UNIVERSITATEA DIN BUCUREȘTI
Facultatea de Matematică și Informatică
Domeniul de Licență Calculatoare și Tehnologia Informației

LUCRARE DE LICENŢĂ

Sistem de manageriere a voluntarilor dintr-un ONG

Coordonator științific:
Conf. dr. Popescu Marius

STUDENT:
Nistor Sabina-Iulia

București
2017

CUPRINS

1. INTRODUCERE

Cu toții traim vieți foarte ocupate și poate că uneori este greu să facem voluntariat. Cu toate acestea, beneficiile voluntariatului sunt considerabile pentru propria persoană și pentru comunitate în general. Această activitate ajută în reducerea stresului, interacțiunea cu persoane noi, învățarea unor abilități noi, îmbunătățirea comunicării și în dezvoltarea personală ce poate duce mai târziu la avansarea în carieră.

S-a demonstrat științific că voluntariatul are ca efect fericirea pură. Multe studii susțin că ajutorul acordat celor din jur conduce la sentimentul de fericire. Cercetătorii de la London School of Economics au studiat legătura dintre voluntariat și sentimentul de fericire pe un grup mare de americani. S-a remarcat că pe parcursul desfășurării mai multor activități de voluntariat, nivelul de fericire creștea considerabil, potrivit unui studiu în științe sociale.

Există patru mari beneficii aduse de voluntariat și anume: voluntariatul te face să interacționezi cu alți oameni, joacă un rol important pentru minte și corp, aduce un plus în dezvoltarea personală și oferă împlinire vieții, un scop.

Interesul meu pentru voluntariat a început încă de când eram în liceu din dorință de a avea o ocupație și de a-mi fructifica timpul cu alte activități. Prima mea interacțiune a fost cu o asociație ce avea ca scop infomarea despre riscurile internetului unde am observat îndeaproape organizarea și felul în care se realiza comunicarea într-un ONG. Dat fiind și scopul asociației ce viza internetul, mi-am dat seama că se puteau face îmbunătățiri la nivel de comunicare.

Mai mult decât atât, am făcut parte și din asociația facultății, ASMI și am interacționat cu o asociație mai bine organizată, reușind, totodată, să identific mai ușor nevoile unei asociații de voluntariat. Această organizație m-a făcut să conștientizez cât de importantă este munca de voluntar, munca de echipă și tot ce oferă ea de-a lungul proiectelor realizate.

Astfel s-a născut ideea de a crea o aplicație ce vine în ajutorul voluntarilor unei asociații de voluntari, pentru o mai bună coordonare și eficientizare a comunicării dintre project manageri și voluntari.

Acesta aplicație va reprezenta un instrument ce va putea fi folosit pentru gestionarea activitătilor de organizare și monitorizare a modului în care se desfășoară un eveniment sau un proiect de voluntariat. Totodată este oferită posibilitatea comunicării între voluntarii organizației, dar și cu cei cărora le vor veni în ajutor, acestea având ca efect final îmbunătățirea performanței organizației.

Cu toții am devenit mai mult sau mai puțin conștienți de infiltrarea tehnologiei in viețile noastre însa nimeni nu ii poate contesta ajutorul. Conectarea la lumea virtuala, care a devenit atat de la îndemână, are drept rezultat includerea sa ca unealtă de organizare si comunicare în timp real cu restul comunității.

Obiectivul acestei lucrări este de a aduce în același loc o organizație ce are un specific aparte și tehnologia ce își propune să îmbunătățească eforturile unui voluntar. Este adevarat ca există aplicații ce se ocupă cu managementul unor sarcini, însă niciuna nu tratează specific compozitia unei asociații de voluntari, precum și scopurile și nevoile acesteia. Organizațiile creează valoare adăugată în momentul în care reușesc să îmbine tehnologia cu noi modalități de a-și desfășura activitatea, atrăgând și mai mulți voluntari.

Poate că tehnologia tinde încă să se adreseze celor tineri, însă targheteaza orice fel de vârstă, la distanța de un minim efort de a-și crea un cont.

Este foarte utilă, spre exemplu, pentru studenții care vor să evadeze din rutină și să se implice într-o altă ocupație cu un scop bine definit, folosind telefoanele, laptopurile și aplicațiile cu care sunt cu toții deja familiari.

Manage Vol adoptă ideea de micro-volunteering ce este o formă de *virtual-volunteering*. Acest concept subliniază ajutorul adus, în realizarea unui task dintr-o asociație, prin intermediul tehnologiei, conectat la internet și eventual logat la o aplicație.

Odată expuse toate beneficiile practicării voluntariatului, este timpul să înțelegem partea tehnică și ceea ce își propune lucrarea de față. Aceasta se împarte în șase mari capitole ce vor ilustra componentele cele mai importante care vor fi discutate.

Prima parte a acestei lucrări are ca scop introducerea în tehnologiile folosite pentru a aduce la viață aplicația Manage Vol. Vom discuta despre limbajul PHP si ce întrebuințări are el în lumea IT și vom face trecerea spre partea web unde vom compara câteva dintre cele mai populare și bine realizate framework-uri PHP ale momentului. Tehnologia cheie care a ocupat cel mai mult din acest capitol reprezintă Laravel și constituie cel mai puternice instrumente pentru dezvoltarea web. Vom aminti toate componentele pentru care acest framework este atât de faimos și vom da și cateva exemple.

Urmatorul mare capitol face referire la fluxul aplicației și la povestea ei. Vor fi identificate publicul țintă, domeniul din care face parte și cerințele. Aceastea din urma vor fi împarțite în funcție de prioritatea lor și de asemenea, se vor analiza avantajele si dezavantajele platformei. Mai mult, vom afla specificațiile funcționale din punctul de vedere al viitorilor utilizatori ai aplicației.

Deoarece capitolul anterior a clarificat partea non-tehnică a sistemului, în urmatorul pas, se va vorbi despre arhitectura aplicației. Astfel, se va face o mică introducere a conceptelor modelului arhitectural MVC (model-view-controller). Aceste elemente vor fi discutate în funcție de felul în care au fost adoptate pentru dezvoltare. Capitolul cuprinde elementele cele mai interesante ale paginilor de vizualizare, explicația modelului de date și câteva soluții ale programării din partea de backend (din spate).

Următoarele capitole vor constitui o reflectare asupra dezvoltării aplicației Manage Vol.

Întrucât vom dori succesul unei asemenea aplicații, exact ca și în producție, ideile si îmbunătățirile ulterioare trebuiesc planificate din timp si vor dispune de un capitol separat.

Ultima parte va fi cea de concluzii în care vor fi reamintite părțile cheie ce au condus la această idee și felul în care a reușit să fie implementată.

2. TEHNOLOGII UTILIZATE:

2.1 Limbajul PHP

2.1.1 Noțiuni generale

Platforma Manage Vol a fost realizată în principal cu ajutorul limbajului de progaramare PHP.

PHP este un limbaj de programare creat și destinat inițial dezvoltării paginilor web. De-a lungul timpului semnificația PHP s-a schimbat, dacă la început limbajul avea ca seminificație **P**ersonal **H**ome **P**age, acum este cunoscut pentru **H**ypertext **P**re**p**rocessor. PHP este un limbaj foarte folosit, întrucât acesta este open-source, adică oricine poate avea acces la codul acestuia și îl poate modifica sau utiliza gratis pentru a-și îndeplini nevoile.

Scopul său inițial a fost acela de a fi înglobat cu limbajul HTML, care este un limbaj static, pentru a-i da acestuia din urmă logică și pentru a se putea implementa pagini web dinamice.

Ulterior, PHP a putut fi folosit ca limbaj de sine stătător, în modul CLI, pentru crearea diverselor aplicații, începand cu versiunea 4.3.

Creatorul acestui limbaj a devenit Rasmus Lerdorf în 1994, iar munca sa a fost dusă mai departe de echipa acestuia denumită astăzi *Grupul PHP*.

Cum PHP este întâlnit în combinație cu HTML, acesta poate fi folosit în diferite sisteme web de tip sablon, sisteme de manageriere a paginilor web sau în frameworkuri.

Există trei utilizări principale ale codului PHP:

Cea mai folosită este cea a scriptingului pe partea serverului. Există trei componente ce fac posibilă funcționarea scripting-ului pentru PHP: interpretorul PHP ce depinde de un plugin (componentă ce ofera o trăsătură specifică ce se poate schimba) al serverului ce îndeplinește si CGI, serverul web propiu-zis si un browser (navigator web).

Se poate realiza si script PHP fără server si browser web, doar pentru linia de comandă. Întrebuințarea acestor tipuri de aplicații își au locul in soft-uri simple de procesare a textelor.

O altă utilizare, mai puțin cunoscută, a acestui limbaj poate fi cea de a realizare a aplicațiilor de tip Desktop. În acest sens, se poate folosi extensia PHP-GTK pentru a realiza interfața grafică.

Limbajul PHP este întalnit pe o multitudine de sisteme de operare precum Linux, Microsoft Windows, Mac OS, întreținând majoritatea serverelor web existente din zilele noastre. PHP se dovedește a fi un limbaj extrem de fiabil, modelându-se ușor după nevoile fiecărui dezvoltator, având liber în alegerea sistemului de operare sau a serverului web.

Este realizabilă și introducerea conceptelor de progaramare orientată pe obiect sau a programării procedurale.

Așa cum s-a menționat mai sus despre interpretorul PHP, acesta funcționeaza uitându-se doar dupa codul determinat de tagurile *<?php* și *?>* iar restul este ignorat. Această delimitare face codul PHP foarte ușor de inclus într-o multitudine de documente.

Un exemplu de cod PHP combinat cu HTML este clasicul program *Hello World*:

```
<DOCTYPE HTML>
<html>
      <head>
         <title> Hello</title>
      </head>
      <body>
          <?php
```

```
            echo „Hello World”;
        ?>
    </body>
</html>
```

În ceeea ce privește sintaxa, PHP este un limbaj destul de simplu de învățat, limbajul fiind similar cu C, pe partea de instrucțiune, iar structura funcțiilor este ca și cea din C++, C#, Java sau Perl. Pe partea de programare orientată pe obiect, PHP are clase cu multiple instanțe, iar accesul proprietăților sau a metodelor se realizează cu sageata „->”.

În lucrarea de față, este folosit PHP 7, fiind ultima versiune și de altfel cea mai completă, mărind astfel performanța a ceea ce este dezvoltat.

2.1.2 PHP – introducere în framework-uri

PHP este atât de raspandit încat ocupă 80% din site-urile web existente. Fie că au fost create platforme e-commerce (comerț electronic), că au fost produse unelte de management, aplicații Desktop, sau rețele de socializare, foto-galerii și multe alte template-uri dinamice, PHP a putut fi pus in practică sub multe idei creative. Cele mai de succes platforme construite cu ajutorul limbajului PHP sunt de asemenea și cele mai folosite, în prezent, de majoritatea dintre noi, exemple precum: Facebook, Wikipedia, Wordpress, Tumbl, Yahoo și multe altele au la baza PHP.

Cum PHP a evoluat destul de mult încă de la primele linii de cod introduse în staticul HTML, în prezent sunt alese frameworkuri, pentru a dezvolta aplicații web cât mai complexe si pentru nevoia de a dezvolta de la zero într-un mod natural mai structurat.

Conceptul de framework a fost introdus ca o unealta de analizare într-un context de dezvoltare. Este folosit pentru a face distincții conceptuale si a organiza idei. Un framework software este un soft universal, reutilizabil care furnizeaza funcționalități ca parte a unor platforme sau ca soluții pentru problemele de dezvoltare software.

Sunt disponibile o multitudine de frameworkuri pentru dezvoltarea web cu PHP, fiecare cu avantajele și dezavantajele aferente. Alegerea framework-ului potrivit poate fi grea uneori, de aceea vor fi prezentate câteva dintre framework-urile de succes ale PHP. Acestea sunt framework-uri menținute active, populare în comunitatea de dezvoltatori, care permit dezvoltarea aplicațiilor de orice mărime si complexitate.

Symfony este un framework ce duce PHP la un alt nivel atunci când vine vorba despre a dezvolta un proiect foarte complex de tip enterprise. Conceptul său constă într-un set mare de componente reutilizabile si dispune de o comunitate foarte activă de dezvoltatori.

CodeIgniter este un framework ce oferă foarte multă libertate programatorilor. Este folosită arhitectura MVC, dar structura aceasta nu este una obligatorie adaptându-se și altora mai vechi, întrucât este un framework care este disponibil încă din anul 2006.

Yii este și el printre cele mai vechi framework-uri PHP apărute, iar odata cu *Yii2* a devenit și printre cele mai rapide comparativ cu celelalte. Se folosește de cel mai nou set de caracteristici pentru dezvoltarea unor aplicații performante ce au nevoie de fluxuri de dezvoltare complexe.

Alte framework-uri de succes sunt si *Phalcon*, recunoscut pentru rapiditate, *Slim*, *CakePHP*, *Zend* și multe altele.

2.2 Framework-ul Laravel

Framework-ul PHP cu cea mai mare popularitate, și totodata, cel ales pentru această lucrare, este **Laravel** cu versiunea **5.3** lansată in 23 august 2016.

În cele ce urmează, vom prezenta principalele componente ale acestui framework.

- Composer

Spre deosebire de alte framework-uri, Laravel deține o mulțime de librării si pachete specifice care necesită o organizare foarte bună.

Aici intervine **Composer**-ul. Acesta reprezintă un manager de dependințe pentru PHP. Cu alte cuvinte, va atrage și va pune la un loc toate bibliotecile și dependințele necesare și le va gestiona pe toate într-un singur loc. Acest concept a fost inspirat de *npm* al *NodeJs-ului* sau *Bundle*r al limbajului *Ruby*.

- Eloquent ORM

Instrumentul **Eloquent ORM** (object-relational mapping) oferă o implementare simplă, activă și elegantă pentru lucrul cu bazele de date în Laravel. Fiecare tabel din baza de date are drept corespondent un „model", adică o clasă, iar instanțele obiectului sunt legate de coloanele din baza de date. Toate acestea, pentru o interacțiune mai ușoara cu baza de date și pentru simplificarea proceselor de interogare și inserare. In același timp sunt oferite metode interne pentru a impune constrângeri.

- Query Builder

Pe langă Eloquent, există ca alternativă **Query Builder**, ce furnizeaza un acces mai direct la baza de date. Query Buider-ul Laravel-ului oferă o serie de clase si metode capabile de a construi în mod programatic interogări.

- Rutarea din aplicație

O altă trăsătură specifică este modul de rutare al aplicației produse prin Laravel. Rutarea inversă conturează o relație dintre legaturile si rutele efective din browser ale paginilor web. Astfel, este posibil ca modificările anterioare ale rutelor să fie propagate automat în legăturile relevante ale acestora. Atunci când legaturile sunt create prin folosirea unor denumiri, identificatorii uniformi de resurse (URI) sunt creați automat de Laravel. Aceste URI-uri sunt deseori aceleași cu URL-urile (localizatorul uniform de resurse).

- Blade

Laravel conține, ca orice alt framework specializat pentru aplicații web, un sistem web de șabloane numit **Blade**. Acesta este un instrument simplu, dar foarte puternic și, spre deosebire de alte instrumente pentru șablonare, Blade nu restricționează folosirea codului

PHP în paginile web. Mai mult, toate paginile de tip Blade sunt compilate cu PHP și stocate în memoria cache până când se găsește o modificare. Fișierele (views) de tip Blade se salvează cu extensia *.blade.php* și sunt de obicei stocate in directorul *views* din *resources* al structurii aplicației.

Principalele beneficii ale folosirii Blade sunt conceptele de: moștenire a șabloanelor si sectiunile. Pentru a putea face mai clare aceste notiuni, ne vom folosi de structura standard a unei pagini web mai complexe.

În primul rând se alege o pagină șablon cu rol principal de pagină „mamă" și este reprezentată de o pagină simpla HTML. În aceasta se pot insera directive precum **@section** ce delimitează o secțiune ce permite includerea de cod necesar doar în anumite pagini, așa cum se înțelege și din nume, și **@yield** folosit pentru a afișa conținutul unei sectiuni.

Odata ce șablonul a fost definit, acum se pot crea pagini web de tip „copil" ce *moștenesc* acest sablon. Pentru a moșteni se folosește directiva **@extends** pentru a specifica ce șablon trebuie moștenit. Fisierele care extind layout-ul de tip Blade permit injectarea de conținut în secțiuni cu ajutorul @section.

Pentru a afișa, în pagina de vizualizare, valoarea unei variabile, Blade permite cuprinderea acesteia între acolade {{ $variabila }}.

Pe langa moștenirea de șabloane si secțiunile, Blade oferă și o modalitate mai scurtă de a scrie instructiuni ale codului PHP într-o pagina web. Aceste structuri condiționale sau repetitive realizează un lucru mult mai curat și clar cu integrarea PHP în HTML. Scurtăturile sunt marcate la fel ca și directivele: @if, @elseif, @for, însă odată deschise, trebuie marcată și încheierea lor: @endif, @endfor etc.

- IoC

Laravel dispune și de conceptul de **IoC** (Inversion of Control) ce constă în rezolvarea dependințelor într-un mod cât mai convenabil. Sunt descrise obiectele în container și de fiecare dată când o clasa este rezolvată, dependițele sunt injectate sau realizate automat.

- Migrările

Migrările sunt o alta caracteristică importantă în suita de beneficii ale Laravel-ului. Migrările oferă un sistem de control al versiunilor pentru schema bazei de date. Ele fac posibilă menținerea unei evidențe a felului în care baza de date a fost creată sau modificată de-a lungul timpului. În lipsa acestora, pentru a avea o idee despre construcția aplicației ar trebui ținută o copie a tot ceea ce a fost dezvoltat în aceasta.

Migrările sunt legate de așa-zisa „fațadă", o interfață statică disponibilă în containerul de servicii al Laravel-ului, numită *Schema Builder*, cu ajutorul căreia se poate crea ușor schema bazei de date configurându-se astfel și legătura aplicației cu aceasta din urmă.

Există o tabela specială care stochează fiecare migrare în parte cu data creării acestora numită „migrations". Pentru a crea o migrare, se folosește în linia de comandă: „php artisan make: migration creaza_numele_tabelei". Inspectând o migrare îndeaproape, observăm că există 2 metode pereche, up() și down(). Cu ajutorul metodei up() se definesc de obicei câmpuri și tipul acestora sau tabele și are drept corespondent crearea, iar down() are rolul de a reface sau a face rollback pentru tot ce a realizat funcția up(). Pentru a rulă în final toate migrările realizate, se execută comandă „php artisan migrate".

- Seeders

Laravel include și „seeding", o metodă simplă de introducere a unor date de testare în baza de date cu ajutorul unor clase specifice. Aceste clase pot avea orice denumire, însă trebuie să întrunească o anumită convenție pentru a putea fi recunoscut de framework: numele clasei trebuie să fie precedat de sintagma „Seeder". Pentru generarea de „seedere" trebuie executată comanda: „php artisan make:seeder NumeTabelaSeeder".

- Tipuri de testare

Framework-ul de față dispune și de testarea unitară ce se realizează foarte ușor din linia de comandă și pune la dispoziție, în

fișier, metode ajutătoare foarte convenabile care fac testarea cât mai relevantă.

Pe langă testarea unitara in care atenția pică pe porțiuni mici si izolate de cod, există si teste „feature” care au ca scop arii mai mari de cod. De exemplu, cum reușesc diverse obiecte să interacționeze între ele sau cum funcționeaza anumite request-uri ale aplicației.

- Mașina virtuală ce rulează local

Odata cu Laravel, mediul de lucru devine ușor si intuitiv datorită instrumentului pe care îl are la dispoziție, *Homestead*, un soft ce ține de *Vagrant* și are scopul de a produce un mediu de dezvoltare local.

Deși Laravel vine cu Homestead în mod implicit, în lucrarea de fața, vom folosi **Valet**. Acesta a fost specific, inițial, utilizatorilor de Mac OS, însa acum poate fi adaptat si pentru Windows. Valet necesită doar instalarea PHP și a unui server de baze de date direct pe mașina locală, având drept avantaje folosirea minima a resurselor și eficientizarea eforturilor, nefiind nevoie de instalarea *Apache*, *Vagrant* sau *Nginx*. Accesarea aplicației, dezvoltate cu ajutorul Valet, se face în navigatorul de internet cu „numeleAplicației.dev”.

- Validarea în Laravel

Laravel deține mai multe căi și abordări de a realiza validarea pe datele din aplicație. Framework-ul pune la dispoziție un *trait* „ValidateRequests” (asemanator unei clase abstracte) ce are ca soluție convenabilă realizarea validării printr-o serie de cateva reguli. Acest tip de validare este realizată prin crearea legăturilor interne a unor evenimente „listener”, rezultând în invocarea automată a metodelor de validare ce vor avea ca rezultat mesaje în paginile web ce conțin formulare, după caz.

- Autentificarea cu Laravel

Recunoașterea framework-ului Laravel este mai ales adusă de simplitatea și în același timp siguranța implementării **autentificării**. Analizând majoritatea autentificărilor aplicațiilor web de pe piață, s-a constatat că la bază sunt la fel, deci foarte rare sunt cele care necesită o modificare suplimentară. Astfel, Laravel a venit cu o configurație ce

constă în realizarea paginilor web, a rutelor, și a logicii din spate ce permite o autentificare standard suficientă cu câmpuri precum numele, e-mailul si parola.

Facilitățile serviciului de autentificare ale Laravel-ului, au la bază doua concepte: „guards” și „providers”. Guards se ocupă cu manifestarea autentificării la fiecare request, cu ajutorul unei stocări temporare și cu cookie-urile. Providers au rolul de a comunica cu stocarea permanentă, cu baza de date, prin Eloquent.

Deși această autentificare se realizează printr-o singură linie de cod in linia de comanda: „php artisan make:auth”, aceasta este facută de un framework specializat pentru aplicații web și care, ca oricare altul din acest domeniu, oferă o atenție deosebită securității. Parola este criptată si există chiar și o coloană *remember_token* pentru utilizatorii care doresc să fie reținuți în aplicație.

2.1.3 Artisan CLI al framework-ului Laravel

O altă componentă de baza a framework-ului Laravel, poate cea mai importantă, este posibiltatea rulării comenzilor din linia de comandă, iar aceste comenzi sunt denumite colectiv **Artisan**. Fiecare comandă plasează dezvoltatorul la baza aplicației create cu ajutorul Laravel, iar Artisan vine implicit cu acesta. Efectele rulării acestor comenzi au rolul de a ușura munca de dezvoltare, creându-se fișiere noi, uneori chiar și scheletul specific de cod care se dorește să fie furnizat.

Funcționalitățile si capacitățile uneltei Artisan se pot extinde prin modificarea comenzilor sau chiar realizarea altora noi, care ajută la automatizarea unor sarcini din procesul de dezvoltare.

Comanda standard de creare, folosind Artisan, este de forma ”*php artisan make numele_componentei numele_oferit_componentei*”. Analizand această comandă, *php* are rolul a spune teminalului să folosească PHP, *make* este acțiunea ce se dorește a fi executată, *numele_componentei* este subcomanda, iar ce urmează sunt de obicei

parametrii precum numele ce se doreste să fie dat componentei respective.

2.3.2 Realizarea conexiunii cu baza de date

Framework-ul Laravel reușește să facă interacțiunea cu baza de date foarte ușoara, fie ca este vorba să se folosească Query Builder-ul fie că este folosit elementul Eloquent ORM. În prezent, Laravel suportă conexiunea cu 4 tipuri de baze de date:

- MySQL
- Postgres
- SQLite
- SQL Server

Pentru configurarea bazei de date a aplicației este nevoie ca dezvoltatorul sa se plaseze in fișierul *database.php* din directorul *config* unde se vor alege conectorii ce trebuiesc folosiți si sa se specifice ce baza de date o sa joace rolul implicit în aplicație.

```
'mysql' => [
        'driver' => 'mysql',
        'host' => env('DB_HOST', '127.0.0.1'),
        'port' => env('DB_PORT', '3306'),
        'database' => env('DB_DATABASE', 'forge'),
        'username' => env('DB_USERNAME', 'forge'),
        'password' => env('DB_PASSWORD', ''),
        'charset' => 'utf8',
        'collation' => 'utf8_unicode_ci',
        'prefix' => '',
        'strict' => false,
        'engine' => null,
    ],
```

În mod implicit, Laravel foloseste *Homestead* pentru a face posibilă dezvoltarea pe mașina locală. Pentru a adapta în funcție de

propriile nevoi mediul de lucru, putem modifica cu ușurință documentul *.env*. Aici are loc conectarea bazei de date prin atribuirea valorilor potrivite variabilelor urmatoare:

`DB_CONNECTION=mysql` - este variabila ce stabilește ce fel de bază de date este folosită, iar în acest caz este vorba despre MySQL si realizează conexiunea cu aceasta

`DB_HOST=127.0.0.1` - reprezintă variabila ce preia valoarea IP-ului local al fiecărei mașini și care face posibilă rularea în browser a aplicației fără nevoia unui server

`DB_PORT=3306` - această variabilă indică portul prin care se poate accesa baza de date prin mașina virtuală (Homestead) si se pot executa comenzi din afara acestei mașini virtuale

`DB_DATABASE=volunteeringdb` - este indicat numele schemei bazei de date a aplicației din MySQL

`DB_USERNAME=root` - identifică numele utilizatorului din baza de date MySQL

`DB_PASSWORD=` - în cazul în care este securizată, se stabilește și parola care acceseaza baza de date din MySQL

Odată realizată conexiunea cu baza de date, crearea tabelelor se poate face în mod automat, foarte simplu. Din linia de comandă se vor crea migrări ce vor fi ulterior completate în funcție de componența dorită a tabelelor.

Clasa *Schema* a Laravel-ului oferă o modalitate accesare a bazei de date și de manipulare a tabelelor. Funcționează bine cu toate bazele de date suportate de Laravel și are un API unificat în toate aceste sisteme.

Pentru a crea o nouă tabelă în baza de date este folosită metoda create(), astfel:

```
Schema::create('users', function($table)
{
    $table->increments('id');
});
```

În exemplul de mai sus, avem tabelul users ce conține un singur câmp definit ca fiind de tipul *increments.* De-a lungul dezvoltării aplicației pot fi folosite și metode de adăugare a unor coloane noi iar acest lucru este posibil prin mdetoda *table(),* asemănătoare cu cea de *create()*. De asemenea, este nevoie și de existența unei metode de ștergere, iar aceasta este disponibilă prin *drop().*

2.3 HTML - Hypertext Markup Language

Plecând de la conceptul de aplicație web, un soft ce se bazează pe modelul client-server utilizând internetul, și concentrând atenția asupra parții de client, se constată că în produsul final partea cea mai relevantă este adesea interfața grafică. Pentru a crea o interfață grafică cât mai prietenoasă, pentru o aplicație web, se va folosi tehnologia HTML.

HTML nu se proclamă ca fiind un limbaj de programare, ci o tehnologie ce reușește să intepreteze scopul și structura unui document web.

2.4 BOOTSTRAP

Bootstrap este indubitabil cel mai bun si recunoscut framework web, disponibil tuturor dezvoltatorilor care vor să realizeze o interfață grafică cu un minim de efort. Această tehnologie oferă o gama largă de piese de cod foarte bine organizate, testate si gândite, urmând întocmai convențiile interfeței utilizatorului, ce sunt cel mai des folosite. Bootstrap are la baza structura HTML, stilurile bazate pe CSS și un bonus opțional de extensii de JavaScript.

Bootstrap a fost inițial cunoscut sub numele de „Twitter Blueprint” și a fost creat cu scopul de a oferi o consistență în interiorul uneltelor de dezvoltare. S-a dorit să fie un limbaj universal care să

combată orice lipsă de compatibilitate din limbajele și framework-urile destinate dezvoltării web.

2.5 Alegerea mediului de dezvoltare

Pentru a realiza dezvoltarea acestei aplicații, a fost ales editorul de text ***Sublime text*** ce poate fi de tip cross-platform (disponibil pentru mai multe platforme). A reușit să fie atractiv prin disponibilitatea sa de a suporta mai multe limbaje de tip markup și de programare dar și de a extinde anumite functionalități cu ajutorul unor plugin-uri. Oferă autocompletare, se poate naviga ușor prin structura proiectului si este o alternativă mai ușoara a unui IDE (mediu integrat de dezvoltare) cu mai puține resurse.

3. DESCRIEREA APLICAȚIEI

Exact cum a mai fost menționat, ideea aplicației Manage Vol a luat viata prin identificarea unei nevoi care exista pe piață, și anume, prezența unei soluții tehnologice care să preia din problemele de organizare și comunicare ale unei organizații de voluntariat. Astfel a fost stabilit un proces în care a fost nevoie de parcurgerea unor pași standard pentru a avea un scop cât mai precis si bine pus la punct.

Primul pas, alături de stabilirea necesității unei astfel de platforme, a fost consultarea mai multor asociații din diverse domenii, a voluntarilor obișnuiăi cât și a unor voluntari care ocupă poziții cu o responsabilitate relevantă, întrucât ei se întruchipează ca posibili clienți ai aplicației Manage Vol.

3.1 Aria de interes a aplicației

Așa cum reiese și din numele aplicației, Manage Vol, își propune să servească drept platformă de organizare și pentru unii utilizatori să fie un suport în care să poată prelua conducerea și responsabilitatea asupra unui proiect din viața reală cu sarcinile aferente lui. Pe langă acestea, aplicația oferă accesul către o monitorizare prietenoasă a activităților ce servesc proiectelor în desfășurare.

Deși există foarte multe platforme care se ocupă cu organizarea timpului și a activităților de interes, precum *Trello, Taskworld* sau *Portofolio* etc, Manage Vol are o structură specifică și relevantă pentru ierarhia componentelor și entităților dintr-o asociație de voluntari.

Odata cu evoluția tehnologiei, aceasta a reușit să patrundă tot mai mult, făcându-se tot mai utilă prin instrumentele pe care le oferă în timp real. Este evident că aplicația de față își are drept **public țintă** organizatorii și cei ce ocupă funcții de conducere într-un ONG, voluntarii din cadrul acestuia și poate fi, chiar, un punct de start pentru

oricine își dorește să îți deschidă propria organizație nonguvernamentală.

Categoria în care se poate încadra această aplicație este aceea de *Management* al unei organizații, în acest caz una nonguvernamentala, deoarece este realizată pentru ținerea evidenței activităților și a atribuțiunilor ce sunt implicate în acest proces. Angajamentul utilizatorilor în aplicație se bazează pe datoria acestora de a se menține informați și vine odata cu dorința lor interioară de a se implica în evenimentele deschise de asociația la care s-au înscris. De asemenea, aplicația mai poate fi încadrată și în sfera de *Comunicare*, deoarece există posibilitatea voluntarilor de a împărtăși informații in ceea ce privește conținutul unui eveniment.

3.2 Prioritizarea sarcinilor

În procesul de dezvoltare apar adesea cerințe neprevăzute, neconcordanțe sau alte probleme care pot fi combătute printr-o organizare a pașilor de dezvoltare. Această organizare poate fi posibilă printr-o prioritizare a cerințelor viitorilor clienți și a nevoilor lor. Pentru ca acestea să poata fi înțelese cat mai bine de la bun început, se va întocmi o listă.

Odata ce această listă este completă și clară pentru creator, este important ca cerințele și sarcinile stabilite să suporte o prioritizare în ciclul de dezvoltare al aplicației.

În urma unei căutări a metodei potrivite de prioritizare a sarcinilor, a fost gasită **Metoda MoSCoW**. Ea a fost creată din nevoia de a soluționa cât mai eficient, o problemă ce necesită încadrarea într-un anumit interval de timp, iar anumite sarcini au nevoie de un anumit grad de importanță în procesul de dezvoltare. Această metodă este folosită atât în domeniul de dezvoltare software dar și în domenii precum: analiza de afaceri, organizarea de evenimente.

Numele acestei metode are ca semnificație:

- *Must* (trebuie) – este categoria ce conține cerințele obligatorii și definitorii pentru aplicație
- *Should* (ar trebui) – în această categorie se încadrează sarcinile ce se plasează pe o poziție secundară ca ierarhie în procesul de dezvoltare, și ele sunt necesare însă doar după ce sunt rezolvate cele din lista „trebuie”; aceste cerințe ar trebui să fie prezente, doar dacă este posibil
- *Could* (ar putea) – cerințele care se află în această categorie sunt niște cerințe dorite însă nu imperios necesare; în același timp au rolul de îmbunătăți experiența utilizatorilor aplicației
- *Would* cu extensia înțelesului de „would like but won’t have” (ar fi de preferat dar nu se va întampla) – categoria în care se află acele idei de viitor care pe lângă faptul că nu sunt necesare, pot fi doar simple îmbunătățiri ce se află în plan pentru viitoare versiuni ale aplicației; ele nu se încadrează în *timpul dedicat pentru creare* și nu vor fi prezentate în cadrul livrării aplicației

De asemenea, „o” are ca scop ușurarea citirii acestei metode, din moment ce numele este format cu inițialele categoriilor incluse.

Astfel, schema organizării aplicației Manage Vol, în funcție de această metodă va arata după cum urmează:

M (trebuie)	☑ Sistem de autentificare prin email și parolă a voluntarilor ☑ Administrator al asociației ce poate crea sau șterge mai multe entități în asociație ☑ Interfața placută de utilizat, intuitivă și atractivă ☑ Crearea unui flux care pleacă de la vizualizarea proiectelor, la alegerea unuia dintre ele și înscrierea in acesta

	☑ Posibilitatea accesării aplicației de pe telefon, adică să fie un design responsive ☑ Baza de date trebuie realizată într-un mod cat mai solid și în același timp specific structurii unui ONG ☑ Existența unei componente de search(cautări) pentru accesarea proiectului de interes ☑ Statistici ce relevă activitatea voluntarilor din asociație
S (ar trebui)	✓ Secțiune de comentarii în interiorul fiecărui proiect pentru feedback(păreri) imediat în ceea ce privește descrierea acestuia din partea voluntarilor ✓ Posibilitatea introducerii unei imagini sugestive în procesul de creare a proiectului ✓ Existența unui sidebar(bară laterală) care să facă navigarea mai ușoara ✓ Formular unde se pot trimite sesizări și alte opinii în ceea ce privește asociațiile și aplicația ✓ Posibilitatea scrierii într-un mod fomatat a proiectelor
C (ar putea)	🖂 Existența rolului de Manager de Proiect ce ar putea avea în plus posibilitatea de creare de taskuri(sarcini) pentru proiectul pe care îl coordoneaza 🖂 Pagina personalizată a fiecarei asociații
W (nu sunt încă realizate)	× Secțiune de anunțuri × Imbunatățirea vizualizării mai multor date din aplicație, precum voluntarii, sarcinile × Sistem de organizare si filtrare a

	comentariilor în aplicație × Calendar al ilustrării sarcinilor × Posibilitatea Managerului de Proiect de a nota activitatea celui mai activ voluntar

Este de precizat faptul că, adesea, noțiuni precum: *trebuie, ar trebui, ar putea* sunt interpretabile, astfel că această clasificare este totuși, una subiectivă și există totodată riscul ca cerințele să fie interpretate și în alt mod.

De asemenea, ideile din cadrul categoriei „Would" vor fi reamintite și, în același timp, mai amplu descrise într-o secțiune urmatoare din cadrul acestei lucrări.

3.3 Analiza SWOT

Denumirea analizei SWOT este, de fapt, un acronim pentru conceptele din engleza: Strenghts (Puncte tari), Weaknesses (Puncte slabe), Opportunities (Oportunități) si Treats (Amenințări). Cele patru categorii se mai pot clasifica și în cruce, astfel că, punctele tari și oportunitățile sunt acele elemente care sunt benefice și ajută în procesul de dezvoltare iar punctele slabe și amenintările pun în pericol ducerea la bun sfârșit a ceea ce este propus. În același timp, punctele tari și slabiciunile fac referire la sursa internă, cu alte cuvinte la dezvoltator, iar oportunitățile și amenințările au legatură directa cu mediul extern, reprezentând factori ce nu țin de procesul de dezvoltare, dar trebuie avuți in vedere.

Scopul unei analize SWOT este acela de a se reuși proiecția in viitor a tuturor aspectelor ce pot interveni în crearea unui nou produs și livrarea lui pe piață. Ideea principală este aceea de a contrui pe baza punctelor tari, de a se încerca eliminarea punctelor slabe, de a se încerca valorificarea oportunităților și nu în ultimul rând, elinimarea amenințărilor.

S (Strenghts – Punctele tari)	W (Weaknesses – Puncte slabe)
• Dispunerea de cunoștințe de web și de baze de date • Aspect plăcut realizat cu Bootstrap, ce permite folosirea aplicației și pe telefon • Folosirea PHP cu Laravel, tehnologii specializate pentru web • Baza de date specifică structurii unui ONG	• Lipsa experienței de a lucra cu PHP și framework-ul Laravel • Timpul alocat învățării
O (Opportunities - Oportunități)	**T (Treats – Amenințări)**
• Crearea de parteneriate cu asociații mari de voluntariat • Captarea atenției unor sponsori ce ar putea să ajute menținerea si creșterea aplicației	• Aplicația implică un public taghetat, nu are o arie atat de mare de utilizare • Este nevoie de a avea o campanie de publicitate pentru a aduce asociații și implicit voluntari (utilizatori) în aplicație – implică costuri • Dezinteresul asociațiilor de a folosi o aplicație nouă

3.4 Specificații funcționale

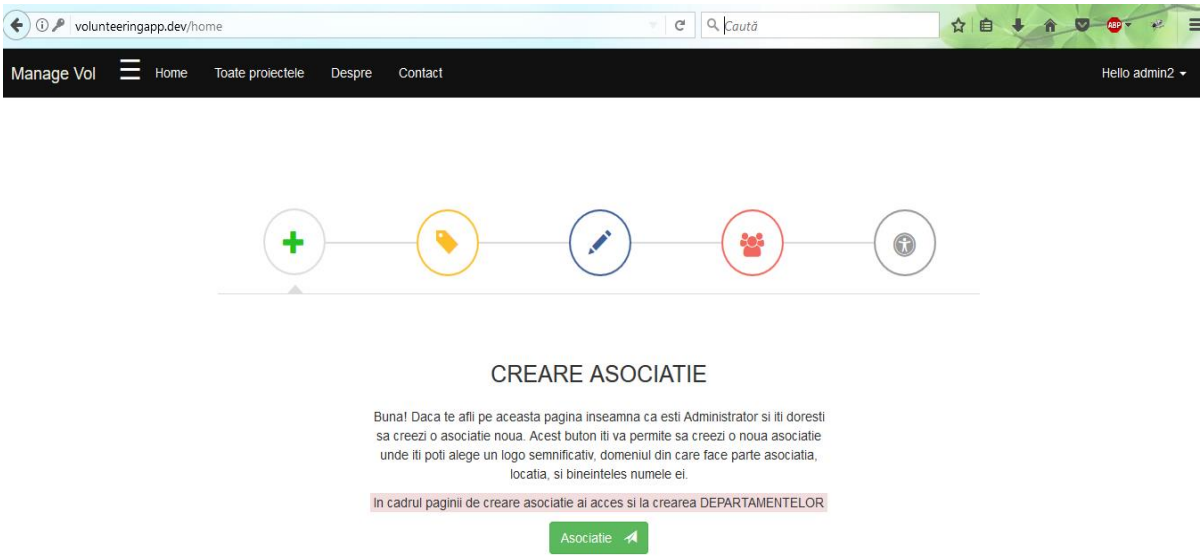
Identificarea unor scenarii de utilizare din punctul de vedere al utilizatorilor este adesea benefică deoarece stabilește scopul acestora în aplicație și le definește existența. De asemenea, prin aceste scenarii, este identificat fluxul ce trebuie parcurs în aplicație de catre un utilizator pentru o prima folosire.

Pentru o mai bună organizare a funcționalităților aplicației, a fost nevoie sa definim mai multi actori ai aplicației. Astfel, există rolul de administrator al aplicației pentru o asociație, cont destinat unui membru din conducerea asociației și rolul de voluntar.

3.4.1 Specificațiile functionale din punctul de vedere al unui Administrator al unei asociații

Odata stabilit pentru un utilizator că acesta va avea rolul de administrator al unei asociații, primul pas pe care trebuie sa îl efectueze este de creare a unui cont în aplicație. Acest tip de cont se realizează în cadrul paginii de *Register* unde pe langă completarea cămpurilor nume, email si parolă se va alege opțiunea din checkbox de creare a unui cont de ADMIN.

Odata ce administratorul este înregistrat în aplicație, aspectul acesteia se schimbă, în dreapta sus a bării de navigare îi va aparea numele ce va juca rol de buton ce desfasoara si optiunea de *Logout*. De asemenea, pe langă legaturile către paginile *Despre* si *Contact*, va aparea și legatura către *Toate proiectele* și un buton ce introduce un sidebar (bară laterală) cu alte legături ale paginilor ce au scopul de a naviga rapid în aplicație.



Din acel moment, administratorul poate vizualiza pagina sa de *Home*, care este reprezentată ca o cronologie pe orizontală a viitoarelor activități ce vor fi de realizate în cadrul aplicației. Pentru fiecare bulină în parte există și o descriere a modului în care se va acționa. Prima bulină este cea care indică crearea unei asociații noi de care și administratorul va aparține ulterior. A doua bulină conduce către realizarea unor cuvinte cheie sau taguri, cum sunt numite în aplicație, ce au un rol important în realizarea căutarii de proiecte. A treia bulină face trimitere către crearea unui nou proiect. Urmatoarea acțiune este aceea de vizualizare a tuturor voluntarilor din cadrul asociației, iar ultima din cadrul panoului de control al administratorului, este cea de acces către aprobarea sau dezaprobarea viitorilor voluntari în asociație.

În cadrul creării unei asociații se seteaza numele, locația, domeniul și se poate chiar urca o imagine ce reprezinta logo-ul asociației. În plus, poate, mai apoi, să vizualizeze infomațiile introduse despre asociație. Urmatoarea acțiune este aceea de adăugare a departamentelor asociației ce are loc în aceeași pagină în care se poate vizualiza listarea acestora. În continuare, va fi nevoie de stabilirea si crearea unor taguri, ce vor încadra viitoarele proiecte realizate în anumite categorii, facând mai ușoară căutarea lor. Există opțiunea de a vizualiza fiecare tag și proiectele ce îl cuprind, dar și de a edita sau șterge tagul respectiv.

Odata ajuns la crearea de proiecte, va întalni, poate, cel mai complex formular al aplicației, în care trebuie sa introducă date pentru titlul proiectului, linkul specific al proiectului, să aleagă departamentul de care sa apartină dintr-o lista, tagurile potrivite, poate să încarce o imagine sugestivă și să descrie proiectul cu ajutorul butoanelor de formatare care îi sunt puse la dispozitie, și nu în ultimul rând să precizeze datele de început și de final. În cazul proiectelor, acestea pot fi vizualizate în listă, dar și individual cu toate detaliile, pot fi modificate și șterse după caz. La fel ca și în cazul asociației, în cadrul proiectelor va fi necesară adăugarea de taskuri specifice pentru fiecare proiect în parte.

Pentru a gestiona voluntarii din asociație, un administrator are posibilitatea de a vizualiza voluntarii care vor să intre in asociație și să decidă cine are permisiune și cine nu. Pe langa această operațțiune, administratorul poate vizualiza toți voluntarii aplicației și le poate

monitoriza activitatea acestora printr-o pagină specială de statistici.

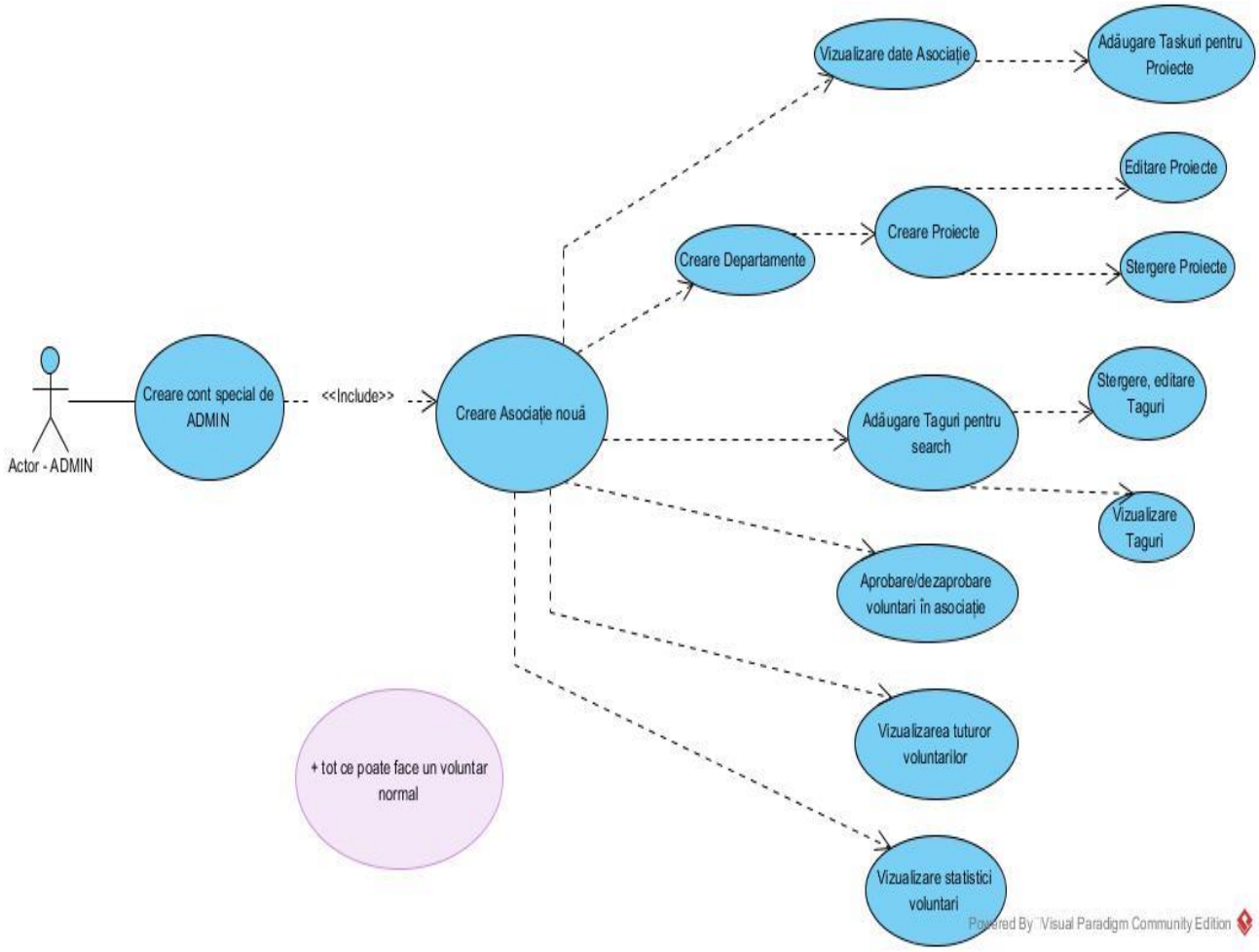


Diagrama UML a cazului de utilizare a Administratorului

În plus, un administrator are la dispoziție posibilitatea realizării tuturor operațiunilor pe care un voluntar normal le poate face.

3.4.2 Specificațiile functionale din punctul de vedere al unui voluntar

Pentru ca un utilizator să primeasca rolul de voluntar în aplicația Manage Vol, acesta este nevoit să își creeze un cont pe pagina de *Register*. În cazul rolului de voluntar, pe langă completarea câmpurilor standard de nume, email si parolă, el este nevoit să aleaga dintr-un dropdown (listă de opțiuni) asociația din care s-a convenit că va face parte și, în funcție de aceasta, departamentul corespunzător.

Odată înregistrat, voluntarul este trimis către pagina de *Home*, care cuprinde butonul către vizualizarea datelor asociației, o legatura către vizualizarea departamentelor asociației și către punctul de interes, anume proiectele.

La fel ca și în cazul administratorului, aspectul aplicației suferă mici modificări, fiindu-i disponibil sidebar-ul, o legatura catre proiecte și numele în bara de navigație.

Una dintre principalele operațiuni ale voluntarului o reprezintă vizualizarea proiectelor și navigarea în cadrul acestora prin input-ul de search disponibil. Voluntarul poate introduce orice cuvant cheie îl interesează. Aceasta se extinde prin posibilitatea privirii în detaliu a proiectului, adăugarea unui comentariu în cazul unor opinii suplimentare, și implicarea în diferite task-uri. Odată ce va apăsa butonul de „Join” al unui proiect, este dus către pagina de listare a task-urilor proiectului ce îi stau la dispoziție. În dreptul fiecarui task va exista un buton care își va schimba starea în blocat, din momentul alegerii task-ului respectiv în care voluntarul a vrut să se implice.

Pentru a-și putea face o idee despre activitatea sa în asociație, pagina de Statistici personale îi va sta la dispoziție cu grafice relevante.

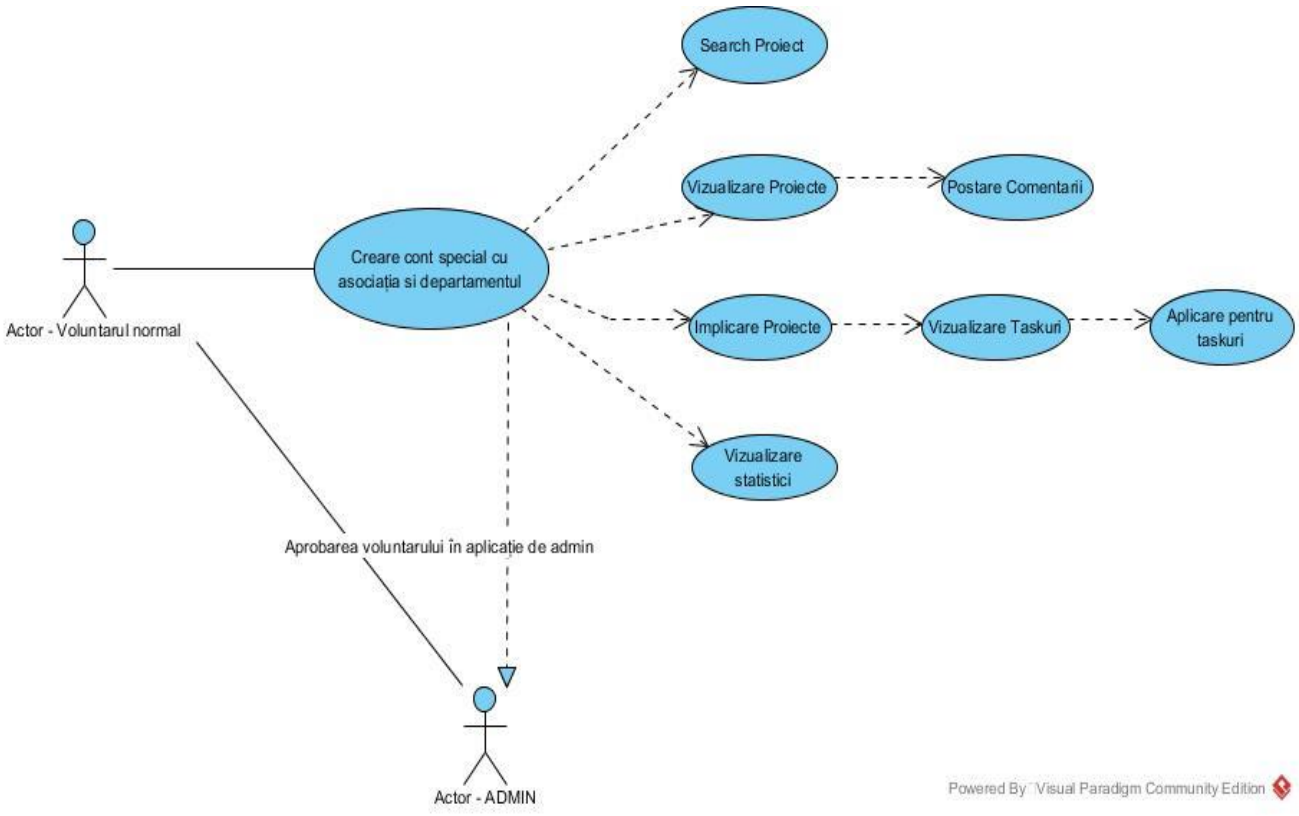
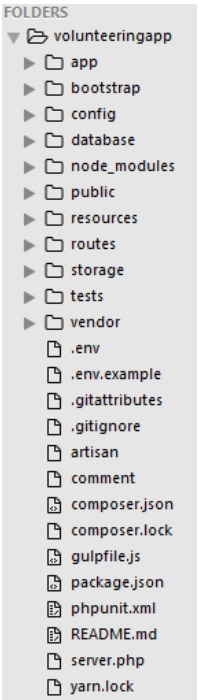


Diagrama UML a cazului de utilizare a voluntarului normal

4. ARHITECTURA APLICAȚIEI

Orice proiect nou, generat cu ajutorul framework-ului Laravel, va fi structurat într-un arbore de directoare foarte bine clasificate și chiar exemple de fișiere, rezultând astfel într-o structură ce permite o pornire rapidă a procesului de dezvoltare. Astfel, se oferă o viziune a structurii proiectului și a regăsirii informației într-o maniera ușor de înțeles.



4.1 Model-View-Controller

Model-View-Controller, pe scurt MVC, reprezintă o paradigmă arhitecturală de procesare, structurare si dezvoltare a elementelor unei soluții software. Deși initial a fost folosită pentru realizarea aplicațiilor de tip Desktop, ulterior a fost adoptată pentru partea de web, fiind chiar integrată in diferite framework-uri de dezvoltare.

Această tehnică a aparut in anul 1979 și este o foma de a gandi structura unei aplicații prin izolarea elementelor logice, de cele vizuale și de informațiile din baza de date. În prezent, MVC este foarte întâlnit în framework-uri precum Rails pentru Ruby, Laravel pentru PHP, Spring pentru Java, Django pentru Python sau Swift pentru IOS etc.

Tehnica MVC promoveaza conceptul folosit în industrie de DRY – „Don't Repeat Yourself" (nu te repeta) ce se referă la organizarea și eficientizarea codului astfel încât să se faciliteze reutilizarea acestuia.

Înainte de a trece mai departe, vom prezenta, mai întâi, ordinea în care circula datele dintr-o aplicație web.

Plecând de la clientul care realizeaza un request (o cerere) pe un browser (navigator web), se realizează randarea și procesarea unor componente precum HTML, CSS sau JavaScript. Request-ul trimis ajunge la server (locul in care furnizeaza functionalități) care rulează limbaje specifice părții de server, precum PHP, Java, Ruby etc. Serverul nu are drept sarcină stocarea infomațiilor deși acolo sunt păstrate variabilele. Astfel apare necesitatea unei baze de date ce poate fi creată si gestionată prin instrumente precum MySQL, PostgreSQL etc, ce va conține toate datele necesare și care pot fi atat SQL cât și NOSQL. Odata identificate informațiile cerute în request, acesta din urmă este întors înapoi la server, care, odata ce a înteles de ce date are nevoie, trimite catre utilizator o interfață prietenoasă ce afișează informațiile dorite.

Acum că fluxul funcționării unui request este explicat, putem întelege și identifica mult mai ușor fiecare element din arhitectura Model-View-Controller.

MODEL

- Este acel element care adaugă si recuperează datele din baza de date
- Procesează datele de la și catre baza de date
- Comunică doar cu Controller-ul

VIEW

- Este componenta afișata utilizatorului
- Comunică sau, cu alte cuvinte, ascultă comenzile de la Controller. Nu ia decizii, ci doar execută ce i se spune să afiseze; comunică doar într-o singură direcție

CONTROLLER

- Este acea parte care proceseaza request-uri de tip GET/POST/PUT/DELETE
- Dețține acel tip de logică specifică părții de server
- Ia informaăii de la utilizator, le procesează și le trimite dacă este necesar mai departe catre baza de date, care finalizează cautarea datelor necesare
- Reprezintă pe scurt partea logică ce manevrează fluxul de la utilizator și înapoi către el

În mod practic, semnificația fiecărei componente a arhitecturii MVC se poate restrânge la: Modelul să fie reprezentat de baza de date, View-ul de către interfața clientului, iar Controller-ul de serverul aplicației web.

Mai mult, tehnica MVC introdusă în framework-uri reușește să simplifice foarte mult procesul de dezvoltare prin introducerea

procesoarelor de rute, sau posibilitatea ca un Controller să poată structura metodele în funcție de fiecare pagina.

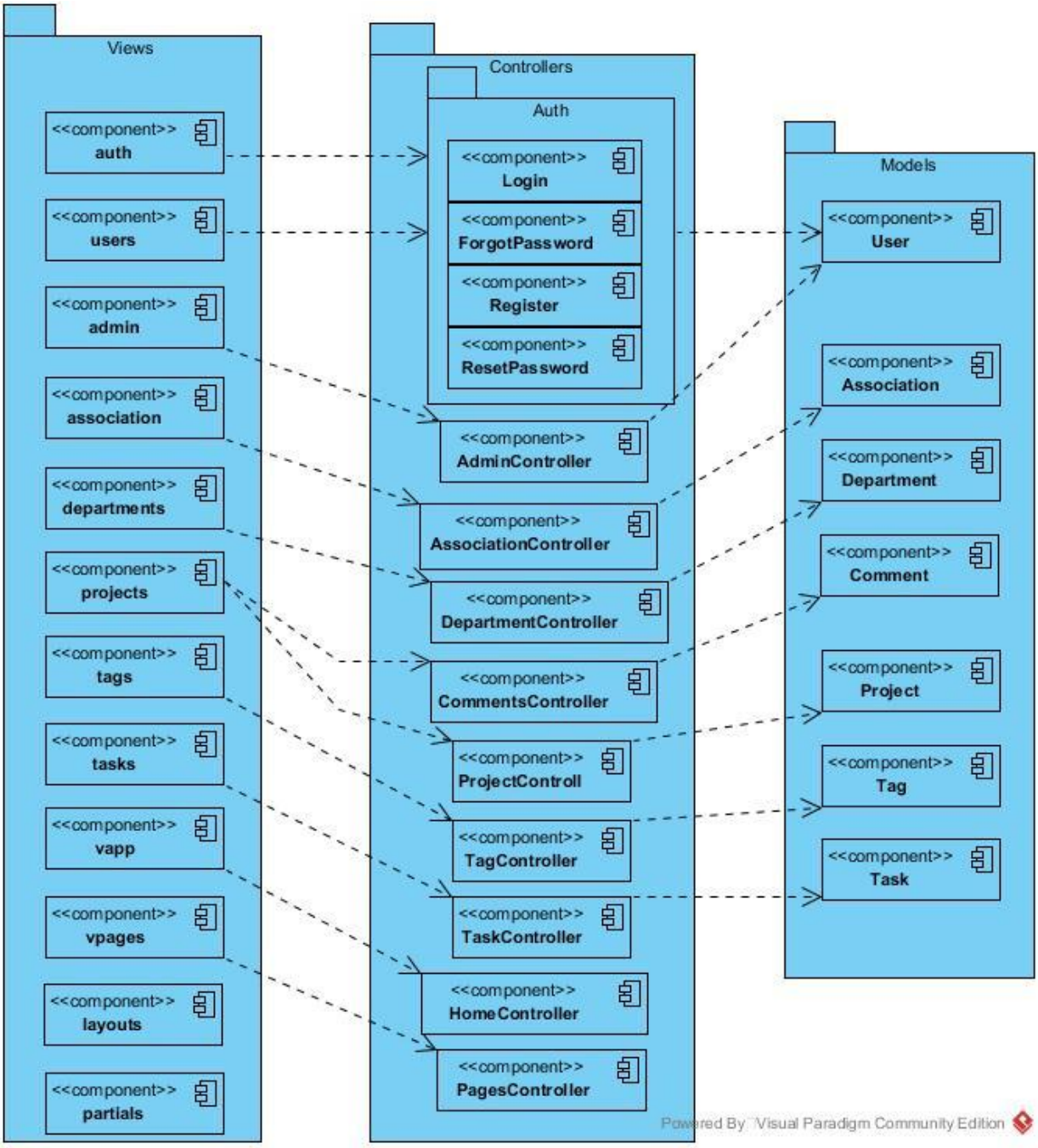


Diagrama UML a componentelor modelului MVC

Figura de mai sus ilustrează, în mare, cum reușesc elementele din modelul MVC să comunice între ele în cazul aplicației Manage Vol. Este de precizat ca elementele din Views reprezintă directoare ce cuprind fișiere cu pagini ce au în general aceeași denumire precum

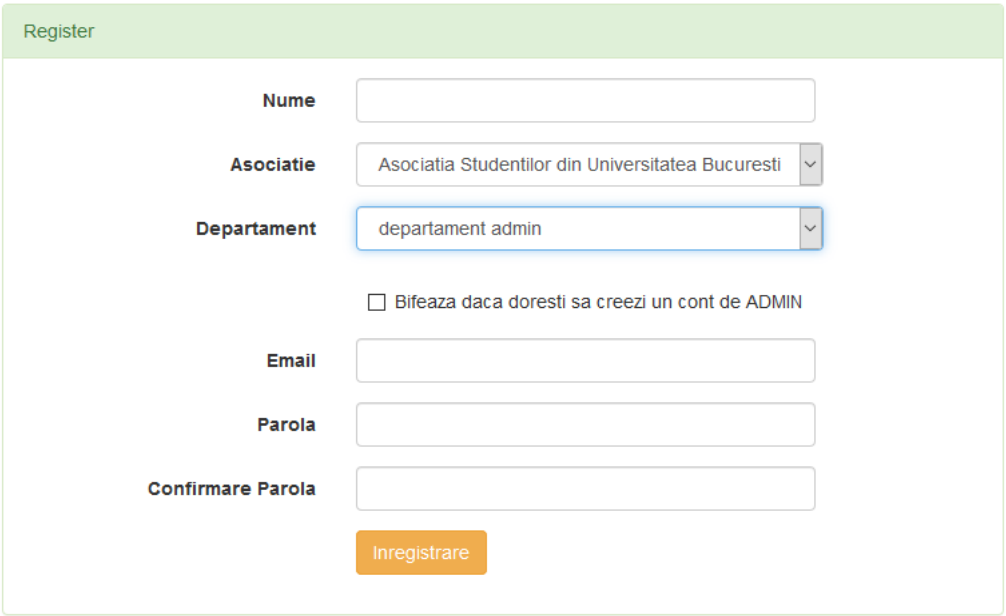
index, creat, edit, show sau altele. Folderul de layouts este unul predefinit din crearea proiectului cu ajutorul Laravel și cuprinde schema, sau planul general al unei pagini web deja stilizată. Vom alege însa, o stilizare proprie ce are drept elemente acele bucăți de cod care s-ar repeta pentru fiecare pagină de site precum _head, conține părțile de styling ale paginilor și alte legături, _navsidebar, ce cuprinde tot ce ține de bara de navigare și de bara laterală, _footer (subsol), partea inferioară a paginii etc.

Deși legăturile dintre Controllere si Modele sunt create astfel, un controller poate folosi mai multe modele deodată și crea mai multe legături între acestea.

4.2 Crearea View-urilor

4.2.1 LOGIN – REGISTER

Având în vedere că aplicația Manage Vol are drept scop managerierea unor task-uri pentru voluntari, adică functionalități interne pentru o asociație, este nevoie ca prima pagină de interacțiune cu aplicația sa fie cea de Login. Această pagina sugerează faptul că pentru a putea fi folosită este nevoie de deținerea unui cont. Pagina de Login este una clasică, realizându-se pe baza email-ului unic în aplicație și a unei parole criptate care pot fi memorate, daca se dorește, prin bifarea opțiunii de „Remember me” (Amintește-ți de mine).



Pentru a realiza un cont este nevoie de accesarea paginii de Register (Înregistrare). În cazul acestei pagini, s-a plecat de la înregistrarea clasică cu nume email și parolă, ajungându-se la una complexă ce ajută în definirea viitorului rol din aplicație. În urma unei analize a modului de înregistrare, a fost nevoie de adăugarea mai multor câmpuri precum asociația și departamentul în care se înscrie utilizatorul, precum și dacă este un utilizator cu rol de voluntar sau de administrator.

Crearea legăturii de apartenență a unui voluntar la o asociație și, mai apoi, la un departament a fost posibilă prin introducerea a două dropdown-uri dinamice și dependente unul de celalalt, astfel că odată aleasă o asociație, are loc identificarea acesteia și afișarea strictă a departamentelor sale. Această acțiune are loc fără o încărcare suplimentară a paginii, folosit fiind instrumentul Ajax împreună cu JavaScript într-un script special al paginii, pentru a crea astfel un eveniment ce se declanșează odată cu alegerea asociației. Mai exact, atunci când este selectată o asociație, se execută o funcție care efectuează un request HTTP de tip GET ce returnează un răspuns sub formatul JSON conținând departamentele din acea asociație. Mai departe, folosim aceste informații pentru a adăuga dinamic opțiuni noi conținând numele departamentelor elementului HTML <select></select>.

Mai mult, vom crea un checkbox (buton de verificare) pentru a putea face disponibilă opțiunea creării unui cont de administrator odată ce este bifat. Această acțiune face update în baza de date pe coloana de admin a user-ului cu valoarea 1, valoare ce identifică administratorul. În același timp, bifarea checkbox-ului, setează drept „indisponibila” starea celor două selecturi dinamice pentru a nu se crea o neclaritate în fluxul creării unui cont.

4.2.2 CONTACT

Ca oricare altă platformă online și care oferă anumite servicii de interes pentru un anumit domeniu, aceasta trebuie să conțină și o pagină de Contact.

Această pagină este vizibilă tuturor vizitatorilor și este reprezentată de un formular ce permite furnizarea unor opinii imediate sau a unor mesaje de colaborarare pe email-ul celui care este desemnat administrator de aplicație.

Pentru a putea realiza transmiterea pe email a mesajelor a fost nevoie încă o data de intervenția modificării mediului de lucru adică a fișierului .env. Aici se vor găsi niște variabile ce vor trebui populate cu valorile dorite. În cazul în care aplicația va ajunge în producție va fi modificat și mail.php din directorul config.

`MAIL_DRIVER=smtp` // se precizează protocolul prin care se realizează transferul de informație

`MAIL_HOST=mailtrap.io` // adaugăm link-ul către motorul de email folosit

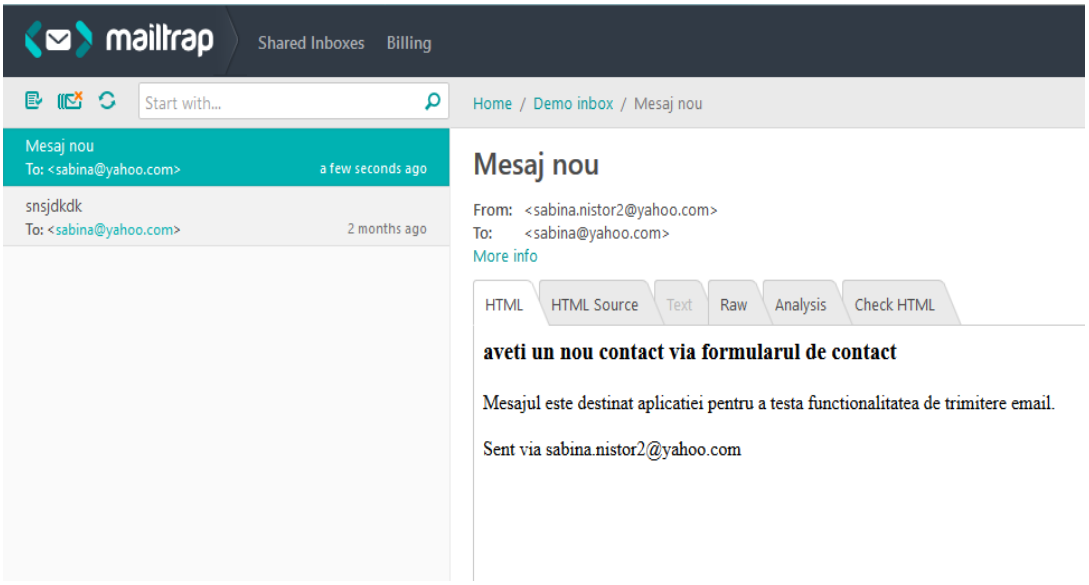
`MAIL_PORT=25` // valoarea portului pe care funcționează protocolo smtp

`MAIL_USERNAME=c8e281c7446428` // numele criptat oferit de mailtrap.io

`MAIL_PASSWORD=c43e4dfc766efb` // parola criptată oferită tot de mailtrap.io

`MAIL_ENCRYPTION=null`

Pentru această funcționalitate s-a folosit mailtrap.io care are ca scop în principal simularea trimiterii unui email. Cu alte cuvinte, Maitrap este un server SMTP fals destinat testării trimiterii, vizualizării și împărtășirii de email-uri fără a fi nevoie de utilizarea unor conturi reale.



Folosirea acestui instrument este ușoară, necesitând doar crearea unui cont. În cadrul acestuia se vor găsi mai multe informații folositoare pentru completarea variabilelor noastre de mai sus. Pentru serviciul SMTP se găsesc valori pentru portul disponibil, numele utilizatorului și parola criptată. Tot cu acest cont vine și posibilitatea vizualizării email-urilor trimise ca test, ale aplicației de față.

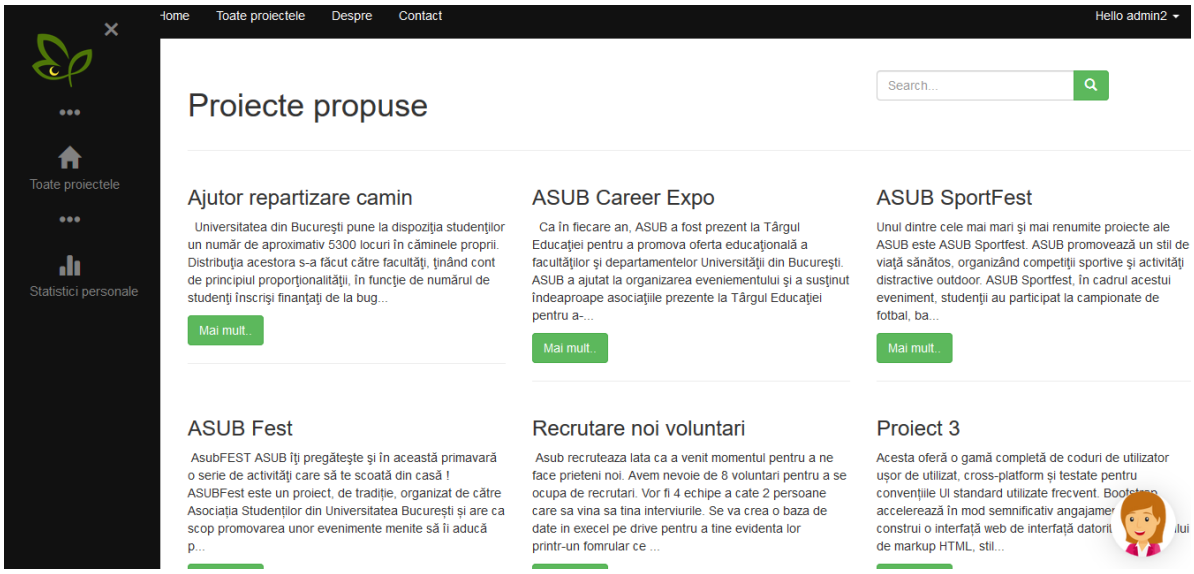
Clasa Mail a Laravel-ului este construită deja prin Librăria PHP SwiftMailer care este o librărie foarte puternică și populară ce permite o mulțime de lucruri precum realizarea trimiterii proprietăților din request mai departe în pagină de contact ce va fi vizibilă în emailul expediat mai departe.

4.2.3 PROIECTE

Proiectele, sau în cazul unor asociații evenimentele, reprezintă punctul de interes, sau altfel spus, ansamblul scopurilor unei asociații, pe termen relativ scurt. Conștientizând acest aspect, vom pune accent pe implementarea implicării în cadrul proiectelor.

Pagina de vizualizare a proiectelor a trebuit sa fie cât mai atractivă vizual, clară și să ofere posibilitatea accesării mai multor detalii pentru unul dintre ele. Stilul paginii, ce cuprinde proiectele, a putut fi realizat cu ajutorul metodei *chunk()* care preia din

ProjectController lista cu toate proiectele asociației si sparge colecția de proiecte în colecții de dimensiuni dorite, în cazul de față 3 proiecte pe linie. Această metodă este în special folositoare atunci când se lucrează cu Bootsrap și se dorește formarea unor grid-uri (grilă) cu date.



Exemplul de mai jos surpinde modul de realizare a paginii de proiecte.

```
@foreach ($projects->chunk(3) as $chunk)
    <div class="row">
      @foreach ($chunk as $project)
        <div class="col-xs-4">
          <h3>{{ $project->title }}</h3>
          <p>{{ substr(strip_tags($project->body), 0, 300)}}{{ strlen(strip_tags($project->body)) >300 ? "..." : ""}}</p>
          <a href="{{ url('vapp/'.$project->slug) }}" class="btn btn-success">Mai mult..</a>
        </div>
      @endforeach
    </div>
@endforeach
```

În plus, pe langa vizualizarea proiectelor, pentru a completa posibilitatea de comunicare din cadrul asociației, a fost introdus un instrument ce comunică în mod direct cu un grup de **sluck**.

Formularul de creare a proiectelor cuprinde si el niște elemente interesante de amintit. Unul dintre aceste elemente este stabilirea unui câmp ce setează denumirea link-ului către proiect. De asemenea pentru adăugarea mai multor taguri a fost utilizată o librarie jQuery numită **select2** ce se comportă precum un tag HTML de select normal însă poate fi setat dupa preferințe, având în plus opținea de a realiza selecturi multiple, scroll la infinit s.a.

Pentru ca experiența scrierii unui proiect, ce poate necesita mai multe detalii, să fie cat mai completă, a fost introdusă o alta tehnologie bazată pe JavaScript ce permite formatarea textului. Aceasta se numeste **TinyMCE** și se dorește a fi cel mai bun editor HTML de tipul **WYSIWYG** („what you see is what you get" – ceea ce vezi este ceea ce primești).

Funcționalitatea postării unui proiect se completează și cu posibilitatea de a introduce o imagine sugestivă pentru domeniul în care se plasează sau chiar dintr-o ediție anterioară. Acest lucru a fost posibil printr-o librarie open-source specifică PHP ce manipulează imaginile, numită **Intervention Image**. În final, introducerea datelor se poate realiza prin intermediul unui datepicker.

Ajungând la partea de vizualizare a proiectului, se poate observa posibilitatea postării de comentarii a utilizatorului autentificat. Pentru a marca standardele unui comentariu, fără ca utilizatorul sa dispună de o poză, s-a folosit un serviciu foarte la îndemână numit **Gravatar**. Acesta este, practic, o imagine reprezentativă în mediul online (Globally Recognized Avatar). Gravatar este automat

introdus în cazul în care emailul utilizatorului este folosit și pentru Wordpress.

4.2.4 TAGURI

Tot pe pagina de proiect există tagurile sau cuvintele cheie prin intermediul cărora se realizează căutarea. Pentru a putea realiza o acoperire cat mai mare în ceea ce privește identificarea unui anumit proiect, a fost nevoie de un sistem de introducere a cuvintelor cheie ce privesc un proiect sau chiar viitoare alte proiecte**.** Deci, inputul de search din pagina de vizualizare a proiectelor se completează cu orice cuvant ce ar putea avea legatura cu proiectul dorit.

Vom da un exemplu de interogare ce reușește să selecteze tagurile proiectelor prin care se realizează search-ul. Acest exemplu este realizat cu ajutorul componentei numite Query Builder, practic, este una dintre modalitățile prin care se pot construi interogări în baza de date. Variabilei *projects* i se vor atribui acele proiecte care vor face join cu tabela de intersecție *project_tag* și mai apoi cu tabela *tags* unde numele tagului se identifică prin inputul de search și astfel, sunt alese doar o data și în ordine crescătoare câte șase per pagină.

```
$projects = DB::table('projects')
->join('project_tag', 'projects.id', '=', 'project_tag.project_id')
->join('tags', 'project_tag.tag_id', '=', 'tags.id')
->select('projects.*')->distinct()-
>where('tags.name','like','%'.$search.'%')->orderBy('id','desc')-
>paginate(6);
```

În acest mod, a fost creată o pagină specială de adăugare a tagurilor, dar și de vizualizarea a acestora. Toate tagurile sunt diponibile tuturor asociațiilor și, fiind doar un cuvant, nu ar trebui să pericliteze informațiile specifice unei asociații. De asemenea un tag se poate șterge sau modifica, dupa caz.

4.2.5 TASKURI

Realizarea taskurilor/sarcinilor voluntarilor a reprezentat o cerintă de o importanță majoră. Acestea sunt ultimul pas din fluxul de manageriere din aplicație. Pagina de acces a taskurilor apare odată ce este selectat butonul de *Join* >> din cadrul vizulizării unui anumit proiect.

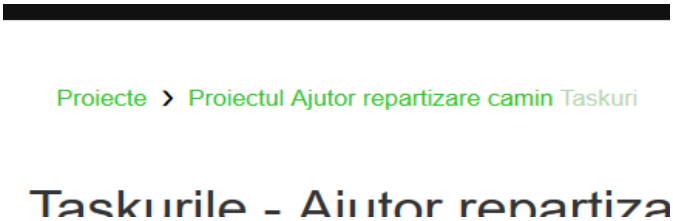
Primul task dintr-un proiect face referire la alocarea proiectului unui utilizator cu rol de Project Manager. Aceasta a fost abordarea pentru crearea unui astfel de rol. În spate se efectueaza un update pe coloana *isManager* a taskurilor, iar pagina de adăugare a taskurilor va putea fi accesibilă pentru utilizatorul ce a fost ales drept Project Manager.

Alegerea taskurilor dorite si vizuzalizarea lor este pe cat posibil de placută și de intuitivă. Taskurile si descrierea lor sunt dispuse într-un tabel si conțin fiecare câte un buton cu un glyphicon sub formă de stea golită, semnificația faptului că taskul este nealocat. Odată ce sunt alese takurile dorite de voluntar, butonul își schimbă diponibilitatea în blocat și glyphicon-ul cu stea goală se schimba într-unul cu stea plină. Toate acestea au fost realizate printr-un formular cu tipul POST ce a avut drept condiție *@if((in_array($task->id, $taskDist)))* ce verifică dacă taskurile se află în lista creată special în *TaskController* și care ține doar acele taskuri pentru care utilizatorul s-a înscris deja.

```
$tasks = Task::all();
$user = Auth::user()->id;  // in acest mod se acceseaza id-ul user-ului autentificat
$taskDistArray=DB::table('user_task')->select(DB::raw('task_id'))->where('user_id',$user)->get()->toArray();
// se aleg taskurile deja alocate user-ului autentificat din baza de date
$taskDist = [];
for ($i=0; $i < count($taskDistArray); $i++) {
```

```
        $taskDist[$i] = $taskDistArray[$i]->task_id;
    } // se introduc intr-o lista noua
```

Pentru a realiza o navigare mai ușoară, de la proiecte la task-uri, vom folosi conceptul de *breadcrumbs din HTML* . Acesta constă într-un fel de listă neordonată de legaturi către paginile dintr-un flux logic al utilizării aplicației. Metoda aceasta reușește să reducă numarul de acțiuni al unui utilizator si să indentifice poziția acestuia în platformă.



Pagina curentă este evidențiată printr-o culoare mai deschisă, iar între legaturile celorlalte pagini există un simbol ce semnifică sensul de parcurgere din aplicație.

4.2.6 STATISTICI

Pagina de statistici reprezintă un alt punct de interes pentru fiecare utilizator. Este de înteles faptul că odata ce se inițiază o aplicație de organizare, este nevoie și de o modalitate de monitorizare a sarcinilor cât se poate de interactivă. Astfel că pagina de statistici este una de mare ajutor pentru conștientizarea aportului personal adus in perfomanța asociației.

Librăria folosită pentru a aduce la viață digramele din cadrul paginii *Statistici* se numește Charts și este special concepută pentru framework-ul Laravel. Această tehnologie oferă acces la o multitudine de tipuri de grafice, tabele sau diagrame. Faptul că este foarte ușor de folosit, reușește să faciliteze munca unui programator. Acesta se poate concentra în continuare pe dezvoltarea unui proiect, deoarece dispune de un instrument foarte bine definit. Mai jos este un exemplu de configurare pentru realizarea unei diagrame.

```
$chart = Charts::create('donut', 'morris')
        ->title('La cate taskuri te-ai inscris pentru fiecare proiect')
        ->labels($tasksUsed->pluck('title')->toArray())
        ->values($tasksNumber->pluck('num')->toArray())
        ->dimensions(300,300)
        ->responsive(false);
```

Tool-ul folosit introduce posibilitatea de a crea un obiect de tip Charts în controller-ul ce se va ocupa de pagina *Statistici*, ce are la dipoziție metoda create si cateva proprietăți ce pot fi personalizate. Primul argument al metodei create reprezintă tipul de chart (ex: line, area, bar, percentage etc), iar al doilea face referire la biblioteca sau tipul de vizulizare al statisticii dorite. Proprietățile au fost inițializate dinamic pentru a lua datele direct din baza de date. „Labels” reprezintă denumirea cămpurilor în funcție de care se realizează diagrama și în acest caz sunt luate titlurile proiectelor în care un utilizator este implicat prin task-urile aferente. Valorile sunt luate și ele în funcție de numarul taskurilor pentru fiecare proiect în parte. Pentru potrivirea în pagină, se pot completa câmpurile „dimensions” sau „responsive”, în funcție de preferințe.

4.3 Valiarea formularelor

Aplicația Manage Vol a presupus folosirea în multiple ocazii a formularelor. Folosirea formularelor aduce după sine și implementarea unor validări care să fie în conformitate cu tipul inputurilor ce se doresc a fi completate. În cazul aplicațiilor dezvoltate cu un framework precum este Laravel, există mai multe posibilități de realizare. Laravel pune la dispoziție un sistem foarte simplu de utilizat. Vom porni explicația de la rutele unui formular cu metoda GET care afișează

formularul, iar POST care va stoca datele introduse. Următorul element este controllerul unde cu o privire mai amănunțită vom observa că acesta foloseste trait-ul ValidatesRequests care furnizează metoda validate() care este folosită în cadrul metodei store().

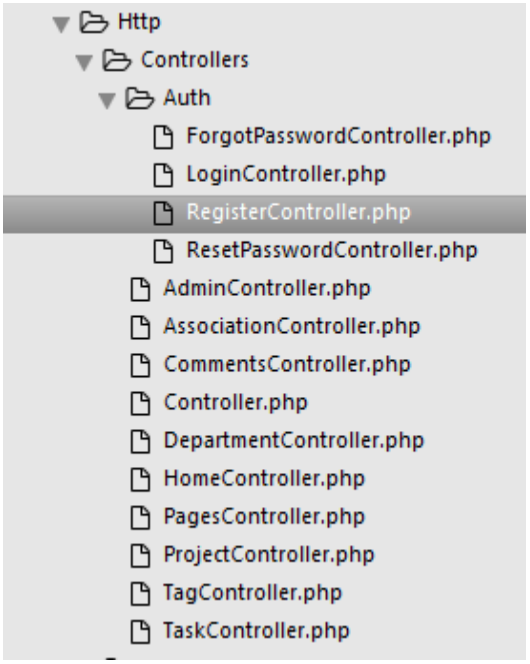
Metoda validate() acceptă o cerere HTTP primită și un set de reguli de validare. Dacă regulile de validare trec, codul va continua să fie executat în mod normal. Cu toate acestea, dacă validarea nu reușește, o excepție va fi aruncată, iar răspunsul corect la eroare va fi trimis automat utilizatorului. În cazul unei solicitări HTTP tradiționale, va fi generat un răspuns de redirecționare, în timp ce un răspuns JSON va fi trimis pentru solicitările AJAX.

```
public function store(Request $request)
    {
      $this->validate($request, array(
         'title'=>'required|max:25',
         'slug'=>'required|alpha_dash|min:5|max:255|
         unique:projects,slug',
          'department_id' =>'required|integer',
          'body'          => 'required',
          'begin_date'    => 'required',
          'end_date'      => 'required',
          'featured_image'=>'sometimes|image'

            ));
}
```

Exemplul de mai sus ilustrează modalitatea de validare pentru inputurile ce sunt puse la dispoziție pentru crearea unui nou proiect. Pentru fiecare câmp din baza de date există un set de reguli. „Required” însemnă că este absolut necesar sa fie completat acel camp, „min” si „max” setează limitele inferioare si superioare ale numărului de caractere folosite și există multe alte proprietăți ce sunt disponibile în documentația oficială Laravel.

4.4 Soluții de logică



Pentru a oferi logică paginilor web de vizualizare, este nevoie de crearea unor Controllere care să gestioneze metodele necesare acțiunilor ce pot fi efectuate.

Pentru un pachet complet al unei componente din baza de date ce are nevoie de afișarea valorilor, crearea acestora, ștergerea și modificarea, Laravel pune la dispoziție o comandă artisan ce creează un Controller, ce vine spre ajutorul dezvoltatorului. Această comandă `php artisan make:controller ExempluController --resources` realizează un fișier cu clasa ExempluController, ce extinde clasa predefinită Controller, ce va cuprinde scheletul fiecărei metode de care va fi nevoie pentru a realiza fiecare acțiune în parte.

De asemenea pentru a face legătura între paginile menționate, în routes se va scrie:

`Route::resource('tags','TagController',['except' =>['create']])` unde „except” indică faptul că o metodă poate să nu fie utilizată, în acest exemplu fiind vorba de create.

În definitiv, aproape fiecare entitate din baza de date va dispune și de un controller care să manevreze fiecare activitate diponibilă în partea vizuală.

*

Problema asignării administratorului la asociația recent creată de el, a beneficiat de o soluție. Deoarece un cont de administrator nou creat, teoretic, nu poate sa aparțină de o asociație, rezolvarea a constat

din crearea automată a unei asociații temporare și a unui departament temporar care să găzduiască administratorul până la operațiunea de creare a unei asociații noi.

`$association = new Association;` //se realizează un obiect nou de tipul modelului Asociatie

`$department = new Department;` // se realizează un obiect nou de tipul modelului Departament

`$user = Auth::user();` // se identifică utilizatorul autentificat și se introduce în variabila $user

`[...] //cod`

`$association->save();` // după ce au fost setate toate proprietățile obiectului, se apelează metoda de salvare a acestora

`$department->name = 'departament admin';` // se folosește hardcode pentru a stabili numele departamentului valabil pentru fiecare departament de administrator al asociațiilor

`$department->description = 'Departament special pentru userul de tip ADMINISTRATOR';` // la fel ca și în cazul numelui, se stabilește descrierea folosită

`$department->association_id = $association->id;` // pentru id-ul asociației se va da valoarea asociației ce tocmai a fost creată, pentru a se crea o dependință între tabelul asociații și departamente

`$department->save();` // odata setate câmpurile, le vom salva in baza de date

`$user->department_id = $department->id;` // petru utilizatorul autentificat, în acest caz administratorul, se va face un update in baza de date pe coloana de department_id cu id-ul departament-ului nou creat pentru a produce legatura dintre admin și departamentul asociației nou create.

`$user->save();` // după orice modificare, fie doar pentru o singură proprietate, se cheamă metoda „save”

4.5 Restricționarea paginilor web

Restricționarea paginilor este o acțiune esențială deoarece există mai multe tipuri de utilizatori și este nevoie de o diferențiere a funcționalităților fiecăruia. Exista mai multe modalități de a realizarestricționarea paginilor în aplicație, dar cea de față cuprinde o combinație dintre acestea.

Prima variantă de a restricționa o pagină este prin crearea unui middleware. Laravel are deja unul creat și este vorba despre cel de autentificare, practic, se verifică dacă utilizatorul este autentiifcat în aplicație și poate accesa conținutul acesteia. Acest middleware este folosit atât în constructorul controllerului ce se ocupa de anumite pagini dar poate fi folosit și în rute, așa cum vom exemplifica mai jos. Pe lângă middleware-ul creat pentru autentificare, vom crea unul special pentru administrator, întrucât acesta deține pagini ce nu ar trebui accesate de voluntari și implicit de vizitatorii neautentificați.

```
class AdminController extends Controller
{
    public function __construct()
    {
        $this->middleware('admin');
    }
[...]
 }
Route::get('admin', ['middleware' => 'auth', 'admin' , 'uses' => 'AdminController@index' , 'as' => 'admin.index1']);
```

// aici se regăsește modalitatea de a crea legătura către pagina de administrator cu panoul său de control

O altă modalitate este pe partea de client, unde se poate restricționa sau chiar schimba conținutul unei anumite pagini web. A

fost folosit în cazul departamentelor, unde pagina de index avea atât rolul de afișare a acestora, cât și cel de creare.

```
@if (auth()->user()->admin === 1)
    [...]
@endif
```

Prin operația de mai sus se verifică valoarea coloanei admin, a utilizatorului, iar pentru rolul de administrator s-a stabilit ca aceasta să fie 1. Dacă este îndeplinită condiția, conținutul este afișat, altfel nu.

*

Partea logică, din spatele scenariului din punctul de vedere al unui utilizator cu rol de voluntar, a fost posibilă cu ajutorul componentelor explicate de mai sus.

In AdminController, sunt selectați toți utilizatorii ce încă nu au fost aprobati prin oprațiunea `$users = User::where('approved',0)->get();` iar mai apoi se creează o listă cu aceștia ce se va folosi în view unde se va verifica daca satisfac condiția următoare:

```
@if((in_array($user->id, $userDist)))
```

Această condiție face determină starea butoanelor care pot aproba sau dezaproba voluntarul în asociație.

4.3 Modelul bazei de date

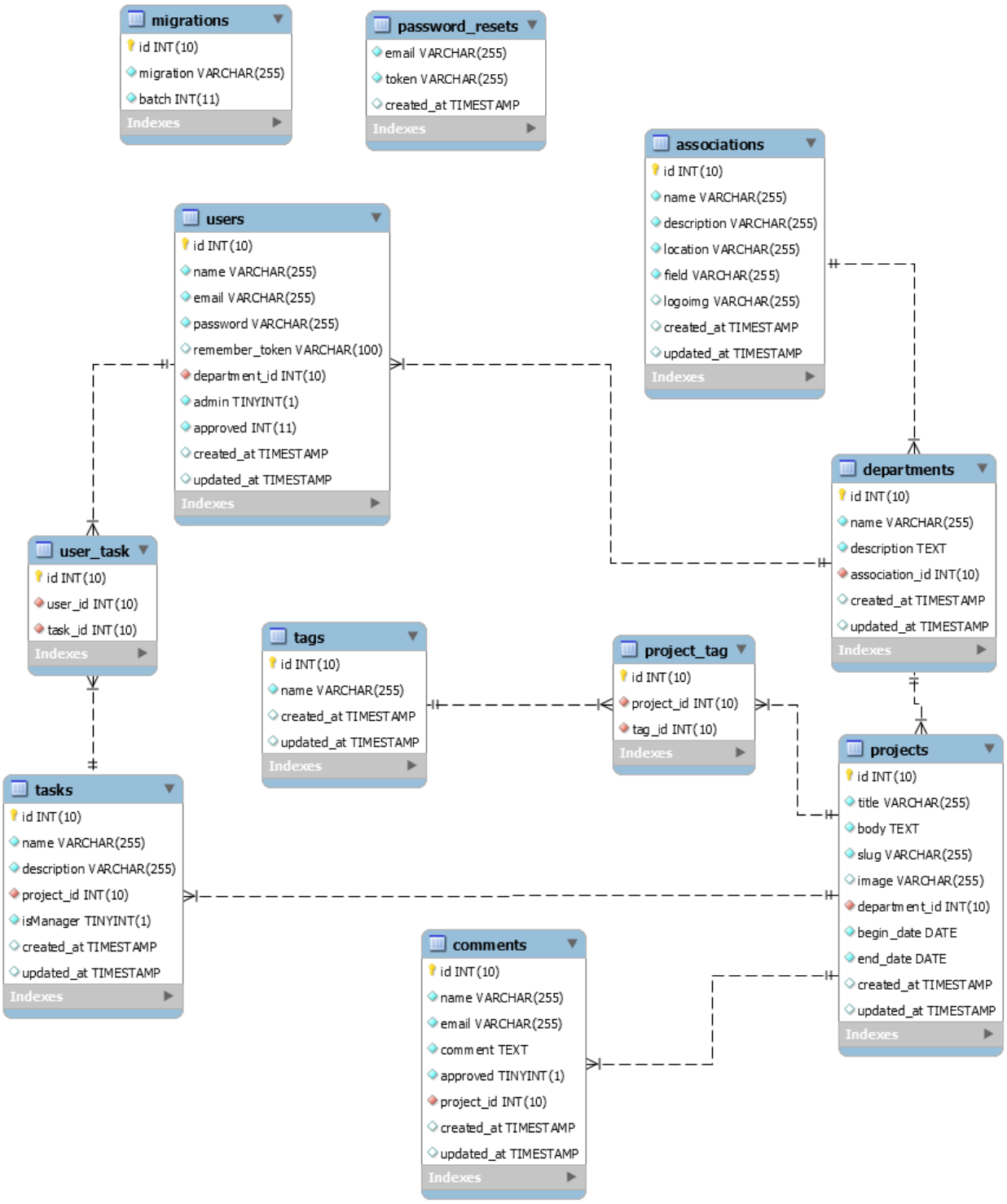
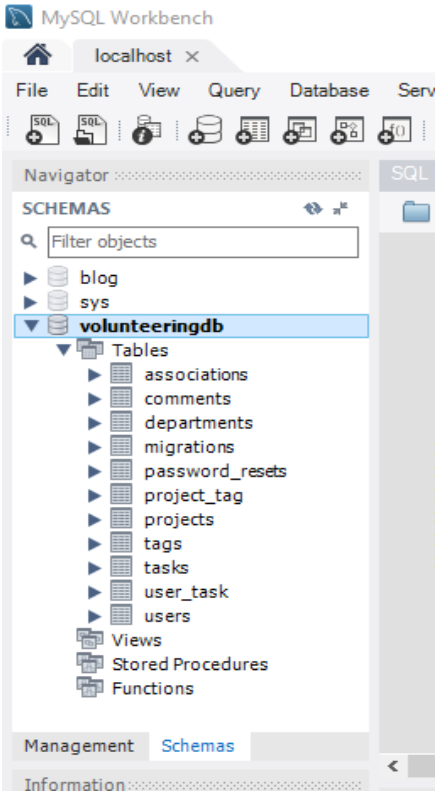


Diagrama bazei de date a fost generată cu ajutorul MySQL, astfel că legaturile relevă structura bazei date creată. Această operație a fost posibilă prin opțiunea Database, urmată de comanda Reverse Engineer, unde se alege mai apoi baza de date a cărei diagrame se dorește a fi realizată.



Baza de date este, probabil, cel mai important aspect de la care se pleacă odata ce se începe un proces de dezvoltare. Înainte de stabilirea funcționalităților și scrierea codului pentru a aduce aplicația la viata, este nevoie de o structură solidă a bazei de date pentru a se identifica fiecare entitate ce joaca un rol în aplicație. Am conceput-o într-un mod cât mai simplist, ușor de urmărit și totodată în conformitate cu realitatea. În cele ce urmează se va discuta fiecare tabelă a bazei de date realizate, pentru a se înțelege mai bine legatura dintre elementele acesteia și platfoma web.

Este de precizat faptul că în realizarea bazei de date id-urile au fost alese de tip „increment” ce realizează incrementarea automata fără o intervenție suplimentară a programatorului, dar și pentru a asigura unicitatea viitoarelor înregistrări.

Aplicația pornește de la tabela ASSOCIATIONS (ASOCIAȚII) ce cuprinde informații despre asociațiile care se înscriu in Manage Vol. Deși s-a mai precizat faptul că, drept clienți sunt voluntarii, înainte de aceștia, vom privi asociația precum client principal. Caracteristicile unei asociații sunt: Id-ul, Numele, Descrierea, Locația sediului, Domeniul activității unei asociații, LogoImg ce a ajutat în stocarea imaginilor – acest câmp reține logo-ul fiecărei asociații în parte și, nu în ultimul rand, Created_at (Creat_la) si Updated_at (Modificat_la) care conțin în mod

automat momentul (data, ora, minut, secunda) la care a fost creată o asociație sau a fost modificat ceva în cadrul ei. Coloanele din urmă, Creat_la si Modificat_la sunt de tip Timestamp care iau data în funcție de zona curentă și sunt create în mod automat de catre farmework-ul Laravel în momentul creării tabelului și precizării campului timestamps() la scrierea migrării.

În continure am presupus că o asociație este compusă din mai multe departamente. Astfel că tabela DEPARTMENTS (DEPARTAMENTE) este legată de tabela asociații printr-o relație de tip „unu la mai mulți” reprezentată prin cheia externă association_id. Aceasta tabela cuprinde cămpuri standard precum: Id-ul, Numele și Descrierea dar și Creat_la si Modificat_la pentru un plus de informație. Pentru a avea siguranța că legatura dintre cele două tabele este una relevantă și în momentul ștergerii, am adăugat alături de constrângerea pentru cheia străină și caracteristica de „onDelete('cascade')” (ștergere în cascadă) care asigură ștergerea departamentelor odată cu ștergerea unei asociații.

Tabela USERS (UTILIZATORI) este una din cele mai importante tabele din această schemă, cuprinzând informații despre utilizatorii aplicației. Această tabelă conține logica din spatele formularelor de Login și de Înregistrare. După cum se poate observa, ea conține cămpuri precum Id, Numele utilizatorului, Email-ul acestuia care are drept caracteristică proprietatea *unique()* deoarece prin intermediul lui un utilizator este identificat în aplicație. Tot în operația de autentificare ia parte și câmpul Remember_token (token de reținut) ce intervine în momentul în care un membru al aplicației este înregistrat în aplicație și are rolul de a stoca un token pentru sesiunea de „remember me” odată ce este bifată, practic, se face apel la cookie-uri pentru a se ține cont dacă utilizatorul este logat sau nu. Parola este una criptată automat de catre framework însă în spate se realizează hash-uirea stringului pentru a nu fi disponibilă în mod direct în baza de date și pentru un bonus de securitate.

Odată realizată autentificarea prin comanda simplă pusă la dispoziție de Laravel (php artisan make:auth), coloanele discutate mai sus au fost create în mod automat. Pentru a putea continua fluxul în baza de date, vom crea o migrare prin care vom adăuga niște cămpuri noi. Admin este cel care identifică utilizatorii de tip administrator, câmpul Approved (Aprobat) este destinat realizării unei filtrări a voluntarilor care vin noi în cadrul unei asociații, necesitând aprobarea administratorului pentru a putea opera în cadrul ei. Pentru a crea legătura cu tabela departamente am creat coloana Department_id ce ilustrează faptul că un departament are în componența sa mai mulți voluntari. În plus, am adăugat și câmpurile de Creat_la si Modifica_la.

Fiecare departament are în același timp proiecte ce aparțin unui departament anume. Astfel a aparut tabela PROJECTS (PROIECTE) ce joaca un rol central in baza de date, iar relația cu departamentele este ilustrată prin cheia străină „department_id”. Aceasta deține coloane precum Id-ul, Titlul proiectului, Body-ul (corpul) proiectului care conține detalii despre acesta. Mai mult, există și un câmp numit Slug ce are rolul de a face posibilă crearea unui link dorit și relevant pentru acel proiect. Acest slug va juca un rol important și în alte funcționalități ale aplicației. Coloana Image (imagine) va stoca la fel ca și în cazul asociației, un string ce ajută la deținerea unor imagini reprezentative pentru proiectul din desfășurare. Pe langă datele clasice la care a fost creat si modificat un proiect, acesta mai are nevoie și de o Begin_date (data de inceput) si de o End_date (data de sfarsit) pentru a identifica proiectele care sunt în desfășurare.

Pentru a ilustra cât mai bine organizarea, un proiect este spart in mai multe sarcini. Tabela TASKS (Sarcini) conține Id-ul, Numele sarcinii, Descrierea sarcinii si datele la care au fost modificate si create pentru o evidenta căt mai bună a acestora. Un câmp interesant este cel de IsManager (EsteManager) care are drept scop modificarea rolulului utilizatorului în Manager de Proiect. Soluția de a crea un manager de proiect este aceea de a crea un task separat ce îi modifică statutul din cel de voluntar în acest rol și a fost abordată în acest fel deoarece de obicei

într-o asociație acțiunile fiecăruia au loc în urma unor discuții și se promoveaza foarte mult încrederea.

Existând o relație de tip „mai mulți la mai mulți” în cadrul legăturii dintre Taskuri si Utilizatori, problema a fost rezolvată prin crearea celei de-a treia tabele, USER_TASK (Utilizator_Sarcina), care joacă rolul de tabela de legătură.

Pentru un plus de efect și în același timp de funcționalitate în ceea ce privește căutarea proiectelor de interes, a fost introdusă tabela TAGS (Taguri). Ea conține strict ceea ce este necesar și anume Id, Nume și datele de creare și modificare.

Tabela de mai sus se află într-o relație de tip „mai mulți la mai multi” cu tabela proiecte, un tag fiind relevant în același timp pentru mai multe proiecte, dar și un proiect deținând în același timp mai multe taguri ce îl reprezintă. Soluția a fost de crearea tabelei de legătură PROJECT_TAG ce conține id-urile proiectelor și a tagurilor.

Posibilitatea utilizatorilor de a lăsa comentarii în cadrul unui proiect a fost posibilă prin tabela COMMENTS (Comentarii). Aceasta conține Id-ul, Numele utilizatorului autentificat, Email-ul corespunzător, Comentariul propriu-zis, coloana Approved (aprobat) pentru o viitoare funcționalitate de a filtra comentariile postate și datele standard pentru evindenta postării acestora.

Datorită framework-ului Laravel, există și tabelele MIGRATIONS (migrări) și RESET_PASSWORD (resetare parolă). Tabelul de migrări este benefic și are rolul de a ține un istoric al creării bazei de date, iar cel de resetare_parolă face parte din elementele create automat odată cu realizarea autentificării.

5. Idei și îmbunătățiri ulterioare

Ideile pentru această aplicație nu se opresc la cele prezentate, ci dimpotrivă, odata ce o aplicație este prezentată în mod oficial, trebuiesc adunate toate ideile ce nu au reușit să fie implementate și luate în considerare pentru o versiune viitoare.

Deși aspectul aplicației este unul destul de omogen, mereu este loc de mai bine și mai interactiv în ceea ce privește modalitatea de vizualizare a diferitelor elemente ale platformei, cum ar fi vizulizarea voluntarilor de către alți voluntari, vizualizarea sarcinilor și cine s-a implicat în ele până la acel moment. Pe lângă aspect, viuzalizarea sarcinilor ar putea fi completată foarte bine de dispunerea acestora într-un fel de calendar dinamic pentru o mai bună evidențiere a cronologiei lor.

S-a putut observa că administratorul are destul control asupra creării și modificării unor componente din aplicație, însă în ceea ce privește fitrarea comentariilor nu există nicio formă de gestionare. O asociație de voluntari este bazată pe încredere și pe respactarea opiniei fiecăruia, însa se pot posta comentarii răutacioase sau poate interveni efectul de spam asupra unui proiect. Astfel că, ar trebui să existe posibilitatea ștergerii comentariilor.

Pentru că este un site ce vizează organizații ce pot atrage parteneri sau sponsori, ar trebui integrate aceste elemente, ori pentru publicitate, ori pentru a se putea ține o evidență a acestora. Aceste modificări ar putea necesita introducerea unor tabele noi în schema bazei de date. De asemenea, ar putea fi benefic un sistem de anunțuri care să fie diferit de proiecte și eventual, marcate distinctiv. Aceaste anunțuri ar putea îmbunătăți modul de comunicare a unor informații în cadrul asociației.

Ca în orice grupare de oameni, în acest caz, conduși de propria motivație, un imbold ar putea deveni evidențierea celui mai activ și mai implicat voluntar, căruia să i se dedice o pagină o data la șase luni, cu

poze, citate de-ale sale și alte feluri de a i se face apreciată munca depusă. O altă idee asemănătoare, este aceea de a premia Project Managerul unui proiect voluntarul care a ajutat cel mai mult în proiectul coordonat de el.

6. CONCLUZII

Conceptul acestei aplicații web a pornit din combinarea unei idei foarte întâlnite printre produsele software actuale, acela de management al diferitelor resurse și un fenomen foarte practicat în zilele noastre și anume, voluntariatul. O organizație precum este cea de voluntariat, nu dispune de o arhitectură specifică când vine vorba de platforme online. Astfel s-a reușit identificarea necesității de întocmire a unui soft ce manageriază și monitorizeaza activitatea din cadrul unui ONG.

Construirea individuală a unei astfel de aplicații, ne poate pregăti spre o vizune mai amplă atunci cand vine vorba despre dezvoltarea software, realizând că elaborarea unei aplicații software este mai mult decât codul în sine. Sunt de asimilat diferite abilități precum cea de filtrare a cerințelor clientului, identificarea cu acesta și imaginarea unui flux cât mai intuitiv și mai prietenos. De asemenea este nevoie de conștientizarea punctelor tari pentru a promova și crede în produsul dezvoltat, dar și a celor slabe pentru a nu omite posibilitatea apariției unor amenințări în procesul de dezvoltare sau folosire.

Este adevărat că drumul în realizarea acestei aplicații, la fel ca oricare alta, a fost pavat cu erori și, totodată, multiple posibilități de rezolvare a unei probleme. Astfel că, fiind un domeniu vast ce se bazează pe logică și gândire, a fost o provocare alegerea unei variante de abordare a unei probleme, necesitând adesea o analiză mai atentă a acestor variante.

Desăvârșirea acestei lucrări și mai ales a proiectului în dicuție a reprezentat o oportunitate foarte bună pentru consolidarea cunoștințelor de SQL și cele de front-end. De asemenea, am învățat framework-ul

Laravel, care este unul foarte puternic și care are de oferit foarte multe în domeniul web. Toate acestea combinate, au dus la o dezvoltare înaintată a cunoștințelor și înțelegerii conceptului de construire a modelului MVC și felul în care comunică fiecare componentă în parte.

În final, platfoma construită dorește să servească pentru îmbunătățirea serviciilor și performanței pentru voluntarii implicați în evenimente și proiecte pe care o asociație nonguvernamentală își propune să le ofere.

Bibliografie

- http://www.health.harvard.edu/special-health-reports/simple-changes-big-rewards-a-practical-easy-guide-for-healthy-happy-living

- http://www.hongkiat.com/blog/best-php-frameworks/

- http://www.amarinfotech.com/why-laravel-is-best-php-framework-in-2016.html

- https://laravel.com/docs/5.3 - documentația oficială Laravel

- https://www.projectsmart.co.uk/moscow-method.php

- https://ro.wikipedia.org/wiki/Wiki