**MINISTERUL EDUCAŢIEI**

**UNIVERSITATEA „1 DECEMBRIE 1918” DIN ALBA IULIA**

**SPECIALIZAREA INFORMATICĂ**

**FORMA DE ÎNVĂŢĂMÂNT ZI**

**LUCRARE DE LICENȚĂ**

**ABSOLVENT:**

**COJOCARU AUGUSTIN-STELIAN**

**COORDONATOR ȘTIINȚIFIC:**

**CONF. UNIV. DR. ROTAR CORINA**

**ALBA IULIA**

**Iulie, 2022**

**MINISTERUL EDUCAŢIEI**

**UNIVERSITATEA „1 DECEMBRIE 1918” DIN ALBA IULIA**

**SPECIALIZAREA INFORMATICĂ**

**FORMA DE ÎNVĂŢĂMÂNT ZI**

**REȚEA DE SOCIALIZARE PENTRU STUDENȚI, PROFESORI, UNIVERSITĂȚI**

**ABSOLVENT:**

**COJOCARU AUGUSTIN-STELIAN**

**COORDONATOR ȘTIINȚIFIC:**

**CONF. UNIV. DR. ROTAR CORINA**

**ALBA IULIA**

**Iulie, 2022**

Contents

[Capitolul 1 Introducere 3](#_Toc107265449)

[1.1. Scopul proiectului si viziunea 3](#_Toc107265450)

[1.2. Task-uri propuse 3](#_Toc107265451)

[Capitolul 2 Requirements engineering 4](#_Toc107265452)

[2.1. Requirements (functional si non-functional) 4](#_Toc107265453)

[2.1.1. Functional requirements (lista) 4](#_Toc107265454)

[2.1.2. Non-functional requirements (lista) 4](#_Toc107265455)

[2.1.3. Traceability matrix 4](#_Toc107265456)

[2.1.4. Procesul de proritizare a requirement-urilor 4](#_Toc107265457)

[2.2. Resurse necesare (technology requirements) 4](#_Toc107265458)

[2.3. Diagrame: Use-case, UML 4](#_Toc107265459)

[Capitolul 3 Descrierea tehnologiilor utilizate 5](#_Toc107265460)

[3.1. Generalizare 6](#_Toc107265461)

[3.2. Partea de Front-end 7](#_Toc107265462)

[3.2.1. HTML & CSS 7](#_Toc107265463)

[3.2.2. JavaScript 8](#_Toc107265464)

[3.3. Partea de Back-end 10](#_Toc107265465)

[Capitolul 4 Procesul tehnologic 11](#_Toc107265466)

[4.1. Generalizare 11](#_Toc107265467)

[4.2. Front-end 14](#_Toc107265468)

[4.2.1. CSS 14](#_Toc107265469)

[4.2.2. JavaScript 16](#_Toc107265470)

[4.3. Back-end 30](#_Toc107265471)

[Capitolul 5 Proces de instalare 34](#_Toc107265472)

[Concluzii și propuneri 34](#_Toc107265473)

[Lista anexelor 35](#_Toc107265474)

[Anexe 35](#_Toc107265475)

# Introducere

## Scopul proiectului si viziunea

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

* Mesagerie,
* Postari,
* Recenzii catre user,
* Comentarii la postari si recenzii,
* Like-uri, dislike-uri la postari, recenzii si comentarii,
* Friendship mecanism, etc.

## Task-uri propuse

Principalele task-uri propuse la faza initiala 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 mentionate mai sus
* Crearea in prim plan a funcțiilor specifice unei retele de socializare.
* Crearea unei interfate atractive, easy-to-use si intuitive pentru acest site.
* Creare unui spatiu pentru administratorul site-ului pentru a facilita managementul acestui site.

# Requirements engineering

## Requirements (functional si non-functional)

### Functional requirements (lista)

### Non-functional requirements (lista)

### Traceability matrix

### Procesul de proritizare a requirement-urilor

## Resurse necesare (technology requirements)

## Diagrame: Use-case, UML, Contextuala, Swimlane

### UML (diagrama de clase)

Diagram, engineering drawing

Description automatically generated

Figure 2‑1 Diagrama UML (de clase) pentru proiectul nostru

Avem 25 de tabele, din care: 5 sunt optionale – pentru *OAuth* – care se refera la Laravel Passport, un mecanism de securitate pentru request-uri de date din API, poate fi inlocuit cu sanctum si in acest caz aceste tabele pot fi omise, insa 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 functionarea corecta a acestuia; failed-jobs - este o tabela ce vine din stock cu Laravel, de asemenea necesara pentru functionarea corecta a acestuia; celelalte 18 tabele sunt tabelele de care se folosesc propriu-zis serviciile site-ului nostru pentru gestiunea datelor.

Tabelele comments si images nu au o relatie cu alte tabele, desi se pare ca ar trebui sa aiba, din motiv ca acestea au ca rol stocarea de date din mai multe tabele, avand o relatie polimorfica. Celelalte tabele lipsite de relatii cu alte tabele au fost instalate asa din stock-ul Laravel-ului, respectiv neavand necesitatea de a fi modificate pentru un lucru corect a framework-ului.

## Descrierea tehnologiilor utilizate

## Generalizare

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

* Este gratis;
* Suporta mai multe limbaje de programare;
* Automat identifica limbajul de programare din fisier, dar utilizatorul poate si el sa schimbe limbajul corespunzator fisierului in care lucreaza;
* Integrare git încorporată;
* IntelliSense;
* Posibilitatea instalarii unui numar mare de extensii;
* Prezenta indexarii.spatiului de lucru.

Inițial elaborarea site-ului Studbud era orientata pe JQuery si PHP curat, insa au aparut mari dificultăți in realizarea unor funcții mai complexe, si respectiv o mare complexitate in munca din cauza numărului mare de funcții ce necesitau o efectuare fără refresh la pagina (partea de Front-end) si 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 usura gestiunea informatiei din tabele, si mai mult in cazul in care tabelele au o relatie intre ele. De asemenea pe partea de securitatea ee necesita multa munca, si anume indeplinirea anumitor funcții in dependenta de tipul user-ului, ce permisiuni are acesta asupra site-ului etc. Evident, utilizarea PHP curat marea complexitatea realizarii acestor chestii cu mult.

Pentru problema de Front-end s-a ajuns la alegerea unui framework pentru reactivitatea site-ului nostru si o alegere s-a facut pentru Vue JS, care pare a fi foarte satisfacator si foarte flexibil fata de concurentii sai asemenatori (React sau Angular). Utilizarea JQuery-ului in cazul nostru necesita dupa fiecare gestiune a intormatiei din Back-end un upload de cod HTML pentru realizarea unor efecte dupa efectuarea acestor funcții, ce nu este foarte comod in cazul unui numar mare de funcții, plus. Deci, cea mai buna alegere este utilizarea anumitor state-uri, variabile, ce ar permite rendering-ul anumitor efecte in baza de informatia din Back-end (primita de asemenea in forma de variabile). Vue js-ul de asmenea ne pune la dispozitie si instrumente ca MIXINS, care ne permite reutilizarea unui cod JS in cazurile in care se necesita aceeasi solutie de rezolvare. Totodata Vue JS ne pune la dispoziție numeroase directive pentru diferite situatii, care reprezinta niste componente cu solutii deja gata (exemple: tabele de gestiune a informatiei pentru admin panel, hover directive etc).

Deci, in 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



Img 2‑1 Logo-urile: 1) Laravel; 2) Bootstrap; 3) Vue JS

## Partea de Front-end

### HTML & CSS

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

Aici s-a ales libraria Bootstrap (versiunea 5.1), deoarece este gratisa si contine 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, contine si el componente bootstrap facute sub tehnologia Vue JS, majoritatea fiind mult mai simplu de utilizat in Vue JS decat in versiunea originala clase de CSS. De asemenea, utilizarea librariei Bootstrap simplifica mult lucrul cu partea responsive a site-ului.

### JavaScript

Pentru o interfata easy-to-use este nevoie de o reactivitate intre actiunile user-ului si informatia ce se rendereaza pe web-site. Pentru reactivitate s-a ales sa se foloseasca 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 controlează nivelul fundamental al MAC-ului, 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 afaceri, 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 cadre. 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 cadru 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. Am vorbit deja despre asta la început. 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 copac. 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 in 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 blocul Cadru 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 cadrelor 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 copac.

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 .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.

* **Scr ipt (script).**

Iată 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 (Faceți clic, derulați, mutații DOM).

* **Styles.**

Fișierele de stil CSS sunt plasate aici.

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.

Pe alocuri s-a utilizat si libraria JQuery, si anume acolo unde se necesita lucrul cu CSS sau cu unele lucruri pentru care nu exista un echivalent in Vue JS iar scrierea lor in Vanilla JS era sa fie cu mult mai complexa decat echivalentul lor in JQuery (spre ex. adaugarea unei clase unui grup de elemente, sau selectarea unui element dupa denumirea clasei, id-ului etc.).

## Partea de Back-end

Pentru realizarea Back-end-ului, s-a ales a se folosi libraria PHP – Laravel 9.

Laravel este un proiect 100% open source. Acest lucru deschide oportunități excelente de personalizare, modificare și extindere, iar aici Laravel depășește multe cadre populare. Hub, GitHub și BitBucket.

Laravel folosește biblioteci partajate cu Symfony. Aceste cadre 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 cadre. 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 aplicații.

Laravel din cutie oferă un sistem de autentificare simplu și eficient. Oferă chiar și controlul accesului la resurse. Acest lucru vă permite să restricționați accesul utilizatorilor neautorizați la anumite resurse.

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.

# Procesul tehnologic

## Generalizare

Pana a incepe a descrie procesul de lucru cu tehnologiile si instrumentele utilizate in proiectul nostru, ar fi bine de mentionat procesul initial de instalare a acestor instrumente (se presupune ca XAMPP si PHP cea mai noua versiune este deja instalat).

Pentru a putea mediul Laravel pentru aplicatia noastra, este nevoie initial de instalat Composer. Procesul de instalare a acestuia se poate gasi pe: [*https://getcomposer.org/download/*](https://getcomposer.org/download/). Dupa instalarea acestuia putem recurge la instalarea proiectului nostru in Laravel cu urmatoarele comenzi in terminalul mediului nostru de programare VSCode:

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

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

Urmatorul instrument pe care l-am instalat este Vue.js. Pentru instalarea acestuia, tot in folderul cu proiectul in terminalul VSCode, este nevoie de introdus urmatoarele 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

Dupa aceste comenzi, in head-ul mentionat mai sus introducem urmatorul cod:

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

Uterior, ca pentru orice cod nou scris in Vue.js, actiunea acestuia sa se observe in timp real, este nevoie de introdus tot in terminal urmatorul cod - *npm run watch.*

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

1. *npm install @vue/composition-api* – instalarea composition – necesara pentru instalarea vuelidate-ului, in cazul daca folosim versiunea 2 de Vue.js.
2. *npm install @vuelidate/core @vuelidate/validator.*
3. *npm install vue bootstrap bootstrap-vue –* instalarea bootstrapVue.

Dupa instalarile date, in fisierul app.js din folderul resources (creat implicit cu instalarea proiectului Laravel) este nevoie sa incarcam dependentele (sau altfel spus – inregistrarea acestor librarii in aplicatia noastra) pentru librariile 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 Incarcarea dependentelor in fisierul app.js

Dupa cum s-a mentionat anterior, am utilizat librarila Bootstrap pentru CSS. S-a folosit conectarea Bootstrap-ului prin amplasarea link-urilor CDN la head-ul site-ului. Libraria Bootstrap necesita conectarea mai multor link-uri pentru a face functionala utilizarea acesteea. Astfel au fost conectate urmatoarele link-uri (conectarea Bootstrap-ului pentru functionarea sa deplina necesita si conectarea librariei 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.min.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, repsectiv, Bundle

## Front-end

### CSS

In cadrul realizarii CSS pentru site-ul nostru am utilizat libraria Bootstrap. Utilizarea Bootstrap-ului consta in atribuirea unor clase ce contin deja niste stiluri prestabilite de librarie, pentru diverse tag-uri HTML in scop-ul de a stiliza pagina web.

In cadrul site-ului nostru s-a utilizat mult sistemul grid oferit de Bootstrap. Plusul mare a acestui system este ca ne ofera si posibilitatea de a face concomitant si responsive site-ul ce-l cream, continand clase pentru 5 diverse marimi ale ecranului:

(Bootstrap, Grid system · Bootstrap v5.1, 2021)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **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

<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 pemtru sistemul grid

O alta clasa des utilizata in cadrul proiectului este flex-ul. Clasele pentru flex cu variantele responsive:

* .d-flex
* .d-inline-flex
* .d-sm-flex
* .d-sm-inline-flex
* .d-md-flex
* .d-md-inline-flex
* .d-lg-flex
* .d-lg-inline-flex
* .d-xl-flex
* .d-xl-inline-flex
* .d-xxl-flex
* .d-xxl-inline-flex

(Bootstrap, Flex · Bootstrap v5.1, 2021)

Pe langa clase, Bootstrap ofera secvente de tag-uri cu clase prestabilite in scopul crearii unor componente. Componente de acest gen utilizate in cadrul proiectului dat sunt:

* **Accordion** - element de colaps vertical pentru a afișa și ascunde conținutul prin modificări de clasă.
* **Alerte** – niste div-uri stilizate ce permite furnizarea unor mesaje pentru diverse actiuni ale user-ului.
* **Badge-uri** – componenta stilizat pentru etichete sau count componente.
* **Butoane** – componenta stilizat pentru butoane.
* **Card** - container de conținut flexibil și extensibil. Include opțiuni pentru anteturi și subsoluri, o mare varietate de conținut, culori de fundal contextuale și opțiuni puternice de afișare.
* **Carousel** - componentă slideshow pentru parcurgerea prin elemente-imagini sau diapozitive de text—ca un carusel.
* **Collapse** – o componenta alcatuita dintr-un button si informatia ce poate fi ascunsa sau aratata in functie de starea sa printr-un click pe button.
* **Dropdowns** – acelasi efect ca in cazul componentei collaspse, doar ca informatia gestionata nu are o pozitie relative, ci absoluta.
* **Liste** – componente stilizate pentru liste.
* **Navbar** – componenta bootstrap stilizata pentru navbar.
* **Placeholders** - pot fi utilizați pentru a îmbunătăți experiența aplicației dvs, atunci cand informatia nu este incarcata inca si este nevoie de tras atentia utilizatorului in ce parte aceasta va fi ulterior incarcata si totodata de a arata si momentul de incarcare a acestei informatii. Sunt construite doar cu HTML și CSS, ceea ce înseamnă că nu aveți nevoie de JavaScript pentru a le crea. Cu toate acestea, avem nevoie de un cod JS pentru a comuta vizibilitatea acestora.
* **Nav-uri si tab-uri** – componente stilizate pentru bare de navigare.
* **Modal** – un component similar cu card ce apare pozitionat deasupra altor elemente. S-au folosit mai mult in cazurile in care se necesita confirmarea unei actiuni de catre utilizator sau pentru galerie.
* **Offcanvas** - este o componentă bara laterală care poate fi comutata prin JavaScript să apară de la marginea din stânga, dreapta, sau de jos a viewport. Butoanele sau ancorele sunt folosite ca declanșatoare care sunt atașate la anumite elemente pe care le comutați, iar atributele de date sunt folosite pentru a invoca JavaScript-ul nostru. De obicei s-au utilizat in cadrul unor filtre sau a unui meniu ce poate fi afisat in cazul ecran-urilor mici (ex. sidebar la admin panel).

### JavaScript

Dupa cum s-a mentionat anterior, in proiectul dat partea cea mai majora in JS il ocupa framework-ul Vue.js.

In cadrul lucrului cu Vue.js in proiectul nostru vom mentiona despre urmatoarele aspecte:

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

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

In cadrul scriptului Vue ne-am folosit de urmatoarele proprietati ale componentului exportat Vue oferite din cutie de Vue.js:

* *Lifecycle* **Hooks**
* *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 *(Img 3‑1)*.

O descriere succinta pentru fiecare Hook se poate citi pe urmatorul link:

[*https://vuejs.org/api/options-lifecycle.html*](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 argumentilor dintr-o functie. Variabilele introduse in obiectul props vor reprezenta niste variabile ce se asteapta a fi trimise din componenta parinte de unde se va apela componenta curenta (ce detine props).

**Components –** reprezinta un obiect ce va include in el denumirile componentelor inregistrate local in componenta curenta.

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

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 returneaza toate state-urile sau variabilele ce pot fi supuse unor modificari in cadrul lucrului cu componena curenta.

(Vue.js, Lifecycle Hooks | Vue.js, 2020)

Diagram

Description automatically generated

Img 3‑1 Diagrama ciclului de viata pentru un component

Proprietatea **computed -** este asemenea unei functii, doar ca fara argumenti/parametri (nu este recomandabil utilizarea unor argumenti in functiile din computed), ce poate returna in timp real odata cu modificarea variabilei de care depinde o valoare compusa (poate returna chiar si doar valoarea proprie a unei variabile) dintr-o valoare a unei variabile din data si alte suplimente (pot fi constante sau chiar alte variabile din data).

computed: {

    errorTotal() {

      let error = 0;

      for (const key in 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 Returneazz eroarea totala ce depinde de erorile mai multor variabile, aceasta valoarea se va schimba concomitent cu vreo modificare a unei variabile de care depinde

**Watch –** reprezinta o optiune a Vue ce permite urmarirea schimbarii valorilor unei anumite variabile din obiectul *data* a componentei in scopul gestionarii unei actiuni cand aceasta modificare are loc. Aici in dependenta de ce variabila urmarim avem doua tipuri de watch-uri:

1. *Simpla*
2. *In adancime*

Watch-ul in adancime se aplica atunci cand urmarim un obiect sai arrauy intreg, adica cand urmarim concomitent modificarile pentru valorile tutoror cheilor obiectului sau array-ului urmarit. In cazul acestul watch este nevoie de mentionat ca dorim o urmarire in adancime, in caz contrar Vue ne va intelege ca urmarim o variabila simpla, ceea ce nu este prea legal in 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 in adancime

* **Directivele built-in in Vue.js**

In construirea website-ului cu Vue.js s-au folosit foarte des built-in directive ale framework-ului, si 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, in aceasta directiva putem atribui si metode/functii care returneaza o anumita valoare booloeana.

<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 in care ai lucrat

 </template>

 <template v-else-if="tip\_user == 'student1'">

 Nr. de institutii in care ai invatat

 </template>

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

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

<template>

  <ul class="students">

    <li v-for="student in 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 afisam.**

* **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. In 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 si src se transmit niste variabile dinamice, care se pot schimba pe parcurs de utilizare a site-ului

* **v-model.** Directiva este intr-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*. In exemplul de mai jos am legat valoarea inputului cu variabila name.search din obiectul data al exemplarului vue. La introducerea unor caractere in inputul dat se vor inscrie aceste caractere in 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, lasand doar doua puncte si denumirea actiunii 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 langa click, vue oferă pseudonime si pentru alte cele mai frecvent utilizate coduri cheie:

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

De asemenea se poate utiliza si keyCode atribute in definirea ce taste trebuie ascultate.

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

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

Totodata Vue ofera si 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 in Vue am utilizat sub-libraria Vuelidate – ce este un instrument pentru validarea input-urilor pe front end pentru Vue.js. Pentru a utiliza acest instrument, este nevoie ca in partea de script a fisierului .vue sa facem un import din sub-libraria Vuelidate inregistrata cu elementul de validare (tipul validatorului) ce ne intereseaza:

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

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

Ulterior, avand deja elementul required la indemana, putem sa determinam care input este necesar sa fie „required”, adica sa aiba cel putin un caracter. Pentru aceasta se creaza un obiect cu numele rezervat „validations”, in care introducem denumirea variabilei v-model din *data* (adica variabila care raspunde pentru input-ul interesat) si introducem tipul validatorului ce se presupune a fi deja importat:

  validations: {

    message: { required },

  },

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

Pe langa required, Vuelidate ofera si alte tipuri de validatori built-in:

* required
* requiredIf
* requiredUnless
* minLength
* maxLength
* minValue
* maxValue
* between
* alpha
* alphaNum
* numeric
* integer
* decimal
* email
* ipAddress
* macAddress
* sameAs
* url
* or
* and
* not
* **Inregistrarea componentelor si reutilizarea acestora in 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 si posibilitatea inregistrarii 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 Inregistrarea locala a componentelor

Componentele inregistrate se pot reutiliza de atatea ori, de cate ori este nevoie. Spre deosebire de inregistrarea acestuia in partea de script a codului, unde putem utiliza PascalCase, la apelarea componentelor in template este obligatorie utilizarea kebab-case pentru functionarea 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 inregistrate cu kebab-case cu utilizarea tag-ului suplimentar de inchidere

Dupa cum se poate observa in ultimele doua exemple de cod, apelarea componentelor este insotita de niste atribute. Atributele insotite de „:” reprezinta niste variable dinamice ce se pot schimba in componenta-parinte si ulterior aceste componente schimbate se transmit si in componenta-copil si in cazul existentei unei dependente fata de aceste variabile in componenta-copil, se va schimba si informatia dependenta de aceste variabile din compoenta-copil. Atributele insotite de „@” reprezinta niste functii ce se vor emite din componenta-copil in componenta-parinte si la emiterea acestor functii se vor realiza functiile din dreptul „=”, codul carora este determinat in componenta-parinte. Aceste atribute sunt strins legate de procesul de provide/inject din cadrul libarariei Vue.js care il vom descrie in paragraful despre ***Error! Reference source not found.***Pe langa componentele noastre, in cadrul proiectului nostru au fost utilizate si componentele BootstrapVue. Mai mult despre componentele Bootstrap se poate citi aici: [*https://bootstrap-vue.org/docs/components*](https://bootstrap-vue.org/docs/components)*.*

* **Provide/inject**

Este un proces de transmitere a datelor intre componente.

(Vue.js, Provide / inject | Vue.js, 2020)

Diagram

Description automatically generated

Img 3‑2 Principiul de lucru provide/inject intr-o ierarhie de componente

Transmiterea variabilelor din componenta parinte catre componenta copil se realizeaza simplu prin atribuirea atributelor insotite cu „:” ce corespund denumirilor variabilelor din props-ul componentei-copil niste valori a variabilelor din data a componentei-parinte, procesul numindu-se „*inject*”. Procesul de atribuire a valorilor in atributele tag-ului componentei-copil se poate vedea in *Fragment de cod 3‑20*. **!Este important ca denumirile atributelor din tag sa corespunda denumirilor de variabile din optiunea props a componentei-copil.**

Procesul invers la inject se numeste provide si consta in transmiterea datelor de la un copil catre parinte. Transmiterea de la copil se realizeaza prin functia $emit. Ca parametri in functie pe primul loc trimitem denumirea functiei care o vom citi in componenta parinte si ceilalti argumenti vor fi valorile ce le vom culege in componenta-parinte. Mai jos avem un exemplu in care pe un click pe button se vor transmite like-urile unui coment dintr-o iteratie 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 3‑21 Trimiterea datelor din componenta-copil (comments-items)

Dupa aceste manipulari, in componenta-parinte vom prinde functia cu denumirea corespunzatoare primului argument din $emit si o vom readresa intr-o noua functie declarata in componenta parinte 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 3‑22 Receptionarea datelor din componenta-copil in componenta-copil

In *Fragment de cod 3‑22* atributul insotit cu caracterul „@” reprezinta acea denumire de functie din componenta-copil pe care am setat-o ca prim argument in $emit. Partea de dupa „=” reprezinta functia din methods a componentei-parinte unde se va realiza manipularea cu datele receptionate.

methods: {

modal\_user\_list(users) {

      this.modal.users\_list = users;

    },

},

Fragment de cod 3‑23 Functia din componenta-parinte ce se ocupa cu manipularea datelor receptionate

**!Este important ca componentele sa fie legate intr-o anumita ierarhie pentru provide si inject lucru, in caz contrar datele nu vor putea trimise catre elementele ce nu se supun ierarhiei.**

* **Utilizarea axios in Vue.js**

Folosind Laravel, nu este nevoie de efectuat o instalare suplimentara a ajax-ului. Axios-ul este un instrument de comunicare direct dintre API si backend-ul. In cadrul proiectului nostru au fost utilizate doua tipuri de request-uri (ambele avand o tratare diferita a datelor) de API catre back-end:

1. *Cu date simple* – in acest caz ca argumenti in tipul metodei de transmitere a datelor se introduce doar route-ul api-ului si ca al doilea argument – informatia propriu zisa, sau request-ul cu alte cuvinte.
2. *Cu date ce contin si fisiere* – se necesita prelucrarea preventiva a informatiei, si anume se creaza un obiect de tipul FormData si in el se insereaza informatia ce necesita trimisa, dupa se urmareste aceleasi actiuni ca si in cazul trimiterii datelor simple, doar ca se mai indica al 3 argument - *headers: { "Content-Type": "multipart/form-data", }* – ce indica ca request-ul trimis contine si fisiere.

Informatia din back-end in axios se captureaza in promise-ul *.then,* unde ca argument se indica variabila ce va contine request-ul din back-end.

De asemenea am utilizat si asa metoda de promise-uri ca: *.catch –* pentru prelucrarea erorilor din back-end si *.finally ­–* pentru realizarea unor functii indiferent daca in back-end au aparut erori sau reqest-ul a fost prelucrat cu succes.

* **Lucrul cu JQuery**

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

* *.addClass* – pentru adaugarea unor clase sau clasa unui grup de elemente sau chiar unui element cu selector-ul corespunzator celui selectat.
* *.removeClass* – invers lui *.addClass –* stergeclasele date ca argument din elementele selectate.
* *.toggleClass* - combina *.addClass* si *.removeClass* in planul ca cand lipseste clasa indicata, aceasta este adaugata, iar daca este prezenta, atunci se sterge.
* *.each* – parcurgerea fiecarul element cu selector-ul dat ca argument.
* *.height* – calcularea inaltimii elementului selectat.
* *.width* – calcularea latimii elementului selectat.
* *.offset* – determinarea pozitiei pe y pentru elementul selectat.
* etc.

## Back-end - Laravel

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

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

* **Model**

Reprezinta cu alte cuvinte obiectul tabelei din baza de date. Un model se creaza in Laravel prin tastarea comenzii *php artisan make:model* ***denumirea\_modelului.*** In cazul proiectului nostru am utilizat si sufixul *-m* pentru majoritatea modelelor ce ne creaza si un fisier pentru structura tabelului**.** Aici avem posibilitatea sa cream atatea campuri, cate credem noi ca avem nevoie in cadrul tabelului, si sa indicam tipul si relatiile cu alte tabele. Toate tipurile disponibile pentru coloane in Laravel le putem gasi pe acest link: [*https://laravel.com/docs/9.x/migrations#available-column-types*](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 3‑24 Codul pentru crearea structurii tabelului user

Dupa ce structura tabelului este definita, utilizam urmatoare comanda in terminal ce ne creeaza tabelul respectiv in baza de date – *php artisan migrate.* Deseori aparea cazul cand era nevoie de o restructurare a tabelului. Pentru aceasta se apela la 2 metode:

1. Se modifica structura bazei de date la starea dorita si dupa in terminal se apeleaza una din comenzile: *php artisan migrate:fresh* sau *php artisan migrate:refresh*. Minusul metodei date este ca sterge toate popularile din baza de date.
2. Prin crearea unei migratii noi ce va cuprinde tabelul ce dorim si (dupa caz, daca nu dorim de exemplu doar denumirea tabelului sa modificam) campurile cu actiunile de modificare. Aceasta migratie se creeaza prin introducerea urmatoarei comenzi in terminal: *php artisan make:migration* ***denumirea\_migratiei***. Dupa efectuarea manipularilor descrise anterior apelam *php artisan migrate* in terminal.

Ce se refera la fisierele proprii ale modelelor – aici, cu ajutorul instrumentelor instalate in Laravel, s-au definit relatiile modelelor cu alte modele (de nu incurcat cu relatiile din structura fisierului de migratiei *(Fragment de cod 3‑24)*). Aceste definiri ne ajuta cu mult la simplificare codului ce necesita scrisa la interactiunea cu baza de date. In proiect s-au utilizat 3 din cele 11 relatii posibile in Laravel: *One to Many, One to Many (Invers) si Many to Many* – care se realizeaza prin indicarea relatiei *One to Many (Invers)* pentru toate modelele intre care exista aceasta relatie.

    public function activities()

    {

        return $this->hasMany(Activities::class);

    }

Fragment de cod 3‑25 Relatie One to Many (user-ul are mai multe activitati)

public function user()

    {

        return $this->belongsTo(User::class);

    }

Fragment de cod 3‑26 Relatie One to Many (Invers) pentru activitati (o activitate are mai multi 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 3‑27 Relatie Many to Many (un anumit User poate avea mai multi useri ca prieteni, precum si orice prieten al sau poate avea mai multi useri ca prieteni)

Pe langa definirea relatiilor, in modele am definit comod si alte functii de afisare a informatiei pentru diverse instante:

    public function hasRole($role)

    {

        if ($this->roles()->where('name', $role)->first()) {

            return true;

        }

        return false;

    }

Fragment de cod 3‑28 Functie ce-mi afiseaza daca un anumit user are vreun rol sau nu

Datorita utilizarii structurii POO in Laravel si instrumentelor sale native, putem cu mult simplifica codul ce trebuia sa-l scriem utilizand un PHP nativ.

* **View.**

Reprezinta de fapt un sablon cu html in care pot fi introduse variabile din back-end si instrumente din PHP, cum ar fi: iterarea unei liste cu *for*, o afisare conditionata cu *if* etc. Laravel se poate lauda cu modelul Blade, care contine instrumente pentru o structurare comoda si cat mai eficienta a view-urilor noastre, cum ar fi: @yield -> @extend + @section, @stack -> @push, @include etc. Afisarea variabilelor numai necesita de a fi insotita de *„echo”*, fiind nevoie de luat intre „{{}}” informatia 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.**

Esteca un mediu in care se realizeaza comuicarea cu baza de date si gestionarea informatiei. Aici primim un request din view-urile noastre, care de fapt reprezinta niste input-uri de la utilizator, si in baza acestor input-uri putem gestiona informatia din baza de date. Laravel ofera instrumente foarte puternice in ceea ce priveste actiunile CRUD, validarea request-ului obtinut, lucrul cu fisierele etc. Pentru a crea un controller este nevoie ca in terminal, in locatia 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 randul sau, avand nevoie de existenta modelelor pentru tabelele cu care dorim sa interactionam. Ambele instrumente au instalate implicit in ele protectia aplicatiei de potentiala injectie SQL. Cu ajutorul instrumentelor oferite de Laravel pentru CRUD, nu mai avem nevoie sa scriem un string lung pentru fiecare actiune CRUD catre baza de date, acestea deja avand optiuni pentru toate cazurile (multiple insert, order by, group by, where etc.).

Validarea request-ului in Laravel are deja instalata din cutie reguli pentru cele mai des intalnite situatii de validare. Aceste reguli le putem gasi pe urmatorul link: [*https://laravel.com/docs/9.x/validation#available-validation-rules*](https://laravel.com/docs/9.x/validation#available-validation-rules). In cazul unei erori de validare, se va face un return catre view-ul de unde a venit request-ul cu o lista in care sunt inscrise toare erorile de validare pentru acest request.

$validated = $request->validate([

            'text' => 'required',

            'type\_of\_post' => 'required',

            'note' => 'exclude\_if:type\_of\_post,post|required|numeric|min:1|max:5',

            'files\*' => 'image',

        ]);

Fragment de cod 3‑29 Exemplu de validare a input-urilor si salvarea request-ului validat in variabila $validated

Pentru a realiza comunicarea dintre input-urile user-ului cu Controller-urile corespunzatoare, precum si accesarea anumitor pagini pe un anumit URL este nevoie de un mecanisum de rutare. Laravel ne ofera acest mecanism, si pe langa asta detine si niste middleware din stock, oferindu-ne posibilitate si de creare unor middleware custom. In proiectul dat au fost utilizate asa middleware ca:

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

In Laravel sunt doua tipuri de rute: rute web si rute API, pentru fiecare din ele exista un fisier aparte in care se pot inregistra aceste rute. In proiectul nostru au fost utilizate ambele rute. Daca pentru web.php middleware-urile de mai sus pot fi apelate fara probleme, in api.php este necesara configurarea lui *„sanctum”.* Mai detaliat despre aceasta se poate gasi pe urmatorul link: [*https://laravel.com/docs/9.x/sanctum*](https://laravel.com/docs/9.x/sanctum).

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

# Proces de instalare

# Concluzii și propuneri

**Bibliografie**

Bootstrap. (2021). *Flex · Bootstrap v5.1*. Preluat de pe getbootstrap.com: view-source:https://getbootstrap.com/docs/5.1/utilities/flex/

Bootstrap. (2021). *Grid system · Bootstrap v5.1*. Preluat de pe getbootstrap.com: https://getbootstrap.com/docs/5.1/layout/grid/

Vue.js. (2020). *Lifecycle Hooks | Vue.js*. Preluat de pe vuejs.org: https://vuejs.org/guide/essentials/lifecycle.html#lifecycle-diagram

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 anexelor

# Anexe