MINISTERUL EDUCAŢIEI

UNIVERSITATEA „1 DECEMBRIE 1918” DIN ALBA IULIA

SPECIALIZAREA INFORMATICĂ

FORMA DE ÎNVĂŢĂMÂNT ZI

LUCRARE DE LICENȚĂ

COORDONATOR ȘTIINȚIFIC:

CONF. UNIV. DR. ROTAR CORINA

ABSOLVENT:

COJOCARU AUGUSTIN-STELIAN

ALBA IULIA

Iulie, 2022

MINISTERUL EDUCAŢIEI

UNIVERSITATEA „1 DECEMBRIE 1918” DIN ALBA IULIA

SPECIALIZAREA INFORMATICĂ

FORMA DE ÎNVĂŢĂMÂNT ZI

„StudBud”

REȚEA DE SOCIALIZARE PENTRU STUDENȚI, PROFESORI ȘI UNIVERSITĂȚI

COORDONATOR ȘTIINȚIFIC:

CONF. UNIV. DR. ROTAR CORINA

ABSOLVENT:

COJOCARU AUGUSTIN-STELIAN

ALBA IULIA

Iulie, 2022

CUPRINS

INTRODUCERE

1.1. SCOPUL PROIECTULUI ȘI VIZIUNEA

Scopul proiectului este crearea unei rețele orientata pe studenți ce sunt în căutare de o instituție unde a-si continua studiile, pe studenți ce deja studiază în vreo instituție de învățământ sau au studiat cândva, pe profesori ce activează sau au activat în vreo instituție de învățământ, precum și pe universități ce funcționează în prezent sau au funcționat. Principalul obiectiv în crearea acestei rețele este de a rezolva problemele studenților în de-a-si alege continuarea potrivita și pe plac a studiilor. Astfel, cu ajutorul rețelei date, userii vor putea interacționa intre ei pentru a se ajuta. Pe lângă obiectivul dat, rețeaua poate fi folosita și ca un simplu instrument de comunicare și interacțiune sociala între useri, oferindu-se următoarele funcții pentru aceasta:

- Mesagerie,
- Postări,
- Recenzii către alți useri,
- Comentarii la postări și recenzii, reply-uri la alte comentarii,
- Like-uri, dislike-uri la postări, recenzii și comentarii,
- Friendship mecanism, etc.

1.2. TASK-URI PROPUSE

Principalele task-uri propuse la faza inițială de proiectare a proiectului propus sunt:

- Crearea unui site web.
- Utilizarea tehnologiilor HTML5, CSS, Javascript, SQL, PHP pentru a crea site-ul dat
- Alegerea unui mediu confortabil de programare.
- Alegerea unor framework-uri potrivite pentru fiecare din tehnologiile menționate mai sus
- Crearea în prim plan a funcțiilor specifice unei rețele de socializare.
- Crearea unei interfețe atractive, easy-to-use și intuitive pentru acest site.
- Creare unui spațiu pentru administratorul site-ului pentru a facilita managementul acestui site.

CAPITOLUL 1 REQUIREMENTS ENGINEERING

1.1. CERINȚE (REQUIREMENTS)

1.1.1. Cerințe funcționale

Tip cerință (Grup)	Cerință ID	Descrierea cerinței	Prioritate
Înregistrare și logare	**RLR_1**	Sistemul trebuie sa-i ofere user-ului un formular de înregistrare.	**High**
	RLR_2	După înregistrare, sistemul trebuie să ii trimită user-ului un email pentru confirmarea poștei electronice introduse de user la înregistrare.	**High**
	RLR_3	După apariția mesajul că email-ul a fost trimis și în caz că nu a fost recepționat niciun email de sistem, user-ului trebuie să i se ofere posibilitatea trimiterea repetată a unui email nou.	**Medium**
	RLR_4	Sistemul trebuie să îi ofere user-ului posibilitatea de logare.	**High**
	RLR_5	La logare, user-ului trebuie să i se ofere posibilitatea să rămână logat permanent sau până la expirarea token-ului.	**Low**
	RLR_6	În cazul în care user-ul uită din anumite motive parola, sistemul trebuie să îi ofere posibilitatea de a-și restabili accesul la cont prin crearea unei parole noi.	**High**
	RLR_7	În cazul în care user-ul este identificat ca administrator, sistemul trebuie să redirecționeze user-ul dat către admin panel.	**Low**
	RLR_8	Sistemul trebuie să îi ofere user-ului posibilitate de dezlogare manuală din cont.	**High**
Confidențialitatea și securitatea infromației	**CSR_1**	Accesul la admin panel trebuie să fie permis doar pentru userii cu rol de administrator.	**High**
	CSR_2	Sistemul trebuie să îi permită user-ului accesul la cont doar dacă aceste este autentificat și își are poșta electronică confirmată.	**Medium**
	CSR_3	Sistemul trebuie să restricționeze accesul user-ilor la paginile user-ilor pentru care sunt blocați.	**Medium**
	CSR_4	Sistemul trebuie să restricționeze altor useri sa publice postări pe păginile străini (de nu încurcat cu recenziile).	**Medium**
	CSR_5	Sistemul trebuie să nu permită user-ilor să aibă posibilitatea ștergerii sau editării postărilor, recenziilor, mesajelor, comentariilor, like-uri și dislike-urilor lăsate de alți useri.	**Medium**

	CSR_6	In admin panel, acțiunilor de setare activ/inactiv, adaugă/scoate rolul de admin, ștergere către alți utilizatori trebuie sa fie permise pentru user doar atunci, dacă user-ul dat are rolul mai mare față de utilizatorii către care acționează.	**Medium**
	CSR_7	În cazul acțiunilor de ștergere sau acțiunilor administratorilor asupra altor utilizatori, sistemul trebuie să însoțească aceste acțiuni cu o confirmare repetată de către utilizator a acestor acțiuni.	**Low**
	CSR_8	În cazul utilizării tabelei pentru gestionarea utilizatorilor din admin panel de către un anumi adminstrator, sistemul trebuie să omită pe lista din aceast tabelă adminstratorul dat.	**Low**
Vizularea informației și content-ului	**CIR_1**	După înregistrare, sistemul trebuie să creeze o pagină pentru user vizibilă și pentru ceilalți useri.	**High**
	CIR_2	Persoanele care au lăsat like sau dislike trebuie să poată fi văzuți în forma de o listă.	**Low**
	CIR_3	Fiecare conversație din chat trebuie să ofere posibilitatea vizionării sub formă de galerie a tuturor pozelor din conversație dată.	**Low**
Editarea/ștergerea informației	**EDR_1**	Sistemul trebuie să ofere utilizatorului posibilitatea editării recenziilor și postărilor lăsate de el.	**Medium**
	EDR_2	Sistemul trebuie să ofere utilizatorilor posibilitatea de ștergere a propriilor: comentarii, postări, like-uri, dislike-uri, recenzii.	**Medium**
	EDR_3	Sistemul trebuie să ofere utilizatorului posibilitatea editării contului personal.	**Medium**
Pentru platforma de administrator	**PAR_1**	Sistemul trebuie să dețină un admin panel.	**High**
	PAR_2	Sistemul trebuie să dețină un tabel pentru gestionarea user-ilor de către administrator.	**Medium**
	PAR_3	Sistemul trebuie să deține un tabel pentru gestionarea email-urilor trimise de către utilizatori către echipa de administratori.	**Low**
User Experience	**UXR_1**	Sistemul trebuie sa fie dotat cu un formular de contact ce poate fi utilizat de către toți utilizatorii site-ului.	**Medium**
	UXR_2	Sistemul trebuie să ofere user-ului un mecanism de căutare a altor useri însoțit cu un filtru.	**Medium**
	UXR_3	Sistemul trebuie să permită user-ului logat să realizeze postări pe pagina sa.	**High**
	UXR_4	Sistemul trebuie să permită user-ilor să realizeze scrierea unor recenzii pe paginile altor useri doar atunci, dacă nu sunt blocați pentru userii pentru care lasă recenziile.	**High**
	UXR_5	Sistemul trebuie să-i ofere user-ilor un chat pentru comunicarea între ei.	**High**

	UXR_6	Sistemul trebuie să permită utilizatorilor posibilitatea lăsării unor comentarii, like-uri, dislike-uri, note (doar în cazul recenziilor) la postările și recenziile la care au accesul.	**High**
	UXR_7	Sistemul trebuie să ofere posibilitatea utilizatorilor de a răspunde la comentariile la care au aceștia accesul.	**Medium**
	UXR_8	Chat-ul trebuie să fie însoțit cu o listă a persoanelor cu care user-ul are conversații sau încearcă la moment să creeze.	**High**
	UXR_9	Sistemul trebuie să-i ofere utilizatorului logat posibilitatea notificării acestuia dacă acesta primește un mesaj de la alt utilizator, dacă cineva îi raspunde la vreun comentariu. dacă cineva îi oferă prietenia, dacă cineva îi comentează postarea, recenzia.	**Medium**
	UXR_10	Sistemul trebuie să dețină un mecanism de friendship pentru a-i oferi user-ilor posibilitatea de gestionare a prietenilor săi.	**High**
	UXR_11	Sistemul trebuie să permită utilizatorilor să se împărtășească cu alții cu tot felul de fișiere office (xls, doc, pdf) și să dețină o pagină specială doar pentru fișiere.	**Medium**
	UXR_12	Sistemul trebuie să permită utilizatorilor să poată crea compartimente pentru fișierele publicate pentru a le sorta în modul dorit.	**Low**

Table 1-1 Lista cu cerințele funcționale

1.1.2. Procesul de proritizare a cerințelor funcționale

Am folosit metoda Three-Level Scale de prioritizare pentru cerințele noastre urmând următorul principiu:

- **High** - sunt cerințele cheie ale sistemului nostru și trebuie să fie puse în aplicare obligatoriu.
- **Medium** - sistemul poate exista fără acestea, dar avem nevoie de aceste funcționalități pentru securitate și alte caracteristici importante pe care să le oferim utilizatorilor.
- **Low** - aceste funcționalități nu sunt vitale pentru sistemul nostru, dar pot fi implementate ulterior pentru o experiență mai bună a utilizatorului.

1.1.3. Cerințe non-funcționale (lista)

1. Retrimiterea cererii de confirmare a email-ului sau a cererii de resetare a parolei trebuie se realizeze cu cel mult 6 cereri pe minut.
2. Timpul de sincronizare a informației cu baza de date în chat trebuie să fie de cel mult 1 secundă.
3. Informația din tabele din admin panel trebuie să fie aranjată în maxim 10 rânduri pe o pagină.
4. În cazul existenței de erori la validarea datelor din pagina de înregistrare sau pagina de contact, în momentul procesului de înregistrare sau de transmitere a unui mesaj de contact efectuat de un anumit utilizator, aceste erori trebuie să se afișeze după cel puțin un click pe butonul submit.
5. Imediat după transmiterea unui mesaj prin formularul de contact, trebuie să se afișeze un mesaj precum că acest mesaj a fost trimis, în caz contrar să fie afișat un semn de eroare.
6. La orice eroare ce apare, sistemul trebuie să înștiințeze imediat utilizatorul cu aceste erori prin semne corespunzătoare (pot fi animații sau afișări text).
7. Sistemul trebuie să permită încărcarea de maxim 10 fișiere per postare sau recenzie și 5 per mesaj în chat.
8. Numărul de caractere nu trebuie să depășească: 500 pentru mesaj din postare/recenzie/forma de contact, 100 pentru descrierea compartimentului creat pentru fișiere, 20 pentru nume și prenume fiecare în parte.
9. Numărul de caractere trebuie să fie minim de: 4 pentru nume, prenume și parola fiecare în parte.

1.1.4. Resurse necesare (cerințe tehnice)

❖ Cerințe minime pentru hardware:

- Conexiune la internet cu o viteză de 16 Mbits/s.
- Procesor: Pentium 4 (sau echivalent)
- 2 GB RAM
- Hard disk space: 50 GB
- Tastatură și mouse/touchpad

❖ Cerințe hardware recomandabile:

- Conexiune la internet cu o viteză de 40 Mbits/s.
- Procesor: Intel i5 (sau echivalent)
- 4 GB RAM
- Hard disk space: 100 GB
- Tastatură și mouse/touchpad

❖ Cerințe software:

- Sistem de operare: Windows 10 version 1507 sau mai nou.
- Mediu de programare VSCode sau PHP storm (https://code.visualstudio.com/, respectiv https://www.jetbrains.com/phpstorm/).
- Node.js (https://nodejs.org/en/).
- Github (https://github.com/) – pentru controlul versiunilor și sincronizarea proiectului cu un cloud.
- Laravel versiunea 9x (https://laravel.com/docs/9.x/releases)
- Vue.js versiunea 2x (https://vuejs.org/)
- XAMPP (https://www.apachefriends.org/download.html)
- Browser cu suport de internet: Chrome – versiunea actuală, Mozilla Firefox – versiunea actuală.

1.2. DIAGRAME: USE-CASE, CONTEXTUALĂ, DE CLASE

1.2.1. Diagrama use-case

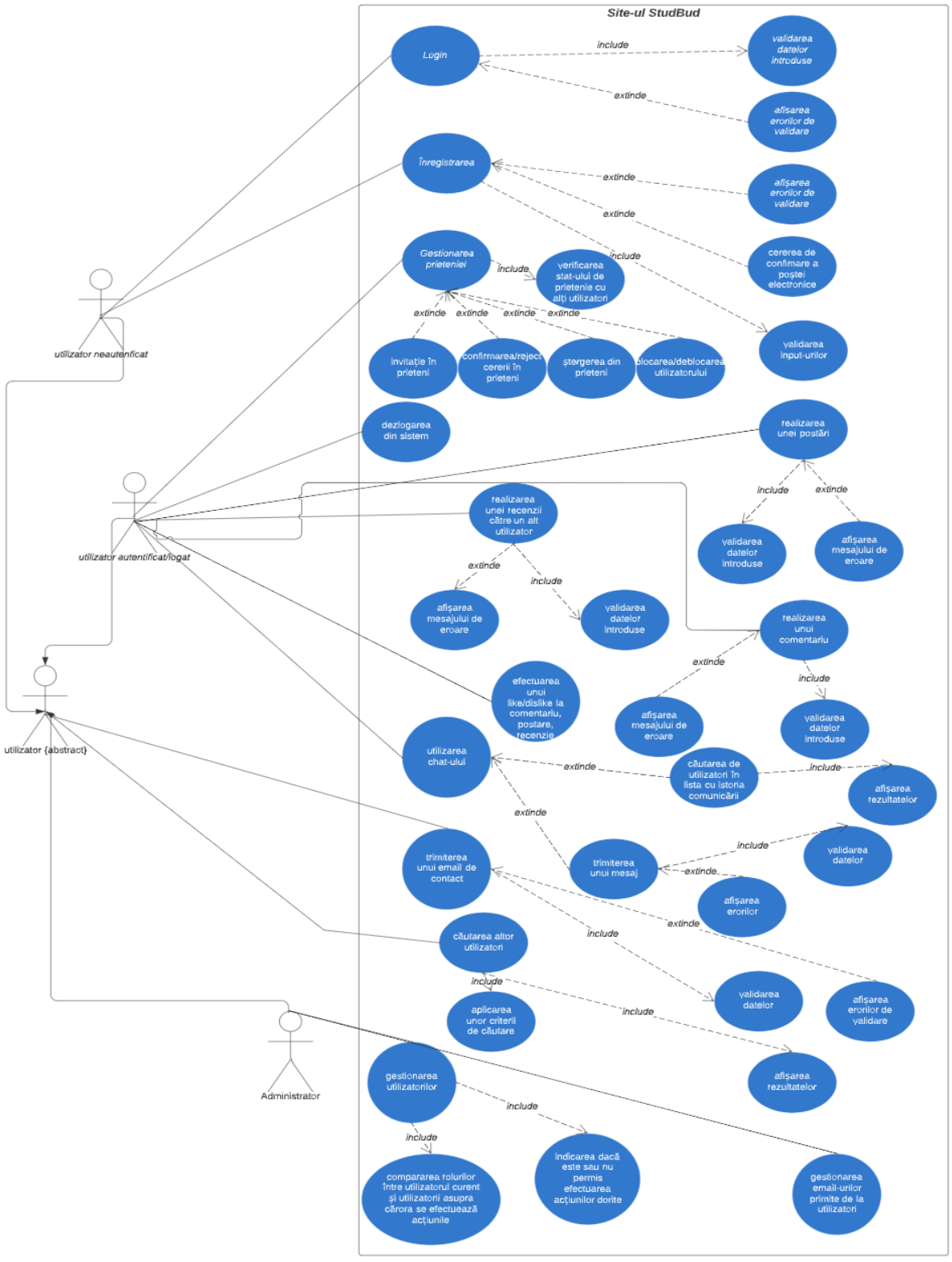


Figure 1-1 Diagrama use-case

1.2.2. Diagama contextuală

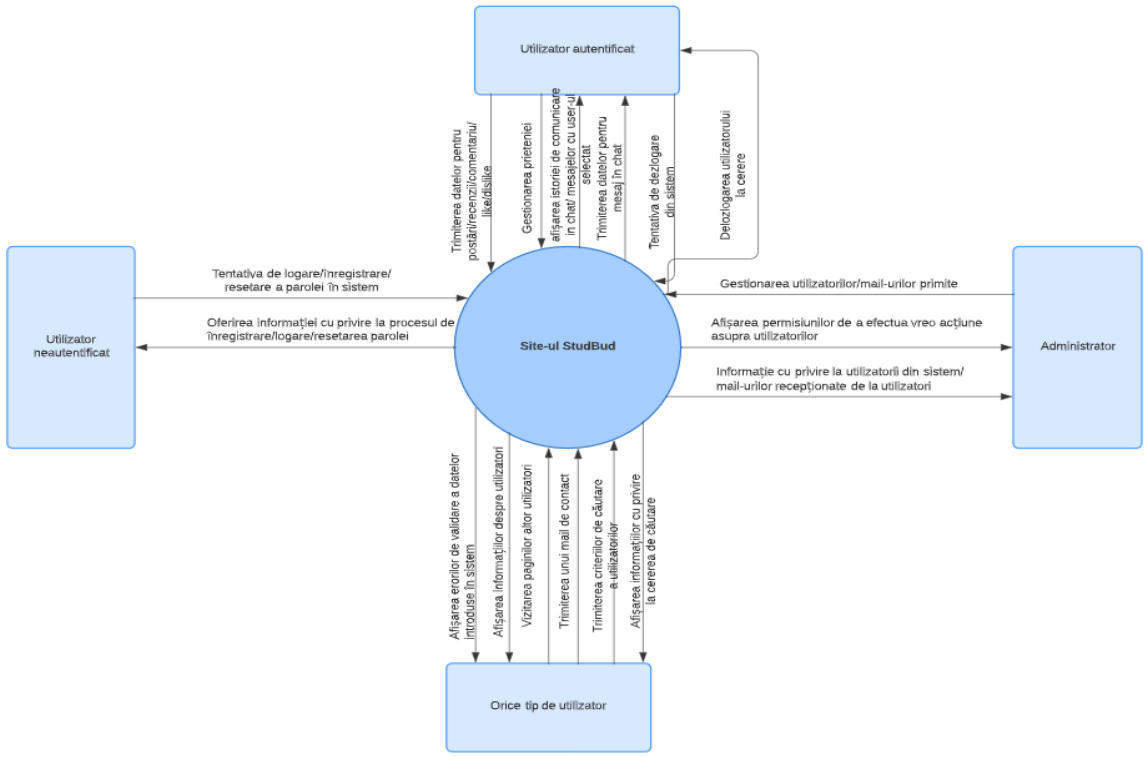


Figure 1-2 Diagrama contextuală

1.2.3. Diagrama de clase – diagrama modelului bazei de date

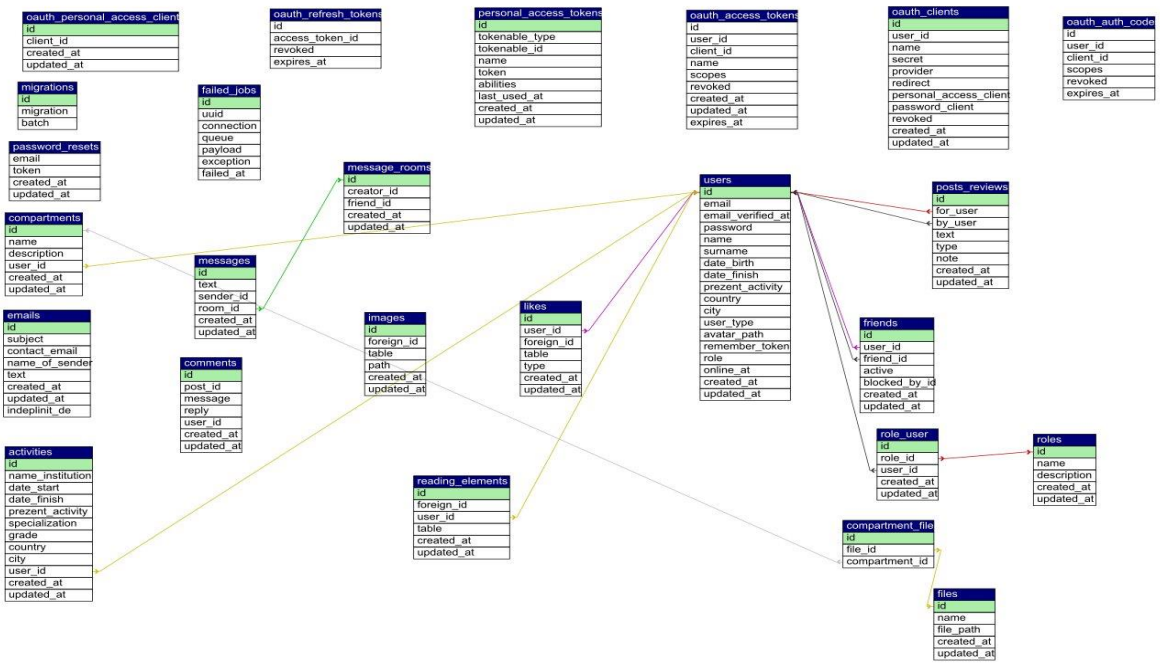


Figure 1-3 Diagrama de clase pentru proiectul nostru

Avem 25 de tabele, din care: 5 sunt opționale – pentru *OAuth* – care se refera la Laravel Passport, un mecanism de securitate pentru request-uri de date din API, poate fi inlocuit cu sanctum și în acest caz aceste tabele pot fi omise, însă daca dorim aplicarea unui mecanism de securitate mai puternic ca Laravel Passport – pot fi utilizate; 1 care nu poate fi modificata, nici stearsa – migrations - vine din stock-ul Laravel-ului, necesar pentru funcționarea corecta a acestuia; failed-jobs - este o tabela ce vine din stock cu Laravel, de asemenea necesara pentru funcționarea corecta a acestuia; celelalte 18 tabele sunt tabelele de care se folosesc propriu-zis serviciile site-ului nostru pentru gestiunea datelor.

Tabelele comments și images nu au o relație cu alte tabele, deși se pare ca ar trebui sa aibă, din motiv ca acestea au ca rol stocarea de date din mai multe tabele, având o relație polimorfică. Celelalte tabele lipsite de relații cu alte tabele au fost instalate așa din stock-ul Laravel-ului, respectiv neavând necesitatea de a fi modificate pentru un lucru corect a framework-ului.

CAPITOLUL 2 DESCRIEREA TEHNOLOGIILOR UTILIZATE

2.1. GENERALIZARE

Ca mediu de programare s-a utilizat Visual Studio. De ce? Pentru că:

- Este gratis;
- Suporta mai multe limbaje de programare;
- Automat identifica limbajul de programare din fișier, dar utilizatorul poate și el sa schimbe limbajul corespunzător fișierului în care lucrează;
- Integrare git încorporată;
- IntelliSense;
- Posibilitatea instalarii unui număr mare de extensii;
- Deține o indexare bună a spațiului de lucru.

Inițial elaborarea site-ului Studbud era orientata pe JQuery și PHP nativ, însă au apărut mari dificultăți în realizarea unor funcții mai complexe, și respectiv o mare complexitate în munca din cauza numărului mare de funcții ce necesitau o efectuare fără refresh la pagina (partea de Front-end) și funcțiilor complexe de pe Back-end. Realizarea La partea de Back-end, din cauza unui număr mare de tabele SQL, se necesita instrumente ce ar ușura gestiunea informației din tabele, și mai mult în cazul în care tabelele au o relație intre ele. De asemenea pe partea de securitatea ce necesita multă muncă, și anume îndeplinirea anumitor funcții în dependență de tipul user-ului, ce permisiuni are acesta asupra site-ului etc. Evident, utilizarea PHP nativ simplifică complexitatea realizării acestor chestii cu mult.

Pentru problema de Front-end s-a ajuns la alegerea unui framework pentru reactivitatea site-ului nostru și o alegere s-a făcut pentru Vue JS, care pare a fi foarte satisfăcător și foarte flexibil fata de concurenții săi asemănători (React sau Angular). Utilizarea JQuery-ului în cazul nostru necesita după fiecare gestiune a informației din Back-end un upload de cod HTML pentru realizarea unor efecte după efectuarea acestor funcții, ce nu este foarte comod în cazul unui număr mare de funcții, plus. Deci, cea mai buna alegere este utilizarea anumitor state-uri, variabile, ce ar permite rendering-ul anumitor efecte în baza de informația din Back-end (primita de asemenea în forma de variabile). Vue js-ul de asemenea ne pune la dispoziție și instrumente ca MIXINS, care ne permite reutilizarea unui cod JS în cazurile în care se necesita aceeași soluție de rezolvare. Totodată Vue JS ne pune la dispoziție numeroase directive pentru diferite situații, care reprezintă niște componente cu soluții deja gata (exemple: tabele de gestiune a informației pentru admin panel, hover directive etc).

Deci, în realizarea site-ului nostru ne-am axat:

Pentru mediul de dezvoltare:

- Visual Studio Code

- *Pentru partea de Front-end:*
  - CSS – Bootstrap
  - Javascript – Vue js, JQuery
- *Pentru partea de Back-end:*
  - PHP – Laravel
- *Pentru baza de date:*
  - Localhost – XAMPP
  - Sistem de gestiune a bazei de date - MySQL

2.2. PARTEA DE FRONT-END

2.2.1. HTML & CSS

Pentru HTML am utilizat versiunea 5 a acestuia (HTML 5), care este cea mai actuala.

Aici s-a ales librăria Bootstrap (versiunea 5.1), deoarece este gratis și conține multe componente necesare site-ului pe care l-am ales sa-l implementam. Plus la aceasta, Vue JS-ul, care s-a ales ca framework javascript pentru Front-end, conține și el componente bootstrap făcute sub tehnologia Vue JS, majoritatea fiind mult mai simplu de utilizat în Vue JS decât în versiunea originala clase de CSS. De asemenea, utilizarea librăriei Bootstrap simplifica mult lucrul cu partea responsive a site-ului.

2.2.2. JavaScript

Pentru o interfață easy-to-use este nevoie de o reactivitate intre acțiunile user-ului și informația ce se renderează pe web-site. Pentru reactivitate s-a ales sa se folosească Vue JS (versiunea 2).

Adaptabilitatea Vue.js este ceea ce atrage în primul rând dezvoltatorii și face ca lucrul cu acesta să fie plăcut și mai eficient. Vue.js oferă o interfață de utilizator mai bună și o integrare ușoară a aplicațiilor.

Pentru a facilita înțelegerea framework-ului, iată câteva dintre cele mai importante avantaje ale utilizării acestuia ca platformă de dezvoltare:

- Framework-ul Vue.js se bazează pe șablonul MVVM (Model-View-ViewModel), derivat din șablonul Clasic Model-View-Controller (MVC). Aspectul său a contribuit la separarea dezvoltării interfeței interfeței grafice a utilizatorului și a logicii interne de business, ceea ce crește semnificativ eficiența creării aplicațiilor. Nucleul MVVM este un strat ViewModel, oarecum similar cu un convertor de valoare responsabil pentru schimbarea obiectelor de date din model.
- Ușor de utilizat și integrare ușoară. Acesta este unul dintre instrumentele cele mai prietenoase pentru începători, cu o curbă de învățare scăzută în comparație cu alte framework-uri. De asemenea, va fi intuitiv pentru dezvoltatorii care au lucrat anterior cu jQuery și Angular. De asemenea, este ușor să îl integrați în alte biblioteci sau să îl utilizați ca proiect separat.
- Se compune din componente. Componenta este o parte importantă a Vue.js. Acestea sunt blocuri de cod care pot fi utilizate în mod repetat, incluzând atât o descriere a interfeței aplicației, cât și implementarea capabilităților sale. Dezvoltatorul are nevoie de ele pentru a crea o bază de cod modulară ușor de întreținut pentru crearea de programe la scară largă.
- Eficiență. Vue.js este un frameork mic și rapid din punct de vedere al performanței, ceea ce îl face unul dintre cele mai importante instrumente pentru dezvoltarea aplicațiilor mobile.
- Legarea reactivă a datelor. De fiecare dată când datele se schimbă, DOM este actualizat automat.
- Dom Virtual. Lumea Vue.js se învârte în jurul DOM. De fapt, servește ca o interfață între scripturi și marcare, ceea ce facilitează trimiterea și primirea datelor din HTML. La schimbarea elementelor CSS, în unele cazuri este necesară reconstruirea întregului arbore. Vue.js creează o copie în Dom virtual, nu în Dom browser-ul. Prin eliminarea nevoii de duplicare a informațiilor, productivitatea generală crește.

Principalul element cheie în cadrul Vue.js este componenta. O componentă este închisă, reutilizabilă, cel mai adesea singura parte a logicii interfeței cu utilizatorul.

În ciuda faptului că folosim expresia "componentă Vue" aici, este important să știm că componentele nu sunt o definiție specifică pentru acest framework special. Putem vedea aceeași structură în React și în framework-ul Svelte.

Abordarea "componentă" este o modalitate de a structura dezvoltarea frontend fără a umfla foarte mult proiectele și a economisi timp prețios. În centrul majorității framework-urilor

cu componente se află reactivitatea, care contribuie la experiența utilizatorului fără a compromite dezvoltatorul. Fiecare componentă are propria stare, marcare și stil, permițând programatorului să creeze aplicații organizate într-un arbore.

Componentele au întotdeauna un nume specific. Aceasta este ceea ce compilatorul Vue caută pentru a crea și monta instanțele sale. Pentru a utiliza o componentă într-un șablon, pur și simplu creați o etichetă HTML cu numele său în paranteze triunghiulare.

Până în prezent, există multe soluții (componente) gata făcute care pot fi utilizate pentru integrarea lor nedureroasă în orice proiect.

Când creați o componentă în fișierul vue, la nivelul de bază veți găsi trei secțiuni pentru introducerea codului:

- **Șablon (șablon).**

Scris într-o versiune extinsă a limbajului șablonului (HTML), acesta servește ca directivă. Acestea sunt un fel de reguli pentru crearea marcajului final al unei componente pe baza stării sale interne.

- **Script (script).**

Reprezintă logica aplicației. Această secțiune include:

Caracteristici. Un set de variabile de intrare utilizate pentru a configura comportamentul componentei. Acestea pot fi filtre de date, ordine de sortare, etichete, comutatoare de vizibilitate etc.

Stare. Acest lucru este opțional, dar este adesea prezent. Aceasta este o structură de date care asigură starea componentei la un moment dat. Se va schimba în timp, în funcție de evenimentele care au loc.

- **Styles.**

Aici sunt plasate style-urile CSS.

Pe alocuri s-a utilizat și librăria jQuery, și anume acolo unde se necesita lucrul cu CSS sau cu unele lucruri pentru care nu exista un echivalent în Vue JS iar scrierea lor în Vanilla JS era sa fie cu mult mai complexa decât echivalentul lor în jQuery (spre ex. adăugarea unei clase unui grup de elemente, sau selectarea unui element după denumirea clasei, id-ului etc.).

2.3. PARTEA DE BACK-END

Laravel este un proiect 100% open source. Acest lucru deschide oportunități excelente de personalizare, modificare și extindere, iar aici Laravel depășește multe framework-uri.

Laravel folosește biblioteci partajate cu Symfony. Aceste framework-uri urmează cele mai bune practici de dezvoltare și proiectare. Prin urmare, aceste proiecte sunt compatibile. Acest lucru este deosebit de important pentru software-ul de întreprindere, deoarece vă permite să mențineți o bază de cod curată, minimalistă și eficientă, ușor de modificat.

Aplicațiile Laravel oferă performanțe mai mari în comparație cu aplicațiile create folosind alte framework-uri. Acest lucru este posibil și datorită sistemului de cache. Driverul de cache a fișierelor stochează multe elemente în sistemul de fișiere. Acest lucru vă permite să dezvoltați rapid aplicațiile.

Laravel asigură securitatea aplicațiilor. Baza de cod a cadrului este protejată de amenințări, de exemplu, injecții SQL sau falsificarea interogării între site-uri (CSRF). Acest lucru protejează utilizatorii de pierderea datelor importante.

Iată câteva caracteristici tehnice care fac din Laravel cel mai bun cadru PHP:

- Built-in Eloquent ORM. Acest sistem vă permite să lucrați cu diferite baze de date utilizând implementarea șablonului ActiveRecord. Datorită Eloquent ORM, puteți lucra cu baze de date fără a fi nevoie să scrieți interogări SQL complexe.
- Șabloane Blade (Șabloane Blade). Laravel susține designul arhitectural MVC. Vă permite să separați interfața cu utilizatorul și logica de business. Laravel acceptă, de asemenea, motorul Blade template, care vă permite să utilizați codul PHP nativ.
- Viteză mare de dezvoltare. Laravel vă permite să creați rapid aplicații, deoarece dezvoltatorii pot face fără cod complex în acest proces. După cum sa menționat mai sus, framework-ul este construit pe arhitectura MVC, ceea ce oferă acces la toată infrastructura necesară pentru crearea unui site web și economisește timp.
- Gestionarea eficientă a traficului. Dacă site-ul este popular, aplicația trebuie să proceseze un număr mare de solicitări în fiecare secundă. Acest lucru crește încărcarea pe server și costul găzduirii. Mai mult, datorită încărcării mari, serverul poate să nu mai răspundă, ceea ce duce uneori la pierderea datelor. Cu Laravel, astfel de riscuri sunt reduse la minimum, deoarece framework-ul implementează un sistem unic de așteptare a mesajelor. Acest sistem echilibrează sarcina pe server, ceea ce asigură funcționarea neîntreruptă și securitatea datelor.

CAPITOLUL 3 DEZVOLTAREA APLICAȚIEI

3.1. GENERALIZARE

Până a începe a descrie procesul de lucru cu tehnologiile și instrumentele utilizate în proiectul nostru, ar fi bine de menționat procesul inițial de instalare a acestor instrumente (se presupune ca XAMPP și PHP cea mai noua versiune este deja instalat).

Pentru a putea mediul Laravel pentru aplicația noastră, este nevoie inițial de instalat Composer. Procesul de instalare a acestuia se poate găsi pe: *https://getcomposer.org/download/*. După instalarea acestuia putem recurge la instalarea proiectului nostru în Laravel cu următoarele comenzi în terminalul mediului nostru de programare VSCode:

1. composer global require laravel/installer
2. laravel new denumirea-aplicatiei (cazul nostru StudBud)
3. cd denumirea-aplicației (cazul nostru StudBud)
4. php artisan serve – pentru deschiderea serverului

După s-a creat în folderul views deja prezent în proiectul Laravel un alt folder - layouts, în care s-a creat un fișier cu denumirea app.blade.php. Aici am creat un layout comun pentru toate paginile ulterioare ale site-ului în head-ul căruia vom atașa ulterior link-urile pentru instalarea celorlalte instrumente ce le-am folosit în proiect.

Următorul instrument pe care l-am instalat este Vue.js. Pentru instalarea acestuia, tot în folderul cu proiectul în terminalul VSCode, este nevoie de introdus următoarele comenzi (se presupune ca pe calculator este deja instalat node.js):

1. $ composer require laravel/ui
2. $ php artisan ui vue
3. $ npm install && npm run dev

După aceste comenzi, în head-ul menționat mai sus introducem următorul cod:

```
<script defer src="{{ mix('js/app.js') }}"></script>
```

Ulterior, ca pentru orice cod nou scris în Vue.js, acțiunea acestuia sa se observe în timp real, este nevoie de introdus tot în terminal următorul cod - *npm run watch.*

In lucrul cu Vue.js am avut nevoie de instalarea altor sub-instrumente oferite de aceasta librărie, și anume de Vuelidate, care este un instrument pentru validarea informației din input-uri ce o introducem și de bootstrap-ul realizat sub Vue. Pentru instalarea acestora, tot în terminal introducem următoarele comenzi:

1. *npm install @vue/composition-api* – instalarea composition – necesara pentru instalarea vuelidate-ului, în cazul daca folosim versiunea 2 de Vue.js.

2. *npm install @vuelidate/core @vuelidate/validator.*
3. *npm install vue bootstrap bootstrap-vue* – instalarea bootstrapVue.

După instalările date, în fișierul app.js din folderul resources (creat implicit cu instalarea proiectului Laravel) este nevoie sa încărcăm dependențele (sau altfel spus – înregistrarea acestor librarii în aplicația noastră) pentru librăriile instalate pentru a le putea utiliza global:

```
import Vuelidate from 'vuelidate';
import VueCompositionAPI from '@vue/composition-api';
// bootstrap import
import { BootstrapVue, IconsPlugin } from 'bootstrap-vue';
import 'bootstrap/dist/css/bootstrap.css'
import 'bootstrap-vue/dist/bootstrap-vue.css'
import PortalVue from 'portal-vue'
// vuelidate
Vue.use(VueCompositionAPI)
Vue.use(Vuelidate)
// bootstrap
Vue.use(BootstrapVue)
Vue.use(IconsPlugin)
Vue.use(PortalVue)
```

Fragment de cod 3-1 Încărcarea dependențelor în fisierul app.js

După cum s-a menționat anterior, am utilizat librăria Bootstrap pentru CSS. S-a folosit conectarea Bootstrap-ului prin amplasarea link-urilor CDN la head-ul site-ului. Librăria Bootstrap necesită conectarea mai multor link-uri pentru a face funcțională utilizarea acesteea. Astfel au fost conectate următoarele link-uri (conectarea Bootstrap-ului pentru funcționarea sa deplina necesita și conectarea librăriei jQuery):

```
<link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css
" rel="stylesheet"
    integrity="sha384-
1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jIW3"
crossorigin="anonymous">
```

Fragment de cod 3-2 CDN pentru CSS

```
<script src="https://code.jQuery.com/jQuery-3.6.0.min.js"
    integrity="sha256-/xUj+3OJU5yExlq6GSYGSHk7tPXikynS7ogEvDej/m4="
crossorigin="anonymous"></script>
<script
src="https://cdn.jsdelivr.net/npm/popper.js@1.12.9/dist/umd/popper.min.js"
    integrity="sha384-
ApNbgh9B+Y1QKtv3Rn7W3mgPxhU9K/ScQsAP7hUibX39j7fakFPskvXusvfa0b4Q"
    crossorigin="anonymous"></script>
<script
src="https://cdn.jsdelivr.net/npm/bootstrap@4.0.0/dist/js/bootstrap.min.js"
    integrity="sha384-
JZR6Spejh4U02d8jOt6vLEHfe/JQGiRRSQQxSfFWpi1MquVdAyjUar5+76PVCmYl"
    crossorigin="anonymous"></script>
<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.mi
n.js"
    integrity="sha384-
ka7Sk0Gln4gmtz2MlQnikT1wXgYsOg+OMhuP+IlRH9sENBO0LRn5q+8nbTov4+1p"
    crossorigin="anonymous"></script>
```

```
<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js"
    integrity="sha384-ka7Sk0Gln4gmtz2MlQnikT1wXgYsOg+OMhuP+IlRH9sENBO0LRn5q+8nbTov4+1p"
    crossorigin="anonymous"></script>
```

Fragment de cod 3-3 CDN conectare pentru librariile js:JQuery, Popper, Bootstrap si, respectiv Bundle

3.2. FRONT-END

3.2.1. CSS

In cadrul realizării CSS pentru site-ul nostru am utilizat librăria Bootstrap. Utilizarea Bootstrap-ului consta în atribuirea unor clase ce conțin deja niste stiluri prestabilite de librărie, pentru diverse tag-uri HTML în scopul de a stiliza pagina web.

In cadrul site-ului nostru s-a utilizat mult sistemul grid oferit de Bootstrap. Plusul mare a acestui sistem este ca ne oferă și posibilitatea de a face concomitent și responsive site-ul ce-l cream, conținând clase pentru 5 diverse mărimi ale ecranului:

	Extra small <576px	**Small** ≥576px	**Medium** ≥768px	**Large** ≥992px	**Extra large** ≥1200px
Max container width	None (auto)	540px	720px	960px	1140px
Class prefix	.col-	.col-sm-	.col-md-	.col-lg-	.col-xl-
# of columns	12				
Gutter width	1.5rem (.75rem on left and right)				
Custom gutters	Yes				
Nestable	Yes				
Column ordering	Yes				

Table 3-1 Descrierea succinta a sistemului grid oferit de Bootstrap
Sursa: (Bootstrap, 2021)

```
<div class="row">
    <div class="col-12 col-md-8 col-lg-12 col-xl-8 left-wrapper"> ... </div>
</div>
```

Fragment de cod 3-4 Exemplu de utilizare a claselor pentru sistemul grid

3.2.2. JavaScript

După cum s-a menționat anterior, în proiectul dat partea cea mai majora în JS il ocupa framework-ul Vue.js.

In cadrul lucrului cu Vue.js în proiectul nostru vom menționa despre urmatoarele aspecte:

- *Directivele built-in în Vue.js*
-
- *Înregistrarea componentelor și reutilizarea acestora în Vue.***js**
- *Validarea input-urilor cu Vue.js*
- *Provide/inject*
- *Utilizarea axios în Vue.js*
- *Lucrul cu JQuery*

- **Lucrul cu partea scriptului Vue.js-ului**

În cadrul scriptului Vue ne-am folosit de următoarele proprietăți ale componentului exportat Vue oferite din cutie de Vue.js:

- Methods
- Props
- Components
- Mixins
- Data
- Proprietatea computed
- Watch

Lifecycle Hooks - Fiecare instanță componentă Vue trece printr-o serie de pași de inițializare atunci când este creată - de exemplu, trebuie să configureze observarea datelor, să compileze șablonul, să monteze instanța la DOM și să actualizeze DOM atunci când datele se schimbă. Pe parcurs, rulează și funcții numite lifecycle hooks, oferind utilizatorilor posibilitatea de a-și adăuga propriul cod în anumite etape ***(Error! Reference source not found.)***.

O descriere succintă pentru fiecare Hook se poate citi pe următorul link:

https://vuejs.org/api/options-lifecycle.html.

Methods – este un obiect asociat cu instanța Vue. Funcțiile sunt definite în interiorul obiectului *Methods*. Methods sunt utile atunci când avem nevoie pentru a efectua unele acțiuni cu directiva *v-on* pe un element ca sa se ocupe de evenimente. Funcțiile definite în interiorul obiectului metode pot fi solicitate în continuare pentru efectuarea acțiunilor sale.

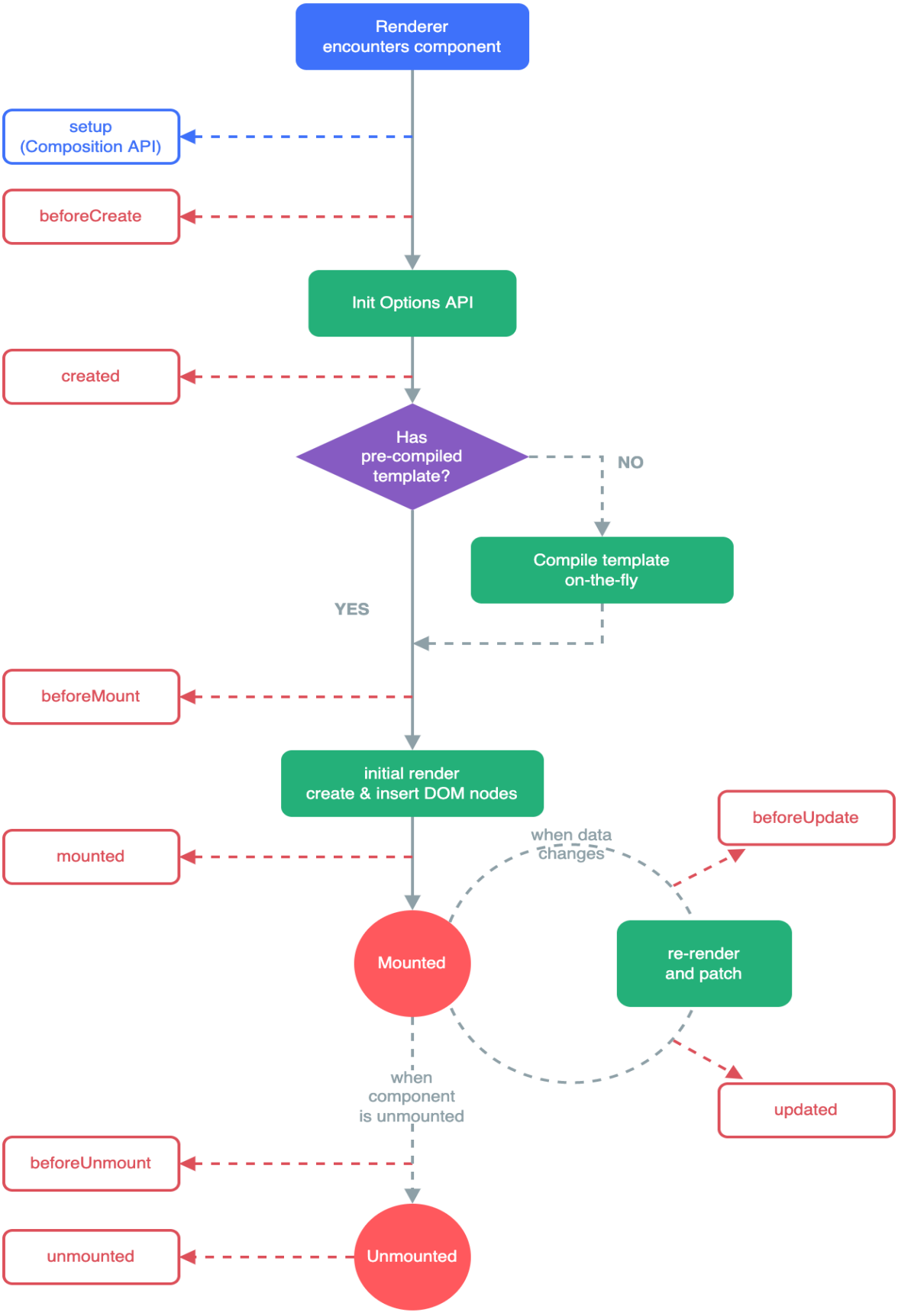
Props – este similar argumentelor dintr-o funcție. Variabilele introduse în obiectul props vor reprezenta niște variabile ce se așteaptă a fi trimise din componenta părinte de unde se va apela componenta curenta (ce deține props).

Components – reprezintă un obiect ce va include în el denumirile componentelor înregistrate local în componenta curenta.

Mixins – este similar obiectului *components,* doar ca reprezintă un array cu denumirile mixin-urilor înregistrate.

Mixins este un instrument flexibil pentru reutilizarea codului în componentele Vue. Obiectul mixin poate conține orice opțiuni corespunzatoare unui component Vue (data, watch, props etc.). Când se utilizează o componentă de mixin, toate opțiunile mixin-ului sunt "amestecate" cu opțiunile proprii ale componentei.

Data – este un obiect ce returnează toate state-urile sau variabilele ce pot fi supuse unor modificări în cadrul lucrului cu componenta curentă.



Img 3-1 Diagrama ciclului de viata pentru un component
Sursa: (Vue.js, Lifecycle Hooks | Vue.js, 2020)

Proprietatea **computed** - este asemenea unei funcții, doar ca fără argumente/parametri (nu este recomandabil utilizarea unor argumente în funcțiile din computed), ce poate returna în timp real odată cu modificarea variabilei de care depinde o valoare compusa (poate returna chiar și doar valoarea proprie a unei variabile) dintr-o valoare a unei variabile din data și alte suplimente (pot fi constante sau chiar alte variabile din data).

```
computed: {
    errorTotal() {
        let error = 0;
        for (const key în this.personalData) {
            if (
                this.$v.personalData[key].$anyError == true ||
                this.$v.personalData[key].required == false
            ) {
                error++;
            }
            if (
                key == "dataNastere" &&
                this.$v.personalData[key].$anyError == false &&
                this.$v.personalData[key].required == true &&
                !(this.personalData[key] <= this.today)
            ) {
                error++;
            }
        }
        return error;
    },
},
```

Fragment de cod 3-5 Returnează eroarea totală ce depinde de erorile mai multor variabile, aceasta valoarea se va schimba concomitent cu vreo modificare a unei variabile de care depinde

Watch – reprezintă o opțiune a Vue ce permite urmărirea schimbării valorilor unei anumite variabile din obiectul *data* a componentei în scopul gestionarii unei acțiuni când aceasta modificare are loc. Aici în dependenta de ce variabila urmărim avem doua tipuri de watch-uri:

1. *Simplă*
2. *In adâncime*

Watch-ul în adâncime se aplica atunci când urmărim un obiect săi array întreg, adică când urmărim concomitent modificările pentru valorile tutoror cheilor obiectului sau array-ului urmărit. în cazul acestul watch este nevoie de menționat ca dorim o urmărire în adâncime, în caz contrar Vue ne va înțelege ca urmărim o variabila simpla, ceea ce nu este prea legal în cazul unui obiect sau array.

```
export default {
    data() {
        return {
            ...
            selected: [],
        };
    }
      watch: {
        selected: {
            handler(newValue, oldValue) {
                if (this.selected.length == 0 && this.showMessages == true) {
```

```
                this.showMessages = !this.showMessages;
            }
            this.updatecheckIfGradeIsMore(newValue);
        },
        deep: true,
    },
},
...
}
```

Fragment de cod 3-6 Exemplu de utilizare a optiunii watch în adâncime

- **Directivele built-in în Vue.js**

In construirea website-ului cu Vue.js s-au folosit foarte des built-in directive ale framework-ului, și anume:

- **v-if**. Această directivă este responsabilă pentru redarea condiționată a elementelor. Dacă controlați vizibilitatea unui element prin if, atunci elementul dvs. va dispărea și va apărea în arborele DOM în funcție de valoarea din directiva. v-if ia valori boolene. De asemenea, în aceasta directiva putem atribui și metode/funcții care returnează o anumita valoare booleană.

```
<div
    class="hstack gap-1"
    v-if="im_user == false || user_login == false"
>
    <template v-if="user.status == 'not_friends'">
        ...
    </template>
</div>
```

Fragment de cod 3-7 Exemplu de utilizare a directivei v-if cu o variabila

- **v-else, v-else-if**. Directiva if este convenabilă deoarece poate fi utilizată împreună cu directivele v-else și v-else-if. În funcție de valori, va fi afișată una sau alta componentă.

```
<template v-if="tip_user == 'profesor'">
    Nr. de institutii în care ai lucrat
</template>
<template v-else-if="tip_user == 'student1'">
    Nr. de institutii în care ai invatat
</template>
```

Fragment de cod 3-8 Exemplu de utilizare a directivei if în combinatie cu v-else

- **v-for.** Pentru a afișa datele unui array într-o lista de elemente exista directiva v-for. Aceasta are următoarea sintaxa – v-for = „item în items” :key = item.id, sau o varianta alternativa – v-for = „(item, index) în items” :key = index. Ulterior în interiorul tag-ului asupra căruia aplicam acest v-for putem sa ne adresam către elementul iterat.

```
<template>
```

```
    <ul class="students">
        <li v-for="student în students">{{student}}</li>
    </ul>
</template>

<script>
    export default {
        name: 'App',
        data() {
            return {
                students: [
                    'Alex',
                    'Robert',
                    'Marta'
                ]
            }
        }
    }
</script>
```

Fragment de cod 3-9 Exemplu cu utilizarea v-for

!Este important ca pentru „:key” sa fie introdusa o valoarea unica a listei pe care vrem sa o afișam.

- **v-bind.** Directiva este utilizată pentru așa - numita legare a datelor. Pentru a transmite date dinamice în interiorul unui atribut html, este necesar să transmiteți acest atribut ca argument la directiva v-bind. Argumentele directivelor sunt transmise după caracterul de doua puncte. în loc de „v-bind:” se poate folosi doar „:”.

```
<a @click="modal.id = image.id" :href="'#image_' + index_gallery" data-bs-
toggle="modal">
    <img class="rounded w-100 h-100" :src="image.path" alt="" />
</a>
```

Fragment de cod 3-10 Pentru href și src se transmit niste variabile dinamice, care se pot schimba pe parcurs de utilizare a site-ului

- **v-model.** Directiva este într-un fel similară cu v-bind și servește pentru legarea bidirecțională între elementele formularului de date și datele din obiectul de date (two-way binded). Directiva ignoră atributele html: *value, checked, selected*, care sunt prezente în elemente. Pentru valoarea reală, directiva ia datele pe care le transmitem de la obiectul *data*. în exemplul de mai jos am legat valoarea inputului cu variabila name.search din obiectul data al exemplarului vue. La introducerea unor caractere în inputul dat se vor înscrie aceste caractere în variabila *name.search*.

```
<input type="text" placeholder="Cauta persoana..." class="form-control form-
control-sm" v-model="name_search" />
```

Fragment de cod 3-11 Exemplu cu utilizarea directivei v-model

- **v-on.** Adăugarea unui eveniment la un element este implementată de directiva v-on. Numele evenimentului trebuie transmis ca argument al directivei după doua puncte. Numele evenimentelor sunt similare cu evenimentele JavaScript. Se poate omite

scrierea v-on, lăsând doar doua puncte și denumirea acțiunii ce trebuie ascultata pe element.

```
<button v-if="my_id == selected_user_id" @click="delete_comment(comment,
'comment')" class="btn btn-sm btn-danger">
    <i class="bi bi-trash3-fill"></i>
</button>
```

Fragment de cod 3-12 La click pe butonul din exemplu se va efectua functia "delete_comment"

Pe lângă click, vue oferă pseudonime și pentru alte cele mai frecvent utilizate coduri cheie:

- enter
- .tab
- .delete (captures both “Delete” and “Backspace” keys)
- .esc
- .space
- .up
- .down
- .left
- .right

De asemenea se poate utiliza și keyCode atribute în definirea ce taste trebuie ascultate.

```
<input v-on:keyup.13="submit">
```

Fragment de cod 3-13 Exemplu de utilizare a v-on cu keyCode

Totodată Vue oferă și posibilitatea definirii pseudonimului pentru diverse taste:

```
// enable `v-on:keyup.f1`
Vue.config.keyCodes.f1 = 112
```

Fragment de cod 3-14 Definirea unui pseudonim pentru tasta cu codul 112

- **Validarea input-urilor cu Vue.js**

Pentru validarea datelor în Vue am utilizat sub-librăria Vuelidate – ce este un instrument pentru validarea input-urilor pe front end pentru Vue.js. Pentru a utiliza acest instrument, este nevoie ca în partea de script a fișierului .vue sa facem un import din sub-librăria Vuelidate înregistrată cu elementul de validare (tipul validatorului) ce ne interesează:

```
import { required } from "vuelidate/lib/validators";
```

Fragment de cod 3-15 Importul elementului "required" pentru input

Ulterior, având deja elementul „required” la îndemână, putem sa determinăm care input este necesar sa fie „required”, adică sa aibă cel puțin un caracter. Pentru aceasta se creează un obiect cu numele rezervat „validations”, în care introducem denumirea variabilei v-model din

data (adică variabila care răspunde pentru input-ul interesat) și introducem tipul validatorului ce se presupune a fi deja importat:

```
validations: {
    message: { required },
},
```

Fragment de cod 3-16 Inițializăm ca v-model message sa fie required

Pe lângă *required*, Vuelidate ofera și alte tipuri de validatori built-in: *https://vuelidate.js.org/#sub-builtin-validator*.

- **Înregistrarea componentelor și reutilizarea acestora în Vue.js**

Cel mai simplu mod de a înregistra componente este să o faceți la nivel global folosind metoda Vue.component. Cu înregistrarea globală, acestea pot fi utilizate oriunde în arborele component al aplicației dvs., de exemplu:

```
// for pages
Vue.component('v-register', require('./pages/Register.vue').default);
Vue.component('v-login', require('./pages/Login.vue').default);
Vue.component('v-contact', require('./pages/Contact.vue').default);
Vue.component('v-search', require('./pages/Search.vue').default);
Vue.component('v-chat', require('./pages/Chat.vue').default);
Vue.component('v-friends', require('./pages/Friends.vue').default);
Vue.component('v-page-reviews', require('./pages/Reviews.vue').default);
Vue.component('v-page-feed', require('./pages/Feed.vue').default);
Vue.component('v-page-files', require('./pages/Files.vue').default);
```

Fragment de cod 3-17 Exemplu de inregistrare globala intr-un fisier app.js a componentelor

Exista și posibilitatea înregistrării locale a componentelor.

```
<script>
    import {required} from "vuelidate/lib/validators";
    // components
    import friendItem from "./FriendItem.vue";
    import postItem from "./Post.vue";
    import commentsItems from "./Comments.vue";
    ...
    export default {
        ...
        components: {friendItem, postItem, commentsItems},
    ...
}
</script>
```

Fragment de cod 3-18 Înregistrarea locală a componentelor

Componentele înregistrate se pot reutiliza de atâtea ori, de cate ori este nevoie. Spre deosebire de înregistrarea acestuia în partea de script a codului, unde putem utiliza PascalCase, la apelarea componentelor în template este obligatorie utilizarea kebab-case pentru funcționarea corecta a acestora.

```
<friend-item :item="user" :im_user="selected_user_id == my_id" />
```

Fragment de cod 3-19 Apelarea componentelor inregistrate cu kebab-case cu inchiderea directa a tag-ului

```
<post-item v-for="post in posts_info" :key="post.id" @send_post="send_post"
@modal_user_list="modal_user_list"
```

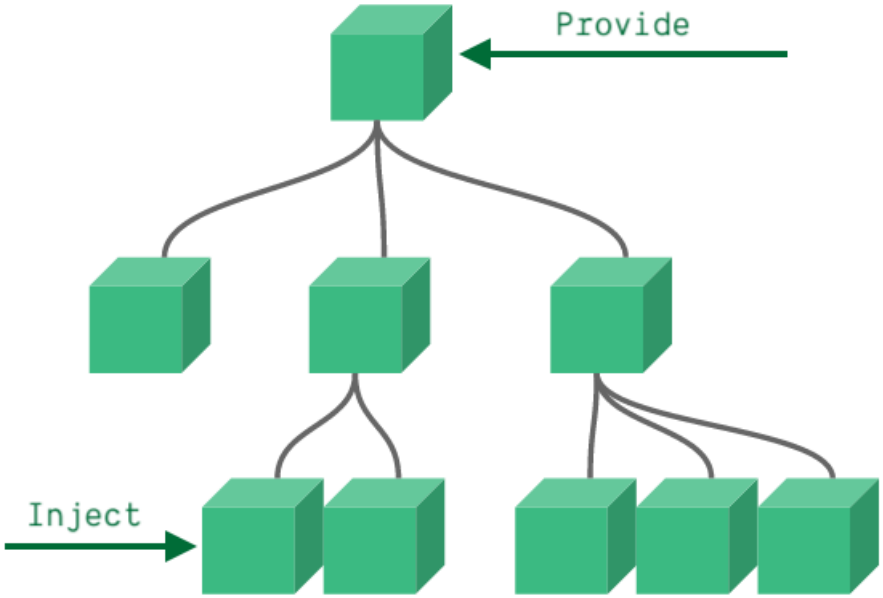
```
    @populate_post_card="populate_post_card" @modal_delete="modal_delete"
:selected_user_id="selected_user_id"
    :get_post="post" :my_id="my_id" :type="type"></post-item>
```

Fragment de cod 3-20 Apelarea componentei înregistrate cu kebab-case cu utilizarea tag-ului suplimentar de inchidere

După cum se poate observa în ultimele doua exemple de cod, apelarea componentelor este însoțită de niște atribute. Atributele însoțite de „:” reprezintă niște variabile dinamice ce se pot schimba în componenta-părinte și ulterior aceste componente schimbate se transmit și în componenta-copil și în cazul existentei unei dependente fata de aceste variabile în componenta-copil, se va schimba și informația dependenta de aceste variabile din componenta-copil. Atributele însoțite de „@” reprezintă niște funcții ce se vor emite din componenta-copil în componenta-părinte și la emiterea acestor funcții se vor realiza funcțiile din dreptul „=”, codul cărora este determinat în componenta-părinte. Aceste atribute sunt strâns legate de procesul de provide/inject din cadrul librăriei Vue.js care îl vom descrie în paragraful despre ***Error! Reference source not found.*** Pe lângă componentele noastre, în cadrul proiectului nostru au fost utilizate și componentele BootstrapVue. Mai mult despre componentele Bootstrap se poate citi aici: https://bootstrap-vue.org/docs/components.

- **Provide/inject**

Este un proces de transmitere a datelor intre componente.



Img 3-2 Principiul de lucru provide/inject intr-o ierarhie de componente
Sursa: (Vue.js, Provide / inject | Vue.js, 2020)

Transmiterea variabilelor din componenta părinte către componenta copil se realizează simplu prin atribuirea atributelor însoțite cu „:” ce corespund denumirilor variabilelor din props-ul componentei-copil niște valori a variabilelor din data a componentei-parinte, procesul numindu-se „*inject*”. Procesul de atribuire a valorilor în atributele tag-ului componentei-copil se poate vedea in:

```
<post-item v-for="post în posts_info" :key="post.id" @send_post="send_post"
@modal_user_list="modal_user_list"
    @populate_post_card="populate_post_card" @modal_delete="modal_delete"
:selected_user_id="selected_user_id"
    :get_post="post" :my_id="my_id" :type="type"></post-item>
```

Fragment de cod 3-21 Exemplu de atribuire a unor valori pentru props-urile copiilor

!Este important ca denumirile atributelor din tag sa corespunda denumirilor de variabile din opțiunea props a componentei-copil.

Procesul invers la inject se numește *provide* și constă în transmiterea datelor de la un copil către părinte. Transmiterea de la copil se realizează prin funcția $emit. Ca parametri în funcție pe primul loc trimitem denumirea funcției care o vom citi în componenta părinte și ceilalți argumente vor fi valorile ce le vom culege în componenta-părinte. Mai jos avem un exemplu în care pe un click pe buton se vor transmite like-urile unui comment dintr-o iterație a unei liste de comentarii.

```
<button data-bs-toggle="modal" :data-bs-target="'#users_list'"
@click="$emit('modal_user_list', comment.likes)"
    class="btn btn-secondary btn-sm">
    <i class="bi bi-caret-down-fill"></i>
</button>
```

Fragment de cod 3-22 Trimiterea datelor din componenta-copil (comments-items)

După aceste manipulări, în componenta-părinte vom prinde funcția cu denumirea corespunzătoare primului argument din $emit și o vom readresa într-o noua funcție declarata în componenta părinte pentru a lucra cu datele primite.

```
<comments-items @modal_user_list="modal_user_list" :my_avatar="my_avatar"
:my_id="my_id"
    :selected_user_id="selected_user_id" :get_post_id="modal.post.id" />
```

Fragment de cod 3-23 Recepționarea datelor din componenta-copil în componenta-copil

Atributul însoțit cu caracterul „@” reprezintă acea denumire de funcție din componenta-copil pe care am setat-o ca prim argument în $emit. Partea de după „=” reprezintă funcția din methods a componentei-părinte unde se va realiza manipularea cu datele recepționate.

```
methods: {
    modal_user_list(users) {
        this.modal.users_list = users;
    },
},
```

Fragment de cod 3-24 Funcția din componenta-părinte ce se ocupa cu manipularea datelor recepționate

!Este important ca componentele sa fie legate într-o anumita ierarhie pentru provide și inject lucru, în caz contrar datele nu vor putea trimise către elementele ce nu se supun ierarhiei.

- **Utilizarea axios în Vue.js**

Folosind Laravel, nu este nevoie de efectuat o instalare suplimentară a axios-ului. Axios-ul este un instrument de comunicare direct dintre API și backend-ul site-ului. În cadrul proiectului nostru au fost utilizate două tipuri de request-uri (ambele având o tratare diferita a datelor) de API către back-end:

1. *Cu date simple* – în acest caz ca argumente în tipul metodei de transmitere a datelor se introduce doar route-ul api-ului și ca al doilea argument – informația propriu zisa, sau request-ul cu alte cuvinte.
2. *Cu date ce conțin și fișiere* – se necesită prelucrarea preventiva a informației, și anume se creează un obiect de tipul FormData și în el se inserează informația ce necesita trimisa, după se urmărește aceleași acțiuni ca și în cazul trimiterii datelor simple, doar ca se mai indica al 3 argument - *headers: { "Content-Type": "multipart/form-data", }* – ce indica ca request-ul trimis conține și fișiere.

Informația din back-end în axios se captureaza în promise-ul *.then*, unde ca argument se indica variabila ce va conține request-ul din back-end.

De asemenea am utilizat și așa metoda de promise-uri ca: *.catch* – pentru prelucrarea erorilor din back-end și *.finally* – pentru realizarea unor functii indiferent daca în back-end au apărut erori sau request-ul a fost prelucrat cu succes.

- **Lucrul cu JQuery**

JQuery s-a utilizat pentru a manipula cu niște clase din cadrul unui grup de elemente ce au un selector comun (pentru a nu lucra cu fiecare element în parte, ce ar mari codul cu mult) sau pentru anumite comenzi ce nu exista în librăria Vue.js, iar scrierea acestora în Vanilla.js ar fi fost puțin mai complex în cod. Aici s-au folosit doar așa comenzi de cod ca:

- *.addClass* – pentru adăugarea unor clase sau clasa unui grup de elemente sau chiar unui element cu selector-ul corespunzător celui selectat.
- *.removeClass* – invers lui *.addClass* – șterge clasele date ca argument din elementele selectate.
- *.toggleClass* - combina *.addClass* și *.removeClass în* planul ca când lipsește clasa indicata, aceasta este adăugată, iar daca este prezenta, atunci se șterge.

- *.each* – parcurgerea fiecărui element cu selector-ul dat ca argument.
- *.height* – calcularea înălțimii elementului selectat.
- *.width* – calcularea lățimii elementului selectat.
- *.offset* – determinarea poziției pe y pentru elementul selectat.
- etc.

3.3. BACK-END - LARAVEL

Pana a începe lucrul cu back-ul, sa realizat o configurare în fișierul *.env* din root-ul proiectului nostru, și anume: s-a configurat baza de date cu care comunicam (aceasta initial cu tot cu nume a trebuia fi creata deja în XAMPP) și s-a configurat mail-ul (s-a introdus login-ul, parola etc.). Pe lângă aceste configurări, în fișierul *.env* este posibila și realizarea altor configurări (cum ar fi configurarea contului pusher), însă pentru proiectul nostru, nu s-au necesitat aceste configurări.

Dupa cum s-a mai mentionat. Laravel se bazează pe structura *Model/View/Controller*. Deci, sa descriem succint fiecare:

- **Model**

Reprezinta cu alte cuvinte obiectul tabelei din baza de date. Un model se creează în Laravel prin tastarea comenzii *php artisan make:model* ***denumirea_modelului. în*** cazul proiectului nostru am utilizat și sufixul *-m* pentru majoritatea modelelor ce ne creează și un fișier pentru structura tabelului**.** Aici avem posibilitatea sa cream atâtea câmpuri, cate credem noi ca avem nevoie în cadrul tabelului, și sa indicam tipul și relațiile cu alte tabele. Toate tipurile disponibile pentru coloane în Laravel le putem găsi pe acest link: *https://laravel.com/docs/9.x/migrations#available-column-types*.

```
...
public function up() {
    Schema:: create('users', function (Blueprint $table) {
        $table -> id();
        $table -> string("email") -> unique();
        $table -> timestamp("email_verified_at") -> nullable();
        $table -> longText("password");
        $table -> string("name");
        $table -> string("surname");
        $table -> date("date_birth");
        $table -> date("date_finish") -> nullable();
        $table -> boolean("prezent_activity") ->default (false);
        $table -> string("country");
        $table -> string("city");
        $table -> string("user_type") -> nullable();
        $table -> string("avatar_path") -> nullable();
        $table -> rememberToken();
        $table -> string('role') ->default ('simple_user');
```

```
            $table -> timestamp('online_at') -> nullable();
            $table -> timestamps();
        });
    }
...
```

Fragment de cod 3-25 Codul pentru crearea structurii tabelului user

După ce structura tabelului este definita, utilizam următoare comanda în terminal ce ne creează tabelul respectiv în baza de date – *php artisan migrate*. Deseori apărea cazul când era nevoie de o restructurare a tabelului. Pentru aceasta se apela la 2 metode:

1. Se modifica structura bazei de date la starea dorita și dupa în terminal se apeleaza una din comenzile: *php artisan migrate:fresh* sau *php artisan migrate:refresh*. Minusul metodei date este ca șterge toate populările din baza de date.
2. Prin crearea unei migrații noi ce va cuprinde tabelul ce dorim și (după caz, daca nu dorim de exemplu doar denumirea tabelului sa modificam) câmpurile cu acțiunile de modificare. Aceasta migrație se creează prin introducerea următoarei comenzi în terminal: *php artisan make:migration* ***denumirea_migratiei***. Dupa efectuarea manipulărilor descrise anterior apelam *php artisan migrate în* terminal.

Pe parcursul proiectării lucrări utilizându-se deseori metoda 1 de restructurare a tabelei, apărea problema de golire a populațiilor din baza de date, fiind, la prima vedere, cazul sa se populeze manual baza. Însă și aici Laravel ne-a venit cu un instrument ajutător, și anume – *db seed*. Laravel are o metodă simplă de umplere a bazei de date cu date de testare folosind clase de populare. Aceste clase sunt stocate în baza de date/seeds. Puteți utiliza orice nume pentru numele clasei de umplere, dar ar fi logic să folosiți nume precum UsersTableSeeder. În mod implicit, clasa DatabaseSeeder a fost deja creată în folder-ul seeders. În această clasă puteți utiliza metoda de apelare pentru a lansa alți seederi, ceea ce vă permite să controlați ordinea umplerii. în cazul proiectului nostru nu ne-am complicat mult și am lăsat doar clasa generală DatabaseSeeders în care am introdus toate populările necesare și după în terminal am apelat comanda - *php artisan db:seed* ca sa pornească procesul de populare a bazei de date. în caz ca aveți clase separate de umplere și nu doriți sa apelați clasa generala, puteți sa porniți aceste umpleri cu tastare în terminal a comenzii: *php artisan db:seed --class=****denumirea_clasei.***

Ce se refera la fișierele proprii ale modelelor – aici, cu ajutorul instrumentelor instalate în Laravel, s-au definit relatiile modelelor cu alte modele (de nu încurcat cu relațiile din structura fișierului de migrației):

```
...
public function up() {
    Schema:: create('users', function (Blueprint $table) {
        $table -> id();
        $table -> string("email") -> unique();
        $table -> timestamp("email_verified_at") -> nullable();
        $table -> longText("password");
        $table -> string("name");
```

```
            $table -> string("surname");
            $table -> date("date_birth");
            $table -> date("date_finish") -> nullable();
            $table -> boolean("prezent_activity") ->default (false);
            $table -> string("country");
            $table -> string("city");
            $table -> string("user_type") -> nullable();
            $table -> string("avatar_path") -> nullable();
            $table -> rememberToken();
            $table -> string('role') ->default ('simple_user');
            $table -> timestamp('online_at') -> nullable();
            $table -> timestamps();
        });
    }
    ...
```

Fragment de cod 3-26 Crearea unui tabel prin Laravel în fișierul de migratie

Aceste definiri ne ajuta cu mult la simplificare codului ce necesita scrisa la interacțiunea cu baza de date. în proiect s-au utilizat 3 din cele 11 relații posibile în Laravel: *One to Many*, *One to Many (Invers) și Many to Many* – care se realizează prin indicarea relației *One to Many (Invers)* pentru toate modelele intre care exista aceasta relație.

```
public function activities() {
    return $this -> hasMany(Activities:: class);
}
```

Fragment de cod 3-27 Relație One to Many (user-ul are mai multe activități)

```
public function user() {
    return $this -> belongsTo(User:: class);
}
```

Fragment de cod 3-28 Relație One to Many (Invers) pentru activități (o activitate are mai mulți useri)

```
// for friends
public function friendOfMine() {
    return $this -> belongsToMany(self:: class, Friends:: class, 'user_id',
'friend_id');
}
public function friendFor() {
    return $this -> belongsToMany(self:: class, Friends:: class, 'friend_id',
'user_id');
}
```

Fragment de cod 3-29 Relație Many to Many (un anumit User poate avea mai multi useri ca prieteni, precum și orice prieten al sau poate avea mai mulți useri ca prieteni)

Pe lângă definirea relațiilor, în modele am definit comod și alte funcții de afișare a informației pentru diverse instanțe:

```
public function hasRole($role) {
    if ($this -> roles() -> where('name', $role) -> first()) {
        return true;
    }
    return false;
}
```

Fragment de cod 3-30 Funcție ce-mi afișează daca un anumit user are vreun rol sau nu

Datorita utilizării structurii OOP în Laravel și instrumentelor sale native, putem cu mult simplifica codul ce trebuia sa-l scriem utilizând un PHP nativ.

- **View.**

Reprezintă de fapt un șablon cu html în care pot fi introduse variabile din back-end și instrumente din PHP, cum ar fi: iterarea unei liste cu *for*, o afișare condiționată cu *if* etc. Laravel se poate lauda cu modelul Blade, care conține instrumente pentru o structurare comoda și cat mai eficienta a view-urilor noastre, cum ar fi: @yield -> @extend + @section, @stack -> @push, @include etc. Afișarea variabilelor numai necesita de a fi însoțita de *„echo”*, fiind nevoie de luat intre „{{}}” informația ce dorim s-o afisam. Pentru a ne putea folosi de instrumentele oferite de Blade din Laravel, este nevoie ca fiecare fisier sa-i oferim extensiunea *„.blade.php”*.

- **Controller.**

Este ca un mediu în care se realizează comunicarea cu baza de date și gestionarea informației. Aici primim un request din view-urile noastre, care de fapt reprezintă niște input-uri de la utilizator, și în baza acestor input-uri putem gestiona informația din baza de date. Laravel oferă instrumente foarte puternice în ceea ce privește acțiunile CRUD, validarea request-ului obținut, lucrul cu fișierele etc. Pentru a crea un controller este nevoie ca în terminal, în locația proiectului nostru, sa apelam urmatoare comanda: *php artisan make:controller* ***denumirea_controller***.

In lucrul cu CRUD avem posibilitatea alegerii intre 2 instrumente: Elequent sau Database query builder de la Laravel, prima, la rândul sau, având nevoie de existenta modelelor pentru tabelele cu care dorim sa interacționam. Ambele instrumente au instalate implicit în ele protecția aplicației de potențiala injecție SQL. Cu ajutorul instrumentelor oferite de Laravel pentru CRUD, nu mai avem nevoie sa scriem un string lung pentru fiecare acțiune CRUD către baza de date, acestea deja având opțiuni pentru toate cazurile (multiple insert, order by, group by, where etc.).

Validarea request-ului în Laravel are deja instalata din cutie reguli pentru cele mai des întâlnite situații de validare. Aceste reguli le putem găsi pe următorul link: https://laravel.com/docs/9.x/validation#available-validation-rules. în cazul unei erori de validare, se va face un return către view-ul de unde a venit request-ul cu o lista în care sunt înscrise toate erorile de validare pentru acest request.

```
$validated = $request -> validate([
    'text' => 'required|min:500',
    'type_of_post' => 'required',
    'note' => 'exclude_if:type_of_post,post|required|numeric|min:1|max:5',
    'files*' => 'image',
]);
```

Fragment de cod 3-31 Exemplu de validare a input-urilor și salvarea request-ului validat în variabila $validated

Pentru a realiza comunicarea dintre input-urile user-ului cu Controller-urile corespunzătoare, precum și accesarea anumitor pagini pe un anumit URL este nevoie de un mecanism de rutare. Laravel ne oferă acest mecanism, și pe lângă asta deține și niște middleware din stock, oferindu-ne posibilitate și de creare unor middleware custom. în proiectul dat au fost utilizate așa middleware ca:

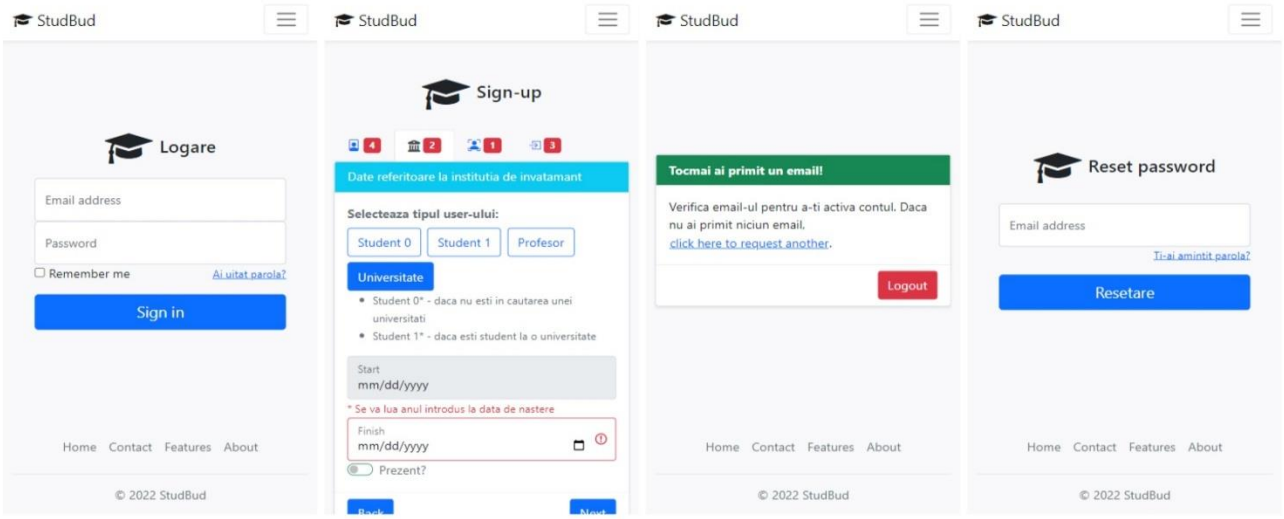
- *guest* - permite accesul doar user-ilor neautorizați.
- *auth* – restricționează accesul user-ilor care nu sunt autorizați.
- *verified* – restricționează accesul la view-uri daca userul curent nu are email-ul verificat.
- *throttle:6,1* – limitează realizarea a 6 request-uri pe minut (s-a utilizat pentru trimiterea email-ul de validare a poștei electronice a utilizatorului).
- *admin* – un middleware custom care restricționează accesul pentru userii ce nu sunt administratori.

In Laravel sunt două tipuri de rute: rute web și rute API, pentru fiecare din ele există un fișier aparte în care se pot înregistra aceste rute. în proiectul nostru au fost utilizate ambele rute. Daca pentru web.php middleware-urile de mai sus pot fi apelate fara probleme, în api.php este necesara configurarea lui *„sanctum”*. Mai detaliat despre aceasta se poate găsi pe următorul link: *https://laravel.com/docs/9.x/sanctum*.

De menționat este faptul ca atunci când primim request dintr-un formular, este nevoie de protejat cu un CSRF token request-ul dat, astfel request-ul nu va fi prelucrat și se va ignora. Mai succint despre aceasta pe: *https://laravel.com/docs/9.x/csrf#csrf-x-csrf-token*.

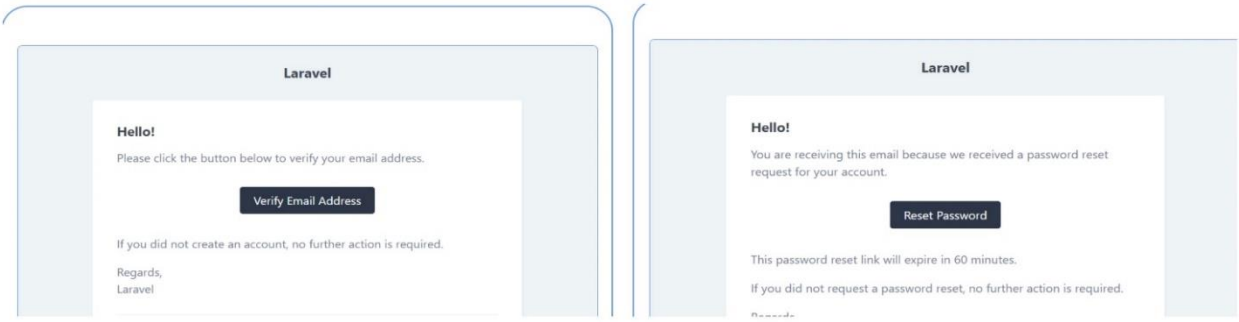
CAPITOLUL 4 DESCRIEREA FUNCȚIONALITĂȚILOR APLICAȚIEI

- **Înregistrare/logare/resetare parolă**



Img 4-1 a) Formularul pentru logare; b) Un compartiment din formularul pentru înregistrare; c) Mesajul că înregistrarea a avut loc cu succes și urmează confirmarea email-ului, c) Formularul pentru resetarea parolei

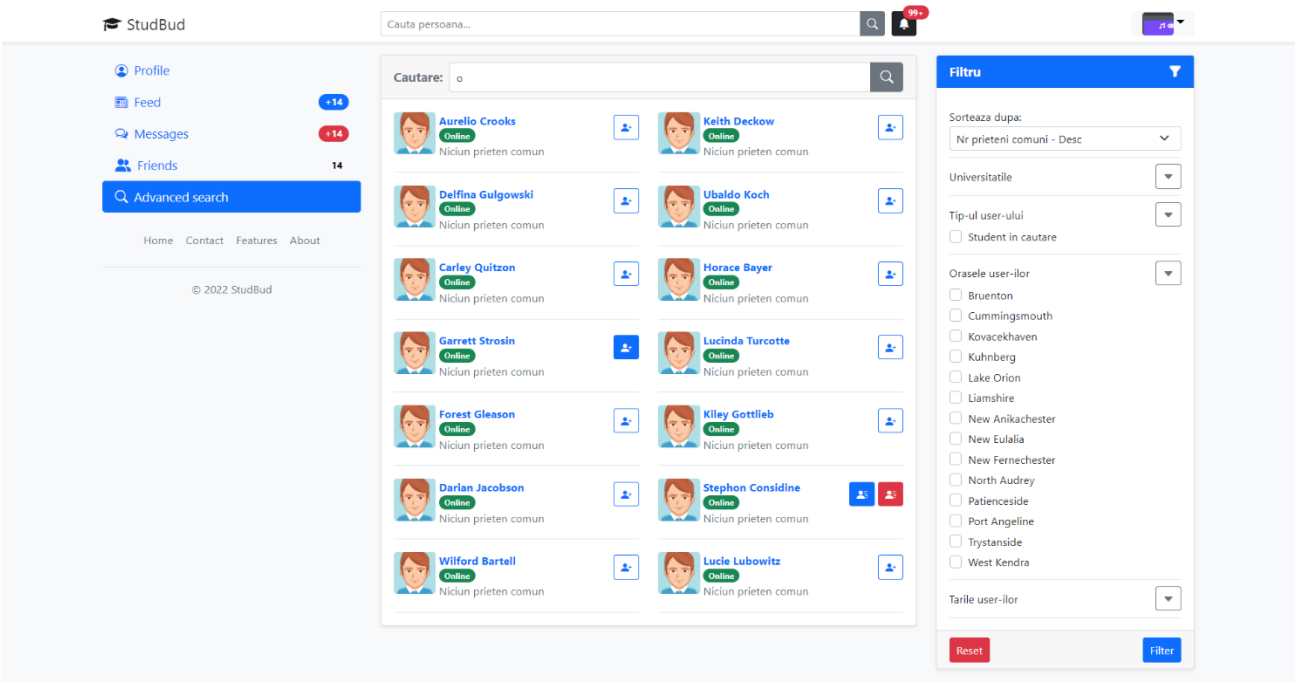
După cum putem observa, formularul de înregistrare conține și posibilitatea monitorizării în timp real a erorilor de validare în cazul în care utilizatorul a tastat butonul submit și erau prezente careva erori la momentul de față - câmpurile greșite sunt colorate cu roșu și suplimentar deasupra fiecărui compartiment este afișat numărul de erori de validare a input-urilor pentru un user experience mai bun. După o înregistrare cu succes, utilizatorul este redirecționat pe o pagină unde avem un card care ne informează că ar trebui sa verificăm poșta electronică. În caz că nu am primit un email corespunzător, putem să facem o retrimitere a cererii sau chiar să ne dezlogăm pentru a reîncerca procesul de înregistrare cu un alt email.



Img 4-2 Mesajul de pe email: a) pentru confirmarea poștei electronice; b) pentru resetarea parolei

- **Căutarea utilizatorilor**

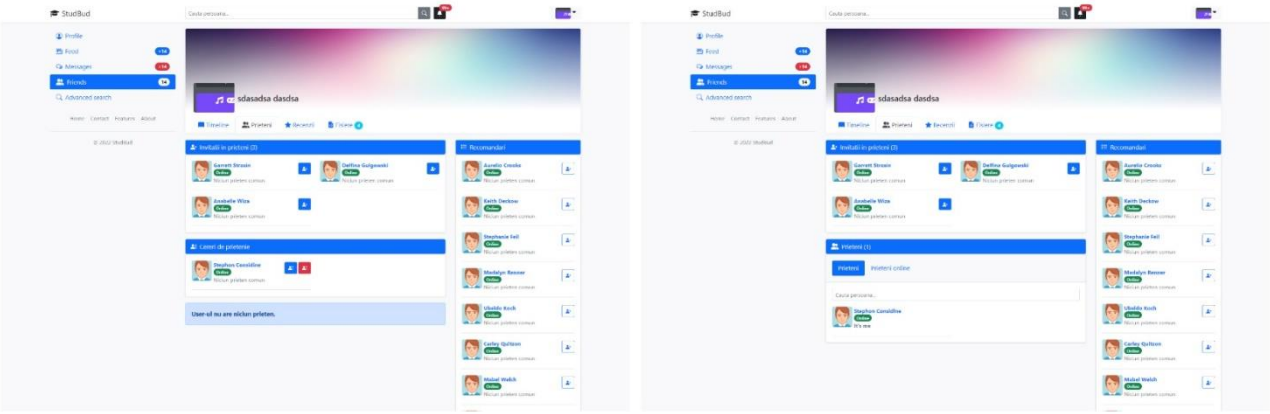
În site-ul nostru avem posibilitatea de căutare a persoanelor. Pentru aceasta avem o pagină specială în care avem posibilitatea să căutăm persoane după nume introducând cuvânt cheie în input-ul deasupra listei cu utilizatori. Putem de asemenea să filtrăm căutarea noastră după anumite criterii și să sortăm lista găsită. În lista găsită putem direct din ea să gestionăm prietenia cu utilizatorii.



Img 4-3 Interfața pentru search users

- **Prietenia**

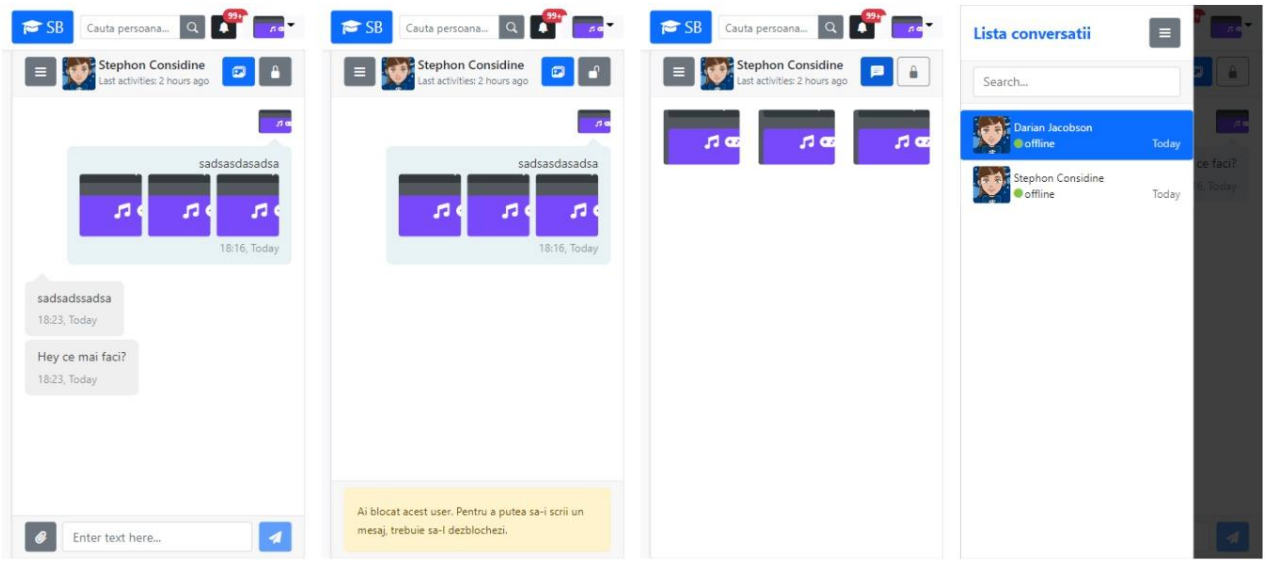
Utilizatorul are o pagină destinată informației despre prietenii săi, unde poate să-și gestioneze relațiile cu aceștia dacă este logat. Aici are o listă cu recomandări, care reprezintă de fapt o listă a tuturor utilizatorilor (în afară de cei care au blocat utilizatorul în cauză) din sistem sortați descrescător numărului de prieteni comuni cu acest utilizator. Din această listă utilizatorul poate adăuga în prieteni anumiți utilizator interesanți lui. Mai în stânga de recomandații întâlnim niște liste cu invitațiile în prieteni făcute de user, listă cu cererile în prieteni venit utilizatorului – aceste liste poate fi văzute doar de utilizatorul al cu este contul, și lista cu prietenii utilizatorului, dacă aceștia există, altfel se va afișa un mesaj corespunzător ca prieteni utilizatorul al moment nu are. Toate listele sunt însoțite cu butoane destinate gestionării prietenii între utilizatori.



Img 4-4 Interfața pentru prietenie

- **Chat**

Aici avem un header cu informația despre utilizatorul cu care comunicăm, de unde putem deschide galeria cu pozele din chat și de unde avem posibilitate să blocăm utilizatorul cu care comunicăm. În momentul când blocăm utilizatorul, câmpul pentru scrierea unui mesaj este indisponibil pentru ambele persoane, conținând un mesaj corespunzător. Chat-ul este însoțit și cu o listă a utilizatorilor cu care am comunicat vreodată, în capul acesteia aflându-se persoana cu care comunicăm în momentul de față.



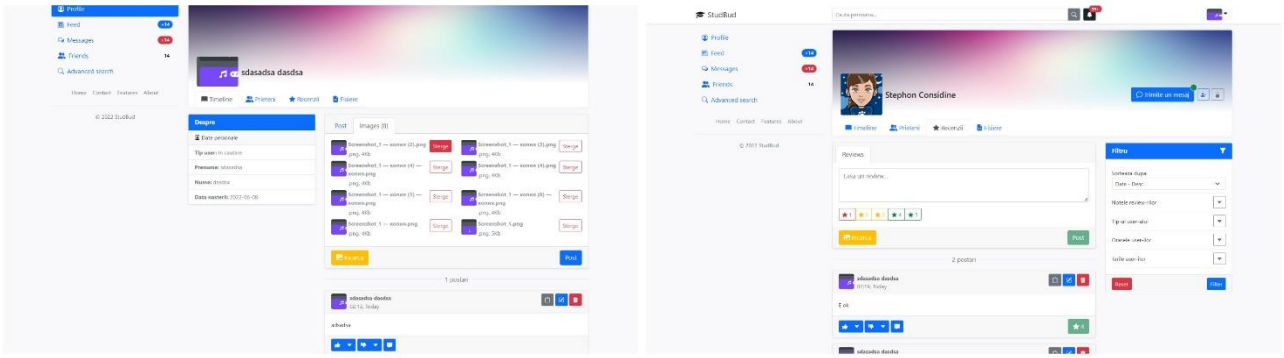
Img 4-5 Interfața de chat pe mobile

- **Timeline/feed și recenzii**

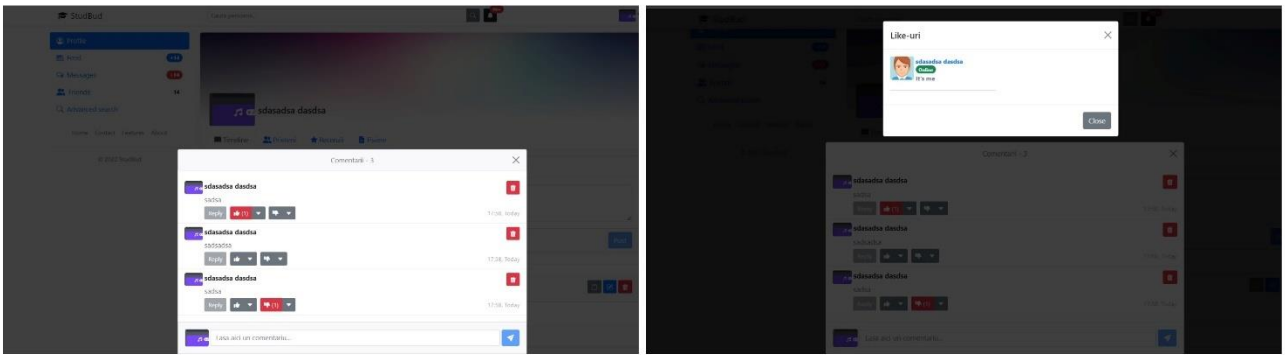
Pe pagina de *timeline* avem un card în stânga privitor la informațiile despre utilizator. În dreapta avem un card cu două tab-uri: 1 – mesajul text al postării, 2 – pozele la postare, cardul este disponibil doar când suntem pe pagina noastră personală, adică nu putem face postări pe pagini străine, și sub el avem lista cu postări. După cum se poate observa – postarea este însoțită de așa butoane ca: copierea link-ului către postare, redactare/ștergere postare (dacă postarea

aparține persoanei curente), butoane pentru like/dislike și un buton ce deschide un offcanvas pentru comentarii.

Pentru pagina de recenzii situația este similară, doar că nu mai avem acea descriere a persoanei, ci în locul ei este pus un filtru pentru filtrarea recenziilor. De asemenea apar mici diferențe la principiul de lucrul pentru cardul cu input-ul informației pentru recenzie – aici acesta apare doar în cazul în care suntem pe o pagină străină, adică putem lăsa recenzii doar pentru alte persoane, nu și pentru noi înșuși, și avem și un câmp nou în acest card – pentru posibilitatea de lăsare a notelor.

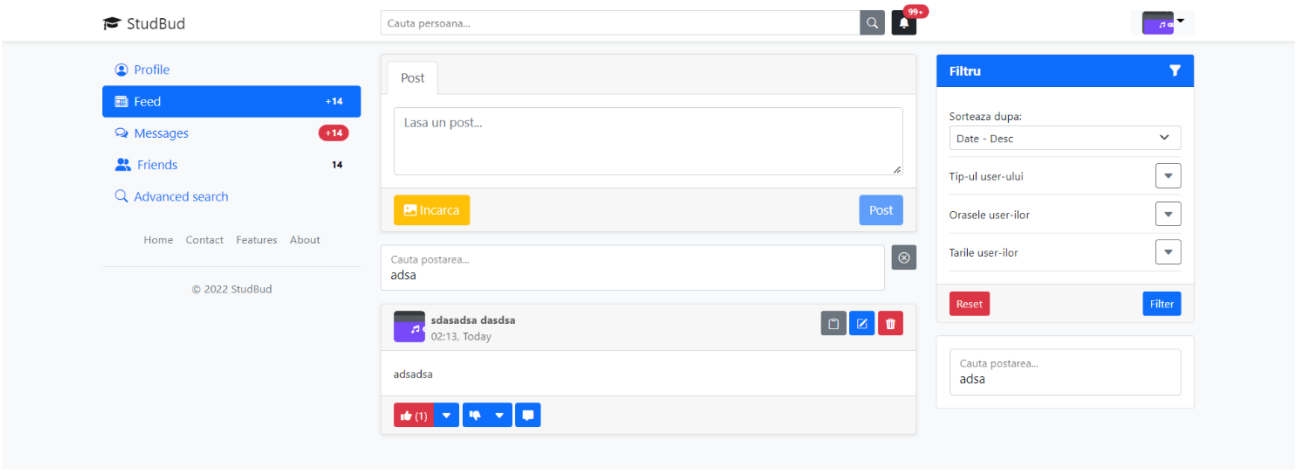


Img 4-6 a) Interfața timeline; b) Interfața pentru recenzii către alt user



Img 4-7 Lista cu comentarii și lista cu like-uri

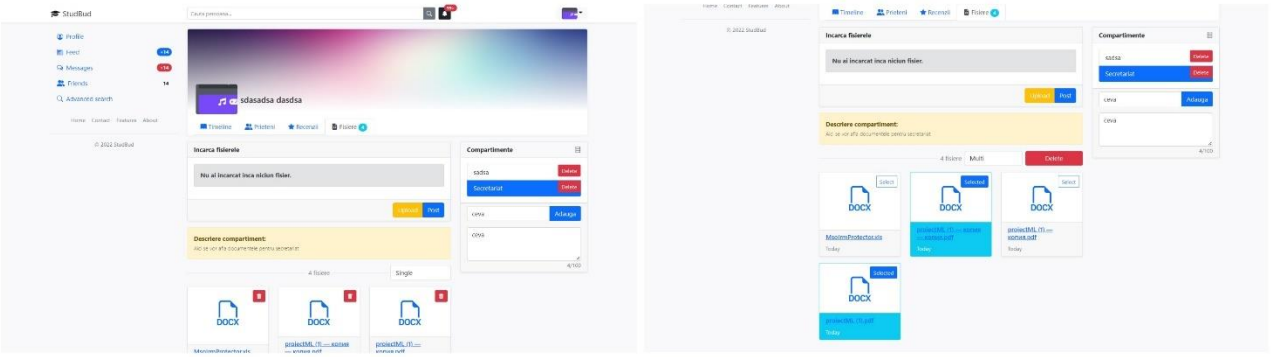
Pagina feed reprezintă o listă cu postările tuturor prietenilor utilizatorilor și cu postările utilizatorilor în sine. Este însoțită cu filtru pentru postări și un input pentru căutarea postărilor după cuvinte cheie.



Img 4-8 Pagina feed

- **Pagina cu fișiere**

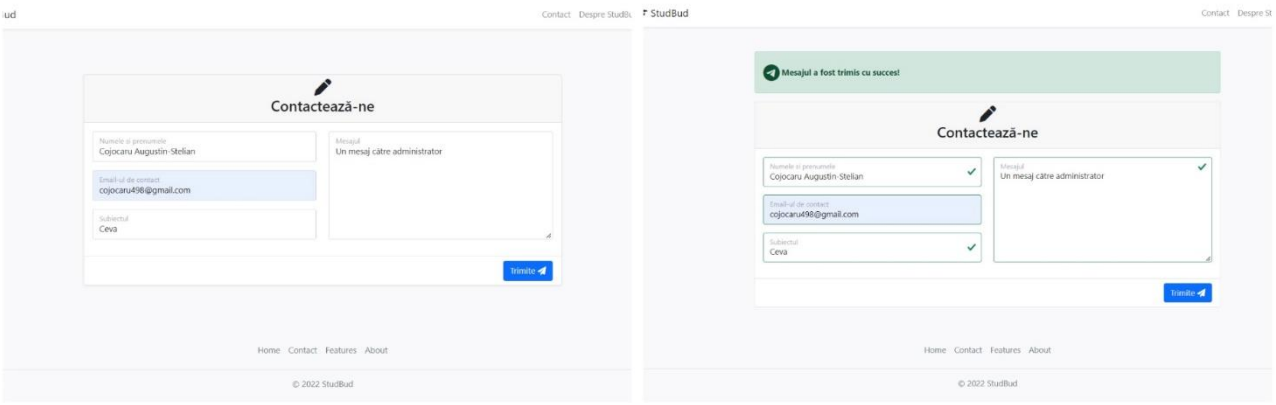
Aici avem un card pentru încărcarea fișierelor, mai jos de el o descriere a secțiunii pentru fișiere unde ne aflăm (dacă există vreo secțiune creată) și după urmează lista cu fișiere din secțiunea dată. În dreapta avem un card pentru comutarea între secțiuni și crearea unei secțiuni/compartiment nou pentru fișiere, dacă este cazul. Ce mai este de menționat – este că avem posibilitatea de a șterge fiecare fișier câte unul sau de a face o ștergere în grup.



Img 4-9 Interfața pentru pagina cu fișiere

- **Formular de contact**

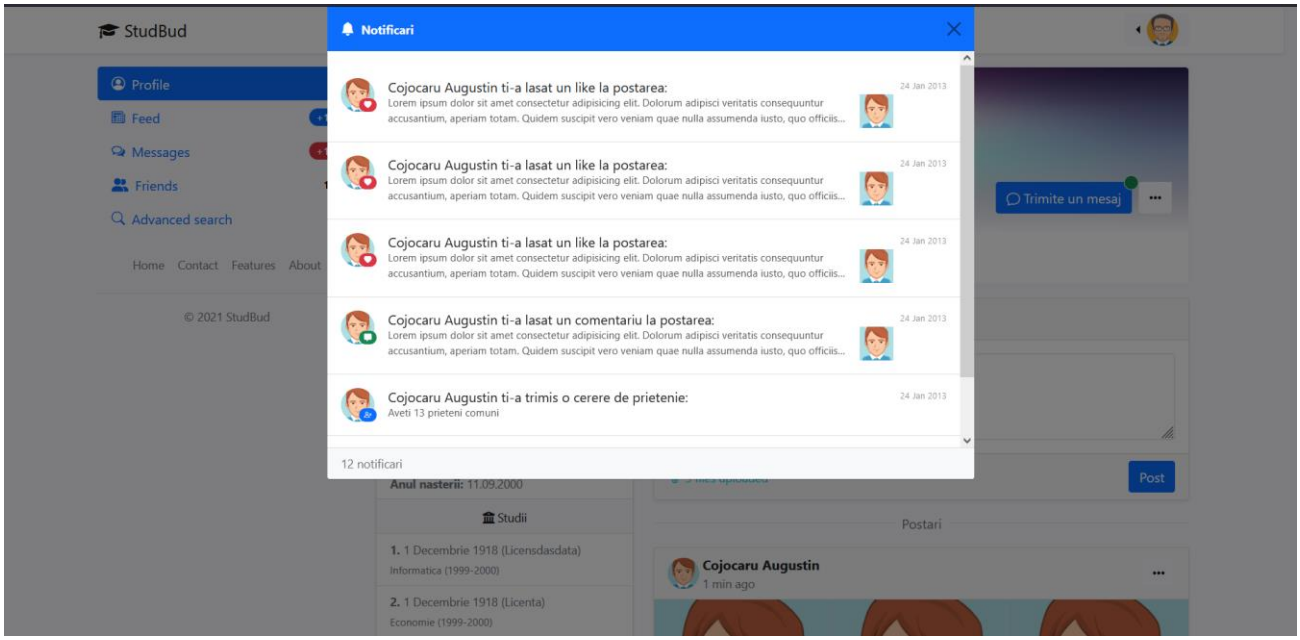
Site-ul este dotat și cu o pagină destinată contactării administratorilor de către utilizatori (fie autentificați sau nu). În urma trimiterii mesajul, utilizatorul este informat cu o fereastră animată în partea de sus a formularului de contact.



Img 4-10 Pagina cu formularul de contact

- **Notificările**

Tastând pe butonul negru pentru notificări din bar-ul de sus a interfeței logate se deschide o listă cu notificări pentru user. Aici utilizatorul poate vedea cine i-a pus like/dislike la comentariu, postare etc., cine i-a lăsat vreun comentariu la postare sau i-a făcut vreun reply la comentariu, cine și când i-a lăsat vreo recenzie etc.

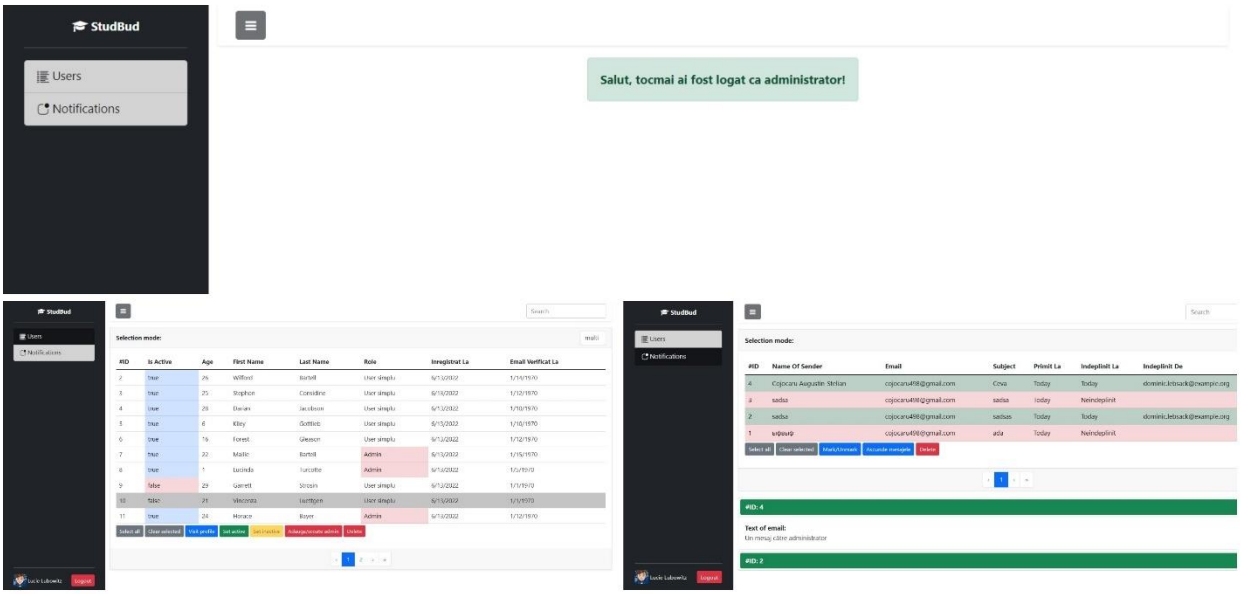


Img 4-11 Lista cu notificări pentru utilizator

- **Admin panel**

Interfața poate fi accesată prin adăugarea la rădăcina URL al site-ului localhost*/admin-panel.* Dacă avem rolul de administratori, atunci ne întâmpină un mesaj că am fost logați ca administrator. În partea stângă avem un sidebar pentru a ne comuta între paginile panoului de administrator, astfel: primul pe listă este tab-ul pentru pagina cu tabela pentru gestionarea utilizatorilor, al doilea este tab-ul pentru pagina cu tabela pentru gestionarea email-urilor primite

de utilizatori. Funcțiile care le putem efectua pentru fiecare tabelă le putem vedea sub fiecare tabelă în ultimele 2 imagini din colajul de mai jos:



Img 4-12 Interfața pentru admin panel

CONCLUZII ȘI PROPUNERI

Ca rezultat al îndeplinirii lucrării de licență, a fost dezvoltată rețeaua socială "StudBud". În timpul procesului de dezvoltare au fost luate în considerare problemele de actualitate și principiile ale creării rețelelor sociale. În același timp, în marea parte - toate sarcinile au fost rezolvate, s-au depus multe eforturi pentru a obține simplitatea maximă și ușurința de utilizare a site-ului.

Toate funcțiile principale ale rețelei sociale sunt implementate: încărcarea fișierelor media, schimbul de mesaje personale, trimiterea și vizualizarea postărilor, blogging-ul, like-uri/dislike-uri etc.

Sunt utilizate cele mai noi tehnologii și metode de dezvoltare, care accelerează activitatea site-ului web și facilitează studiul și înțelegerea acestuia.

În viitor, rețeaua socială „StudBud" rămâne sa fie completată și cu alte funcții noi, funcții care au fost în plan să fie adăugate și din anumite motive (insuficiență de timp) nu au reușit să fie realizate și funcții care nu au fost încă puse în plan de a le realiza sau pentru care competențele actuale nu sunt suficiente pentru a le realiza, însă care vor putea fi cercetate sau vor putea veni ca idee de a le realiza în viitor.

Referitor la tehnologiile utilizate, pentru proiecte one-man vue + laravel = must have. Viteza de dezvoltare, o comunitate mare, o grămadă de pachete gata făcute sunt unele din plusurile ce le oferă această combinație de framework-uri. Combinarea Laravel și Vue js în proiectul dvs., poate ridica proiectul la niveluri mult mai mari. Puteți obține toate avantajele oferite de fiecare framework în parte alăturându-le în proiectele dvs. Ambele oferă caracteristici extraordinare ale lor și, prin urmare, combinarea lor poate crea doar rezultate mai bune, care nu sunt posibile în timp ce se folosește doar una dintre ele. Astfel, integrarea ambelor este necesară pentru rezultate mai bune. În majoritatea versiunilor moderne ale Laravel, este de obicei inclus Vue.js împreună cu Bootstrap și JQuery. O aplicație web creată prin integrarea caracteristicilor puternice ale Laravel și Vue JS ar avea toate calitățile bune oferite de React, Angular. Laravel are un sistem excelent și angajat cu sprijin comunitar. Aceasta face din Laravel o un framework uimitor. Instrumentele de ultimă oră care sunt Vue și Laravel fac o combinație excelentă pentru a crea site-uri web uber cool care au cea mai bună interactivitate.

BIBLIOGRAFIE

1. Bootstrap. (2021). *Grid system · Bootstrap v5.1*. Preluat de pe getbootstrap.com: https://getbootstrap.com/docs/5.1/layout/grid/
2. Borodovskii, D. (2022, Ianuarie 14). *JavaScript-фреймворк Vue.js: особенности и примеры реализации*. Preluat de pe highload.today: https://highload.today/javascript-vue-js/
3. Dementii, D. (2020, Iunie 08). *Почему Laravel — один из лучших PHP-фреймворков для стартапов и энтерпрайза*. Preluat de pe ru.hexlet.io: https://ru.hexlet.io/blog/posts/pochemu-laravel-odin-iz-luchshih-php-freymvorkov-dlya-startapov-i-enterprayza#:~:text=%D0%9F%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D1%8F%20%D0%BD%D0%B0%20Laravel%20%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B8%
4. dobromir-hristov. (2020). *Vuelidate | A Vue.js model validation library*. Preluat de pe vuelidate.js.org: https://vuelidate.js.org/
5. jQuery. (2022). *jQuery*. Preluat de pe jquery.com: https://jquery.com/
6. Laravel. (2022). *Installation - Laravel - The PHP Framework For Web Artisans*. Preluat de pe laravel.com: https://laravel.com/docs/9.x
7. MDN. (2022). *JavaScript | MDN*. Preluat de pe developer.mozilla.org: https://developer.mozilla.org/ru/docs/Web/JavaScript
8. PHP. (2022). *PHP: Hypertext Preprocessor*. Preluat de pe php.net: https://www.php.net/
9. Vue.js. (2016). *Vue.js*. Preluat de pe https://v2.vuejs.org/: https://v2.vuejs.org/
10. Vue.js. (2020). *Lifecycle Hooks | Vue.js*. Preluat de pe vuejs.org: https://vuejs.org/guide/essentials/lifecycle.html#lifecycle-diagram
11. Vue.js. (2020). *Provide / inject | Vue.js*. Preluat de pe vueframework.com: https://vueframework.com/docs/v3/ru/ru/guide/component-provide-inject.html

LISTA DE FIGURI