentru încarcarea componentei ce va afișa toate informațiile și a unei subrute /admin/numecomponenta/add sau /admin/numecomponenta/edit ce vor încarca amandouă aceiași componentă.

În majoritatea modulelor putem remarca existența a două componente, acest lucru este datorat faptului că majoritatea respectă același pattern ce constă în existența unei clase ce afișează toate rezultatele (exemplu: disciplines.component, notes.component, specializations.component, students.component, users.component) și o alta utilizată pentru ruta de editare sau adăugare de informații. Am decis să utilizez aceiași componentă și pentru adăugare cât și pentru editare, deoarece amandouă utilizează aceleași câmpuri, iar diferența este sesizabilă doar în rândul metodelor utilizate. În cadrul acestor componente se utilizează funcționalitatea de data binding la întreaga ei capacitate, aceste componente având în mare parte formulare. În cazul în care datele vor fi adăugate, vor merge de la formular către model, iar pentru editare va trebui să funcționeze în ambele sensuri. Asemenea celor menționate în cazul serviciilor, în clasa atașată componentelor sunt declarate obiecte, inițializate cu valori nule în constructor. Acestea au rolul de a permite legatura dintre model și template. Sunt folosite stări de viața ale Angular pentru a inițializa o parte din aceste obiecte apelând metode responsabile cu apelul unor metode de tipul getAll() din servicii, în ngOnInit(). De asemenea se remarcă existența unor metode pentru adăugare add(), editare edit(), ștergere delete() și alte metode necesare implementării acestor funcționalități. Ștergerea se realizează prin apelul metodei de ștergere din serviciul aferent, după confirmarea faptului că utilizatorul este sigur prin utilizarea unei funcții implicite javascript confirm(). Tot în metoda ngOnInit() în cadrul componentelor ce definesc adăugarea/editarea se face verificarea dacă url-ul conține cuvântul edit sau nu pentru că în cazul în care ruta este pentru editare, din serviciu să fie extrase valorile ce trebuie afișate în formularul pentru editare. Acest lucru se face cu ajutorul metodelor de tip get (exemplu getStudent()) și se stochează într-un obiect local al componentei ce va fi trimis către server la apăsarea butonului de editare. Având la bază utilizarea obiectului din serviciu se poate întampla ca un utilizator sa acceseze o rută (exemplu /admin/students/edit) care deși este o rută validă, iar utilizatorul poate să aibă drepturile de acces necesare, datele să nu existe, pentru că practic nu există detalii referitoare la ce anume se dorește editat. Astfel se verifică dacă există date stocate în obiectul din serviciu, iar în cazul în care nu există, redirectionez utilizatorul către ruta părinte (conform exemplului anterior către /admin/students). Există totuși posibilitatea ca în url să adaug id-ul înregistrării, însă am preferat această metodă, pentru a evita cazurile în care se încearcă atacuri. Totodată, aici se utilizează o variabilă *editMode* ce stabilește dacă ruta curentă este de editare sau nu, și este utilizată pentru afișarea id-ului înregistrării, deoarece în cazul în care se dorește adăugarea unei noi înregistrări, nu există încă un id. Metodele utilizate în cadrul componentelor utilizează aceiași structură ca și cele din servicii, așteptând răspunsul promisiunii din serviciu în instrucțiunea .then(){...}, iar în cazul unei erori aceasta este prinsă și returnată în blocul .catch(message){...}

## Stiluri și design

În cadrul acestui subcapitol doresc să clarific aspecte ce țin de design, de librăria Materialize la care am tot făcut referire pe parcursul acestui capitol dar și despre conținutul directorului */app/public/assets,* loc în care am stocat stilurile SASS și CSS. De asemenea trebuie să menționez faptul că design-ul ales pentru site respectă normele Material Design.

Material Design este un limbaj de programare grafică dezvoltat de google începând cu 2014 și introdus pentru prima dată în cardurile din Google Now [19]. Acest design pune accentul pe utilizarea de layout-uri regulate, având la bază forme geometrice simple precum pătrate sau cercuri, animații ce se redimensionează, padding, efecte de lumini și umbre [19][20]. Ceea ce m-a făcut să optez pentru un astfel de design sunt și tendințele de pe piață, dar și dorința de a utiliza un design simplu și eficient, care să îmi permită dezvoltarea unui mediu ușor de utilizat de către utilizatori, indiferent de nivelul de utilizare al tehnologiei. Material Design este un sistem unic de a combina teoria, resursele și uneltele pentru o experiență digitală de maestru [22]. Obiectivul realizării acestui limbaj a avut la bază dorința creatorilor de a sintetiza principii clasice ale unui design bun în inovațiile existente și viitoare aduse de tehnologie și știintă [22]. Dezvoltarea unui singur sistem aliniat care acceptă o experiență unică pe toate platformele și rezoluțiile [22]. Principiile pe care se bazează acest sistem este că materialul este inspirat adânc din hârtie și cerneală și că acestea trebuiesc îmbinate și în tehnologie [22]. Suprafețele și colțurile materialelor oferă indicii vizuale care sunt înrădăcinate în realitate [22]. Fundamentul utilizării luminii, suprafețelor și mișcării sunt cheia  mișcării, interacțiunii și existenței obiectelor în spațiu și a relației dintre ele [22].

Exemple de principii Material Design pot fi observate în exemplele următoare:

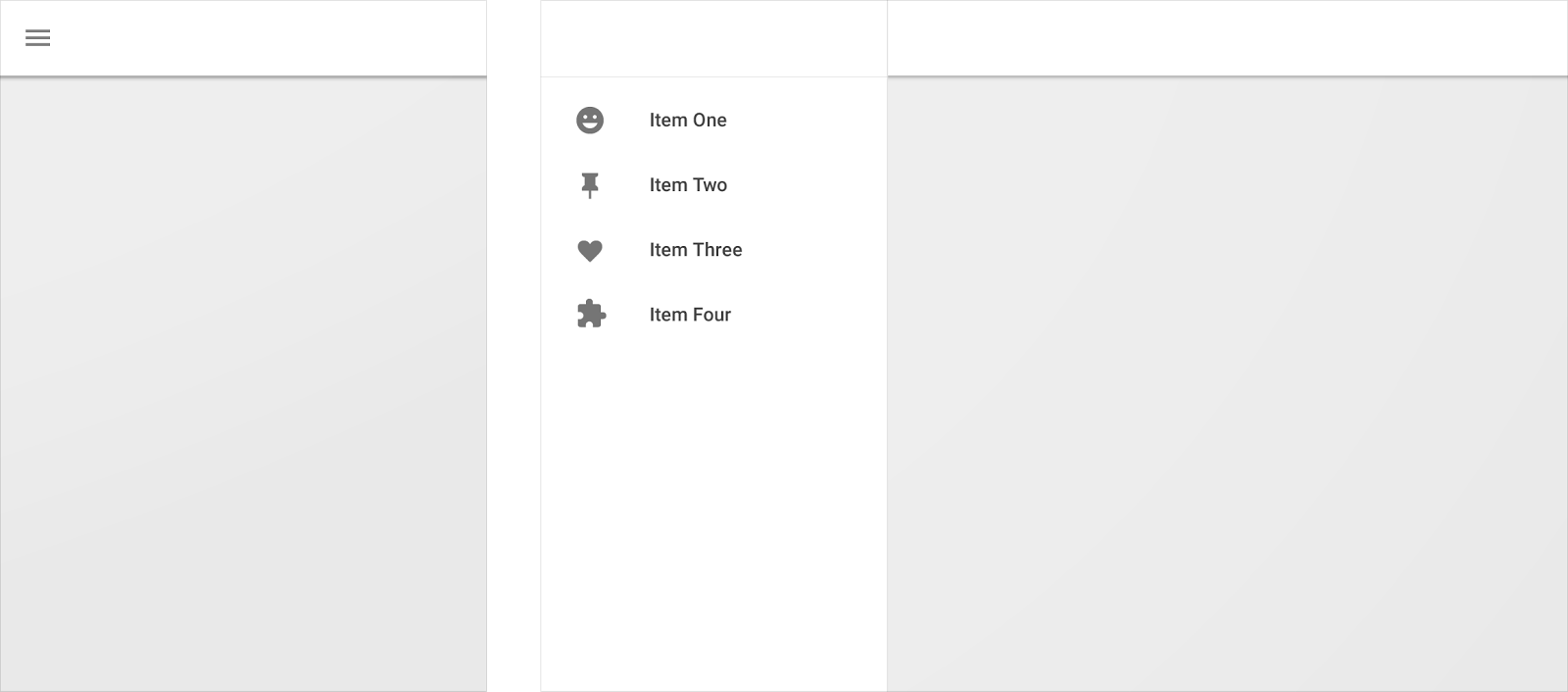


Figura 2.11 Principii Material Design bară laterală

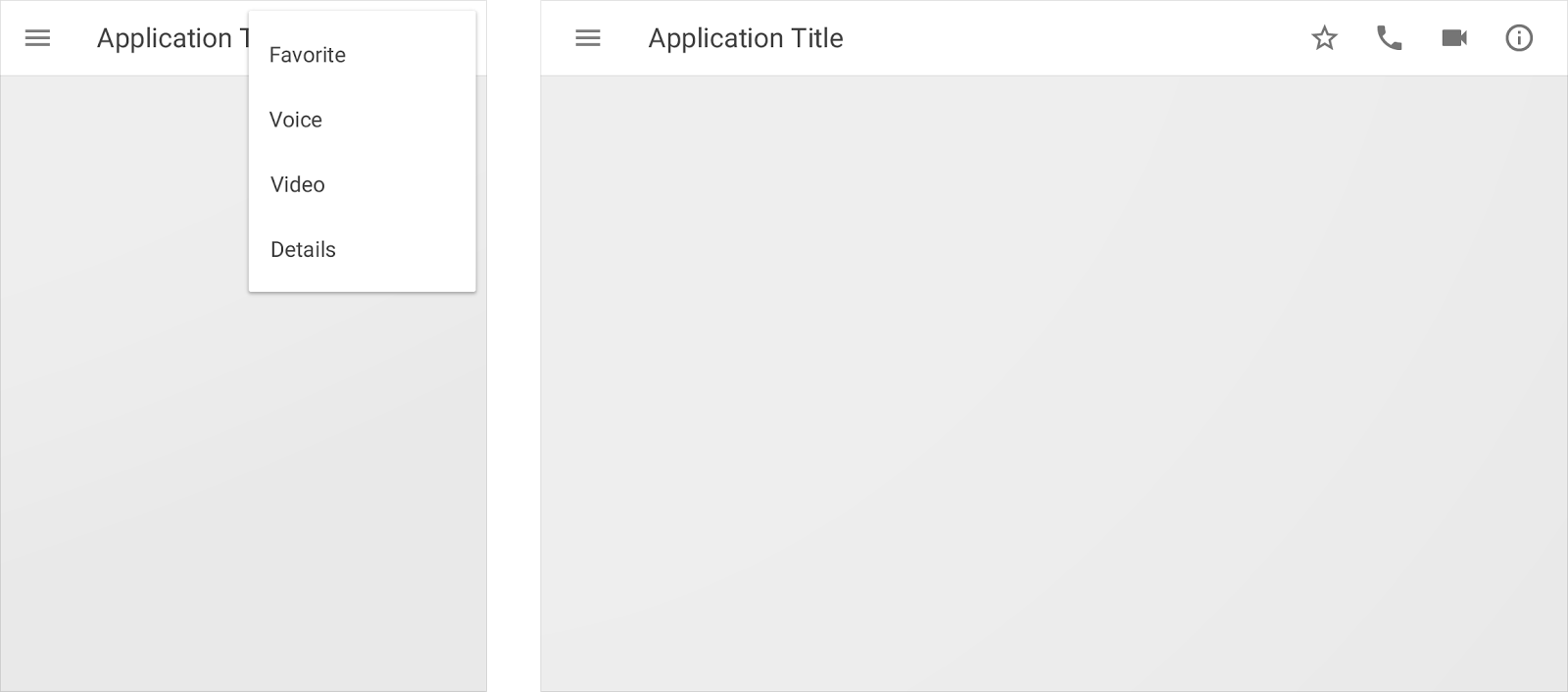


Figura 2.12 Principii Material Design meniu

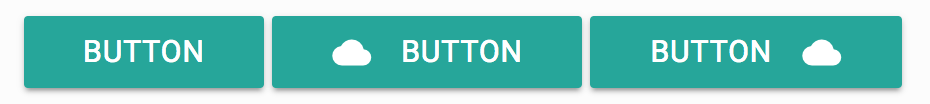


Figura 2.13 Principii Material Design butoane

Îmbinarea tehnicilor și principiilor referitoare la Material Design m-au făcut să caut o librărie care sa îmi permită integrarea tuturor acestor principii. Am căutat de asemenea ca librăria sa îmi pună la dispoziție o serie de avantaje precum utilizarea unui design responsive, bazat pe grile, disponibilitatea pentru stilizarea unor template-uri bazate pe material design, integrarea unor stiluri pentru butoane și formulare, iconițe, dar și a unor serii de calități javascript precum: carousel, suprafață de colaps, butoane dropdown, modale, parallax design, bară de navigare pentru mobile, notificări, dar nu în ultimul rând foarte importantă era integrarea acesteia cu Angular 2. Am căutat printre mai multe librării de specialitate, una dintre cele mai răspândite fiind Bootstrap, însă deși îndeplinea majoritatea cerințelor, nu îndeplinea una dintre cele mai importante, faptul că nu are la baza un concept de Material Design, iar în final am conchis asupra librăriei Materialize.js.

Datorită expandării și beneficiilor pe care mi le-a pus la dispoziție librăria Materialize, nu a fost necesară implementarea unor stiluri customizate, conținutul filelor din folderul /app/assets/sass fiind redus. Conform celor menționate mai sus în cadrul acestui proiect am utilizat un modul de automatizare numit Gulp, pentru a compila codul sass în cod css, deoarece browserul nu poate interpreta nativ cod sass, ci trebuie convertit. Motivul pentru care am ales utilizarea unui astfel de limbaj îl reprezintă permisivitatea de a utiliza instrucțiuni din programarea clasică într-un limbaj de design. Implicit în css nu este posibilă declararea unor condiții de tip if then, iterații pentru definirea unor situații similare. Sass este un limbaj de scripting interpretat în css, care permite două sintaxe, una bazată pe indentări similară cu modul de utilizare HAML (limbaj similar pentru a scrie cod HTML) și distinsă prin salvarea fișierelor cu extensia .sass și ce-a de-a doua sintaxă, cea nou introdusă, prin care blocurile de clase sau id-uri sunt separate precum în CSS prin acolade și finalul proprietăților fiind marcat cu punct și virgula, poartă denumirea de SCSS, filele fiind salvate cu extensia .scss [24]. În cadrul acestui proiect am utilizat sintaxa scss, însă nu am exploatat foarte tare benficiile oferite de acesta, din lipsa necesității de a defini prea multe stiluri custom. În conținutul fișierului se pot remarca fișiere al căror nume începe cu underline, acestea vor fi excluse de compilator, singurele fișiere compilate fiind acelea fără underline în față denumirii. Astfel am folosit un singur fișier denumit ns.scss, în conținutul căruia am importat celelalte file, \_navbar și \_container. Am făcut acest lucru pentru a putea administra mai ușor fișierele pe care le doresc compilate sau nu. Toate sursele compilate sunt mutate în fișiere css în folderul */app/public/css.* Am fost nevoit să definesc înalțimea minimă a paginii, în cazul în care conținutul nu era suficient de înalt pentru a acoperi întregul ecran, deoarece existau cazuri în care footer-ul era afișat la jumătatea paginii și nu la final. De asemenea culoarea implicită a Materialize era roz, astfel am modificat-o cu o nuanță de albastru, totodată respectând principiile Material Design.

## Concluzii

În cadrul acestui capitol am prezentat aspecte ce descriu funcționalitatea aplicației client, noțiuni descriptive ale tehnologiile utilizate pentru realizarea acestui subsistem precum: Angular 2, TypeScript, Materialize, dar și a modului de integrare a componentelor și modulelor, aspecte ce țin de securitatea implementată la nivelul aplicației client. Pot spune ca beneficiul major al redactării dar și realizării acestei aplicații îl reprezintă cunoștintele dobândite, noțiunilor noi acumulate dar și a articolelor și materialelor de documentație citite.

# Interacțiunea cu factorul uman

## Introducere

Un citat celebru spune “Nu judeca o carte după copertă!”, însă consider ca o copertă care relevă valoarea ce se află în conținutul cărții este capabilă să vândă acea carte. Și continuând în aceilași context, după primul contact cu acea carte suntem tentați să aruncăm o privire în partea de conținut și/sau prefață, pentru a înțelege cât de abordabilă și bine structurate sunt informațiile. Această mică asociere cu lumea cărților este valabilă și în contextul aplicației realizate, coperta vinde, dar conținutul susține. În acest capitol mi-am propus să prezint aspecte ce țin de interacțiunea utilizatorului cu aplicația, indiferent de statutul pe care acesta îl rezervă ca student, administrator, profesor sau dezvoltator.

În rândurile ce urmează voi prezenta aspecte legate de experiența utilizatorului, cunoscută în literatura de specialitate sub denumirea de User Experience(UX), noțiuni reprezentative pentru deciziile de aranjament în pagină, efecte, animații, umbre, capturi de ecran și explicațiile aferente.

Așa cum reiese din titlul capitolului, factorul uman poate fi și programatorul sau echipa de dezvoltatori, iar din acest punct de vedere voi detalia motivul pentru care am ales să integrez un sistem de versionare, GIT, încă de la începutul desfășurării proiectului.

## Experiența Utilizatorului

Experiența Utilizatorului se referă la emoțiile și atitudinile unei persoane despre utilizarea unui produs, sistem sau serviciu [25]. Include aspecte practice, experimentale, afective, reprezentative și importante din punct de vedere al relației dintre om și computer [25]. În plus include percepțiile unei persoane despre aspectele unui sistem precum utilitatea, ușurința în utilizare și eficiență [25]. Experiența utilizatorului poate fi considerată subiectivă, în funcție de gradul de cunoștințe ale utilizatorului [25].

Cea mai puternică însemnătate a experienței în raport cu utilizatorul o reprezintă însă chiar tranzlatarea programatorului în pielea utilizatorului de rând și încercarea de a aplica un design cât mai intuitiv, notificări sau mesaje de succes/erori, dar și animații sugestive pentru a permite utilizatorului să înțeleagă ce se întamplă [26]. Făcând o tranzlatare în pielea utilizatorului, am încercat să fac o listă cu idei de care trebuie să țin cont, pentru ca utilizatorii să se bucure de o experiență cât mai placută. Astfel am încercat să țin cont de următoarele specificații:

1. **Să exprime clar cum rezolvă problemele și care sunt beneficiile pe care utilizatorii le au.**

Din acest punct de vedere încă de la începutul navigării, pe pagina de home utilizatorii sunt întampinați cu un mesaj care le explică funcțiile și beneficiile pe care aplicația  le poate aduce. Meniul este afișat în partea de sus a paginii cu denumiri sugestive, iar pentru mobil acesta este ascuns, fiind afișat prin apăsarea clasicului buton hamburger menu cu 3 linii orizontale suprapuse, ce poate fi vizibilă în imaginea 3.2.

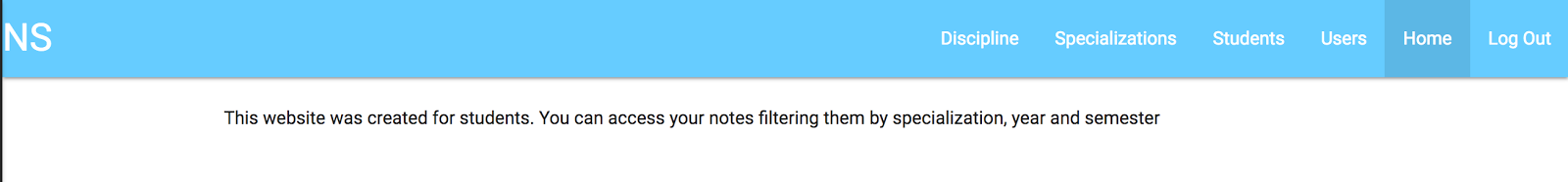


Figura 3.1 Bară de navigare în modul desktop

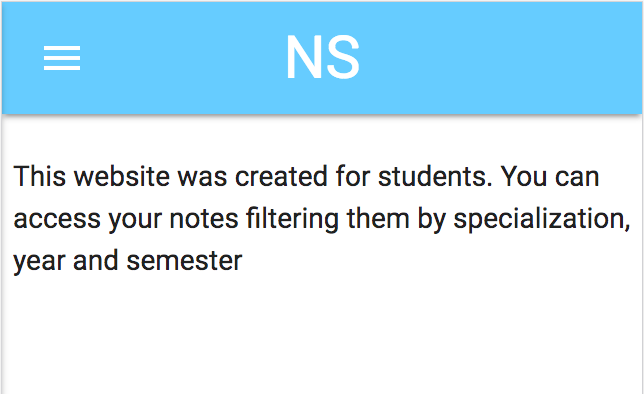


Figura 3.2 Bară de navigare în modul mobile (Hamburger Menu)

1. **Poziționarea**

Studii realizate în rândul utilizatorilor arată că elementele importante și totodată reprezentative pentru utilizatori trebuiesc poziționate în prima jumătate a paginii pentru a putea fi găsite cu ușurință la o simplă privire. Totodată trebuie luat în calcul faptul că nu trebuie să fie susceptibil conceptul de haos, prin încercarea de a aglomera totul în prima jumătate a paginii.

1. **Ilustrații**

Este bine știut faptul ca informațiile vizuale sunt ingredientul principal în captarea atenției umane, procesarea vizuală fiind de mult mai rapidă decât un simplu text. Acestea fiind spuse afișarea de imagini, grafice, clipuri video, notificări, animații sunt mult mai vizibile decât afișarea unui simplu text. În acest context am optat pentru afișarea unor notificări de tip Toast din librăria Materialize. Câmpurile din formulare sunt evidențiate cu nuanțe de roșu în cazul în care informația introdusă nu este validă și cu nuanțe de verde în cazul în care utilizatorul introduce date într-un camp pentru a evidenția câmpul activ, acest tipar fiind respectat în cadrul întregii interfețe pentru a obișnui utilizatorul.

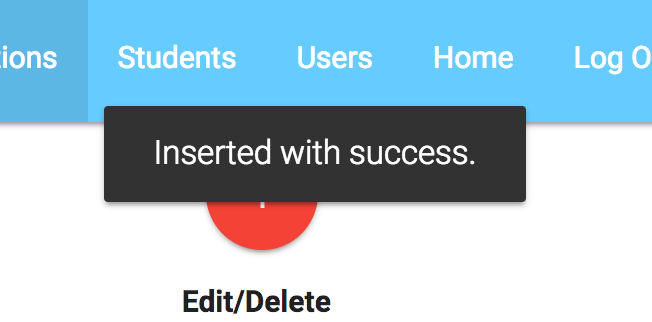


Figura 3.3 Afișare notificare Toast pentru mesaj de success

1. **Adăugarea de spații în conținut**

În cazul paginilor ce au mult conținut, urmărirea paginii poate deveni dificilă, iar găsirea informațiilor necesare poate reprezenta o adevarată provocare pentru utilizator, motiv pentru care este de preferat ca paragrafele să fie separate prin spații libere care să evidențieze începutul sau sfârșitul unui bloc de date/informații.

1. **Pornirea acțiunilor**

Acțiunile sunt pornite în principal cu ajutorul unor butoane. Este de preferat ca design-ul acestora să respecte un model în toată implementarea website-ului pentru a nu induce în eroare utilizatorii. Un aspect important îl reprezintă claritatea, deoarece nu de multe ori mi-a fost dat să observ ca utilizator, faptul că trebuie să intuiesc locul în care un buton este poziționat sau chiar mai rău, existența mai multor butoane cu aceleași atribute. Este de preferat ca butoanele ce servesc acțiuni precum trimiterea unui formular să fie plasate după acestea pentru ca utilizatorul să fie familiarizat.

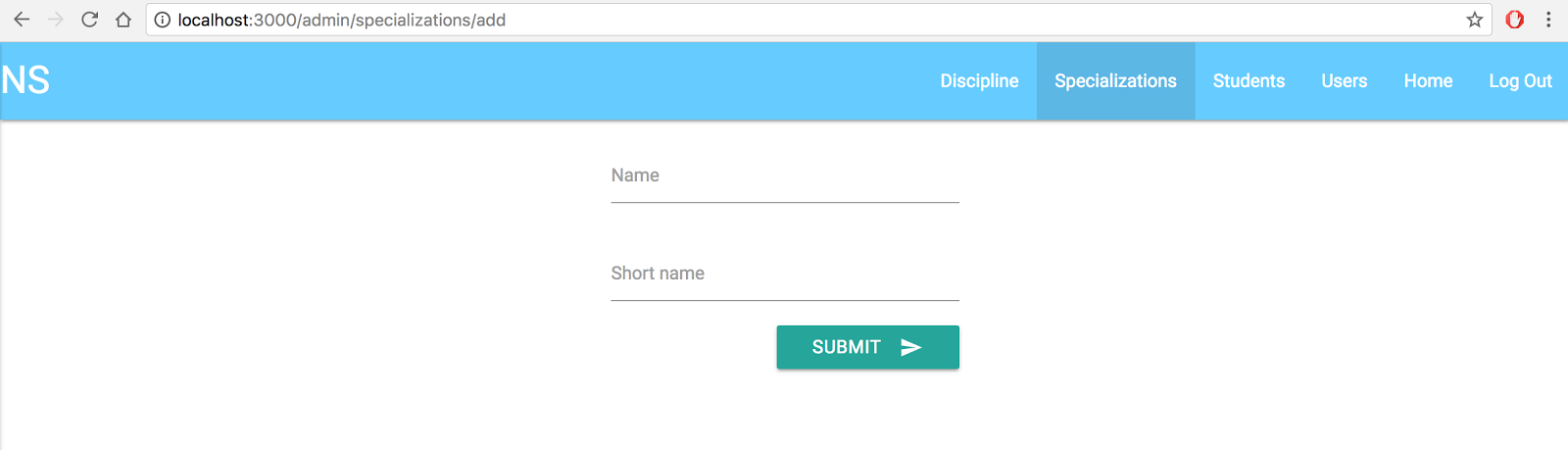


Figura 3.4 Exemplu de poziționare a butonului sub formular

În cadrul acestei aplicații se poate observa repetabilitatea unui model de a utiliza butoane pe care scriu submit cu un avion în partea dreapta pentru înregistrarea unor valori ce provin din formular, un buton circular roșu cu un plus în mijloc pentru adăugarea de studenți, discipline, utilizatori, specializări, un creion pe un fundal verde pentru editare și un coș de gunoi pe fundal verde pentru ștergere, toate acestea fiind intuitive utilizatorilor referitor la operațiile pe care trebuie să le îndeplinească.

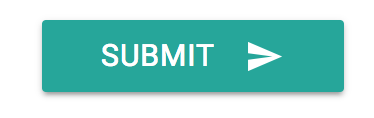


Figura 3.5 Buton pentru înregistrare valori din formulare



Figura 3.6 Buton pentru adăugare



Figura 3.7 Buton pentru editare

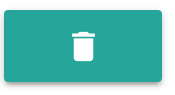


Figura 3.8 Buton pentru ștergere



Figura 3.9 Buton pentru note

1. **Linkuri defecte**

Este important ca toate url-urile existente să fie funcționale, deoarece un link ce afișează o pagină goală este asociată din punctul de vedere al utilizatorului de rând cu o eroare și de asemenea îi induce o stare de susceptibilitate [26]. Este la fel de important pentru utilizatori ca în momentul în care sunt redirectați din diverse motive de pe o pagină pe alta să fie informați cu privire la ceea ce se întamplă, deoarece devine frustrant pentru ei să nu primească informațiile pe care au încercat să le acceseze.

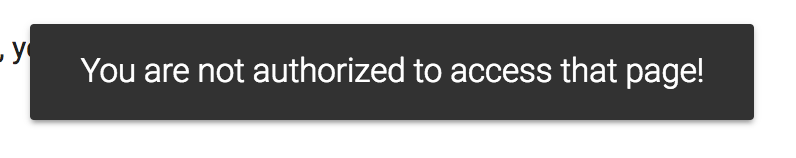


Figura 3.10 Notificare pentru atenționare acces nepermis

1. **Pagini mari consumatoare de timp**

Este bine știut faptul ca în epoca vitezei în care trăim timpul este foarte prețios și odată cu dezvoltarea puternică a tehnologiei au apărut foarte multe sisteme software cu o viteză foarte bună. Este important ca utilizatorul să beneficieze de o tranziție fluentă între pagini, tocmai de acea este de preferat ca timpul de acces la orice conținut să fie cât mai mic. Aspect ce este îndeplinit cu succes datorită tehnologiilor noi, folosite.

## Utilizarea aplicației

Am tot vorbit pe parcursul acestei lucrări despre diversele implementări în materie de cod, despre interfețe grafice, diferite stiluri, librării utilizate și tehnicile de implementare, însă a venit timpul pentru a demonstra toate cele menționate mai sus și pentru ce au fost necesare toate orele de studiu și lucru. În continuarea acestei lucrări voi prezenta imagini din cadrul aplicației și explicații aferente pe baza acestora. Voi face un tur virtual de la înregistrarea unui student pe site și până la administrarea platformei de către un administrator.

Înainte de a începe explicațiile aferente trebuie să menționez faptul că în momentul realizării acestui document sistemul a fost rulat pe un server local, fapt pentru care url-ul de bază, la care va răspunde aplicația client este <http://localhost:3000/>, iar aplicația server <http://localhost:8080/>. După instalarea să pe un server ce va fi disponibil pe internet îi poate fi atribuit și un DNS (Domain Name Server).

### Înregistrarea studentului pe site

Pentru ca un student să se poată înregistra, acesta trebuie să fie înregistrat la universitate și astfel să dețină un număr matricol. Primul pas pe care trebuie să îl facă un student este să acceseze url-ul <http://localhost:3000/> și va fi redirecționat automat către pagina de login <http://localhost:3000/login> așa cum am precizat mai sus în partea de implementare.

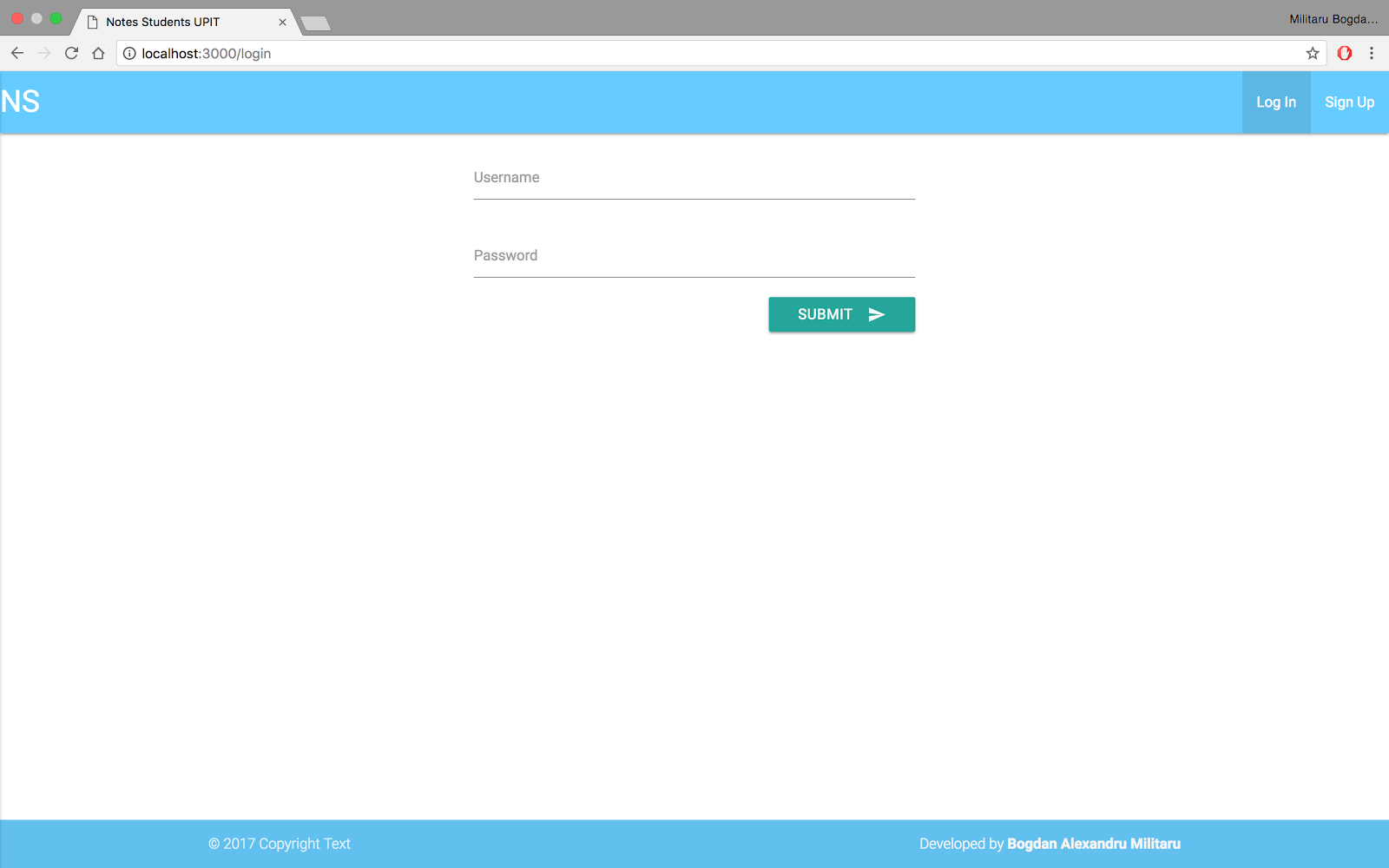


Figura 3.11 Pagina de logare

În continuare în colțul din dreapta al ferestrei, pe bara de navigare poate fi observat butonul *Sign Up*, se face click pe acesta pentru a fi redirecționați către pagina de înregistrare a unui utilizator. Înregistrarea este simplă și actualizată noilor standarde, fiind realizată în 2 pași. În primul pas utilizatorul trebuie să ofere încredere sistemului că este o persoană credibilă și să insereze numărul matricol.

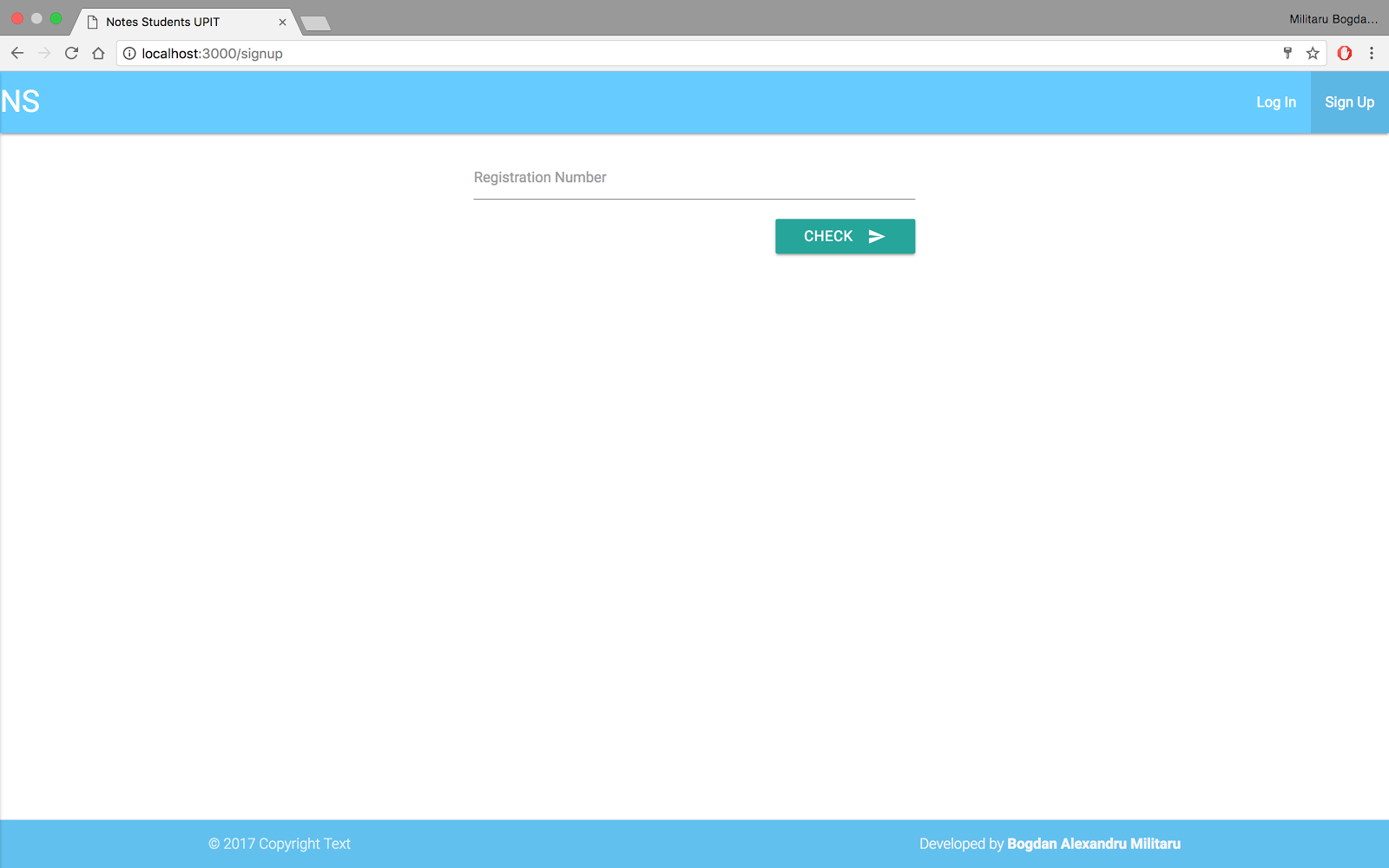


Figura 3.12 Înregistrare utilizator, pasul 1

După introducerea numărului matricol în câmpul *Registration Number*, este necesară apăsarea butonului *CHECK*, dacă există student înregistrat în baza de date cu numărul matricol introdus, se va trece la pasul 2 conform imaginii 3.14, altfel o notificare va fi afișată în colțul din dreapta, conform imaginii 3.13.

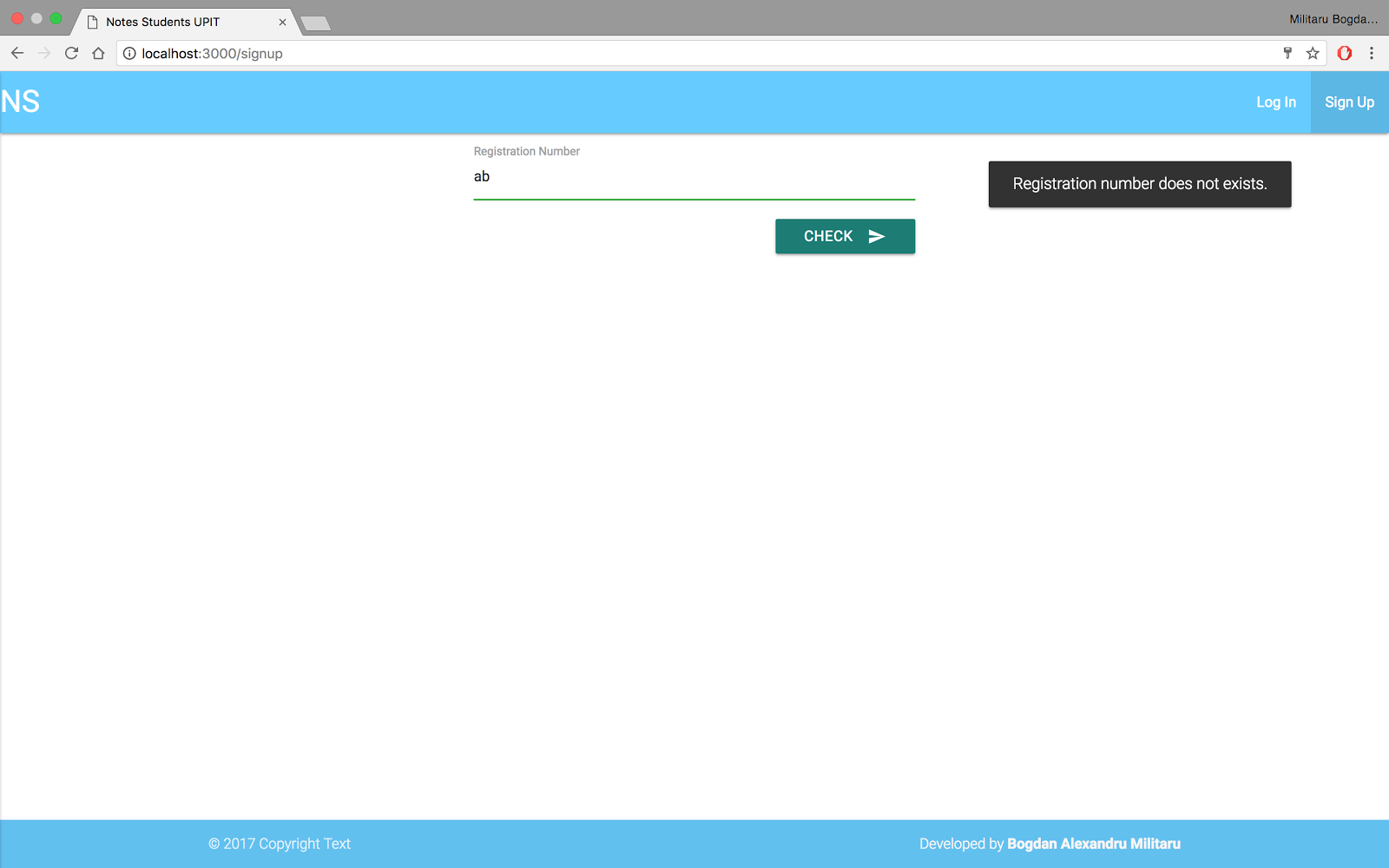


Figura 3.13 Înregistrare utilizator, eroare număr matricol incorect

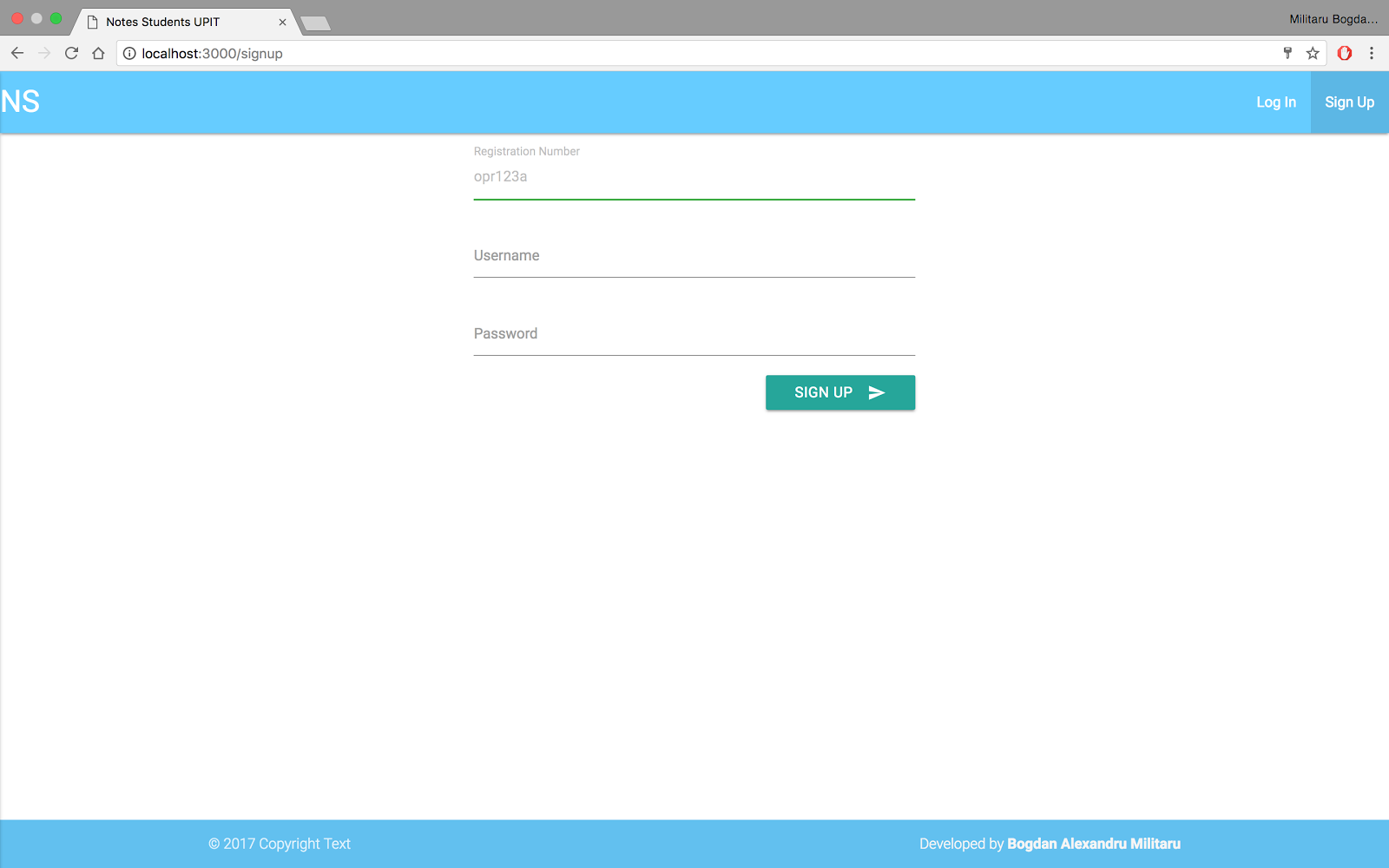


Figura 3.14 Înregistrare utilizator, pasul 2

În cazul în care utilizatorul este valid se solicită numele de utilizator dorit și o parolă, apoi se apasă butonul *SIGN UP*. În acest moment utilizatorul introdus este verificat dacă există un duplicat în baza de date, iar în cazul în care există, o notificare va fi afișată în colțul din dreapta sus, figura 3.15, altfel o notificare de succes va fi afișată și apoi utilizatorul va fi redirecționat către pagina de logare.

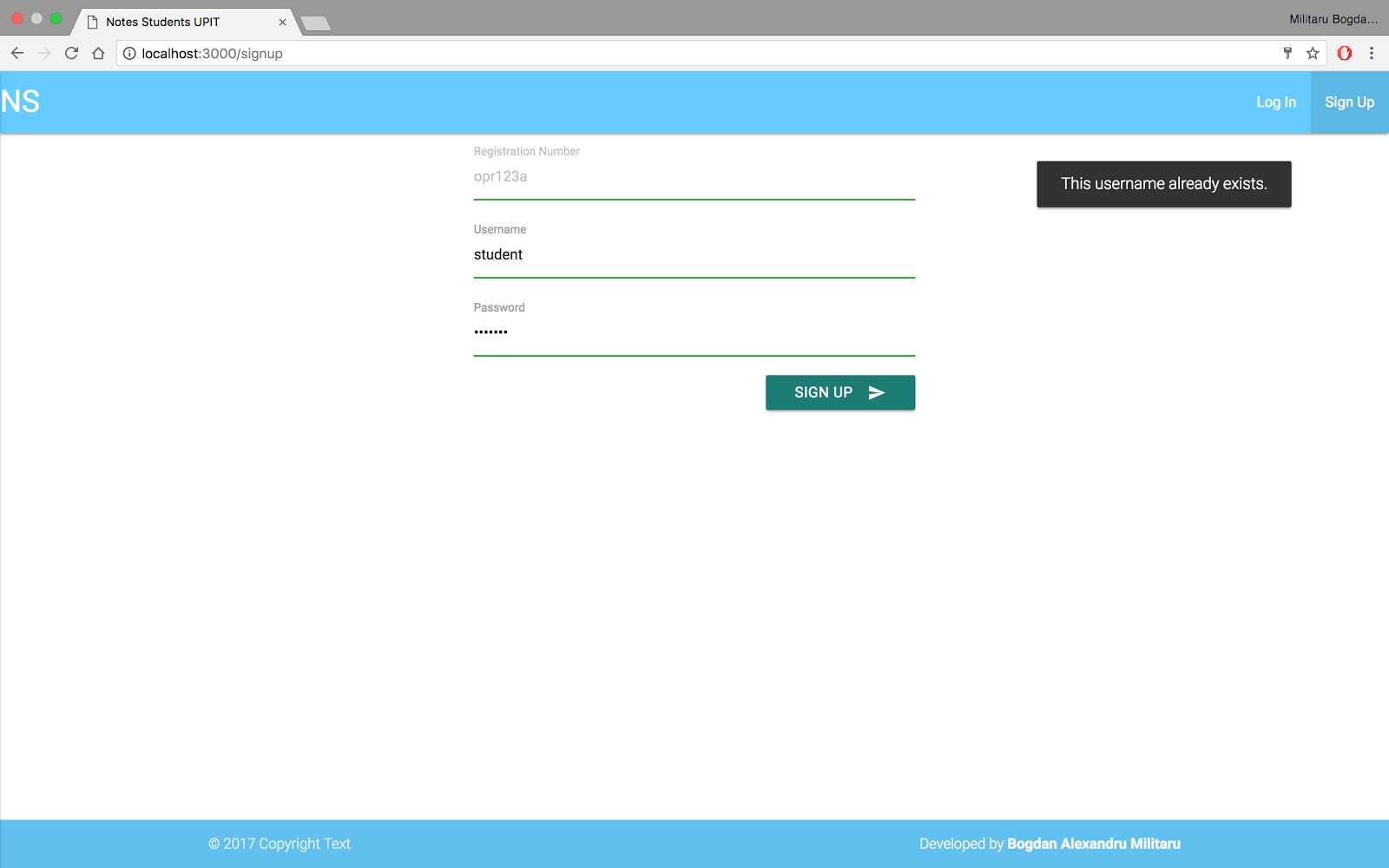


Figura 3.15 Înregistrare utilizator, username-ul există

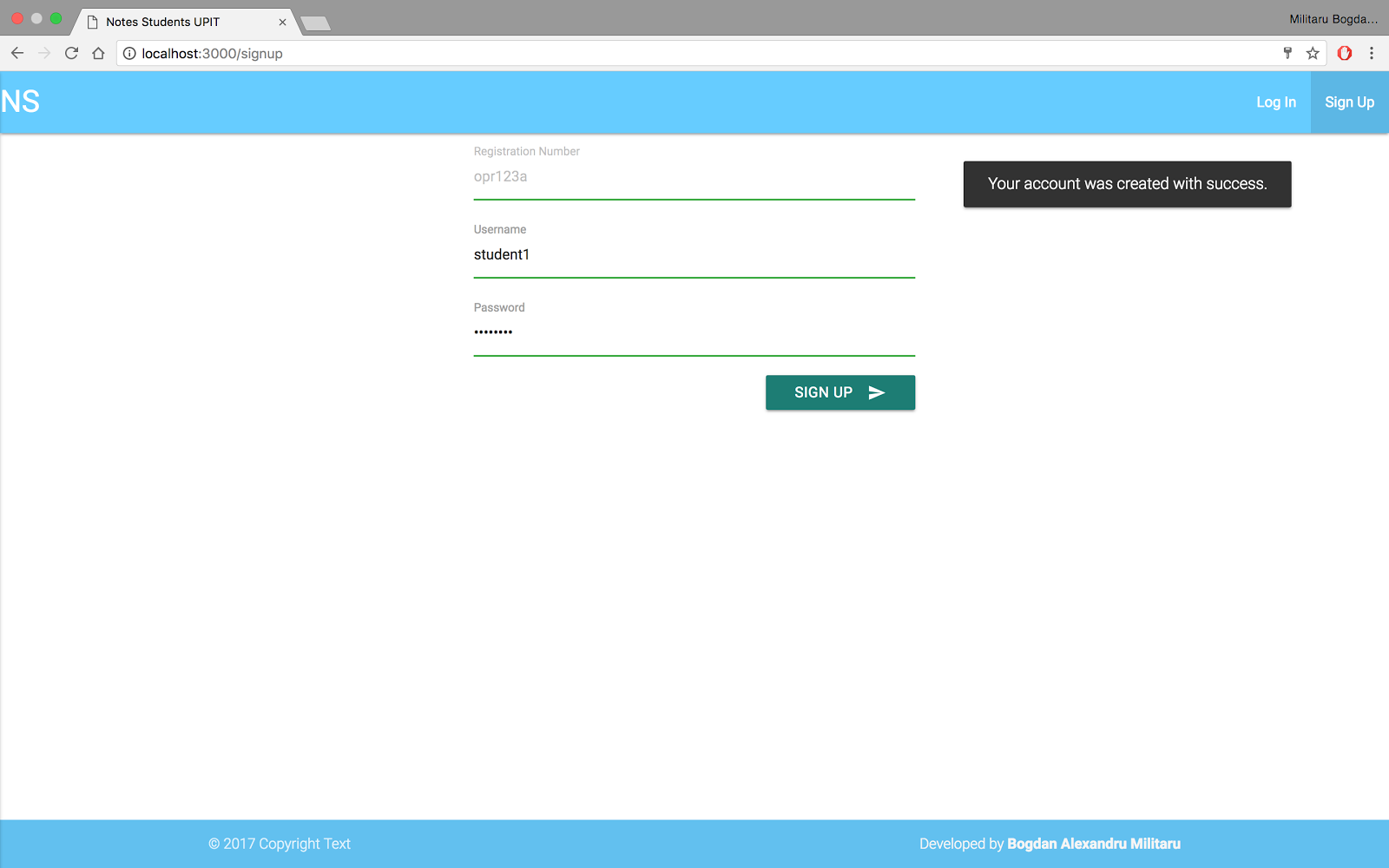


Figura 3.16 Înregistrare utilizator cu success

### Logare Student

Suntem la pasul în care utilizatorul are un cont și dorește să se logheze pe site. Fie că este vorba despre un administrator sau un student procedeul este același. Este suficientă accesarea url-ului de bază <http://localhost:3000/>. Se va face un redirect deoarece nu este logat către link-ul [http://localhost:3000/login/](http://localhost:3000/) și va fi afișată o pagina ca în figura 3.11. Trebuiesc completate cele două câmpuri cu username-ul și parola setate la pasul de înregistrare. În cazul în care numele de utilizator este introdus greșit o notificare corespunzătoare va fi afișată - figura 3.17, dacă parola este introdusă corect, un mesaj corespunzator va fi afișat și de această dată - figura 3.18.

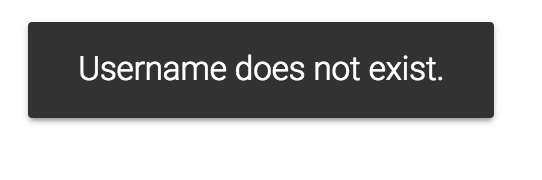


Figura 3.17 Logare site, utilizatorul nu există

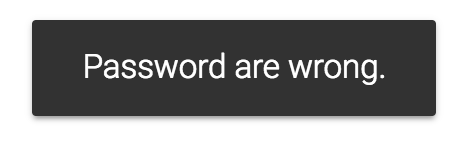


Figura 3.18 Logare site, parola greșită

În cazul în care logarea se efectuează cu succes, utilizatorul va fi redirecționat pe pagina home, <http://localhost:3000/home> și un mesaj de notificare va fi afișat conform figurii 3.19.

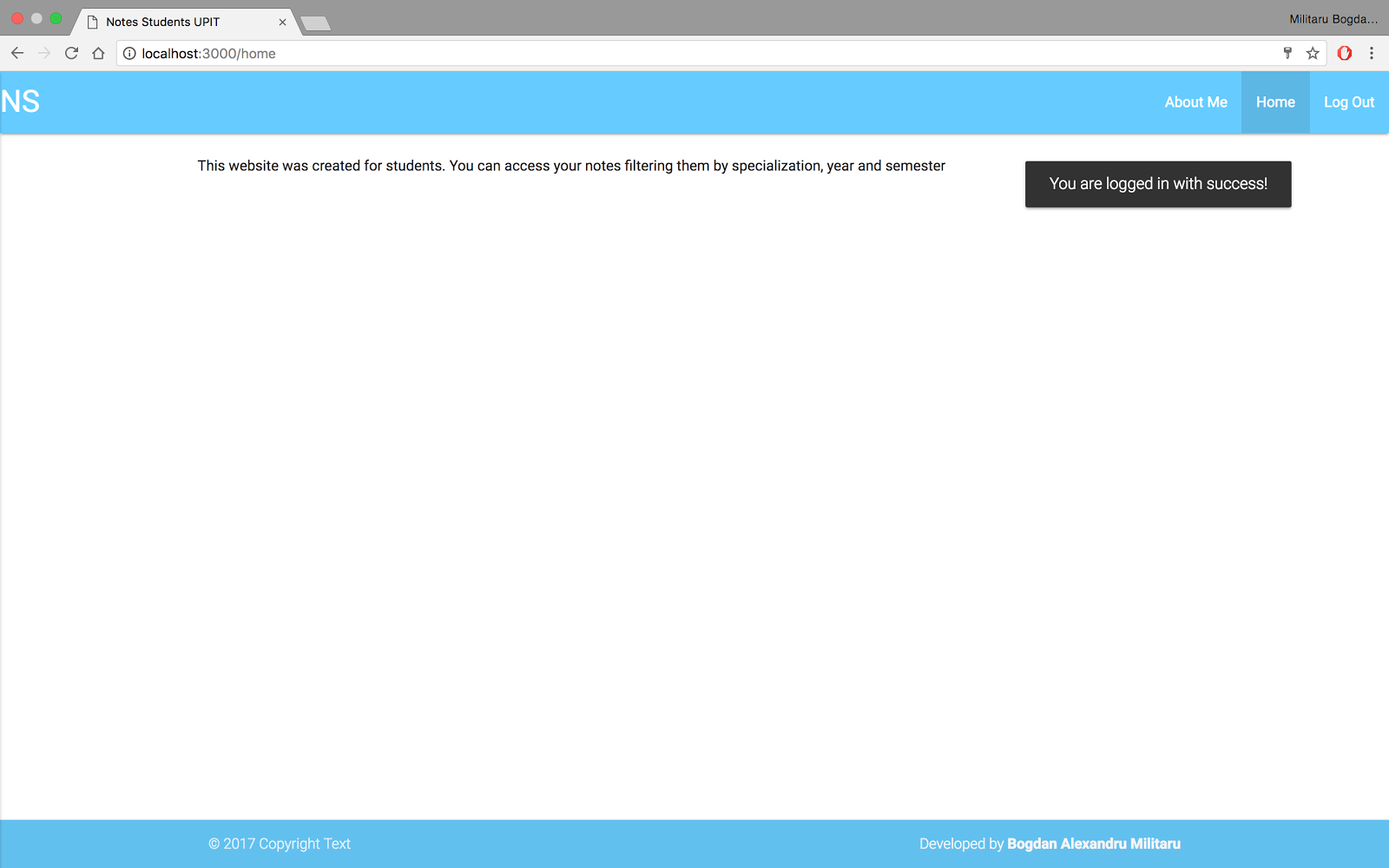


Figura 3.19 Pagina home, cont student

Contul curent este un cont de student, acesta are posibilitatea de a se deloga de pe acest cont prin click pe butonul *Log Out* din bara de meniu, colțul dreapta sus, în acest caz fiind redirecționat către pagina de login, <http://localhost:3000/login> sau să intre pe pagina cu informațiile personale și note, prin click pe *About Me*, din bara de navigare. Vom fi redirecționati pe url-ul <http://localhost:3000/my-notes>, o pagină ca în figura 3.20 va fi afișată.



Figura 3.20 Pagina de note a studentului

După cum se poate observa în imaginea de mai sus în partea de sus a paginii există un chenar, în care este afișat textul *About You*, în cazul în care se da click pe acel chenar o suprafață cu informații despre studentul logat se va afișa, conform figurii 3.21. Se va afișa numele, numărul matricol, situația financiară a studentului dacă este la cu taxă sau la subvenție și specializarea. În cazul în care este înscris la mai multe specializări se vor afișa cu virgulă. Sub acest chenar există diferite butoane de selecție a specializării, anului și semestrului. Implicit sunt afișate toate notele studentului pentru o specializare.

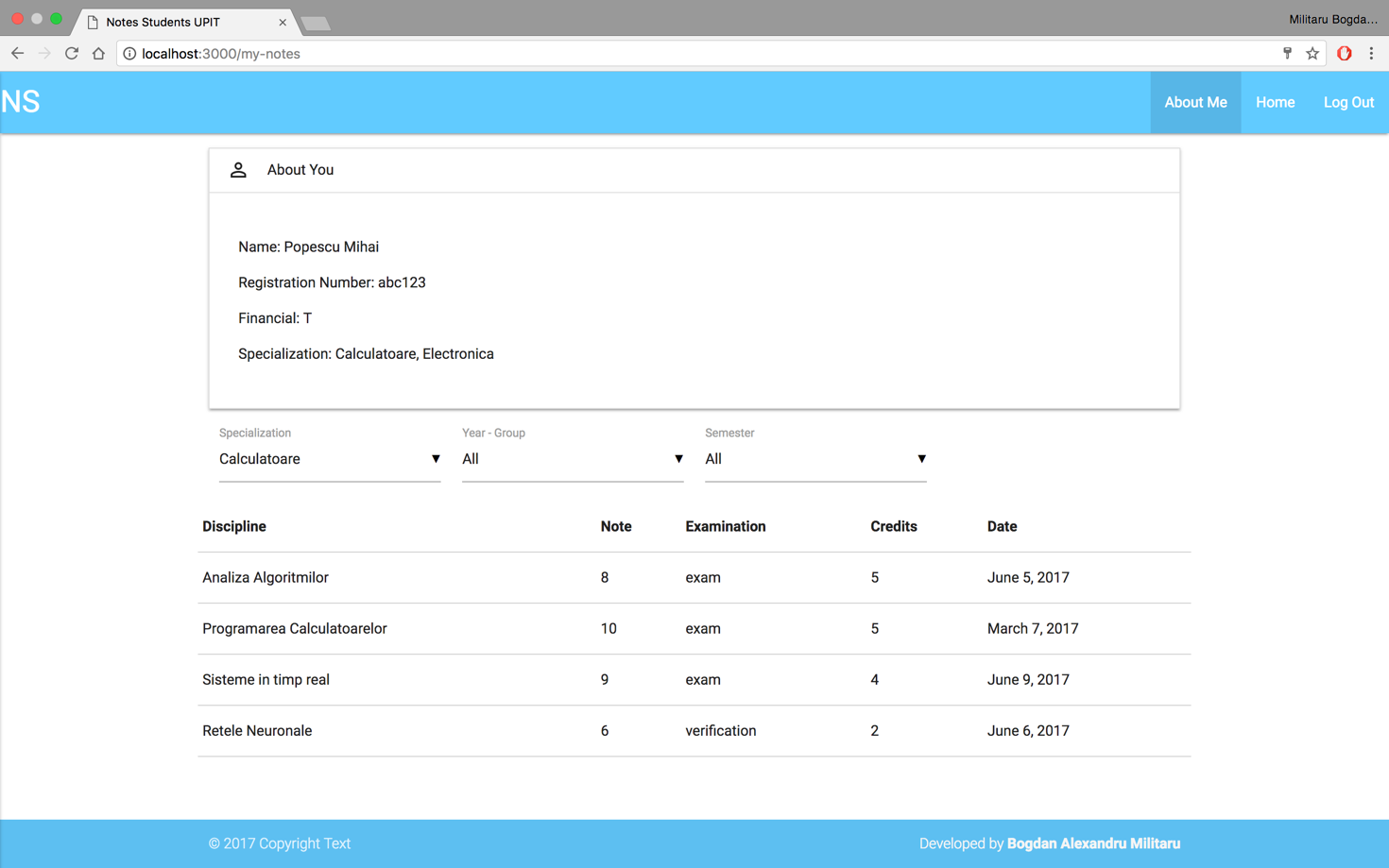


Figura 3.21 Suprafața extinsă cu informații despre student

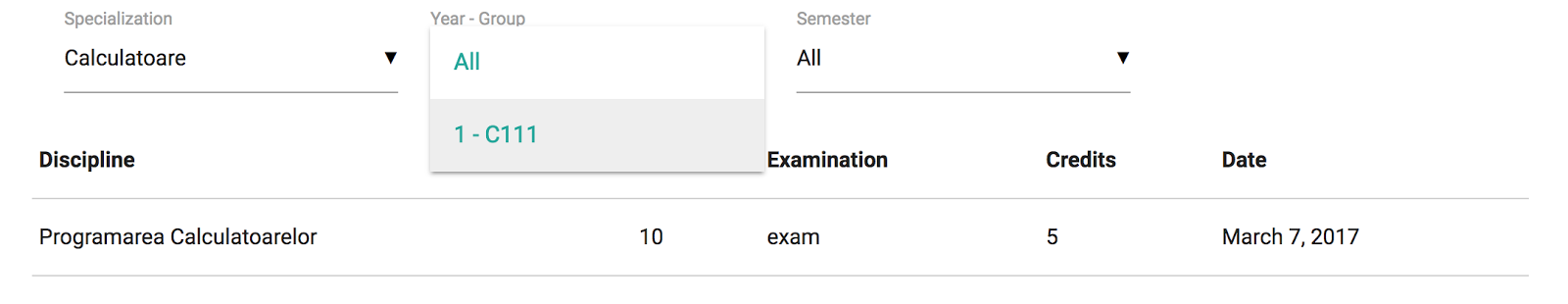


Figura 3.22 Selecția notelor ce se doresc afișate

### Administrare

Administratorul este persoana care va manageria conținutul site-ului, acesta va avea o interfață grafică asemănătoare cu cea a studentului, însă cu posibilitatea de a accesa mai multe secțiuni. Are posibilitatea de a edita discipline, specializări, studenți, note și utilizatori. Asemenea studentului după logare acesta este redirecționat pe pagina de home, iar în partea de sus a paginii poate accesa secțiunile din meniu conform figurii 3.23.

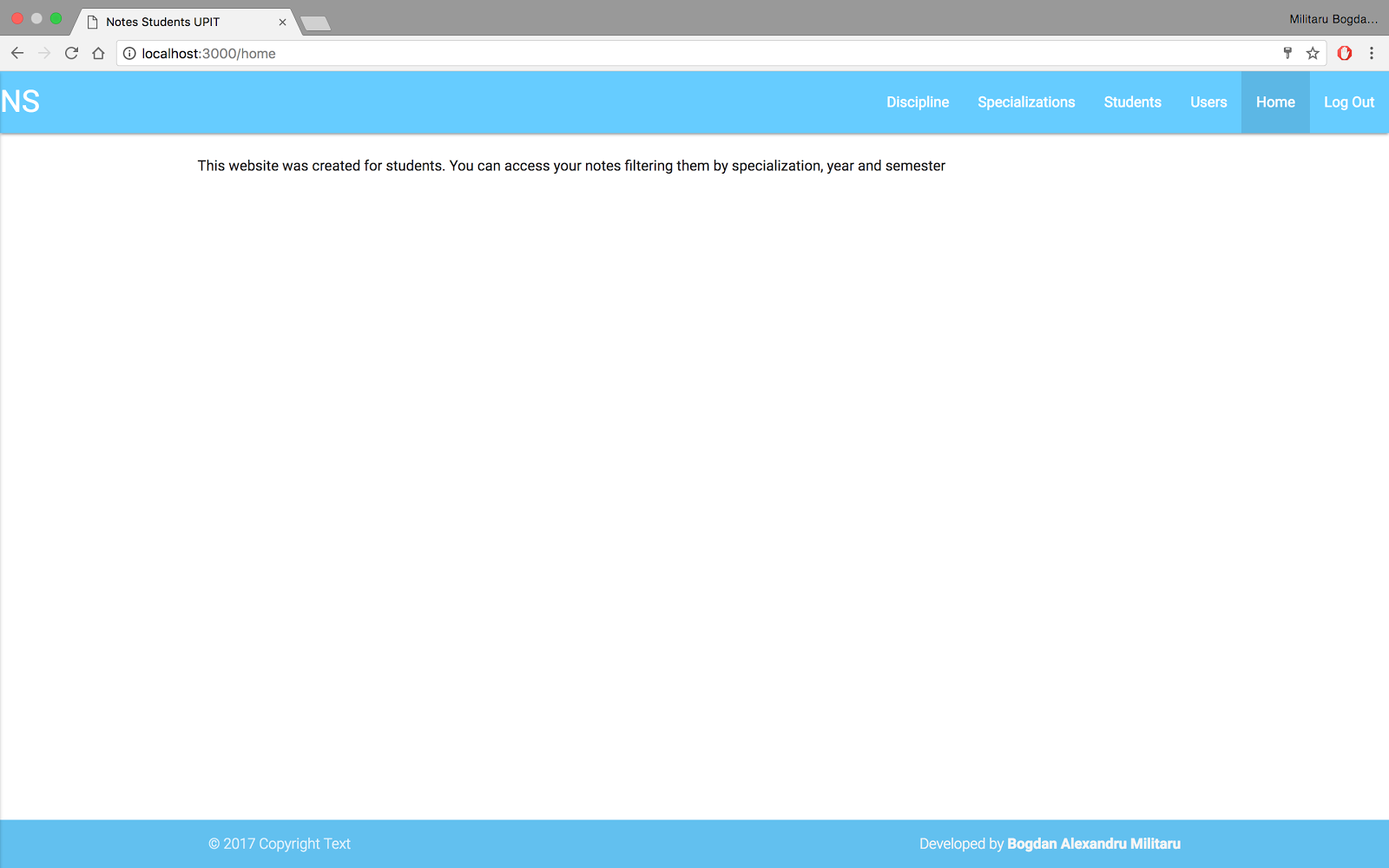


Figura 3.23 Pagina home, profil administrator

### Administrare Discipline

Pentru administrarea disciplinelor este nevoie să se acceseze link-ul <http://localhost:3000/admin/disciplines> sau administratorul trebuie să dea click pe *Discipline*, din bara de meniu și o fereastră nouă va fi deschisă, conform figurii 3.24.

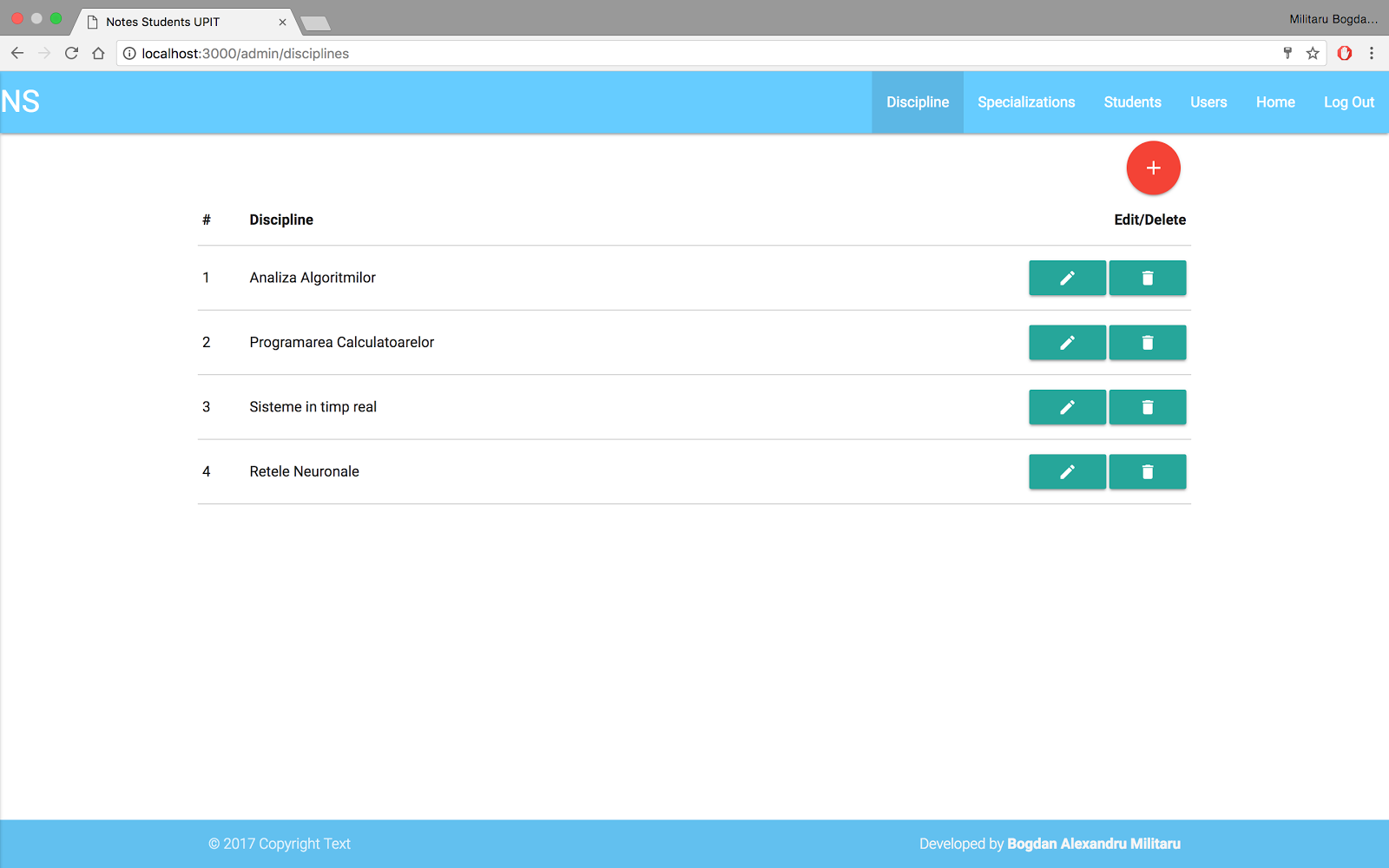


Figura 3.24 Administrare Discipline

În imaginea de mai sus se poate observa un tabel în care sunt afișate disciplinele. În coloana 1 este afișat index-ul din tabela, apoi numele disciplinei, iar în partea dreptă pot fi identificate două butoane, de editare și stergere. Pentru adăugarea unei noi discipline se apasă pe butonul roșu din partea dreaptă a imaginii și vom fi redirecționați către pagina <http://localhost:3000/admin/disciplines/add> asemănătoare din punct de vedere al conținutului cu <http://localhost:3000/admin/disciplines/edit> afișată în figura 3.25. Aceasta conține mai multe câmpuri ce pot fii inserate: numele disciplinei va fi disponibil doar în modul editare, deoarece în modul adăugare nu există până în momentul în care nu se apasă butonul Submit, numele disciplinei, numele scurt al disciplinei, cărui an îi aparține disciplina, forma de examinare, afișată ca un meniu derulant, numărul punctelor de credit, semestrul și specializarea căreia aparține.

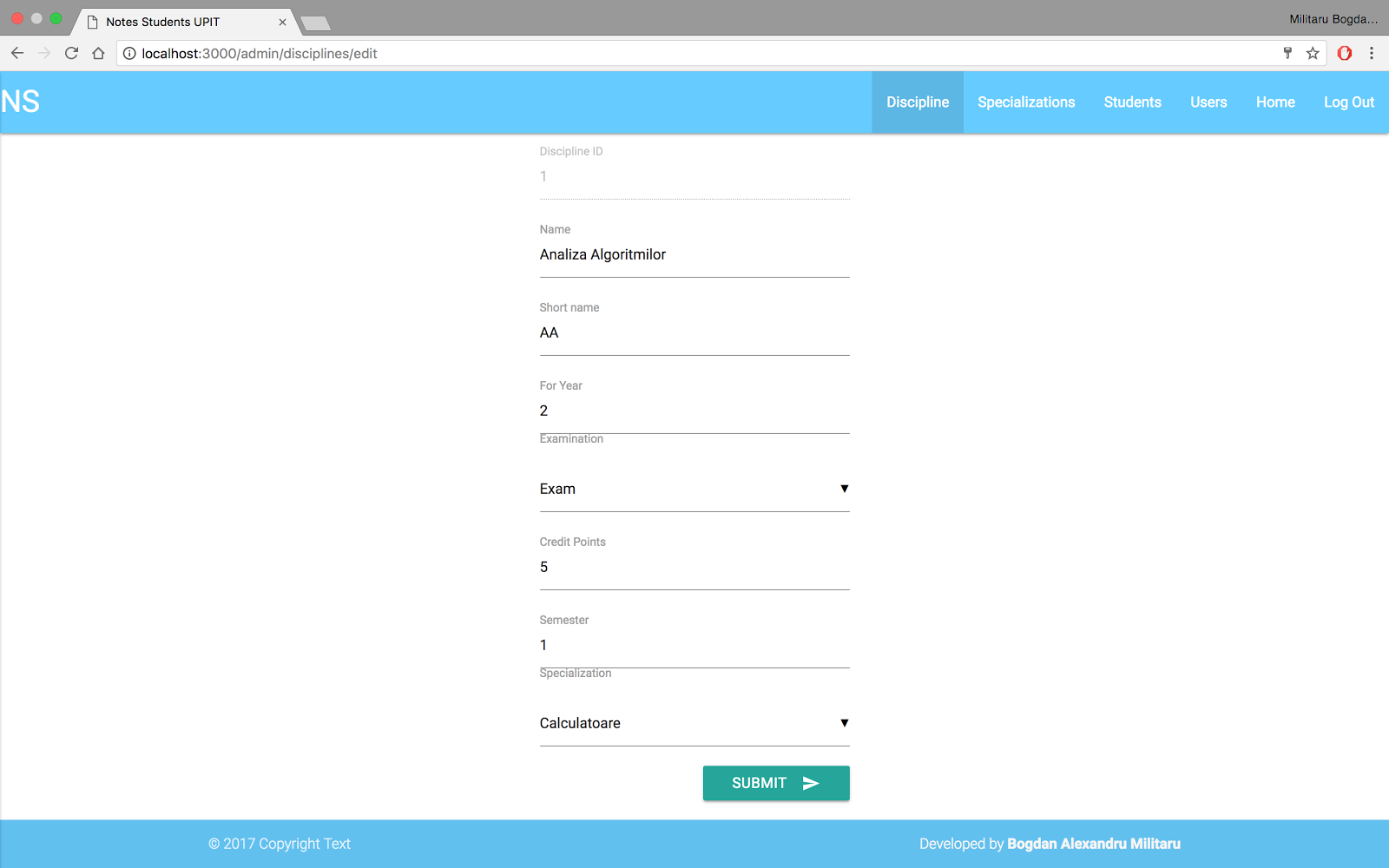


Figura 3.25 Pagina Adăugare/Editare Discipline

Așa cum am precizat și anterior, la implementarea modului de ștergere, în cazul în care administratorul va da click pe iconița de ștergere, se va afișa un pop-up, de tip Confirm. Daca acesta va da click pe butonul Confirm, înregistrarea curentă va fi ștearsă.

### Administrare Specializări

Pentru administrarea specializării este necesar să se meargă pe adresa url <http://localhost:3000/admin/specializations> sau click pe butonul Specializations din bara de meniu și se va deschide o fereastră conform figurii 3.26. Aici vor fi afișate într-un tabel în formatul: numărul specializării din baza de date, numele specializării și două butoane verzi, unul pentru editare și unul pentru ștergere, ambele poziționate în partea dreaptă a ecranului. Pentru adăugarea unei înregistrări noi se va da click pe butonul roșu din colțul din dreapta și apoi utilizatorul va fii redirecționat către pagina de adăugare cu adresa, <http://localhost:3000/admin/specializations/add>, aceasta având același template ca și pagina de editare afișată în figura 3.27.

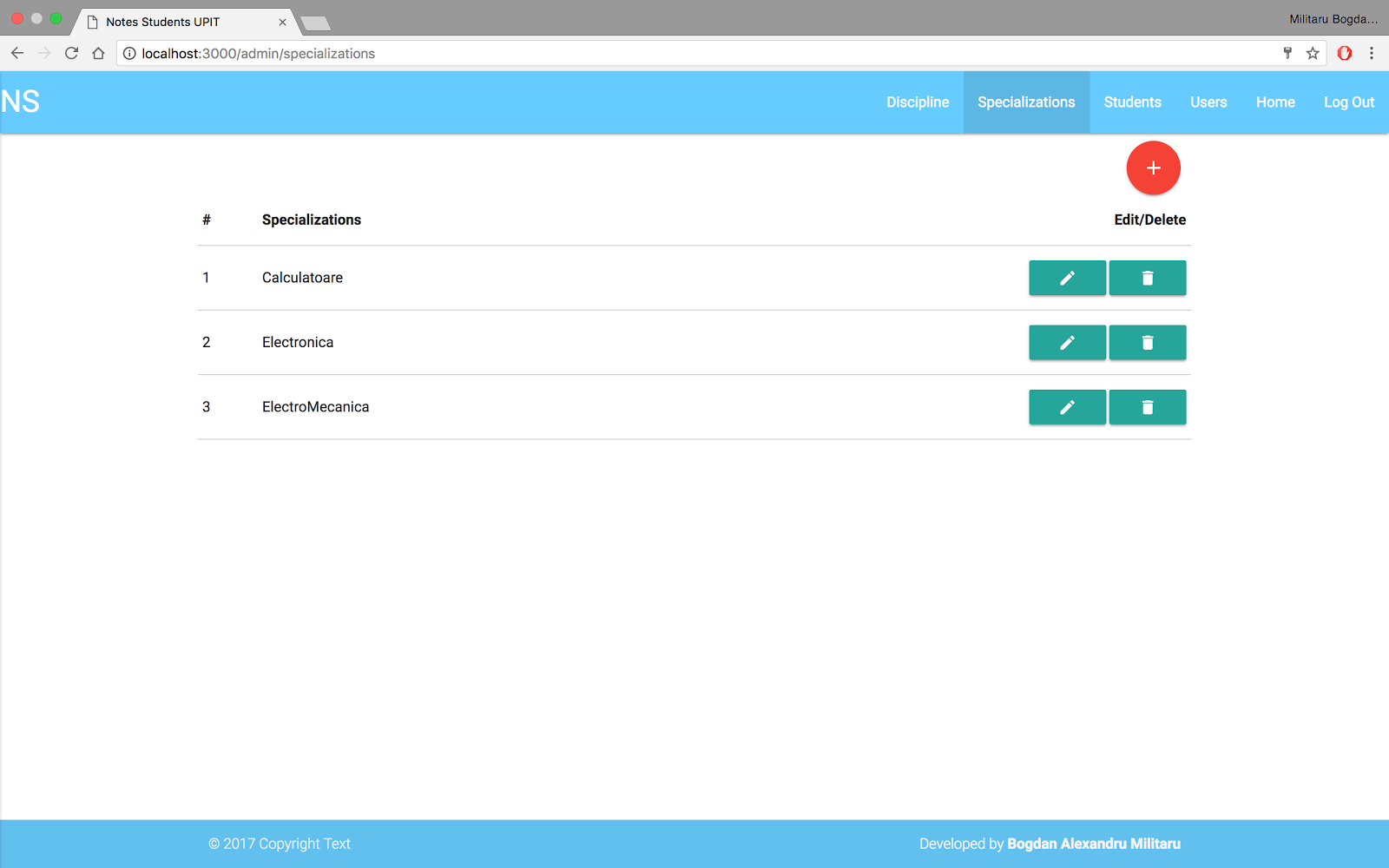


Figura 3.26 Pagină de vizualizare Specializari

Se poate observa același template prezentat anterior în cazul Disciplinelor. Pagina de editare va avea url-ul <http://localhost:3000/admin/specializations/edit> conform figurii 3.27.

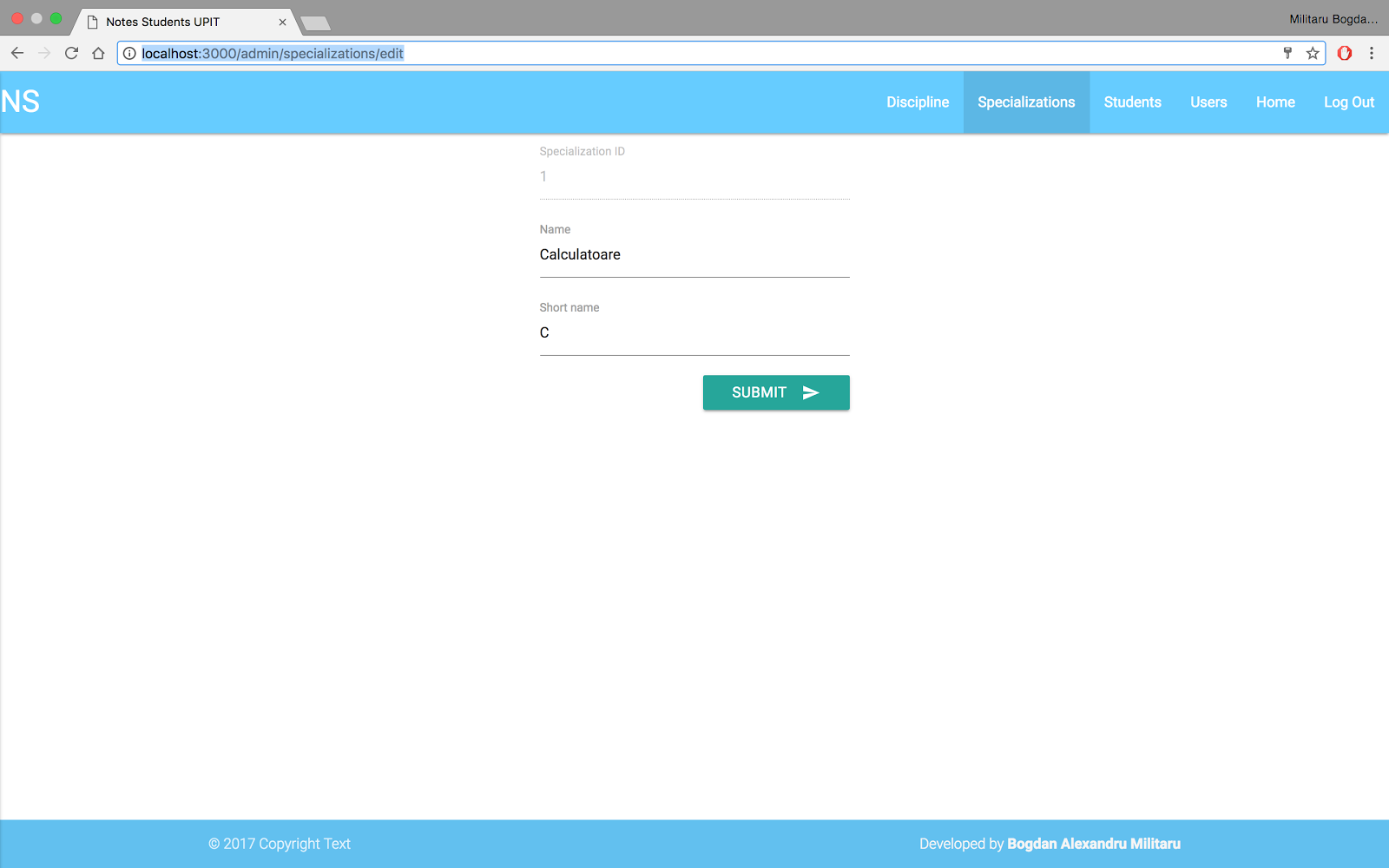


Figura 3.27 Pagină Adăugare/Editare Specializare

În cazul în care se modifică o valoare și se dă click pe butonul *Submit*, modificarea se va afișa în timp real și un mesaj de succes va fi afișat.

### Administrare Studenți

Dacă până acum totul a arătat similar în ceea ce privește interfața și administrarea, în aceasta zonă apar câteva elemente noi. Pagina de administrare studenți poate fi accesată din bara de meniu sau apelând link-ul <http://localhost:3000/admin/students>.

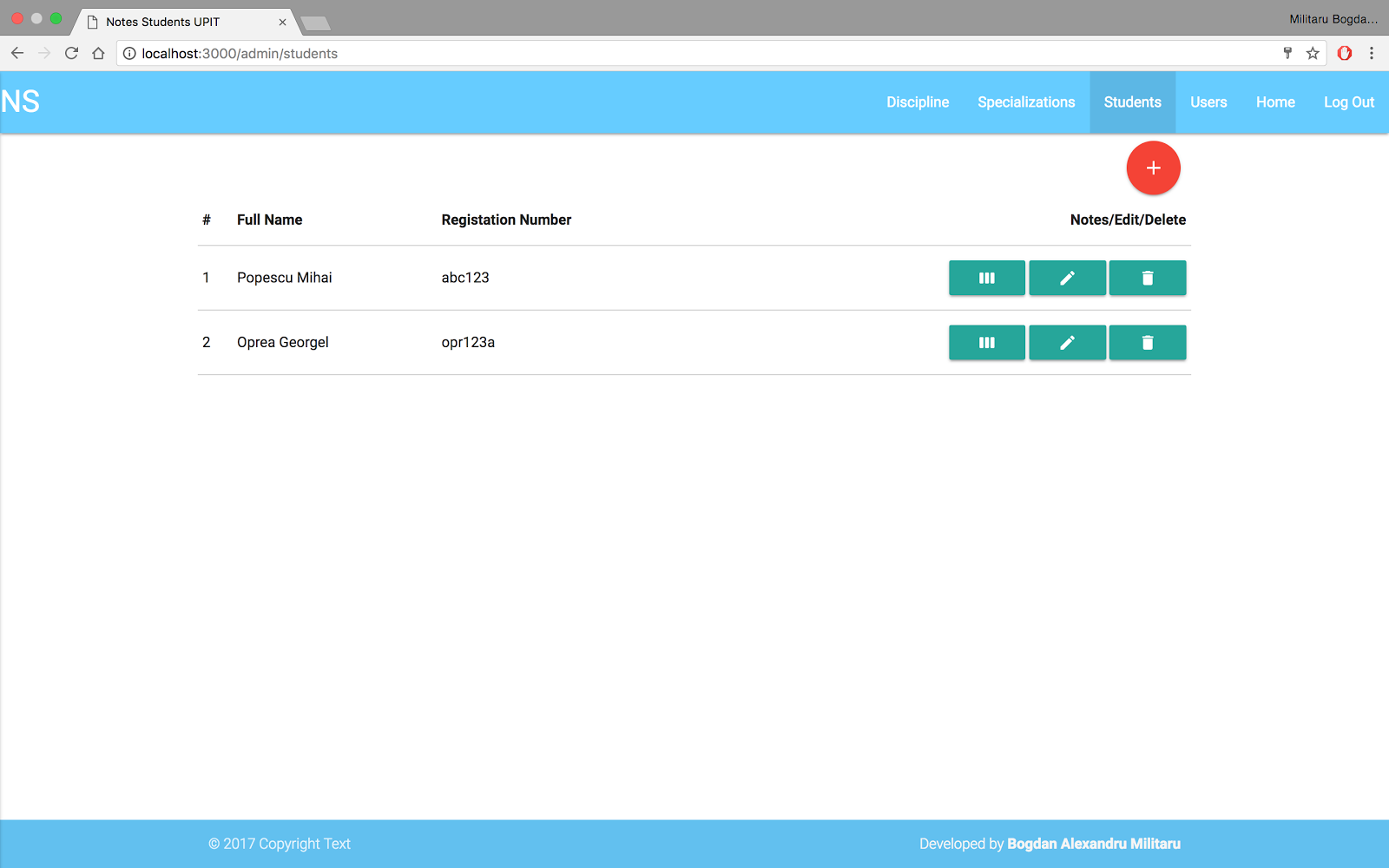


Figura 3.28 Pagină afișare studenti

Așa cum reiese din figura anterioara se poate observa o listă a tuturor studenților înregistrați cu nume și număr matricol, iar în partea dreaptă pe lângă clasicele butoane de ștergere, editare, apare un al treilea, cu 3 linii verticale pe un fundal verde care redirecționează administratorul către pagina de administrare a notelor pentru studentul în dreptul căruia a fost apăsat butonul.

În cazul în care se apasă pe butonul roșu cu plus, se va face un redirect către pagina de adăugare student Figura 3.29, figură în care se poate observa împărțirea ecranului în două zone, delimitate printr-un spațiu alb. O secțiune pentru datele studentului în partea stângă, în care se vor preciza numele, numărul matricol și situația financiară, iar în partea dreaptă studentul poate fi înscris la una sau mai multe specializări.

Dacă se apasă butonul plus din colțul dreapta o nouă specializare va fi adăugata. Specializările vor fi adăugate sub forma de lista, cu posibilitatea de a se expanda, oferind informații suplimentare precum grupele din care acel student a făcut parte, conform figurii 3.30.

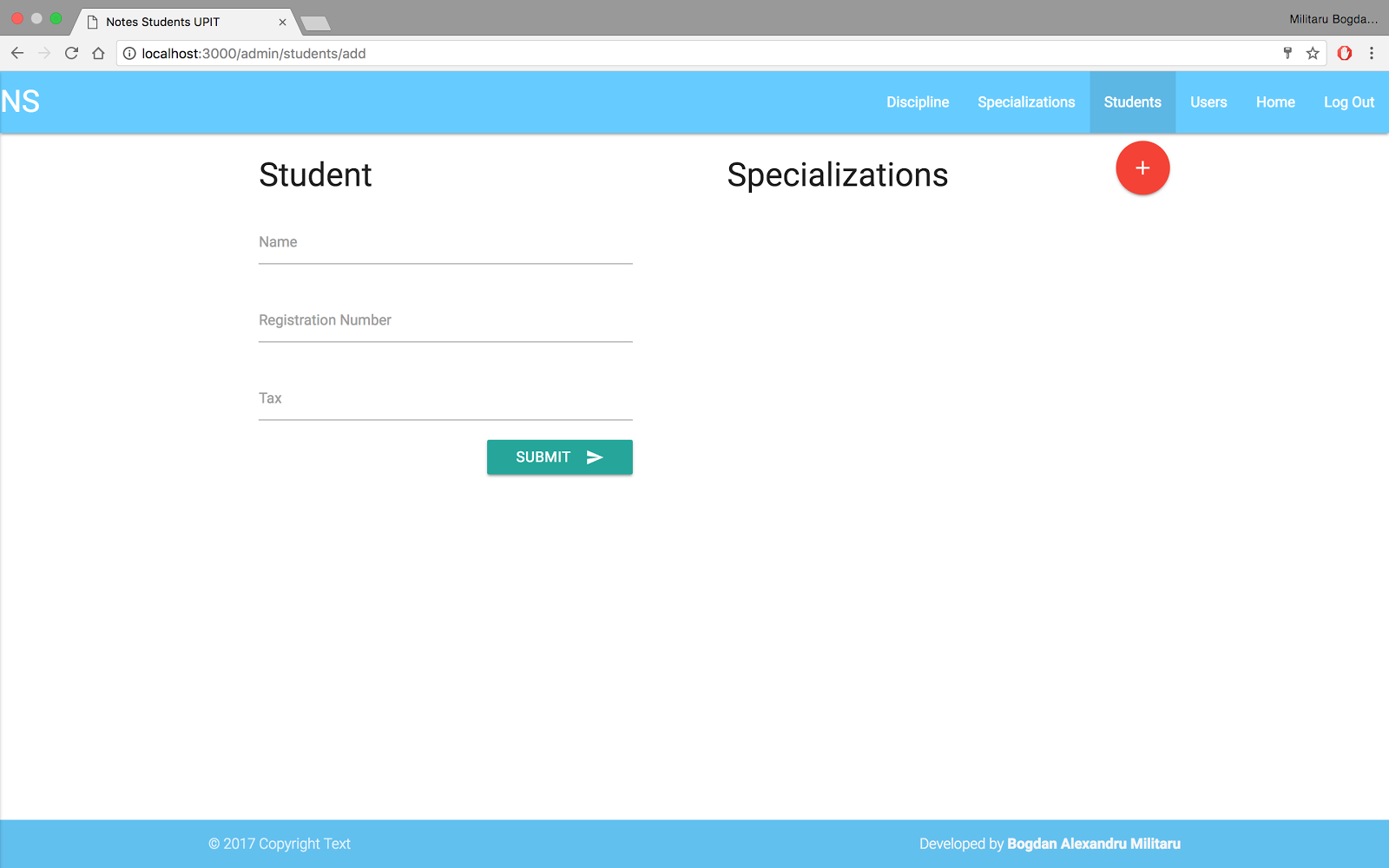


Figura 3.29 Pagină adăugare student



Figura 3.30 Pagină editare student

Se poate observa în colțul din dreapta jos un buton de ștergere. În cazul în care se acționează acel buton, studentul va fi eliminat din cadrul acelei specializări. Tot în figura 3.30 se poate observa în secțiunea expandată titlul *Groups*, iar sub aceasta în același format ca pentru specializări o lista cu grupe. În cazul în care se dă click pe o grupa, spre exemplu în cazul de fata *C111*, se va deschide o nouă secțiune expandată ce va permite editarea datelor despre acea grupă, conform figurii 3.31. Butonul de plus din dreptul titlului Groups, permite adăugarea altor grupe printr-un procedeu foarte simplu, ce poate fi observat în figura 3.32.

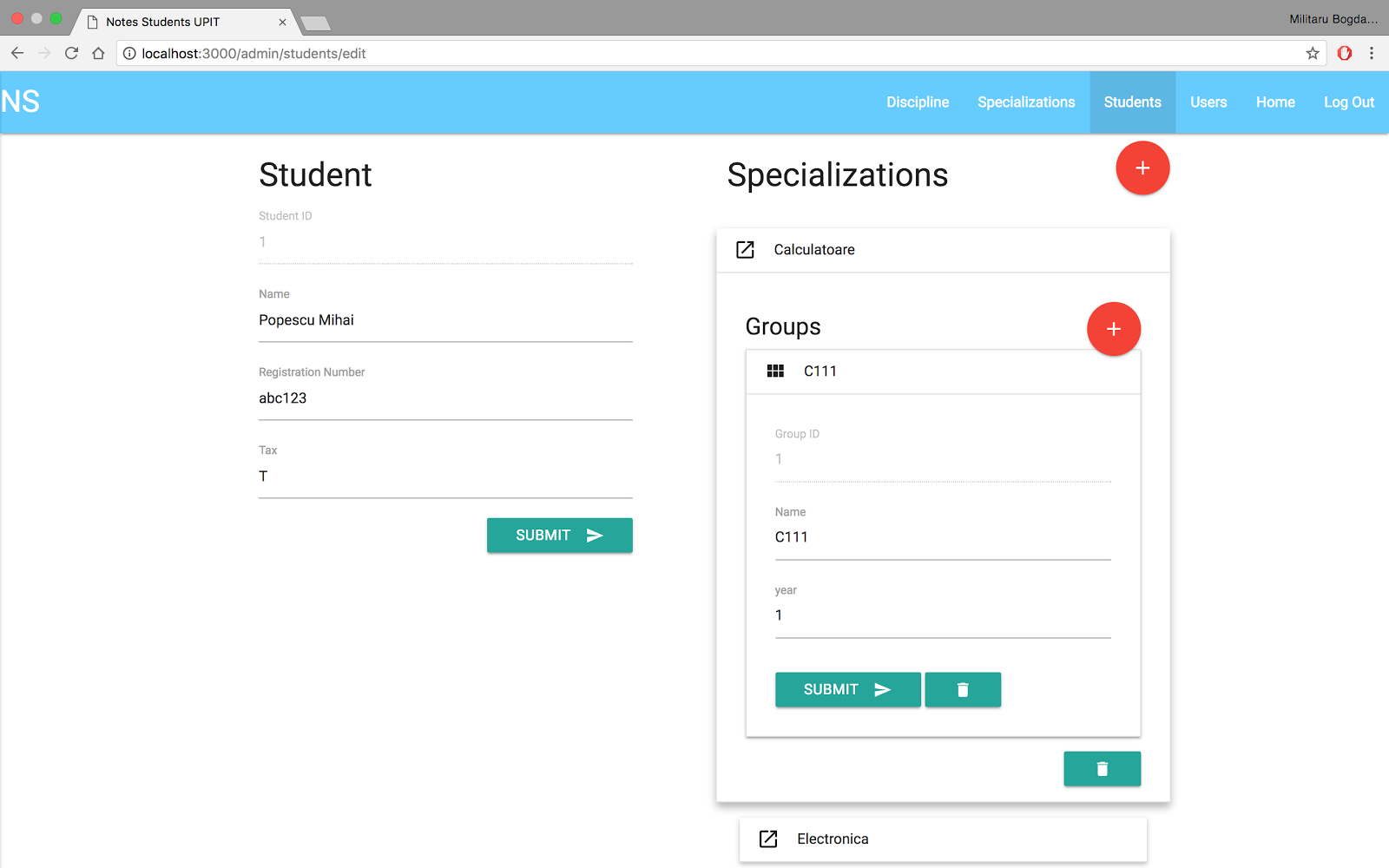


Figura 3.31 Editare detalii grupa pentru studentul administrat curent

În secțiunea de editare din figura anterioară se pot remarca zonele disponibile pentru editare, *Name* și *Year*. Câmpul *Group ID* este dezactivat deoarece acesta nu poate fi modificat, iar în partea de subsol a grupei curente, ce se dorește editată există un buton de Submit și unul de ștergere. Prin acționarea butonului de ștergere studentul va fi șters din cadrul grupei curente, iar prin acționarea butonului *Submit*, modificările efectuate asupra grupei vor fi efectuate.

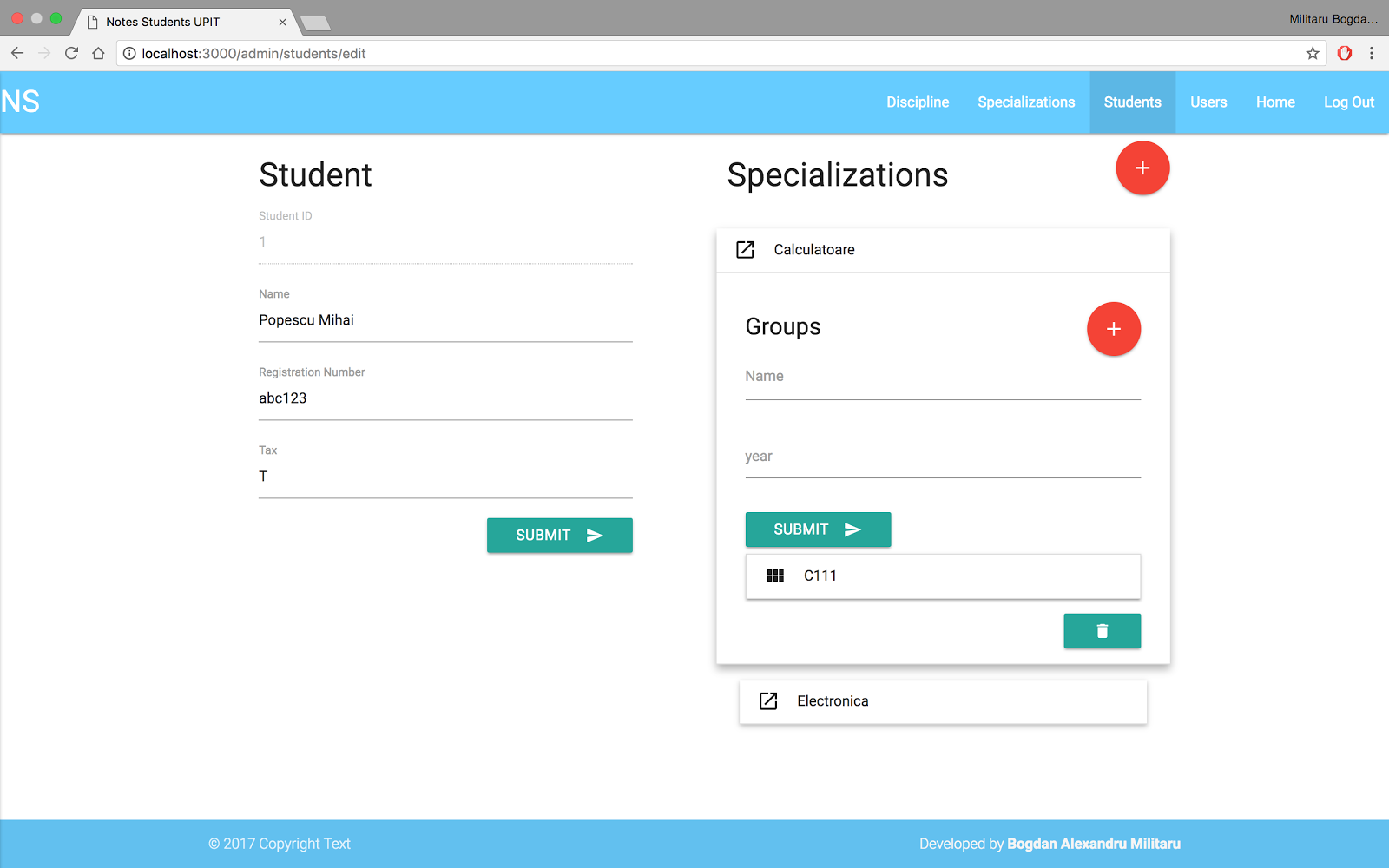


Figura 3.32 Înscrierea studentului într-o grupă

Dupa apăsarea butonului plus din dreptul titlului *Groups* va apărea această nouă fereastră cu posibilitatea de a introduce numele unei noi grupe și anul de studiu. După adăugarea datelor și apăsarea butonului *Submit*, grupa va fi salvată și o notificare va fi afișată în colțul din dreapta sus.

Am ales acest mod de editare al studenților, deoarece am considerat că este de preferat ca toate informațiile să existe într-o singură pagină, administrarea lui fiind mult mai ușoară. De asemenea am ascuns informațiile irelevante, pentru a nu produce confuzie prin aglomerarea paginii cu o sumedenie de informații.

### Administrarea notelor

În subcapitolul anterior am amintit despre posibilitatea de a administra notele unui student prin click pe butonul verde cu trei linii verticale din dreapta numelui studentului de pe pagina <http://localhost:3000/admin/students>, iar apoi vom fi redirecționati pe pagina notelor <http://localhost:3000/admin/notes>, figura 3.33. Este exclusă posibilitatea de a ajunge pe această pagină în alt mod, deoarece după cum se poate observa url-ul este curat și nu prezintă alte informații suplimentare, care să identifice studentul asupra căruia se doresc administrate notele. În cazul în care se încearcă accesarea acestui url prin introducerea lui în bara de link-uri, se va face un redirect către pagina de studenți pentru a selecta întâi un student.

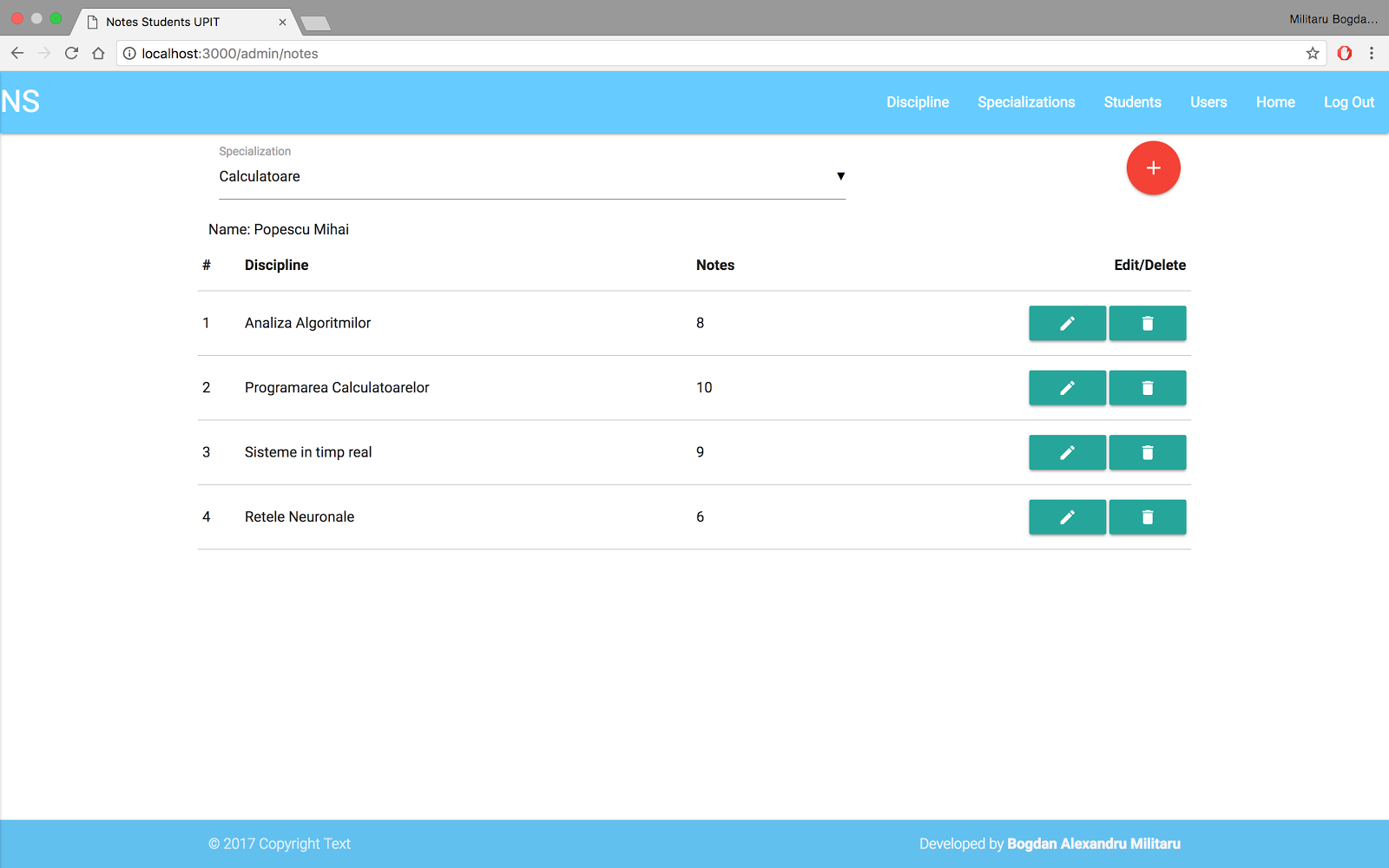


Figura 3.33 Pagină administrare note student

Pe pagina de administrare a notelor se poate remarca în partea de sus a paginii un selector *Specialization*, de unde administratorul poate selecta specializarea asupra căreia se doresc administrate notele studentului curent. Sub acest buton de tip dropdown se poate observa numele studentului editat, iar în partea dreaptă butonul de adăugare note. În continuarea paginii se pot observa toate notele pe care studentul le are cu posibilitatea de a edita sau șterge o notă. Pagina de adăugare, ce răspunde la adresa <http://localhost:3000/admin/notes/add>, cât și pagina de editare, <http://localhost:3000/admin/notes/edit>, au un design asemănător, ce poate fi observat în figura 3.34.

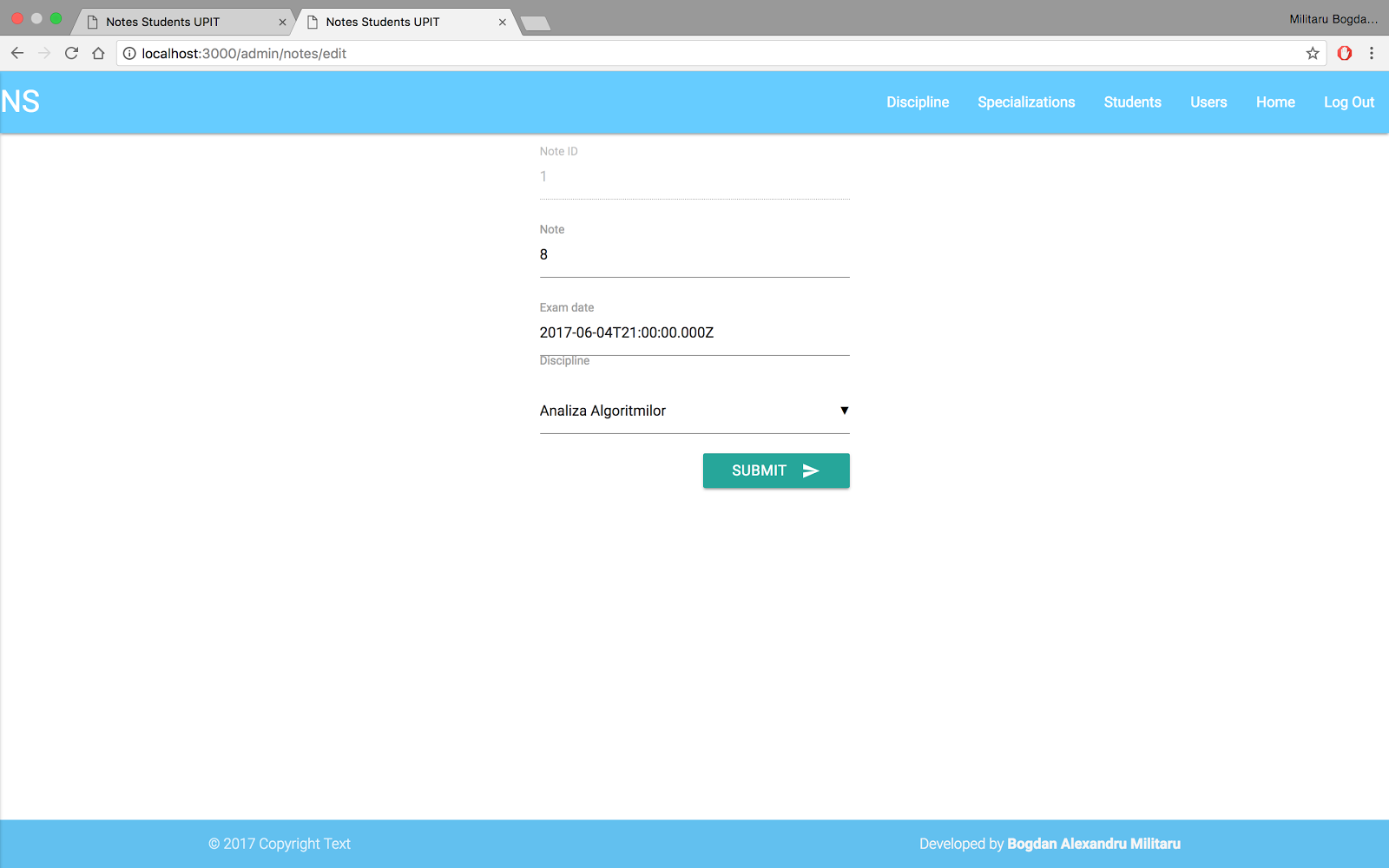


Figura 3.34 Pagină adăugare/editare notă student

Se poate observa în figura anterioara 4 câmpuri disponibile, primul *Note ID,* fiind dezactivat, neputând fii modificat, al doilea reprezentând nota, cel de-al treilea data ce poate fi selectată dintr-un pop-up ce apare după ce se dă click în interiorul câmpului, figura 3.35, și un buton de tip dropdown de unde pot fi alese toate disciplinele disponibile pentru studentul editat, figura 3.36.

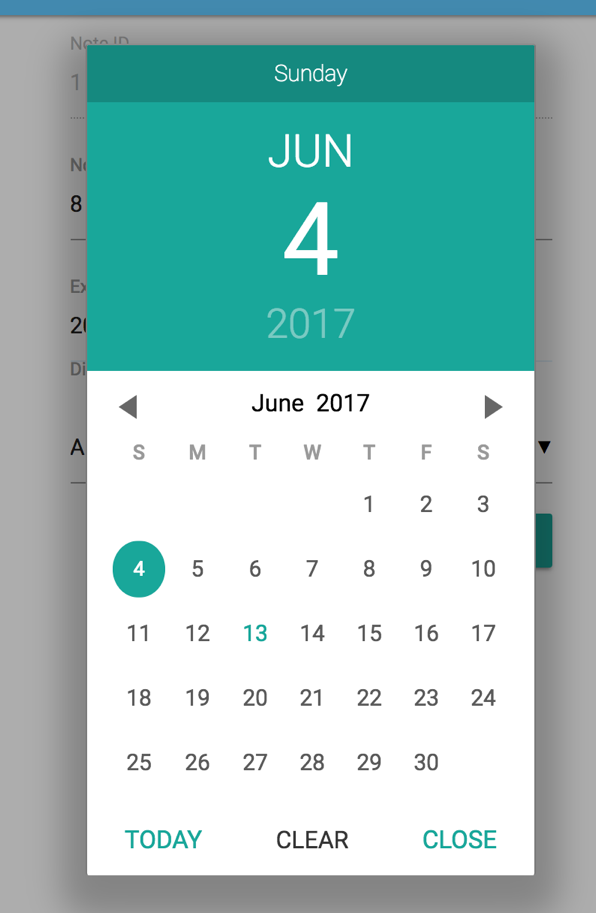


Figura 3.35 Selector dată

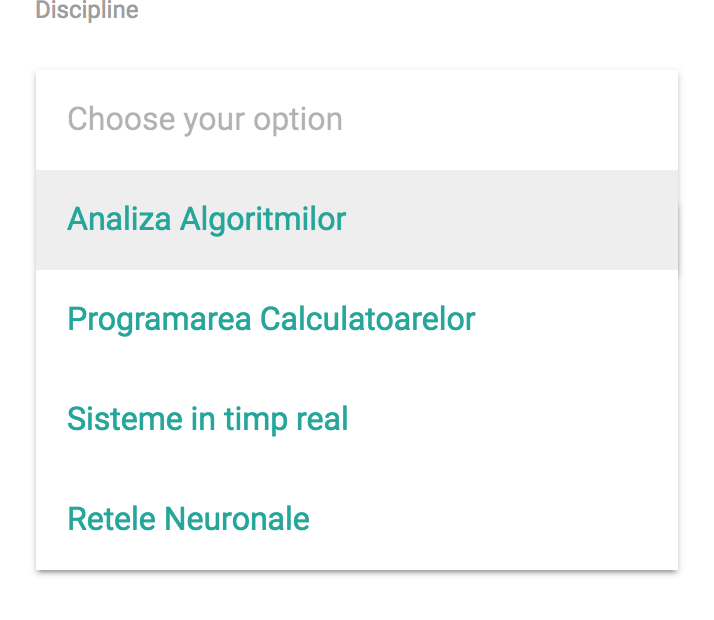


Figura 3.36 Selector discipliăa pentru o notă

### Administrare Utilizatori

Zona de administrare a utilizatorilor reprezintă o zonă în care conturile asignate studenților pot fi administrate, modificate sau șterse. Am creat această secțiune în ideea în care studenții își uită parolele, își crează conturi în locul studentului de drept sau chiar administratorul dorește să adauge și alți administratori în platformă. Secțiunea poate fi accesată din bara de meniu prin click pe *Users* sau prin accesarea directă a url-ului <http://localhost:3000/admin/users>. În figura 3.37 se poate observa formatul pe care l-am prezentat și în subcapitolele anterioare cu butonul roșu, în colțul dreapta sus pentru adăugarea unui utilizator, o listă cu utilizatorii creați, posibilitatea de editare sau ștergere a unui utilizator. În această listă de previzualizare am ales să afișez id-ul userului, numele și numărul matricol. În cazul în care este vorba de un administrator în zona de număr matricol nu va fi afișat nimic. Pentru adăugarea sau modificarea unui utilizator se va afișa un template similar, disponibil la adresele <http://localhost:3000/admin/users/add> sau <http://localhost:3000/admin/users/edit>.

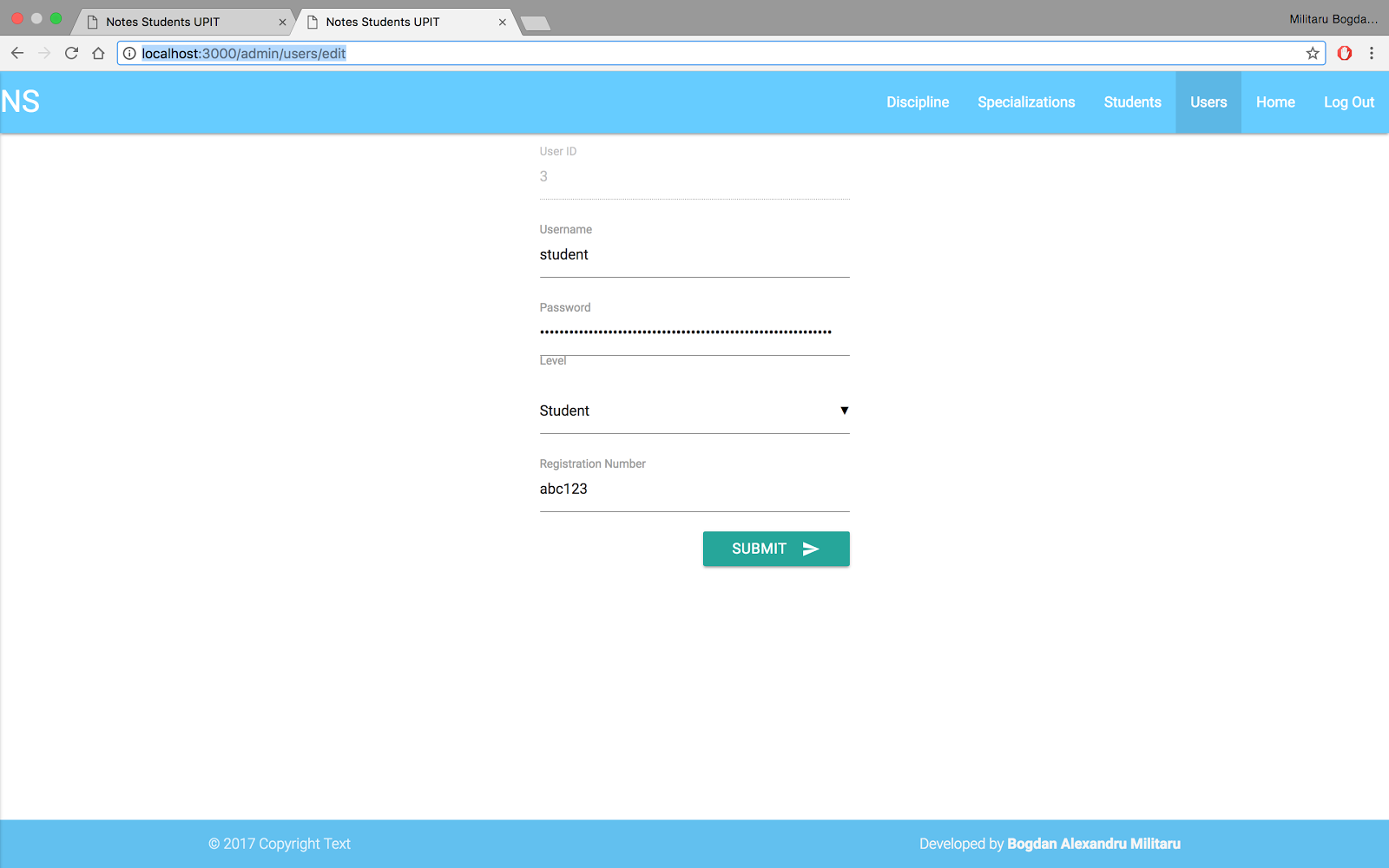


Figura 3.37 Pagină adăugare/modificare utilizator

Se poate observa în imaginea anterioară faptul ca studentului i se poate modifica numele de utilizator și parola, nivelul de permisiuni printr-o lista dropdown. În cazul în care butonul este selectat pe Student, un nou câmp pentru număr matricol va fi afișat, iar în caz contrar acesta nu va fi utilizat, figura 3.38. Toate modificările pot fi realizate prin simpla apăsare a butonului *Submit*.

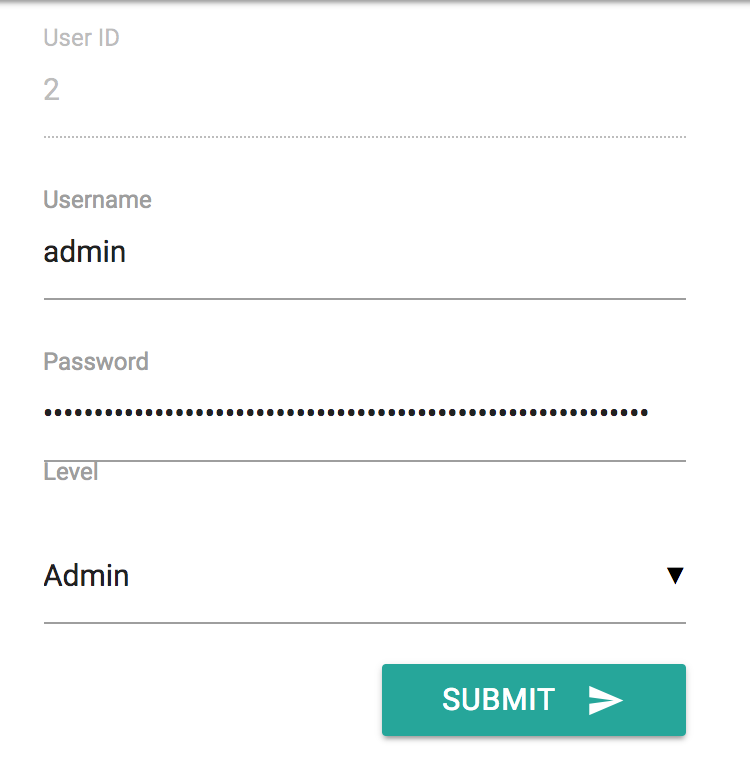


Figura 3.38 Formular pentru utilizator de tip administrator

Spre încheierea acestui subcapitol aș dori să afișez câteva fotografii ce vor prezenta aspectul website-ului în rezoluția unui telefon mobil.

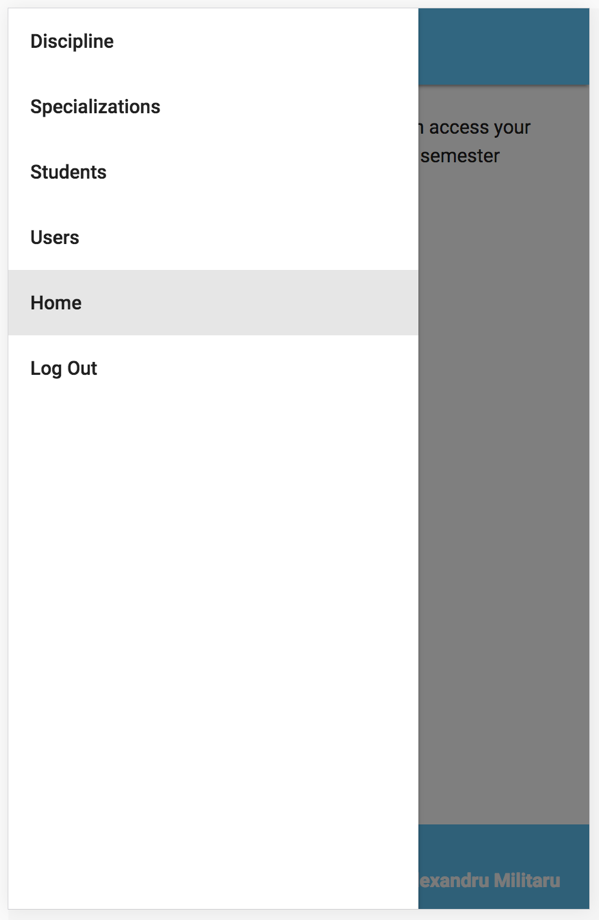


Figura 3.39 Afișare meniu în modul mobil

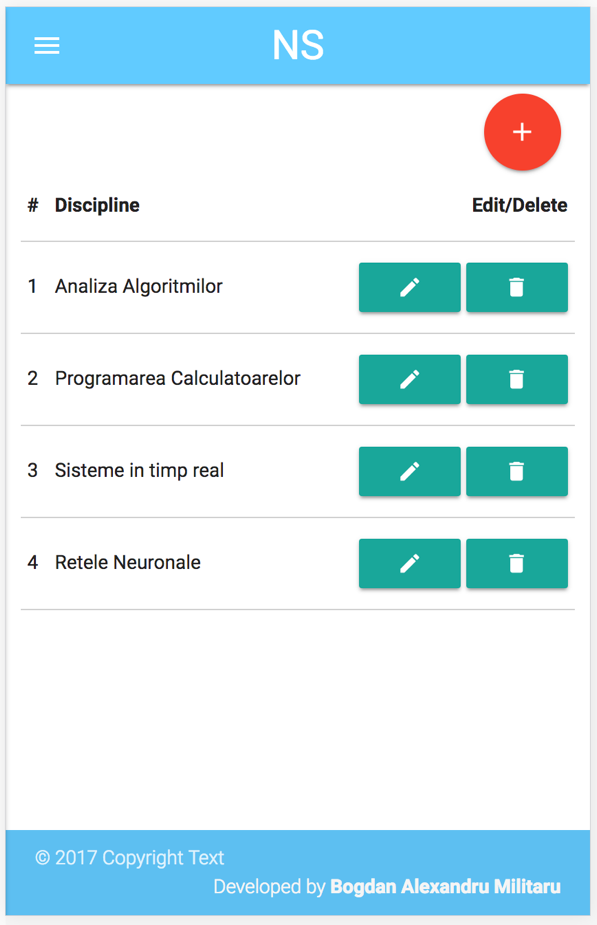


Figura 3.40 Pagină pentru discipline în modul mobil

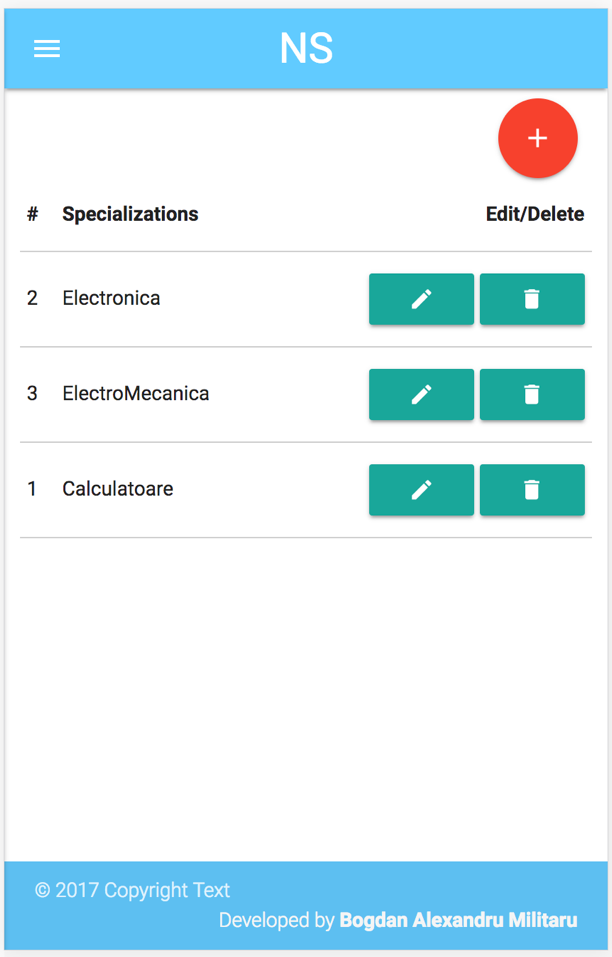


Figura 3.41 Pagină pentru specializări în modul mobil

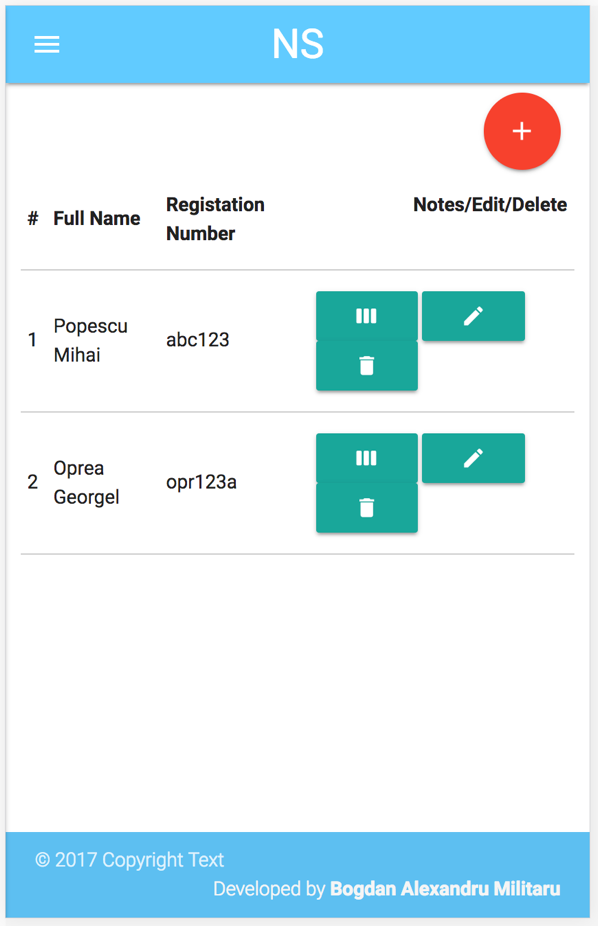


Figura 3.42 Pagină Studenți în modul mobil

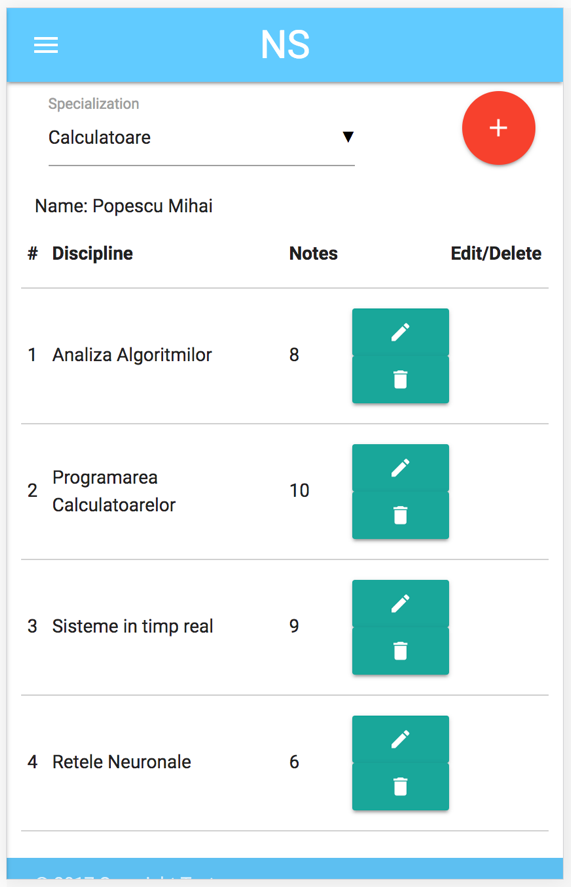


Figura 3.43Pagină Note în modul mobil

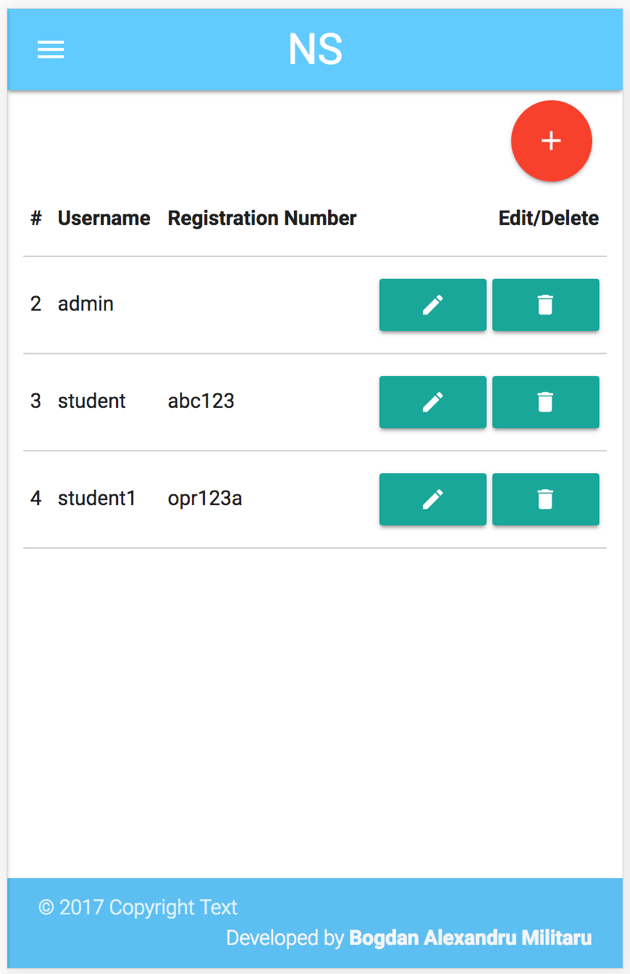


Figura 3.44 Pagină Utilizatori în modul mobil

## Mod de instalare

Dacă pe parcursul acestui capitol am vorbit despre cum se poate utiliza acest sistem și care sunt elementele ce îl definesc, în cele ce urmează voi prezenta aspecte ce vor fi destul de importante pentru un administrator de sistem. Instalarea este foarte simplă și se face într-un număr redus de pași. Voi presupune că gazda pe care se dorește rulat nu are instalat nici-unul din utilitarele necesare. Deoarece depinde foarte mult în funcție de sistemul de operare folosit și pentru că este mult mai simplă realizarea unei astfel de configurări pe un sistem de operare cu interfață grafică, dar și pentru ca majoritatea gazdelor oferă o distribuție Linux, voi considera acest sistem pentru explicațiile următore:

### Instalare baza de date

1. Se intră în Terminal
2. Se execută comanda următoare pentru a instala serverul pe care va fi rulat SGBD-ul

sudo apt-get install postgresql postgresql-contrib [29]

1. Procedeul de instalare a creat un utilizator implicit "postgres", acum va trebui să îi setez o parolă, un prompt nou va apărea după rularea comenzii

sudo -i -u postgres [29]

1. Va trebui să intri în consola postgresql prin executarea următoarei comenzi

psqld [29]

1. Tot ce rămâne de făcut este să creez o nouă bază de date

createdb Students

Un ghid mai complet poate fi găsit în referințe la punctul 29. Crearea tabelelor și a celorlalte componente, reprezintă responsabilitatea modulului Sequelize.

### Instalare server Node JS

1. Se intră in Terminal
2. Se execută comanda următoare pentru a instala serverul Node JS

sudo apt-get install -y nodejs [30]

Un ghid mai complet în funcție de distribuția de linux poate fii găsit în referințe la punctul 30.

### Rulare Sistem

1. Primul pas este să copiez sursele proiectului, iar pentru asta ne vom folosi de git, vom instala acest utilitar prin rularea comenzii în Terminal

sudo apt-get install git-all

1. Se merge în folderul în care se dorește instalarea, din terminal
2. Se ruleaza urmatoarea comandă. In prealabil trebuiesc cerute permisiuni de access la depozitul de pe Github

git clone <https://github.com/boobo94/notes-student>

1. Acum ca fișierele există pe server trebuiesc instalate dependențele

cd /server

sudo npm install

1. Se introduce parola utilizatorului curent
2. Se apasa Enter
3. Trebuiesc adăugate în fișierul de configurări ale serverului datele de conectare ale bazei de date

cd /server/database/config

sudo vi database.json

1. Se completează valorile corecte în obiectul production
2. Se apasa tasta ESC
3. În caseta de inserare se tastează :wq și se apasă enter
4. În continuare trebuie să ne asigurăm că este setat modul production

cd ../config

sudo vi server-config.json

1. În obiectul environment se seteaza production, iar obiectul production se seteaza conform necesităților
2. Se apasa tasta ESC
3. În caseta de inserare se tastează :wq și se apasă enter
4. Va trebui să modificăm pentru moment linia ce pune în execuție sincronizarea Sequelize cu baza de date

cd ..

sudo vi server.js

1. În partea de jos a filei se decomentează linia ce pune în execuție funcția initDB(); prin eliminarea celor doua slash-uri din fața liniei //
2. Se apasă tasta ESC
3. În caseta de inserare se tastează :wq și se apasă enter
4. Acum serverul poate fi rulat, prin executia comenzii următoare:

sudo npm start

1. Se introduce parola de utilizator al computerului. În cazul în care se primește un mesaj de genul " Server running at http://localhost:8080/", serverul a pornit. Această fereastră nu trebuie închisă pe toata durata cât se dorește funcționare web-service-ului
2. Se deschide un nou Terminal si se rulează comenzile

cd server

sudo vi server.js

1. Se comentează înapoi linia ce conține funcția initDB(); prin adăugarea celor doua slash-uri // în fața acesteia
2. Se apasă tasta ESC
3. În caseta de inserare se tastează :wq și se apasă enter
4. Acum că web-service-ul este pornit, este necesară pornirea serverului de client.

cd ../public

sudo npm install

1. În continuare vom modifica adresa url de conectare la webservice

cd /app/core

sudo vi urls.config.ts

1. Pe linia 5 vom modifica adresa web la care răspunde webservice-ul, modificând valoarea variabilei baseApiUrl
2. Se apasa tasta ESC
3. În caseta de inserare se tastează :wq și se apasă enter
4. Înainte de rularea serverului avem nevoie să compilăm fișierele sass

cd ../../

gulp sass

1. Putem porni serverul pentru client

cd /public

sudo npm start

1. Se introduce parola utilizatorului și se apasă ENTER

Acesta este tot procesul pe care un administrator de sistem trebuie sa îl urmeze pentru a rula proiectul.

## Versionarea codului

Am considerat că acest subiect trebuie tratat spre încheierea lucrării, deoarece trebuiau atinse aspecte ce țin concret de implementarea și realizarea proiectului, însă am pornit la drum în realizarea sa cu gândul de a utiliza un sistem de versionare.

Cunoscut în literatura de specialitate sub denumirea de version control system (VCS), reprezintă o unealtă de administrare a schimbărilor documentelor, programelor, site-urilor web și altor colecții de informații. Schimbările sunt de obieci identificate printr-un număr sau un cod format din litere și cifre [27]. Fiecărei revizuiri îi este asociată un timestamp (o secvență de caractere care încapsulează codat ora la care s-a produs un eveniment) și persoana care a făcut modificările. Revizuirile pot fi comparate, restaurate și chiar îmbinate [27]. Nevoia de organizare într-un mod logic și un control al revizuirilor a existat încă de când a apărut scrisul, însă a devenit mult mai importantă, odată cu apariția computerelor [27]. VCS-urile rulează de obicei ca aplicații de sine stătătoare, dar controlul revizuirilor este de asemenea încorporat în diferite tipuri de programe, precum procesoare de cuvinte sau tabele, CMS-uri (Content Management System) și au ca principal scop organizarea sistemelor proiectate de mai mulți dezvoltatori [27].

Un exemplu de astfel de sistem este Git, folosit de mine în cadrul acestui proiect [28]. Acesta este un VCS care urmărește modificările filelor de pe computer și coordonează lucrul mai multor programatori asupra acelor fișiere [28]. Este folosit în principal pentru administrarea codului în dezvoltarea sistemelor software, dar poate fi folosit și pentru a urmări schimbările fișierelor [28]. A fost creat în 2005 de Linus Torvalds, creatorul Linux și este disponibil gratuit [28].

Pentru a putea utiliza un astfel de sistem este nevoie de cel puțin un repository(depozit) central care să păstreze fișierele tuturor modificărilor. Fapt pentru care am utilizat un serviciu oferit de site-ul Github, iar depozitul poate fi găsit la adresa: <https://github.com/boobo94/notes-student>, în momentul de față este privat, iar pentru a putea fi accesat necesită permisiuni. Modul în care operează poate fi observat în figura 3.45.

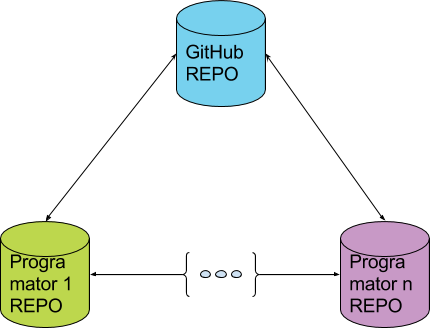


Figura 3.45 Mod de operare GIT

Pot spune ca acest sistem m-a ajutat enorm, chiar dacă am lucrat de unul singur la acest proiect, deoarece au existat și momente în care mergeam pe o pistă greșită, astfel a fost suficient să urc modificările curente în depozitul central și de fiecare dată când constatam că trebuie să mă întorc, era suficient să introduc o comandă în linia de comandă sau să utilizez un program special, precum SourceTree. Am lucrat cu 3 ramuri (branch-uri), unul primar, *master*, creat implicit în momentul instalări depozitului și alte două *server* și *client,* pentru a putea dezvolta sistemul fără perturbații din cadrul altei aplicații, aspect ce permite și dezvoltarea ulterioară a proiectului de către o echipă de programatori, fiecare echipă având posibilitatea de a lucra fără griji la una dintre aplicații. Pe parcursul lucrării am creat și alte branch-uri pentru a testa diferite moduri de implementare, pe care le-am șters ulterior, deoarece nu conțineau informații importante.

# Concluzii

  În final aș dori să prezint informații referitoare la situația în care se află aplicația, o comparație între cerințele prezentate la începutul lucrării și stadiul în care se află în aceste momente, câteva concluzii personale, aspecte care m-au influențat pe parcursul lucrării să îmbunătățesc și să aduc noi elemente proiectului și care sunt îmbunătățirile pe care doresc să le aduc în viitorul apropiat, dar și momente de cumpană, care m-au pus în dificultate.

În momentul în care am ales această temă am pornit la drum cu gândul că voi încerca să realizez o aplicație care să permită studenților din cadrul FECC să își verifice situația universitară online, neexistând un sistem bine pus la punct până în acel moment. Aplicația pe care urma să o realizez trebuia să colaboreze cu alte 2 sisteme, unul de înmatriculare a studenților în cadrul facultății și un al doilea care să gestioneze situația studenților de către secretariat. Toate aceste 3 sisteme trebuiau conectate împreună pentru a realiza o gestiune și administrare completă a studenților, însă ca de fiecare dată, practica nu se îmbină perfect cu teoria, motiv pentru care am întampinat dificultăți în colaborarea cu ceilalți doi colegi, responsabili cu dezvoltarea celorlalte două teme. Încercarea de a colabora cu cei doi a existat pentru o perioadă de 2 luni, perioadă în care am început dezvoltarea proiectului de care eram responsabil, însă am realizat faptul că cei doi colegi nu se vor implica foarte curând în realizarea sistemelor de care erau responsabili și am decis să preiau o parte din atribuții pe cont propriu. Astfel am ajuns ca de la realizarea unei interfețe grafice pe care studentul să își poată verifica notele în baza unui cont, să gestionez în interiorul aplicației dezvoltate de mine și partea administrativă de înmatriculare a studenților, administrare a cataloagelor cu note, administrare a materiilor, a specializărilor, dar și a utilizatorilor platformei. Astfel proiectul inițial a crescut cu un procentaj de 80%, dificultatea crescând preponderent cu acesta.

Obstacolele peste care a trebuit să trec la început, nu au fost însă singurele existente. Pe parcursul dezvoltării sistemului am început cu proiectarea aplicației server și astfel am fost nevoit să mă documentez despre tehnologia care se potrivește cel mai bine, iar în urma deciziei de a utiliza Node JS am fost obligat să înteleg modul de utilizare al acestuia și despre modul în care pot folosi managerul de pachete NPM în favoarea mea, pentru a nu reinventa câte o mică rotiță. Am întampinat dificultăți în încercarea de a face din aplicație una sigură, deoarece nu înțelegeam cum pot configura modulul JWT astfel încât să satisfacă cerințele aplicației mele. În același context, în cadrul aplicației server am întampinat probleme și în încercarea de a insera un middleware care sa verifice dacă cererea venită de la client are permisiunile necesare a i se răspunde cu datele solicitatate. În cealaltă parte, în aplicația client, lucrurile nu au fost cu mult diferite, existând și acolo momente precum cele în care am descoperit faptul ca rutele nu erau protejate și puteau fii accesate de oricine, indiferent de nivelul de permisiuni sau indiferent daca erau simpli vizitatori, chiar daca paginile afișate erau goale, fără informații oferite de server prin procedeele implementate. Un alt moment crucial constă în problemele de sincronizare dintre librăria Materialize și Angular 2, pe care a trebuit să le rezolv de unul singur, mai exact, librăria Materialize dispune de o serie de îmbunătățiri ale design-ului formularelor, însă prin extinderea acestor funcționalități, nu s-a ținut cont de procesul de data-binding implementat în Angular 2, am fost nevoit sa caut soluții, una din ele constând în utilizarea unui modul NPM, care nu a rezolvat integral problema, fiind nevoit să actualizez acele câmpuri manual. Au existat și alte mici dificultăți de moment, însă nu la fel de reprezentative precum acestea, dar ceea ce reprezintă cu adevarat un succes este faptul că am reușit să trec peste toate. Așa cum am explicat în cadrul primelor două capitole, referitor la alegerea tehnologiilor utilizate, este foarte important de la început să se țină cont de documentația oferită și suportul oferit de comunitate, fapt ce s-a adeverit total în soluționarea problemelor în fața cărora am fost pus. Pe lângă documentațiile oficiale oferite de producători, am folosit informații ajutătoare si de pe site-uri de specialitate precum platformele StackExchange sau forumuri si grupuri, dar și secțiunile de urmărire a bug-urilor de pe platforme de versionare precum Github.

În prezent sistemul creat este capabil să pună la dispoziție studenților informații ce privesc date personale, situația financiară, dacă studentul este înregistrat pe un loc subvenționat sau cu plată, acestea reprezentând scenariul inițial de la care am pornit, suplimentar fiind introdusă partea de administrare. Cu toate acestea consider ca există o serie de îmbunatățiri ce pot fi aduse proiectului pentru o experiență mai buna a utilizatorilor. În primul rand îmi doresc sa introduc pe prima pagină, de home, un sistem de notificări pentru studenți care să permită administratorilor postarea unor anunțuri de interes public pentru facultate, aceste note fiind transmise în mod automat către toți utilizatorii înregistrați în platforma pe adresa de e-mail. O altă îmbunătățire ce poate fi adusă sistemului în zona de înregistrare a utilizatorilor, este ca după crearea unui nou cont, noului utilizator creat să îi fie trimis un mesaj pe adresa de e-mail specificată pentru a confirma contul creat. Consider de asemenea necesară introducerea unei secțiuni suplimentare în contul de student cu privire la afișarea restanțelor, deși nu este un subiect foarte placut, nu de multe ori am remarcat în randul colegilor mei o problemă cu privire la acest aspect, aceștia neștiind materiile la care au restanțe. Cateva îmbunătățiri semnificative și extrem de utile vor urma să fie introduse în secțiunea de administrare unde exista o mare nevoie de împlementarea unui sistem, prin care administratorul să poată înmatricula în mod automat mai mulți studenți.În acest moment studenții noi pot fi înmatriculați în platformă pe rând, unul cate unul, acest lucru devenind mare consumator de timp pentru administratori. O altă îmbunătățire pe care o pot aduce lucrării și cred că va aduce totodată un plus de recunoștință din partea administratorilor constă în introducerea unui câmp de căutare în fiecare secțiune, cu o importanță majoră în secțiunea de administrare a studenților, în acest moment căutarea fiind manuală, în cazul în care se dorește modificarea unor date despre student. De asemenea consider importantă dezvoltarea unei aplicații client pentru telefoanele mobile, deoarece această piață este în continuă expansiune, iar realizarea acesteia nu ar pune mari dificultăți, web-service-ul fiind deja creat.

Doresc în această parte a lucrării să menționez faptul că deși se pot observa cu precădere în zona de bibliografie, referințe către site-uri web, acestea nu sunt singurele surse de informare care au stat la baza proiectului final. Nu am precizat anumite cărți sau articole pe care le-am citit pe parcursul dezvoltării, deoarece acestea nu au reprezentat o sursă directă de inspirație, ci au reprezentat mai degrabă o fundație pentru structurarea codului într-un format care să evite apariția de bug-uri și erori sau vulnerabilități de sistem. Printre astfel de materiale utile pe care le-am utilizat țin sa mentionez site-ul Wikipedia pe care l-am folosit ca sursă principală a informațiilor, deoarece mi s-au parut foarte bine structurate și fără dezbateri suplimentare pe marginea subiectului, site-urile oficiale ale limbajelor si tehnologiilor utilizate, StackOverflow. Printre cărțile cu o importanță deosebită, pe care le-am utilizat țin sa menționez Clean Code, o carte pe care am utilizat-o pentru a îmi îmbunătăți modul de redactare al codului și seria de cărți You don't know JS [31][32][33][34][35][36], în număr de 6 volume care pune accentul în mod deosebit pe modul de redactare al codului în javascript și prezintă de asemenea lacune și erori ale acestui limbaj de programare.

În concluzie pot spune că acest proiect este unul dintre cele mai mari proiecte realizate de mine de unul singur și primul realizat cu tehnologii web bazate pe javascript, Node JS și Angular 2. Beneficiile pe care mi le-a adus acest proiect sunt neînsemnate și într-un numar extrem de mare. Pe parcursul acestuia am dezvoltat capacități de proiectare a unui sistem pornind de la zero, am învățat cât de importantă și necesară este etapa de proiectare a sistemului, înainte de a se începe dezvoltarea propriu-zisă. Am învățat să utilizez tehnologiile bazate pe Javascript pentru crearea aplicațiilor, un beneficiu semnificativ pentru cariera pe care doresc să o urmez, aceste tehnologii fiind în continuă expansiune în aceasta perioadă. A trebuit să îmi dezvolt modul de organizare pentru a putea face față problemelor aparute, dar totodată sa gestionez timpul și să îmi stabilesc termeni limită pentru realizarea diverselor funcționalități.

# Bibliografie

1. <https://en.wikipedia.org/wiki/PHP> - accesat pe 30 iulie 2016
2. <https://en.wikipedia.org/wiki/Node.js> - accesat pe 30 iulie 2016
3. <https://en.wikipedia.org/wiki/Go_(programming_language)> - accesat pe 30 iulie 2016
4. <https://jaxbot.me/articles/benchmarks_nodejs_vs_go_vs_php_3_14_2013> - accesat pe 30 iulie 2016
5. <https://stackoverflow.com/research/developer-survey-2016> - accesat pe 15 octombrie 2016
6. <https://en.wikipedia.org/wiki/PostgreSQL> - accesat pe 20 octombrie 2016
7. <https://en.wikipedia.org/wiki/Real-time_web> - accesat pe 3 martie 2017
8. <https://en.wikipedia.org/wiki/Open-source_model> - accesat pe 20 aprilie 2017
9. <https://en.wikipedia.org/wiki/Asynchronous_I/O> - accesat pe 10 ianuarie 2017
10. <https://en.wikipedia.org/wiki/Npm_(software)> - accesat pe 18 octombrie 2016
11. <https://expressjs.com/> - accesat pe 2 decembrie 2016
12. <https://github.com/sequelize/sequelize/wiki> - accesat pe 22 noiembrie 2016
13. <https://jwt.io/introduction/> - accesat pe 17 februarie 2017
14. <https://en.wikipedia.org/wiki/Cross-origin_resource_sharing>
15. Robert C. Martin, Clean Code: A Handbook of Agile Software Craftsmanship, Prentice Hall, 2009
16. <https://en.wikipedia.org/wiki/AngularJS> - accesat pe 20 martie 2017
17. <https://en.wikipedia.org/wiki/TypeScript> - accesat pe 20 martie 2017
18. <https://en.wikipedia.org/wiki/List_of_HTTP_header_fields> - accesat pe 21 martie 2017
19. <https://en.wikipedia.org/wiki/Material_Design> - accesat pe 19 iunie 2017
20. <https://material.io/guidelines/material-design/introduction.html> - accesat pe 19 iunie 2017
21. <http://materializecss.com/> - accesat pe 19 iunie 2017
22. <https://material.io/> - accesat pe 19 iunie 2017
23. <https://en.wikipedia.org/wiki/Sass_(stylesheet_language)> - accesat pe 19 iunie 2017
24. <https://daveceddia.com/typescript-es6-questions-answers/> - accesat pe 19 iunie 2017
25. <https://en.wikipedia.org/wiki/User_experience> - accesat pe 19 iunie 2017
26. <http://uxmag.com/articles/how-to-design-your-website-for-better-user-experience> - accesat pe 19 iunie 2017
27. <https://en.wikipedia.org/wiki/Version_control> - accesat pe 25 iunie 2017
28. <https://en.wikipedia.org/wiki/Git> - accesat pe 25 iunie 2017
29. <https://www.digitalocean.com/community/tutorials/how-to-install-and-use-postgresql-on-ubuntu-14-04> - accesat pe 1 iulie 2017
30. <https://nodejs.org/en/download/package-manager/> - accesat pe 1 iulie 2017
31. Kyle Simpson, You Don't Know JS: Up & Going, O'Reilly, martie 2015
32. Kyle Simpson, You Don't Know JS: Scope & Closures, O'Reilly, martie 2014
33. Kyle Simpson, You Don't Know JS: this & object Prototypes, O'Reilly, iulie 2014
34. Kyle Simpson, You Don't Know JS: Type & Grammer, O'Reilly, februarie 2015
35. Kyle Simpson, You Don't Know JS: Async & Performance, O'Reilly, februarie 2015
36. Kyle Simpson, You Don't Know JS: ES6 & Beyond, O'Reilly, decembrie 2015
37. Leo S. Hsu, Regina O. Obe, PostgreSQL: Up & Running, O'Reilly, martie 2012

# Anexe

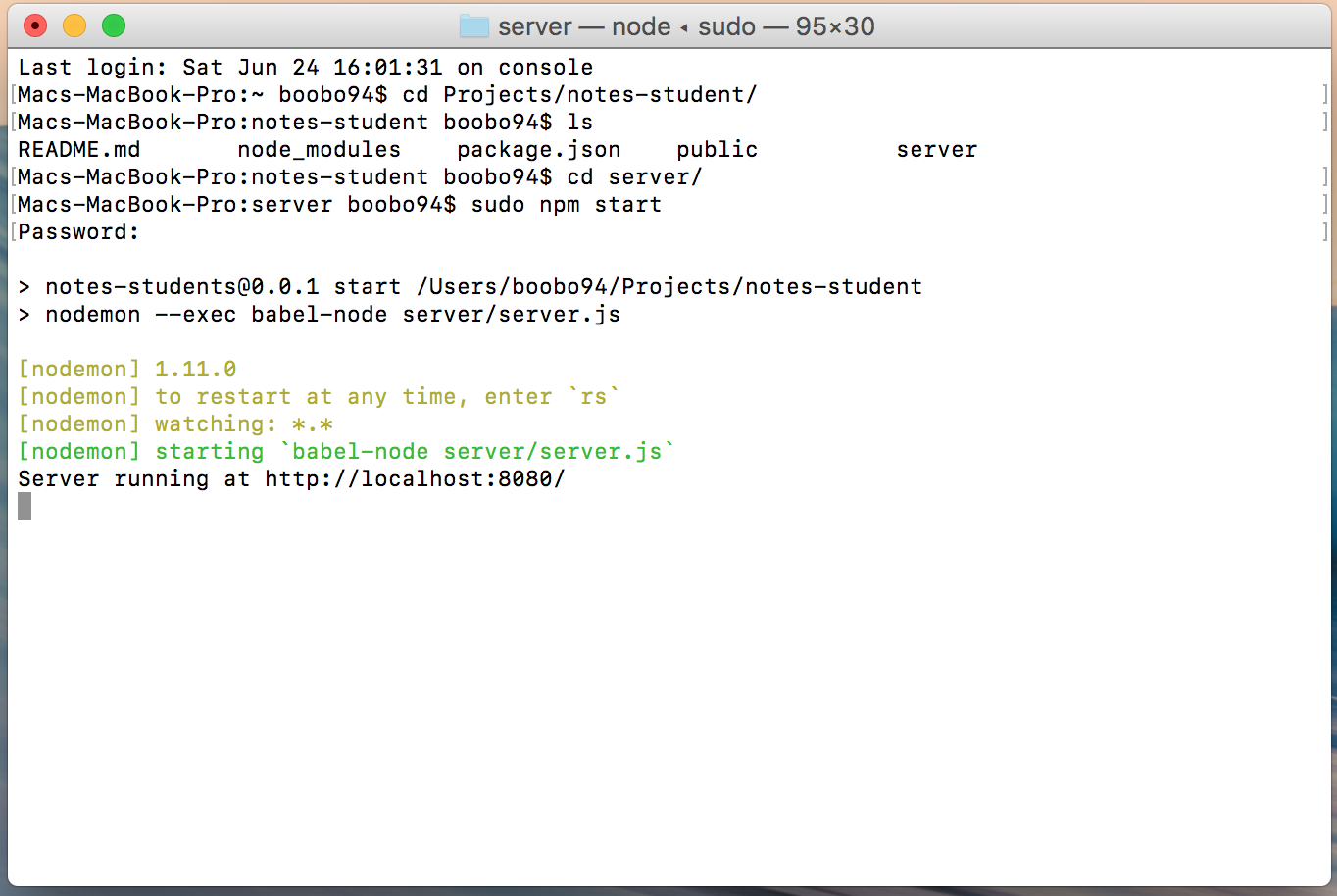


Figura 4.1 - Lansare în execuție web-service

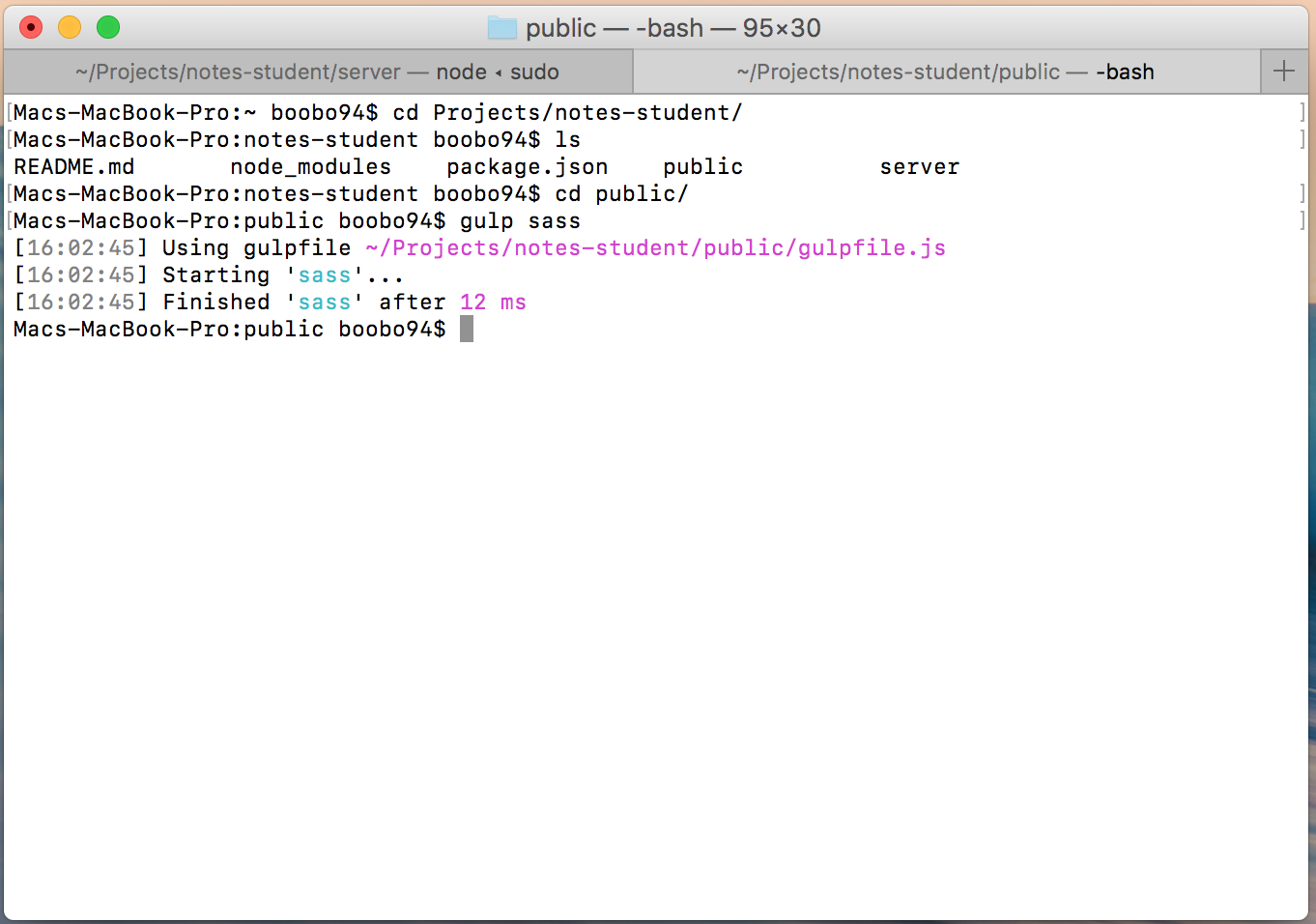


Figura 4.2 - Compilare cod SASS

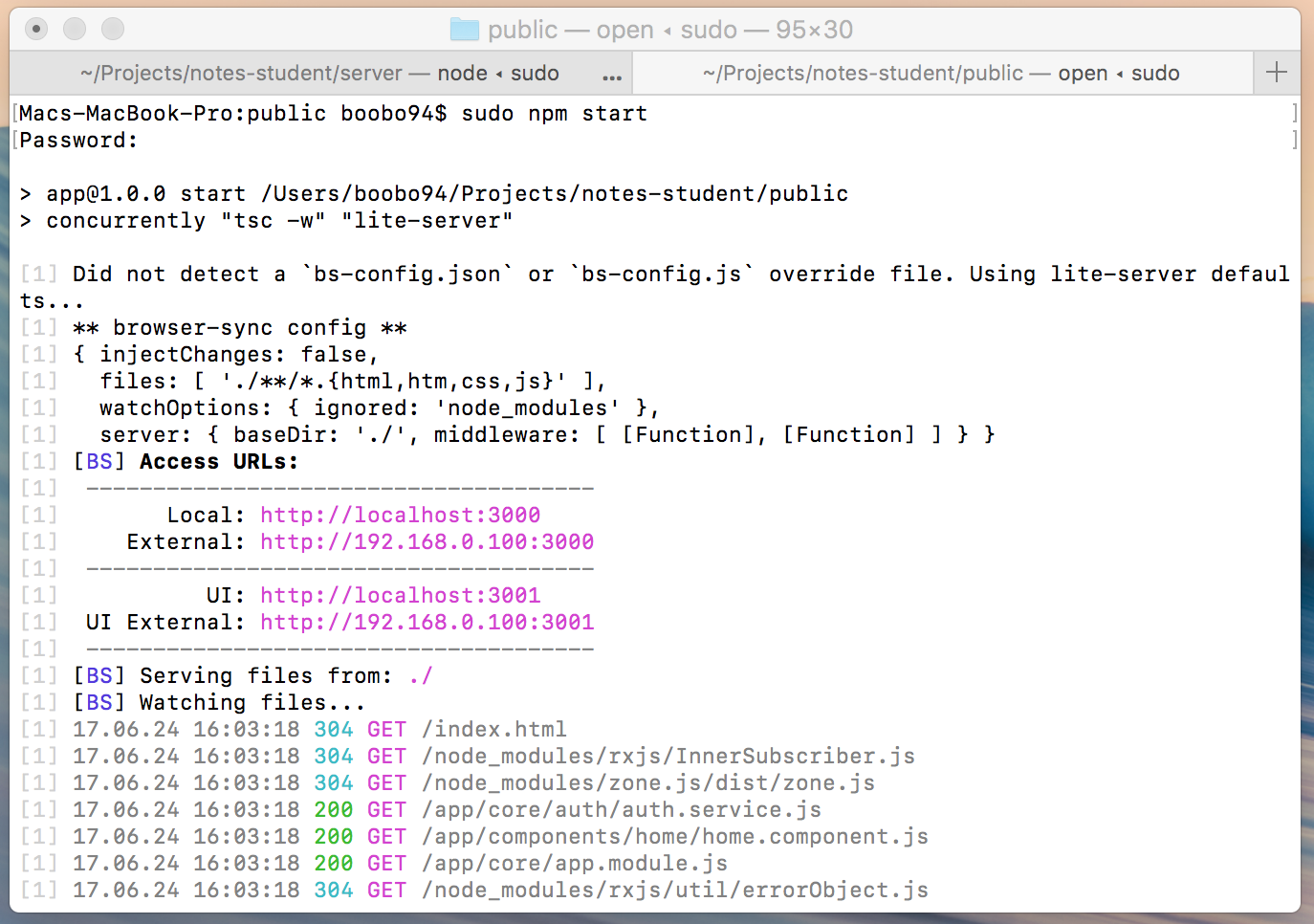


Figura 4.3 - Lansare în execuție aplicație client

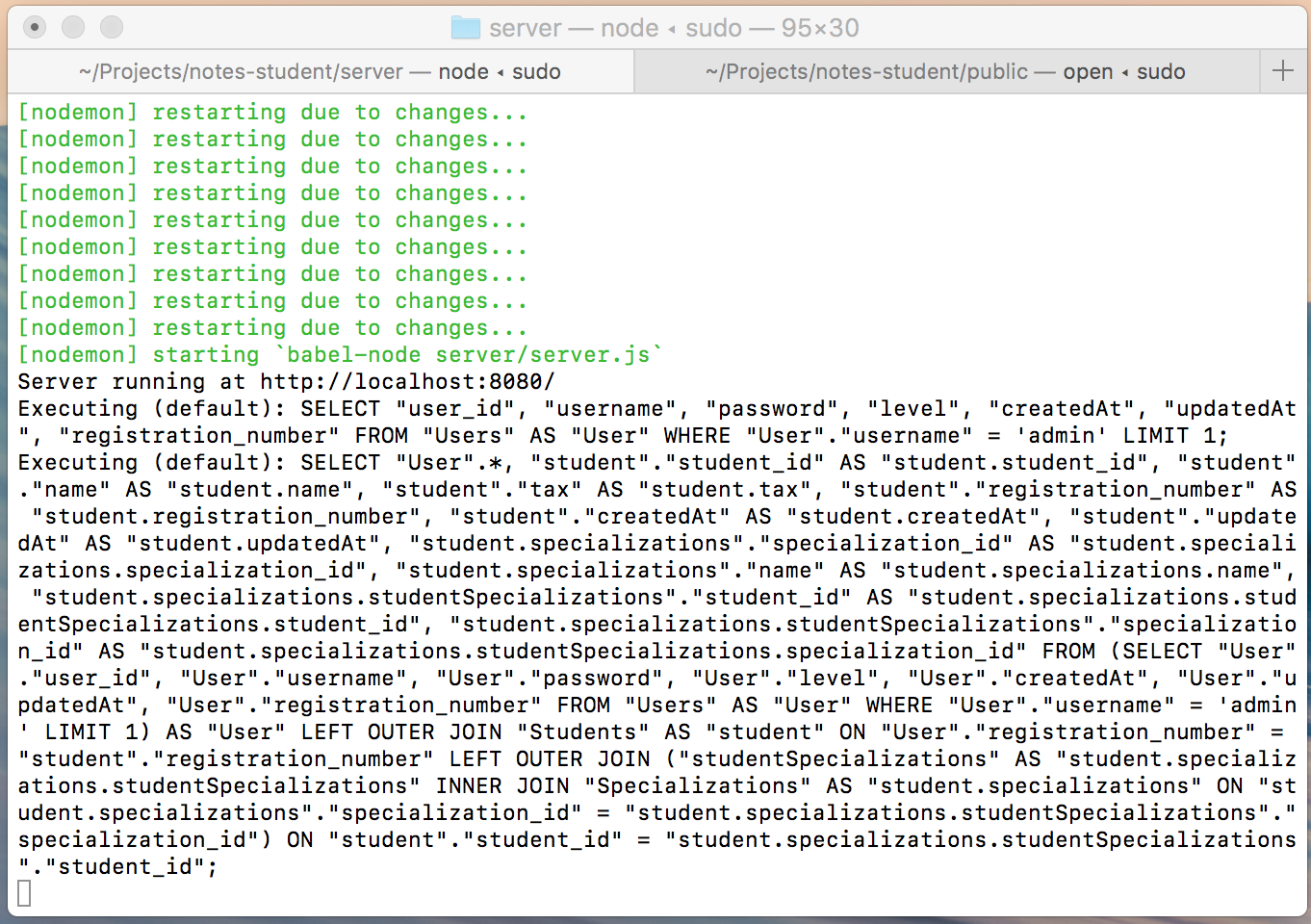


Figura 4.4 - Rezultate consolă pentru logare utilizator

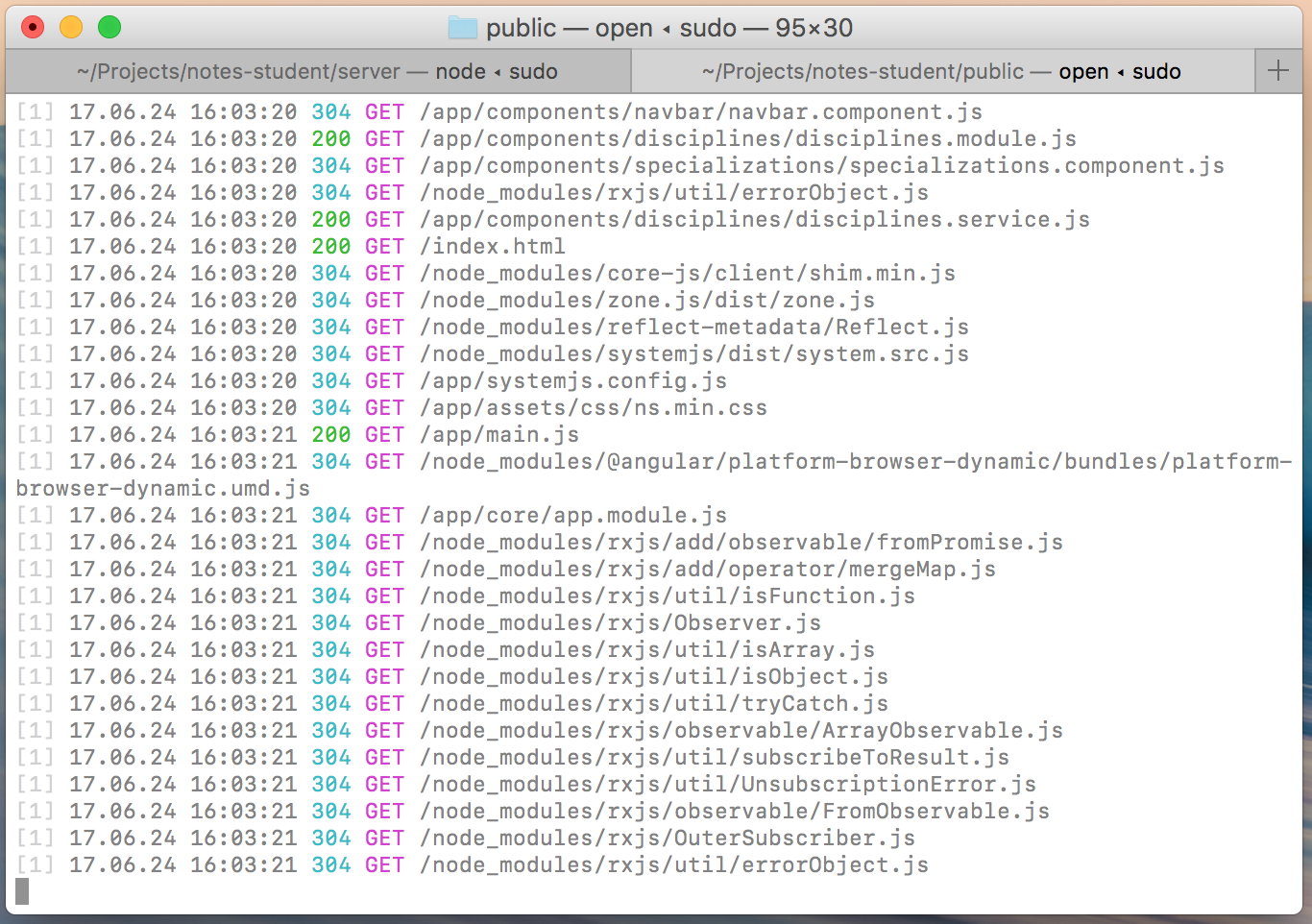


Figura 4.5 - Consolă de execuție aplicație client

# CV

|  |  |
| --- | --- |
| PERSONAL INFORMATION | Bogdan Alexandru Militaru |
|  | |
|  | Glambocata-Deal, nr. 75, 115500 Leordeni (Romania) |
| 0727900500 |
| militaru.bogdanalexandru@gmail.com |
| [http://boobo94.xyz](http://boobo94.xyz/#_blank) |
| Sex Male | Date of birth 09/10/1994 | Nationality Romanian |

|  |  |
| --- | --- |
| STUDIES APPLIED FOR | Degree Thesis |

|  |  |
| --- | --- |
| WORK EXPERIENCE |  |

|  |  |
| --- | --- |
| 04/07/2016–Present | Software Developer |
| Airtouch New Media SRL, Bucharest (Romania) |
| • front-end development • back-end development • mobile development using hybrid technologies ( cordova, ionic ) • bug fixes • maintenance • team work• project management​​ |

|  |  |
| --- | --- |
| 04/2015–11/2015 | Web Developer |
| Roweb Way2way Plus, Pitesti (Romania) |
| • web development (front-end and back-end) • web applications maintenance • database manipulation • website optimization • SEO• Wordpress​​ |

|  |  |
| --- | --- |
| 06/2014–02/2015 | Web Developer - Internship |
| Roweb Development SRL, Pitesti (Ro) |
| • I learned about HTML, CSS, PHP, JavaScript and also I used frameworks like Bootstrap, jQuery, Codeigniter. • Developed Layouts using HTML and CSS • Developed an online books library developed with native PHP and after that I rebuilt it using Codeigniter. The user can creates an account,set his profile and then he have access to add books to the cart, increase or decrease the quantity in the cart, delete the products from the cart, sending an order. The admin can publish,modify or delete the books and manage the ordering process. • Developed scripts in Javascript • Work with Wordpress administration panel and plugins and themes development • Team Work• Communications skills​​ |

|  |  |
| --- | --- |
| EDUCATION AND TRAINING |  |

|  |  |  |
| --- | --- | --- |
| 2013–2017 | Certificate of Computer Engineering |  |
| Universitatea din Pitești, Pitesti (Romania) | |
| • web development • desktop development: C, C#, Java • graphic:AutoCAD • hardware development: vhdl, orcad, assembly​• working on fpga, circuits, electronically devices​​ | |

|  |  |  |
| --- | --- | --- |
| 2009–2013 |  |  |
| "Ion Mihalache" HighSchool, Topoloveni (Romania) | |
| • Member in Students Committee • head of the class • public speaking• team work​​ | |

|  |  |
| --- | --- |
| PERSONAL SKILLS |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Mother tongue(s) | Romanian | | | | |
|  |  | | | | |
| Other language(s) | UNDERSTANDING | | SPEAKING | | WRITING |
| Listening | Reading | Spoken interaction | Spoken production |  |
| English | B2 | B2 | B2 | B2 | B2 |
|  | Levels: A1 and A2: Basic user - B1 and B2: Independent user - C1 and C2: Proficient user  [Common European Framework of Reference for Languages](http://europass.cedefop.europa.eu/en/resources/european-language-levels-cefr) | | | | |