MINISTERUL EDUCAȚIEI NAȚIONALE

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA
FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

Student Planner
Aplicație pentru planificarea calendarului studențesc

LUCRARE DE LICENŢĂ

Absolvent: **Adelina - Codruța Milionean**

Coordonator științific: **Asist. Prof. Ing. Cosmina Ivan**

2020

MINISTERUL EDUCAȚIEI NAȚIONALE

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA
FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

DECAN,
Prof. dr. ing. Liviu MICLEA

DIRECTOR DEPARTAMENT,
Prof. dr. ing. Rodica POTOLEA

Absolvent: Adelina Codruta MILIONEAN

Student Planner
Aplicație pentru planificarea calendarului studențesc

1. **Enunțul temei:** Proiectul descris în această lucrare are ca scop definirea, proiectarea și implementarea unui sistem informatic de tip aplicație web, destinată studenților. Ea va permite organizarea activităților universitare într-un mod ușor de folosit și interactiv cu scopul îmbunătățirii performanțelor academice. Sistemul va oferi și funcționalități specifice unui administrator.

2. **Conținutul lucrării:** Pagina de prezentare, Introducere, Obiectivele Proiectului, Studiu Bibliografic, Analiză și Fundamentare Teoretică, Proiectare de Detaliu și Implementare, Testare și Validare, Manual de Utilizare, Concluzii, Bibliografie, Anexa 1 Lista Procedurilor Stocate, Anexa 2 Glosar

3. **Locul documentării**: Universitatea Tehnică din Cluj-Napoca, Departamentul Calculatoare

4. **Consultanți**: asist. prof. ing. Cosmina-Daniela Ivan

1. **Data emiterii temei:** 1 martie 2020

2. **Data predării:** 8 iulie 2020

Absolvent: _____

MINISTERUL EDUCAȚIEI NAȚIONALE

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA
FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

Coordonator științific: _____

Declarație pe proprie răspundere privind
autenticitatea lucrării de licență

Subsemnatul(a)_____
_____,
legitimat(ă) cu _____ seria _____ nr. _____
CNP _____, autorul lucrării
_____
_____
_____elaborată în vederea susținerii examenului de finalizare a studiilor de licență la Facultatea de Automatică și Calculatoare, Specializarea _____ din cadrul Universității Tehnice din Cluj-Napoca, sesiunea _____ a anului universitar _____, declar pe proprie răspundere, că această lucrare este rezultatul propriei activități intelectuale, pe baza cercetărilor mele și pe baza informațiilor obținute din surse care au fost citate, în textul lucrării, și în bibliografie.

Declar, că această lucrare nu conține porțiuni plagiate, iar sursele bibliografice au fost folosite cu respectarea legislației române și a convențiilor internaționale privind drepturile de autor.

Declar, de asemenea, că această lucrare nu a mai fost prezentată în fața unei alte comisii de examen de licență.

In cazul constatării ulterioare a unor declarații false, voi suporta sancțiunile administrative, respectiv, *anularea examenului de licență*.

Data

_____

Nume, Prenume

_____

Semnătura

Cuprins

Capitolul 1. Introducere – Contextul proiectului

1.1. Contextul proiectului

Planurile bune și pregătirea conduc la performanțe bune. Planurile și pregătirile remarcabile conduc la performanțe remarcabile. Planificarea anticipată ajută la îmbunătățirea performanței.

În ziua de astăzi, mai ales în rândul tinerilor, tehnologia joacă un rol important în viața de zi cu zi. Evoluția tehnologică împreună cu conectarea a mai tuturor echipamentelor la internet ne permite să ne organizăm timpul de oriunde și oricând. Telefoanele noastre inteligente sunt mai mult decât capabile să țină evidența zilnică a programelor noastre, dar folosirea unei aplicații de tip calendar poate consuma destul de mult din timpul unei persoane pentru a reuși să își organizeze într-un mod logic și intuitiv tot ce are de făcut, deoarece acestea nu sunt customizate pentru a satisface nevoile personale, ci sunt gândite pentru a deservi un public larg.

De asemenea, mulți ar putea argumenta că folosirea unei agende fizice ar putea fi o alternativă mai bună decât folosirea unei aplicații în vederea păstrării de notițe. Utilizarea uneia ar limita accesul la notițe, deoarece ar putea fi vizualizate doar când utilizatorul o are în posesie, pe când majoritatea persoanelor care ar fi interesate de a folosi o asemenea modalitate de a-și organiza timpul au un telefon sau un PC conectat la internet în cea mai mare parte a timpului.

Scopul lucrării a fost dezvoltarea unei platforme online prin care un student își poate organiza activitățile universitare și timpul alocat acestora într-un mod interactiv. Sistemul va avea mai multe funcționalități utile in acest sens, cum ar fi vizualizarea împărțirii timpului in mod calendar, vizualizarea informațiilor despre orar, dar și interacțiunea cu alți studenți prin intermediul grupurilor de studiu. La acest sistem informatic au acces și utilizatori de tip administrator, ce pot customiza informațiile legate de cursuri, studenți sau activități astfel încât un student să își poată organiza calendarul într-un mod cât mai ușor, specific facultății, specializării și anului de studiu.

1.2. Motivație

Cu toate ca există diverse platforme ce oferă multe dintre funcționalitățile mai sus menționate, principalul avantaj al acestui sistem îl constituie faptul ca propune o abordare cat mai completa , oferindu-le in maniera integrata utilizatorilor ce se înrolează in sistem pe baza unui cont si a unei parole. Un alt avantaj este dezvoltarea platformei ca platforma web, astfel ea va fi accesibila utilizatorului din orice browser, indiferent de dispozitivul folosit. Indiferent daca el este mobile sau desktop interfața își va păstra funcționalitatea și se va scala conform rezoluției, eliminând nevoia unei aplicații separate salvată local pentru dispozitivele mobile. De asemenea, aplicația este menită să poată fi customizată în conformitate cu nevoile studenților, ei nefiind nevoiți să își definească în fiecare semestru materii specifice cărora să le asigneze activitățile create, acestea fiind deja disponibile în cadrul aplicației în funcție de statutul studentului.

1.3. Abordare

Pentru a dezvolta prezenta aplicație am urmat mai mulți pași pentru a ajunge de la concept la prototip funcțional. În primă fază m-am pus în postura de posibil utilizator al aplicației și am dezvoltat diverse scenarii numite „user-stories” pentru a determina ce funcționalități mi-aș dori să aibă această aplicație. Apoi am împărțit aceste scenarii pe module pentru a-mi da seama care vor fi paginile principale ale acesteia. A urmat o perioadă de cercetare în care m-am documentat pentru a găsi alternative sau aplicații cu scop similar pentru a determina dacă aceasta ar fi sau nu utilă în viața de zi cu zi a unui student. După analiza alternativelor am decis că există loc de a îmbunătății ceea ce există și că o astfel de aplicație de planificare destinată exclusiv studenților ar aduce beneficii, astfel am început documentarea despre și alegerea tehnologiilor ce într-un final le-am folosit pentru a concretiza aplicația. După alegerea tehnologiilor și a strategiei de dezvoltare, am început implementarea efectivă a aplicației web mergând pe ideea de a obține un cuplaj cât mai mic între module.

1.4. Structura lucrării

Prezenta lucrare este structurată sub forma a opt capitole după cum urmează:

Capitolul 1, intitulat Introducere-Contextul proiectului, propune realizarea unei scurte descrieri a domeniului din care face parte sistemul. În acest capitol se conturează tema și contextul problemei pe care lucrarea o va aborda.

Capitolul 2, numit Obiectivele proiectului, cuprinde obiectivele principale pe care aplicația dezvoltată este menită să le atingă, dar și cerințele funcționale și non-funcționale ale sistemului.

În capitolul 3, Studiu bibliografic, este descrisă calea de soluționare a problemei alese și este realizată o analiză a resurselor și tehnologiilor utilizate.

Capitolul 4 se intitulează Analiză și fundamentare teoretică și descrie soluționarea problemei și este prezentată soluția propusă.

Titlul capitolului 5 este Proiectare de detaliu și implementare. Funcțiile, metodele, clasele și rolul lor în aplicație vor fi prezentate pe parcursul acestuia.

Capitolul 6 este numit Testare și validare și prezintă parcursul activității de testare al aplicației, acesta cuprinzând rezultatele și metodele folosite.

Capitolul 7 se intitulează sugestiv Manual de instalare și utilizare. Aici vor fi menționate resursele necesare pentru instalarea și rularea aplicației, și o prezentare de tip step – by – step a utilizării programului.

Capitolul 8, Concluzii, este un rezumat al lucrării și o analiză în retrospectivă a modului de realizare al acesteia.

Capitolul 2. Obiectivele Proiectului

2.1. Specificația proiectului

Ideea proiectului a pornit ca urmare a constatării faptului ca in universitățile din tara un astfel de software, cu valoare certa pentru activitatea curenta a studenților nu a fost identificat și de la interesul personal pentru existența unui astfel de sistem. Prin intermediul acestei aplicații web, studentul ca utilizator va avea acces la toate resursele necesare pentru o mai buna organizare a timpului din punct de vedere al activităților ce țin de viată universitara, dar va aduce in plus posibilitate de a interacționa cu alți studenți ce contractează aceleași materii. În plus experiența sa în aplicație va fi customizată din perspectiva anului de studiu și a specializării de care face parte, prin intermediul datelor introduse de către un utilizator de tip administrator în aplicație. Se vor stoca informații despre grupe și cursuri specifice acestora, iar studenții obligatoriu vor aparține de una din grupe. Astfel în cadrul aplicației vor vedea doar informațiile relevante pentru ei, nu și cele legate de alți ani de studiu sau alte specializări și nici nu vor fi puși în situația de a trebui să își organizeze manual activitățile în funcție de materia asociată. Excepție de la această regulă o vor face evenimentele sau vacanțele ce sunt de interes pentru întreaga facultate.

Aplicația este formată din trei componente principale, acestea fiind componenta de planificare, componenta de socializare și componenta de administrare. Funcțiile ce le îndeplinesc fiecare dintre acestea vor fi descrise în subcapitolul următor.

În concluzie, s-a urmărit implementarea unui sistem cât mai complet ce vine in ajutorul studenților cu scopul de a le permite sa își organizeze mai bine și într-un mod cat mai intuitiv timpul alocat studiului.

2.2. Obiective principale

Pentru ca sistemul descris sa aibă comportamentul și funcționalitățile așteptate a fost nevoie de implementarea a trei componente principale ce agregate formează aplicația ca un tot unitar. Aceste trei componente sunt:

- Componenta de planificare: componenta prin intermediul căreia se vor putea vedea într-un mod ușor de interpretat și înțeles activitățile
- Componenta de socializare: componenta ce are ca scop încurajarea interacțiunilor între studenți
- Componenta de administrare: componenta vizibilă doar utilizatorilor de tip administrator, responsabilă cu managementul bazei de date

2.2.1. Componenta de socializare

În cadrul acesteia există posibilitate de a crea un grup de studiu de către utilizator, de a vedea detaliile despre și de a adera la un grup de studiu și în final posibilitatea de a interacționa prin intermediul paginii de discuții cu ceilalți studenți înscriși în același grup.

2.2.2. Componenta de planificare

În cadrul aceste componente există posibilitatea de a vizualiza si organiza calendarul personal, de a vizualiza activitățile introduse de administratori în calendar

(examene, evenimente, zile libere), posibilitatea de a adăuga activități în calendar de tip Study, Assignment, Deadline sau Other și de a asocia o materie din cele contractate acesteia. În plus, în cadrul calendarului se vor putea face orice fel de modificări. Această componentă include și un dashboard cu un sumar al activităților din ziua curentă, dar și un board kanban (asemănător cu cel utilizat în cadrul dezvoltării proiectelor) ce prezintă pe categorii activitățile din calendarul studentului.

2.2.3. Componenta de administrare

În cadrul acesteia, ca utilizator din acest grup este posibilă administrarea utilizatorilor, a cursurilor și a orarului, dar și adăugarea de evenimente de tip zile libere ce vor apărea în calendarul tuturor studenților.

2.3. Aspecte urmărite

Există o diferență foarte mare între aplicațiile web și aplicațiile desktop atât din perspectiva tehnică și din perspectiva utilizatorului. Pentru ca aplicația să fie dezvoltată cu succes am urmărit atingerea anumitor aspecte. Cele trei aspecte considerate a fi relevante sunt:

1. Performanță - Unul dintre cele mai mari dezavantaje aplicații web a fost la început faptul că aveau o performanță scăzută, datorată timpului mare de răspuns și a tehnologiilor disponibile la acel moment. Totuși recent dezvoltarea browserlor și a capabilităților acestora a deschis posibilități dezvoltatorilor care doresc să se dezvolte aplicații mai mari și produse mai complexe. Prin implementarea folosind tehnologiile si metodologiile alese, se urmărește ca timpii de răspuns pentru apelurile către server să nu depășească 200ms pentru un utilizator obișnuit.
2. Utilizabilitate – Conform articolului *Improving the Usability of Web Applications* [1], utilizabilitatea unei aplicații nu se rezumă doar la interfața grafică ci și la modul în care ea e construită. Astfel utilizabilitatea este definită atât prin ușurința cu care utilizatorul final folosește aplicația, ci și cât de dificil este în viitor ca aceasta să fie extinsă. Dacă utilizabilitatea interfeței grafice se poate obține prin implementarea de acțiuni intuitive, cea de pe partea de implementare trebuie atinsă prin implementarea unei arhitecturi corespunzătoare tipului de aplicație.
3. Scalabilitate – Un aspect important de urmărit în cadrul dezvoltării unei aplicații web este capacitatea acesteia de a ajunge la un număr cât mai mare de utilizatori țintă. O aplicație web trebuie să suporte un număr mai mare de utilizatori, dacă este necesar, fără ca cei existenți să fie afectați. Acest lucru se poate obține prin alegerea unei arhitecturi scalabile, modulare și a unui hardware potrivit.

Capitolul 3. Studiu Bibliografic

Acest capitol explică în detaliu resursele care au stat la baza conceptualizării sistemului, cât și prezentarea soluției pe care o oferă proiectul în comparație cu soluțiile existente deja în uz.

3.1. Contextul general de dezvoltare a sistemului

3.1.1. Planificarea procesului de învățare

Un plan de studiu este un program organizat pe care studenții îl creează și care conturează orele de studiu și obiectivele de învățare. La fel ca și în cazul orarelor școlare, studenții ar trebui sfătuiți să își elaboreze un program de studiu în care să blocheze zilele și orele din calendarul dedicat studiului. Crearea unui plan de studiu ajută studentul în a deveni mai organizat, ceea ce se reflectă în rezultatele obținute de acesta. Un astfel de plan de studiu este important, deoarece duce la dezvoltarea capabilității de auto-disciplina și motivației de a finaliza studiile. În plus folosirea unui planificator oferă o metodă ușoară și creativă de a monitoriza temele și termenele limită. Acestea le oferă studenților o abordare sistematică pentru înregistrarea zilnică a informațiilor necesare. Acestea permit studenților să se pregătească în conformitate cu cerințele și activitățile lor de clasă. Ele permit studenților să-și urmărească obiectivele. Ei pot folosi planificatorul pentru a-și scrie listele de obiective și sarcini. Pe măsură ce își ating obiectivele pe parcursul săptămânii, ei sunt capabili să se inspire să muncească mai mult. Stabilirea obiectivelor este un pas important în strategia de a învăța pentru activitatea universitară.

3.1.2. Grupurile de studiu

Conform unui articol [2] realizat de Universitatea Naționala Florida, participarea sau organizarea unui grup de studiu este benefica din diverse motive pe lângă înțelegerea subiectului, precum dezvoltarea abilităților de studiu in echipa, combaterea procrastinării sau posibilitatea de a fi motivat sau de a-i motiva pe alții. Aceste aspecte motivează integrarea componentei de grup de studiu in aceasta platforma.

3.2. Abordări în planificarea sarcinilor

3.2.1. Board-ul Kanban

Vizualizarea activităților diverse de efectuat sub forma de categorii este un bun mod pentru o persoană de a și le planifica în așa fel încât să le finalizeze. În metodologia Agile de dezvoltare a proiectelor software, acest tip de board este larg utilizat pentru managementul acestora. Cuvântul *Kanban* este în sine o definiție a acestei metodologii, el fiind format din cuvintele japoneze *kan* – vizual și *ban* - cartonaș [3]. Astfel, acest board este, de fapt, o colecție de cartonașe așezate într-un mod vizual, ușor de înțeles, cu scopul de a determina starea în care se află sarcinile. el poate fi folosit în mod eficient și de către studenți pentru a-și putea organiza mai bine ce au de făcut, împărțindu-și sarcinile pe categorii, cum ar fi „neîncepute”, „începute” și „terminate”.

3.2.2. Calendarul

Folosirea calendarului este una dintre cele mai răspândite și vechi metode pentru a face planuri, fie ele pe termen scurt sau lung. Conform unui articol scris de jurnalistul Srinivas Rao [4], unul din fondatorii aplicației ce poartă acum numele de Google Calendar susține că lucrurile ce sunt reprezentate în calendar au mai multe șanse de a fi realizate de către un individ față de cele ce nu sunt. De asemenea, una din ideile ce îndeamnă la folosirea unui calendar în favoarea unei simple liste cu lucruri de făcut este faptul că un calendar este strâns legat de noțiunea de timp, lucru ce duce la cuantificarea unei sarcini.

3.3. Alternative existente

3.3.1. Calendarele online

Unul dintre beneficiile majore ale uni calendar online este că acesta poate fi vizualizat pe toate dispozitivele. Odată adăugat ceva în calendarul online, acesta se va sincroniza automat, astfel încât evenimentul să apară pe computer, pe tabletă sau pe orice alt dispozitiv ce poate accesa internetul.

Calendarele online permit de asemenea suprapunerea de diferite activități pe același șablon. Se pot vizualiza simultan toate calendarele sau se poate alege vizualizarea numai a evenimentele. Acest lucru este util pentru a obține o idee despre imaginea de ansamblu. Calendarele online sunt o alegere foarte populară și datorită ușurinței cu care pot fi accesate și coordonate împreună cu alții.

3.3.2. Google Calendar

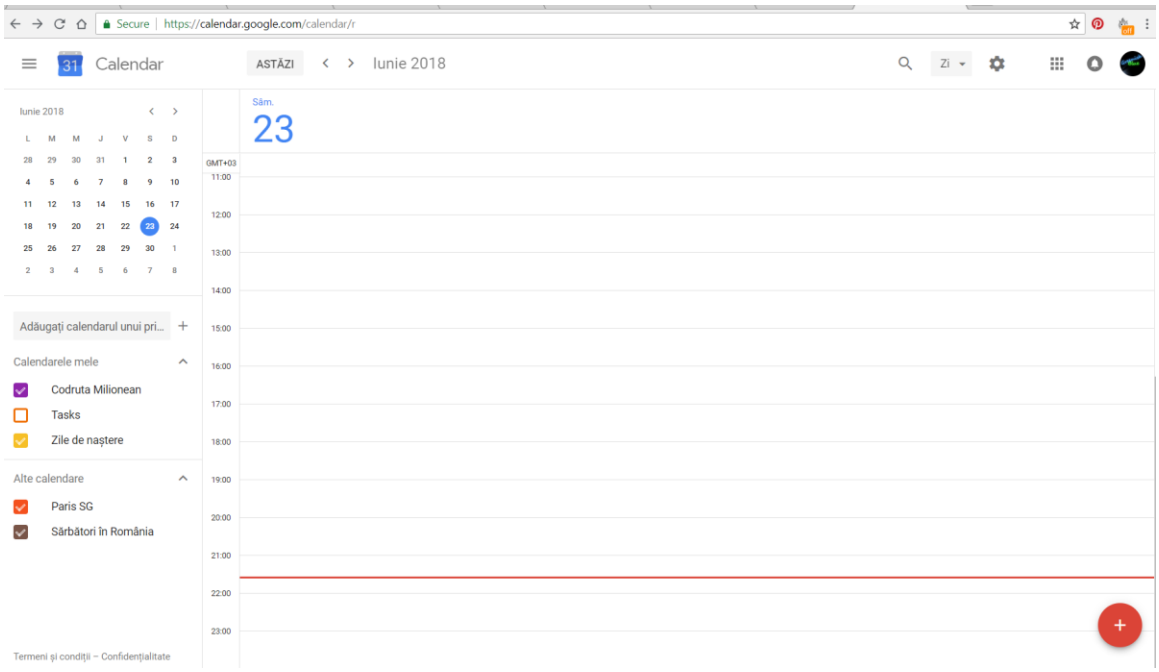

Figura 3.1 Interfață Google Calendar

Google Calendar[1] este un serviciu de gestionare a timpului și planificare calendaristică dezvoltat de Google. A devenit disponibil ca versiune beta limitată pe 13 aprilie 2006 și a ieșit din etapa beta în iulie 2009. Este disponibil pe web sau sub formă de aplicații mobile pentru sistemele de operare mobile Android și iOS.

Google Calendar permite utilizatorilor să creeze și să editeze evenimente. Notificările pot fi activate pentru evenimente, cu opțiuni disponibile pentru tip și timp.

Printre cele mai utilizate funcționalități ale acestui calendar se numără:

- Crearea de multiple calendare pentru diferite nevoi
- Programarea de întâlniri cu grupuri de prieteni folosind funcția "Find a Time" sau "Suggested Time"
- Adăugarea unui event de tip Google Hangout
- Adăugarea de atașamente
- Trimiterea de e-mailuri către invitații la un eveniment

3.3.3. My Study Life

O prima referință este platforma MyStudyLife[2] ce permite utilizatorului înregistrat sa își seteze Task-uri, Exams si sa își construiască un orar (denumit Schedule în cadrul aplicației).

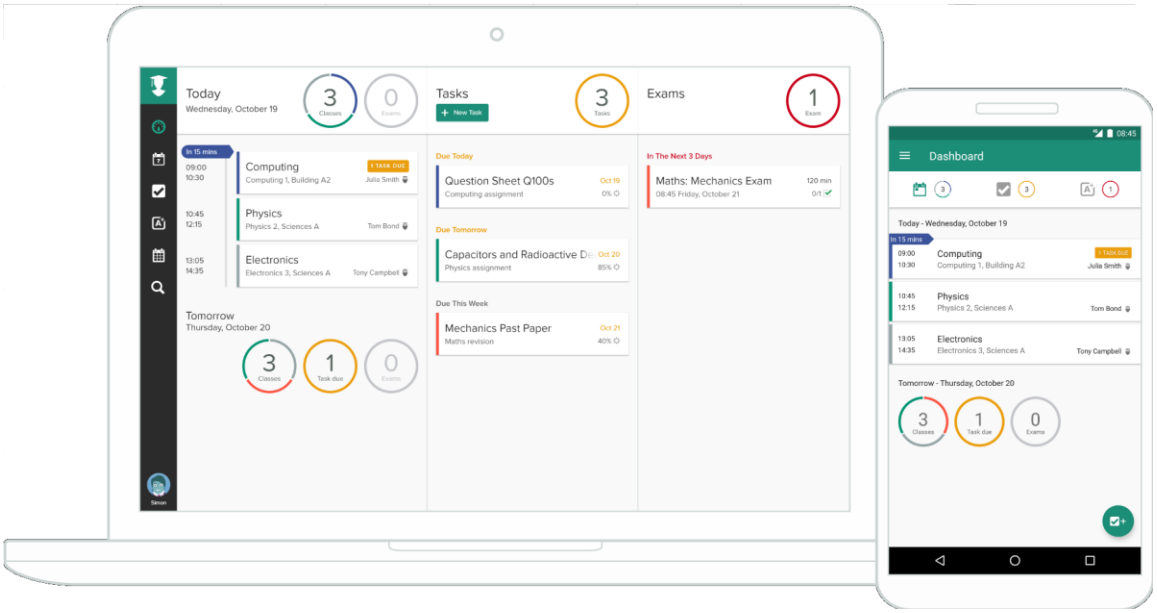

Figura 3.2 Interfață My Study Life

Prin ce se va diferenția aplicația dezvoltată în cadrul acestui proiect, denumită în continuare Student Planner, de acesta este faptul ca studentul nu trebuie sa își definească detaliile legate de examene si structura anului universitar, deoarece acestea vor fi importate automat in funcție de activitățile adăugate de administrator si detaliile pe care le stabilesc profesorii responsabil de cursurile contractate. În anul 2010, fondatorul acestei platforme a absolvit școala secundară și a început să studieze la universitate.

[1] https://support.google.com/a/users/answer/9302892?hl=en
[2] https://www.mystudylife.com/tour

Frustrat de complexitatea și combinația de platforme de învățare, de intranet și de planificatori de hârtie, a decis sa dezvolte o platformă care ar putea să le agrege pe toate.

Printre cele mai utile funcționalități ale acestei aplicații se află:

- Capabilitatea de a defini ani, semestre și vacanțele dintre ele
- Posibilitatea de a marca ce procent dintr-o anumită temă a fost completat
- Capabilitatea de a crea obiective de tipul „recapitulate”
- Setarea de alerte

3.4. Aplicațiile web

O aplicație web este un program care se accesează printr-o rețea de calculatoare folosind un browser web. De obicei, o asemenea aplicație este compusă din două părți: o componentă de server care furnizează și stochează date pentru utilizator și o componentă client care este executată în browserele web ale utilizatorilor ca HTML.

Principalul avantaj al unei asemenea aplicații este răspândirea browserelor web pe orice echipament electronic ce suportă o conexiune la internet. De aceea aplicațiile web sunt considerate a fi una dintre platformele software cu cea mai largă utilizare. Un alt avantaj important ce m-a făcut să aleg dezvoltarea unei aplicații web este capacitatea acestora de a fi actualizate și menținute fără a trebui să se distribuie sau să instaleze un software. Dezvoltatorul unei astfel de aplicații trebuie doar să actualizeze conținutul pe server pentru ca toți clienții să obțină versiunea actualizată, astfel efortul din partea clientului este minim în a folosi ultima versiune disponibilă.

De-a lungul timpului, o componentă importantă ce a dus la adoptarea aplicațiilor web este capabilitatea de a le stiliza folosind Cascading Style Sheets (CSS). Datorită acestora, paginile web pot fi decorate și așezate fără a afectată conținutul propriu-zis al acestora. Stilizarea paginilor web permite dezvoltatorilor să facă ca acestea să se comporte și să arate întocmai unei aplicații tradiționale desktop.

Pentru a dezvolta o aplicație web există un anumit set de reguli numite „best practices” ce ar trebui urmate. În primul rând, un rol foarte important îl constituie alegerea tehnologiilor. Selectarea tehnologiei potrivite este o decizie importantă care trebuie făcută atât pe baza experienței cât și a tendințelor în programare. Unele dintre deciziile importante care trebuie luate vor include selectarea limbajelor de programare, framework-urilor, modalității de păstrare a bazei de date și a serverului web ce va găzdui aplicația. De asemenea este important să se cunoască pentru ce platforme trebuie să fie disponibilă aplicația. Acest lucru poate varia de la un produs la altul, deoarece anumite aplicații web sunt utile doar pe un laptop sau un computer desktop, iar pentru dispozitive mobile să fie necesare alte produse disponibile pentru a augmenta produsul existent. Unul dintre aspectele cele mai provocatoare ale dezvoltării aplicațiilor web este crearea de aplicații care să fie scalabilă. Pentru o sesiune normală de utilizare, trebuie stabilit ce procent din unitatea centrală de procesare este utilizat (poate fi mare dacă există interogări care nu sunt optimizate și prelucrează informația prea mult timp pentru a obține un singur rezultat), lățimea de bandă utilizată, câte scrieri și citiri se fac în baza de date și ce capacitate de stocare este necesară. Optimizări ale acestor metrici pot fi făcute prin optimizarea bazei de date, a interogărilor, folosirea unei strategii de cachare pentru a minimiza numărul de scrieri și citiri către baza de date și folosirea de servere dedicate pentru module independente.

Există mai multe tipuri arhitecturi de aplicații web, una din acestea fiind single-page application, caracterizată prin faptul că întreaga funcționalitate a aplicației poate fi vizibilă într-o singură pagină HTML. Pe partea de client, această pagină are la bază layer-ul de JavaScript prin care se realizează comunicarea cu serviciile web de pe server. Astfel, utilizând datele provenite prin intermediul serviciilor web, se vor face actualizări în timp real în pagină. Modul în care se realizează comunicarea poate fi observat și în diagrama de mai jos:

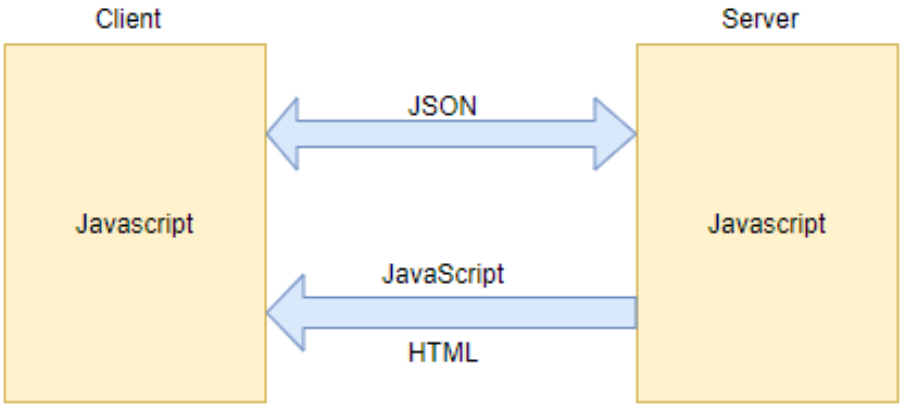

Figura 3.3 Comunicarea între client și server în single-page application

Capitolul 4. Analiză și Fundamentare Teoretică

4.1. Cerințe funcționale

În crearea aplicației am dorit ca aceasta să poată fi accesată de două tipuri diferite de utilizatori (student și administrator) care au fiecare diferite drepturi.

În cazul unui utilizator acesta va putea să realizeze următoarele operațiuni prezentate în tabelul de mai jos în funcție de rolul său în aplicație.

Tabel 4.1 Cerințe funcționale

Nr. crt.	Caz de utilizare	Actor
1	Autentificarea/înregistreze în aplicație	student
2	Vizualizarea unui sumar al activităților din ziua curentă	student
3	Adăugarea în calendar de teme pentru oricare din materiile contractate	student
4	Adăugarea în calendar a unei date de predare (deadline) pentru oricare din materiile contractate	student
5	Organizarea unui grup de studiu	student
6	Vizualizarea de grupuri de studiu deja existente	student
7	Aderarea la grupuri de studiu deja existente	student
8	Adăugarea unui mesaj la pagina de discuții a grupului de studiu la care s-a aderat	student
9	Vizualizarea calendarului in care vor apărea zilele de predare ale temelor, zilele examenelor, evenimente, vacanțe, zilele asignate pentru studiu sau alte activități marcate de acesta și zilele de întâlnire ale grupurilor de studiu	student
10	Ștergerea unei activități din calendar	student
11	Vizualizarea activităților sub formă de board	student
12	Adăugarea unui curs	administrator
13	Administrarea cursurilor	administrator
14	Setarea de date pentru examene	administrator
15	Setarea de evenimente ce vor apărea în calendarul tuturor studenților	administrator
16	Setarea de vacanțe	administrator
17	Administrarea utilizatorilor de toate tipurile	administrator

4.2. Cerințe non-funcționale

Constrângerile non-funcționale reprezintă constrângeri ale serviciilor și funcțiilor oferite de sistem, cum ar fi constrângeri de timp, standarde, constrângeri ale procesului de dezvoltare, etc.

Aplicația dezvoltată urmărește să îndeplinească cerințele non-funcționale prezentate în cele ce urmează.

CNF1. **Gradul de concurență**. Utilizatori diferiți vor putea avea acces la aplicație simultan.

CNF2. **Mentenabilitatea**. Se urmărește utilizarea unei arhitecturi de tip layers pentru a asigura separarea clară între prezentare, servicii, logica de business, metodele de acces la baza de date și modele și implicit a mentenabilității și a unui cuplaj mic.

CNF3. **Calitatea**. Funcționarea corespunzătoare se asigură prin testare.

CNF4. **Securitatea**. Accesul în pagini nu va fi permis utilizatorilor neautentificați sau cu rol necorespunzător.

CNF5. **Viteza**. Se urmărește implementarea eficientă pentru a oferi un timp de răspuns bun request-urilor utilizatorilor.

CNF6. **Extensibilitatea.** Aplicația este construită în așa fel încât să poată fi extinsă în viitor.

CNF7. **Adaptabilitatea**. Aplicația este construită în așa fel încât să poată fi adaptată cerințelor noi.

CNF8. **Utilizabilitatea**. Interfața cu utilizatorul este construită astfel încât orice operație să fie intuitivă și să se realizeze cu efort minim din partea utilizatorului.

CNF9. **Integritatea datelor**. Se asigură prin back-up periodic al bazei de date.

CNF10. **Portabilitate**. Această cerință ar trebui implicit satisfăcută de faptul că aplicația este una web. Astfel, este accesibilă utilizatorilor de pe diverse platforme.

CNF11. **Securitate.** Securitatea sistemului se realizează prin mai multe modalități. În primul rând, accesul la contul individual se face pe baza unui nume de utilizator și a unei parole. Astfel nici o persoană nu poate accesa contul unei alte persoane. În al doilea rând, paginile specifice unui anumit rol nu pot fi vizualizate de rolurile neautorizate. În ultimul rând, fiecare vizită e aferentă unei sesiuni de utilizare. Sesiunea se încheie în momentul în care browserul se închide. Astfel, în cazul folosirii unui computer partajat pentru a intra în aplicație, riscul ca altcineva să folosească neautorizat contul cuiva este diminuat.

4.3. Tehnologii folosite

4.3.1. Limbajul C#

C# este un limbaj modern de programare orientat obiect, creat și dezvoltat de Microsoft împreună cu platforma .NET. Există o multitudine de programe software dezvoltate cu C# și platforma .NET: aplicații de birou, aplicații web, site-uri web, aplicații desktop, aplicații mobile, jocuri și multe altele. Acesta este un limbaj de nivel înalt, similar cu Java și C++ și, într-o oarecare măsură, cu limbaje precum Delphi, VB.NET și C. Toate programele C# sunt orientate pe obiect [5]. Acestea constă dintr-un set de definiții de clasele ce conțin metode, iar metodele la rândul lor conțin logica

programului, adică instrucțiunile pe care le execută computerul. În ziua de astăzi C# este unul din cele mai populare limbaje de programare, fiind folosit de milioane de dezvoltatori din întreaga lume. Datorită faptului că este construit de către Microsoft ca parte a platformei lor de dezvoltare și execuție, îl face să fie răspândit în rândul companiilor orientate către tehnologii Microsoft (spre exemplu care lucrează cu Project Server, Project Online sau SharePoint), dar și printre dezvoltatori independenți.

Limbajul C# este în exclusivitate deținut de Microsoft și nu este deschis către terțe părți. De aceea alte mari corporații preferă să folosească Java ca principal limbaj de programare pentru a-și dezvolta propriile produse, acesta fiind open-source.

C# este distribuit împreună cu un mediu special pe care este executat, numit Common Language Runtime (CLR). Acest mediu face parte din platforma .NET Framework, care include CLR, un pachet de biblioteci standard care oferă funcționalități de bază, compilatoare, depanatoare și alte instrumente de dezvoltare. Datorită programului cadru, programele CLR sunt portabile și odată ce au fost scrise, pot funcționa cu mici sau fără modificări pe diferite platforme hardware și sisteme de operare. Programele C# se execută cel mai frecvent pe Windows, însă .NET Framework și CLR suportă de asemenea telefoane mobile și alte dispozitive portabile bazate pe Windows Mobile, Windows Phone și Windows 8. Programele pot fi rulate și sub Linux, FreeBSD, iOS, Android, MacOS X sau alte sisteme de operare prin .NET Framework Mono, care cu toate că este gratuit, nu este suportată în mod oficial de Microsoft.

4.3.2. .NET Framework

După cum am menționat și mai sus, limbajul C# nu este distribuit ca produs de sine stătător ci face parte din platforma Microsoft .NET Framework. [6]

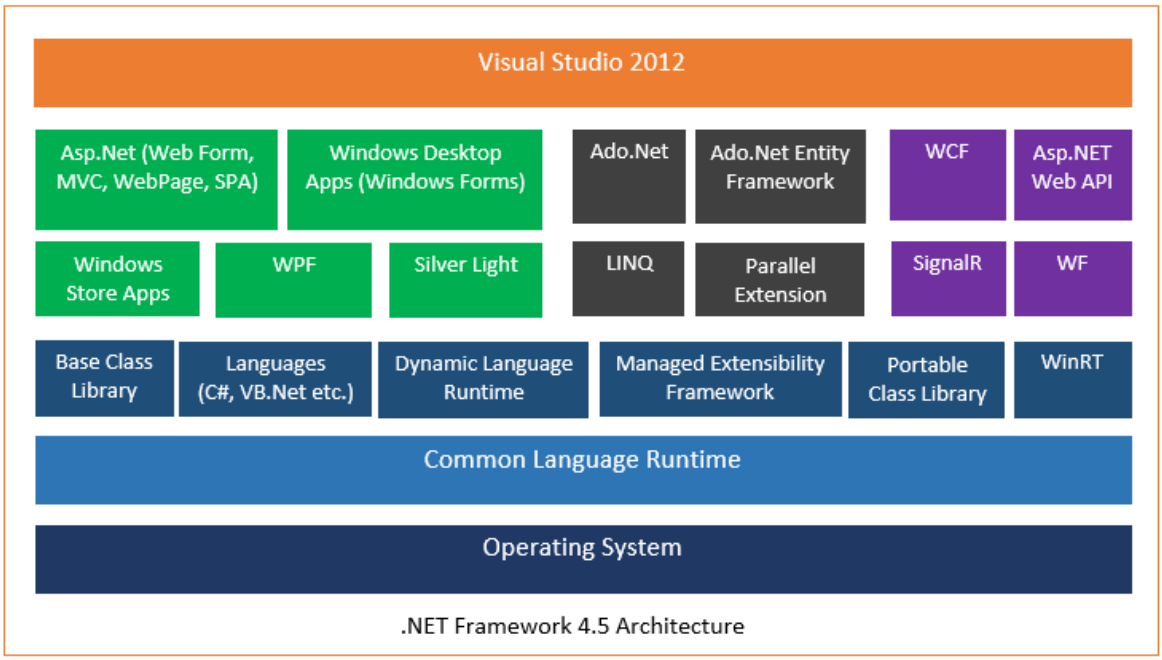

Figura 4.1 Arhitectura .Net Framework

Aceasta este un mediu pentru dezvoltarea și executarea programelor scrise în C# sau în alte limbaje, compatibilă cu .NET, cum ar fi VB.NET, Managed C++, J# sau F#. Acest Framework este format din:

- limbajele de programare suportate .NET (C #, VB.NET și altele);
- mediul CLR pentru execuția codului, care execută programele C# într-un mod controlat;
- un set de instrumente de dezvoltare, cum ar fi compilatorul CSC, al cărui scop este să transforme programele C# în cod intermediar (numit MSIL), pe care CLR îl poate înțelege;
- un set de biblioteci standard, cum ar fi ADO.NET, care permit accesul la baze de date sau WCF care conectează aplicațiile prin căi standard de comunicare și protocoale precum socket-uri HTTP, REST, JSON, SOAP și TCP.

.NET Framework face parte din toate distribuțiile moderne Windows și are mai multe versiuni.

Cele menționate mai sus și conceptele moderne disponibile prin acest Framework îl face să fie un mediu de lucru propice pentru dezvoltatori. Printre acestea concepte se numără capabilitatea de a face interogări asupra colecțiilor prin intermediul LINQ și posibilitatea de a folosi variabile generice.

4.3.3. SQL și SQL Server

SQL (Structured Query Language) este un limbaj de baze de date conceput pentru gestionarea datelor în sistemele de gestionare a bazelor de date relaționale (RDBMS). SQL este standardizat și a fost inițial dezvoltat de IBM pentru interogarea, modificarea și definirea bazelor de date relaționale, folosind declarații. Capabilitățile limbajului SQL sunt următoarele:

- poate executa interogări către baze de date
- poate prelua date
- poate insera înregistrări într-o bază de date
- pot actualiza înregistrările într-o bază de date
- pot șterge înregistrări dintr-o bază de date
- pot crea noi baze de date
- pot crea noi tabele într-o bază de date
- pot crea proceduri stocate într-o bază de date
- poate crea vederi într-o bază

În plus, un alt mare avantaj al acestui limbaj este capabilitatea de a crea permisiuni asupra bazelor de date, tabelelor sau vederilor pentru diverși utilizatori sau grupuri de utilizatori.

Chiar dacă SQL este un standard, multe dintre sistemele de gestiune ale bazelor de date implementează propria versiune a limbajului SQL, iar printre acestea se numără și SQL Server. SQL Server utilizează T-SQL (Transact-SQL), acesta fiind o extensie foarte asemănătoare cu SQL standard, dar suportă și câteva funcționalități suplimentare, încorporate cum ar fi includerea programării procedurale, variabile locale sau diverse funcții ajutătoare pentru prelucrare șirurilor de caractere.

4.3.4. JavaScript

JavaScript a apărut în 1995 ca o modalitate de a adăuga programe funcționale în paginile web suportate de browserul Netscape Navigator. Limbajul a fost adoptat între timp de toate celelalte browsere web grafice. [7] Cel mai mare avantaj pe care îl prezintă acest limbaj este faptul că utilizatorul poate interacționa cu pagina fără a o reîncărca după fiecare acțiune. După adoptarea sa în afara Netscape, a fost scris un document standard pentru a descrie modul în care ar trebui să funcționeze limbajul JavaScript numit standardul ECMAScript. De-a lungul timpului au apărut diverse versiuni, cea mai recentă fiind versiunea 6, apărută în anul 2015.

JavaScript este un limbaj preferat de dezvoltatori deoarece este înțeles în mod direct de către orice browser, lucru care face o aplicație scrisă în acest limbaj să fie versatilă.

4.3.5. Bootstrap

Bootstrap este un framework puternic destinat să îmbunătățească modalitățile de dezvoltare a interfețelor web. Versiunea 3 vine cu o multitudine de caracteristici, cum ar fi variabile LESS[3], componente personalizate și pluginuri care ajută utilizatorii proiectează interfețe de utilizator dinamice. Aceste variabile LESS sunt utile prin faptul că permit dezvoltatorului să definească valorile repetitive din cadrul fișierelor de stilizare în variabile, ce vor fi ulterior înlocuite cu valoare efectivă la postprocesare. Odată cu dezvoltarea de rețelei de telefoane mobile inteligente, acestea împreună cu tabletele devin un standard utilizarea Internetului, deci este esențial ca site-urile web să fie dezvoltate dintr-o perspectivă mobilă și apoi adaptată la ecrane mai mari pentru desktopuri și notebook-uri [8].

S-a optat pentru Bootstrap, deoarece principalul avantaj al ultimei versiuni de Bootrap este faptul că a venit cu conceptul de „mobile first”, adică orice aplicație dezvoltată cu ajutorul acestui framework are trebui să fie în primul rând funcțională pe ecranele dispozitivelor mobile.

4.3.6. AJAX

AJAX, acronim pentru Asynchonus JavaScript and XML, este un set de tehnici de creare a site-urilor web interactive și a aplicațiilor web. Principala idee din spatele acestui concept este de a face ca ceea ce apare într-o pagină web să se asemene cât mai mult cu aplicațiile desktop. [9]

Folosind apeluri AJAX se face posibilă actualizarea undei pagini web, utilizând datele preluate de la browser, fără a reîmprospăta pagina și fără a aștepta după alte operații să se termine de efectuat.

4.3.7. IIS

IIS vine de la Internet Information Service, în trecut cunoscut și ca Internet Information Server. Acesta este dezvoltat de Microsoft și joacă rolul de web server. Acesta este folosit pentru a găzdui o aplicație pe serverul Web și este accesibil din sistemele client prin intermediul browserului web. În cazul prezentei aplicații, după cum

[3] Leaner Style Sheets – tip de limbaj postprocesat folosit pentru designe-ul paginilor web

am menționat este o aplicație web a fost nevoie de o asemenea tehnologie pentru a putea hosta API-ul și interfața grafică. Am ales IIS, deoarece este integrat cu mediul de dezvoltare și se mapează pe restul tehnologiilor folosite de către mine în construirea aplicației.

4.3.8. Mediul de dezvoltare

Pentru a construi această aplicație web am configurat într-o mașină virtuală ce rulează Windows Server 2016 un mediu de dezvoltare ce conține toate utilitarele necesare în acest scop. Principalul avantaj în folosirea unei mașini virtuale ca mediu de hosting este faptul că poate fi ușor portată pe un alt server prin replicare. Portarea se poate realiza chiar și fără a opri în primă fază conexiunea către serverul inițial, acest lucru oferind utilizatorului un timp de funcționare mai mare. În plus, ca mediu de lucru, o mașină virtuală va rula toate procesele necesare dezvoltării de aplicații doar când este nevoie.

Ca sistem de operare am ales Windows Server 2012 R2 cu modulul IIS instalat și configurat. Ca IDE am ales Visual Studio 2017. Acesta este folosit pentru a dezvolta programe de calculator pentru Microsoft Windows, dar și site-uri web, aplicații web sau servicii web. Visual Studio utilizează platforme de dezvoltare software Microsoft cum ar fi Windows API, Windows Forms sau Windows Presentation Foundation. Pentru a testa aplicația pe parcursul implementării am optat pentru browserul Google Chrome datorită funcționalității sale numită Developer Tools ce mi-a permis să urmăresc aplicația în diferiți pași ai execuției sale pe client, dar și performata acesteia. De asemenea am ales și utilitarul Postman pentru a urmări rezultatele ce vin de la backend prin API.

4.4. Strategii de dezvoltare

În dezvoltarea aplicației am abordat strategii diferite de dezvoltare pentru partea de *back-end* și pentru partea de *front-end*.

În ceea ce privește partea de *back-end* am mers pe ideea de dezvoltare bottom-up. Implementarea bottom-up începe cu cele mai de jos componente ale sistemului. Acestea sunt componentele pe care le folosește întreaga aplicație, dar care nu folosesc nimic din ceea ce este dezvoltat după ele. [10] Acest tip de implementare începe cu un design arhitectural software. Cu toate că este o abordare foarte bună pentru a crea o bază solidă a aplicației, principalul dezavantaj este că nu există un sistem vizibil funcțional până la o etapă mult mai târzie de dezvoltare. Cu toate că sistemul poate fi testat treptat, nu există până la final o idee concretă vizuală a ceea ce face aplicația finală.

Pentru partea de *front-end* am ales strategia Feature Driven Development (FDD). Aceasta este una des întâlnită și recomandată în practicarea unei dezvoltări AGILE a unui produs. Folosind Feature Driven Developement m-am bazat pe dezvoltarea modulelor în jurul cerințelor funcționale. Aceasta are cinci stagii, după cum urmează:

- Stagiul unu: constă în crearea unui model ce reprezintă întreaga funcționalitate. În cazul de față, s-a decis crearea unui set de mock-up-uri ale interfeței grafice.
- Stagiul doi: presupune crearea unei liste de funcționalități, acestea fiind abordate deja la începutul dezvoltării.
- Stagiul trei: implică crearea unui plan în jurul funcționalităților. Aici s-a realizat împărțirea în cele patru module descrise într-un capitol anterior

- Stagiul patru: reprezentat de faza de design. În acest stagiu s-a împărțit interfața în cele patru module reieșite din stagiul trei.
- Stagiul cinci: faza de construcție. Dezvoltarea se face cu scopul de a cerea câte o funcționalitate pe rând ce poate fi testată când este gata.

Atât cele două strategii alese pentru implementarea părților de *front-end* și *back-end* au fost combinate cu principiul testării timpurii (Early Testing Principle). Acesta sugerează că testarea ar trebui făcută cât de devreme posibil pentru a prinde eventualele erori de programare din timp. Astfel, pe măsură ce faza de implementare a fiecărei dintre funcționalități s-a terminat acestea au fost testate manual după cazurile de utilizare specifice.

4.5. Cazuri de utilizare

Un caz de utilizare este folosit pentru a identifica și categorisi cerințele unui sistem din perspectiva utilizatorului. Fiecare caz de utilizare implică un actor, un eventual set de condiții de îndeplinit înaintea efectuării scenariului și o secvență de pași ce duc la una sau mai multe finalități ale scenariului descris. În general, finalitățile acțiunilor pot fi considerate pozitive sau negative. În cele pozitive se ajunge prin fluxul așteptat de pași. În cele negative se ajunge prin cazuri speciale de utilizare și duc de obicei la erori ce ar trebui tratate.

Cazurile de utilizare oferă o privire de ansamblu asupra acțiunilor pe care un anumit tip de utilizator le poate face când este autentificat în aplicație. Astfel, comportamentul sistemului poate fi dedus din acestea.

4.5.1. Actori

În acest sistem se pot autentifica două tipuri diferite de utilizatori. Fiecare dintre aceste două tipuri de utilizatori are dreptul de a executa acțiuni specifice rolului atribuit. Aceste două tipuri sunt:

- Student – utilizator normal, ce poate executa acțiuni doar asupra propriilor activități și asupra grupurilor
- Administrator – utilizator ce poate executa acțiuni asupra utilizatorilor și de management

Cum se poate observa și în figura 4.2, în cazul studentului acesta poate:

- să se autentifice
- să își organizeze propriile activități
- să efectueze operații asupra grupurilor de studiu

În ceea ce privește administratorul, după cum se poate deduce și din figura 4.3, acesta poate:

- să efectueze acțiuni de management asupra conturilor de utilizator
- să administreze cursurile
- să administreze profesorii în aplicație
- să administreze orarul

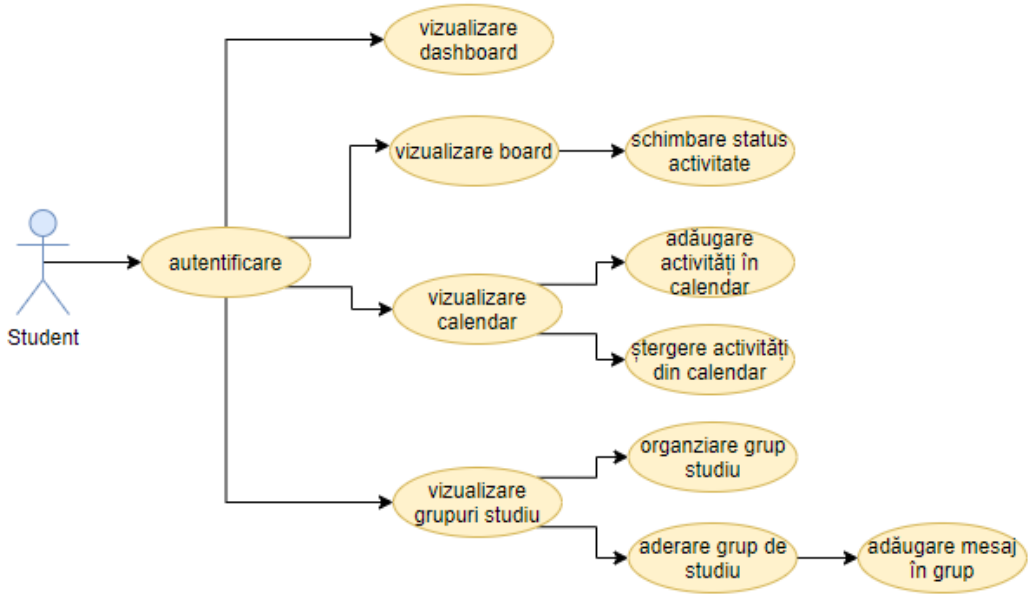

Figura 4.2 Diagrama cazurilor de utilizare student

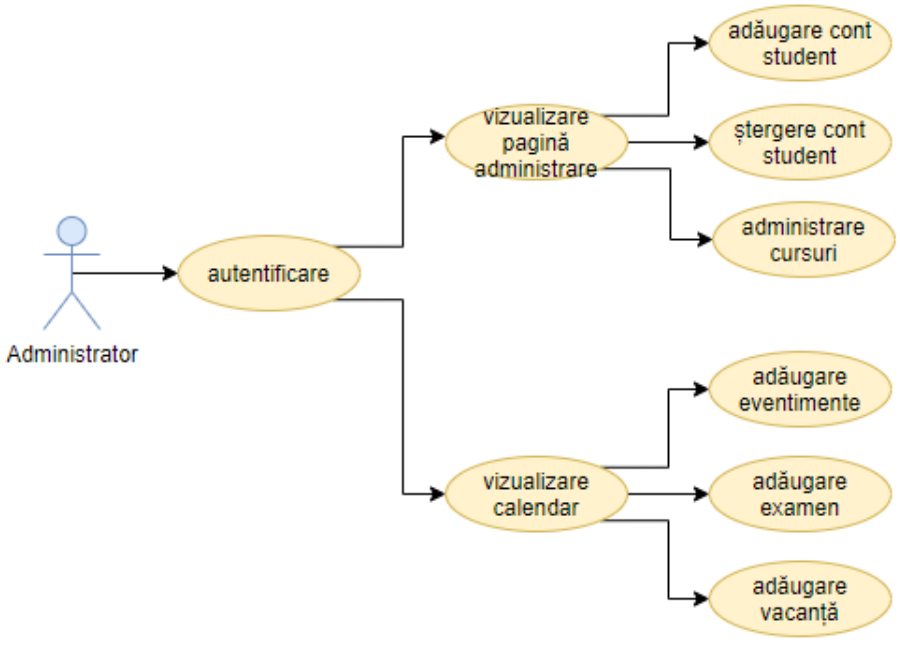

Figura 4.3 Diagrama cazurilor de utilizare administrator

4.5.2. Descrierea detaliată a cazurilor de utilizare

Caz de utilizare 1

Titlu: **Autentificare**

Descriere: În cadrul acestui caz de utilizare, actorul se autentifică în aplicație pentru a putea avea acces la paginile dedicate rolului său.

Actor: Student, Administrator

Precondiții: Utilizatorul trebuie să aibă un cont valid de utilizator și trebuie să introducă combinația corectă de nume de utilizator și parolă

Postcondiții: Pentru a accesa paginile dorite, utilizatorul trebuie să fie autentificat

Scenariu principal de succes:

1. Utilizatorul accesează pagina de autentificare
2. Pagina de autentificare este afișată, cerându-se completarea câmpurilor aferente numelui de utilizator și parolei
3. Utilizatorul introduce informațiile cerute în formularul destinat lor
4. Se preiau și se verifică informațiile
5. Dacă sunt corecte, se afișează pagina de început pentru rolul curent

Final: Autentificarea a fost făcută cu succes

Scenariu alternativ:

4. Informațiile sunt greșite
5. Un mesaj de eroare este afișat
6. Se afișează în continuare pagina de autentificare

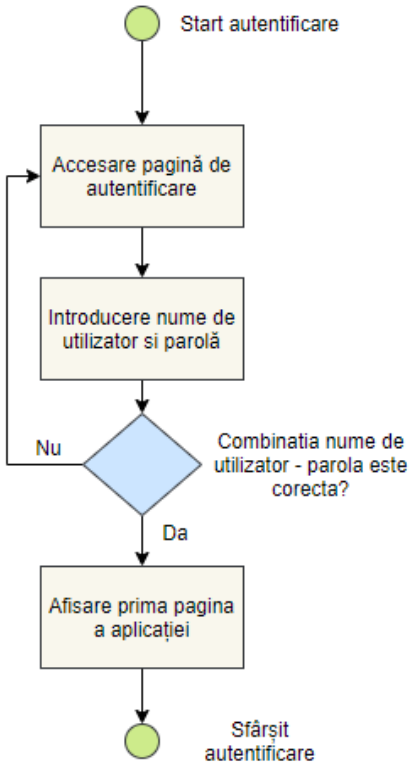

Figura 4.4 Organigrama pentru autentificare – Cazul 1 de utilizare

Caz de utilizare 2

Titlu: **Vizualizare dashboard**

Descriere: În cadrul acestui caz de utilizare, actorul vizualizează pagina Dashboard din cadrul aplicației în care i se prezintă toate activitățile setate pentru ziua curentă, atât de către el cât și cele comune tuturor utilizatorilor.

Actor: Student

Precondiții: Utilizatorul trebuie să fie autentificat în aplicației și rolul său trebuie să fie de student.

Postcondiții: Nu se aplică.

Scenariu principal de succes:

1. Din oricare dintre pagini, utilizatorul apasă butonul „Dashboard” din meniul de navigare

Final: Dashboard-ul va fi vizualizat ca pagina principală după autentificare.

Caz de utilizare 3

Titlu: **Vizualizare board**

Descriere: În cadrul acestui caz de utilizare, actorul vizualizează pagina Board din cadrul aplicației în care i se prezintă toate activitățile împărțite în trei categorii: New, In Progress și Done.

Actor: Student

Precondiții: Utilizatorul trebuie să fie autentificat în aplicației și rolul său trebuie să fie de student.

Postcondiții: Nu se aplică.

Scenariu principal de succes:

2. Din oricare dintre pagini, utilizatorul apasă butonul „Board” din meniul de navigare

Final: Utilizatorul vizualizează pagina cu board-ul.

Caz de utilizare 4

Titlu: **Vizualizare calendar**

Descriere: În cadrul acestui scenariu de utilizare, actorul vizualizează o pagină din cadrul aplicației în care i se prezintă toate activitățile setate pentru ziua curentă, atât de către el cât și cele comune tuturor utilizatorilor sub forma unor etichete într-un calendar web.

Actor: Student

Precondiții: Utilizatorul este autentificat în aplicație cu un cont ce are rolul de student setat.

Postcondiții: Nu se aplică.

Scenariu principal de succes:

3. Din oricare dintre pagini, utilizatorul apasă butonul „Calendar” din meniul de navigare

Final: Utilizatorul vizualizează pagina cu calendarul.

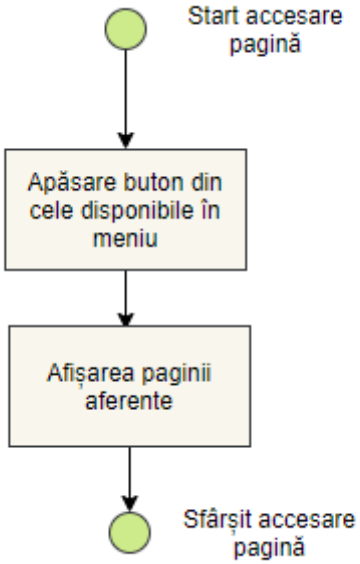

Figura 4.5 Organigrama accesării undei pagini - Cazul 2,3,4 de utilizare

Caz de utilizare 5

Titlu: **Adăugare activități în calendar**

Descriere: În cadrul acestui scenariu de utilizare, actorul se află în pagina „Calendar” și dorește să își seteze activități noi de orice tip din cele disponibile pentru rolul său.

Actor: Student, Administrator

Precondiții: Utilizatorul este autentificat în aplicație cu un cont ce are rolul de student sau administrator setat și se află în pagina „Calendar” a aplicatei.

Postcondiții: Utilizatorul a completat toate câmpurile cu informațiile cerute și a asignat noua etichetă creată pentru activitate pentru o dată din calendar.

Scenariu principal de succes:

1. Utilizatorul se află în pagina „Calendar”
2. Utilizatorul completează câmpul „Name” din formularul de adăugare a unei noi activități, îi alege un tip și îi asociază o materie.
3. Utilizatorul apasă butonul „Add” pentru a crea o nouă etichetă aferentă activității cu informațiile furnizare
4. Utilizatorul pune eticheta printr-o acțiune de tip „drag and drop” într-una din căsuțele calendarului, aferentă datei pentru care dorește să seteze activitatea

Final: Activitatea apare în calendar și este adăugată în baza de date de pe server

Scenariu alternativ:

2. Utilizatorul nu completează câmpul „Name”.
3. Utilizatorul apasă butonul „Add” pentru a crea o nouă etichetă aferentă activității cu informațiile furnizare
4. Un mesaj de eroare este afișat.
5. Eticheta nu este creată.

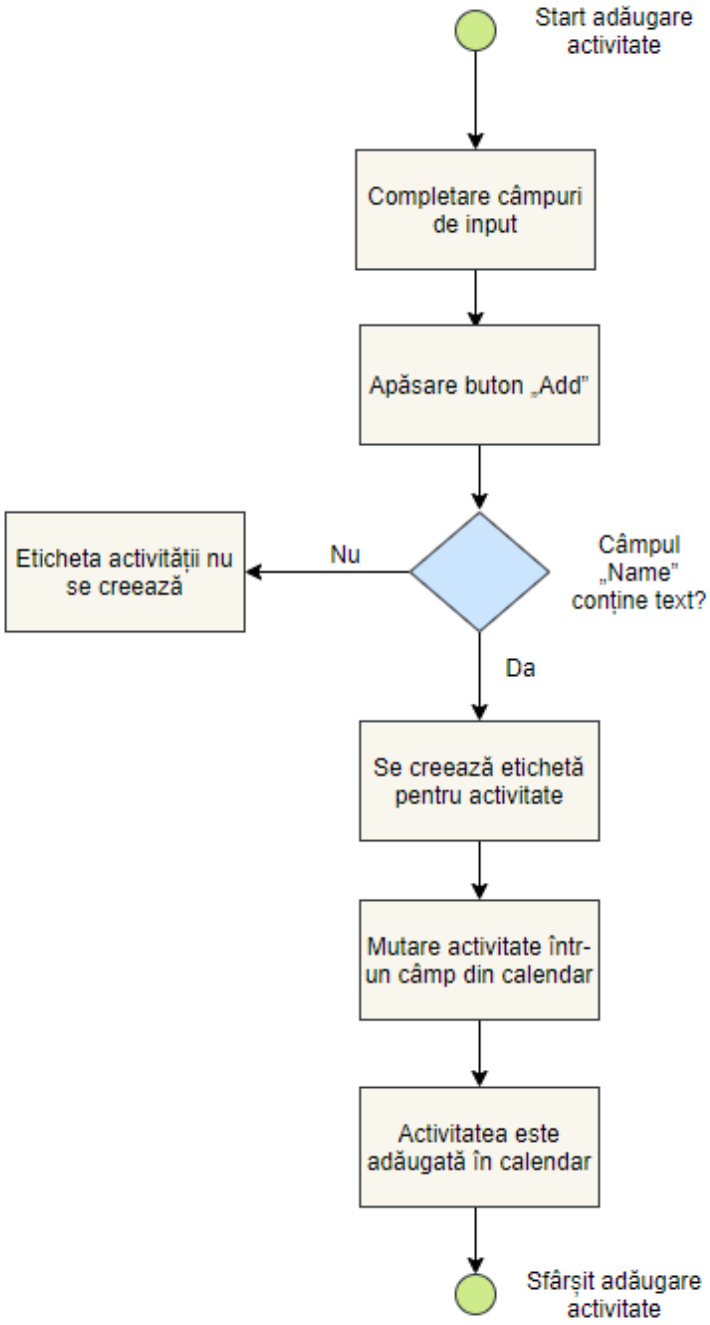

Figura 4.6 Organigrama adăugării de activităților în calendar

Caz de utilizare 6

Titlu: **Ștergere activitate din calendar**

Descriere: În cadrul acestui scenariu de utilizare, actorul se află în pagina „Calendar” și dorește să șteargă una din activitățile setate anterior ce apare în calendar.

Actor: Student, Administrator

Precondiții: Utilizatorul este autentificat în aplicație cu un cont ce are rolul de student sau administrator setat și se află în pagina „Calendar” a aplicației, iar activitatea ce dorește să o șteargă se află deja în calendar.

Postcondiții: Activitatea se regăsește în baza de date de pe server.

Scenariu principal de succes:

1. Utilizatorul este autentificat în aplicație
2. Utilizatorul se află în pagina „Calendar”
3. Utilizatorul identifică eticheta activității ce dorește a o șterge
4. Utilizatorul apasă butonul ce indică acțiunea de ștergere de pe eticheta activității

Final: Activitatea dispare din calendar și este ștearsă și din baza de date de pe server.

Caz de utilizare 7

Titlu: **Organizare grup de studiu**

Descriere: Student

Actor: În cadrul acestui scenariu de utilizare, actorul se află în pagina „Study Groups” și dorește să creeze un grup de studiu.

Precondiții: Utilizatorul este autentificat în aplicație cu un cont ce are rolul de student setat și se află în pagina „Study Groups” a aplicației.

Postcondiții: Utilizatorul a completat formularul de creare a unui nou grup de studiu cu toate datele cerute.

Scenariu principal de succes:

1. Utilizatorul este autentificat în aplicație
2. Utilizatorul se află în pagina „Study Groups”
3. Utilizatorul completează câmpurile din formularul de adăugare al unui nou grup de studiu.
4. Utilizatorul apasă butonul „Add” pentru a crea un nou grup de studiu cu informațiile furnizare

Final: Grupul este creat și apare în pagină.

Scenariu alternativ:

3. Utilizatorul introduce date invalide în câmpurile din formularul de adăugare

4. Un mesaj de eroare este afișat.

5. Grupul de studiu nu este creat.

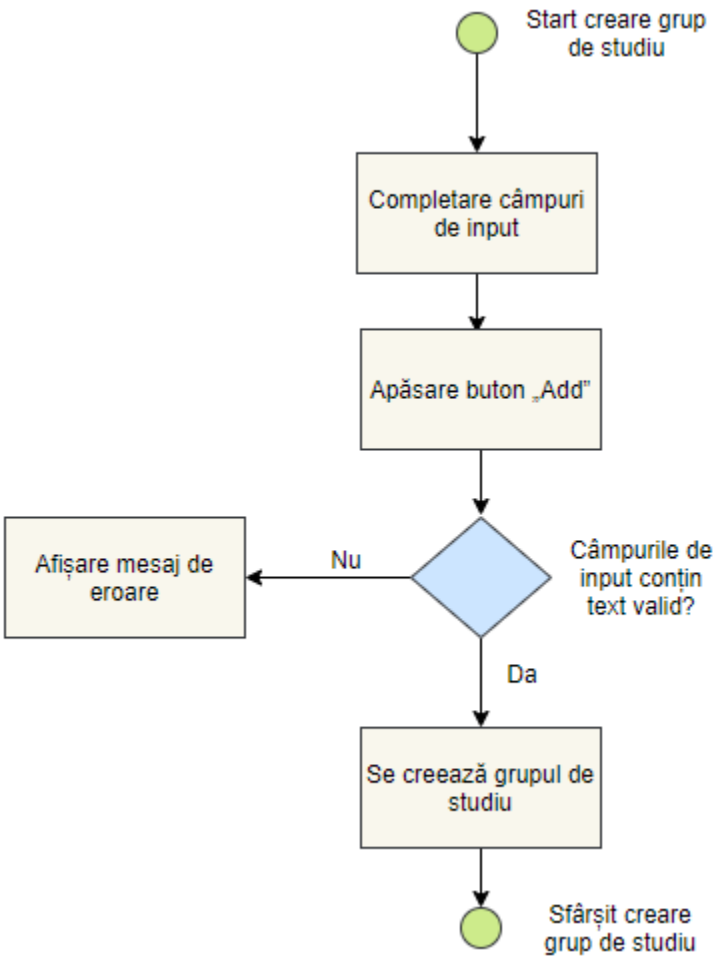

Figura 4.7 Organigrama adăugării unui grup de studiu

Caz de utilizare 8

Titlu: **Participare grup de studiu**

Descriere: În cadrul acestui scenariu de utilizare, actorul se află în pagina „Study Groups” și dorește să adere la unul din grupurile de studiu existente.

Actor: Student

Precondiții: Utilizatorul este autentificat în aplicație cu un cont ce are rolul de student setat și se află în pagina „Study Groups” a aplicației, iar grupul de studiu la care dorește să participe este vizibil în pagină.

Postcondiții: Grupul de studiu se regăsește în baza de date, iar utilizatorul nu a mai aderat la acesta

Scenariu principal de succes:

1. Utilizatorul este autentificat în aplicație
2. Utilizatorul se află în pagina „Study Groups”
3. Utilizatorul identifică grupul de studiu la care își dorește să participe
4. Utilizatorul apasă butonul ce indică acțiunea de aderare din cadrul cardului aferent grupului de studiu

Final: Utilizatorul aderă la grupul de studiu, iar informațiile sunt trimise către baza de date pentru a fi introduse în tabelele corespunzătoare.

Caz de utilizare 9

Titlu: **Adăugare mesaj în grupul de studiu**

Descriere: În cadrul acestui scenariu de utilizare, actorul se află în pagina „Study Groups” și dorește să lase un mesaj în pagina de discuții a unui grup de studiu la care a aderat anterior.

Actor: Student

Precondiții: Utilizatorul este autentificat în aplicație cu un cont ce are rolul de student setat și se află în pagina „Study Groups” a aplicației. Utilizatorul se află deja în grupul de studiu în cadrul căruia vrea să lase un mesaj.

Postcondiții: Nu se aplică

Scenariu principal de succes:

1. Utilizatorul este autentificat în aplicație
2. Utilizatorul se află în pagina „Study Groups”
3. Utilizatorul identifică grupul de studiu în cadrul căruia dorește să lase un mesaj.
4. Utilizatorul apasă butonul ce îl duce la pagina cu mesaje din cadrul grupului de studiu.
5. Utilizatorul completează câmpul destinat scrierii mesajului cu textul dorit
6. Utilizatorul apasă butonul „Send”

Final: Mesajul este postat pe pagina de discuții a grupului.

Caz de utilizare 10

Titlu: **Creare conturi studenți**

Descriere: În cadrul acestui scenariu de utilizare, actorul se află în pagina „Manage” și dorește să creeze un cont de utilizator nou pentru un student.

Actor: Administrator

Precondiții: Utilizatorul este logat în aplicație cu un cont ce are rolul de administrator setat și se află în pagina „Manage” a aplicației.

Postcondiții: Nu se aplică

Scenariu principal de succes:

1. Utilizatorul este logat în aplicație
2. Utilizatorul se află în pagina „Manage” a aplicației
3. Utilizatorul completează formularul aferent adăugării unui nou utilizator cu informațiile cerute
4. Utilizatorul apasă butonul „Add”

Final: Contul de utilizator pentru student este creat, informațiile fiind salvate în baza de date de pe server.

Scenariu alternativ:

3. Utilizatorul nu completează câmpurile sau datele nu sunt valide.
4. Utilizatorul apasă butonul „Add”
5. Contul nu este creat.

Caz de utilizare 11

Titlu: **Administrare cursuri**

Descriere: În cadrul acestui scenariu de utilizare, actorul se află în pagina „Manage” și dorește să creeze un curs nou.

Actor: Administrator

Precondiții: Utilizatorul este logat în aplicație cu un cont ce are rolul de administrator setat și se află în pagina „Manage” a aplicației.

Postcondiții: Nu se aplică.

Scenariu principal de succes:

1. Utilizatorul este logat în aplicație
2. Utilizatorul se află în pagina „Manage” a aplicației
3. Utilizatorul completează formularul aferent adăugării unui nou curs cu informațiile cerute
4. Utilizatorul apasă butonul „Add”

Final: Cursul este creat, informațiile fiind salvate în baza de date de pe server.

Scenariu alternativ:

3. Utilizatorul nu completează câmpurile sau datele nu sunt valide.
4. Utilizatorul apasă butonul „Add”
5. Cursul nu este creat.

Caz de utilizare 12

Titlu: **Administrare profesori**

Descriere: În cadrul acestui scenariu de utilizare, actorul se află în pagina „Manage” și dorește să adauge un profesor nou.

Actor: Administrator

Precondiții: Utilizatorul este logat în aplicație cu un cont ce are rolul de administrator setat și se află în pagina „Manage” a aplicației.

Postcondiții: Nu se aplică.

Scenariu principal de succes:

1. Utilizatorul este autentificat în aplicație
2. Utilizatorul se află în pagina „Manage” a aplicației
3. Utilizatorul completează formularul aferent adăugării unui nou profesor cu informațiile cerute
4. Utilizatorul apasă butonul „Add”

Final: Profesorul este adăugat, informațiile fiind salvate în baza de date de pe server.

Scenariu alternativ:

3. Utilizatorul nu completează câmpurile sau datele nu sunt valide.
4. Utilizatorul apasă butonul „Add”
5. Profesorul nu este adăugat.

Capitolul 5. Proiectare de Detaliu si Implementare

În cadrul acestui capitol se prezintă arhitectura sistemului și modalitatea prin care componentele sale interacționează între ele. De asemenea, fiecare componentă este detaliată atât pentru partea ce ține de server cât și pentru partea ce ține de web.

5.1. Arhitectura generală a aplicației

5.1.1. Diagrama arhitecturii generale a sistemului

În diagrama ce urmează a fi prezentată mai jos se poate observa că aplicația este construită pe nivele. Nivelurile comunică între ele pe baza următorului principiu: fiecare nivel știe și se folosește doar de nivelul ce se află imediat sub el. În cadrul nici unuia dintre ele nu există dependințe sau referințe către un nivel superior. Singura excepția este nivelul auxiliar de modele, ce nu se folosește de nici unul dintre ele dar este folosit în cadrul tuturor pentru a asigura consistența în cadrul aplicației.

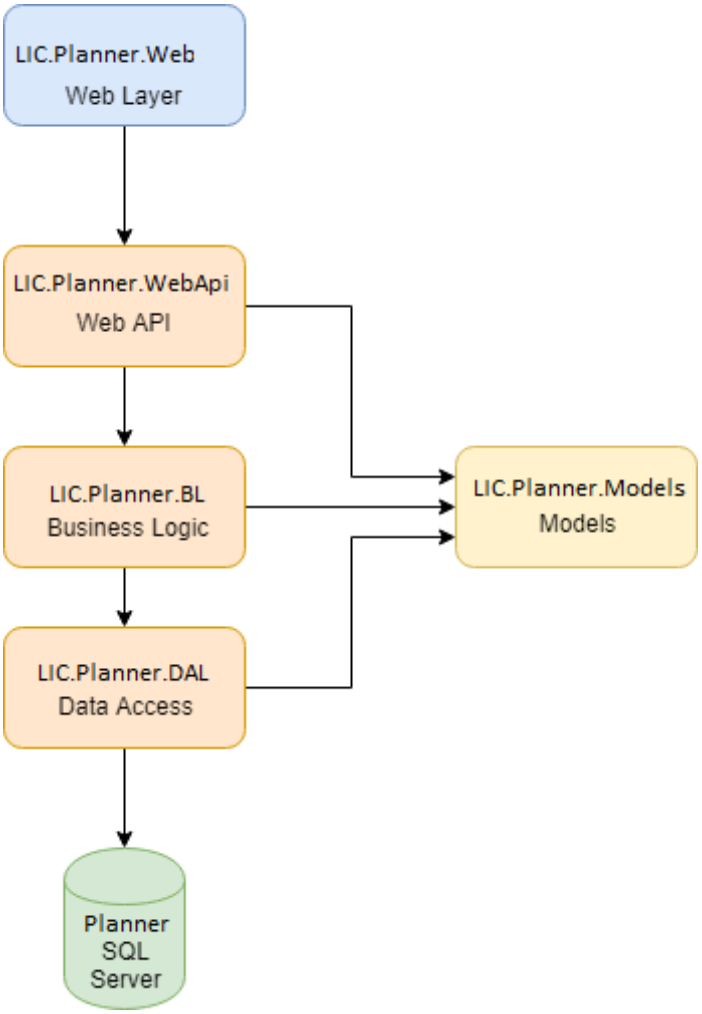

Figura 5.1 Arhitectura pe nivele și împărțirea în proiecte a sistemului

Se pot observa trei mari componente: baza de date, partea de back-end și partea de front-end. Datorită acestei separări clare, se poate spune că aplicația respectă structura uneia de tip three-tier. Baza de date reprezintă nivelul de date, partea de back-end reprezintă nivelul logic, iar partea de front-end nivelul de prezentare. În cadrul nivelului de date, se stochează informația într-o bază de date. Aceasta este trimisă sau primită de la nivelul logic. În cadrul nivelului logic, există mai multe alte sub-nivele, ce procesează comenzi, fac calcule, sau orice alte procesări necesare asupra datelor. Nivelul de prezentare este cel mai de sus nivel, iar rolul său principal este acela de a interfața capabilitățile aplicației și de a afișa informația într-un mod pe care utilizatorul să îl înțeleagă.

Baza de date este reprezentată de SQL Server. Back-end-ul este format din nivelul de acces la date (Data Access), nivelul de logică de business (Business Logic), Web API, ce expune întreaga logică de business prin servicii și nivelul auxiliar de modele (Models). Partea de front-end este reprezentată de interfața web. În cazul de față, am ales implementarea unei aplicații de tip single page, formată din controllere scrise folosind JavaScript și JQuery și o singură pagină HTML.

5.2. Nivelul de date

5.2.1. Descrierea bazei de date la nivelul serverului SQL

În scopul persistării tuturor datelor ce sunt necesare pentru o bună funcționare a aplicației, am decis să folosesc SQLServer pentru a le păstra într-o bază de date relațională. Baza de date poartă numele de Planner și la momentul implementării este găzduită pe același server pe care se află și partea de back-end a aplicației. Aceasta conține zece tabele: Activity, Attendees, Class, Course, Event, Group, Message, Professor, StudyGroup și User. Diagrama bazei de date poate fi consultată în figura 5.2, iar descrierea detaliată a acesteia va fi realizată în cele ce urmează.

Într-un rând din tabela Activity sunt stocate informații despre o anumită activitate setată de unul din actori în calendar. Coloanele acestei tabele sunt:

- ActivityID – identificator unic de tip uniqueidentifier ce are rol de cheie primară a tabelului
- Title – titlul activității de tip nvarchar cu dimensiunea maximă de 50 de caractere
- Type – tipul activității, stocat sub formă de întreg; fiecare întreg îi va corespunde unui anumit tip, iar folosirea de întregi în favoarea șirurilor de caractere este o bună practică pentru a elimina greșeli de scriere și pentru a ușura modificarea corespondentului tipului în back-end sau front-end oricând, fără a schimba intrările anterioare
- Date – data în care are loc activitatea de tipul datetime
- State – statusul în care se află activitatea (reprezentat prin 0, 1 sau 2 și corespunzând valorilor New, In Progress și Done)
- CourseID – identificatorul unic al cursului de care este legată activitatea ce are rol de cheie străină, făcând corespondența între tabelele Activity si Course
- UserID – identificatorul unic al utilizatorului care a creat activitatea ce are rol de cheie străină, făcând corespondența între tabelele Activity si User

Tabela Attendees este formată doar din două chei străine, aceasta având rolul de a rezolva legătura de tip many-to-many dintre tabelele StudyGroup și User.Explicația acesteia este dată de faptul că un utilizator de tip student poate face parte din mai multe grupuri de studiu, iar dintr-un grup de studiu pot face parte mai mulți utilizatori de tip student. Coloanele acestei tabele sunt:

- StudyGroupID – identificator de tip uniqueidentifier, ce are rol de cheie străină, făcând corespondența între tabelele Attendees și StudyGroup
- UserID – identificator de tip uniqueidentifier, ce are rol de cheie străină, făcând corespondența între tabelele Attendees și User

Pe rândurile din tabela Class se stochează toate informațiile necesare despre orele de curs sau laboratoarele adăugate de către administrator. Coloanele acestei tabele sunt:

- ClassID - identificator unic de tip uniqueidentifier ce are rol de cheie primară a tabelului
- CourseID - identificatorul unic al cursului de care este legată clasa ce are rol de cheie străină, făcând corespondența între tabelele Class și Course
- ProfessorID - identificatorul unic al profesorului de care este legată clasa ce are rol de cheie străină, făcând corespondența între tabelele Class și Professor
- GroupID - identificatorul unic al grupei de care este legată clasa ce are rol de cheie străină, făcând corespondența între tabelele Class și Group
- StartDate – data de tip datetime ce ziua din semestru în care începe cursul
- EndDate – data de tip datetime ce ziua din semestru în care se termină cursul

Tabela Course are rolul de a stoca pe rândurile acesteia informații despre un anumit curs ce se predă în cadrul facultății. Coloanele acestei tabele sunt:

- CourseID - identificator unic de tip uniqueidentifier ce are rol de cheie primară a tabelului
- Title – denumirea cursului, stocată de tipul nvarchar cu un număr maxim de 50 de caractere

Pe un rând din tabela Event sunt stocate informațiile despre un eveniment setat de către un utilizator de tip administrator în calendar. Coloanele acestei tabele sunt:

- EventID – identificator unic de tip uniqueidentifier ce are rol de cheie primară a tabelului
- Title – denumirea eventimentului, stocată de tipul nvarchar cu un număr maxim de 50 de caractere
- StartDate – data de tip datetime ce ziua din semestru în care începe evenimentul
- EndDate - data de tip datetime ce ziua din semestru în care se încheie evenimentul

În tabela Group sunt stocate informații despre grupele în care sunt repartizați studenții. Coloanele acestei tabele sunt:

- GroupID - identificator unic de tip uniqueidentifier ce are rol de cheie primară a tabelului
- Name - denumirea grupei, stocată de tipul nvarchar cu un număr maxim de 50 de caractere
- Year – anul de studiu, stocat ca întreg

Message este tabelul în care se stocheze mesajele din cadrul paginilor de discuții aferente grupurilor de studiu create de studenți, împreună cu detaliile despre acestea. Coloanele acestei tabele sunt:

- MessageID - identificator unic de tip uniqueidentifier ce are rol de cheie primară a tabelului
- StudyGroupID - identificatorul unic al grupului de studiu de care este legat mesajul ce are rol de cheie străină, făcând corespondența între tabelele Message și StudyGroup
- Timestamp – data de tip dateime ce indică data și ora la care a fost publicat mesajul pe pagina de discuții
- Description – continutul mesajului, stocat sub tipul nvarchar cu limita maximă de caractere admisă
- UserID - identificatorul unic al utilizotolului care a creat mesajul ce are rol de cheie străină, făcând corespondența între tabelele Message și User

Tabela Professor ține toate datele despre toți profesorii ce predau cursuri la facultatea pentru care este folosi sistemul. Coloanele acestei tabele sunt:

- ProfessorID - identificator unic de tip uniqueidentifier ce are rol de cheie primară a tabelului
- FirstName – numele de familie al profesorului, stocat sub forma de nvarchar de dimensiune maximă 50 de caractere
- LastName - numele mic al profesorului, stocat sub forma de nvarchar de dimensiune maximă 50 de caractere

StudyGroup este tabela pe rândurile căreia se află toate informațiile necesare pentru stocarea grupurilor de studiu create de către utilizatorii de tipul student. Coloanele acestei tabele sunt:

- StudyGroupID - identificator unic de tip uniqueidentifier ce are rol de cheie primară a tabelului
- CourseID - identificatorul unic al cursului ce are rol de cheie străină, făcând corespondența între tabelele Course și StudyGroup
- Title - denumirea grupului de studiu, stocată ca nvarchar cu un număr maxim de 50 de caractere
- Date – data la care are loc întâlnirea grupului de studiu, stocată ca datetime
- Location – locul în care are loc întâlnirea grupului de studiu, stocat sub forma de nvarchar de dimensiune maximă 50 de caractere

- Description – orice alte detalii suplimentare ce se vor a fi introduse pentru a fi cunoscute despre grupul de studiu, stocat sub forma de nvarchar de dimensiune maximă admisă

User este tabela ce asigură persistarea tuturor informațiilor despre utilizatorii înscriși în aplicație. Această tabelă conține următoarele coloane:

- UserID - identificator unic de tip uniqueidentifier ce are rol de cheie primară a tabelului
- FirstName – numele de familie al utilizatorului, stocat sub forma de nvarchar de dimensiune maximă 50 de caractere
- LastName - numele mic al utilizatorului, stocat sub forma de nvarchar de dimensiune maximă 50 de caractere
- CNP – codul numeric personal al studentului, stocat sub formă de nvarchar cu dimensiune maximă 13 caractere
- StudentNumber – numărul unic cu care studentul este identificat în cadrul facultății (spre exemplu numărul matricol), stocat sub formă de nvarchar cu dimensiune maximă 50 caractere
- Username – pseudonimul ales pentru a se face autentificarea utilizatorului în aplicație, stocat sub formă de nvarchar cu dimensiune maximă 50 caractere
- Password – parola aleasă de utilizator pentru a se face autentificarea sa în aplicație, stocat sub forma de nvarchar de dimensiune maximă admisă
- Type – tipul utilizatorului, stocat sub formă de întreg; în momentul de față 0 reprezintă tipul student, iar 1 tipul administrator
- GroupID - identificatorul unic al grupei din care face parte studentul ce are rol de cheie străină, făcând corespondența între tabelele Group și User

După cum se poate observa și in figura 5.2, baza de date respecta primele trei din cele formele de normalizare existente [11]. Asfel, se poate spune că ea se află la forma normală Boyce-Code, deci prin incluziune forma normală 3. Prima formă de normalizare este respectată prin faptul că fiecare câmp din toate tabelele conține o valoare atomică, conform numelui coloanei din care face parte. De asemenea, fiecare înregistrare este definită printr-o cheie unică. Îndeplinirea celei de a doua forme de normalizare se poate observa ușor prin faptul că toate tabelele au cheia primară formată dintr-un singur atribut. De asemenea, se poate observa faptul că toate câmpurile unei înregistrări depind de acea cheie primară. Toate atributele din relații între tabele ce nu sunt chei primare depind numai de chei primare ale acelei relații. În aces mod cea de a treia formă de normalizare este îndeplinită în varianta Boyce-Codd a acesteia, deoarece se pot observa că în tabele nu vor exista reduntanțe cauzate de chei. Spre exemplul, în cazul relației între utilizatori (User) și grupurile de studiu (Study Group), prin faptul că s-a adăugat tabela Atendees, nu vor exista intrari duplicate pentru utilizatorii ce fac parte din mai multe grupuri de studiu.

Pe lângă tabele, baza de date mai conține și proceduri stocate. Aceste proceduri stocate sunt folosite pentru a realiza operațiuni de citire, inserare, actualizare și ștergere (CRUD). Prin intermediul procedurilor stocate se face stocarea, respectiv aducerea de date. Parametrii de intrare se vor presupune a fi valizi. Înainte să ajungă la baza de date,

parametri de intrare sunt validați pe client și pregătiți în nivelul de business sub formă de obiecte ce vor mapa 1-1 structura tabelelor. Pentru fiecare tabel existent în baza de date s-au realizat minim cinci proceduri stocate. Denumirile acestora au fost date după următorul șablon: dbo.usp_[NumeTabel][NumeOperație]. Nume tabel este pus primul, cu scopul de a identifica ușor procedurile stocate ce țin de un anumit tabel și poate fi oricare dintre tabelele existente în baza de date, mai precis Activity, Attendees, Class, Course, Event, Group, Message, Professor, StudyGroup sau User. Numele operației reflectă și ceea ce face procedura stocată. Mai precis, tipurile de bază pentru care există proceduri stocate în toate tabelele sunt: SelectAll, Select, Insert, Update sau Delete. Pe lângă aceste cinci, unele tabele au și alte operații specifice implementate, cum ar fi SelectByUserID în cazul tabelei Activity. În tabelul de mai jos pot fi consultate procedurile stocate de bază împreună cu rolul lor. În anexă, există o listă detaliată cu toate procedurile stocate.

Tabel 5.1 Proceduri stocate

Tip	Denumire generică	Descriere	Parametrii
P1	dbo.usp_[NumeTabel][Select]	Aduce o linie a tabelului, corespunzătoare parametrului dat	@ID – identificatorul unic al liniei
P2	dbo.usp_[NumeTabel][SelectAll]	Aduce toate liniile din tabel	•
P3	dbo.usp_[NumeTabel][Insert]	Inserează o nouă linie în tabel conform parametrilor furnizați	Toate coloanele tabelului sunt furnizate ca parametru
P4	dbo.usp_[NumeTabel][Update]	Actualizează o anumită linie din tabel conform parametrilor furnizați	Toate coloanele tabelului sunt furnizate ca parametru
P5	dbo.usp_[NumeTabel][Delete]	Șterge o linie din tabel, corespunzătoare parametrului dat	@ID – identificatorul unic al liniei

5.2.2. Integritatea datelor

Una din cerințele non funcționale ce s-a urmărit a fi îndeplinită este păstrarea integrității datelor. Pentru a asigura acest lucru s-a decis adăugarea pe serverul bazei de date a unui task de back-up în cadrul serverului SQL. Acesta s-a realizat prin intermediul interfeței cu utilizatorul din cadrul SQL Server Management Studio. Astfel, s-a creat un plan de mentenanță ce rulează în fiecare săptămână, iar prin intermediul său se face o copie de rezervă pe disc a întregii baze de date ce poate fi folosită în cazul unui eșec al serverului, dar nu numai. În cadrul planului de mentenanță se verifică automat integritatea bazei de date și se face o curățare a memoriei prin ștergerea fișierelor ce nu mai sunt necesare. Task-urile ce rulează pot fi văzute in figura 5.2.

În cadrul planului de mentenanță toate cele trei taskuri menționate rulează la aceeași oră stabilită și anume în fiecare duminică la ora 00, așa cum se poate observa in figura 5.2.

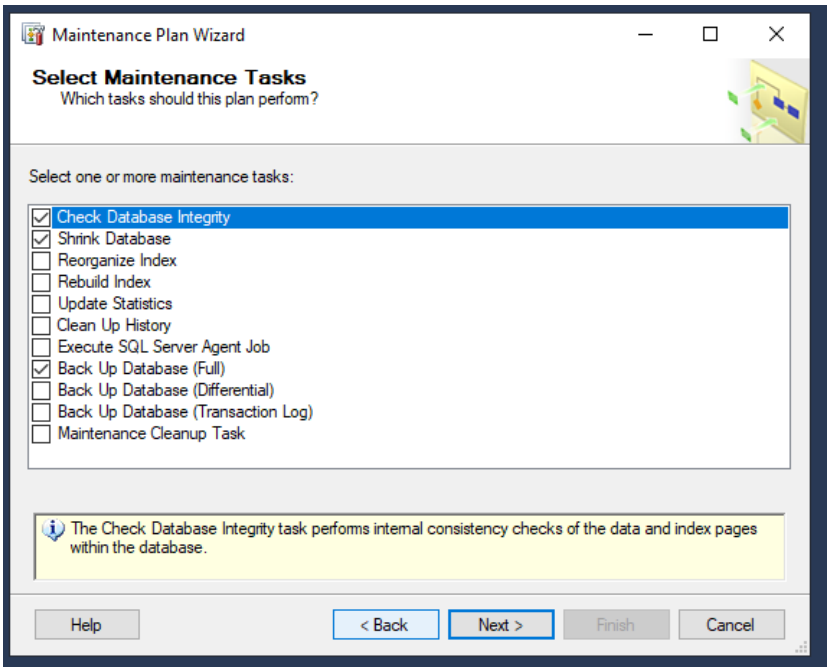

Figura 5.2 Task-urile ce rulează in cadrul planului de mentenanța

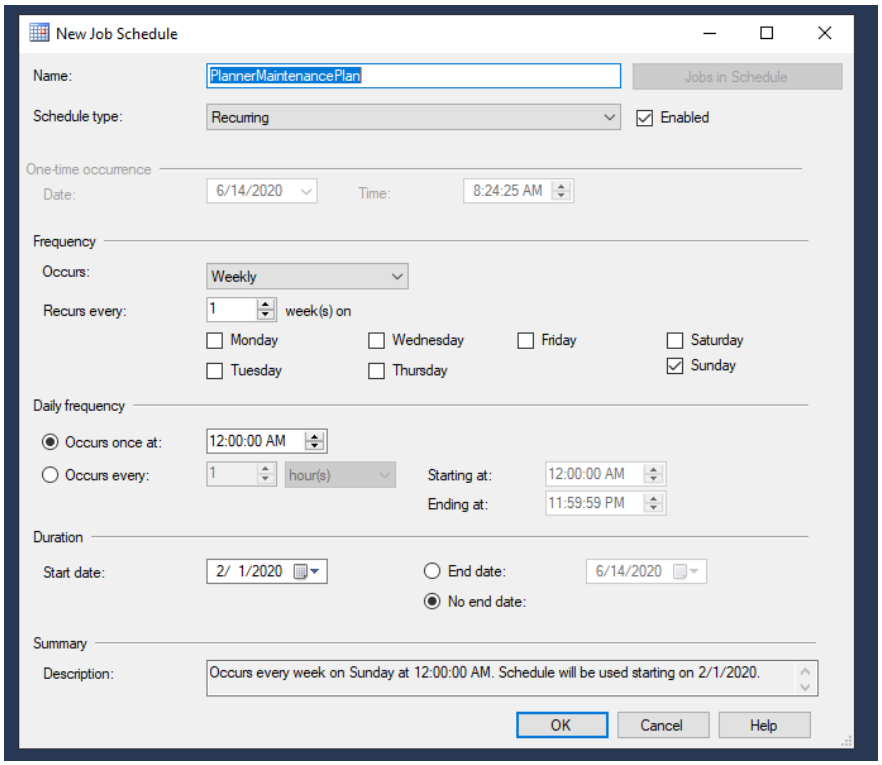

Figura 5.3 Orarul planului de mentenanța

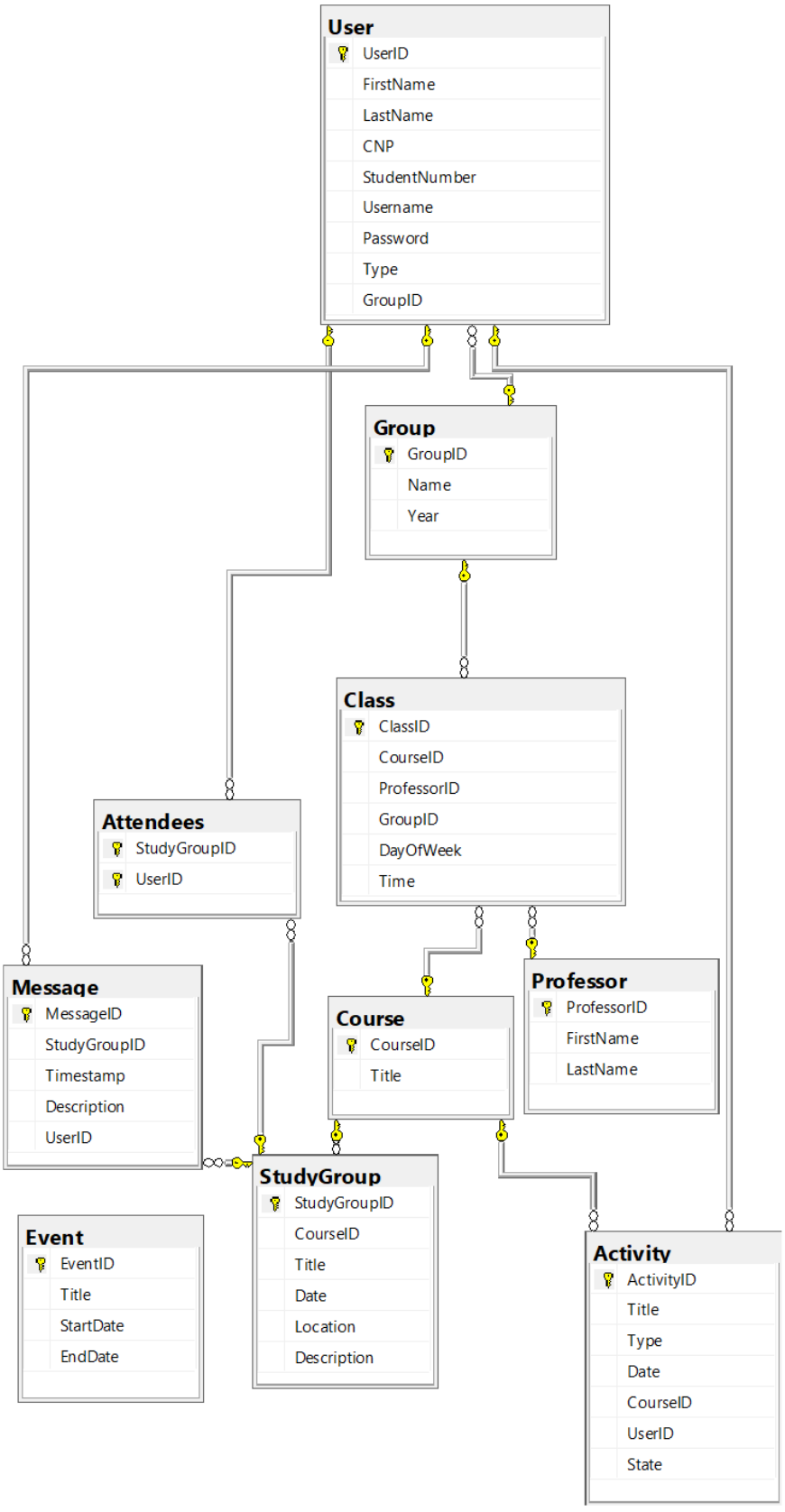

Figura 5.2 Diagrama bazei de date

5.3. Nivelul logic

5.3.1. Models

După cum am mai precizat, acest nivel este unul auxiliar. El este referențiat de către toate celelalte nivele ale aplicației, pentru a asigura consecvență între ele și pentru a păstra aplicarea principiului de single responsability. Aces lucru înseamnă că fiecare clasă se va ocupa de descrierea proprietăților unui singur tip de obiect.

Figura 5.3 Diagrama de clase a nivelului Models

Așa cum se poate observa în diagrama de clase de mai sus, fiecare dintre clase are drept corespondent un tabel din baza de date. Prin urmare, în cadrul fiecărei clase, vor exista proprietăți publice ce corespund coloanelor tabelului ce îl reflectă. Cum este de așteptat, nici una dintre clase nu depinde de o alta, fiecare având scopul de a ține proprietățile propriului obiect corespondent.

5.3.2. Data Access Layer

Nivelul de acces la date este folosit pentru a păstra codul pe care îl utilizez pentru a extrage datele, mai precis, în cazul de față, pentru a aduce datele de la serverul SQL la back-end. Acesta este bine separat de logica de business și interfața web. Astfel, acest nivel urmează ideea de "separation of concerns", prin care toate logica necesară pentru interacțiunea logicii de business cu nivelul de date, adică baza de date este izolată într-un singur set de clase (strat). Clasele ce sunt conținute în cadrul acestui nivel pot fi vizualizate în diagrama de clase din figura 5.5.

Clasele mai sus menționate și reprezentate în figură sunt împărțite în așa fel încât o clasă să se ocupe de manipularea datelor unui singur tabel al bazei de date. Acest lucru ușurează înțelegerea și mentenanța codului prin neamestecarea conceptelor într-un singur obiect.

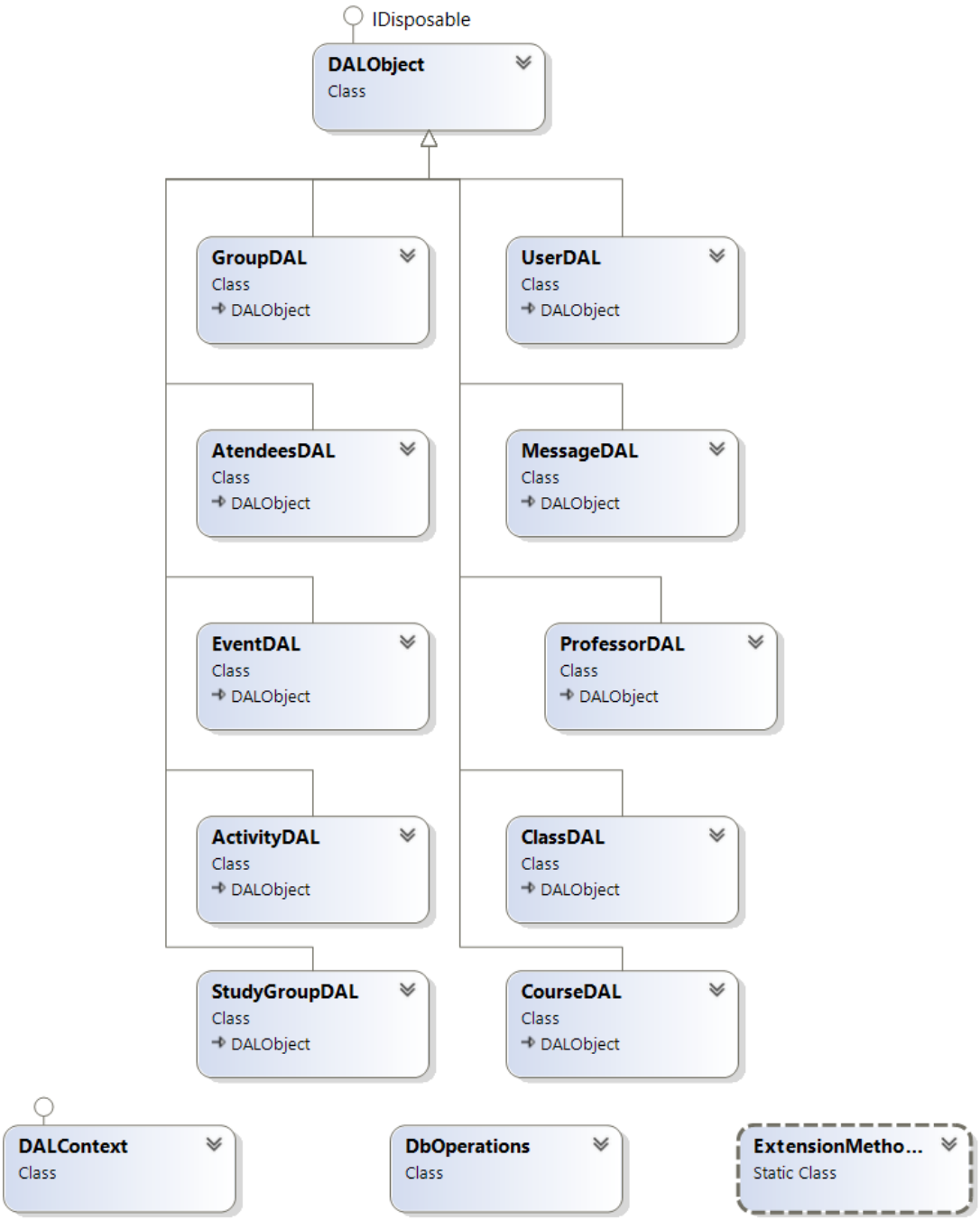

Figura 5.4 Diagrama de clase a nivelului de acces la date

După cum se observă și în diagramă, clasele de acces la date aferente fiecărui tabel sunt, de fapt, subclase ale DALObject. DALObject la rândul ei implementează interfața IDisposable. Aceasta conține constructorul și implementarea metodei de Dispose.

Interfața IDisposable are care utilizare primară eliberarea resurselor dacă acestea nu sunt folosite. Chir daca eliberarea automată a memoriei alocate unui obiect atunci când acesta nu mai este utilizat este de datoria garbage collector-ului, nu se poate știi cu

exactitate când va avea lor următoare acțiune de colectare a acestuia. Prin utilizarea metodei Dispose a acestei interfețe se vor elibera în mod explicit resursele ce nu mai sunt folosite. Metoda este apelată din clasă atunci când obiectul nu mai este necesar.

```
public void Dispose()
{
    Dispose(true);
    GC.SuppressFinalize(this);
}
```

Figura 5.5 Implementarea metodei Dispose

Figura anterioară ilustrează implementarea prin suprascriere a metodei Dispose în cadrul clasei DALObject, din interfața IDisposable. Cea de-a doua linie de cod, reprezintă solicitarea către CLR (Common Language Runtime) de a nu apela finalizatorul pentru obiectul specificat, în acest caz DALObject sau toate obiectele ce au ca tip subclase ale acestuia.

Fiecare dintre clasele ce moștenesc DALObject implementează pentru tabelul aferent o metodă ce aduce toate obiectele din tabel (ReadAll), o metodă ce aduce un singur obiect din tabel pe baza identificatorului său unic (ReadByID), o metodă de inserare a unui obiect în tabel (Insert), o metodă de de actualizare a datelor unui obiect din tabel (Update) și una de ștergere a unui obiect pe baza idenificatorului său unic (Delete). Mai specific, metoda ReadAll nu va primi nici un parametru, metoda ReadByID va primi un parametru de tip Guid (globally unique identifier sau identificator unic global), metodele Update și Insert vor primi câte un obiect de tipul aferent clasei în care sunt implementate, iar metoda Delete va primi de asememenea un parametru de tip Guid.

Metodele ReadAll și ReadByID vor executa câte un query, pe când Insert, Update și Delete un command. Aceste două tipuri de operații vor fi executate prin apelul metodelor ExecuteQuery și ExecuteCommand, implementate în clasa DbOperations.

Metoda ExecuteCommand din clasa DbOperations va crea mai întâi o conexiune către baza de date, după care va instanția un obiect de tipul SqlCommand pentru procedura stocată ce trebuie apelată. Tipul comenzii va fi setat ca StoredProcedure, conexiunea va fi setată cu cea creată anterior, iar parametrii vor fi cei trimiși către metoda ExecuteCommand. Metoda va deschide mai apoi conexiunea către baza de date și va executa comanda ca non-query. După ce toate acestea sunt finalizate, memoria ocupată de către aceste obiecte va fi eliberată până la un eventual apel ulterior. Metoda ExecuteQuery este asemănătoare cu cea anterioară în ceea ce privește pașii efectuați. Și în acest caz se va instanția un obiect de tipul SqlCommand pentru procedura stocată ce trebuie apelată, tipul comenzii va fi setat ca StoredProcedure, conexiunea va fi setată cu cea creată anterior, iar parametrii vor fi cei trimiși către metoda ExecuteQuery. După deschiderea conexiunii către baza de date, aici se va folosi un obiect de tip SqlDataReader în care se aduc rezultatele obținute în urma apelării metodei ExecuteReader pentru comanda creată inițial. Mai apoi, cât timp în obiectul SqlDataReader mai sunt înregistrări, acestea sunt adăugare într-o listă de obiecte ce va

fi returnată ca rezultat la final. Metoda fiind generică, tipul obiectelor din listă depinde de tipul pentru care a fost apelată. Acest tip poate fi oricare dintre cele existente în nivelul auxiliar de modele (Models). Implementarea celor două metode poate fi văzută în detaliu în figurile 5.6 și 5.7.

Pentru a obține și trimite parametrii către cele două metode, se folosește metoda GenerateSqlParametersFromModel implementată în clasa ExtensionMethods. Aceasta folosește tehnica de reflexion pentru a genera pe baza oricărei clase din Models tipul parametrilor necesari.

În metoda ExecuteQuery, se folosește tot din clasa ExtensionMethods metoda GetObjectFromReader, pentru a transforma rezultatul adus de la baza de date într-un obiect de tipul celui cerut, pentru a putea fi introdus în lista de rezultate și pentru a se lucra mai ușor pe viitor cu datele din listă când este nevoie de acestea în procesare.

Pentru a avea mai ușor acces la oricare dintre clasele din acest nivel și pentru a asigura faptul ca un obiect de acces la date este instanția o singură dată pe parcursul rulării, am implementat clasa DALContext. Din cadrul unui obiect instanția de acest tip, este permis accesul la proprietățile sale publice ce corespund fiecărui obiect de acces la date din cadrul acestui nivel. Pentru a asigura faptul că o instanță a unui astfel de obiect este creată o singură dată, pentru fiecare dintre proprietăți se păstrează un membru corespunzător. Când se accesează proprietatea corespunzătoare obiectului de acces la date, se verifică dacă membrul analog lui a mai fost instanția, iar dacă nu se creează o nouă instanță a sa.

```
public static void ExecuteCommand(string connectionString, string storedProcedureName,
    params SqlParameter[] parameters)
{
    using (SqlConnection connection = new SqlConnection(connectionString))
    {
        using (SqlCommand command = new SqlCommand(storedProcedureName))
        {
            command.CommandType = CommandType.StoredProcedure;
            command.Connection = connection;

            command.Parameters.AddRange(parameters);

            connection.Open();
            command.ExecuteNonQuery();
        }
    }
}
```

Figura 5.6 Implementarea metodei ExecuteCommand

```
public static IEnumerable<T> ExecuteQuery<T>(string connectionString, string storedProcedureName,
    params SqlParameter[] parameters) where T : new()
{

    List<T> result = new List<T>();
    try
    {
        using (SqlConnection connection = new SqlConnection(connectionString))
        {
            using (SqlCommand command = new SqlCommand(storedProcedureName))
            {
                command.CommandType = CommandType.StoredProcedure;
                command.Connection = connection;

                if (parameters.Length != 0)
                    command.Parameters.AddRange(parameters);

                connection.Open();
                using (SqlDataReader reader = command.ExecuteReader())
                {
                    while (reader.Read())
                    {
                        result.Add(reader.GetObjectFromReader<T>());
                    }
                }
            }
        }
    }
    catch (Exception exception)
    {
        throw exception;
    }
    return result;
}
```

Figura 5.7 Implementarea metodei ExecuteQuery

5.3.3. Business Logic Layer

În acest nivel se pregătesc datele pentru a fi trimise către API-ul aplicației. Arhitectura acestui nivel este foarte asemănătoare cu cea a nivelului de acces la date prezentată în subcapitolul anterior. Și aici se respectă principiul de single-responsability, prin urmare pentru fiecare dintre obiectele de acces la date există câte o clasă dedicată ce se va ocupa de procesarea datelor și pregătirea lor fie pentru trimiterea către interfața cu utilizatorul când datele sunt cerute, sau pregătirea lor pentru a fi trimise către nivelul de date când datele sunt trimise.

Clasa BusinessObject este clasa de bază ce va fi moștenită de toate obiectele concrete de business. Aceasta implementează IDisposable tot în scopul de a permite curățarea memoriei ocupate de un astfel de obiect în momentul în care nu mai este folosit fără a aștepta ca acest lucru să fie făcut de Garbage Collector. Concret, această clasă va conține constructorul, implementarea metodei Dispose și destructorul obiectului.

Clasa BusinessContext este clasa ce conține instanțele obiectelor de tipul BusinessObject. Ca în cazul obiectelor de tip acces la date, acest lucru va asigura faptul că o singură instanță a unui obiect de un anumit tip exista la un moment dat în aplicație, iar aceasta va fi creată doar în momentul în care este nevoie de aceasta. Acest lucru

asigură faptul că memoria nu este ocupată de către obiecte ce nu sunt necesare. Și această clasă implementează interfața IDisposable.

Fiecare obiect concret ce extinde BusinessObject va avea implementate metode aferente acțiunilor uzuale, implementate atât în nivelul de acces la date cât și prin proceduri stocate în baza de date. Mai precis, fiecare va avea metodele Insert, ReadByID, ReadAll, Update și Delete.

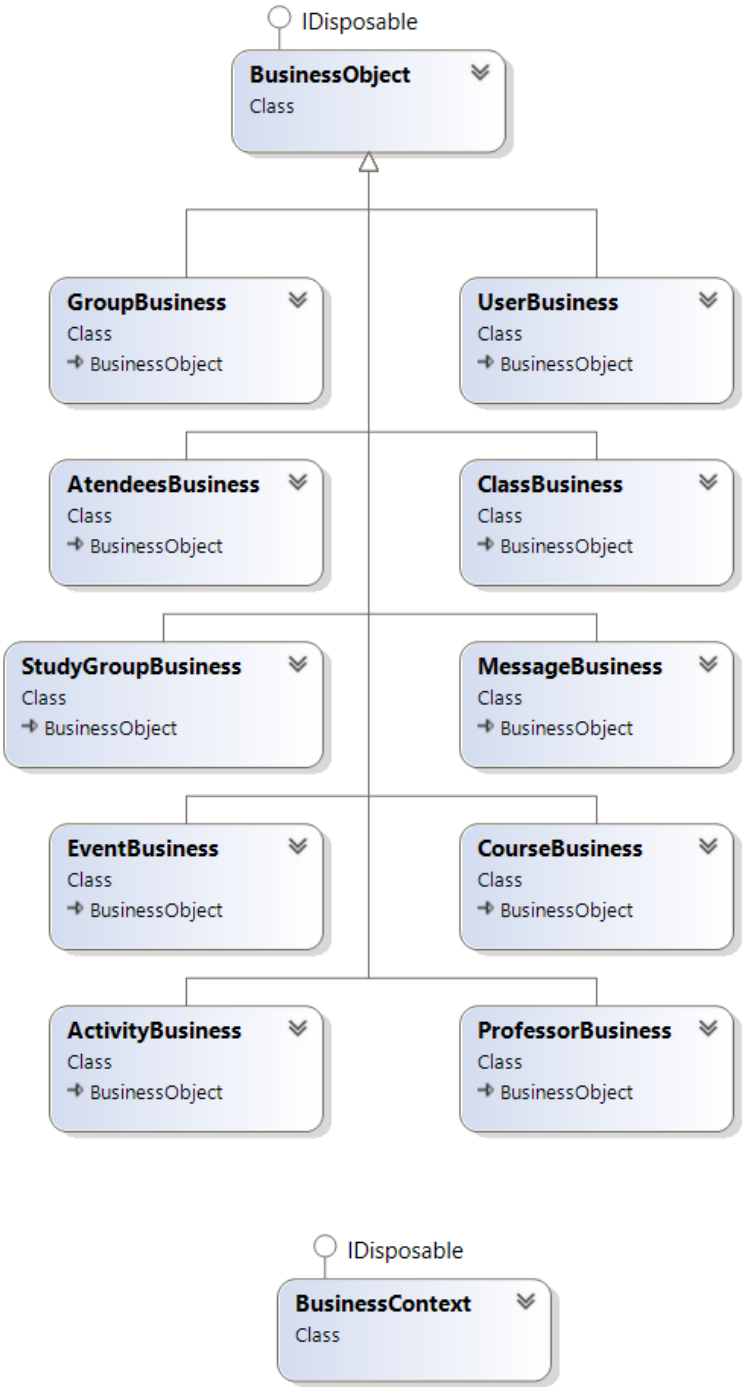

Figura 5.8 Diagrama de clase a nivelului de business

5.3.4. Web API

Acest nivel reprezintă o interfață între nivelul de business și nivelul web, al aplicației dezvoltate. După cum îi sugerează și numele, această interfață poate fi accesată prin web, cu ajutorul protocolului HTML. Pentru a putea fi accesat prin web, acest nivel este hostat cu ajutorul Windows Server IIS și este complet detașat de restul aplicației, el oferind doar acces la funcționalitățile sale expuse prin intermediul său. Astfel, API-ul ar putea fi reutilizat în orice altă aplicație web cu o interfață complet diferită de cea implementată, iar funcționalitățile serverului ar rămâne aceleași fără a face nici o schimbare în nivelele inferioare.

Nivelul de Web API a fost construit folosind framework-ul ASP.NET, special conceput pentru dezvoltarea de astfel de servicii bazate pe comunicarea prin protocolul HTTP.

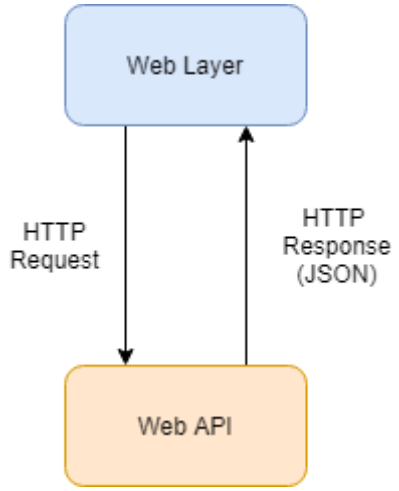

Figura 5.9 Comunicarea cu Web API

În mod concret, după cum se poate observa și în figura precedentă, layerul de prezentare va face un request HTTP către API, iar API va trimite înapoi un răspuns cu informațiile cerute în format JSON. După ce API-ul primește request-ul HTTP, el va apela metoda corespunzătoare din cadrul nivelului de logică de business pentru a primi sau trimite informațiile mai departe.

Așa cum este prezentat și în diagrama de clase din figura 5.10, fiecare obiect din aplicație are un controller specific, iar fiecare dintre aceste controllere moștenesc clasa MainApiController. Aceasta extinde clasa ApiController din framework-ul ASP.NET și conține ca proprietate un obiect de tipul BusinessContext. Prin intermediul acestuia metodele ce se ocupa de request-urile HTTP vor putea accesa metodele implementate in clasele de logică de business, datorită implementării descrise în subcapitolul anterior. De asemenea, este implementată și metoda Dispose, pentru ca memoria folosită de obiectul instanția pentru proprietatea BusinessContext să fie eliminat din memorie în momentul în care nu mai este folosit.

Clasa statică WebApiConfig este reprezentarea fișierului de configurare XML ce conține toate proprietățile setate pentru acesta. Dintre ele cea ce merită menționată este

connectionString, ce conține modalitatea prin care se ajunge la sursa de date, adică la baza de date Planner din SQL Server.

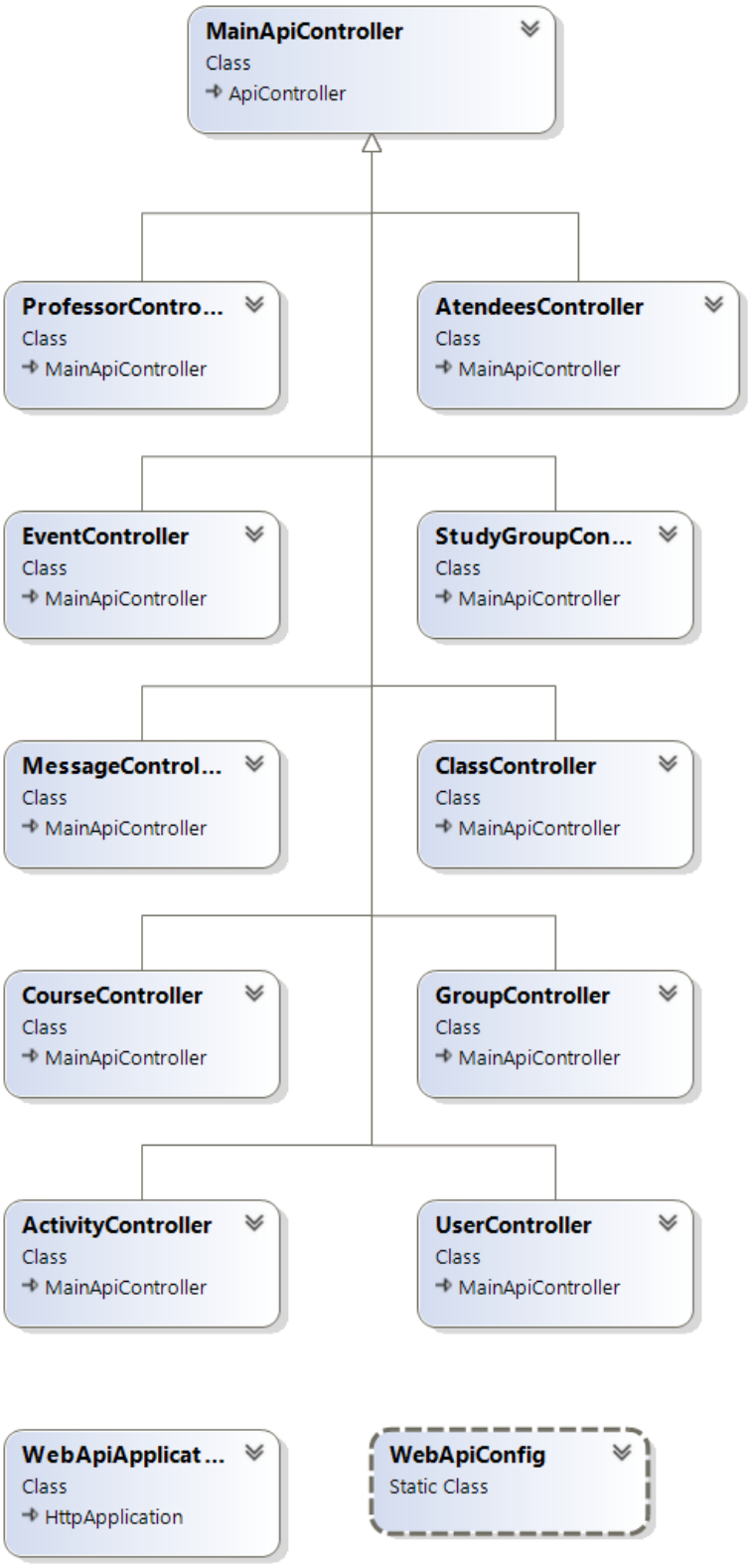

Figura 5.10 Diagrama de clase a nivelului Web API

Fiecare dintre clasele ce moștenesc MainApiController sunt decorate cu atributul RoutePrefix. Acesta descrie ruta prin intermediul căreia se fac requesturile către fiecare dintre controllere. Tabelul de mai jos descrie rutele împreună cu metodele care se apelează din nivelul de business.

Tabel 5.2 Rutele pentru requst-urile făcute către Web API

Ruta	Tip request	Metoda
http://host:port/api/controller	Get	BusinessContext.ControllerBusiness. ReadAll()
http://host:port/api/controller/ {object}	Post	BusinessContext.ControllerBusiness. Insert({object})
http://host:port/api/controller/ {Guid}	Get	BusinessContext.ControllerBusiness. ReadByID({Guid})
http://host:port/api/controller/ {object}	Put	BusinessContext.ControllerBusiness. Update({object})
http://host:port/api/controller/ {Guid}	Delete	BusinessContext.ControllerBusiness. Delete({Guid})

După cum se observă, se pot face patru tipuri de request-uri: GET, POST, PUT și DELETE. Get va aduce informații de la server și le va returna sub formă de obiect JSON sau șir de obiecte JSON. Post va prelua din corpul request-ului un obiect JSON și îl va trimite mai apoi către nivelul de business pentru ca ulterior procesărilor să fie inserat în baza de date. Put de asemenea preia din corpul request-ului un obiect JSON, dar pe acesta îl va trimite mai apoi către nivelul de business pentru ca ulterior procesării să obiectul cu același Guid din baza de date să fie actualizat cu noile informații trimise. Request-ul de Delete este asemănător cu cel de Get, deoarece primește ca parametru un Guid cu scopul ca în cele din urmă obiectul care are acest identificator unic să fie șters din baza de date. Pentru ca fiecare metodă din cadrul controllerelor din API să cunoască tipul de request pe care trebuie să îl deservească, va fi decorată cu unul din descriptorii aferenți tipului. Acești descriptori sunt [HttpGet], [HttpPost], [HttpPut] sau [HttpDelete]. De asemenea, pentru a se cunoaște dacă prin intermediul rutei vor primi și parametrii se mai adaugă descriptorul Route. Aceștia se aplică doar în cazul metodelor ce deservesc citirea după un ID sau ștergerea după ID și sunt de forma [Route("{ID:Guid}")]. În cazul celorlalte tipuri de request-uri se va primi parametrul din corpul său, dacă este necesar.

5.4. Nivelul de prezentare

Nivelul de prezentare este ultimul nivel din ierarhia arhitecturală, reprezentând partea afișată pe client. Acesta reprezintă interfața grafică prin intermediul căreia utilizatorul interacționează cu aplicația.

Am ales să expun funcționalitățile aplicației printr-un single-page application. Unul din avantajele principale ale acestuia este că utilizatorii nu pot accesa pagini pentru

care nu au autorizare, deoarece la încărcarea paginii li se aduce doar ceea ce le este permis să vadă, în funcție de rolul setat pentru ei. Pentru un utilizator de tip Administrator meniul va conține doar butoanele Calendar și Administration, pe când pentru unul de tip Student, vor fi disponibile butoanele Dashboard, Calendar și Study Groups.

Fiind vorba de un single-page application, exista o singură pagină HTML principală numită index. Ca orice pagină HTML, este formată dintr-o structură ierarhică de noduri. Pagina este inițial divizată de două noduri principale: head și body. În head se pot regăsi toate fișierele de tip .css sau .js incluse ce asigură aspectul și funcționalitatea paginii, iar în body se află structura principală de elemente de tip div ce vor fi populate în funcție de tipul de utilizator și datele disponibile pentru acesta.

Pagina este populată dinamic la încărcare cu ajutorul controllerelor și a serviciilor. Prin intermediul serviciilor se aduc informațiile de la server, iar controllerele sunt responsabile de generarea elementelor HTML în funcție de informațiile primite de la server.

5.4.1. Structura fișierului index.html

În cele ce urmează se va prezenta și explica structura fișierului ce stă la baza nivelului de prezentare.

5.4.1.1. Secțiunea head

După cum am menționat anterior, în secțiunea head au fost introduse referințele către fișierele de tip .css și .js.

În secțiunea precedată de comentariul <!--Styles-->, au fost introduse referințele către fișierele responsabile de stilul paginii, adică cele de tipul .css (cascading style sheet). În figura 5.11 se pot observa denumirile fișierelor incluse.

```
<!--Styles-->
<link href="assets/css/bootstrap.css" rel="stylesheet" />
<link href="assets/css/font-style.css" rel="stylesheet" />
<link href="assets/css/main.css" rel="stylesheet" />
<link href="assets/css/mycss.css" rel="stylesheet" />
<link href="assets/css/activity.css" rel="stylesheet" />
<link rel="stylesheet" type="text/css" href="assets/fullcalendar/fullcalendar.css">
```

Figura 5.11 Fișierele .css

Primele trei, boostrap.css, font-style.css și main.css aparțin de template-ul de Bootrsrap pe care l-am utilizat în dezvoltarea acestei părți, pentru a avea posibilitatea să reutilizez componente deja stilizate și pentru a mă putea folosi de capacitatea bootstrap de a diviza pagina în rânduri și coloane în funcție de rezoluția ecranului. Această capabilitate mi-a permis să setez atribute elementelor HTML, astfel încât ele să fie afișate cât mai optim atât la rezoluții mari cât și la rezoluții mai mici. Următoarele două, mycss.css și activity.css, sunt create de mine cu scopul de a suprascrie sau adăuga stilizări noi de care am avut nevoie pentru a aranja elementele în pagină după cum am dorit. Ultimul fișier inclus aparține de script-ul folosit pentru afișarea calendarului în aplicație.

Ulterior, am inclus fișierele externe cu script-uri pe care le-am utilizat pentru a implementa partea de prezentare a aplicației. Acestea pot fi vizualizate în figura de mai jos. Am inclus librăria jQuery împreună cu extensia sa jQuery UI și librăria Bootstrap. În plus am inclus moment.js și fullcalendar.js, responsabile pentru funcționalitățile calendarului din pagina Calendar.

```
<!--External-->
<script src="Scripts/External/jquery-3.2.1.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/moment.js/2.22.2/moment.js"></script>
<script src="Scripts/External/jquery-ui.min.js"></script>
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>
<script src="Scripts/External/d3.min.js"></script>
<script type="text/javascript" src="assets/fullcalendar/fullcalendar.js"></script>
```

Figura 5.12 Fișierele externe .js

Următoarele secțiuni sunt delimitate de comentariile <!--Services-->, <!--Utils-->, <!--Controllers--> și <!--Core-->. Ordinea de includere a acestora este dată de următoarea regulă: fișierele ce depind de alte script-uri trebuie incluse ulterior acestora.

Mai întâi am inclus script-urile responsabile de servicii. Baza acestora o reprezintă ajaxController.js. Aici se află implementată funcția responsabilă de apelurile Ajax către WebAPI. Această funcționalitate va fi detaliată într-un subcapitol ulterior.

```
<!--Services-->
<script src="Scripts/Services/Core/ajaxController.js"></script>
<script src="Scripts/Services/coursesService.js"></script>
<script src="Scripts/Services/activitiesService.js"></script>
<script src="Scripts/Services/groupsService.js"></script>
<script src="Scripts/Services/attendeesService.js"></script>
<script src="Scripts/Services/studyGroupsService.js"></script>
<script src="Scripts/Services/usersService.js"></script>
<script src="Scripts/Services/classesService.js"></script>
<script src="Scripts/Services/proffessorService.js"></script>
<script src="Scripts/Services/serviceContext.js"></script>
```

Figura 5.13 Fișierele incluse în secțiunea Services

Ulterior este inclus script-ul utils.js ce conține funcții folosite în mai multe locuri, cum ar fi cea de generare a unei culori aliatoare sau cea de generare a unui Guid.

```
<!--Utils-->
<script src="Scripts/Utils/utils.js"></script>
```

Figura 5.14 Secțiunea Utils

Secțiunea Controllers precedă includerea fișierelor cu script-urile responsabile cu popularea paginilor și preluarea informațiilor din pagină. Acestea sunt aduse sau trimise din respectiv către Web API prin intermediu serviciilor.

```
<!--Controllers-->
<script src="Scripts/Controllers/dashboardController.js"></script>
<script src="Scripts/Controllers/activitiesController.js"></script>
<script src="Scripts/Controllers/coursesController.js"></script>
<script src="Scripts/Controllers/studyGroupsController.js"></script>
<script src="Scripts/Controllers/groupsController.js"></script>
<script src="Scripts/Controllers/usersController.js"></script>
<script src="Scripts/Controllers/proffessorController.js"></script>
<script src="Scripts/Controllers/classesController.js"></script>
<script src="Scripts/Controllers/tooltip.js"></script>
```

Figura 5.15 Secțiunea Controllers

În cele din urmă sunt incluse în secțiunea Core script-urile responsabile cu inițializarea controllerelor la încărcarea paginii în funcție de rol, application.js și sartup.js.

```
<!--Core-->
<script src="Scripts/application.js"></script>
<script src="Scripts/startup.js"></script>
```

Figura 5.16 Secțiunea Core

5.4.1.2. Secțiunea body

În această secțiune sunt incluse principalele elemente de tip div ce vor simula împărțirea în funcționalități a aplicației web. Deoarece vorbim există un singur fișier HTML, dar nu am dorit prezentarea tuturor funcțiilor aplicației în același timp și afișarea către utilizator doar a funcțiilor disponibile lui am introdus patru elemente div ce corespund celor patru butoane disponibile în meniu. Un al cincilea div este folosit pentru meniu.

În div-ul identificat prin atributul id „dashboard-container” se vor încărca toate elementele ce trebuie afișate pe prima pagină unui utilizator de tip Student.

În cel identificat prin atributul id „calendar-container” se va încărca calendarul ce va fi disponibil cu funcționalitățile specifice atât unui utilizator de tip Student cât și unuia de tip Administrator.

În elementul div identificat prin atributul id „studyGroup-container” vor fi afișate elementele pentru interacțiunea cu grupurile de studiu.

Nu în ultimul rând, în elementul div identificat de atributul id ce ia valoarea „adminPage-container” se vor încărca elementele responsabile pentru afișarea paginii din care un utilizator de tip Administrator va face managementul informațiilor.

Tot în body se află și declararea structurii ierarhice de elemente ce formează un modal. Acesta este folosit în aplicație pentru a afișa diverse informații în această fereastră de tip pop-up fără a părăsi pagina curentă.

Structura principală a elementului body se poate observa în figura următoare.

```
<body>
    <div id="userInfoCard" class="half-unit">...</div>

    <div id="mainContainer" style="display:none" )>
        <div class="navbar-nav navbar-inverse navbar-fixed-top">...</div>

        <div id="calendar-container" class="container">...</div>

        <div id="dashboard-container" class="container">...</div>

        <div id="studyGroups-container" class="container">...</div>

        <div id="adminPage-container" class="container">...</div>
    </div>

    <div id="joinedUsersModal" class="modal fade">...</div>
</body>
```

Figura 5.17 Structura elementului body din index.html

5.4.2. Structura fișierelor cu script-uri

Spre deosebire de alte limbaje de programare JavaScript nu permite structurarea funcțiilor în clase, însă acest comportament se poate simula prin diverse metode pentru o mai bună organizare a codului. Primul pas în structurarea codului a fost să creez fișiere pentru fiecare controller și fiecare serviciu, la care s-au adăugat fișierele responsabile de inițializarea serviciilor și controllerelor la încărcarea paginii. Mai apoi, pentru a simula existența unei clase, în fiecare fișier există o funcție principală atribuită unei variabile ce poartă un nume sugestiv. În interiorul acelei funcții se află declararea de variabile și funcții. Pentru ca o funcție sau o variabilă să fie publică, aceasta va fi pusă pe elementul „this” adică pe prototipul funcției principale. Pentru ca o funcție să fie privată și să nu poată fi accesată din exterior, aceasta se declară ca orice variabilă sau funcție în JavaScript. Mai mult, pentru a respecta principiile ce duc la scrierea unui cod cât mai curat și ușor de citit, am respectat convenția ca funcțiile publice să înceapă cu litere mari, cele private cu litere mici, iar membrii privați cu litere mici.

Un cod generic menit să exemplifice comportamentul descris în paragraful anterior poate fi consultat în figura 5.18.

```
var Example = function () {
    var _member = "member";

    this.PublicFunction = function () {
        return "This is a public JavaScript function."
    }

    function privateFunction() {
        return "This is a private JavaScript function."
    }
}
```

Figura 5.18 Simularea claselor în Javascript

5.4.3. Controller-ul Ajax

Figura de mai sus reprezintă implementarea AjaxController ce conține funcția publică Call, prin intermediul căreia se face un apel asincron către API-ul aplicației. La momentul implementării acesta era hostat pe serverul local, disponibil prin pe portul 8107.

După cum se observă, URL-ul la care se face apelul se compune din URL-ul serverului la care se concatenează calea către controller și numele metodei ce se vrea a fi apelată. Formatul datelor trimise și primite ca răspuns va fi JSON. Atributul crossDomain este setat pe true, acest lucru permițând ca apelul să se facă din domenii diferite, astfel Web API poate fi hostat pe un alt server decât aplicația web. Pentru tratarea eventualelor erori, în cazul în care nu se poate efectua cu succes apelul Ajax se va genera în pagină o alertă cu un mesaj sugestiv ce semnalizează faptul că a avut loc o eroare.

```
var AjaxController = function (controller) {
    var _serverUrl = "http://localhost:8107/api";

    this.Call = function (requestType, methodName, data) {
        return $.ajax({
            url: _serverUrl + controller + "/" + methodName,
            dataType: 'json',
            type: requestType,
            crossDomain: true,
            contentType: 'application/json; charset=utf-8',
            data: JSON.stringify(data),
            error: function () {
                alert("An error occured.");
            }
        });
    };
}
```

Figura 5.19 Implementarea controller-ului Ajax

5.4.4. Serviciile

În aplicația web sunt implementate serviciile corespunzătoare pentru fiecare dintre obiectele a căror informații pot fi păstrate în baza de date. Acestea conțin funcții responsabile de a efectua apeluri asincrone către Web API hostat pe server în scopul manipulării prin interfața grafică a bazei de date. Concret, făcând o paralelă între baza de date și serviciul implementat în aplicația web, un serviciu va deservi un singur tabel și va exista câte o funcție corespunzătoare fiecăreia din procedurile stocate existente pentru acest tabel.

În figura 5.20, se poate vedea un exemplu de implementare pentru un astfel de serviciu. Acesta corespunde tabelului Classes și conține funcțiile publice ReadAll, ReadByID, Insert, Update și Delete. Fiecare va returna răspunsul apelului Ajax corespunzător. ReadAll și Read vor efectua apeluri prin metoda HTTP GET, Insert prin POST, Insert prin PUT iar Delete prin DELETE. Numele metodei HTTP ce trebuie folosită va fi trimisă ca parametru funcției Call din controller-ul Ajax descris în

subcapitolul precedent, împreună cu calea către metodă și parametrii, dacă este cazul. În cazul fiecărui serviciu, controller-ul Ajax va fi inițializat cu URL-ul specific.

```
var ActivitiesService = function () {
    var _activityControllerUrl = "/activities";
    var _ajaxController = new AjaxController(_activityControllerUrl);

    this.ReadAll = function () {
        return _ajaxController.Call("GET", "");
    }

    this.ReadByID = function (id) {
        return _ajaxController.Call("GET", id);
    }

    this.Insert = function (activity) {
        return _ajaxController.Call("POST", "", activity);
    }

    this.Update = function (activity) {
        return _ajaxController.Call("PUT", "", activity);
    }

    this.Delete = function (activityID) {
        return _ajaxController.Call("DELETE", activityID);
    }
}
```

Figura 5.20 Implementarea serviciului pentru Activities

5.4.5. Controllere

Controllerul implementat în fișierul startup.js are ca scop stabilirea la încărcarea paginii dacă există deja un utilizator autentificat sau nu. Daca nu există, se va face redirectarea către pagina de login. Dacă există, va fi apelată funcția Start din controllerul Application.

La apelul funcției Start, se va crea controllerul de servicii și controllerele pentru funcțiile aplicației. Controllerele pentru funcțiile aplicației vor fi mai apoi inițializate și li se va trimite obiectul creat pentru controllerul de servicii.

Am implementat câte un controller responsabil pentru popularea fiecărui element de tip div menționat în subcapitolul 5.4.1.2 pentru a respecta principiul de single-responsability. Fiecare dintre acestea conțin funcția publică Initialize ce se apelează din metoda Start a controllerului Application. În această metodă se vor face apelurile către toate funcțiile private responsabile cu aducerea datelor de la servicii și generarea de elemente și adăugarea lor în secțiunile potrivite din pagină. De asemenea, vor fi apelate și funcțiile responsabile de înregistrarea de evenimente ce trebuie tratate când utilizatorul face diverse acțiuni în pagină. Un exemplu în acest sens ar fi apelul unei metode ce va înregistra pe un anumit buton dintr-un formular evenimentul de „click” împreună cu funcția care se apelează pentru a prelua datele din formular și a le trimite mai departe către server prin intermediul serviciului potrivit.

5.4.6. Păstrarea sesiunii

Pentru a păstra sesiunea utilizatorului logat, se folosește proprietatea sessionStorage de pe obiectul window al browser-ului. Sesiunea va fi păstrată atâta timp cât browserul este deschis. Dacă tab-ul în care s-a deschis sesiunea este închis, sesiunea va fi păstrată în cazul în care se face restaurarea tab-ului (dacă browserul are această funcționalitate). În cadrul sesiunii se păstrează numele de utilizator al celui logat și rolul acestuia pentru a facilita anumite operațiuni in interfața web.

5.5. Deployment-ul aplicației

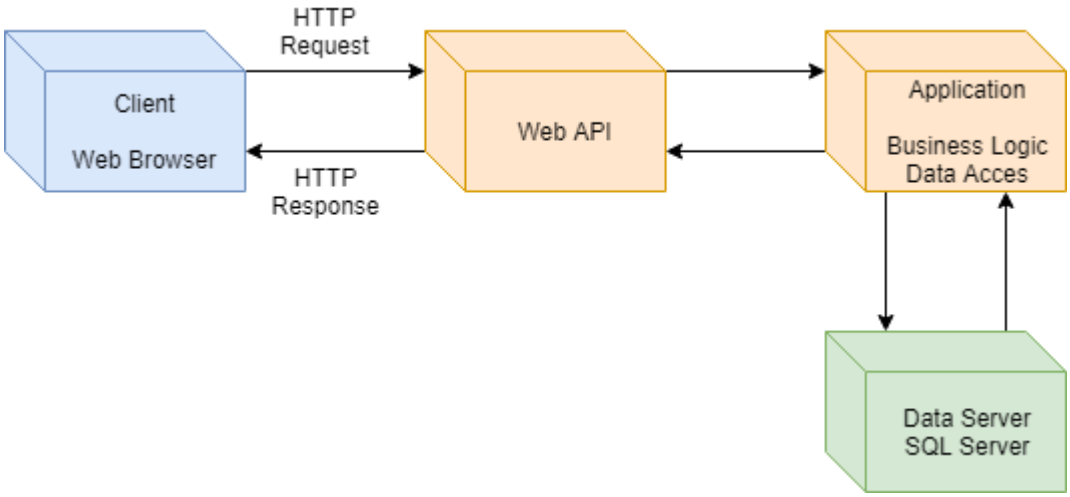

Figura 5.21 Diagrama de deployment a aplicației

În figura anterioară, se poate observa diagrama de deployment a aplicației dezvoltate. De la client se va face un request prin protocolul HTTP către Web API-ul hostat prin intermediu IIS. Web API-ul va apela metoda corespunzătoare request-ului din nivelul de business al aplicației. Metoda din nivelul de business se va folosi de obiectul de acces la date, iar din obiectul de acces la date se va face un request către serverul de SQL pe care este persistată baza de date a aplicației. În acest request se va specifica care dintre procedurile stocate trebuie apelată. După execuția procedurii stocate, serverul SQL va trimite răspunsul înapoi la aplicație. Nivelul de business va trimite obiectul ce conține informațiile cerute înapoi către Web API, iar Web API va trimite răspunsul înapoi către client pentru a fi afișat.

Acest deployment permite sistemului să fie scalabil, datorită faptului că o parte din module, sau chiar a fiecărui modul individual poate fi hostat pe servere diferite. Prin folosirea serverelor multiple, s-ar obține o creștere a capacității de procesare și/sau stocare și implicit posibilitatea procesării a unui număr mai mare de requesturi. Un exemplu de asfel de configurare poate fi:

- Server 1: serverul de date ce va hosta SQL Server
- Server 2: serverul de business ce va hosta nivelul Application
- Server 3: serverul web API unde vor ajunge în primă fază cererile de la client și de unde se vor trimite răspunsurile
- Server4: serverul ce va hosta interfața grafică a aplicației cu care utilizatorul final interacționează

Capitolul 6. Testare și Validare

În cele ce urmează vor fi prezentate principalele procese prin care s-a făcut testarea și validarea aplicației atât pe parcursul testării cât și la finalizarea implementării. Testarea este crucială în procesul de dezvoltare de aplicații software, deoarece determină în mod direct calitatea produsului final. Prin aplicarea a diverse metodologii de testare se asigură faptul că utilizatorul final are parte de o experiență plăcută de folosire a sistemului.

6.1. Testarea timpurie (aplicarea Early Testing Principle)

Pe tot parcursul implementării s-a realizat testarea manuală a fiecărei componente, atât pe parte de *front-end* cât și de *back-end*. Acest lucru a asigurat faptul că orice eroare de implementare este găsită înainte să se treacă la implementarea unei noi părți.

Avantajul acestei abordări l-a constituit faptul că s-au minimizat rescrierile ulterioare ale codului. Astfel nu a fost necesar ca după o perioadă lungă de timp să se revină la funcționalități deja implementate pentru a fi regândite din cauza unor erori găsite pe parcurs.

6.2. Testarea răspunsului de la server

Pentru a valida faptul că partea de Web API este funcțională, am folosit utilitarul Postman. Prin intermediul acestuia, am putut să fac apeluri către Web API-ul aplicației după ce acesta a fost hostat pe server. Apelurile făcute prin Postman mi-au permis atât să validez că toate apelurile prin intermediu protocolului HTTP se fac corect și fără erori, cât și să evaluez timpul între request și răspuns.

La începutul lucrării s-a stabilit ca fiind acceptat obținerea unui timp de răspuns sub 200 ms pentru toate cererile către server. În tabelele de mai jos se pot vizualiza rezultatele obținute pentru fiecare controller.

Pentru scenariile specifice utilizatorului de tip student, măsurătorile au fost făcute folosind ca mediu de testare browserul Google Chrome, pe care s-a realizat autentificarea în aplicație. Utilizatorul avea deja definite mai multe activități. Pentru scenariile specifice utilizatorului de tip administrator, măsurătorile au fost făcute folosind ca mediu de testare browserul Google Chrome, pe care s-a realizat autentificarea în aplicație. În aplicație erau deja definite mai multe obiecte de tipul student, profesor și curs.

Tabel 6.1 Timpii de răspuns ai metodelor Web din controllerul clasei Activity

ActivityController http://localhost:8107/api/activities			
Tip metoda HTTP	Metoda apelată	Status	Timp răspuns
GET	ReadAll()	200 OK	121 ms
GET	ReadByID(id)	200 OK	88 ms
POST	Insert(activity)	204 No Content	86 ms
PUT	Update(activity)	204 No Content	32 ms
DELETE	Delete(id)	204 No Content	51 ms

Tabel 6.2 Timpii de răspuns ai metodelor Web din controllerul clasei Atendees

AtendeesController http://localhost:8107/api/atendees			
Tip metoda HTTP	Metoda apelată	Status	Timp răspuns
GET	ReadAll()	200 OK	98 ms
GET	ReadByID(id)	200 OK	81 ms
POST	Insert(activity)	204 No Content	83 ms
PUT	Update(activity)	204 No Content	29 ms
DELETE	Delete(id)	204 No Content	57 ms

Tabel 6.3 Timpii de răspuns ai metodelor Web din controllerul clasei Class

ClassController http://localhost:8107/api/classes			
Tip metoda HTTP	Metoda apelată	Status	Timp răspuns
GET	ReadAll()	200 OK	104 ms
GET	ReadByID(id)	200 OK	95 ms
POST	Insert(activity)	204 No Content	82 ms
PUT	Update(activity)	204 No Content	25 ms
DELETE	Delete(id)	204 No Content	55 ms

Tabel 6.4 Timpii de răspuns ai metodelor Web din controllerul clasei Course

CourseController http://localhost:8107/api/courses			
Tip metoda HTTP	Metoda apelată	Status	Timp răspuns
GET	ReadAll()	200 OK	108 ms
GET	ReadByID(id)	200 OK	97 ms
POST	Insert(activity)	204 No Content	87 ms
PUT	Update(activity)	204 No Content	24 ms
DELETE	Delete(id)	204 No Content	60 ms

Tabel 6.5 Timpii de răspuns ai metodelor Web din controllerul clasei Event

EventController http://localhost:8107/api/eventobjes			
Tip metoda HTTP	Metoda apelată	Status	Timp răspuns
GET	ReadAll()	200 OK	101 ms
GET	ReadByID(id)	200 OK	100 ms
POST	Insert(activity)	204 No Content	84 ms
PUT	Update(activity)	204 No Content	23 ms
DELETE	Delete(id)	204 No Content	59 ms

Tabel 6.6 Timpii de răspuns ai metodelor Web din controllerul clasei Group

GroupController http://localhost:8107/api/groups			
Tip metoda HTTP	Metoda apelată	Status	Timp răspuns
GET	ReadAll()	200 OK	104 ms
GET	ReadByID(id)	200 OK	95 ms
POST	Insert(activity)	204 No Content	84 ms
PUT	Update(activity)	204 No Content	26 ms
DELETE	Delete(id)	204 No Content	58 ms

Tabel 6.7 Timpii de răspuns ai metodelor Web din controllerul clasei Message

MessageController http://localhost:8107/api/messages			
Tip metoda HTTP	Metoda apelată	Status	Timp răspuns
GET	ReadAll()	200 OK	100 ms
GET	ReadByID(id)	200 OK	91 ms
POST	Insert(activity)	204 No Content	81 ms
PUT	Update(activity)	204 No Content	23 ms
DELETE	Delete(id)	204 No Content	52 ms

Tabel 6.8 Timpii de răspuns ai metodelor Web din controllerul clasei Professor

ProfessorController http://localhost:8107/api/professors			
Tip metoda HTTP	Metoda apelată	Status	Timp răspuns
GET	ReadAll()	200 OK	99 ms
GET	ReadByID(id)	200 OK	87 ms
POST	Insert(activity)	204 No Content	81 ms
PUT	Update(activity)	204 No Content	21 ms
DELETE	Delete(id)	204 No Content	53 ms

Tabel 6.9 Timpii de răspuns ai metodelor Web din controllerul clasei StudyGroup

StudyGroupController http://localhost:8107/api/studyGroups			
Tip metoda HTTP	Metoda apelată	Status	Timp răspuns
GET	ReadAll()	200 OK	103 ms
GET	ReadByID(id)	200 OK	99 ms
POST	Insert(activity)	204 No Content	89 ms
PUT	Update(activity)	204 No Content	30 ms
DELETE	Delete(id)	204 No Content	57 ms

Tabel 6.10 Timpii de răspuns ai metodelor Web din controllerul clasei Users

UserController http://localhost:8107/api/users			
Tip metoda HTTP	Metoda apelată	Status	Timp răspuns
GET	ReadAll()	200 OK	106 ms
GET	ReadByID(id)	200 OK	99 ms
POST	Insert(activity)	204 No Content	87 ms
PUT	Update(activity)	204 No Content	27 ms
DELETE	Delete(id)	204 No Content	58 ms

6.3. Testarea după implementare – Blackbox

Folosind tehnica de testare Blackbox am testat manual toate funcționalitățile disponibile la finalul implementării prin intermediul interfeței grafice a aplicației web. Pentru a aplica această tehnică a trebuit să fac abstracție de cunoașterea modului de implementare, aceasta fiind și principalul concept ce stă la baza tehnicii. Pentru a face acest lucru au fost folosite cazurile de testare ce au dat descrierea funcțională a aplicației.

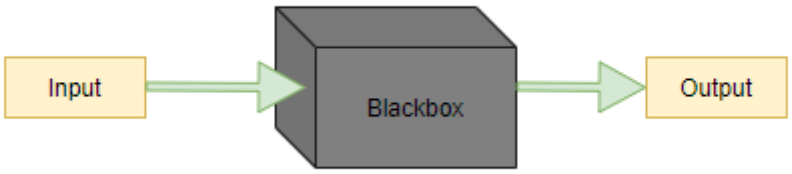

Figura 6.1 Principiul de funționare al metodei de testare Blackbox

Urmărind cazurile de utilizare descrise în capitolul 4, am parcurs pașii descriși în pentru fiecare dintre ele pentru a mă asigura că rezultatele obținute sunt în concordanță cu starea finală în care cazul respectiv de utilizare trebuie să ajungă. De asemenea, m-am asigurat că în cazurile invalide (scenariile alternative) de utilizare se va ajunge la un mesaj de eroare. Acest lucru a asigurat faptul că pe parcursul testării timpurii nu au fost scăpate din vedere anumite aspecte o data cu implementarea de noi funcționalități nu au fost introduse erori noi.

Capitolul 7. Manual de Instalare si Utilizare

Aplicația web este menită să funcționeze pe orice sistem, cu condiția ca partea de server și partea de client să fie setate corespunzător. Pentru a se putea urmări pașii de instalare prezentați mai jos trebuie pe servere trebuie să existe următoarele:

- Microsoft .NET Framework 4.6.1 sau mai nou
- SQL Server 14.0 sau mai nou
- IIS Manager 8 sau mai nou

  În ceea ce privește clientul, pentru ca aplicația să poată fi accesată este necesar doar un browser ce suportă JavaScript și convenția ECMAScript 5 sau mai nou .

7.1. Hostarea Web API

1. Se deschide SQL Server
2. Se face conectare la server fie folosind un nume de utilizator și o parolă
3. Se creează baza de date executând scrip-ul SQL oferit împreună cu aplicația. Acesta va crea o bază de date numită Planner, toate tabelele și toate procedurile stocate necesare bunei funcționări.
4. În fișierul Web.config din directorul ce conține Web API-ul se va completa pe linia connectionString după cum urmează:
   a. Data Source va fi numele serverului SQL pe care se află baza de date
   b. User Id va fi numele de utilizator cu care s-a făcut conectarea la SQL Server iar Password parola corespunzătoare acelui utilizator
5. Se deschide IIS Manager
6. Din secțiunea Connections se face click dreapta pe Sites și se alege opțiunea Add Website

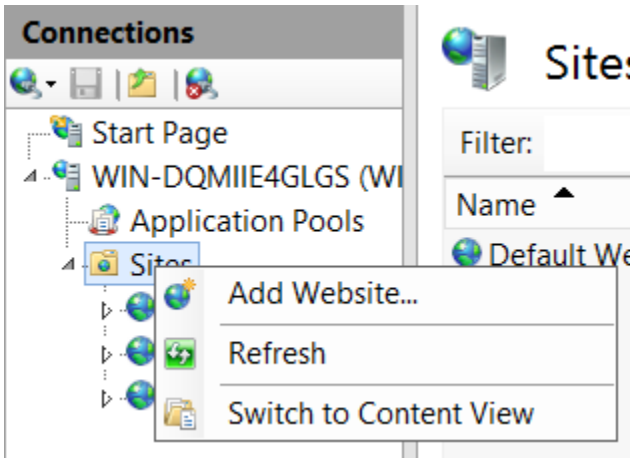

Figura 7.1 Meniul contextual pentru Sites

7. În fereastra apărută se vor completa câmpurile cerute. În secțiunea Site name se completează cu numele dorit. Acesta va fi numele sub care site-ul creat va apărea in IIS Manager. În secțiunea Content Directory se va alege calea absolută către directorul ce conține configurațiile Web API-ului și fișierele .dll specifice nivelului de business logic și nivelului de modele, pe care acesta le referețiază. În secțiunea Binding se va alege tipul protocolului ca http, adresa

IP unde se dorește hostarea Web API-ului și portul deschis în acest scop. Un exemplu de configurare poate fi consultat în figura 7.2.

8. După apăsarea butonului OK, site-ul va fi creeat, iar drept consecință Web API-ul va fi hostat la adresa specificată.

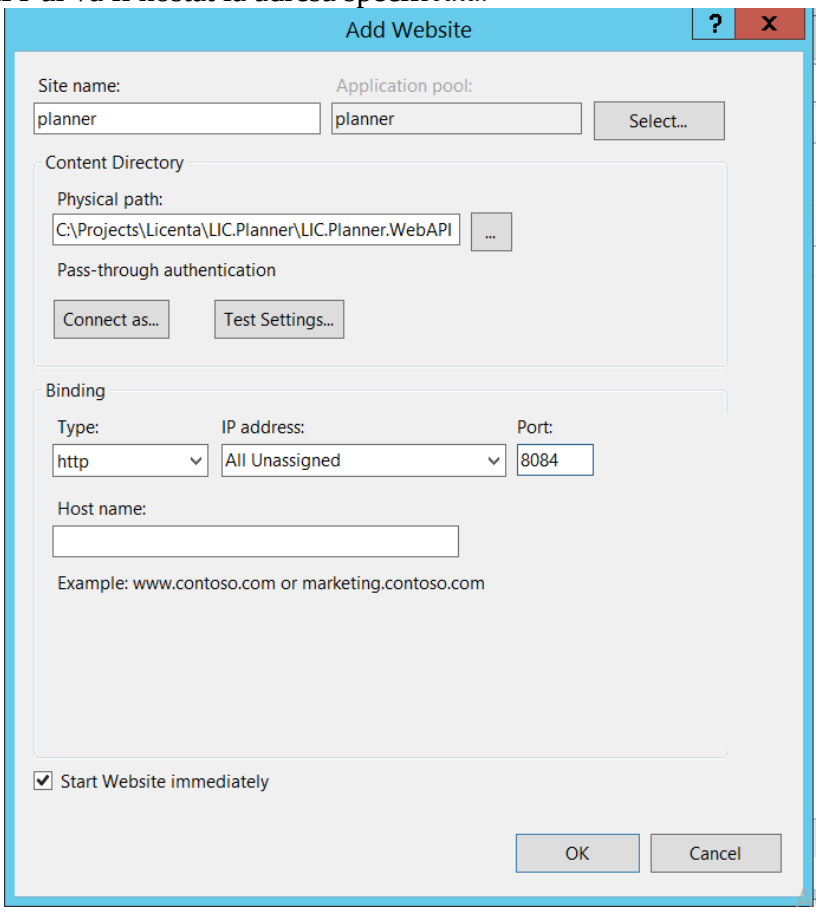

Figura 7.2 Crearea unui nou site in IIS

La finalizarea pașilor de mai sus, serverul va hosta la adresa aleasă Web API-ul prin care clientul va comunica cu serverul.

7.2. Setarea aplicației web

1. Se va deschide IIS pe serverul pe care se va face hostarea
2. Din secțiunea Connections se face click dreapta pe Sites și se alege opțiunea Add Website
3. În fereastra apărută se vor completa câmpurile cerute. În secțiunea Site name se completează cu numele dorit. Acesta va fi numele sub care site-ul creat va apărea in IIS Manager. În secțiunea Content Directory se va alege calea absolută către directorul ce conține configurațiile aplicației și toate fișierele acesteia. În secțiunea Binding se va alege tipul protocolului ca http, adresa IP unde se dorește hostarea aplicației și portul 8086 ce va fi deschis în acest scop.
4. În fișierul ajaxController.js regăsit în /Scripts/Services/Core, se va înlocui variabila _serverUrl cu URL-ul la care s-a hostat Web API-ul.

5. După apăsarea butonului OK site-ul va fi creat. Aplicația poate fi acum accesată de pe orice client folosind adresa la care aceasta a fost hostată urmând pașii anteriori.

7.3. Utilizare

7.3.1. Autentificare

Aplicația se va accesa dintr-un browser web folosind adresa la care aceasta a fost. Pentru a obține acces în aplicație, în formularul de autentificare ce apare în fereastra browserului se va introduce o combinație nume de utilizator și parolă validă.

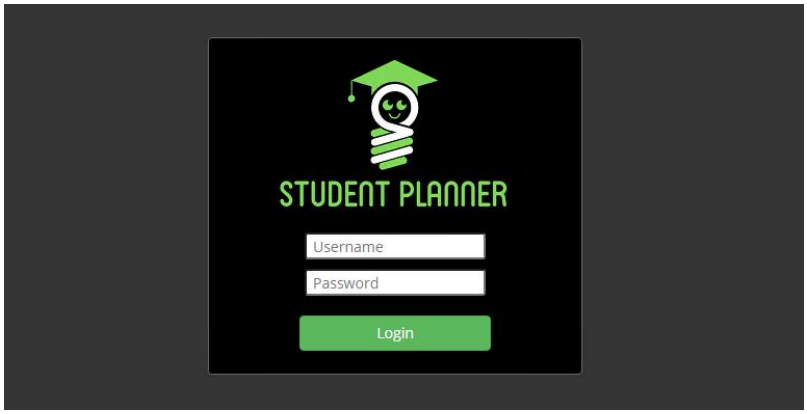

Figura 7.3 Formularul de autentificare

În cazul în care rolul setat pentru utilizator este Student, va apărea pagina din figura de mai jos. În prim plan se va afla Dashboard-ul unde se pot vizualiza toate activitățile din ziua curentă.

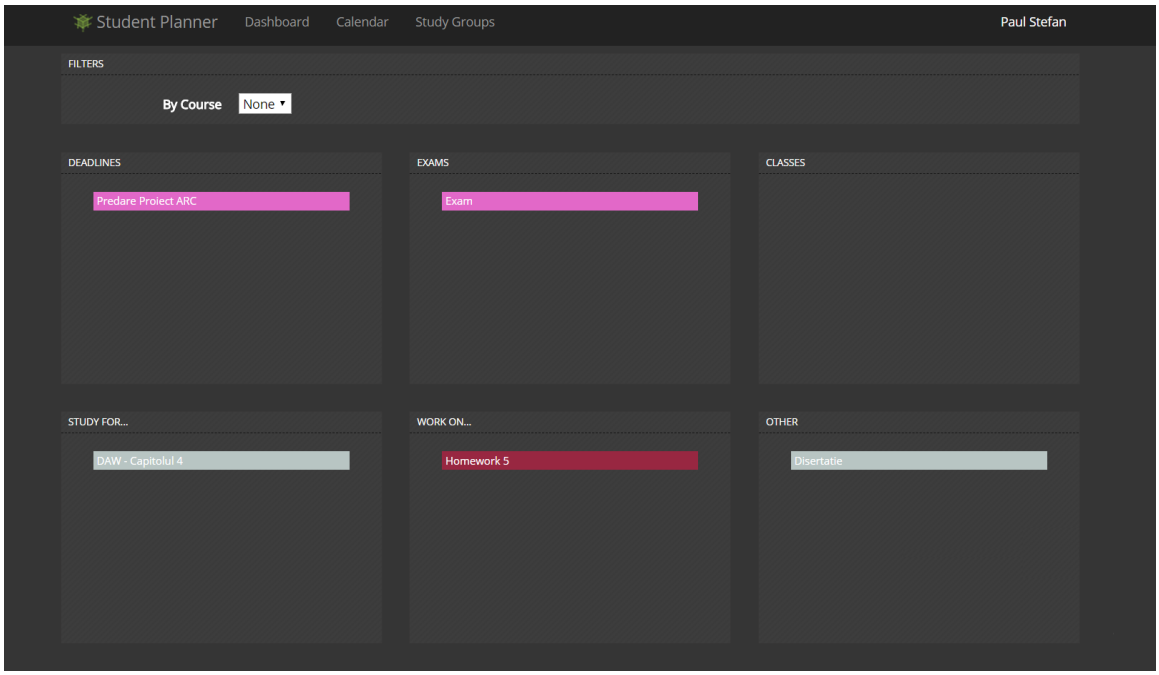

Figura 7.4 Dashboard-ul studentului

Dacă rolul utilizatorului este de administrator, atunci va apărea pagina din figura 7.5 și anume Administration.

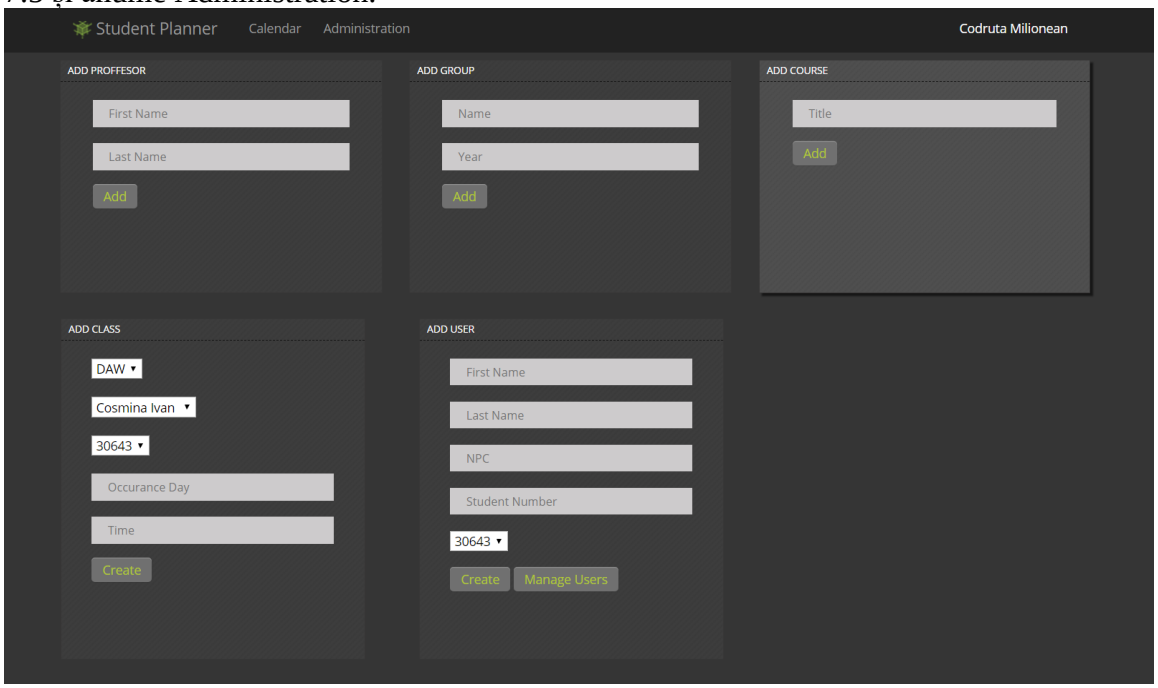

Figura 7.5 Pagina de administrare

7.3.2. Utilizarea formularelor din pagina Administration

În calitate de administrator accesul la pagina disponibilă sub butonul Administration va fi posibilă. Aici se va putea introduce un nou profesor în aplicație, o nouă grupă, un nou curs, sau o nouă clasă atribuită acelui curs. De asemenea se vor putea introduce utilizatori noi de tip student în aplicație.

Toate acestea se vor putea face prin completarea câmpurilor cerute si apăsarea butonului Add de la finalul formularului. Formularele disponibile administratorului pot fi vizualizate în figura 7.5.

7.3.3. Utilizarea Calendarului

Calendarul apare atât pentru administratori cât și pentru studenți dar se populează cu date specifice fiecărui rol în parte.

În calendar apar activitățile deja definite. Pentru a fi mutată o activitate într-o altă zi, este suficient ca eticheta corespunzătoare ei să fie mutată în ziua dorită din calendar printr-o acțiune „drag-and-drop”.

Pentru a adăuga o nouă activitate în calendar, o etichetă pentru aceasta trebuie creată mai întâi creată cu ajutorul formularului Add Activities. Acesta se completează cu informațiile cerute, iar la apăsarea butonului Add va apărea sub el noua etichetă. Pentru a seta activitatea pentru o anumită zi, este suficientă punerea ei în calendar prin „drag and drop” în data dorită.

Interfața paginii Calendar poate fi consultată în figura 7.6.

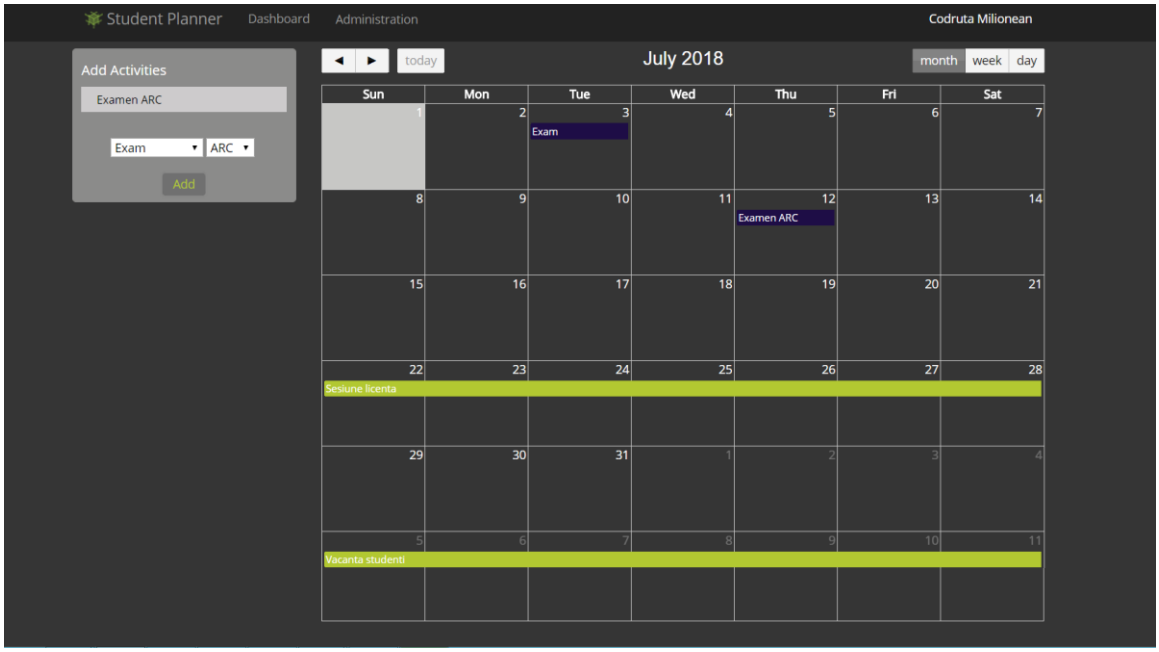

Figura 7.6 Pagina Calendar

7.3.4. Utilizarea paginii Study Groups

Această pagină este disponibilă doar dacă rolul utilizatorului este setat ca student. Interfața acesteia poate fi vizualizată în figura de mai jos.

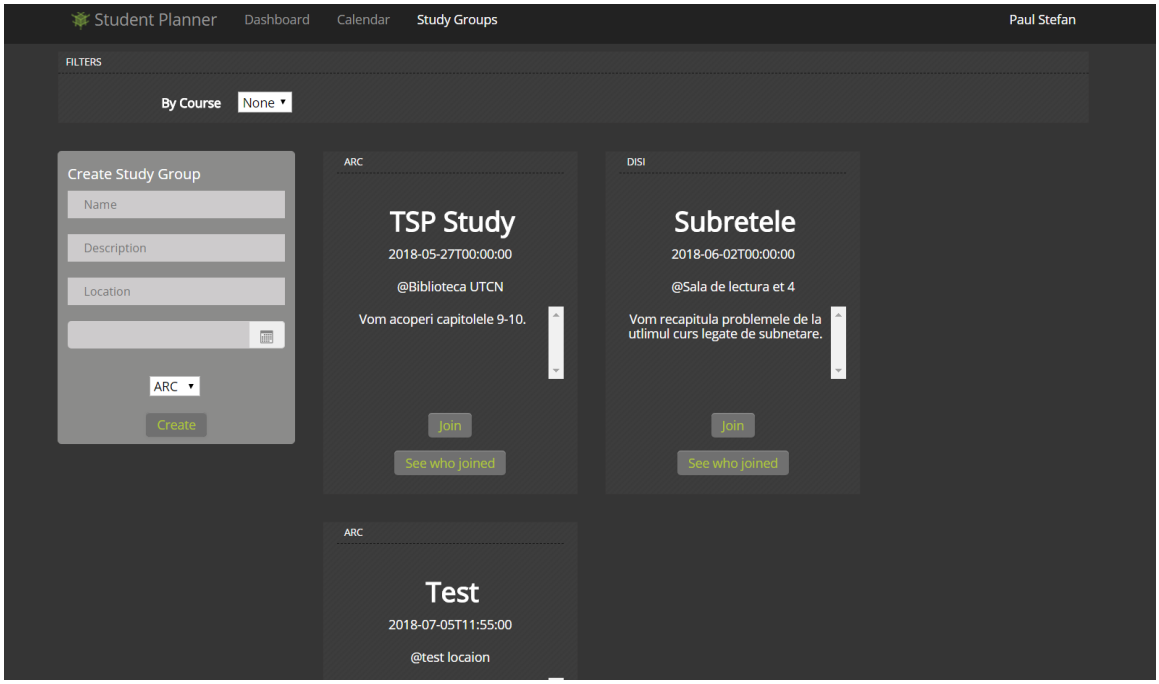

Figura 7.7 Pagina Study Groups

Pentru a adera la un grup de studiu, se va apăsa butonul Join disponibil în cardul grupului. Pentru a vedea o listă cu studenții ce au aderat deja la grupul de studiu, se va apăsa butonul See who joined.

Dacă se dorește crearea unui nou grup de studiu, se va folosi formularul din stânga paginii. Se vor completa toate datele cerute, iar mai apoi se va apăsa butonul Create. După ce se validează datele, grupul de studiu va fi creat, iar un card cu datele specifice lui va apărea în pagină.

7.3.5. Utilizarea paginii Board

În această pagină este disponibilă vizualizarea activităților sub forma unui board de tip Kanban. Împărțirea este făcută în trei categorii. *New* este categoria corespunzătoare activităților încă neîncepute, *In Progress* le arată pe cele în lucru, iar *Done* pe cele terminate deja. Utilizatorul de tip student are posibilitatea de a muta o anumită activitate din stânga spre dreapta, mai specific din *New* în *In Progress* sau din *In Progress* în *Done*.

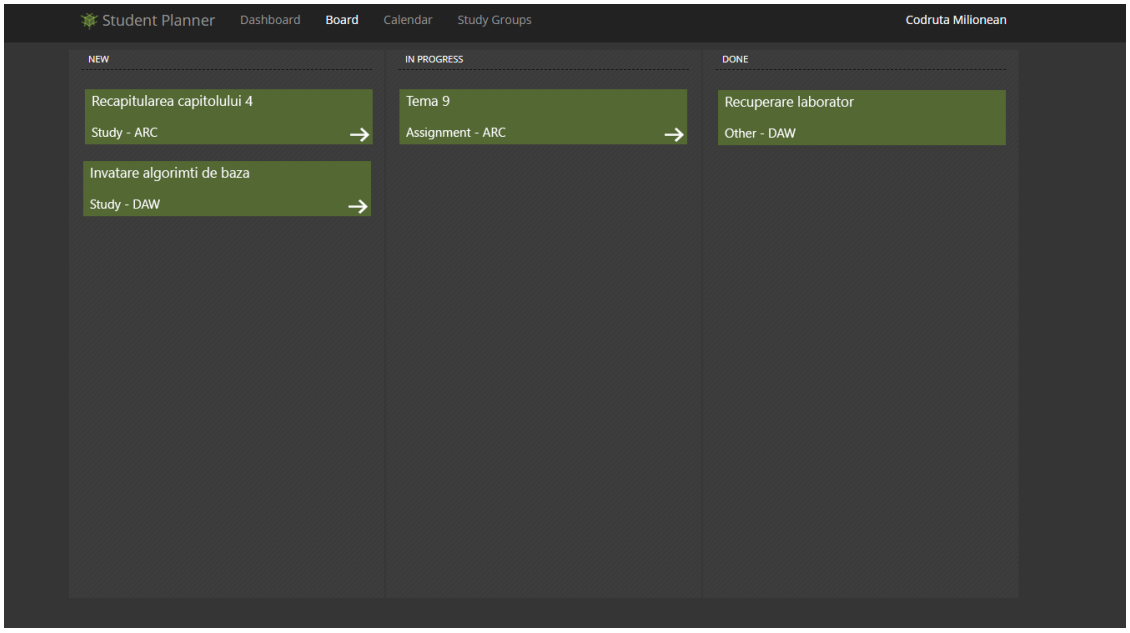

Figura 7.8 Pagina Board

Capitolul 8. Concluzii

8.1. Realizarea obiectivelor propuse

Sistemul dezvoltat a reușit să atingă obiectivele setate inițial, înaintea începerii dezvoltării propriu-zise. Astfel, a fost implementată o aplicație web funcțională, disponibilă celor două categorii de utilizatori stabilite prin intermediul unui client (browser web). De asemenea, aplicația persistă toate datele prin intermediul nivelelor ce țin de server.

S-a reușit implementarea tuturor celor trei componente stabilite ca obiective principale prin intermediul cărora se îndeplinesc cerințele funcționale stabilite. Astfel în aplicație există o componenta de socializare prin intermediul căreia studenții pot participa și pot crea grupuri de studiu, încurajând socializarea între ei. De asemenea, a fost implementată componenta de planificare, componentă prin intermediul căreia se pot planifica diverse activități legate de viața universitară ce pot fi vizualizate într-un mod ușor de înțeles prin intermediul unui calendar. Nu în ultimul rând, există și componenta de administrare disponibilă doar utilizatorilor de tip administrator, responsabilă cu managementul datelor. Prin intermediul ei un administrator poate adăuga noi profesori, cursuri, clase ce vor defini orarul studenților. De asemenea ei pot adăuga examene, evenimente sau vacanțe ce vor fi mai apoi disponibile în calendarul studentului.

Aplicația poate fi accesată de mai mulți utilizatori simultan, fără ca interacțiunile unuia să le afecteze pe celelalte datorită realizării asincrone a operațiilor și a timpilor mici de răspuns. În cazul în care există un număr prea mare de utilizatori ce doresc să folosească aplicația simultan, problema se poate rezolva prin mutarea a o parte sau a tuturor modulelor pe un server ce suportă o încărcătură mai mare, fără a schimba detaliile de implementare. Acest lucru demonstrează extensibilitatea si adaptabilitatea aplicației, dar și cuplajul mic între componente. O altă abordare ce poate rezolva problema încărcăturii mari de requesturi este mutarea aplicației pe un server de tip cloud. Majoritatea furnizorilor a unui astfel de tip de serviciu îl oferă pe principiul „Pay as you use”, adică în perioade cu trafic mare vor fi disponibile mai multe resurse, iar în perioade cu trafic mic mai puține, clientul plătind doar pentru numărul de resurse folosite.

O viteză bună de răspuns a fost atinsă, lucru susținut și de rezultatele prezentate în capitolul 6. Pentru toate acțiunile, răspunsul de la server s-a realizat in limita timpului propus de 200ms.

8.2. Dezvoltări ulterioare

În orice sistem informatic există loc de îmbunătățiri aduse de dezvoltări ulterioare. În acest caz, există numeroase funcționalități noi ce pot fi aduse sistemului pentru îmbunătățirea lui, iar o parte dintre acestea pot fi consultate în lista de mai jos.

- Implementarea posibilității de a urmări progresul unei teme prin marcarea sa cu o valoare numerică ce reprezintă ce procent a fost completat
- Marcarea temelor cu un status ce reprezintă dacă au fost sau nu predate la timp
- Generarea unui raport prin care studentul poate vedea în ce măsură reușește să își predea temele la timp

- Oferirea posibilității ca profesorii să aibă conturi în aplicație. Astfel ei ar putea seta termenele limită pentru predarea temelor în calendar. De asemenea, tot profesorii ar putea și să trimită notificări către studenți în cazul în care un curs s-ar amâna sau reprograma.
- Oferirea posibilității de a marca anumite activități ca fiind prioritare față de altele. Acest lucru s-ar putea realiza prin predefinirea unei ierarhii de priorități din care studentul ar putea alege când creează o activitate
- Îmbunătățirea securității prin generarea de token-uri la autentificare

Bibliografie

[1] N.A. Sireteanu, *Improving the Usability of Web Applications*, 2008 [disponibil online https://ssrn.com/abstract=1322013]

[2] Florida National Univeristy, *10 Reasons Why You Should Form a Study Group*, 2019 [disponibil online https://www.fnu.edu/10-reasons-form-study-group]

[3] M. Hammarberg, J. Sunden, *Kanban In Action*, 2014, Manning Publications

[4] S. Rao, Why Calendars are More Effective Than To Do Lists, 2017 [disponibil online https://medium.com/the-mission/why-calendars-are-more-effective-than-to-do-lists-9bc6ce3bee50]

[5] S. Nakov, *Fundamentals of Computer Programming with C#*, 2013, Faber Publishing

[6] H.-P. Halvorsen, *ASP.NET Web Programming*, 2016, [disponibil online https://www.halvorsen.blog]

[7] M. Haverbeke, *Eloquent JavaScript*, 2018, No Starch Press US

[8] A. Shenoy, *Learning Bootstrap*, 2014, Packt Publishing Limited

[9] S. Holzner, *Ajax A Beginner's Guide*, 2008, McGraw-Hill Education

[10] H. L. Stephan Weißleder, „Top-Down and Bottom-Up Approach for Model-Based Testing of Product Lines", *MBT* , 2013,

[11] C.J. Date, Database Design and Relational Theory: Normal Forms and All That Jazz, 2019, Apress

Anexa 1 Lista procedurilor stocate

1. dbo.usp_ActivitySelect
1. dbo.usp_ActivityInsert
2. dbo.usp_ActivityUpdate
3. dbo.usp_ActivityDelete
4. dbo.usp_CourseSelect
5. dbo.usp_CourseInsert
6. dbo.usp_CourseUpdate
7. dbo.usp_CourseDelete
8. dbo.usp_EventSelect
9. dbo.usp_EventInsert
10. dbo.usp_EventUpdate
11. dbo.usp_EventDelete
12. dbo.usp_MessageSelect
13. dbo.usp_MessageInsert
14. dbo.usp_MessageUpdate
15. dbo.usp_MessageDelete
16. dbo.usp_ProfessorSelect
17. dbo.usp_ProfessorInsert
18. dbo.usp_ProfessorUpdate
19. dbo.usp_ProfessorDelete
20. dbo.usp_StudyGroupSelect
21. dbo.usp_StudyGroupInsert
22. dbo.usp_StudyGroupUpdate
23. dbo.usp_StudyGroupDelete
24. dbo.usp_UserSelect
25. dbo.usp_UserInsert
26. dbo.usp_UserUpdate
27. dbo.usp_UserDelete
28. dbo.usp_GroupSelect
29. dbo.usp_GroupInsert
30. dbo.usp_GroupUpdate
31. dbo.usp_GroupDelete
32. dbo.User_ReadByID
33. dbo.usp_UserSelectAll
34. dbo.usp_ActivitySelectAll
35. dbo.usp_ActivitySelectByUserID
36. dbo.usp_CourseSelectAll
37. dbo.usp_AttendeesSelect
38. dbo.usp_AttendeesInsert
39. dbo.usp_AttendeesUpdate
40. dbo.usp_AttendeesDelete
41. dbo.usp_AttendeesSelectAll
42. dbo.usp_GroupSelectAll
43. dbo.usp_StudyGroupSelectAll

44. dbo.usp_UserSelectByUsername
45. dbo.usp_ProfessorSelectAll
46. dbo.usp_ClassSelectAll
47. dbo.usp_ClassSelect
48. dbo.usp_ClassInsert
49. dbo.usp_ClassUpdate
50. dbo.usp_ClassDelete

Anexa 2 Glosar

Termen	Explicație
Query	Metoda care returnează apelantului date
Command	Metodă care face acțiuni când este apelată
Single responsibility Principle	Principiul responsabilității unice de dezvoltare al aplicațiilor software
Layer al aplicației	Nivel al aplicației, de obicei reprezentat de un namespace sau proiect cu o anumită responsabilitate
API	Application programming interface, set de funcții și proceduri care permit crearea de aplicații ce accesează funcționalități sau date dintr-o aplicație
Three-tier Application	Aplicație pe trei nivele; arhitectură uzuală în dezvoltarea software
Garbage Collector	Program implementat în Common Language Runtime, ce se ocupa de eliminarea din memorie a datelor ce nu mai au scop în contextul curent
Framework	Abstractizare folosită în dezvoltarea aplicațiilor software
JSON	JavaScript Object Notation, sistem de reprezentare al datelor
Single Page Application	Tip de aplicație software ce rulează într-o singură pagina html
Drag and Drop	Acțiunea de a putea trage un element de interfață grafică dintr-un container în altul

Anexa 3 Lista figurilor

Anexa 4 Lista tabelelor