

UNIVERSITATEA "ALEXANDRU IOAN CUZA" DIN IAȘI  
**FACULTATEA DE INFORMATICĂ**



LUCRARE DE LICENȚĂ

# **Elixir - Sistem integrat pentru gestionarea activității academice**

propusă de

***Robert-Ionuț Iacob***

***Sesiunea: Iulie, 2017***

Coordonator științific  
**Lector, Dr. Cristian Frăsinaru**

# **Elixir - Sistem integrat pentru gestionarea activității academice**

*Robert-Ionuț Iacob*

*Sesiunea: Iulie, 2017*

**Coordonator științific**  
**Lector, Dr. Cristian Frăsinaru**

# Declarație privind originalitatea și respectarea drepturilor de autor

Prin prezenta declar că Lucrarea de licență cu titlul „Elixir – Sistem integrat pentru gestionarea activității academice” este scrisă de mine și nu a mai fost prezentată niciodată la o altă facultate sau instituție de învățământ superior din țară sau din străinătate. De asemenea, declar că toate sursele utilizate, inclusiv cele preluate de pe Internet, sunt indicate în lucrare, cu respectarea regulilor de evitare a plagiatului:

- toate fragmentele de text reproduse exact, chiar și în traducere proprie din altă limbă, sunt scrise între ghilimele și dețin referința precisă a sursei;
- reformularea în cuvinte proprii a textelor scrise de către alți autori deține referința precisă;
- codul sursă, imaginile etc. preluate din proiecte *open-source* sau alte surse sunt utilizate cu respectarea drepturilor de autor și dețin referințe precise;
- rezumarea ideilor altor autori precizează referința precisă la textul original.

Iași,

Absolvent *Robert-Ionuț Iacob*

---

(semnătura în original)

# Declarație de consimțământ

Prin prezenta declar că sunt de acord ca Lucrarea de licență cu titlul „Elixir – Sistem integrat pentru gestionarea activității academice”, codul sursă al programelor și celelalte conținuturi (grafice, multimedia, date de test etc.) care însoțesc această lucrare să fie utilizate în cadrul Facultății de Informatică.

De asemenea, sunt de acord ca Facultatea de Informatică de la Universitatea „Alexandru Ioan Cuza” din Iași, să utilizeze, modifice, reproducă și să distribuie în scopuri necomerciale programele-calculator, format executabil și sursă, realizate de mine în cadrul prezentei lucrări de licență.

Iași,

Absolvent Robert-Ionuț Iacob

---

(semnătura în original)

# Acord privind proprietatea dreptului de autor

Facultatea de Informatică este de acord ca drepturile de autor asupra programelor-calculator, în format executabil și sursă, să aparțină autorului prezentei lucrări, Robert-Ionuț Iacob.

Încheierea acestui acord este necesară din următoarele motive:

- continuarea dezvoltării aplicației și adăugarea de noi funcționalități
- integrarea aplicației în diferite medii academice

Resursele utilizate în realizarea produsului-program au fost următoarele:

- Personal: absolventul Robert-Ionuț Iacob
- Tehnologii: Spring, AngularJS, PostgreSQL și alte tehnologii cu caracter open-source
- Fonduri: personale

Resursa descrisă mai sus implică procesul de dezvoltare integral al aplicației.

Iași,

*Decan Adrian Iftene*

Absolvent Robert-Ionuț Iacob

---

*(semnătura în original)*

---

*(semnătura în original)*

# Cuprins

Introducere .....	7
Contribuții .....	9
1. Elixir.....	10
1.1 Despre proiect .....	10
1.2 Aplicații similare .....	10
1.3 Funcționalități.....	11
1.4 Extensii.....	14
1.5 Arhitectura.....	15
1.6 Tehnologii utilizate .....	15
2. Aplicație Web.....	18
2.1 Noțiuni de bază.....	18
2.2 Interfața .....	18
2.3 Modale.....	20
2.4 Localizare .....	22
2.5 Autentificare.....	22
2.6 Secțiuni.....	22
2.7 Panou de control.....	23
2.8 Activități.....	24
2.9 Cursuri.....	26
2.10 Grupe .....	27
2.11 Utilizatori .....	28
2.12 Setări.....	30
2.13 Implementare.....	31
2.14 Concluzie.....	34
3. Platformă (API).....	35
3.1 Noțiuni de bază.....	35
3.2 Arhitectură.....	35
3.3 Structura .....	39
3.4 API REST .....	43
3.5 Concluzie.....	44
4. Baza de date .....	45
4.1 Noțiuni de bază.....	45
4.2 Structură .....	45
4.3 Extensibilitate.....	46
4.4 Concluzie.....	46
5. Deployment .....	47
Concluziile lucrării.....	50
Bibliografie .....	52

# Introducere

## **Motivație**

Ideea platformei Elixir pentru gestiunea activității academice într-o instituție de învățământ a apărut datorită lipsei unei unelte generale care să faciliteze atât accesul studenților la detaliile unui curs (e.g note, activități, prezențe) cât și accesul profesorilor la fișierele și activitatea studenților.

Lipsa unui sistem generalizat de acest tip și implementările diferite la fiecare curs, ce sporesc gradul de confuzie, pentru fiecare curs în parte au dus la materializarea acestei idei. Spre exemplu se elimină necesitatea trimerii temelor cu un format specific de către studenți prin email sau pastrarea situației studenților într-un mod descentralizat.

## **Gradul de noutate**

Aplicația are o idee specifică cu nuanțe noi aplicate unei idei generice și anume gestionarea activității academice într-o platformă. Acest sistem se remarcă prin ușurința utilizării, prezentarea facilă a informației, integrarea mai multor funcționalități ajutătoare și plierea pe necesitățile reale dintr-un sistem academic și rezolvarea acestora.

## **Obiective generale**

Sistemul de management Elixir pentru gestionarea activităților academice are ca obiectiv centralizarea activității tuturor studenților la toate cursurile și prezentarea informației într-un mod facil și securizat pentru toate tipurile de utilizatori. Astfel se va oferi un mod facil studenților de a-și vedea notele/prezențele la cursurile la care sunt înscriși și de a încărca fișiere cât și profesorilor de a nota și avea acces ușor la toate informațiile relevante cursurilor pe care le coordonează (e.g prin rapoarte, exportare date).

Scopul final al acestui sistem fiind acela de a oferi un suport pentru desfășurarea cursurilor și ușurarea muncii administrative pentru ambele părți implicate (studenți/profesori) și nu acela de înlocui cursurile sau de a le muta în mediul online.

## **Metodologie**

Metodologia folosită în dezvoltarea acestei platforme a fost una Agile. Procesul de dezvoltare a avut la baza un board de tip SCRUM realizat pe platforma Trello, în care a fost realizată o listă de task-uri de realizat și actualizat progresul aplicației odata cu finalizarea acestora.

Prioritatea sarcinilor de lucru a fost modificată frecvent în urma feedback-ului primit. De asemenea proiectul a fost integrat cu un sistem de versionare Git.

## Descriere

Arhitectura platformei Exlir este una de tip CMS<sup>1</sup> într-un mod simplificat pentru a deservi exact necesitățile și pentru a face aplicația ușor de folosit și învățat. Utilizatorii autorizați vor putea crea conținut direct din aplicație (e.g activități). De asemenea este o aplicație SPA<sup>2</sup> deoarece toată interacțiunea cu utilizatorul se realizează foarte fluid și fara reîncărcări de pagină. Mai mult aplicația are și o arhitectură Multi-Tenant întrucât se pot accesa diferite baze de date în funcție de preferințele utilizatorului.

Aplicația realizată are 3 părți principale detaliate în următoarele secțiuni ale lucrării și acestea sunt: aplicația web, platforma și bazele de date. Existența mai multor baze de date este motivată de persistența datelor pentru fiecare an academic.

Modul principal de interacțiune cu aplicația este prin intermediul interfeței Web prin care toate tipurile de utilizator se pot conecta și vor avea acces la conținutul relevant acestora. În cadrul aplicației web se va putea gestiona activitatea studenților, vizualiza note, genera rapoarte, importa date și altele.

De asemenea aplicația web este compatibilă și cu dispozitivele mobile (e.g telefon, tabletă) întrucât interfața este una responsive iar informația este separată în unități specifice pentru a fi ușor de recepționat.

## Structura lucrării

Lucrarea va avea 5 capitole principale:

- **Elixir:** capitol ce detaliază funcționalitățile aplicației
- **Aplicație Web:** capitol ce descrie interfața web și modul în care această ofera acces la funcționalități
- **Platformă:** capitol ce detaliază API<sup>3</sup>-ul REST<sup>4</sup> ce constituie platforma și oferă toate funcționalitățile
- **Baza de date:** structura și modul de organizare și stocare a datelor
- **Deployment:** capitol dedicat deployment-ului aplicației într-un mediu de lucru real

---

<sup>1</sup> Content Management System

<sup>2</sup> Single Page Application

<sup>3</sup> Application Programming Interface

<sup>4</sup> Representational State Transfer



# Contribuții

Contribuțiile în realizarea platformei Elixir au avut loc în toate aspectele aplicației. Pornind de la ideea aplicației, crearea arhitecturii și implementării propriu-zise până la design-ul aplicației web și deployment-ul într-un mediu online.

Lista generală a contribuțiilor este detaliată mai jos în funcție de secțiunile platformei este următoarea:

- **Aplicația Web:** realizarea design-ului interfeței web în Photoshop și implementarea acesteia cu HTML 5 / CSS 3, integrarea cu serviciile platformei, realizarea arhitecturii aplicației cu AngularJS și integrarea diferitelor servicii folosite de interfață (e.g Bootstrap, Materialize), asigurarea compatibilității cu dispozitivele mobile, asigurarea securității și consistenței aplicației web
- **Platforma de tip API REST:** realizarea arhitecturii platformei, utilizarea framework-ului Spring și mai exact a versiunii Spring Boot pentru realizarea aplicației, crearea funcționalității de backend și expunerea endpoint-urilor pentru asigurarea facilităților unui API REST, integrarea unui sistem de stocare a fișierelor, integrarea cu bazele de date, asigurarea securității aplicației și a consistenței datelor, restricționarea accesului la date după anumite criterii
- **Baza de date:** crearea schemei baze de date Postgres și a script-ului de creare al acesteia, adăugarea constrângerilor necesare și crearea view-urilor utile pentru platformă și interfață, crearea de baze de date multiple pentru realizarea arhitecturii Multi-Tenant
- **Deployment:** instalarea pe un server de Ubuntu 16.04 a unui server de Apache pentru rularea aplicației Web, instalarea unui server de Tomcat și instalarea platformei dezvoltate în Spring, instalarea și crearea bazelor de date Postgre, realizarea setărilor necesare (e.g drepturi) pentru comunicarea eficientă a părților implicate și expunerea funcționalităților la o adresa web online
- **Documentație online:** crearea unei documentații online pentru aplicație și crearea unor cazuri de utilizare, scrierea documentației în format Markdown
- **Alte contribuții:** convertor de date Python, realizarea fișierelor cu date pentru import în aplicație

# 1. Elixir

## 1.1 Despre proiect

Elixir este o platformă de management a activității studenților la cursurile la care sunt înscriși. Această activitate poate fi supervizată de profesori și li se oferă acestora o suită de funcționalități ce îi vor asista în procesul didactic.

Această platformă a fost realizată într-un mod extensibil pentru a putea fi îmbunătățită cu ușurință și pentru a putea fi folosită în diverse medii academice.

## 1.2 Aplicații similare

În momentul actual nu există aplicații care să aibă exact aceleași funcționalități ca cele implementate în aplicația de față. Există aplicații similare însă cu un scop mult mai general și cu un grup țintă mult mai extins decât al platformei Elixir. Aceste aplicații sunt mai degrabă un eco-sistem de desfășurare al cursurilor decât de asistare al acestora.

### **Moodle** - <https://moodle.org>

Moodle este o aplicație open-source ce permite gestionarea unor cursuri online și a participanților la acestea. Această aplicație pune accentul pe învățarea online și pe crearea de medii de lucru online. Multitudinea de funcționalități și modul în care acesta gestionează fiecare curs face această aplicație mult mai complexă pentru nevoile noastre și mai greu de familiarizat pentru utilizatorii noi.

### **eFront** - <http://www.efrontlearning.net>

Efront este o aplicație de nivel enterprise având grupul țintă sectorul business. În principal această aplicație este destinată companiilor care vor să-și pregătească angajații oferindu-le o platformă online prin care aceștia își pot gestiona munca și realiza sarcinile asignate. Și această aplicație ca și cea anterioară pune accent pe desfășurarea în mediul online a cursurilor ceea ce nu se potrivește cu scopul nostru. De asemenea această aplicație nu este open-source și implicit utilizarea ei are costuri.

**Dokeos** - <http://www.dokeos.com>

Dokeos este o aplicație similară de tip CMS ce are ca scop gestionarea cursurilor online și vizualizarea activității cursanților. De asemenea și această aplicație pune accentul pe desfășurarea online a cursurilor și nu se conformează scopului aplicației noastre.

În concluzie aplicațiile similare prezentate și cele existente nu se pliază suficient de bine pe necesitățile mediului academic pentru care este dezvoltată această aplicație.

### 1.3 Funcționalități

Aplicația de administrare Elixir are diferite funcționalități pentru a oferi un real ajutor profesorilor/administratorilor în organizarea activității didactice din mediul academic dar și studenților prin oferirea unei situații actualizate și facilitarea încărcării de fișiere la cursurile la care sunt înscriși.

În paragrafele următoare vor fi menționate câteva din funcționalitățile principale ale acestei platforme și funcționalități prin care iese în evidență.

#### Experiență personalizată în funcție de utilizator

Aplicația are trei tipuri de utilizatori: studenți, profesori și administratori. Fiecare va avea responsabilități diferite și acces la diferite funcționalități și acțiuni.

- **Student.** Poate încărca fișiere pentru teme, laboratoare, proiecte, chiar și examene. De asemenea poate face parte din grupe/echipe cu alte studenți (studenții sunt organizați pe ani, semianii și grupe). Poate vizualiza prezențele, notele și fișierele proprii la cursurile la care este înscris. Nu are acces la alte informații (e.g. note ale altor studenți, date de la cursuri la care nu este înscris) și lipsesc anumite funcționalități (e.g. import date, generare rapoarte).
- **Profesor.** Are acces doar la cursurile pe care le predă. De asemenea poate vizualiza toate fișierele încărcate de studenți la cursurile sale, poate crea activități la cursurile sale (e.g. laboratoare, proiecte etc), poate acorda note și adăuga prezențe la activitățile sale studenților înscriși. Poate utiliza unelte utile cum ar fi generarea de rapoarte cu situația curentă la un curs (situația prezențelor, notelor sau fișierelor) sau poate importa/exporta date în sistem din formate externe (e.g. CSV). Poate modifica lista studenților înscriși la cursurile sale.
- **Admin.** Are acces la toate funcționalitățile aplicației și este singurul care are acces deplin la toate funcționalitățile aplicației. Conținutul vizualizat de administrator este integral

(e.g toate cursurile, toate notele). Acesta are acces la lista utilizatorilor sistemului în snapshot-ul curent al bazei de date, poate crea/șterge utilizatori și poate gestiona grupele de studenți, cursurile și asocierea de cursuri/profesori.

### **Manager fișiere integrat**

Una din funcționalitățile centrale ale aplicației care a stat la baza realizării acestei platforme este posibilitatea a încărcă fișiere, mai precis laboratoare, proiecte și lucrări ale studenților pentru ca acestea să fie evaluate de către profesori. Studenții încărcă fișierele la care au lucrat în cadrul activității aferente de la cursul asociat.

Acest mod de organizare le va oferi studenților o modalitate ușoară de a-și gestiona activitatea și încărcă cu ușurință fișierele. Profesorii vor avea acces la tot conținutul încărcat de studenți cât și la funcționalități suplimentare.

Conținutul este structurat pe cursuri, semestre și ani pentru o organizare foarte bună a informației. De asemenea stocarea fișierelor este separată în funcție de versiunea bazei de date pe care se lucrează la un moment dat. Spre exemplu fișierele studenților din semestrul II anul 2016-2017 vor fi stocate separat de fișierele studenților din semestrul II anul 2017-2018 chiar și în cadrul unui curs și student comun.

### **Generare rapoarte**

O altă funcționalitate principală foarte utilă pentru profesori este generarea de rapoarte. Profesorii pot genera un raport la unul din cursurile pe care le predau. Rapoartele sunt de 3 tipuri: prezențe, note și fișiere.

Acest raport oferă o imagine generală a situației unui curs întrucât sunt afișate toate detaliile tuturor studenților de la toate activitățile acelui curs. Acest fapt facilitează calcularea punctajelor finale, prezențelor și descărcarea fișierelor. De asemenea profesorul poate modifica foarte ușor datele din raport, filtra datele sau chiar exporta în formate ușor modificabile și reutilizabile (e.g CSV). Un exemplu al unui astfel de raport poate fi consultat în *Anexa 3*.

### **Gestionare activități**

Gestionarea activităților este o parte fundamentală a aplicației. Activitățile sunt modalitatea de bază de organizare a cursurilor. Un curs conține o mulțime de activități, iar o activitate conține prezențe, note și fișiere. Astfel gestionarea acestora este foarte importantă și este facilitată printr-o unealtă de paginare și împărțire a informației în secțiuni de dimensiuni mici

și ușor de urmărit. Aceste unelte sunt asistate de modale ce oferă funcționalități adiționale într-un format ușor de utilizat (e.g adăugare note). Se oferă și funcționalități de editare și ștergere a informațiilor activităților.

### **Modale de interacțiune**

Modalele de interacțiune sunt ferestre de tip pop-up ce completează experiența fluidă a aplicației oferind funcționalități mici și ușor de utilizat. Cele mai importante modale sunt cele pentru crearea de activități, acordarea de note, adăugarea de prezențe și încărcarea de fișiere. Aceste modale pot fi deschise din orice parte a aplicației oferind astfel o experiență consistentă. Modalele pot fi accesate în contexte relevante pentru funcționalitatea lor.

### **Import/Export date**

Platforma oferă și posibilitatea de a importa/exporta date pentru a facilita migrarea către platformă din diferite formate de date.

Importul în aplicației este de 3 tipuri: entități, relații și date. Importul de entități poate fi realizat doar de către administratori și se referă la importarea de studenți, profesori, cursuri și grupe dintr-un format extern. Similar importul de relații importă legături între profesori/cursuri, studenți/cursuri și grupe/studenți. Ultima secțiune se referă la importul propriu-zis de date variabile în sistem (e.g activități, note, prezențe).

Exportul de date din aplicației are loc momentan doar la nivelul rapoartelor și fișierelor încărcate. Fișierele încărcate de studenți pot fi descărcate prin intermediul aplicației. De asemenea rapoartele de date generate pot fi exportate în formate reutilizabile.

### **Multi-Tenant**

Arhitectura Multi-Tenant integrată în aplicație a plecat de la nevoia salvării datelor în momentul trecerii la un an universitar/ semestru nou. O dată cu schimbarea cursurilor și resetarea datelor, cele vechi ar fi șterse. De aceea această funcționalitate ofera acces baze de date diferite în funcție de anul universitar și semestru (e.g 2016-2017, Semestrul 1). Fiecare bază de date are date diferite în funcție de activitatea academică de la acel moment.

La un moment dat, cea mai recentă bază de date este considerată de actualitate, iar celelalte versiuni sunt doar pentru arhivarea datelor. Însă accesul este foarte facil, utilizatorul putând chiar de la logarea în sistem să selecteze la ce bază de date vrea să se conecteze. Baza de date curentă este cea implicită.

## **Filtrarea datelor**

Toate secțiunile aplicației sunt organizate în sub-secțiuni și sunt paginate oferind astfel filtrarea datelor și un acces facil asupra acestora. De asemenea se pot efectua căutări în datele afișate sub formă de tabel și în contexte specifice se pot filtra în funcție de tip. Aceste aspecte împreună cu modalele și cu segmentarea informației în funcție de conținut oferă informația sub o formă foarte ușor de recepționat.

## **Localizare**

Aplicația oferă 2 opțiuni legate de localizare: Engleza și Română. Aplicația a fost dezvoltată inițial cu structura în engleza (astfel endpoint-urile sunt în engleză), dar în setările aplicației poate fi schimbată limba acesteia. Această setare are ca efect imediat traducerea interfeței și actualizarea datelor din interfață pentru a fi conforme cu tipul selectat. De asemenea se actualizează și setările librăriilor utilizate pentru a oferi o experiență consistentă.

## **Funcționalitate API REST**

Platforma aplicației are o arhitectură de tip API REST și a oferi acces facil la conținut și comenzi și în afara interfeței web prin expunerea unor endpoint<sup>5</sup>-uri ușor de utilizat. Astfel serviciul este extensibil și poate fi legat de alte servicii ce vor putea oferi funcționalități suplimentare.

## **Aplicatie Web Responsive**

Aplicația web din cadrul sistemului Elixir a fost construită cu o arhitectură SPA și are o structură responsive. Astfel utilizatorii vor avea o experiență consistentă indiferent de device-ul pe care îl utilizează pentru a accesa aplicația. Secțiunile sunt optimizate pentru a afișa conținutul integral și ușor de recepționat indiferent de mediul de utilizare.

## **1.4 Extensii**

Datorită arhitecturii sistemului, aceasta este extensibil și poate fi integrat cu diferite extensii pentru a se putea adauga noi funcționalități într-un mod facil și pentru a extinde utilitatea platformei. Cateva din ideile de extensii ce ar putea îmbunătăți platforma:

- **Modul de compilare:** are ca funcționalitate posibilitatea de a compila diferite tipuri de fișiere încărcate de studenți pe platformă pentru evaluarea sau testarea acestora.

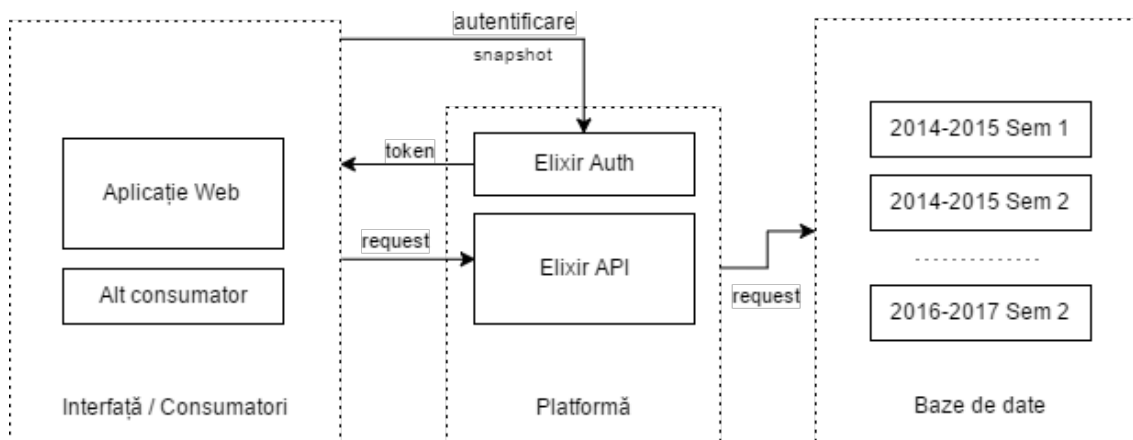
---

<sup>5</sup> Punct de intrare și utilizare în cadrul unui serviciu web

- **Modul verificator plagiere:** are ca funcționalitate verificarea factorului de plagiere între fișierele încărcate de studenți.
- **Modul autentificare API extern:** va avea caracteristica principală de a facilita conexiunea aplicației și cu alte servicii (e.g Fenrir) pentru autentificare.
- **Modul cerere prezențe:** oferă posibilitatea studenților de a cere prezența la o activitate și posibilitatea profesorului de a valida aceste cereri.

## 1.5 Arhitectura

Arhitectura aplicației combină mai multe tipuri de arhitecturi (Multi-Tenant, Multi-Layer) și fiecare componentă este descrisă în secțiunea relevantă. O schemă arhitecturală detaliată mai detaliată poate fi consultată în *Figura 1* de mai jos sau în *Anexa 1*.



*Figura 1: Arhitectura aplicației Elixir*

## 1.6 Tehnologii utilizate

Platforma de management Elxir este o aplicație web responsive implementată folosind tehnologii moderne variate. Aplicația este împărțită în 3 părți: aplicație web, platformă și baza de date.

### Front-end – Aplicație Web

Pe partea de front-end a aplicației se folosesc diverse tehnologii pentru a oferi utilizatorului o experiență cât mai plăcută și un acces cât mai facil. Arhitectura unei aplicații SPA și structura de CMS sunt realizate cu ajutorul mai multor tehnologii.

Se folosesc tehnologii și resurse precum: HTML 5, CSS 3, Javascript (Jquery), Flexbox. De asemenea se folosesc și framework-uri: AngularJS, Jquery, Bootstrap cât și librării pentru

diverse funcționalități: AirDatePicker, Mixpanel, Material Icons. În etapa de dezvoltare a aplicației s-au mai utilizat și unelte ca: GruntJS, Jade Templating.

### **Back-end – Platforma**

Pe partea de back-end, adică platforma propriu-zisă a aplicației, s-a folosit framework-ul Spring ce are la bază limbajul Java, mai exact pachetul Spring Boot, care conține mai multe facilități față de versiunea de bază. Motivul alegerii limbajului Java și a acestui framework a fost stabilitatea platformei, maturitatea limbajului și cantitatea ridicată de documentație și informație disponibilă pentru această combinație.

Pe lângă framework-ul Spring Boot și dependențele sale mai sunt utilizate și alte librării ce completează funcționalitatea necesară platformei. Acestea sunt: Server Tomcat, Spring Boot Starter Web / Data-JPA / DevTools / Starter Security, Jackson Mapper, Javax Servlet API, Apache Commons, ModelMapper, Postgresql Driver, Commons.io, Json WebToken și Spring JDBC.

### **Baza de date**

Datorită importanței stocării datelor este folosită o bază de date stabilă de tip SQL. Pentru a păstra caracterul open-source al aplicației a fost aleasă o variantă gratuită și anume PostgreSQL. Arhitectura Multi-Tenant a aplicației se reflectă prin baze de date diferite cu date de autentificare similare. Gestiunea conexiunilor este realizată de driver-ul platformei.

### **File System Path - Stocare**

Fișierele încărcate de studenți nu vor fi stocate în baza de date datorită dimensiunilor ridicate, formatelor diverse și necesității facilitării accesului. Detaliile acestora vor fi stocate în baza de date, iar fișierele propriu-zise vor fi salvate în sistemul de fișiere al serverului urmând o structură consistentă.

```
Storage / Snapshot / Id Curs / Id Tip / Nume Activitate /  
Id Student / fișier.extensie
```

*Fragment 1: Structura sistemului de fișiere*



Adresa de stocare a datelor este împărțită astfel:

- **Storage:** root-ul adresei de stocare ce va conține toate datele
- **Snapshot:** versiunea bazei de date în care a fost încărcat fișierul (e.g 2016\_2017\_2 înseamnă semestrul 2 din anul universitar 2016 – 2017)
- **Id Curs:** id-ul cursului la care a fost încărcat fișierul
- **Id Tip:** id-ul tipului activității la care a fost încărcat fișierul (e.g laborator, proiect)
- **Nume Activitate:** numele activității la care a fost încărcat fișierul (e.g Laborator 1)
- **Id Student:** id-ul studentului care a încărcat fișierul
- **Fișier:** fișierul propriu-zis încărcat de student

Această modalitate de stocare asigură unicitatea fișierelor și regăsirea acestora cu ușurință. Toate aceste detalii legate de fișier care alcatuiesc adresa din sistemul de fișiere sunt salvate în baza de date și sunt accesate pe baza intrărilor din aceasta.

## 2. Aplicație Web

Platforma Elixir poate fi accesată la adresa **elixir.ionutrobert.com**. Aplicația web este principala metodă de a interacționa cu platforma. Aceasta facilitează utilizarea tuturor funcționalităților printr-o interfață ușor de folosit. De asemenea aceasta poate fi folosită pe orice device, fiind dezvoltată sub o formă responsive care se adaptează în funcție de dimensiunile ecranului pe care este utilizată.

Aplicația a fost realizată folosind: AngularJS<sup>6</sup>, HTML 5 (Jade<sup>7</sup>), CSS (SASS<sup>8</sup>), Javascript (jQuery<sup>9</sup>, GruntJS<sup>10</sup>). Alte dependențe: Material Icons<sup>11</sup>, Flexbox, Bootstrap<sup>12</sup>. De asemenea în platforma web a fost integrat și serviciul Mixpanel<sup>13</sup> cu ajutorul caruia se salvează anumite statistici de utilizare ale aplicației.

### 2.1 Noțiuni de bază

Noțiuni de bază despre utilizarea aplicației. Aplicația web are rolul de suport pentru activitățile didactice ale unui mediu academic. Prin intermediul aplicației se pot accesa toate datele din sistem și se pot modifica/adăuga/actualiza date în funcție de necesitate și de permisiunile utilizatorului autentificat.

### 2.2 Interfața

Un cadru din interfața aplicației poate fi consultat în *Anexa 4*. Interfața aplicației este separată în mai multe secțiuni:

- Bara de meniu (în stânga)
- Secțiunea de conținut (în dreapta)
  - Bara de stare
  - Titlul
  - Bara de unelte
  - Conținut

---

<sup>6</sup> AngularJS: [angularjs.org](http://angularjs.org)

<sup>7</sup> Jade: [pugjs.org](http://pugjs.org)

<sup>8</sup> SASS: [sass-lang.com](http://sass-lang.com)

<sup>9</sup> jQuery: [jquery.com](http://jquery.com)

<sup>10</sup> GruntJS: [gruntjs.com](http://gruntjs.com)

<sup>11</sup> Material Icons: [material.io/icons](http://material.io/icons)

<sup>12</sup> Bootstrap: [getbootstrap.com](http://getbootstrap.com)

<sup>13</sup> Mixpanel: [mixpanel.com](http://mixpanel.com)

- Elemente speciale
- Secțiunea de notificări (în dreapta, suprapusă)
- Modale (cu diferite funcționalități)

### Bara de meniu

Bara de meniu conține 2 tipuri de elemente:

- Tipul utilizatorului
- Butoane către paginile aplicației cu rol de navigare

Opțiunile disponibile sub forma de butoane diferă în funcție de utilizatorul conectat. Se poate observa schema de acces la aceste secțiuni în *Tabelul 1*. Lista integrală a opțiunilor este:

- Panou de control (*Dashboard*)
- Activități (*Activities*)
- Rapoarte (*Reports*)
- Cursuri (*Courses*)
- Grupe (*Groups*)
- Utilizatori (*Users*)
- Import
- Setări (*Settings*)
- Delogare (*Logout*)

	Student	Lecturer	Admin
Dashboard	X	X	X
Activities	X	X	X
Reports		X	X
Courses	X	X	X
Groups	X	X	X
Users			X
Import		X	X
Settings	X	X	X

*Tabel 1: Opțiunile disponibile în funcție de tipul utilizatorului conectat*

*Notă: Deși unele secțiuni sunt accesibile tuturor tipurilor de utilizatori funcționalitatea secțiunilor diferă în funcție de utilizator. Comportamentul individual este descris în capitolele următoare.*

## Secțiunea de conținut

Secțiunea de conținut are de asemenea 5 subsecțiuni principale:

- **Bara de stare:** are rol de navigare, conține detalii legate de path-ul curent în aplicației (breadcrumbs), un buton pentru afișarea/ascunderea meniului și o scurtătură către pagina de profil a utilizatorului
- **Titlul:** conține titlul paginii curente cât și link-uri către alte opțiuni de vizualizare a informației
- **Bara de unelte:** conține diferite scurtături către funcționalități utile (e.g adăugare note, prezențe, fișiere) și este dependentă de contextul în care este afișată (e.g în pagina de cursuri o să conțină butoane relevante)
- **Conținut:** conținutul propriu-zis ce poate consta în tabele, rapoarte, grafică și altele
- **Elemente speciale:** aceste elemente fac parte din conținut, dar au un comportament special (e.g paginare, input-ul de căutare, elemente de filtrare, butoane)

## Secțiunea de notificări

Secțiunea de notificări este situată în partea dreaptă, suprapusă peste secțiunea de conținut. Notificările sunt vizibile doar atunci când este necesar și oferă opțiunea de a fi închise printr-un buton. Dacă nu sunt închise acestea vor dispărea după 10 secunde.

Notificările sunt compuse din 3 părți: Titlu, Conținut și Adresă (dacă este cazul). Se oferă astfel un volum robust de informație. Notificările sunt de 3 tipuri:

- **Succes:** notificările care confirmă realizarea unei acțiuni cu succes (e.g logarea în cont cu succes, adăugarea de date, ștergerea unei informații)
- **Eroare:** notificările care afișează un mesaj de eroare (e.g accesarea unei resurse fără suficiente drepturi - student să acceseze profilul altui student, eșuarea importului de date, date de autentificare incorecte)
- **Neutru:** notificările care afișează un mesaj de informare (e.g alertă cu privire la expirarea token-ului în scurt timp, probleme de conexiune)

## 2.3 Modale

Modalele (pop-uri) aplicației sunt modalitatea principală a aplicației de a interacționa cu conținutul. De obicei acestea conțin diferite formulare pentru a modifica informația prezentată. Sunt elemente ce vin în sprijinul secțiunilor pentru a completa experiența utilizatorului.

Fiecare secțiune are acces la anumite modale prin intermediul subsecțiunii de unelte. Distribuția modalelor în funcție de secțiune poate fi consultată în *Tabelul 2*. Lista completă a modalelor disponibile:

- **Vizualizare activitate:** afișează detalii legata de o activitate a unui student (e.g prezență, notă sau fișier)
- **Adaugă activitate:** oferă posibilitatea adăugării unei activități la unul din cursurile existente în sistem
- **Adaugă prezențe:** oferă posibilitatea de a adauga prezențele unor studenți înscriși la cursul activității selectate
- **Adaugă note:** oferă posibilitatea de a adauga notele unor studenți înscriși la cursul activității selectate
- **Importă fișier:** un student poate încărca un fișier la o activitate de la un curs selectat
- **Adaugă curs:** oferă posibilitatea adăugării unui curs nou direct din interfață
- **Adaugă profesori la un curs:** oferă posibilitatea adăugării unor profesori la un curs direct din interfață
- **Adaugă studenți la un curs:** oferă posibilitatea adăugării unor studenți la un curs direct din interfață
- **Adaugă utilizator:** oferă posibilitatea adăugării unor utilizatori direct din interfață

	Dashboard	Activități	Rapoarte	Cursuri	Grupe	Users	Import	Setări
Vizualizare activitate		X	X					
Adaugă activitate		X	X					
Adaugă prezențe		X						
Adaugă note		X						
Importă fișier		X						
Adaugă curs				X				
Adaugă profesori la un curs				X				
Adaugă studenți la un curs				X				
Adaugă utilizator						X		

*Tabel 2: Distribuirea modalelor în funcție de secțiune*

## 2.4 Localizare

Aplicația web are și opțiuni de localizare. În secțiune de setări poate fi selectată limba aplicației și implicit formatul orar aferent acelei limbi. Cele 2 opțiuni de localizare sunt:

- Engleză: limba implicită, este și limba utilizată de către endpoint-urile expuse de către platformă
- Română: limba alternativă pentru oferirea unei experiențe ușoare în utilizare mai multor categorii de utilizatori

Traducerea aplicației a fost realizată prin utilizarea unui fișier de configurare în funcție de limba selectată. Această modalitate facilitează actualizarea traducerilor, îmbunătățirea lor sau chiar și adăugarea de noi limbi suportate.

Datorită arhitecturii SPA a aplicației, traducerea are loc instant atunci când utilizatorul selectează o limbă, fara niciun ecran de încărcare.

## 2.5 Autentificare

Ecranul de autentificare în sistem este ecranul de bază și primul în interacțiunea cu utilizatorul. Aici sunt necesare un **nume de utilizator** (*username*) si o **parola** (*password*) pentru autentificarea cu succes in sistem.

Datorită arhitecturii Multi-Tenant se poate selecta și versiunea bazei de date în cadrul careia sa va face autentificarea. În funcție de baza de date aleasă, conținutul aplicației va fi din acea baza de date. Daca nici o versiune nu este aleasa se va alege implicit cea mai recentă bază de date și se va încerca autentificarea pe aceasta.

Această secțiune de autentificare este afișată și în cazul expirării token-ului de autentificare. De asemenea în cazul unor erori, acestea sunt afișate sub formă de notificări.

## 2.6 Secțiuni

Secțiunile (paginile) aplicației sunt principala sursă de conținut și reprezintă modalitatea aleasă pentru afișarea dinamică a informației. Fiecare secțiune este explicată în detaliu pentru a ușura înțelegerea structurii aplicației și a tuturor funcționalităților oferite.

Prezentarea secțiunilor are loc în mod integral, adica specificând toate funcționalitățile (chiar și cele pentru modul admin) pentru a oferi o imagine de ansamblu a aplicației cât mai bună. În

secțiunile relevante vor fi făcute mențiuni cu privire la limitarea accesului pentru anumite categorii de utilizatori.

### **Grupe de secțiuni**

Secțiunile aplicației sunt grupate în 3 zone. Fiecare zonă conține secțiuni specifice care au un rol comun. Aceste zone sunt:

- General (*Overview*): Panou de control (*Dashboard*), Activități (*Activities*), Rapoarte (*Reports*),
- Administrativ (*Management*): Cursuri (*Courses*), Grupe (*Groups*), Utilizatori (*Users*), Import
- Cont (*Account*): Setări (*Settings*)

## **2.7 Panou de control**

Secțiunea **Panou de control** (*Dashboard*) reprezintă prima pagina văzută de utilizator după ce se autentifică. Aici sunt agregate mai multe tipuri de informație relevante și funcționalități utile pentru utilizator cum ar fi:

- acces la secțiunile relevante pentru utilizator ale aplicației (e.g cursuri)
- scurtături către diferite funcționalități ale aplicației (e.g adăugare fișier, adăugare note, importare date)
- diferite statistici despre activități

Această secțiune nu are conținut propriu și va avea rolul unui fel de sinteză al informațiilor recente și relevante pentru utilizator. Această secțiune are un aspect diferit în funcție de utilizatorul autentificat.

Spre exemplu un student va avea în această secțiune acces la modalul de încărcare de fișiere și acces la cele mai recente note primite de acesta. Pe de altă parte un profesor va avea acces la modulele de creare de activități și la cele mai recente fișiere încărcate de studenții săi la cursurile sale.

Această diferențiere oferă o experiență personalizată pentru fiecare utilizator în parte punând accent pe necesitățile fiecăruia.

Tip utilizator	Tip acces
Student	Limitat (conținut propriu)
Profesor	Limitat (conținut propriu)
Admin	Nelimitat (conținut integral)

*Tabel 3: Distribuirea tipului de acces la această secțiune în funcție de tipul utilizatorului*

## 2.8 Activități

Secțiunea **Activități** (*Activities*) conține lista tuturor activităților disponibile pentru utilizatorul curent. În cadrul acestei secțiuni există 2 tipuri de afișare:

- listă de activități (*e.g activitățile de la un anumit curs, activitățile de la un anumit laborator*)
- listă de sub-activități (specifică) ale unei activități (*e.g prezențele de la un anumit laborator*)

Modale disponibile:

- vizualizare activitate (*e.g nota de la un laborator luată de un student*)
- adăugare activitate
- adăugare prezențe
- adăugare note
- încărcare fișier

Activitățile sunt de 3 tipuri:

- Prezențe (*Attendances*)
- Note (*Grades*)
- Fișiere (*Files*)

Atât activitățile cât și sub-activitățile pot fi editate/șterse de către profesorii care predau cursul acelor activități sau de către un administrator.

### Lista de activități

Pagina conține lista tuturor activităților vizibile utilizatorului curent (e.g studentul poate vedea doar activitățile de la cursurile în care este înscris). Această listă poate fi filtrată pentru a afișa



doar activitățile unui curs. Din această listă de activități se poate accesa lista tuturor sub-activităților (prezențe, note, fișiere) de la o anumită activitate (e.g. laborator).

### Lista de sub-activități

În modul de vizualizare ale sub-activităților unei activități sunt 4 opțiuni: Toate Activitățile (*All Activities*), Prezențe (*Attendances*), Note (*Grades*), Fișiere (*Files*). Aceste opțiuni filtrează sub-activitățile după tipul lor. Din lista sub-activităților se poate accesa modalul de vizualizare activitate pentru afișarea mai multor detalii (e.g. notă).

### Vizualizarea unei activități (modal)

Vizualizarea unei activități va prezenta detaliile acelei activități. Se vor afișa detaliile de baza ale unei activități:

- **Student:** detalii despre studentul care a realizat acea activitate (e.g. nume)
- **Activitate:** detalii despre activitatea propriu-zisă (e.g. Cursul 1 din data x)
- **Cursul:** detalii despre cursul la care a fost realizată acea activitate (e.g. cursul de Programare Avansată)
- **Extra:** detalii extra despre activitatea curentă (e.g. valoarea notei dacă vizualizăm o notă, link către fișier dacă vizualizăm un fișier încărcat de un student)

```
[ ] > Activități > Java - Programare Avansată > Curs 1 > Prezență
```

Această adresă face referire la faptul că vizualizăm **Activitățile** (*Activities*), mai specific activitățile de la cursul **Java - Programare Avansată**, mai specific sub-activitățile de la activitatea **Curs 1**, mai specific prezențele de la această activitate.

*Notă: Se poate utiliza bara de stare (ce conține adresa curentă a utilizatorului), pentru a accesa secțiuni părinte ale acestei pagini (e.g. activitățile de la cursul Java - Programare Avansată)*

Tip utilizator	Tip acces
Student	Limitat (activități proprii)
Profesor	Limitat (activități de la cursurile predate)
Admin	Nelimitat (toate activitățile)

*Tabel 4: Distribuirea tipului de acces la această secțiune  
În funcție de tipul utilizatorului*

## 2.9 Cursuri

Secțiunea **Cursuri** (*Courses*) conține lista tuturor cursurilor disponibile pentru utilizatorul curent. În cadrul acestei secțiuni avem 2 tipuri de afișare:

- listă de cursuri (e.g toate cursurile utilizatorului)
- vizualizare curs (e.g detalii despre curs)

### Lista de cursuri

Lista de cursuri este afișată în cadrul acestei secțiuni oferind o privire de ansamblu. Lista de cursuri depinde de tipul utilizatorului înregistrat, aceasta fiind specifică acestuia. Orice rând dintr-un tabel poate fi accesat (dacă există drepturile necesare) făcându-se tranziția la vizualizarea aceluia curs.

Cursurile dispun de paginare și se poate folosi funcția de search pentru găsirea unui anumit curs. Există de asemenea opțiuni de administrare (editare/ștergere) disponibile pentru administrator.

### Vizualizarea unui curs

Vizualizarea unui curs va prezenta detaliile aceluia curs. Sunt afișate date legate de profesorii ce predau acel curs, de studenții care sunt înscriși la acel curs și date legate de activitățile ce au avut loc în cadrul aceluia curs.

Titlul paginii conține 4 subtitluri care fac referire la sub-sectiuni pentru un anumit tip de detalii pentru acel curs:

- **General** (*Overview*): secțiune de control, cu butoane către celelalte secțiuni și detalii generale
- **Profesori** (*Lecturers*): profesorii de la acel curs
- **Studenți** (*Students*): studenții de la acel curs
- **Activități** (*Activities*): activitățile aceluia curs

*Notă: Accesul la aceste sub-sectiuni este restrictionat în funcție de tip (e.g. un student poate vedea doar profesorii de la un curs)*

### Exemplu de path de navigare din bara de stare

Aceasta adresă face referire la faptul că vizualizăm secțiunea **Cursuri** (*Courses*), mai specific cursul **Java - Programare Avansată**, mai specific profesorii de la acest curs.

*Notă: Se poate utiliza bara de stare (ce conține adresa curentă a utilizatorului), pentru a accesa secțiuni părinte ale acestei pagini (e.g lista cursurilor accesand Cursuri din path).*

Tip utilizator	Tip acces
Student	Limitat (doar profesorii cursului, doar la cursurile la care este înscris)
Profesor	Limitat (detalii integrale legate de curs, doar la cursurile predate de acesta)
Admin	Nelimitat (toate detaliile legate de toate cursurile)

*Tabel 5: Distribuirea tipului de acces la secțiunea Cursuri în funcție de tipul utilizatorului*

## 2.10 Grupe

Secțiunea **Grupe** (*Groups*) conține lista tuturor grupurilor disponibile pentru utilizatorul curent. În cadrul acestei secțiuni avem 2 tipuri de afișare:

- listă de grupuri (*e.g toate grupurile utilizatorului*)
- vizualizare grup (*e.g detalii despre grup*)

### Lista de grupuri

Lista de grupuri este afișată în cadrul acestei secțiuni oferind o privire de ansamblu. Lista de grupuri depinde de tipul utilizatorului înregistrat, aceasta fiind specifică acestuia. Orice rând dintr-un tabel poate fi accesat (dacă există drepturile necesare) făcându-se tranziția la vizualizarea acelui grup.

Grupurile dispun de paginare și se poate folosi funcția de search pentru găsirea unui anumit grup. Administratorii au drepturi de editare/ștergere asupra grupurilor.

### Vizualizarea unui grup

Vizualizarea unui grup va prezenta detaliile acelui grup. Sunt afișate date legate de componenta grupei, adică o listă cu studenții care fac parte din acea grupă.

Titlul paginii conține detalii legate de grup: an și numărul de studenți.

*Notă: Accesul la această secțiune este restricționat în funcție de tip (e.g. un student poate vedea doar lista grupurilor din care face parte nu și componența unei grupe)*

### Exemplu de path de navigare din bara de stare

[ ] > Grupe > A5

Această adresă face referire la faptul că vizualizăm secțiunea **Grupe** (Groups), mai specific grupa **A5**.

*Notă: Se poate utiliza bara de stare (ce conține adresa curentă a utilizatorului), pentru a accesa secțiuni părinte ale acestei pagini (e.g lista grupurilor accesând Groups din path)*

Tip utilizator	Tip acces
Student	Limitat (doar lista grupurilor din care face parte, nu poate vedea componența unei grupe)
Profesor	Nelimitat (poate vedea toate grupurile și componența acestora)
Admin	Nelimitat (poate vedea toate grupurile și componența acestora)

*Tabel 6: Distribuția tipului de acces la secțiunea Grupe în funcție de tipul utilizatorului*

## 2.11 Utilizatori

Secțiunea **Utilizatori** (*Users*) conține lista tuturor utilizatorilor platformei, disponibilă doar admin-ului cât și posibilitatea de a vedea pagina unui anumit user, funcționalitate disponibilă mai multor tipuri de utilizator. Aceste tipuri de afișare sunt:

- listă de utilizatori (*e.g toți utilizatorii, toți studenții*)
- vizualizare utilizator (*e.g detalii despre un student*)

Utilizatorii sunt de 3 tipuri:

- Studenți (*students*)
- Profesori (*lecturers*)
- Administratori (*admins*)

### Listă de utilizatori

Lista de utilizatori este afișată în cadrul acestei secțiuni oferind o privire de ansamblu. Lista de utilizatori este disponibilă doar unui administrator.

Titlul paginii conține 4 subtitluri ce fac referire la un anumit tip de utilizatori: Toți Utilizatorii (*All Users*), Studenți (*Students*), Profesori (*Lecturers*), Administratori (*Admins*). Accesarea oricărei sub-pagini va afișa doar rezultatele specifice ale acelei pagini filtrând datele din pagina principală (e.g lista studenților). Orice rând dintr-un tabel poate fi accesat (dacă există drepturile necesare) făcându-se tranziția la vizualizarea profilului acelui utilizator.

Utilizatorii dispun de paginare și se poate folosi funcția de search pentru găsirea unui anumit utilizator. Ștergerea și editarea utilizatorilor este de asemenea posibilă.

### **Vizualizarea unui utilizator**

Vizualizarea unui utilizator va prezenta detalii despre acel utilizator. Sunt afișate date (în funcție de utilizator) legate de prezențele, notele, fișierele, cursurile și grupele asociate utilizatorului. În funcție de tipul utilizatorului unele detalii pot lipsi (e.g un profesor nu are grupe, note, prezențe sau fișiere ci doar cursuri).

Permisunile vizualizării profilului sunt diferite în funcție de tipul utilizatorului. Studenții își pot vedea doar profilul personal, în timp ce profesorii/administratorii pot vedea profilele tuturor studenților. Această situație este ilustrată în *Tabelul 8*.

Titlul paginii poate conține până la 6 subtitluri care fac referire la sub-sectiuni pentru un anumit tip de detalii pentru acel utilizator:

- General (*Overview*): secțiune de control, cu butoane către celelalte secțiuni și detalii generale (e.g numărul de prezențe/note)
- Prezențe (*Attendances*): prezențele utilizatorului
- Grades (*Grades*): notele utilizatorului
- Fișiere (*Files*): fișierele utilizatorului
- Groups (*Groups*): grupele utilizatorului
- Cursuri (*Courses*): cursurile utilizatorului

	Student	Profesor	Admin
General	X	X	X
Prezențe	X		
Note	X		
Fișiere	X		
Grupe	X		
Cursuri	X	X	

Tabel 7: Detaliile vizibile în funcție de tipul utilizatorului

### Exemplu de path de navigare din bara de stare

[ ] > Utilizatori > Studenți > Popescu Ioan > Fișiere

Aceasta adresă face referire la faptul că vizualizăm secțiunea **Utilizatori**, mai specific subsecțiunea **Studenți**, mai specific profilul studentului Popescu Ioan, mai specific fișierele acestui student.

*Notă: Se poate utiliza bara de stare (ce conține adresa curentă a utilizatorului), pentru a accesa secțiuni părinte ale acestei pagini (e.g lista studenților accesând Studenți din path)*

Tip utilizator	Tip acces
Student	Limitat (nu poate vedea lista utilizatorilor, poate vedea doar profilul propriu de student/profilurile profesorilor)
Profesor	Limitat (nu poate vedea lista utilizatorilor, poate vedea profilurile studenților/profesorilor)
Admin	Nelimitat (poate vedea lista utilizatorilor și toate profilurile aferente)

Tabel 8: Distribuția tipului de acces la secțiunea Utilizatori în funcție de tipul utilizatorului

## 2.12 Setări

Secțiunea **Setări** oferă acces la diferite setări pentru utilizatorul curent. Funcționalitățile disponibile în această secțiune sunt următoarele:

- vizualizarea setărilor utilizatorului
- schimbarea localizării aplicației prin selecția limbii utilizate

Informațiile despre localizarea aplicației și selecția limbii utilizate au fost detaliate în secțiunea 2.4.

## 2.13 Implementare

Detalii legate de implementarea interfeței web.

### HTML/CSS

Aplicația are un template general care este apoi extins cu ajutorul framework-ului AngularJS să conțină template-urile specifice secțiunii afișate. Fiecare secțiune sau element din pagină a fost gândită ca o componentă independentă pentru a ușura reutilizarea și mentenanța acestora.

În procesul de development template-urile au fost scrise în Jade care este convertit în fișiere HTML. Acestea au fost organizate în funcție de scopul lor: secțiuni, modale și elemente reutilizabile.

De asemenea stilurile au fost realizate cu ajutorul preprocesorului SASS, stiluri care au fost convertite apoi în CSS. Stilurile au fost organizate în funcție de pagina/elementul pe care îl stilizează.

### AngularJS

Arhitectura în AngularJS este o arhitectura ce respectă pattern-ul MVC<sup>14</sup>. Logica aplicației este împărțită în 3 tipuri de componente:

- Controller: conține logica aferentă unei secțiuni
- Serviciu: conține funcții sau logică reutilizabilă
- Configurație: conține configurări ale aplicației

### Controller

La nivel de aplicație există un controller de bază ce conține logica de bază și configurația aplicației. De asemenea există un controller pentru fiecare secțiune a aplicației (e.g listă activități) care conține logica acelei secțiuni și expune funcționalitățile necesare. Fiecare rută accesată de utilizator este asignată unui controller specific. Rutele de acces sunt construite într-o manieră similară cu un API Rest pentru a fi ușor de reconstruit.

---

<sup>14</sup> Model-View-Controller

## Serviciu

Serviciile oferă funcționalitate comună indiferent de secțiunea accesată. Serviciile cele mai importante din aplicație sunt:

- Autentificare: oferă informații despre utilizatorul autentificat și primitive de autentificare și delogare (date salvate într-un cookie)
- Notificare: oferă funcționalități de creare și trimitere de notificări către utilizator în cadrul unor evenimente
- Snapshot: oferă informații despre versiunea bazei de date utilizată și funcționalități legate de schimbarea acesteia

Serviciul de autentificare și snapshot comunică cu platforma prin setarea anumitor date specifice în header-ul request-urilor făcute. Astfel serviciul de autentificare adaugă un token de tip JWT care autentifică request-ul dacă este cazul. Iar serviciul de snapshot adaugă în header-ul fiecărui request făcut și versiunea bazei de date pentru care se face acel request.

## Configurare

Configurarea aplicației conține anumite setări legate de aplicație și utilizate de către celelalte entități. Logica de configurare a aplicației are 2 secțiuni importante:

- Setări generale: conține setări legate de durata notificărilor, server-ul la care se conectează aplicației și permisiunile de accesare în funcție de utilizatorul autentificat
- Localizare: conține textul folosit de aplicația web în toate limbile disponibile

## Exemple de cod

În paragrafele următoare au fost incluse fragmente de cod reprezentative pentru aplicația web și implementării descrise mai sus.

```
snapshotService.setSnapshot = function(target) {
    this.snapshot = target;
    $http.defaults.headers.common.Snapshot = this.snapshot.id;
}

snapshotService.getAllSnapshots = function() {
    return $http.get(config.apiEndpoint+'snapshots').then(function (response) {
        return response.data;
    });
}
```

*Fragment 1: Serviciul Snapshot*



```

authService.login = function (credentials) {
    var deferred = $q.defer();
    $http
        .post(config.apiEndpoint+'login', credentials)
        .then(function (response){
            $cookies.putObject('authUser', response.data, {expires: new
Date().addHours(2)});
            $rootScope.authUser = $cookies.getObject('authUser');
            $state.go('base.dashboard');
            deferred.resolve({
                'user': $rootScope.authUser
            })
        }, function(response){
            deferred.reject(response.data);
        });
    return deferred.promise;
};

```

*Fragment 2: Serviciul de autentificare*

```

notifications: {
    showDate: false,
    autoDismissTime: 8000, //milliseconds
},

authorizedRoles: {
    activities: {
        list: ['*'],
        sublist: ['*'],
        view: ['*'],
        create: ['ADMIN', 'LECTURER'],
        edit: ['ADMIN', 'LECTURER'],
        delete: ['ADMIN', 'LECTURER']
    },
},

```

*Fragment 3: Fișier de configurare*

```

$scope.filterStudents = function(){
    $scope.loading = true;
    var paramGroup = $scope.selectedGroup;
    angular.forEach($scope.report.students, function(student, key){
        if (typeof paramGroup !== 'undefined' && paramGroup !== null){
            student.visible = false;
            angular.forEach(student.groups, function(group, gKey){
                if (group.id === paramGroup){
                    student.visible = true;
                }
            });
        } else{
            student.visible = true;
        }
    });
    $scope.loading = false;
};

```

```
};
```

*Fragment 4: Funcție din ReportController*

## **2.14 Concluzie**

Aplicația Web oferă funcționalități multiple și este principală metoda și cea recomandată de a interacționa cu platforma aplicației. S-a pus accent pe consistența datelor și robustețea sistemului prin dezvoltarea simultană a platformei și a aplicației web.

Mai mult se asigură securitatea platformei prin protejarea la tipuri de atacuri cunoscute și utilizarea autentificării pe bază de token oferită de platformă.

Arhitectura aplicației web este una extensibilă și se pot adăuga sau integra ușor funcționalități noi în aplicație.

## 3. Platformă (API)

Platforma aplicației și cea mai importantă componentă a proiectului Elixir este un API REST realizat cu ajutorul framework-ului Spring (Spring Boot) în limbajul Java.

### 3.1 Noțiuni de bază

Acest API oferă o suită de funcționalități și asigura atât securitatea și integritatea datelor cât și comunicarea dintre consumator (e.g aplicația web) și bazele de date. De asemenea integrează un sistem de stocare a fișierelor și restricționarea accesului la date după anumite criterii.

### 3.2 Arhitectură

#### Arhitectura Multi-Tier

Platforma aplicației Elixir are o arhitectură Multi-Tier prin organizare și modul de comunicare al datelor. Astfel există 3 nivele principale ce pot fi identificate în API-ul aplicației:

- Presentation: de obicei reprezentat de interfață dar în cadrul platformei, controllere-le sunt cele care constituie acest layer întrucât ele apelează serviciile necesare
- Service: reprezentat de suita de servicii oferită de aplicație
- Data: reprezentat de modelele disponibile în platformă și repository-urile care oferă acces la baza de date

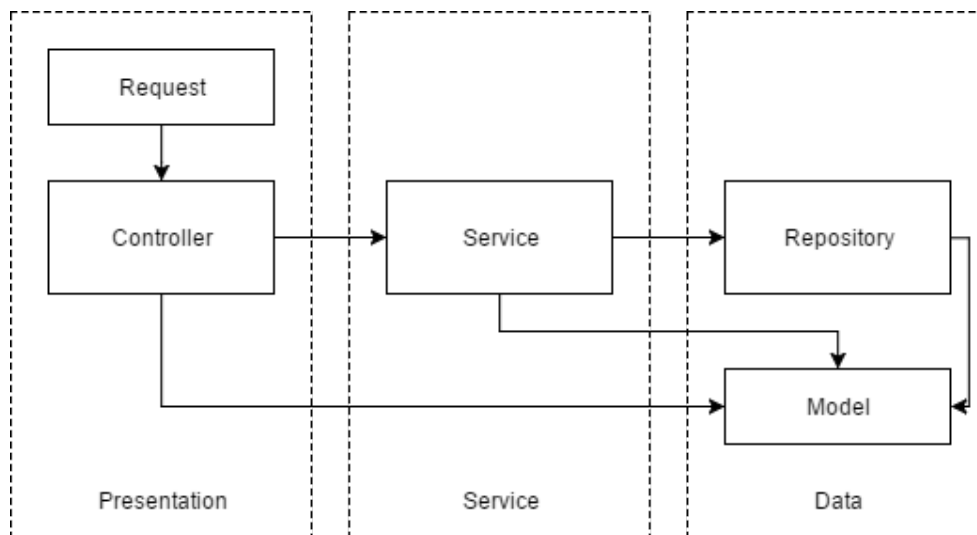
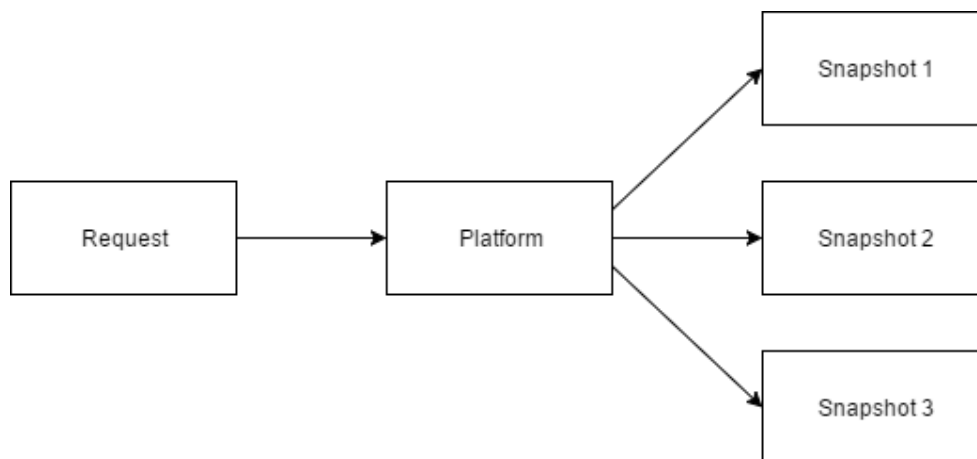


Figura 2: Arhitectură Multi-Tier

## Arhitectură Multi-Tenant

Arhitectura Multi-Tenant a aplicației este dată de faptul că utilizatorul prin setarea unui header poate selecta către ce bază de date (snapshot<sup>15</sup>) să redirecționeze request-urile. Astfel se poate face schimbul între baze de date multiple și pot fi utilizate mai multe baze de date în același timp de către utilizatori diferiți.

Decizia de conectare la o anumită bază de date este realizată în cadrul platformei. Schema acestei arhitecturi poate fi observată atât în *Anexa 1* cât și în *Figura 3* prezentată mai jos.



*Figura 3: Arhitectură Multi-Tenant*

Implementarea acestei arhitecturi s-a făcut cu ajutorul mai multor mecanisme printre care *AbstractRoutingDataSource* din framework-ul *Spring*, fișiere de configurare și filtre de prindere și selectarea a versiunii utilizate. Procesul se desfășoară astfel:

- Primirea unui request cu un header ce conține Snapshot-ul vizat de acel request (snapshot-ul cel mai recent este selectat în cazul lipsei acestui header)
- Se selectează din lista de versiuni de baze de date disponibile baza de date identificată
- Se setează într-un thread local versiunea bazei de date utilizată pentru request-ul curent
- Mecanismul de *RoutingDataSource* din *Spring* (ce acționează ca un *Connection Pool*) va returna baza de date utilizată
- Se va face request-ul către baza de date identificată

Câteva secvențe de cod ce ilustrează implementarea arhitecturii descrise mai sus sunt prezentate în fragmentele următoare:

---

<sup>15</sup> Versiune a bazei de date la un moment dat

```

public class RoutingDataSource extends AbstractRoutingDataSource {

    @Override
    protected Object determineCurrentLookupKey() {
        DbSnapshot currentSnapshot = DbSnapshotHolder.getDbSnapshot();
        return currentSnapshot != null ? currentSnapshot.getKey() :
        DbSnapshotHolder.getDefaultSnapshot().getKey();
    }
}

```

*Fragment 5: Implementarea clasei RoutingDataSource*

```

@Override
    public void doFilter(ServletRequest request, ServletResponse response,
        FilterChain chain) throws IOException, ServletException {
        HttpServletRequest httpRequest = (HttpServletRequest) request;
        String snapshotKey = httpRequest.getHeader("Snapshot");
        DbSnapshot snapshot = snapshotKey != null ? new DbSnapshot(snapshotKey) :
        DbSnapshotHolder.getDefaultSnapshot();
        DbSnapshotHolder.setDbSnapshot(snapshot);
        try {
            chain.doFilter(request, response);
        } finally {
            DbSnapshotHolder.clearDbSnapshot();
        }
    }
}

```

*Fragment 6: Filtrul de extragere al header-ului Snapshot*

```

private Map<Object, Object> getTargetDataSources() {
    HashMap<Object, Object> targetDataSources = new HashMap<>();
    for (DbSnapshot dbSnapshot : DbSnapshotHolder.SNAPSHOTS) {
        final DriverManagerDataSource dataSource = new
        DriverManagerDataSource();

        dataSource.setDriverClassName(env.getProperty(dbSnapshot.getProperty("driver-
        class-name")));
        dataSource.setUrl(env.getProperty(dbSnapshot.getProperty("url")));

        dataSource.setUsername(env.getProperty(dbSnapshot.getProperty("username")));

        dataSource.setPassword(env.getProperty(dbSnapshot.getProperty("password")));
        targetDataSources.put(dbSnapshot.getKey(), dataSource);
    }
    return targetDataSources;
}

```

*Fragment 7: Salvarea conexiunilor inițiale cu toate bazele de date disponibile*

## Stocare în sistemul de fișiere

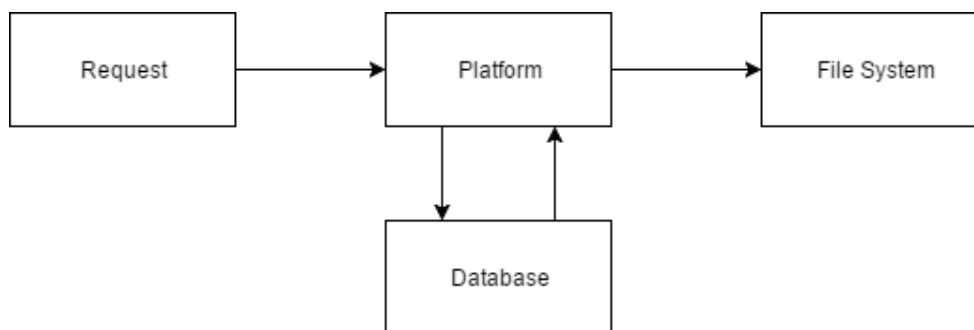
Fișierele încărcate de utilizatori sunt stocate în sistemul de fișiere al server-ului și doar detalii despre acestea sunt salvate în baza de date. Se folosește această metoda datorită flexibilității sistemului de fișiere și spațiului de stocare permisiv.

În baza de date se salvează date necesare identificării unice a fișierului. Structura sistemului de fișiere este descrisă în *Fragment 1* din cadrul capitolului 1.

Pentru accesarea unui fișier din sistemul de fișiere (similar și la încărcare) vor avea loc următorii pași:

- Realizarea request-ului cu un token valid
- Verificarea permisiunilor utilizatorului care face request-ul
- Interogarea datelor legate de fișierul căutat (e.g nume, curs)
- Construirea adresei fișierului din datele interogate la pasul anterior
- Accesarea fișierului și returnarea acestui pe baza adresei construite anterior

Acest proces este ilustrat și în *Figura 3* disponibilă mai jos.



*Figura 3: Accesarea unui fișier din sistemul de fișiere*

De asemenea fragmentele de mai jos ilustrează implementarea acestei funcționalități.

```
String snapshotKey = request.getHeader("Snapshot");
DbSnapshot snapshot = snapshotKey != null ? new DbSnapshot(snapshotKey) :
DbSnapshotHolder.getDefaultSnapshot();
User user;
try{
    user = jwtService.verifyToken(key);
} catch (Exception e) {
    return new ResponseEntity<> (new ErrorMessageWrapper ( emp.NOT_AUTHORIZED),
HttpStatus.BAD_REQUEST);
}
if (user==null) {
```

```

        return new ResponseEntity<>(new ErrorMessagesWrapper(emp.NOT_AUTHORIZED),
        HttpStatus.BAD_REQUEST);
    }
    ServletContext context = request.getServletContext();
    ActivityFile activityFile = activityFileService.findById(fileId);
    if (!user.getEmail().equals(activityFile.getStudent().getEmail()) &&
    user.getType().equals("student")) {
        return new ResponseEntity<>(new ErrorMessagesWrapper(emp.NOT_AUTHORIZED),
        HttpStatus.BAD_REQUEST);
    }
    ResourceDto resourceDto = new ResourceDto(activityFile, snapshot);
    if (!resourceDto.isValid()) {
        return new ResponseEntity<>(HttpStatus.BAD_REQUEST);
    }
    Path filePath = storageService.load(resourceDto.toRelativePath());
    if (filePath == null) {
        return new ResponseEntity<>(new ErrorMessagesWrapper(emp.NOT_FOUND),
        HttpStatus.NOT_FOUND);
    }
    File file = filePathToFile();

```

*Fragment 8: Implementarea metodei de returnare a unui fișier*

## Principii

În procesul de dezvoltare al aplicației au fost respectate pe cât posibil cât mai multe concepte de programare. S-a urmarit crearea unor module decuplate și cu o coeziune ridicată.

Pattern-ul cel mai utilizat în aplicație este cel MVC<sup>16</sup>. Acesta apare atât în cadrul aplicației web dar mai ales în cadrul platformei. De asemenea s-au mai folosit și alte design pattern-uri cum ar fi: Singleton, Decorator, Data Mapper și altele. De asemenea s-a urmărit respectarea principiilor SOLID<sup>17</sup> pentru dezvoltarea unei aplicații orientată obiect.

Securitatea aplicației este asigurată atât de autentificarea pe bază de token cât și folosirea unui algorithm de hash (Bcrypt) pentru stocarea parolelor într-o manieră securizată.

## 3.3 Structura

Aplicația are o structură complexă descrisă în secțiunile următoare. Aplicația este structurată într-un mod specific unui limbaj OOP<sup>18</sup> și anume pe clase.

### Modularizarea claselor

<sup>16</sup> Model View Controller

<sup>17</sup> Single-responsibility principle, Open-closed principle, Liskov substitution principle, Interface segregation principle, Dependency Inversion Principle

<sup>18</sup> Object Oriented Programming

Clasele sunt împărțite pe pachete pentru o decuplare a acestora și o îmbunătățire a procesului de dezvoltare și mentenanță. Pachetele din cadrul platformei sunt următoarele:

- Auth: conține clasele legate de autentificarea pe platformă
- Config: conține clase de configurare pentru aplicație
- Controller: conține clasele ce expun endpoint-uri către consumatori
- Exceptions: conține excepții personalizate folosite de aplicație
- Model: conține modelul datelor, realizat după structura bazei de date
- Repository: conține clasele ce comunică cu baza de date pe baza modelului
- Service: clase ce folosesc repositoryele pentru a expune anumite funcționalități către controllere

Funcționalitatea claselor este descrisă în funcție de pachetul din care fac parte.

### **Pachetul Auth**

Pachetul de autentificare conține toate clasele care au legătură cu autentificarea utilizatorului în sistem. În cazul curent clasele componente sunt: JwtAuthenticatedUser, JwtAuthenticationProvider, JwtAuthFilter, JwtAuthToken, SnapshotFilter.

Clasele adnotate cu Jwt oferă funcționalități ca extragerea token-ului din header și autentificarea acestuia, cât și o clasă suport pentru acel token.

Filtrul Snapshot a fost încadrat tot în pachetul de autentificare deoarece acesta verifică existența unui header care să seteze bază de date utilizată în momentul autentificării. În cazul absenței acelui header, se va utiliza cea mai recentă bază de date disponibilă.

### **Pachetul Config**

Pachetul de configurare conține toate setările aplicației care se aplică în momentul în care serverul cu aplicația este inițializat. Framework-ul Spring prin implementarea Spring Boot oferă posibilitatea de a configura diferite aspecte ale aplicației prin anumite clase adnotate corespunzător. Aceste setări sunt diverse pornind de la configurarea modului de stocare al fișierelor până la setări legate de CORS<sup>19</sup>.

Clasele conținute de acest pachet sunt: ApplicationStartup, DbSnapshotHolder, MvcConfig, RoutingDataSource, SecurityConfig, StorageConfig. Aceste clase configurează aspecte precum:

---

<sup>19</sup> Cross Origin Resource Sharing



- Autorizarea request-urilor în funcție de adresă
- Setarea locației de salvare în sistemul de fișiere
- Setarea bazei de date utilizate în funcție de snapshot-ul setat
- Setarea parametrilor CORS pentru a oferi acces doar consumatorilor autorizați

### **Pachetul Model**

Pachetul model este pachetul de bază al aplicației care modelează toate datele utilizate de aceasta. Clasele din acest pachet sunt utilizate de toate celelalte clase, fiind modelate după datele stocate în baza de date.

Acest pachet conține mai mult sub-pachete: Activity, Common și User. Aceste sub-pachete au rolul de a organiza multiplele clase din acest pachet. Gasim astfel în acest pachet toate clasele de bază necesare aplicației cum ar fi: Student, Lecturer, Course, DbSnapshot, Pager, LoginCredentials, ActivityFile și altele.

Mai există 2 sub-pachete cu un rol specific:

- Dto
- Serializer

Sub-pachetul Dto conține cele mai utilizate clase într-o formă modificată cu rolul de DTO<sup>20</sup> utilizate de obicei la rezolvarea request-urilor posibile. Urmatorul sub-pachet, Serializer, are rolul de a seta anumite detalii legate de serializarea modelelor în format JSON atunci când acestea sunt returnate ca răspuns unui request. Acesta din urmă oferă controlul exact al informațiilor returnate (e.g parola unui user nu este returnată deși face parte din model și DTO).

### **Pachetul Repository**

Acest pachet conține doar interfețe de comunicare cu baza de date. Aceste interfețe extind o clasă a framework-ului Spring și sunt implementate la rularea server-ului în funcție de driver-ul bazei de date oferind astfel o decuplare de un cod specific al bazelor de date.

Astfel tipul bazei de date poate fi schimbat foarte ușor. Această decuplare are un comportament similar cu API-ul JPA.

---

<sup>20</sup> Data Transfer Object

La fel ca in cazul pachetului Model, clasele sunt grupate in sub-pachete unde este cazul. Interfețele repository-urilor oferă astfel acces la operațiunile de bază CRUD<sup>21</sup> dar și la metode mai complexe utilizate la căutare sau la extragerea datelor specifice la care un utilizator are acces.

### **Pachetul Service**

Pachetul Service conține serviciile de bază ce expun funcționalitățile principale ale platformei. Practic, aici se orchestrează toată logica aplicației utilizându-se repository-urile pentru accesul la date.

Acest pachet are mai multe sub-pachete utile pentru organizare: Activity, Common, Storage, User. Aceste servicii sunt injectate în controllere atunci când sunt necesare și oferă funcționalitatea cerută.

Fiecare serviciu are o interfață și o implementare pentru a decupla clasele și pentru a putea schimba implementările în caz de necesitate.

Serviciile oferite sunt variate printre care se numără:

- Serviciu validare token Jwt
- Serviciu salvare fișiere în sistemul de fișiere
- Servicii de interacționare cu entitățile (e.g Studenți, Cursuri, Activități)
- Serviciu de autentificare

La nivelul serviciilor se verifică permisiunile asupra informației prin verificarea tipului de utilizator autentificat. Spre exemplu un student nu va avea acces decât la profilul personal, deci un request la o metoda care returnează datele unui utilizator va returna un răspuns gol dacă profilul cerut nu este cel al studentului care face request-ul. Astfel de limitări sunt realizate în cadrul tuturor metodelor pentru a oferi un acces personalizat și a spori securitatea și integritatea aplicației.

### **Pachetul Controller**

Acest pachet conține toate controllere-le aplicației care expun endpoint-uri în cadrul serverului. Acestea sunt organizate de asemenea în sub-pachete: Activity, Common, Course, Group, User.

---

<sup>21</sup> Create Read Update Delete

Un controller are mai multe metode adnotate corespunzător care vor răspunde mai multor tipuri de request-uri. Fiecare metodă se folosește de serviciile necesare pentru a realiza acțiunea asignată. Maparea datelor primite de un controller se face cu ajutorul modelelor Dto.

Tot în controllere se verifică și integritatea datelor primite/trimise. De asemenea metodele trimit pe lângă răspunsul propriu-zis și un status HTTP pentru a oferi o semantică răspunsului și a ușura comunicarea cu aplicațiile consumatoare (e.g BAD\_REQUEST).

### 3.4 API REST

Platformă ofera funcționalitățile unui API REST întrucât expune funcționalitățile la endpoint-uri numite folosind convențiile specifice unui astfel de API. Mai mult returnează și un cod de răspuns HTTP pentru a oferi semantică acestora și a ușura integrarea.

Astfel adresele aplicației sunt construite după modelul prezentat în fragmentul de mai jos.

```
/api/{resource}/{attribute}/
```

GET

*Fragment 9: Structura unui endpoint expus de platformă*

O adresă de tip endpoint are următoarele componente:

- Prefixul api: poate fi schimbat pentru versiunile mai noi, apare pentru a putea oferi versiuni diferite ale aceluiași sistem în caz de actualizări
- Resursa: resursa asupra careia se face request-ul (e.g *students* va returna lista studenților)
- Alte attribute: se selectează o resursă mai specifică pe baza unor attribute (e.g *students/2* va returna studentul cu id 2 în timp ce *courses/year/3* va returna toate cursurile din anul 3)
- Request-ul realizat va accesa un alt endpoint în funcție de tipul acestuia (e.g GET, POST, PUT, PATCH sau DELETE)

```
/api/attendances/3/5
```

DELETE

*Fragment 10: Exemplu de request care va cere ștergerea prezenței de la activitatea cu id-ul 3 a studentului cu id-ul 5*

```
/api/courses/4/activities
```

```
GET
```

*Fragment 11: Exemplu de request ce va returna toate activitățile de la cursul cu id-ul 4*

### 3.5 Concluzie

Platforma este piesa de baza a aplicației întrucât aceasta asigură atât consistența și securitatea datelor cât și comunicarea dintre baza de date și consumatori.

În realizarea acestei platforme s-au integrat diferite tipuri de arhitecturi cum ar fi: Multi-Layer, Multi-Tenant, s-au folosit pattern-uri cum ar fi MVC și au fost realizate funcționalități mai complexe cum ar fi: integrarea cu sistemul de fișiere și oferirea accesului condiționat asupra datelor.

Acest API oferă o suită de funcționalități și asigură atât securitatea și integritatea datelor cât și comunicarea dintre consumator (e.g aplicația web) și bazele de date. De asemenea integrează un sistem de stocare a fișierelor și restricționarea accesului la date după anumite criterii.

Din punct de vedere al dezvoltării platforma poate fi extinsă foarte ușor datorită structurării acesteia în pachete variate, decuplării claselor și utilizării interfețelor. Utilizarea unui framework modern (Spring) asigură longevitatea platformei și suportul din partea comunității.

Pe de altă parte, integrarea API-ului este facilitată de arhitectura de tip API Rest și utilizarea unor convenții în denumirea endpoint-urilor expuse.

## 4. Baza de date

Baza de date a platformei Elixir este cea mai importantă parte din punctul de vedere al datelor deoarece aici sunt salvate toate datele ce sunt transmise.

### 4.1 Noțiuni de bază

Tipul bazei de date utilizat este SQL datorită stabilității oferite și modului facil de integrare. Versiunea utilizată este versiunea PostgreSQL datorită caracterului open source al acesteia. Schema bazei de date poate fi consultată în *Anexa 1*.

#### Multi-Tenant

Arhitectura Multi-Tenant a aplicației se reflectă în mai multe baze de date construite după aceeași schemă, doar cu un conținut diferit în funcție de versiunea pe care o reprezintă. Astfel prezentarea schemei unei baze de date implică prezentarea oricărei alte versiuni.

### 4.2 Structură

Datele din baza de date sunt structurate în 2 tipuri de obiecte: tabele și view-uri. Tabelele stau la baza aplicației în timp ce view-urile disponibile există pentru a agrega diferite date și pentru a ușura integrarea cu platforma.

#### Tabele

Constrângerile bazei de date există la fiecare tabel pentru a asigura consistența datelor. Indecșii folosiți sunt cei pe cheile primare ale tabelelor, optimizând astfel timpul de răspuns al cererilor către baza de date.

Tabelele sunt împărțite în 2 categorii:

- Entități: utilizate pentru modelarea obiectelor de bază utilizate în aplicație (*students, lecturers, admins, activities, activity\_types, activity\_attendances, activity\_grades, activity\_files*)
- Legătură: utilizate pentru maparea relațiilor Many-to-Many dintre tabele (*groups\_students, course\_ownerships, course\_attendants*)

## View-uri

Obiectele de tip view din baza de date au rolul de a agrega anumite date din mai multe tabele pentru a ușura integrarea cu platforma și a optimiza cererile făcute către baza de date.

Sunt disponibile 2 view-uri în baza de date:

- Users: conține toți utilizatorii platformei combinând datele comune ale studenților, profesorilor și administratorilor
- Activities\_join: conține toate activitățile (prezențe, note, fișiere) agregate într-un singur tabel pentru oferirea unei imagini de ansamblu într-un mod facil

Utilitatea acestor view-uri este vizibilă în cadrul aplicației web întrucât este optimizat numărul de cereri către baza de date și timpul de răspuns. Acestea stau la baza unor unelte importante cum ar fi administrarea utilizatorilor sau generarea de rapoarte.

## 4.3 Extensibilitate

Un tabel special este tabelul *activity\_types* care conține tipurile de activități suportate de sistem. Tipurile de activități suportate de versiunea inițială a aplicației sunt: Curs, Laborator, Seminar, Proiect, Test, Examen, Colocviu. Extinderea datelor din acest tabel va oferi suport pentru mai multe tipuri de activități ce pot fi adăugate în sistem.

De asemenea prin adaugarea de noi tabele de tipul *activity\_\** se pot adăuga noi sub-tipuri de sub-activități (e.g premiu). Aceste noi sub-activități vor fi atașate unei activități și unui utilizator (nu neapărat student).

## 4.4 Concluzie

În concluzie, baza de date a aplicației oferă suport pentru toate serviciile oferite de platformă și are un caracter extensibil facilitând adăugarea de noi funcționalități. Datele sunt salvate într-un mediu consistent și versionarea acestora se face cu ajutorul arhitecturii Multi-Tenant.

## 5. Deployment

Aplicația Elixir este formată din cele 3 componente descrise în capitolele anterioare: Aplicația Web, Platforma API și baza de date. Aceste 3 componente trebuie instalate și configurate pentru a putea instala aplicația într-un mediu de dezvoltare sau într-un mediu de producție.

### Aplicația Web

Aplicația Web conține în folderul de bază toate fișierele necesare în procesul de dezvoltare (e.g task-uri automate, stiluri SASS, template-uri Jade). În cadrul acestui folder există un folder *public* care conține toate fișierele necesare pentru varianta de producție a aplicației.

Pentru generarea cu succes a folderului public se vor realiza următorii pași:

- de la linia de comanda se va executa comanda **npm install** pentru a instala toate dependențele (este necesară instalarea environment-ului NodeJS<sup>22</sup>) printre care și GruntJS necesar la următorul pas
- se vor verifica configurările din fișierul *js/config.js*, setarea cea mai importantă fiind adresa platformei cu care va comunica aplicația, aceasta trebuie setată către server-ul și portul la care este disponibil API-ul (e.g *http://elixir.ionutrobert.com:8080/elixir-api/api/*)
- se va executa comanda **grunt build** tot de la linia de comandă, comandă ce va compila și converti toate fișierele din modul de development în modul de producție și le va salva în directorul *public*
- se va genera directorul *public* care va putea fi transferat pe server

Apoi pe server, trebuie pus fișierul public în folderul de unde va fi expusă aplicația. Se recomandă instalarea unui server de Apache care să facă host la fișierul *index.php* din folderul *public*. De restul configurărilor și de securitate se ocupă framework-ul aplicației.

### Platforma

API-ul REST al aplicației necesită construirea unui fișier de tip *.war* care împachetează toate clasele într-un fel de arhivă ușor de utilizat. Această arhivă va fi urcată pe server și va fi rulată de un server de Tomcat.

---

<sup>22</sup> Nu este necesară instalarea acestui environment și pe server-ul pe care se va face instalarea. Toți acești pași sunt pentru generarea folderului public din cadrul aplicației web. Doar acel folder va fi folosit pe server și implicit nu va avea nevoie de toate aceste dependențe.

Realizarea build-ului se face cu ajutorul tool-ului Gradle. Pentru generarea fișierului `.war` al platformei trebuie realizați următorii pași:

- Se vor seta detaliile build-ului în fișerul **build.gradle** (se verifică setările legate de existența comenzii *apply plugin war* și a denumirii fișierului `.war` ca *elixir-api.war*)
- Se adaugă dependența *providedRuntime 'org.springframework.boot:spring-boot-starter-tomcat'* în cazul care aceasta nu există (necesară pentru rularea cu succes pe un server)
- Se actualizează alte setări din pachetul Config al API-ului (e.g setarea unei adrese de unde se pot face request-uri către API prin modificarea setarilor din metoda *addCorsMappings* din clasa *MvcConfig*)
- Se actualizează datele de conectare la bazele de date dacă este cazul din fișierul *application.properties* din folderul *src/main/resources* (e.g date de autentificare, nume)
- Se va executa comanda **gradlew build** din folderul de root al API-ului care va genera fișierul `.war` al aplicației
- Se va extrage fișierul *elixir-api.war* generat din folderul **build/libs**

Pe serverul propriu-zis se va încărca fișierul `.war` extras. Serverul necesită instalarea și configurarea unei instanțe de Tomcat. Serviciul Tomcat va despacheta fișierul `.war` și va expune funcționalitățile API-ului după cum a fost configurat.

Configurările serviciului sunt legate de permisiunile de acces și expunerea API-ului la o anumită adresă. Adresa la care este expus API-ul trebuie să fie identică cu adresa la care se conectează aplicația web.

Legat de compatibilitatea cu accesul la sistemul de fișiere anumite setări de securitate trebuie realizate. Utilizatorul (e.g serverul de Tomcat în cazul de față) trebuie să aibă acces deplin la folderul *storage* din cadrul API-ului (*read/write*) pentru a putea salva și returna fișiere aplicației.

### Baza de date

Pentru instalarea bazei de date pe server este necesară instalarea versiunii PostgreSQL. După instalarea acestei versiuni se va realiza un utilizator nou cu un *username* și o *parolă*. Aceste date de autentificare vor fi folosite pentru conexiunea la baza de date de către API. Configurarea curentă face referire la utilizatorul *postgres* cu parola din fișierul *application.properties* al API-ului.



După configurarea serviciului PostgreSQL, se pot crea bazele de date necesare (una sau mai multe după caz) cu ajutorul script-ului de inițializare disponibil în folderul de root al platformei și anume *core.sql*.

Serviciul PostgreSQL trebuie configurat să fie expus la adresa trecut în fișierul de configurare *application.properties*. De asemenea bazele de date trebuie să fie denumite conform versiunilor definite în același fișier de configurare.

Pentru o utilizare ușoară după configurarea bazei de date în mediul de producție se poate utiliza un utilitar (e.g PgAdmin) care ajută la gestionarea bazei de date. Scriptul de inițializare poate fi rulat și din cadrul acestuia.

### Verficarea deployment-ului

Dacă toate configurările necesare au fost realizate cu succes și toate fișierele necesare au fost încărcate atunci aplicația web ar trebui să fie disponibilă la adresa configurată din serverul de Apache.

Aplicația web se conectează la API-ul expus de către serverul de Tomcat care la rândul lui comunica cu serviciul PostgreSQL care oferă acces la bazele de date. Acest proces are loc cu succes în cazul unei configurări corecte.

Aplicația Elixir<sup>23</sup> este disponibilă într-un mediu de lucru online. Toate aceste configurări au fost realizate pe un server oferit de Digital Ocean cu un sistem de operare Ubuntu 16.04 LTS instalat.

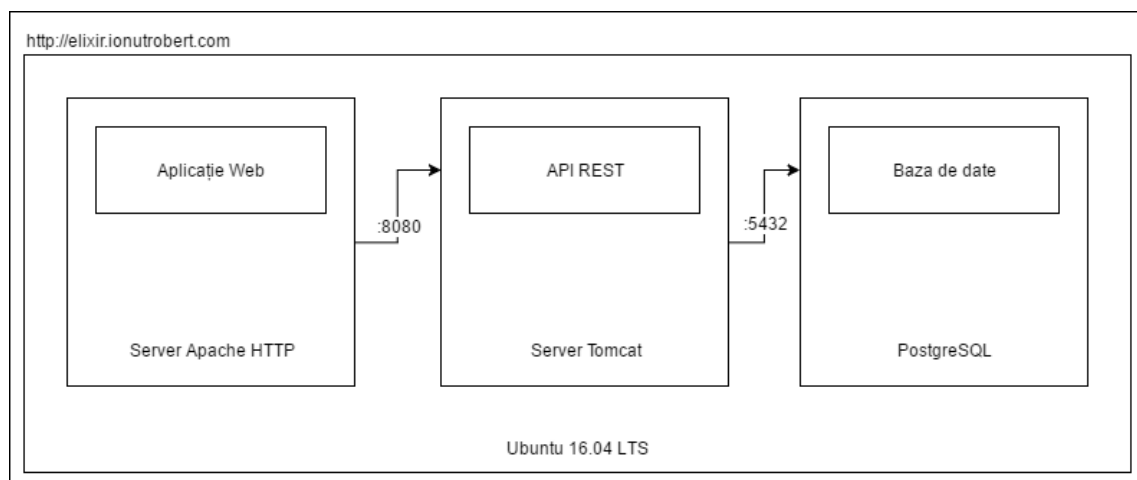


Figura 4: Diagrama de deployment a aplicației

<sup>23</sup> <http://elixir.ionutrobert.com>

# Concluziile lucrării

În această parte a lucrării de licență se regăsesc cele mai importante concluzii din lucrare, opinia personală privind rezultatele obținute în lucrare, precum și potențiale direcții viitoare de cercetare legate de tema abordată. Concluziile lucrării nu se numerotează ca și capitol.

## Scopul

În concluzie, platforma Elixir prezentată oferă un suport extins pentru activitățile academice. Astfel se ușurează procesul de notare și salvare a prezențelor la cursurile introduse în sistem.

De asemenea se facilitează accesul la fișierele încărcate de către studenți și se pun la dispoziție unelte care oferă o privire de ansamblu (e.g rapoarte) asupra situației unui curs. Mai mult toate datele din sistem sunt extensibile, acestea având posibilitatea de a fi importate sau exportate din cadrul sistemului.

Sistemul se remarcă prin ușurința în utilizare, viteza aplicației, consistența datelor și securitatea accesului la date oferite. Toate aceste aspecte au fost realizate cu ajutorul unor tehnologii moderne.

## Opinie

Consider că platforma Elixir îndeplinește cerințele riguroase ale unui sistem de management academic și oferă o suită de funcționalități foarte utile. De asemenea cred că va ușura foarte mult gestionarea activităților.

Mai mult consider că această platforma va face procesul de încărcare de fișiere mult mai ușor pentru studenți și le va oferi siguranța încărcării acestora eliminând nevoia trimiterii fișierelor prin email cu denumiri specifice. Acum toate aceste task-uri repetitive și ce pot introduce erori vor fi înlocuite de sistemul automat.

În concluzie consider că sistemul are potențialul de a fi integrat în toate cursurile unui mediu academic. Mai mult platforma este extensibilă și în viitor poate fi extinsă cu o mulțime de funcționalități noi în funcție de nevoi.

## **Direcții de dezvoltare**

Aplicația a fost dezvoltată într-un mod extensibil și facilitează dezvoltarea ulterioară a oricărei dintre componente. De asemenea aplicația are o structură modulară ce permite fiecărei componente să fie utilizată independent de celelalte. Acest fapt asigură longevitatea și extensibilitatea platformei.

Astfel se pot adăuga funcționalități noi atât în cadrul aplicației web cât și în cadrul platformei. Mai multe și baza de date are o structură extensibilă oferindu-se posibilitatea de adăugare de noi obiecte în schema acesteia.

În viitor sunt plănuite mai mult extensii (pe lângă extensiile descrise în secțiunea 1.4) care vor îmbunătăți platforma și vor adăuga cazuri de utilizare noi. Câteva dintre aceste extensii sunt:

- Adăugarea de restricții la încărcarea de fișiere după anumite variabile setate de profesori
- Setarea ponderii unei activități în media finală și setarea unor formule de calcul a acestia
- Îmbunătățirea securității platformei și includerea de coduri captcha
- Integrarea cu diferite servicii (e.g fenrir, webmail, conturi ale profesorilor)
- Integrarea cu servicii ce vor oferi reprezentări vizuale ale datelor (e.g Highcharts)
- Integrarea cu Google Calendar pentru salvarea activităților, oferirea de notificări
- Realizarea unui sistem de prezențe cu care pot interacționa și studenții
- Convertirea bazelor de date la un sistem de tip soft-delete pentru a păstra datele vechi

Varietatea extensiilor prezentate mai sus susțin potențialul platformei de a se extinde și de a deveni o unealtă utilă în mediul academic.

# Bibliografie

Pentru realizarea acestui proiect s-au folosit doar resurse cu licențe open-source de tip Apache 2.0, MIT sau similare. În bibliografie sunt incluse toate resursele folosite de către proiect și toate sursele de inspirație care au dus la materializarea acestuia.

1. **Air DatePicker**, Timofey Marochkin  
<http://t1m0n.name/air-datepicker/docs/>
2. **Angular JS Framework**, Google  
<https://docs.angularjs.org/api>
3. **Apache Commons**, Apache Software Foundation  
<https://commons.apache.org/>
4. **Bootstrap 3**, Mark Otto, Jacob Thornton (Twitter)  
<http://getbootstrap.com/>
5. **Configure Spring Boot for PostgreSQL**, John Thompson  
<https://dzone.com/articles/configuring-spring-boot-for-postgresql>
6. **Creating Database Queries With the JPA Criteria API**, Petri Kainulainen  
<https://www.petrikainulainen.net/programming/spring-framework/spring-data-jpa-tutorial-part-four-jpa-criteria-queries/>
7. **Data Mapper Jackson**, Codehaus  
<https://mvnrepository.com/artifact/org.codehaus.jackson/jackson-mapper-asl>
8. **Google Fonts**, Google  
<https://fonts.google.com/>
9. **Jade Language**, Forbes Lindesay  
<http://jadelang.net/>
10. **Java Servlet API**, GlassFish Community  
<https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api/3.1.0>
11. **Javascript Task Runner GruntJS**, Ben Alman, Kyle Robinson Young, Tyler Kellen  
<https://gruntjs.com/>

12. **Jquery**, JS Foundation  
<http://jquery.com>
13. **JSON Web Token for Java**, Les Hazlewood (Stormpath)  
<https://github.com/jwtkt/jjwt>
14. **JsPDF**, James Hall  
<https://github.com/MrRio/jsPDF>
15. **JWT authentication with Spring Web**, Sadique Ali  
<https://sdqali.in/blog/2016/07/02/jwt-authentication-with-spring-web---part-1/>
16. **Make Spring Boot application multi-tenant aware**, William Meints  
<http://fizzylogic.nl/2016/01/24/Make-your-Spring-boot-application-multi-tenant-aware-in-2-steps/>
17. **Material Icons**, Google  
<https://material.io/icons/>
18. **Model Mapper**, Jonathan Halterman  
<http://modelmapper.org/user-manual/>
19. **Multi-Tenancy using JPA, Spring and Hibernate**, Jose Manuel Garcia  
<https://dzone.com/articles/multi-tenancy-using-jpa-spring-and-hibernate-part>
20. **PostgreSQL**, PostgreSQL Global Development Group  
<https://www.postgresql.org/>
21. **PostgreSQL JDBC Driver**, PostgreSQL Global Development Group  
<https://jdbc.postgresql.org/>
22. **REST Security with JWT Spring**, Dejan Milosevic  
<https://www.toptal.com/java/rest-security-with-jwt-spring-security-and-java>
23. **Spring Boot**, Phillip Webb, Dave Syer, Josh Long  
<http://docs.spring.io/spring-boot/docs/current-SNAPSHOT/reference/htmlsingle/>
24. **Spring Framework**, Rod Johnson, Juergen Hoeller, Keith Donald  
<https://docs.spring.io/spring/docs/current/spring-framework-reference/htmlsingle/>

**25. Spring Guides, Spring Team**

<https://spring.io/guides>

**26. Spring MVC example for downloading files, Code Java**

<http://www.codejava.net/frameworks/spring/spring-mvc-sample-application-for-downloading-files>

**27. Spring Security – Password Encoding, Eugen Paraschiv**

<http://www.baeldung.com/spring-security-registration-password-encoding-bcrypt>

**28. Techniques for authentication in AngularJS applications, Gert Hengeveld**

<https://medium.com/opinionated-angularjs/techniques-for-authentication-in-angularjs-applications-7bbf0346acec>

**29. Other resources, Stackoverflow Community**

<https://stackoverflow.com/questions/41407921/eliminate-circular-json-in-java-spring-many-to-many-relationship>

<https://stackoverflow.com/questions/41726813/serializing-multiple-onetomany-collections-to-json-in-java-spring>

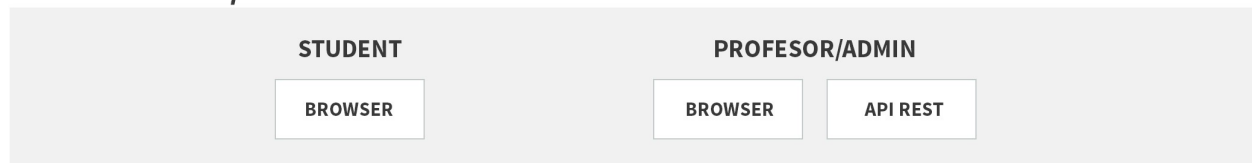
<https://stackoverflow.com/questions/42351528/java-spring-manymany-call-error-object-references-an-unsaved-transient-ins>

<https://stackoverflow.com/questions/42745285/create-object-in-many-to-one-relationship-in-spring-not-null-property-reference>

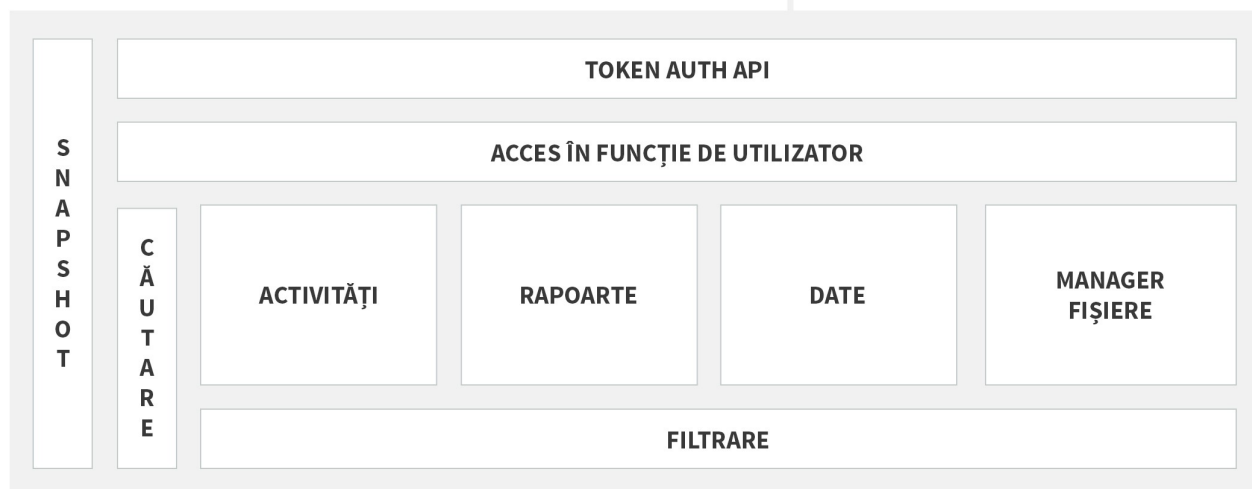
<https://stackoverflow.com/questions/43845441/java-spring-repository-delete-not-working-onetomany-relationship>

<https://stackoverflow.com/questions/14964035/how-to-export-javascript-array-info-to-csv-on-client-side>

## NIVEL - INTERFAȚĂ

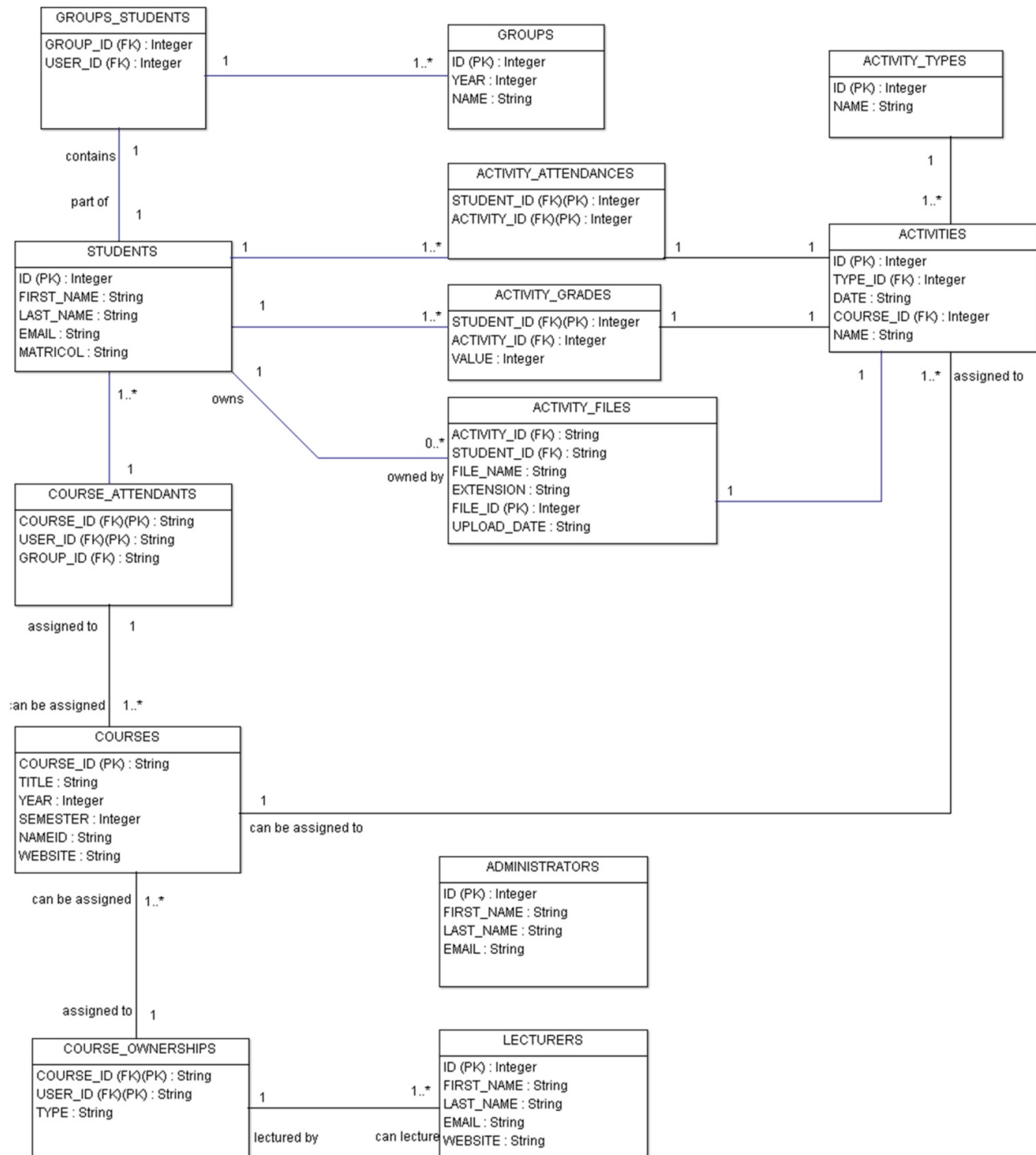


## NIVEL - PLATFORMĂ



## NIVEL - DEPOZITARE








### Anexa 3 - Exemplu de raport de prezențe generat


STUDENT \ ACTIVITATE	C1	LABORATOR 1	LABORATOR 2	LABORATOR 3	LABORATOR 4	TOTAL
Adăscăliței Anca-Luiza						1
Albișteanu Sebastian						1
Băetu Ciprian						1
Bălan Vladimir						2
Ciobanu Sebastian						0
Giosanu Eveline						0
Gordîn Ștefan						2
Iacob Robert-Ionuț						1
Jîjîe Cătălina						0
Nastasă Dan						1
Nechita Mihai						1
Popușoi Cătălin						0
Roman Ilinca						1
Stan Rareș						0
Țucăr Liana						0


## Anexa 4 – exemplu de interfață




ADMIN


GENERAL


 Tablou de bord


 Activități


 Rapoarte

ADMINISTRATIV


 Cursuri


 Grupe


 Utilizatori


 Import

CONT

 Setări

 Delogare



 > Cursuri

admin@mail.com

Cursuri

Unelte

ADAUGĂ CURS

ADAUGĂ PROFESOR LA CURS

ADAUGĂ STUDENȚI LA CURS

Caută

Q

Pagina















<

1

2

3

>

#	TITLU	AN	SEMESTRU	PROFESORI	ACȚIUNI
1	Fundamente algebrice ale informaticii	1	2	0	 
2	Probabilități și statistică	1	2	0	 
3	Programare orientată-obiect	1	2		 
4	Proiectarea algoritmilor	1	2	0	 
5	Sisteme de operare	1	2	0	 
6	Ingineria Programării	2	2	0	 
7	Introducere în criptografie	2	2	1	 
8	Modele continue și Matlab	2	2	1	