MINISTERUL EDUCAȚIEI NAȚIONALE
UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

PLATFORMĂ INTELIGENTĂ DE EVALUARE ȘI PROGNOZĂ PENTRU START-UPURI

LUCRARE DE LICENŢĂ

Absolvent: **Eduard LUPACESCU**

Coordonator ştiinţific: **Asis. ing. Cosmina IVAN**

2016

DECAN,
Prof. dr. ing. Liviu MICLEA

DIRECTOR DEPARTAMENT,
Prof. dr. ing. Rodica POTOLEA

Absolvent: **Eduard LUPACESCU**

PLATFORMĂ INTELIGENTĂ DE EVALUARE ȘI PROGNOZĂ PENTRU START-UPURI

1. **Enunţul temei:** Sistemul își propune realizarea unui mediu de lucru online, care ar facilita întreg procesul de creare, gestionare și implementare a unui plan de afaceri. Proiectul își propune să ofere acces rapid la toate datele de importanță critică în cadrul firmei pe care se dorește implementarea planului (planurilor) de afaceri. Se încearcă a se oferi alternative metodelor clasice de calcul a rapoartelor financiare cu aspect standard, se vizează eficientizarea proceselor de gestionare a capitolelor planurilor de afaceri, a datelor financiare și a echipelor de lucru din firma. Oferirea posibilității de configurare a tuturor parametrilor pe care se lucreaza, a tipurilor de firma, taxa TVA, perioada de monitorizare a proiectului (ani), precum și alte configurări generale care se vor pe fiecare tip de organizare juridica.
2. **Conţinutul lucrării*:*** Cuprins, Introducere, Obiectivele Proiectului, Studiu Bibliografic, Analiză și Fundamentare Teoretică, Proiectare de Detaliu și Implementare, Testare și Validare, Manual de Instalare și Utilizare, Concluzii, Bibliografie, Anexe.
3. **Locul documentării**: Universitatea Tehnică din Cluj-Napoca, Departamentul Calculatoare
4. **Consultanţi**: Asis. ing. Cosmina IVAN
5. **Data emiterii temei:** 1 noiembrie 2016
6. **Data predării:** 28 Iunie 2016

Absolvent: _____

Coordonator ştiinţific: _____

Cuprins

Capitolul 1. Introducere

În momentul de față, în majoritatea domeniilor se încearcă o creștere a eficienței activităților de rutină prin introducerea, pe cît de mult posibil, a sistemelor automatizate. Astfel se obțin performanțe mult mai mari, iar costuri și resurse umane tot mai mici.

Dezvoltarea serviciilor si a soluților web pentru diferite probleme a ajuns sa să posede astăzi una dintre cele mai eficiente creșteri la nivel mondial. Dezvoltarea limbajelor și tehnologiilor au o crestere exponențială datorită dezvoltării comunităților pe tehnologii și respectiv a produselor libere de consum (open source). Arhitecturile, numărul de nivele de logică, bazele de date - deasemenea au o ascensiune uimitoare, ajungând sa se uitilizeze baze de date mixte pentru soluționarea anumitor tipuri de probleme. Scripturile de gestionare și eficientizare a cautărilor la nivelul bazelor de date de asemenea încep sa aibă abordări mixte, pentru anumite situatii, uneori se folosesc chiar librarii din limbaje inrudite, sau optimizari de protocoale in comunicare.

1.1. Contextul proiectului

Trăim într-o lume în care tehnologia joacă un rol foarte important, accesul rapid la informație, actualizarea acesteia în timp real, sunt astăzi niște standarde fără de care nu ne-am putea imagina existența. În majoritatea sferelor de activitate, în care se pot realiza inovații tehnologice – se fac cercetări pentru a găsi metode de eficientizare a proceselor și produselor deopotrivă. Domeniul business-ului și a afacerilor este motorul care mișcă multe alte sfere, de aceea ar fi foarte necesar de implementat un sistem unic (la nivel de țară), complet din punct de vedere al prevederilor economice ale statului, scalabil si stabil pentru a fuziona toate aspectele afacerilor și a oferi un suport inteligent în suportul acestora.

Sistemul propus în această lucrare face parte dintr-un context mai larg al sistemelor de management, drept exemplu - BPM. Sistemele de management ale proceselor de business (BPM – business process management) reprezintă abordări structurate de modelare și de optimizare a activităților curente ale unei firme, operând cu acestea sub forma de procese[1].

Acest sistem va veni în ajutor tuturor actorilor implicați în procesul de business, proprietarul organizației, echipa de lucru, parteneri de afaceri, îndrumători de activitate. Proprietarul firmei are responsabilitatea de a-și pune la punct planul de afaceri, a forma echipa de lucru, de a face niște previziuni financiare pe anumite termene de activitate. Echipa de lucru va face parte dintr-o organizație, și va trebui să introducă și să supervizeze datele de intrare, plățile și încasările firmei pe baza rapoartelor lunare. Partenerii de afaceri sunt o pîrghie foarte importantă în evoluția fiecei afaceri, de aceea trebuie să se creeze o relație, la nivel de interese, între aceștia. Ei au responsabilitatea de a se conecta între ei prin intermediul sistemului, colaborând pe diverse aspecte financiare. Îndrumătorii de business reprezintă etalonul spre care,

de cele mai multe ori, un tînăr antreprenor, încearcă să ajungă. Astfel îndrumătorii sunt cei care pot să se înregistreze în cadrul organizației pe bază de invitație, ei au responsabilitatea de a urmări și prognoza evoluția firmei pe baza rapoartelor. În cazul în care se observă abateri de la prognoza inițială, îndrumătorul îl va înștiința pe proprietarul afacerii de aceste neregularități pentru a lua o anumită decizie. Datorita faptului că în sistem se pot separa activitățile pe roluri și permisiuni, acesta pune la dispoziție un spectru larg de configurații personalizate care poate modela toate situațiile actorilor de mai sus. Activitățile într-o firmă sunt foarte variate, fiind desigur în funcție de ce ofera, fie servicii sau produse. Sistemul realizat înglobează această componentă, fiind ușor configurat în funcție de necesitățile firmei.

1.2 Motivația proiectului

În urma analizei activităților și a timpului consumat pentru acestea în cadrul dezvoltării afacerilor, s-a constatat că multe dintre acestea fac parte din domeniul calculelor rapoartelor, care au în spate multe formule destul de complexe. Pe lângă acestea, multe activități sunt organizaționale, de planificare a pașilor următori de lucru, delegarea activităților în cadrul echipei și de categorizare a acestora. În urma acestor analize, a apărut concluzia ca majoritatea acestor procese pot fi ușor organizate într-un sistem automatizat, care să respecte caracterul juridic al firmei în care are loc. Această concluzie duce la ideea că, având un modul de configurare destul de permisibil, toate aceste activități pot fi adaptate oricarui tip de business în cadrul acestui sistem online.

Necesitatea unui astfel de sistem în cadrul firmelor, mai ales la început de drum, ar putea deveni indispensabilă și de un real ajutor celor deja în funcțiune, reprezentând un instrument auxiliar activității fizice care este realizată în dezvoltarea firmei. În urma analizei la nivel de țară, s-a constatat că în România, încă nu există un astfel de sistem centralizat, cu extensiuni configurabile, înglobând totodată colaborarea dintre mai multe organizații, de care motiv, acest sistem s-ar încadra foarte bine pe fundalul multor firme, accelerând în mod uimitor evoluția acestora.

Performanțele unui astfel de sistem, ar îmbunătăți situația multor afaceri la nivel de țară, și ar fi de ajutor tuturor membrilor din cadrul afacerii. Proprietarul ar beneficia rapid de access la toate punctele de lucru ale firmei, la toate rapoartele financiare ale acesteia pe intervale de timp, ar avea acces în timp real la lista de activități a tuturor angajaților și a membrilor echipelor. Managerii ar beneficia de un serviciu prin care să poată actualiza lista de probleme ce împiedică evoluția echipei sale, prezentând soluțiile sale, dar în același timp avînd posibilitatea de a cere părerea unui consultant din orice loc al țării, și nu numai, acordând-ui o invitație în sistem, pentru ca acesta din urma sa facă cunoștință cu situația actuala și istoricul financiar al firmei.

O confirmare și imbold spre un astfel de sistem l-a avut și studiul realizat în urma discuției cu câțiva proprietari de afaceri din București și din Cluj Napoca. Toți trei au confirmat faptul că necesitatea unui astfel de sistem în România este indispensabil, deoarece la nivelul tării antreprenorii încearcă să utilizeze sisteme de această natură, dar fiind proprietăți ale altor state,

nu pun la dispoziție necesitățile juridice locale, ceea ce implică un impediment uriaș când vine vorba de raportare la sfârșit de an și nu numai.

Tot în urma studiului cu alti antreprenori, s-a făcut rost de toate formulele pentru generarea previziunilor financiare, s-au realizat liste de necesități implicate în activitățile lor de zi cu zi. Aceste date au fost adunate din mai multe surse, pentru a se face o reprezentare a necesităților, privită prin ”Clopotul lui Gauss”, fiind o pistă clară în dezvoltarea unui astfel de sistem automatizat.

1.3 Conținutul lucrării

În cele ce urmează se va prezenta pe scurt structura lucrării, capitolele și continutul acestora.

Capitolul 1 – Introducere – Acest capitol conține o scurtă prezentare a contextului problemei care se dorește a fi rezolvată prin sistemul automatizat care s-a realizat.

Capitolul 2 – Obiectivele proiectului – Se va face prezentarea si descrierea obiectivului principal al proiectului, dar și obiectivele secundare care au fost propuse spre îndeplinire de către sistemul de management al planurilor de afacere.

Capitolul 3 – Studiu Bibliografic – Acest capitol conține studiul necesar dezvoltării și întreținerii aplicațiilor web. Va fi prezentat un studiu relevant cu privire la tendințele tehnologice din ultimii ani. Se vor realiza comparații între tehnologii, modalități de suport ale platformelor cu un număr mare de intrări. Se vor clarifica câteva dintre cele mai importante protocale de comunicare utilizate de sistem. Totodată, acest capitol evidențiază asemănările și deosebirile în ceea ce privește funcționalitățile, modelul de arhitectura, tehnologiile utilizate, dintre acest proiect și proiecte asemănătoare ca tematică.

Capitolul 4 - Analiză și fundamentare teoretică – Aici sunt structurate tehnologiile utilizate în realizarea sistemului, fiind structurate pe tipuri și subtipuri. Se vor face analize de securitate și scalabilitate al tehnologiilor. Se va justifica combinarea tehnologiilor utilizate în sistem. Se vor prezenta cerințele funcționale și non-funcționale, si cazurile de întrebuințare ale acestora în cadrul aplicației. Tot aici se vor dezvolta utilitarele folosite în proiect, și o reprezentare sugestivă în ceea ce privește cazurile de utilizare ale acestui sistem.

Capitolul 2. Obiectivele Proiectului

Acest capitol va cuprinde descrierea detaliată a obiectivelor primordiale și secundare ale sistemului realizat. Sistemul are ca scop general eficientizarea dezvoltării planurilor de afacere în conformitate cu politica economică din România, venind în sprijinul antreprenorilor.

2.1 Obiectivul principal

Obiectivul primordial al acestui sistem automatizat este de a defini o platformă centralizată pentru gestiunea planurilor de afaceri, cu posibilitatea configurărilor personalizate în funcție de domeniul de activitate, istoricul activității firmei, conformarea după ultimele standarde economice ale statului, dispunând în acest sens de un modul inteligent de sugestii și beneficii colaborative cu alte firme și investitori sau îndrumători.

2.2 Obiective specifice

- **Obiective generale propuse**

Un obiectiv general propus al acestei platforme informatice de planificare a resurselor întreprinderii este implementarea unui sistem integrat de management al planurilor de afaceri din cadrul organizației, cu oferirea accesului rapid la **informațiile din domeniul consultanței financiare** și **previziune financiara in funcție de anumite decizii luate de utilizator.**

Efectuarea de analize economico-financiare pentru ca platforma să poată lua decizii în funcție de clienții pe care îi administrează, de vânzări, echipe etc. Previziunile financiare fiind gandite pentru diferite tipuri de firma cu o perioada de calcul de pana la 5 ani. Se va ocupa și de diagnosticarea financiară și expertiza financiară asupra întregii activități, elaborări economico-financiare, **elaborarea documentațiilor (export PDF)**, calcul intrărilor și ieșirilor din firmă, date necesare pentru obținerea de credite bancare, elaborarea și urmărirea planurilor de afaceri pe parcursul mai multor ani în cadrul unei baze de date integrate.

Totodată sistemul trebuie să ofere posibilitatea de **autentificare și recuperare a parolei**, prin metode securizate desigur. Capabilitatea sistemului de a întreține mai mulți utilizatori, cu **diferite roluri și permisiuni**.

Platforma trebuie să dispună de un contact pentru suport rapid de la administratorul platformei.

Fiecare acțiune pe care o face utilizatorul în primă fază este documentată, prezentându-se ca un **ghid de utilizare**, care duce utilizatorul prin toate funcționalitățile principale ale sistemului, indicând utilitatea fiecărui instrument de lucru.

- **Posibilitatea previziunii financiare**

  Sistemul informatic oferă un suport integrat de module care îndeplinesc funcția de gestionare a planurilor de lucru, utilizatorilor conectați prin sistem, previziunile financiare, condițiilor de eligibilitate și a produselor acestora și care oferă o interfață grafică, în limba română, modernă și intuitivă. Acest sistem de modele funcționale reprezintă o soluție software destinată gestionării, conducerii și managementului de informații ale unei companii distribuite geografic și faciliteaza accesul la orice tip de informație din baza de date utilă în desfășurarea activității de planificare, având capabilitatea de a procesa paralel un volum foarte mare de date cu un timp de răspuns mic, datorită tehnologiilor utilizate în procesul dezvoltării aplicației.

  Dispunând de o arhitectură modulară și o baza de date unică, consistentă, sistemul permite utilizatorilor autorizați lucrul online, folosind un browser web, dar și de servicii REST API pentru extinderea sistemului pe dispozitive mobile. Accesul se face pe baza introducerii unui email de utilizator și a unei parole, care au fost aprobate de către un administrator de sistem.

- **Asigurarea managementului intern**

  Dat fiind faptul că utilizatorii aplicației sunt structurați pe roluri, se permite accesul diferențiat la informații. Fiind realizat modular, acest sistem se poate adapta la nevoile oricărui grup, fiind posibilă construirea unui sistem propriu, personalizat. Această platformă informatică oferă și acoperă o varietate de funcții esențiale care integrează departamentele firme, prin monitorizarea în timp real a datelor și proceselor din domeniul de activitate al firmei.

  Sistemul simplifică și facilitează, de asemenea, fluxurile de lucru din fiecare departament al firmei prin:

  - Raționalizarea procesului de afaceri și a fluxuri de lucru din domeniul consultanței privind elaborarea de tablouri de finantare;
  - Oferirea de consultanță managerială pentru fundamentarea proceselor decizionale și efectuarea de analize economico-financiare și studii de fezabilitate;
  - Asigurarea unei bune asistențe și a unor servicii ireproșabile pentru clienți;
  - Diagnosticarea financiară și expertiza financiară asupra întregii activități a societății comerciale;
  - Evaluarea economico-financiară a societăților;
  - Elaborarea documentațiilor necesare în vedera obținerii de consultanță din partea unui expert;

- Oferirea de suport în întocmirea și analiza fluxurilor de numerar;
- Facilitarea întocmirii de finanțare a afacerii;
- Oferirea unei viziuni globale - în timp real - a situației diferitelor planuri de afaceri paralele ale companiei dumneavoastră, care va permite luarea de decizii rapide și eficiente;
- Garantează limitarea cantității de documente fizice (sub formă de hârtie), elimină dublarea intrărilor;
- Deținerea de informații centralizate într-un singur program informatic, actualizate în timp real, utilizatorii vor putea dispune de toate datele necesare pentru a lua decizii informate, care să asigure creșterea companiei.

Sistemul automatizat trebuie să permită gestiunea nomenclatoarelor definite și a rapoartelor centralizate. Această platformă trebuie să suporte următoarele tipuri de utilizatori (care au acces la meniuri diferite): administrator platformă, proprietar plan, invitat plan (consultant) care poate face editare și invitat plan, care poate doar vizualiza secțiunile si eventual pune comentarii.

Capitolul 3. Studiu Bibliografic

În acest capitol se vor prezenta, analiza și evalua un set de sisteme similare care vizează aceeași temă, sistemele analizate fac parte din spațiul european deoarece sisteme similare în spațiul local nu au fost identificate. Sisteme de tip business process management au un impact uriaș asupra multor domenii de activitate, de aceea asupra software-ului dedicat acestui domeniu s-a lurat intens, oferindu-se posibilitatea de a gestiona procesele de afaceri cap coadă, asigurarea unui circuit financiar transparent, vizualizarea și asignarea activităților de lucru echipei, dar și clasificarea tipurilor de activități pe subactivități cu estimare de timp pentru fiecare.

Dat fiind faptul că se prezintă o listă mare de aplicații care dispun de un conținut și gen de activitate comun, înaintea implementării acestui sistem, s-a ținut cont de avantajele și dezavantajele acestora, pentru ca în cele din urmă să se obțină un sistem pe care să-l poată utiliza cu ușurință chiar și un utilizator neexperimentat.

Deasemenea în acest capitol vor analiza câteva concepte de arhitectură similare, integrarea sistemului realizat peste arhitectura utilizată, evidențiindu-se astfel avantajele tehnologiilor alese.

3.1 Dezvoltarea aplicațiilor web

În dezvoltarea aplicațiilor web trebuie să se țină cont de câteva criterii, astfel încât să se asigure o scalabilitate și mentenanță cât mai ușor posibil. Un alt aspect foarte important al aplicațiilor web, este acela de a permite accesul la serviciile realizate mai multor tipuri de clienți (mobile, desktop, alte aplicații web etc.), acest lucru poate fi realizat prin intermediul serviciilor și separarea definitivă a conceptelor de client și server, în acest mod se va putea realiza un produs, serviciile căruia vor putea fi utilizate la maxim. În fond, o aplicație web este un mediu online de prezentare și acces la anumite resurse. Aceste resurse sunt servite de către un server în urma unei cereri de pe orice tip de client, comunicarea se realizează prin intermediul protocolului HTTP (Hypertext Transfer Protocol), iar interogarea resurselor se face prin accesarea unui URI (Uniform Resource Identifier) în browser.

Primele aplicații web puneau la dispoziție utilizatorului doar un conținut static, care era scris în HTML/CSS, cu putin Javascript pentru comunicarea aplicației cu utilizatorul final. Lucrurile au evoluat foarte mult, limbajele de dezvoltare pentru server s-au dezvoltat continuu, dar și interpretoarele de JavaScript, nucleul V8 din browsere s-au dezvoltat deopotrivă, astfel încât, în prezent, majoritatea aplicațiilor web au conținut dinamic. Conținutul dinamic reprezintă răspunsul serverului la acțiunile clientului, fiecare eveniment pe partea de client este prins de către JavaScript și în urma unei comunicări cu serverul, schimbă starea paginii curente.

Arhitectura de comunicare este Client-Server, în care Clientul reprezintă browserul, cel ce procesează toate interacțiunile cu utilizatorul, prinde evenimentele, validează pe partea de

interfață anumite stări, iar Serverul este cel care răspunde cererilor (pe baza adresei URI) Clientului, în linii generale Serverul poate fi privit ca o unitate fizică, pe care rulează un serviciu capabil să proceseze interceptări de tip HTTP (de ex. Apache sau Tomcat), portul implicit pentru acest serviciu este optzeci. Atunci când browserul deschide o conexiune (pe baza adresei) cu serverul, se face o cerere către anumite resurse, dar odată cu această cerere se mai trimit date suplimentare către server, cum ar fi un token de autentificare, serverul analizează datele de intrare, și dacă utilizatorul care a cerut resursa respectivă are acces la ea, i se va trimite înapoi un cod de răspuns (status) și conținutul propriu-zis.

Prin utilizarea arhitecturii Client-Server prin protocolul HTTP se permite accesul la 8 tipuri de interogări: get, post, put, delete, options, head, trace si connect. De obicei introducerea datelor se realizează prin interogarea POST, în acest caz este necesară o validare suplimentară pe partea de server, evitându-se astfel injectările nevalide la nivelul bazei de date.[26]

3.2 Arhitectura Client – Side

Dezvoltarea părții de client, în prezent este realizată prin utilizarea mai multor librării/framework-uri, care au la bază limbajul Javascript. Adaptarea librăriei se face în funcție de necesitățile platformei, de regulile de validare pe client, de logica și diferențierea stărilor (cât de repede se tranzitează de la o stare la alta). Fiecare eveniment care trebuie procesat pe partea de client este înlesnită de intermedierea unei librării specifice. Validarea formularelor pe partea de client este o deprindere foarte utilă deoarece permite utilizatorului să adapteze datele de intrare înainte ca acestea sa ajungă la server, acest lucru economisește din timpul de lucru și îmbunătățește experiența utilizatorului. Un aspect de care trebuie să se țină cont, orice validare pe partea de browser (client) trebuie repetată pe server, acest lucru este necesar deoarece javascript-ul fiind în cadrul browser-ului utilizatorului, acesta în mod intenționat poate să modifice codul (sau să-l dezactiveze) pentru validare, și acest lucru ar trebui evitat pentru a nu băga date neconsistente în baza de date. [26]

3.3 Tendințele sistemelor web

Istoria platformelor web începe încă din jurul anilor 1992, în depărtatul 1995 compania Netscape a introdus un limbaj de scripting pe client – JavaScript. În 1999, conceptul “aplicație web” a fost introdus în limbajul Java în specificația servlet versiunea 2.2 [1]. Atunci deja erau inventate JavaScriptul si XML ca limbaje, dar încă nu era implementată procesarea AJAX, iar obiectul XmlHttpRequest era introdus abia recent în Internet Explorer ca un obiect ActiveX[2]. În prezent JavaScript s-a dezvoltat foarte mult prin compilatorul V8, iar limbajul, conform standardelor w3 a ajuns la versiunea ECMAScript[25] 6 (sau 2015) [6].

Evoluția limbajelor de programare, a claselor, tehnologiilor a dus în prezent la o utilizare masivă a limbajelor JavaScript, C#, Java, PHP, pe baza acestor limbaje continuu se dezvoltă librării, framework-uri și comunități mari care le întreține. Problemele actuale se orientează către partea

de securitate și administrare de sisteme, administrarea sistemelor pentru servere oferă o scalabilitate mare și o viteză mare de procesare.

3.4 Sisteme similare

Dat fiind faptul ca necesitatea de a crea un sistem integru și unic pentru gestiunea tuturor necesităților in scopul realizării unui plan de afaceri, a crescut exponential in ultimul deceniu, sistemele de analiza a acestor probleme, pe baza anumitor algoritmi de implementare și sisteme economice special acomodate mediului de lucru grupului țintă a crescut intr-o maniera exponențiala. Acest lucru este pe de o parte îmbucurator, fiind in primul rand un indicator al unei mișcări la nivel de gândire al tinerilor, pe de alta parte este un instrument care ajuta si pe cei care deja au implementat o structura organizationala in acest scop - aplicatiile de acest gen pun la dispozitie si un modul bine definit pentru a previziona si analiza o organizatie care dispune de un istoric de activitate.

In continuare se vor detalia 3 dintre cele mai populare sisteme similare cu avantajele si dezavantajele asociate. Cele 3 sisteme au stat la baza ideei care a fost implementată, dar e de mentionat ca nici unul dintre ele nu dispune de politică și analiza financiară specifică României.

3.4.1 Liveplan

Primul sistem, cel mai riguros din punct de vedere al securității și al implementării arhitecturii, este *Liveplan (www.liveplan.com).* Cu alte cuvinte, este un sistem care dipune în primul rând de un nivel înalt de securitate, dar e de menționat faptul că timpul de răspuns la interogări este relativ scurt, serverul de Centos ruleaza un *Apache* care, în relație cu nginx, oferă o viteză de răspuns a serviciilor consumate foarte mare. Astfel în urma testelor de intrări multiple a oferit cele mai bune rezultate.

Acest sistem este implementat pe baza de microservicii, și este implementat in php(ca limbaj de bază pentru server). Mai jos se vor prezenta cateva dintre punctele cheie ale platformei, nuanțele tehnice precum și cele de perfomanță și securitate.

3.4.2 Bizplan

O altă platformă din această categorie, este *Bizplan(bizplan.com)*. Aceasta este specifică prin faptul ca este destinată strict startup-urilor și are o vechime mai mare decât platforma anterior enunțată. Dat fiind faptul că înca este pe piață și are o mare popularitate, incluzând actualizări anuale atât în ceea ce priveste design-ul cât și modulele de sevicii pentru aplicațiile mobile - putem face o concluzie clară, platforma are un succes real in fața clienților săi.

3.4.3 Enloop

Enloop (www.enloop.com) este o platfomă în cloud care are o versiune gratis. Are constrâgerea de a nu permite exportul in Word. Suporta șabloane de lucru pentru multiple industrii, în special pentru organizatii non-profit. Este o platforma foarte simplu de utilizat, dispunând de instrucțiuni pas-cu-pas, care iți fac introducere peste toată platforma. Pe lângă lista de funcționalități clasică de funcționalități, acest sistem dispune de un modul de autoscriere, acesta se deosebește prin inteligență, ideea este că în momentul în care se face o descriere a planului, se poate selecta acest regim, care în funcție de previziunile pe care le realizează, le introduce în context. Capacitatea de inviții și distribuire a planurilor este foarte bine pusă la punct de către acest sistem, spre deosebire de alte sisteme similare, Enloop permite distribuirea unui proiect către un utilizator care nu are un cont în sistem, acest lucru oferă o mobilitate ridicată.

Se poate observa că toate sistemele de acest gen au și avantaje și dezavantaje, însă rămân a fi niște unelte foarte puternice care ajută un start-up în alegerea nișei de lucru, oferă servicii pentru distribuire și promovare, oferă un real suport în interacțiunea cu antreprenori cu experiență dispuși să vină cu anumite obiecții.

În cele ce urmează s-a făcut o analiză a celor trei platforme în comparație cu sistemul care s-a realizat. Analiza comparativă a fost realizată la în baza unor criterii de funcționare, și nu în baza aspectelor tehnice ale acestora.

Mai jos se găsește analiza tehnică comparativă sistemelor.

	Liveplan	**Bizplan**	**Enloop**	**Sistemul realizat**
Suportarea rolului de administrator	x	x	x	x
Suportarea rolului de organizație	x	x	x	x
Suportarea rolului de angajat al echipei				x
Suportarea rolului de collaborator	x			x
Administrare cont	x	x	x	x
Recuperare parola	x	x	x	x
Administrare conturi ca superadministrator				x
Modul adaptabil (configurabil) după tipul firmei	x		x	x
Suport politica economică Română				x
Modul de instrucțiuni		x	x	x
Suport realizare plan de afaceri	x	x	x	x
Suport dublicare plan de afaceri				x
Previziuni financiare pentru 5 ani	x			x
Grafice de evoluție	x		x	x
Configurarea echipei și a rolurilor cheie				x
Configurarea pistei țintă și a oportunităților				x
Capitole dinamice pentru planul de afaceri	x			x
Gestionarea activităților	x	x		x
Grafice Gantt de productivitate				x
Trimitere invitație de colaborare	x			x
Generare rapoarte financiare PDF	x	x	x	x
Generarea planului de afaceri PDF				x
Modul de plata euplatesc.ro				x
Trimitere raport generat direct prin email				x
Expirarea conturilor cu invitații	x			x
Notificare prin email despre anumite probleme, sau anunțuri generale		x	x	x

Tabel 3.1 Tabel comparativ între sistemul realizat și cele similar

Capitolul 4. Analiză și fundamentare teoretică

Acest capitol încorporează tehnologiile utilizate în dezvoltarea acestei aplicații web, analiza comparativă cu tehnologii similare, posibilități de îmbunătățire viitoare. Dintre tehnologiile utilizate se menționează: Laravel/PHP framework, Angular 1.4 client framework, Blade, NodeJs, Gulp.

4.1 Tehnologii și concepte utilizate pentru dezvoltarea aplicației web

4.1.1 Laravel/php framework

Laravel 5.1 este un framework bazat pe limbajul PHP, care are ca scop facilitarea dezvoltării părții de server side a aplicațiilor web. Acesta vine cu multe librării disponibile prin utilitarul Composer, și care au codul sursă liber, cu licență MIT. Framework-ul folosește tehnicile orientate pe obiect, fiind în primul rând destinat aplicațiilor MVC, dar nu are o limitare în această direcție. Detalii suplimentare se pot găsi pe site-ul dezvoltatorului [3]. În graficul de mai jos se observă popularitatea acestei tehnologii în ceea ce privește proiectele personale ale programatorilor [4].

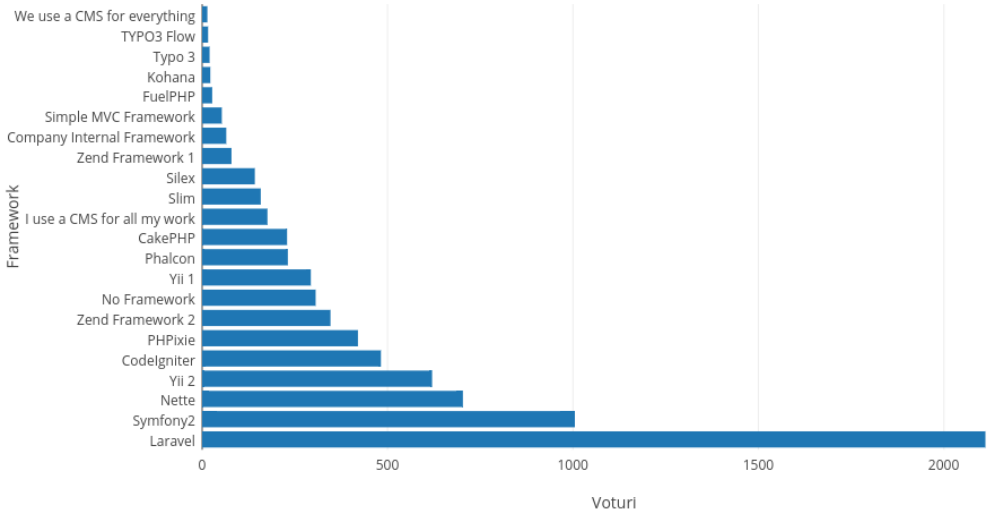

Fig. 4.1 Popularitatea framework-urilor PHP pentru proiecte personale 2015

Mai jos se prezintă popularitatea framework-urilor PHP la locul de muncă [4], după cum se poate observa, Laravel depășește cu mult pe celelalte tehnologii, cu toate că a apărut relativ recent.

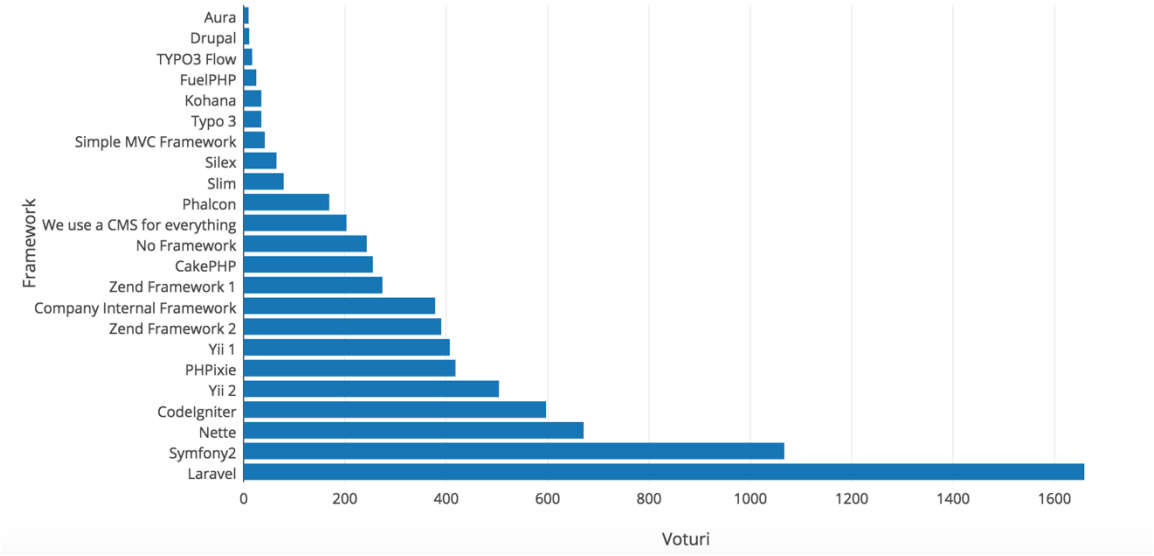

Fig. 4.2 Popularitatea framework-urilor PHP la locul de muncă 2015

În ceea ce privește performanța framework-ului, se poate menționa faptul că este undeva la mijloc, datele statistice de care dispunem sunt bazate pe versiunea 3.2 a framework-ului, și arată în felul următor [5]:

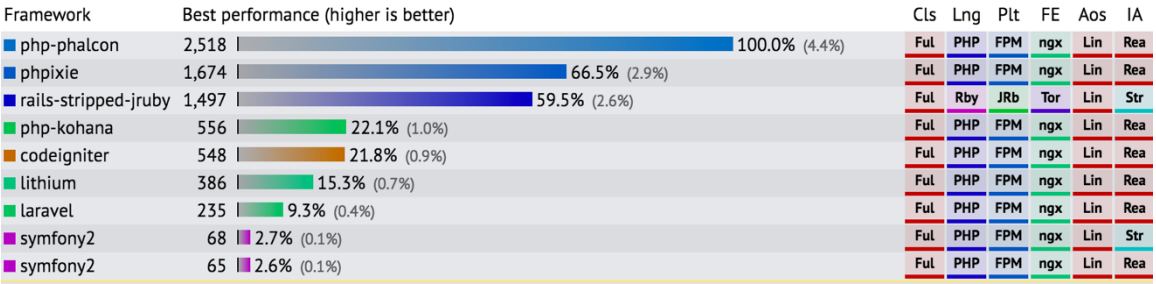

Fig. 4.3 Performanța framework-ului comparativă cu a altor framework-uri PHP

4.1.2 Eloquent ORM

Pentru lucrul cu bazele de date în prezent sunt utilizate utilitare spciale, librării de clase destinate mapării tabelelor (în cazul MySQL) în obiecte pe care se poate opera la nivelul

arhitecturii. Un dezavantaj al acestui tip de abordare este faptul că reduc din viteza de procesare (comparativ cu o procedură stocată de exemplu). Eloquent are la baza unul dintre cele mai populare ORM-uri pentru PHP - Doctrine. Utilizarea unui astfel de ORM este foarte simplă, pentru a mapa spre exemplu tabelul din baza de date cu o clasă utilizată, este suficient să se extindă clasa de bază (Model), iar numele clasei trebuie sa coincidă cu numele tabelului din baza de date (în cazul MySQL).
Tipurile de relații care se pot realiza nu sunt limitate (unu la unu, unu la mai multi, mai mulți la unu). Deasemenea cu ajutorul acestui ORM se pot realiza foarte usor ștergeri inteligente în baza de date, adică nu facem ștergerea fizică, ci actualizăm o coloană (deleted_at) cu momentul de timp în care s-a realizat ștergerea, iar la filtrare acea coloana se va cerceta dacă este nulă.
Eloquent permite captarea de evenimente pe tabelul ce este mapat, acestea pot fi prelucrate înainte de a se valida, sau se pot face alte operații de analiză ca să nu se permită introducerea datelor neconsistente în baza de date.

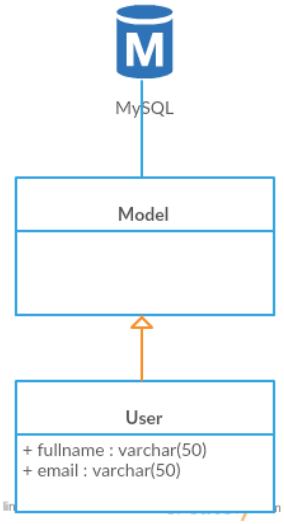

Fig. 4.4 UML pentru reprezentarea generică a arhitecturii Eloquent

4.1.3 Sessions Drivers

Sesiunea nu este altceva decât o conexiune deschisă și întreținută pe toată perioada comunicării între client și server. Durata acestei conexiuni de obicei este nedeterminată, dar poate fi stabilită și de către server (programator). Avantajul sesiunilor este că se evită reautentificarea utilizatorului la anumite request-uri care sunt trecute prin filtre. În Laravel putem utiliza mai multe tipuri de drivere de sesiune, acest lucru este foarte important deoarece în funcție de caracteristicile aplicației și ale numărului de utilizatori unici in fiecare moment de timp, se poate configura în mod diferit acest driver (precum și parametrii acestuia).

Driverele de sesiune pot fi :

- file - sesiunea va fi salvată în fișiere
- cookie - sesiunea va fi salvată într-un cookies securizat și encryptat
- database - sesiunea va fi salvată în baza de date folosită de aplicație
- memcached / redis - sesiunea va fi salvată într-o memorie cache foarte rapidă
- array - sesiunea va fi salvată în tablouri php unidimensionale, dar nu va fi disponibilă pe mai multe request-uri

Deasemenea sunt puse la dispoziție posibilități de a salva sesiunea doar pe perioada unui singur request (cel următor), această sesiune este numită - flash, și are o utilitate deosebită de exemplu în cazul notificării utilizatorului cu anumite mesaje de succes sau eroare.

4.1.4 Swift Mailer

Pentru a face posibilă comunicarea prin emai-uri, se folosește un protocol special - SMTP, prin intermediul căruia se pot face atașamente, trimitere și primire de mesaje (email). SwiftMailer este o librărie ce pune la dispoziție un API simpu și curat, peste această librărie se poate realiza foarte ușor o implementare de clase pentru a o utiliza în contextul proiectului, pentru aceasta se poate utiliza pachetul "guzzle" [7]. Peste swiftmailer putem folosi mai multe drivere pe care le pune la dispoziție Laravel prin libraria *guzzle* - acestea sunt: *mailgun și mandrill.*
Configurarea serviciului de email se poate realiza în fișierul *config/mail.php,* de obicei aici trebuiesc setate căsuța de ieșire, portul utilizat este (de obicei) 587, criptarea mesajelor se realizează cu tls. Când avem de trimis mesaje multiple, se poate folosi o implementare de cozi, în fiecare coadă se poate pune câte anumite categorii de mesaje, care vor fi trimise asincron atunci când serverul se eliberează ca solicitare.

4.1.5 Blade template engine

Blade [8] este o tehnologie puternică, totodată este un instrument simplu și util pentru interpretarea părții de *view* din arhitectura MVC. Acest motor de transformare a instrucțiunilor în partea de *view* provine de la ideea template engine-ului Mustache[9]. Toate fisierele de tip blade sunt compilate în cod PHP și stocate în cache până când nu este modificat. Fișiere blade sunt stocate în directorul *resources/views*.
Această tehnică are mai multe avantaje:

1. **Definirea template-ului** - pentru fiecare pagină servită utilizatorului (pe cât de mult posibil) este utilizată o singură instanță generică, care rămâne fixă, iar

paginile care se modifică sunt injectate în zone specificate de programator prin acronimul *@yield('nume_zona')*[8].

```
<html>
    <head>
        <title>App Name - @yield('title')</title>
    </head>
    <body>
        @section('sidebar')
            This is the master sidebar.
        @show

        <div class="container">
            @yield('content')
        </div>
    </body>
</html>
```

2. **Extinderea template-ului** - după ce s-a definit pagina generică, toate celelalte pagini o vor extinde, și vor injecta în zonele specificate cu *@yield* anumit conținut. Conținutul poate fi injectat doar în cazul în care fișierul pe care se lucrează (care este servit spre compilare) extinde explicit template-ul generic prin acronimul *@extends('nume_template'),* în loca de *nume_template* se va indica calea de la directorul *resource* până la fișierul generic, fiecare pas în adâncime se va separa prin "." (punct).

De exemplu:

```
@extends('layouts.master')

@section('title', 'Page Title')

@section('sidebar')
@parent

<p>This is appended to the master sidebar.</p>
@endsection

@section('content')
<p>This is my body content.</p>
@endsection
```

3. **Afișarea și interpretarea datelor** - din controller este foarte simplu de a trimite date spre view, dar interpretarea acestora devine destul de greoaie dacă nu se

utilizează un template engine. Blade însă permite interpretarea datelor cu ajutorul acoladelor duble *"{{ $nume_variabila }}"* , este o manieră curată și foarte utilă în acest scop.

4. **Structuri de control** - adevărata putere a acestui compilator se vede în utilizarea intrucțiunilor de control. De obicei aceasta este o practică rea, logica ar trebui separată doar în controllere, dar sunt și situații în care acest lucru nu poate fi evitat. Blade ne permite utilizarea urmatoarelor structuri:
   - **Instrucțiunea if -** pentru această instrucțiune se pun la dispoziție următoarele construcții *@if, @elseif, @else* și *@endif*. Exemplu de utilizare [8]:

```
@if (count($records) === 1)
    I have one record!
@elseif (count($records) > 1)
    I have multiple records!
@else
    I don't have any records!
@endif
```

   - **Cicluri (loops)** - Blade vine cu un set util de directive pentru lucrul cu structurile de cicluri suportate de PHP. Aceste directive funcționează identic cu instrucțiunile asociate din PHP. Directivele suportate sunt: *@for, @foreach, @forelse, @while*. Exemplu de cod:

```
@for ($i = 0; $i < 10; $i++)
    The current value is {{ $i }}
@endfor
```

4.1.6 Baza de date

Baza de date este modalitatea de stocare a unor informații și date pe un suport extern (un dispozitiv de stocare). Datele sunt stocate în mai multe fișiere, unele dintre acestea pot fi fragmentate (dacă includ un număr mare de intrări). Manipularea bazelor de date se realizează cu ajutorul sistemelor de gestiune a bazelor de date. Pentru proiectele mari, de obicei se folosesc un model de baze de date relaționale, propus de E.F. Codd[14] în 1970, care organizează datele sub formă de tabele (coloane și rânduri) cu o cheie unică pentru fiecare rând nou [10]. Acest tip de baze de date permite:

- să se creeze relații între entitățile fizice create (chei străine)
- să se creeze operații consistente prin tranzacții ACID
- să se indexeze anumite coloane pentru a permite o căutare mai rapidă
- să se genereze proceduri stocate, pentru a fi ulterior apelate de către server

4.1.6.1 MySQL

MySQL Community Edition este cea mai populară bază de date open source din lume, ce vine cu o comunitate foarte dezvoltată [15]. Baza de date poate fi manipulată atât din linia de comandă, cât și din programe specializate pentru acest scop.

La nivel mondial MySQL a fost și este, pentru anumite tipuri de date, folosit pentru și la scară largă de către așa companii ca Google[16][17], Facebook[18], [19],[20] și Twitter [21].

4.1.7 Composer

Composer este un tool pentru a rezolva managementul dependințelor PHP. El permite declararea acestor librării (dependințe) într-un fișier special - *composer.json,* iar cu ajutorul CLI(command line interface) se poate rula comanda *composer install (update)* și toate dependițele definite vor fi instalate.

Pe lângă acest lucru, composer mai permite compilarea claselor, și generarea fișierelor de cache cu view-uri. Tot în composer.json se specifică fisierele (directoarele de fișiere) care se doresc a fi compilate pentru autoload, iar în urma rulării comenzii *composer dump-autoload -o* - se va genera un fișier autoload.php, care va ține referințe spre toate dependințele, clasele și funcțiile globale din proiect.

4.1.8 AngularJS

AngularJS este un framework JavaScript foarte puternic. Acesta folosește arhitectura Single Page Application (SPA). Framework-ul extinde DOM-ul HTML cu atribribute funcționale suplimentare care permit interacțiunea cu utilizatorul, captarea evenimentelor, afișarea datelor, interpretarea și serializarea datelor introduse etc. Fiind un proiect inițiat de gigantul Google, ce are codul sursă disponibil, comunitatea acestuia a ajuns la mii de dezvoltatori din toată lumea care contribuie la dezvoltare și detectare de excepții [22].

Design pattern-ul în care se încadrează AngularJs este - MVC (model-view-controller).

- **Model** - drept model servește starea aplicației, datele comunicate spre view de către controller (prin directive sau direct)
- **View** - partea de afișare și interpretare expresii, AngularJS folosește Mustache personalizat, ca fiind propriul său template engine
- **Controller** - aici se găsește logica de control, captarea evenimentelor, procesarea datelor înainte de serializare, validarea formularelor etc.

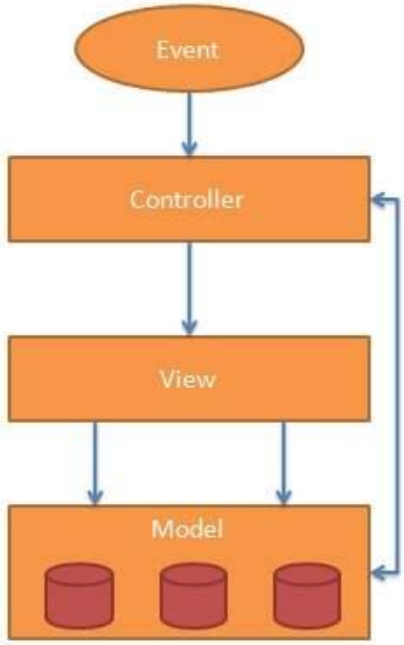

Fig. 4.5 Reprezentarea design pattern-ului MVC în cazul AngularJS

4.1.9 Bower

Pentru a controlara o aplicație mare în faza de dezvoltare, o atenție deosebită trebuie acordată modularizării acesteia. Trebuie găsite cele mai simple metode de a gestiona dependințele injectate, fie pe partea de client (frontend) sau server (backend). Mai sus a fost remarcat *Composer* pe partea de server, pe partea de client, este un tool similar - numit *Bower*[23]. Fișiere de configurare sunt: bower.json (dependințele) și .bowerrc (configurarea locației de instalare a dependințelor).

Pe lângă avantajul de rezolvare a dependințelor, bower oferă o listă de comenzi CLI pentru operarea cu cache-ul fișierelor de JavaScript și CSS. Tot cu bower se poate realiza o configurare CDN, care să permită încărcarea fișierelor statice de pe servere apropiate locației fizice a clientului care a accesat resursa dată. Această practică este folosită de toate platformele care au acoperire globală (Google, Facebook, Ebay, Amazon etc.).

4.2 Cerințe

Sistemul pune la dispoziție o lista de cerințe, constrângeri, obiective și funcții care trebuie să se mapeze peste așteptările utilizatorilor. Acestea au fost organizate în așa fel încât să fie ușor citite, dispunând de o organizare pe capitole și subcapitole.

În parea de cerințe functionale se descriu operațiile efective pe care le poate efectua sistemul în acest stadiu de proiectare, cu anumite indicii de intrări și eventuale așteptări de ieșire.

În sectiunea de cerințe non-functionale se vor detalia constrângerile sub care trebuie să opereze sistemul, precum și standardele fizice necesare pentru sistemul software să funcționeze.

4.2.1 Cerințe funcționale

Mai jos se vor detalia capacitățile funcționale ale sistemului. Pentru o mai bună înțelegere a cerințelor funcționale, acestea vor fi prezentate într-o organizare de capitole și subcapitole în funcție de tip.

	Descrierea cerinței funcționale	**Nivelul de utilizator**
CF 1	**Autentificare - recuperare parola**	**Toate**
CF 1.1	Administrarea contului personal	Toate
CF 1.2	Adăugare imagine de profil	Toate
CF 1.3	Posibilitatea înregistrării prin invitație	Toate

Tabel 4.1 Cerințe funcționale generale

CF 2	**Administrare conturi utilizatori**	**Administrator**
CF 2.1	Administrare conturi pentru orice tip de utilizatori	Administrator
CF 2.2	Banare utilizatori	Administrator
CF 2.3	Administrare cont financiar (bani) utilizatori	Administrator
CF 2.4	Retrimitere email de activare cont	Administrator
CF 3	Administrarea planurilor de afaceri disponibile	Administrator
CF 4	Administrarea instrucțiunilor de ajutor pentru capitolele planului de afaceri	Administrator
CF 5	Administrarea instrucțiunilor și descrierilor pentru modulul financiar	Administrator

Tabel 4.2 Cerințe funcționale ale administratorului

CF 6	Gestionarea planurilor de afaceri	Utilizator ce poate edita
CF 6.1	Administrarea unui plan de afaceri	Utilizator ce poate edita
CF 6.2	Selectarea unui plan de lucru	Utilizator ce poate edita

Tabel 4.3 Cerințe funcționale generale pe un plan de afaceri

CF 7	Gestionarea modului financiar	Utilizator ce poate edita
CF 7.1	Adaugare venituri	Utilizator ce poate edita
CF 7.1.1	Posibilitate configurare valori pe luni sau generale	Utilizator ce poate edita
CF 7.1.2	Vizualizare grafice, Instrucțiuni, Ipoteze, Creanțe	Utilizator ce poate edita
CF 7.1.3	Vizualizare fără posiblitate de editare	Utilizator care poate doar vizualiza
CF 7.2	Adăugare costuri variabile	Utilizator ce poate edita
CF 7.2.1	Vizualizare grafic, Stocuri, Achiziții, Furnizori	Utilizator ce poate edita
CF 7.2.2	Vizualizare fără posiblitate de editare	Utilizator care poate doar vizualiza
CF 7.3	Adăugare costuri fixe	Utilizator ce poate edita
CF 7.3.1	Vizualizare grafic, Stocuri, Achiziții, Furnizori	Utilizator ce poate edita
CF 7.3.2	Vizualizare fără posiblitate de editare	Utilizator care poate doar vizualiza
CF 7.4	Adăugare costuri legate de personal	Utilizator ce poate edita
CF 7.4.1	Configurare costuri în funcție de linii de venit (volumul total vândut, sau o anumită linie de venit), în funcție de vânzările totale ale firmei etc.	Utilizator ce poate edita
CF 7.4.2	Vizualizare grafic, raport persoane și raport salarii brute	Utilizator ce poate edita
CF 7.4.3	Vizualizare fără posiblitate de editare	Utilizator care poate doar vizualiza
CF 7.5	Adăugare resurse active	Utilizator ce poate edita
CF 7.5.1	Vizualizare grafic, valoare brută, amortizare lunară, amortizare cumulată	Utilizator ce poate edita
CF 7.5.2	Vizualizare fără posiblitate de editare	Utilizator care poate doar vizualiza

CF 7.6	Adăugare finanțare din capitaluri	Utilizator ce poate edita
CF 7.6.1	Vizualizare fără posiblitate de editare	Utilizator care poate doar vizualiza
CF 7.7	Adăugare finanțare din credite	Utilizator ce poate edita
CF 7.7.1	Vizualizare grafic, rambursare capital, dobândâ credit, încasări	Utilizator ce poate edita
CF 7.7.2	Vizualizare fără posiblitate de editare	Utilizator care poate doar vizualiza
CF 7.8	Adăugare finanțare linie de credit	Utilizator ce poate edita
CF 7.8.1	Vizualizare Sold credit și Dobândă credit	Utilizator ce poate edita
CF 7.9	Vizualizare raport Contul de profit și pierderi	Utilizator ce poate edita
CF 7.10	Vizualizare raport Bilanț	Utilizator ce poate edita
CF 7.11	Vizualizare raport Cash flow	Utilizator ce poate edita

Tabel 4.4 Cerințe funcționale pentru gestiunea modululi financiar

CF 8	**Gestionare modul prezentare plan afacere**	**Utilizator ce poate edita**
CF 8.1	Configurare nume firmă și logo pe care se realizează planul curent selectat	Utilizator ce poate edita
CF 8.2	Adăugare descriere firmă	Utilizator ce poate edita
CF 8.3	Descrierea problemei sau a unei liste de probleme pe care le rezolvă firma pentru clienții țintă	Utilizator ce poate edita
CF 8.4	Descrierea soluțiilor sau a unei liste de soluții pe care o pune la dispoziție firma	Utilizator ce poate edita
CF 8.5	Descrierea pieții țintă și vizualizarea raportului dintre importanța pieților	Utilizator ce poate edita
CF 8.6	Vizualizare și editare peisaj competitiv	Utilizator ce poate edita
CF 8.7	Descrierea unei idei de finanțare (în cazul în care firma are nevoie de finanțare)	Utilizator ce poate edita
CF 8.8	Descriere sau listă ce reprezintă metodele prin care se intenționează să se vândă produsele	Utilizator ce poate edita
CF 8.9	Descrierea activităților din cadrul firmei în ceea ce privește etapa de marketing	Utilizator ce poate edita
CF 8.10	Editarea unei prognoze în ceea ce privește fluxul de venituri, costurile majore și totaluri	Utilizator ce poate edita
CF 8.11	Adăugare câteva repere importante pentru firmă, o listă de activități care au prioritate	Utilizator ce poate edita
CF 8.12	Editarea echipei și a rolurilor în cadrul echipei	Utilizator ce poate edita

CF 8.13	Descrierea partenerilor și a resurselor	Utilizator ce poate edita
CF 8.14	Vizualizarea modulului de prezentare fără posiblitate de editare/adăugare	Utilizator care poate doar vizualiza

Tabel 4.5 Cerințe funcționale pentru gestiunea modulului de prezentare

CF 9	**Gestionare capitole plan afaceri**	**Utilizator ce poate edita**
CF 9.1	Configurare capitol "Sumar executiv", cu subcapitolele: a) Cine suntem b) Ce vindem c) Cui vindem d) Sumar fiananciar	Utilizator ce poate edita
CF 9.2	Descrierea capitolului "Prezentare generală" cu subcapitolele: a) Cine suntem b) Ce vindem c) Cui vindem d) Sumar fiananciar	Utilizator ce poate edita
CF 9.3	Descrierea capitolului "Analiza de piață. Segmentul de piață", cu subcapitolele: a) Descrierea pieței b) Mediul extern apropiat c) Segmentul de piață	Utilizator ce poate edita
CF 9.4	Descrierea capitolului "Produsele/Serviciile oferite"	Utilizator ce poate edita
CF 9.5	Descrierea capitolului "Strategia de marketing și vânzări"	Utilizator ce poate edita
CF 9.6	Descrierea capitolului "Organizare, conducerea personalului"	Utilizator ce poate edita
CF 9.7	Descrierea capitolului "Procesul de producție"	Utilizator ce poate edita
CF 9.8	Descrierea capitolului "Investiția"	Utilizator ce poate edita
CF 9.9	Descrierea capitlului "Plan de implementare"	Utilizator ce poate edita
CF 9.10	Vizualizarea datelor finaciare, preluate de la modulul financiar. Datele vizualizate fac parte din cheltuielile firmei.	Utilizator ce poate edita
CF 9.11	Vizualizare rapoarte financiare (Contul de profit și pierderi, Bilanțul și Cash flow-ul).	Utilizator ce poate edita

Tabel 4.6 Cerințe funcționale pentru gestionarea capitolelor planului de afaceri

CF 10	**Gestionarea activităților**	Utilizator ce poate edita
CF 10.1	Adăugare activități	Utilizator ce poate edita
CF 10.2	Setare stare activitate: Derulare, Finalizat	Utilizator ce poate edita
CF 10.3	Setare activități dependente	Utilizator ce poate edita
CF 10.4	Vizualizare grafic Gantt	Utilizator ce poate edita
CF 10.5	Vizualizare fără posiblitate de editare	Utilizator care poate doar vizualiza

Tabel 4.7 Cerințe funcționale pentru gestionarea activităților

CF 11	**Configurare profil firmă**	Utilizator ce poate edita
CF 11.1	Configurarea datelor ce țin de firmă: a) Nume companie b) Sigla companie c) Început modul financiar d) Perioada monitorizare e) Moneda preferată f) Tipul de impozit	Utilizator ce poate edita

Tabel 4.8 Cerințe funcționale pentru modulul de configurare

CF 12	**Administrarea contului**	**Utilizator ce poate edita**
CF 12.1	Alegerea tipului de abonament	Utilizator ce poate edita
CF 12.2	Trimite invitație de atașare la contul curent	Utilizator ce poate edita
CF 12.3	Vizualizare utilizatori conectați	Utilizator ce poate edita
CF 12.4	Editare cont, atașament poză, telefon	Utilizator ce poate edita
CF 12.5	Modificare parola (prin confirmare parolă veche)	Utilizator ce poate edita

Tabel 4.9 Cerințe funcționale pentru administrarea contului

CF 13	**Generare document PDF cu toate capitolele realizate în planul financiar selectat pentru lucru**	Utilizator ce poate edita
CF 13.1	Posibilitatea trimiterii raportului PDF generat prin email.	Utilizator ce poate edita

Tabel 4.10 Cerințe funcționale pentru exportul datelor

4.2.2 Cerințe non-funcționale

Cerințele non-funcționale sunt niște parametri de calitate ai proiectului, reprezintă cerințele de performanță și disponibilitate ai platformei. În acest capitol se vor descrie detaliile legate de caracteristicile de comportament ale sistemului după hostarea acestuia pe un server și aducerea acestuia în producție.

Siguranța funcționării - acest lucru se referă la faptul că produsul software va funcționa întotdeauna, fără întreruperi sau erori pe parcursul exploatării acestuia. Deoarece toate formularele sunt validate atât pe partea de client cât și pe partea de server, iar baza de date nu permite intrări neconsistente, numărul de sesiuni deschise în fiecare moment de timp nu are o limită fizică, dar se vor folosi anumite protocoale care vor minimiza numărul sesiunilor pentru un număr mare de utilizatori – toate aceste caracteristici dau siguranța că (dacă nu se fac modificări în codul sursă) platforma nu are nici un motiv să producă erori sau pauze de lucru.

Performanță - capabilitatea sistemului de a procesa un număr mare de tranzacții/ordine pe durată îndelungată pentru a putea asigura funcționalitatea și utilizarea în condiții optime a sistemului. Pentru operații care au stări fixe, în sistem sunt tratate în cozi (queue), iar operațiile dinamice sunt realizate paralel pentru fiecare utilizator. La teste fizice platforma a demonstrat ca suporta până la 2000 de intrări simultan fără a avea latențe. Performanța astfelor de sisteme este compensată de caracteristicile și optimizările sistemului de operare pe care este hostat. De exemplu pentru a optimiza platforma pe un Linux (Ubuntu) se poate utiliza un server apache pentru compilarea PHP, dar și un Nginx cu protocolul SPDY pentru a mări performanțele la maxim.

Latență minimă /timp de raspuns - sistemul oferă o performanță mare deoarece toate operațiile care erau greu de realizat, s-au minimizat la nivel de complexitate utilizând diverse pattern-uri și principii OOP. Întârzierea produsă de această platformă este minimă, asigurând astfel accesul cât mai rapid la vizualizarea proiectelor, rapoartelor, graficelor și datelor generale (capitole etc.).

Reziliența la căderi - acest lucru presupune capacitatea de a funcționa întotdeauna având cel puțin două servere astfel încât în cazul în care unul este oprit, sau timpul de răspuns de la el este prea mare (decât o valoare stabilită), celălalt va prelua responsabilitatea de a procesa request-urile. Acest lucru se face în mod transparent pentru utilizatorul final. Baza de date în acest caz este unică, sesiunea este de asemenea unică și salvată în memcache. Sincronizarea serverelor trebuie realizată la nivelul fișierelor statice salvate pe HDD, în acest caz se utilizeaza anumite soluții (cum ar fi linux Unison).

4.3 Cazuri de utilizare

Cazurile de utilizare au scopul de a oferi o viziune globală asupra funcționalităților, posibilităților, nivelulurilor de utilizatori, dar și permite o vizualizare a relațiilor dintre tipurile de utilizatori din cadrul sistemului. Cazurile de utilizare se vor realiza desigur în baza cerințelor funcționale ale sistemului descrise în subcapitolul 2.2.1. Operațiile de gestiune generale (create, read, udate, delete) dar și operațiile de consistență și validare nu necesită reprezentare în cadrul cazurilor de utilizare descrise mai jos.

Evident cazurile de utilizare trebuie definite pe niveluri de utilizatori. De aceea se vor reprezenta și actorii sistemului.

4.3.1 Actorii sistemului

Mai jos sunt reprezentate principalele roluri și permisiuni pentru utilizatorii care pot folosi sistemul:

- Administrator
- Utilizator care poate edita
- Utilizator care poate citi și comenta
- Utilizator care poate doar vizualiza

4.3.2 Cazurile de utilizare

În figura 4.6 este reprezentat actorul care administrează tot sistemul, el este actorul suprem în ceea ce priveste managementul utilizatorilor și al rolurilor acestora în cadrul sistemului.

Platforma ar putea fi privită ca având mai multe tipuri de administratori, pe lângă administratorul întregului sistem - care se ocupă de datele generale legate de toți utilizatorii sistemului, nu mai puțin important ar fi și administratorul în cadrul unei organizații, acesta trebuie să aibă posibilitatea de a edita toate datele din cadrul organizației (firmei) pe care o deține. Acest nivel de administrare este privit ca o dezvoltare ulterioară în sistem.

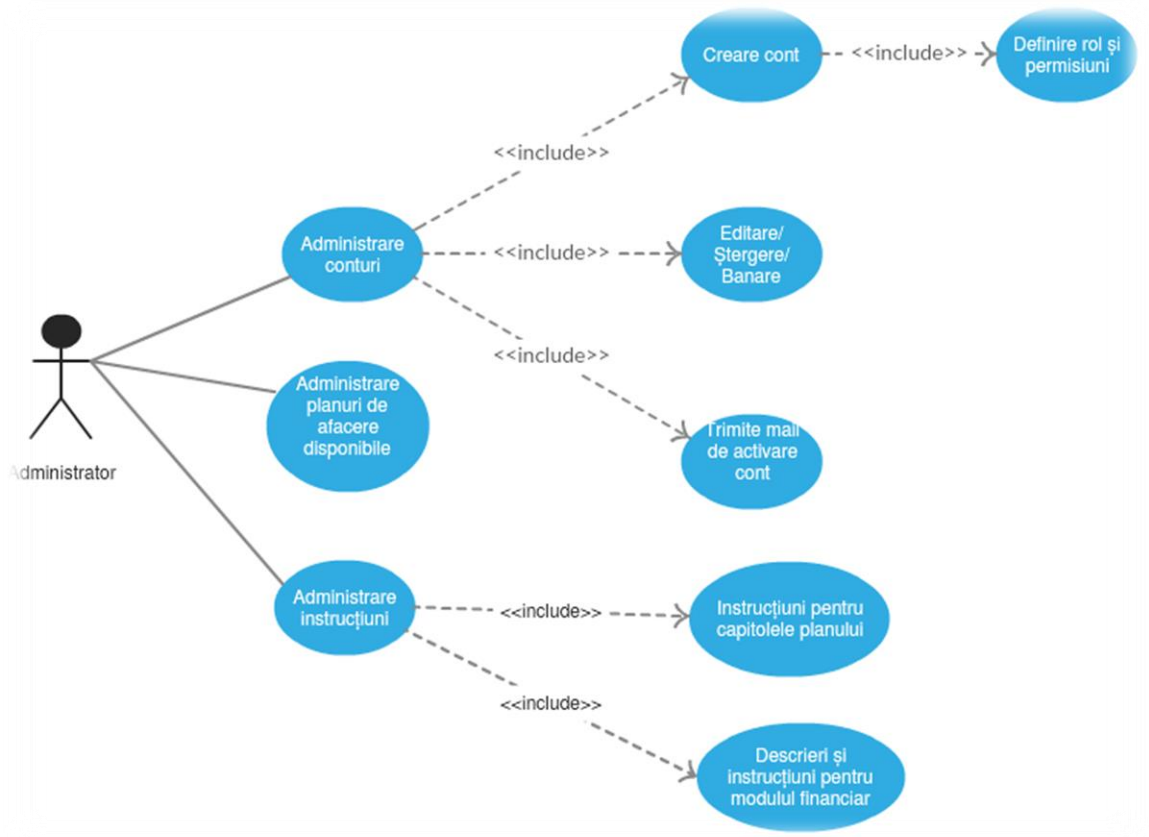

Fig. 4.6 Cazuri de utilizare pentru Administrator

În diagrama din Figura 4.7, sunt prezentate cazurile de utilizare ale actorului care are posibilitatea deplină asupra planurilor de afacere ce le crează. Acest tip de rol a fost denumit "Edit" sau "Utilizator care poate edita".

Utilizatorul care poate edita planurile de afaceri, este privit ca un utilizator cu drepturi absolute pe organizația de care aparține, acesta poate să realizeze procese de orice natură, începând cu editarea configurărilor generale și terminând cu acordare de invitații și generare raport PDF.

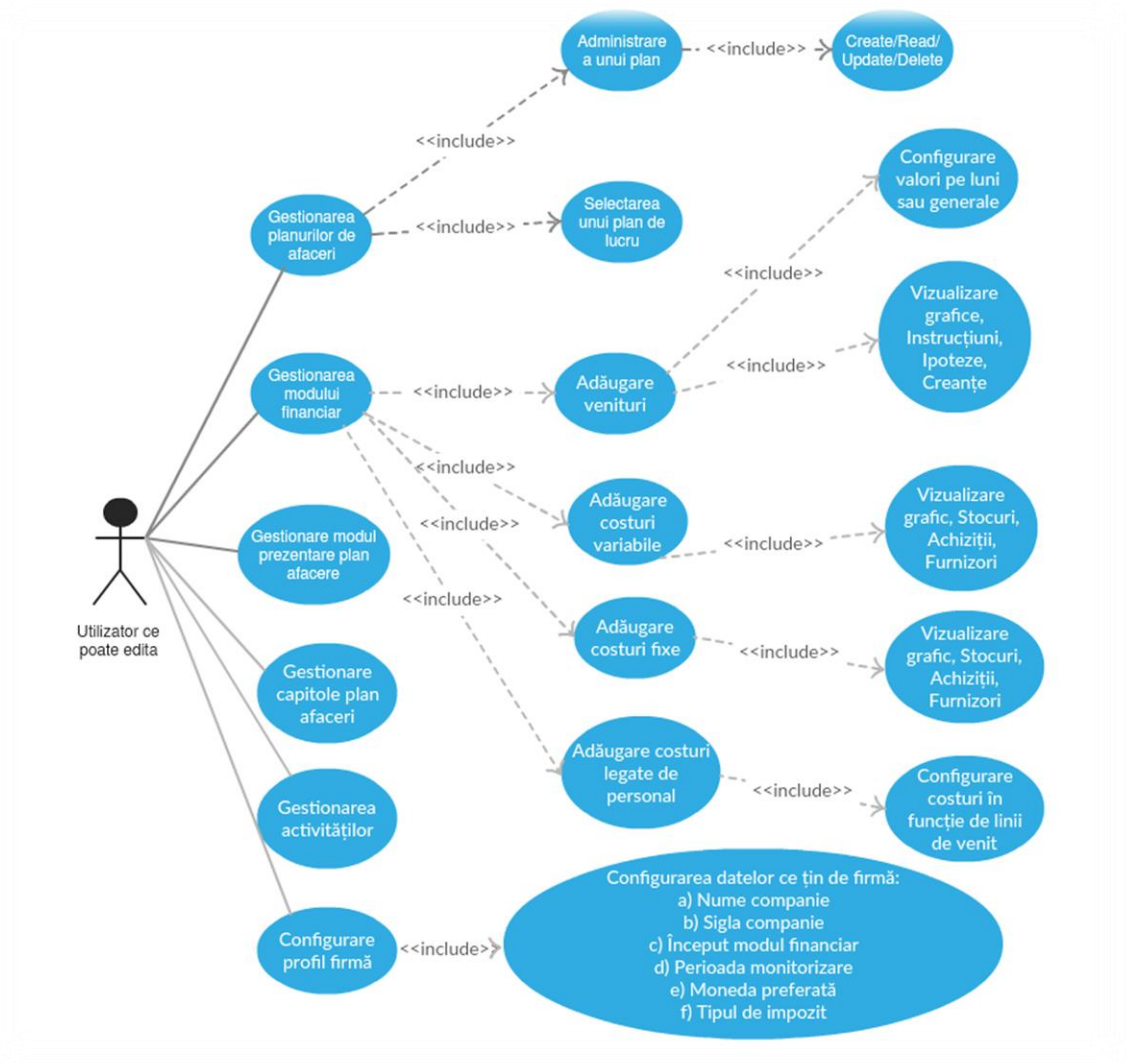

Fig. 4.7 Cazuri de utilizare pentru un Utilizator ce poate edita planurile

În figura de mai jos (Figura 4.8) este reprezentată diagrama pentru cazul de utilizare a utilizatorilor ce au doar posibilitatea de vizualizare și comentare. Acești utilizatori sunt înregistrați pe platformă în urma invitației de către alt utilizator care are dreptul de editare și creare planuri, dar și permisiunea de a realiza invitații.

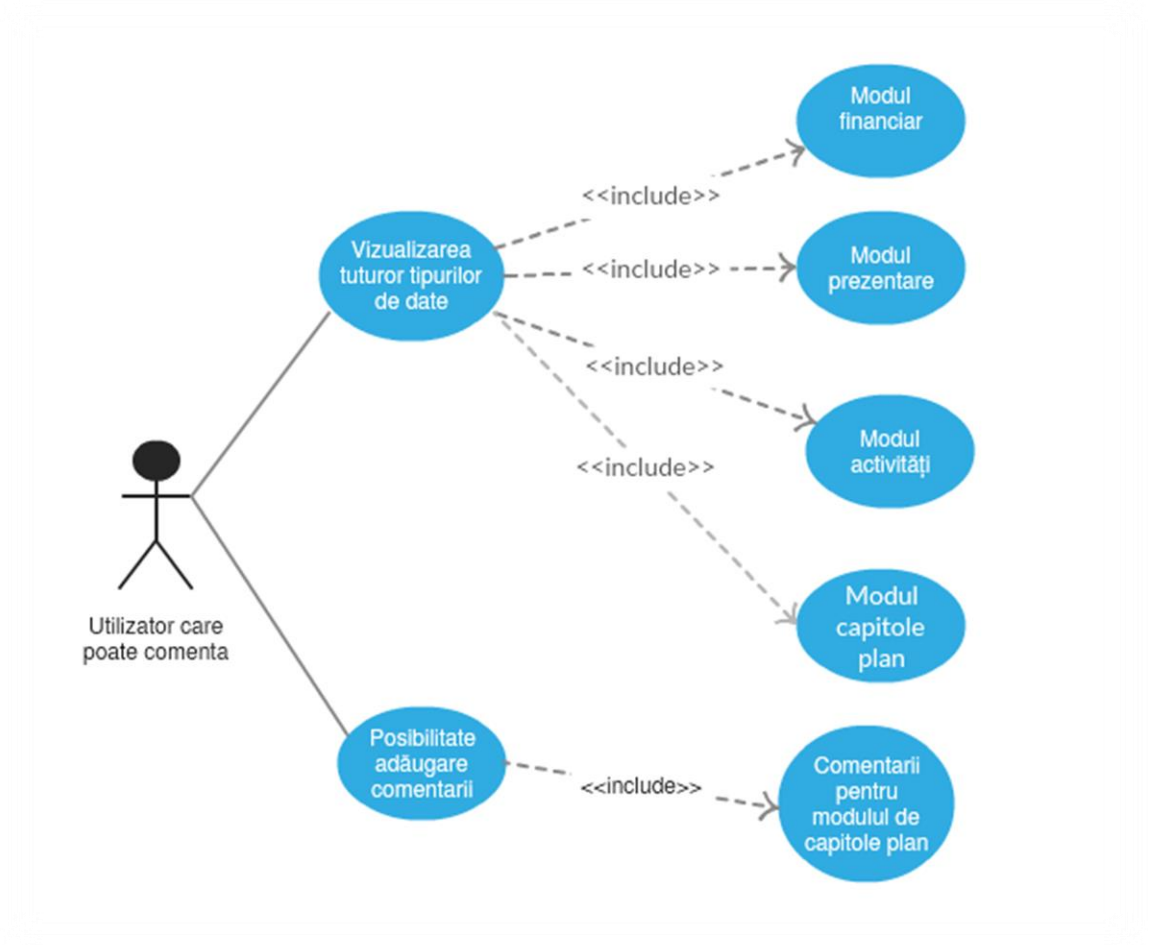

Fig. 4.8 Cazurile de utilizare pentru utilizatorii care pot comenta

Utilizatorii care au cont provenit din invitație de tip "Read" (doar citire), pot să vizualizeze toate modulele, dar nu pot nici să editeze, nici să comenteze. Figura 4.9 reprezintă acest tip de utilizatori.

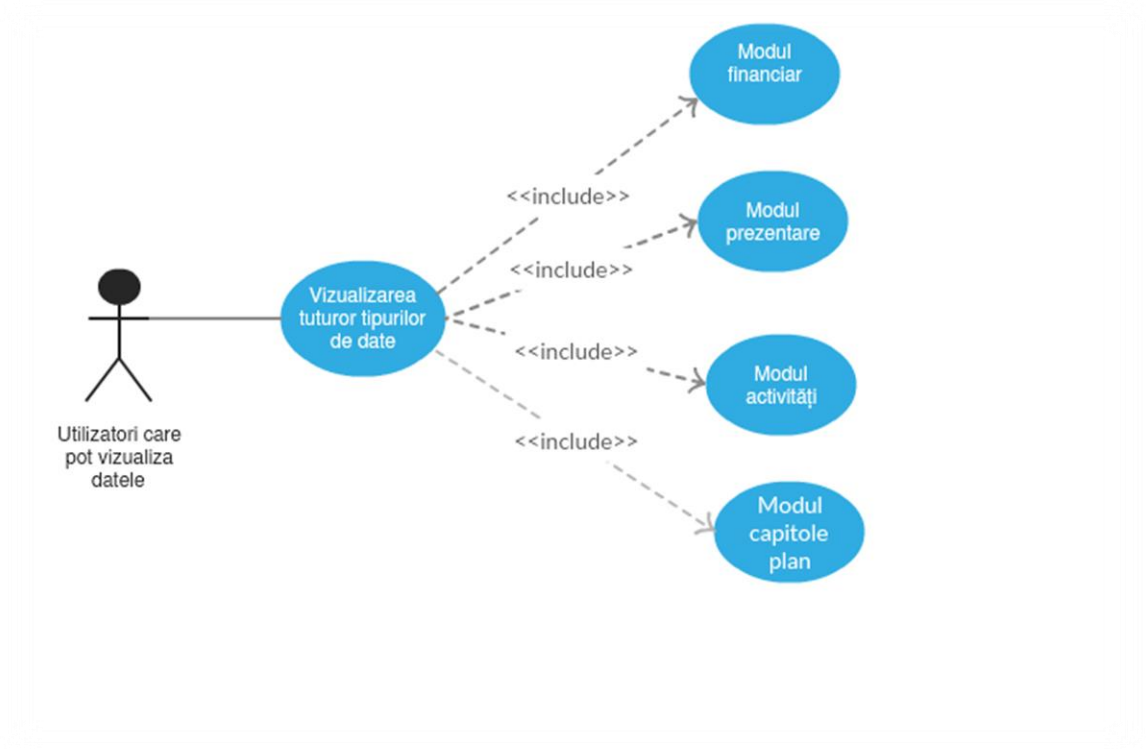

Fig. 4.9 Reprezentarea cazurilor de utilizare pentru Utilizatorii care pot doar să vizualizeze datele din sistem

4.3.2.1 Descriere detaliată a planurilor de utilizare

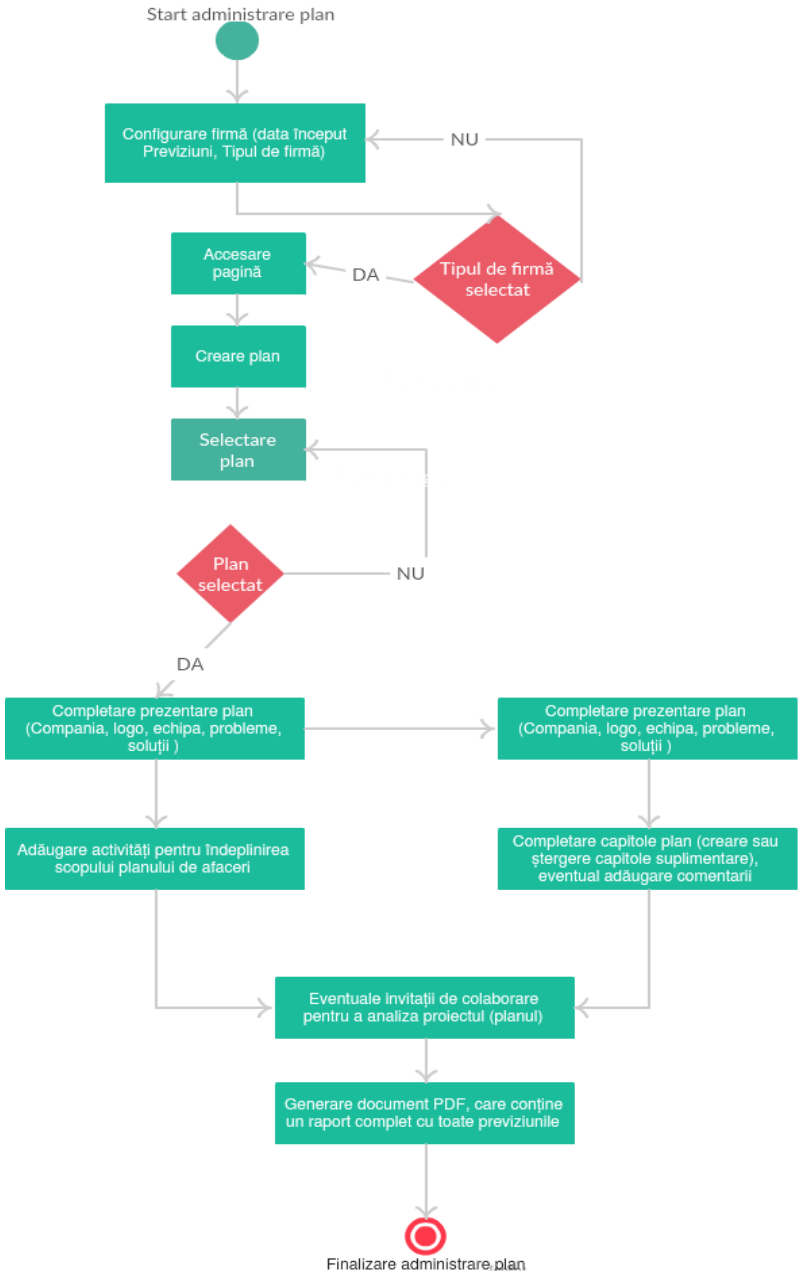

Fig. 4.10 Fluxul de lucru pentru administrarea unui plan de afaceri

În organigrama de mai sus (Fig. 4.10) se poate observa este ordinea pașilor pentru a crea

și administra un plan de afaceri.

CU1

Numele cazului de utilizare: Administrarea unui plan de afaceri

Actor principal: Utilizator cu drept de editare

Precondiții: Utilizatorul trebuie sa fie autentificat și trebuie sa posede rolul de editare planuri.

Postcondiții: Planul, împreună cu toate legăturile sale pe aplicație, inclusiv modulul financiar - sunt salvate în baza de date. Posibilitatea generării (sau trimiterii pe un mail) a documentului PDF generat după ce s-au completat toate câmpurile din module.

Scenariu de succes:

1. Pentru a realiza modulul financiar al unui plan de afaceri, este nevoie să fie configurat tipul de firmă pe care se realizează planul, data de început a previziunilor financiare, tipul monedei etc.
2. Accesarea paginii de creare plan de afaceri.
3. Crearea unui plan de afaceri (pentru a-l crea este suficient să completăm câmpul cu numele planului).
4. Pentru a completa toate câmpurile, toate modulele necesare unui plan de afaceri, acesta trebuie selectat ca fiind planul de lucru curent.
5. După selectarea planului de afaceri, trebuie completate într-o ordine aleatoare modulele: Prezentare, Capitolele planului, Previziunile financiare, Activități.
6. Se pot realiza consultații pe baza planului, acestea se pot realiza cu ajutorul

CU2

Numele cazului de utilizare: Comentarea unui plan de afaceri

Actor principal: Utilizator cu drept de comentare și vizualizare

Precondiții: Utilizatorul trebuie să aibă un mail valid

Postcondiții: Planul de afaceri pe care s-a realizat invitația este comentat și printat, astfel s-a trimis feedback către proprietarul planului

Scenariu de succes:

1. Proprietarul contului trebuie să se asigure că are posibilitatea de a trimite invitații de colaborare de tip "Comentare"
2. Proprietarul planului de afaceri accesează pagina Cont/Conturi conectate, și jos, în câmpul ”Adresa de email” introduce adresa de email a utilizatorului pe care dorește să-l invite pentru a colabora. În câmpul ”Selectează rolul” alege *Comment,* acest lucru îi va permite utilizatorului care se va înregistra să aibă drepturi de comentare pe acest plan.
3. Utilizatorul invitat își face cont prin accesarea link-ului din mailul primit (acesta va fi direct atașat la planul de lucru pe care a fost invitat).

4. Navigând prin proiect, acesta poate comenta la modulul de capitole, comentariile pot fi realizate pe orice tip de capitol sau subcapitol realizat de către proprietar

Capitolul 5. Proiectare de detaliu și implementare

În acest capitol vor fi detaliate elementele de proiectare și arhitectură ale proiectului. Vor fi prezentate exemple de cod, se vor analiza tipurile de vulnerabilități care se pot întâlni, se vor vizualiza pachetele de lucru pe module funcționale. Se vor prezenta diagramalele arhitecturii alese pentru implementarea aplicației, se va reprezenta diagrama arhitecturii baze de date și a relațiilor dintre entități, diagrama de deployment și modul de întreținere a actualizărilor.

5.1 Arhitectura sistemului

Sistemul prezentat în această lucrare se încadrează în categoria aplicațiilor pe trei nivele. Nivelul de interacțiune cu utilizatorul final, în care se prind acțiunile acestuia și sunt prelucrate de către JavaScript. Nivelul doi, sau middle-tier - este cel mai mare ca volum de logică, aici sunt procesate toate validările, pregătirea datelor pentru preprocesare și salvare în baza de date, aici se realizează modulele de calcul al rapoartelor pentru capitolul financiar. Și desigur ultimul nivel, cel de stocare, el comunică prin protocoalele standard cu nivelul de mijloc.

Modelul utilizat se bazează, la nivelul comunicării, pe arhitectura client-server, în care interfața cu end-userul, business logic, baza de date pot fi stocate pe servere diferite.

Ca design pattern pentru primul nivel a fost utilizat MVC. Iar partea de V-view, a fost prelucrată cu AngularJs.

Cele trei nivele care au fost numite mai sus, reprezintă: Nivelul de prezentare, Nivelul de logică și Nivelul de stocare (și accesare) a datelor.

Nivelul de prezentare reprezintă niveul superior al aplicației, cel de comunicare cu utilizatorul final. Acest nivel este de obicei întreținut de JavaScript, gridul aplicației și toate componentele de design. Această componentă, fiind unica ineracțiune fizică cu clientul, se încarcă în browserul clientului, rulează local, iar comunicarea cu logica platformei se realizeaza prin servicii întreținute de protocolul HTTP.

Servirea paginilor către client se poate realiza prin două moduri (ținând cont de această arhitectură), primul mod este cel în care serverul generează paginile, și le serververște clientului pentru vizualizare, iar la schimbarea stării părții de view acesta este regenerat de către server și servit din nou. Modul ăsta de abordare este destul de rapid pentru client, dar solicită prea mult serverul, și la un număr mare de intrări în fiecare moment de timp, aceasta ar aduce o latență destul de mare la nivelul serverului. Se pot utiliza desigur balancere de loading care să posede configurații speciale, dar această abordare nu este una optimă.

Al doilea mod de servire al paginilor, este abordarea client rendering, aceasta presupune că toate actualizările de stare ale view-ului vor fi rerandate doar de către clientul utilizatorului (browserului). Actualizările desigur au nevoie de date noi, pe care să le solicite de la server prin request-uri ajax, si acest lucru este foarte usor realizabil, daca nu mai folosim plain text pentru comunicare, ci se va folosi comunicarea json. Astfel, la fiecare actualizare de stare, nucleul de Angular va stabili un observer pe modelul prezentat, și va reranda conținutul parțial fără a aduce latențe pe partea de server, dar nici pe partea de client. Avantajul major al unei astfel de abordări

este că observerul va actualiza doar părți mici de pagină, iar utilizatorul nici nu va fi capabil să realizeze tranzițiile.

Nivelul logic acoperă toată robustețea aplicației, el este stocat pe un server și este compilat real-time de către un apache. Desigur că se utilizează câteva tool-uri de optimizare a conținutului compilat, unul dintre ele este *composer* folosit pentru a compila dependențele logicii, și un tool intern, *artisan,* care este utilizat pentru a face un singur fisier de logica, care menține referințe la toate pachetele și clasele proiectului. Optimizarea se mai realizează și la stocarea referințelor des solicitate în memoria RAM sub formă de cache (memcache).

Nivelul de stocare a datelor, accesul și stocarea datelor este un proces complex din punct de vedere al securității și vitezei de acces. Baza de date poate fi stocată pe un server diferit, sau pe un server principal, la care să oferim un acces partajat tuturor instanțelor de lucru în cazul în care se utilizează balancere. Nivelul acesta comunică cu nivelul de mijloc doar (Nivelul logic), comunicarea se realizează prin drivere speciale și o conexiune mereu deschisă pe portul specific configurării acestui driver.

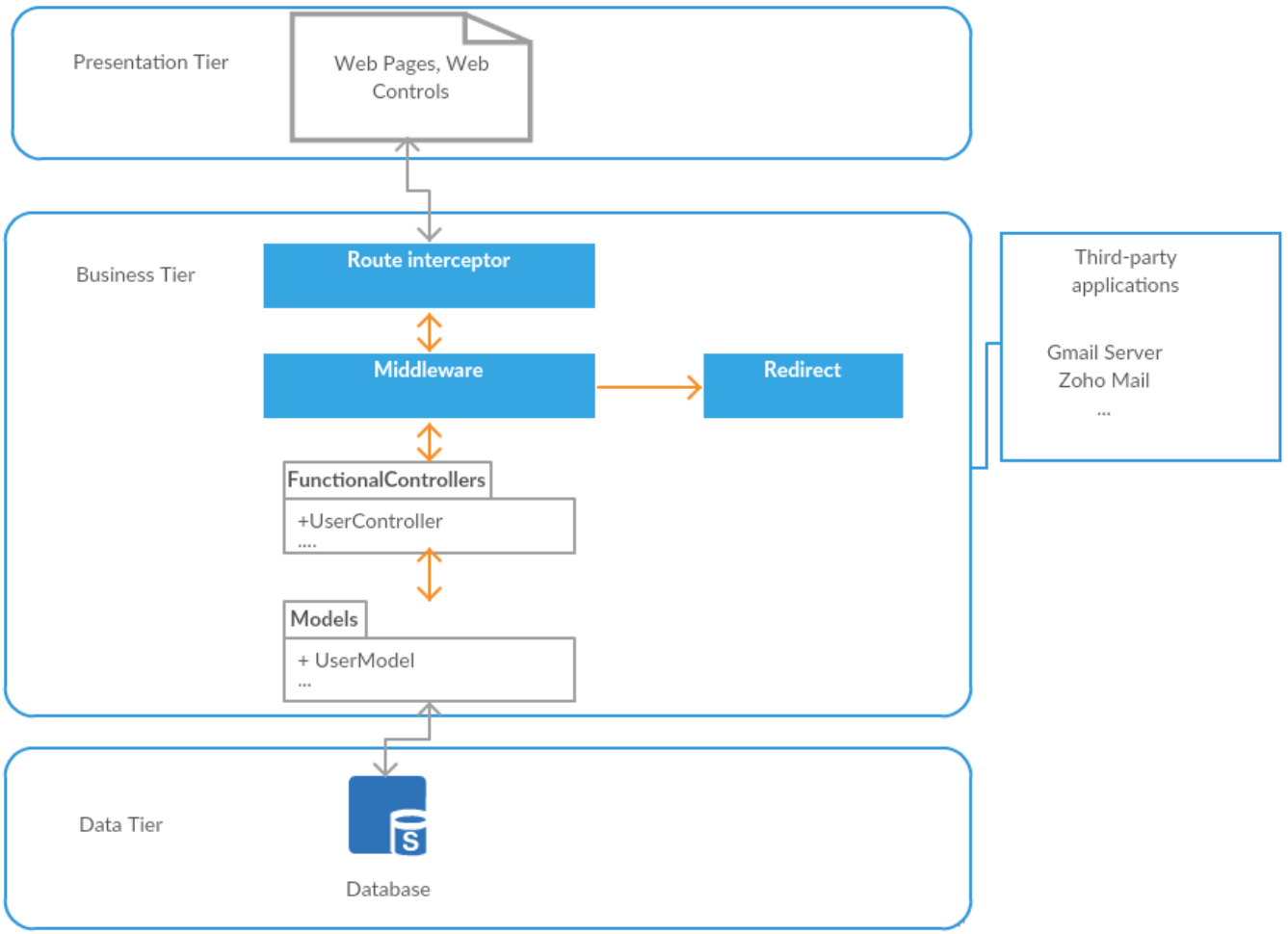

Fig. 5.1 Diagrama arhitecturii aplicației

În figura 5.1 de mai sus, se poate observa reprezentarea comunicării dintre nivelele sistemului. La comunicarea nativă se mai adaugă o relație cu aplicații, servicii API externe pe care le poate folosi sistemul pentru a întreține toate funcționalitățile necesare.

5.1.1 Presentation Tier

În acest subcapitol se va prezenta în detaliu nivelul de prezentare al aplicației, modul de interacțiune cu utilizatorul final. Se vor parcurge posibilitățile de procesare a evenimentelor, ascultarea evenimentelor pe partea de client. Dar se vor vizualiza și paginile pe roluri de utilizatori care au acces să le vizulizeze.

Figura de mai jos reprezintă paginile comune pentru toate tipurile de utilizatori.

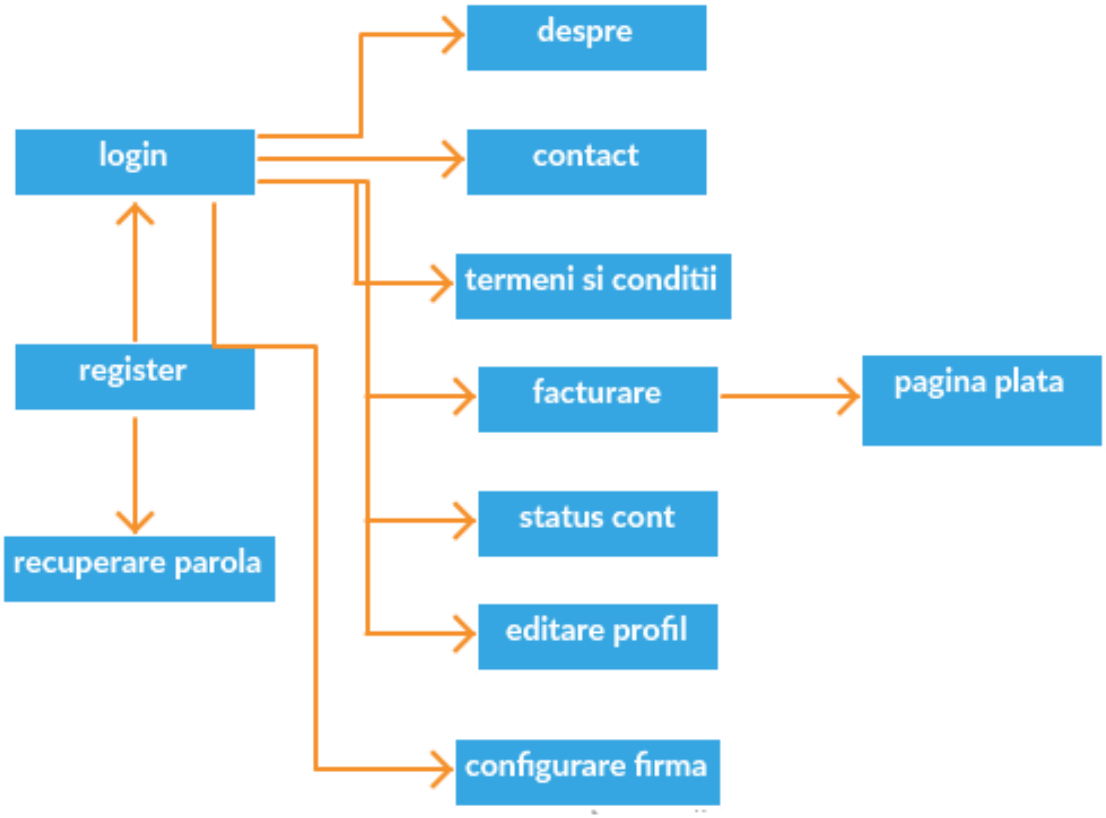

Fig. 5.2 Paginile web comune pentru toți utilizatorii

După cum se poate observa, pentru a intra în sistem este nevoie de un mod de autentificare. În urma înregistrării și confirmării căsuței de e-mail, contul este validat ca fiind activ. În mod natural sistemul dispune și de o procedură de resetare a parolei pe baza informațiilor deja cunoscute de utilizator - email și/sau nume utilizator. Toate acțiunile clientului sunt prelucrate și se menține o comunicație deschisă între sistem și utilizator, de exemplu, pentru afișarea erorilor sau a informațiilor de succes - se folosesc mesaje de tip flash.

Paginile la care un anumit rol de utilizator nu are dreptul să le vadă sunt securizate, iar servirea lor spre client nu se face sub nici o formă, acest lucru protejează sistemul de atacuri de

tip client. Sesiunea, care rămâne activă atât cât browserul (clientul) este activ, păstrează informații despre tipul de utilizator. Aceste date se trimit la fiecare request prin header. Iar validarea paginilor se face în partea de server, filtrându-se de către serviciile de middleware și verificări suplimentare, specifice, implementate de către dezvoltator ca o măsură suplimentară.

Structura paginilor este una elastică, dat fiind faptul că template engine-ul utilizat (Blade) permite extinderi, structurile care rămân statice pe tot cursul activității fac parte dintr-un sablon, acesta este extins cu structuri suplimentare în funcție de pagina accesată. De fragmentele suplimentare se ocupă un sistem special de frontend (Angular Router).

În continuare se va ~~încerca~~ prezentarea paginilor disponibile pentru anumite nivele de utilizatori.

- **Administrator** - mai jos se pot vizualiza posibilitățile de navigare ale administratorului

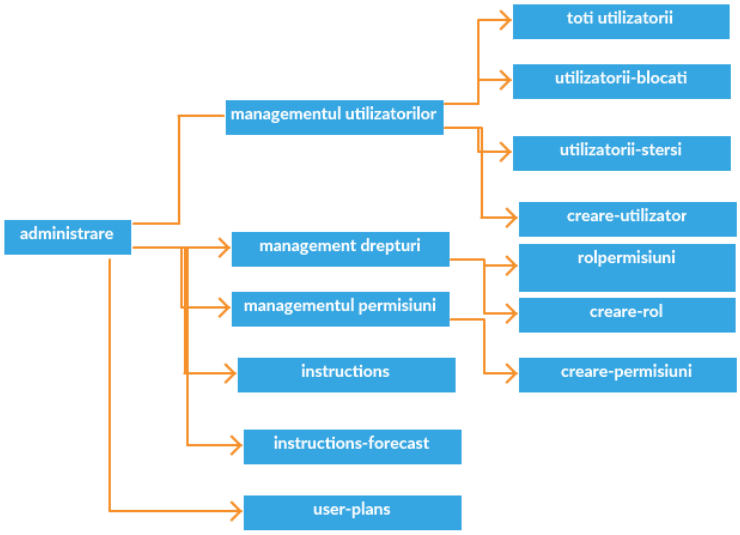

Fig. 5.3 Paginile disponibile - administrator

- **Utilizator care poate edita**

Imaginea de mai jos cuprinde paginile disponibile pentru cel mai mare spectru de posibilitități - ale utilizatorului cu un cont valid și posibilitate de editare planuri de afaceri.

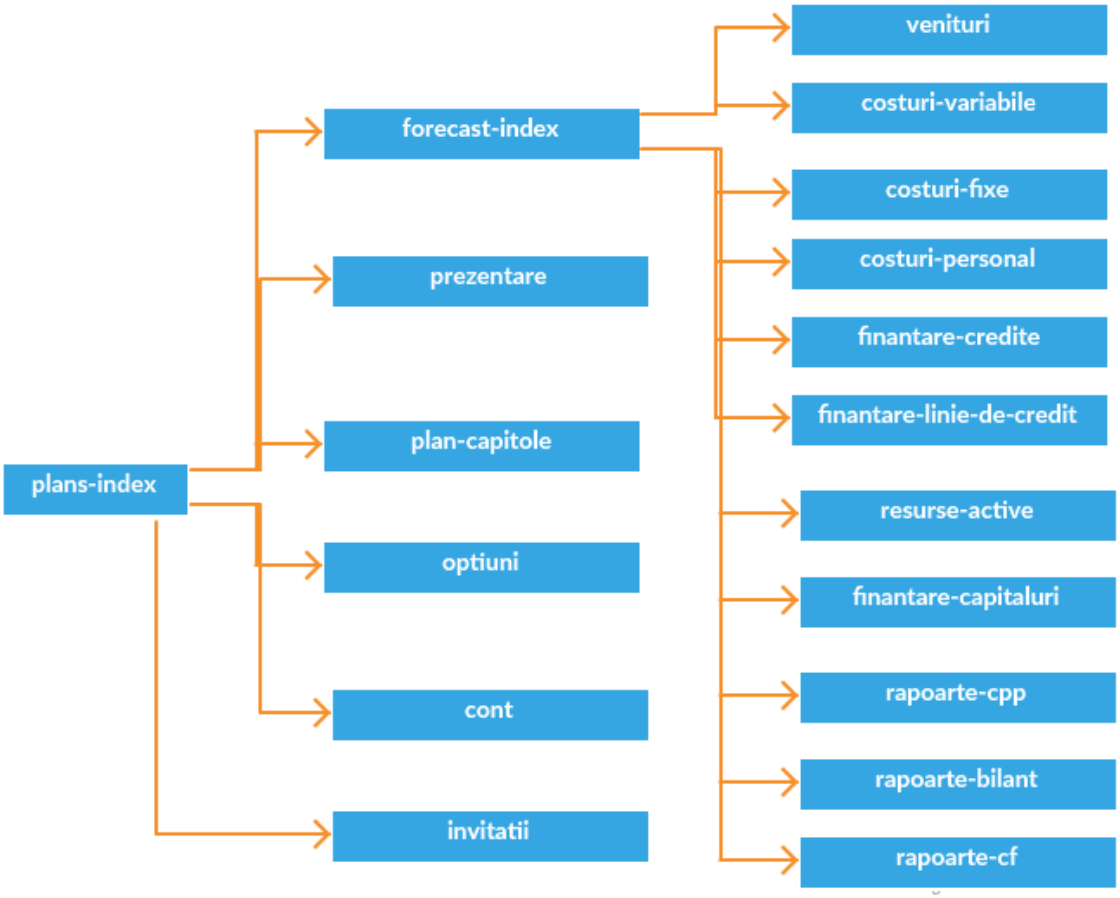

Fig. 5.4 Paginile disponibile - utilizator care poate edita

5.1.2 Business Tier

Nivelul de business logic al sistemului este unul complex, incluzând mai multe pachete de lucru. Fiecare pachet reprezintă un modul din aplicație, și răspunde de serviciile unei entități stabilite. Pe lângă definirea clasică, mai dispune și de câteva componente globale de ajutor (helpers), aceste unelte includ cuvinte cheie personalizate (funcții) pentru anumite operații (cum ar fi injectarea dependințelor css sau javascript).

În următoarea figură se va prezenta la nivel de pachete partea de business logic. Pachetele includ clase care tratează request-urile din spatele paginilor blade.

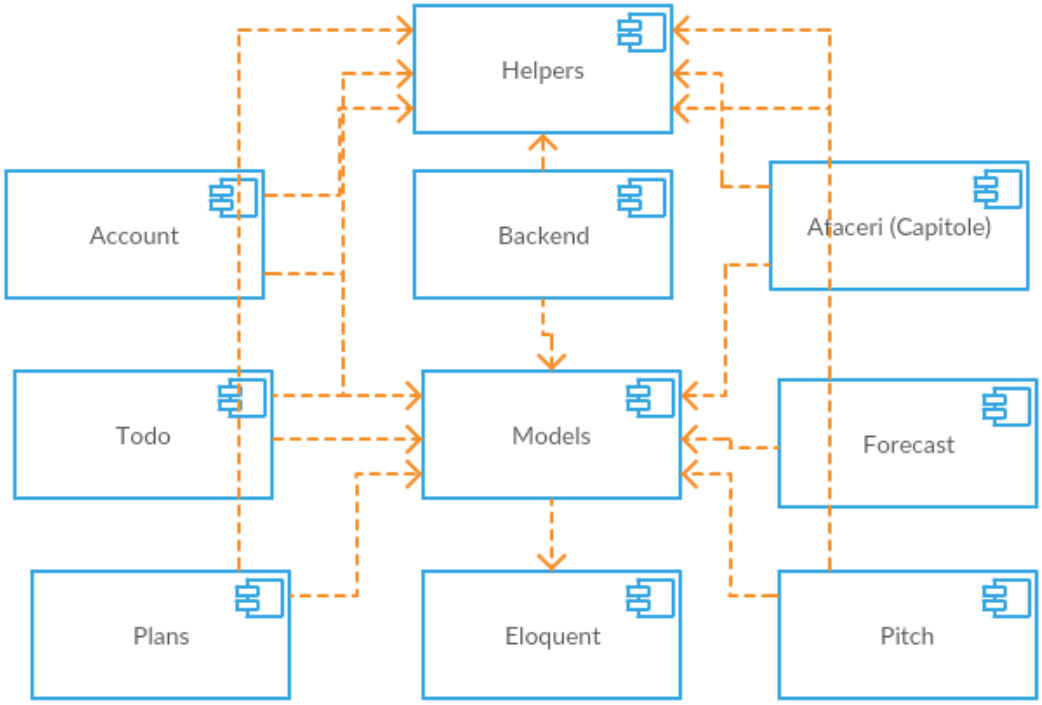

Fig. 5.5 Diagrama de pachete din business logic

5.1.2.1 Business logic - componente

1. Account

Acest pachet include toate funcționalitățile necesare gestionării contului pe partea de client. Prin gestionarea contului se presupun operațiile de creare cont, editare, resetare parola, activare cont, vizualizare modul plată și setări generale (poza de exemplu).

Modulul este primordial, organizat sub forma de componente - repositoriu pentru operațiile CRUD, Model pentru interacțiunea cu baza de date, un algoritm de criptare a parolei - mcrypt.

Componentele vor fi prezentate în imaginea de mai jos.

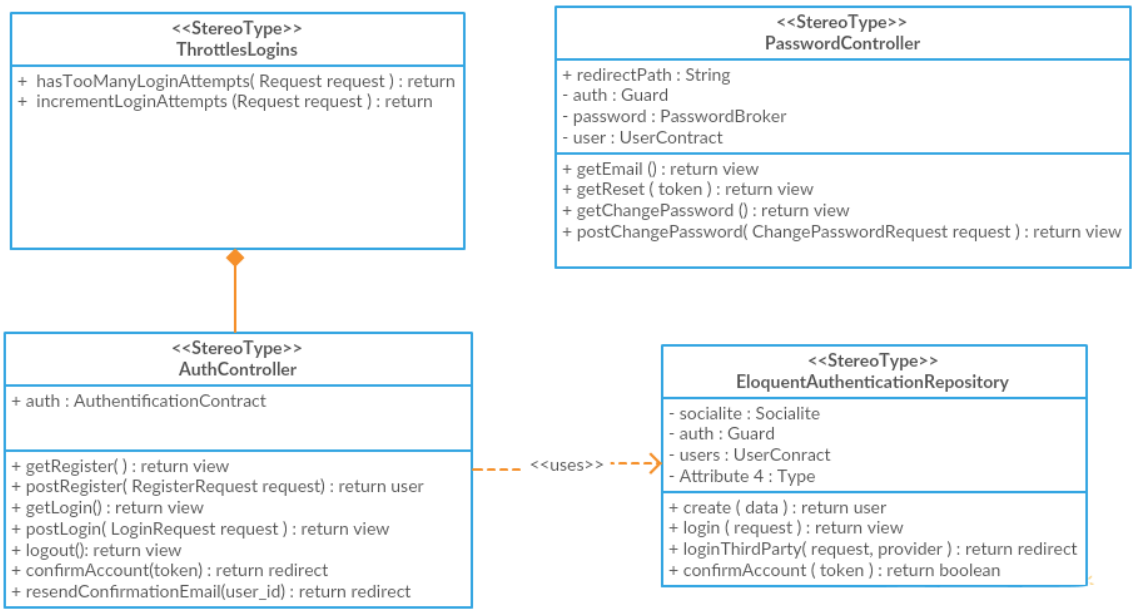

Fig. 5.6 Diagrama de clase - Account

2. Forecast

Sub numele de Forecast se găsește pachetul care include cea mai consistentă logică din sistem. Acestea realizează serviciile consumate de pagina cu previziuni financiare. Dat fiind faptul că în acest modul se întâlnesc multe entități care necesită calcule, s-a făcut o analiză pe marginea tuturor tipurilor de controale utilizate, și s-a ajuns la concluzia ca multe dintre entitățile calculate au o structură similară și respectiv pot fi implementate în mod similar.

În această ordine de idei, pentru fiecare modul financiar (de exemplu - Costuri Variabile), s-au realizat câte trei clase specifice. Clasele sunt repartizate pe funcționalități:

A - clasă de bază pentru a răspunde la toate interogările pe entitatea respectivă, ca operații primordiale sunt create/read/update/delete pe entitatea sa

B - clasă specială pentru a genera (în contextul MVC) controalele necesare, această clasă parcurge lista de controale necesare, și le randează cu ajutorul fragmentelor de view spre care trimite anumite date tipizate, exemple de controale pot fi: input text, datepicker, selectbox etc.

C - clasă specifică cu destinația de a face calculele matematice pentru a le servi graficelor, tabelelor, rapoartelor. În această clasă de obicei se fac toate calculele reale, conform normelor din România (legate de TVA de exemplu), în relația de dependență între cele trei clase, aceasta se situează de obicei în vârful dependențelor.

În figura următoare se prezintă o astfel de relație pe modelul Costurilor variabile.

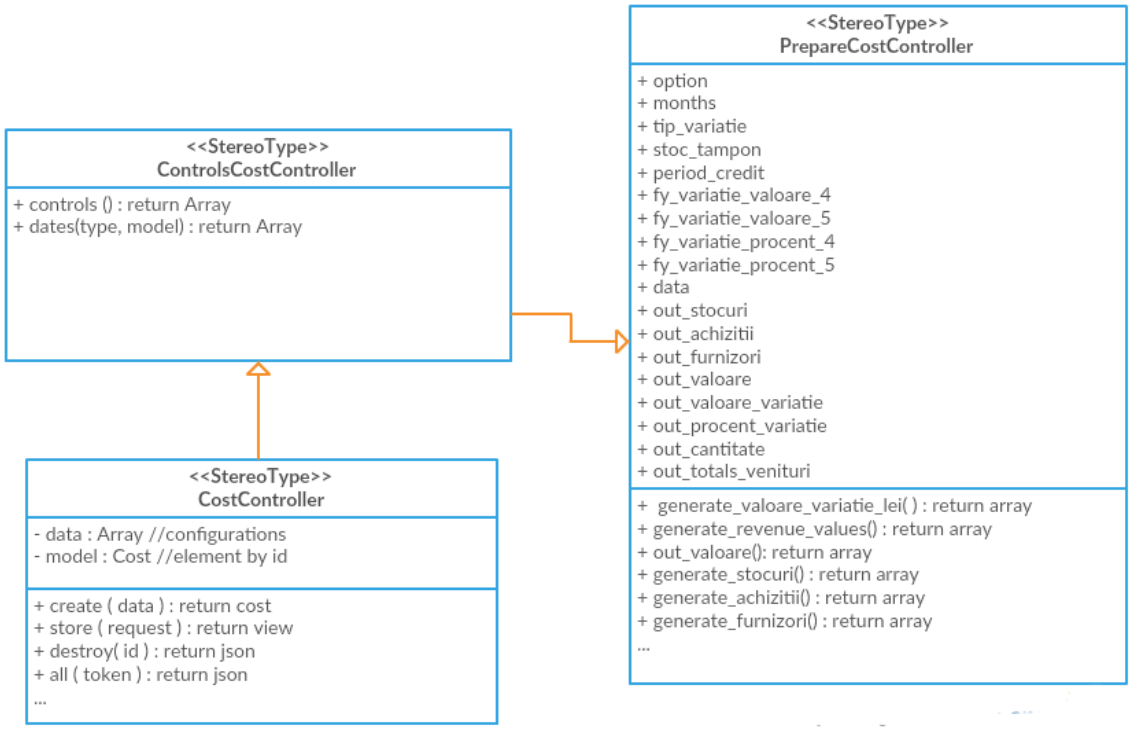

Fig. 5.7 Diagrama de clase - Costuri variabile

Cum se menționa mai sus, pentru aproape toate modulele de la partea de previziuni, s-a încercat o tipizare atât la nivelul structurii business logic, cât și la nivelul controalelor utilizate și a componentelor de interfață, cum ar fi modalul de adăugare/editare/ștergere. Pentru a obține o tipizare reușită, este nevoie de a utiliza structuri de inițializare și configurare. Mai jos se găsește codul pentru configurarea interfeței, inclusiv a controalelor, care se realizează în *CostController*, chiar în constructorul acestuia:

```
$this->data = [
    'title'           => 'Spune-ne despre costuri variabile: ',
    'submit_function'   => 'store()',
    'content'          => 'forecast.partials.cost.content',
    'save_function'     => 'store()',
    'remove_function'   => 'remove()',
    'prev_function'     => 'previous()',
    'next_function'     => 'next(form)',
    'visible'          => 'wizard',
    'controls'          => $this->controls($this->model),
];
```

Deoarece AngularJs prin request-urile pe care le realizează, așteaptă un răspuns sub formă de JSON, pe partea de business logic, se utilizează o clasă specială *Response,* care transformă orice tip de date (de exemplu array), în format JSON, pe care îl poate trimite spre procesare către

client:

```
Response::json(['elements' => $out_personnels, 'summ_total' => _summ_in_array($out_personnels, 'valoare')]);
```

Tipizarea de care se bucură partea de business logic este implementată și pe structura Angular. Continuând pe ideea costurilor variabile, mai jos se prezintă codul de instanțiere a controller-ului Angular:

```
app.controller(
        'CostCtrl', ['$scope', '$http', '$rootScope', '$compile', '$timeout','cost_service','FormService','$http',
   function CostCtrl($scope, $http, $rootScope, $compile, $timeout, cost_service,FormService) {
```

Se poate observa că se injectează câteva servicii native ($http, $rootScope etc.), și câteva personalizate, acestea sunt *cost_service* și *FormService.* Configurația tipizată este la nivelul serviciilor, dacă privim în codul sursă al serviciilor, se poate observa o structură similară, și anume:

```
app.factory('cost_service', ['$rootScope','$http','$timeout', function($rootScope, $http, $timeout){
  var mixin      = {};
   mixin.expenses = [];

   mixin.create = function(){
      var promise = $http.get($rootScope.config.r_get_modal).then(function(response){
         return response.data;
      });
      return promise;
   }

   mixin.store = function(data){
      var promise = $http.post($rootScope.config.r_post_save, {data: data}).then(function(response){
         return response.data;
      });
      return promise;
   }
.....
```

Se constatează cu ușurință procesele standard de operare pentru operațiile de CRUD și rutele configurabile, transmise prin intermediul *$rootScope*.

FormService - este o clasă (un serviciu) utilizat pentru a popula formularele sau a prelua datele din ele. Este configurabil pentru orice tip de control din formulare (input, datepicker, selectbox etc.), această clasă este foarte utilă, mai ales în cazul în care utilizăm servicii și avem nevoie de serializare manuală în obiecte json.

5.2 Deployment

Diagrama de deployment prezintă componentele fizice pe care rulează sistemul software

implementat, precum și modul în care aceste componente fizice comunică între ele pentru a aduce o performanță maximă. Scopul unei astfel de diagrame este de a prezenta structura hardware necesară pentru a rula sistemul dat.

Serverul Ubuntu 14.04, deține un compilator PHP (apache), care are în față un serviciu de *nginx*, acest serviciu accelerează viteza de răspuns la interogările clienților săi.

Aplicația web este proiectată stabil și responsive, deci poate fi accesată cu același succes și de pe un dispozitiv cu o rezoluție mică, acest lucru nu va afecta în mod radical partea de interfață.

Chiar dacă este o aplicație cu o arhitectură three tier, ea nu pune la dispoziție un API clasic, serviciile ce le pune la dispoziție, sunt întreținute doar de componentele interne pe care a fost programată aplicația, pentru a mai adăuga un consumator (de exemplu o aplicație mobilă) va fi nevoie de o refactorizare.

Mai jos se găsește diagrama de deployment:

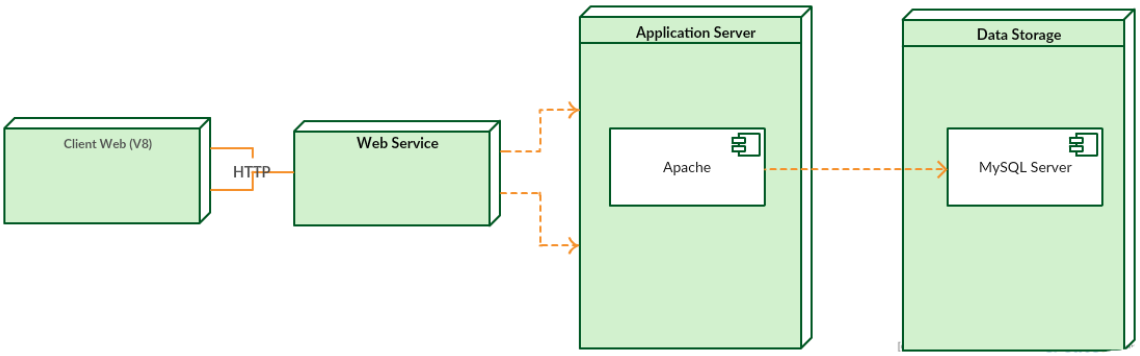

Fig. 5.8 Diagrama de deployment a sistemului

În partea stângă se găsește clientul - care este browserul web al utilizatorului, aici se vor asculta toate interacțiunile cu utilizatorul. Serverul - Application Server - genereaza Web services, care comunica prin JSON (prin protocolul HTTP). Ultimul nivel, serverul de baze de date, s-a utilizat MySQL, acesta comunică cu serverul de aplicație (situat pe același server) prin drivere speciale (mysqld).

5.3 Securizare și middleware

Un aspect nu mai puțin important în proiectarea unui sistem îl reprezintă, nemijlocit, asigurarea securității datelor. Primul aspect care trebuie tratat, este bineânțeles, autentificarea pe platformă. De multe ori se practică salvarea parolei utilizatorului în plain text în baza de date, această practică e considerată vulnerabilitate, chiar dacă niciodată (se presupune) nu se vor oferi detalii legate de câmpul respectiv. Pentru a evita acest lucru, în sistemul dat, s-a utilizat un algoritm de criptare unidirecțional (mcrypt), acest lucru înseamnă că parola odată criptată și salvată în baza de date, nu va mai putea fi decriptată niciodată, nici chiar de admninistrator.

Desigur configurările permit setarea lungimii numărului de caractere în care se criptează parola, care determină în mod direct complexitatea acesteia.

Următorul tip de vulnerabilități, așa zis-ul XSS Cross Site Scripting, acest lucru presupune că, în mod intenționat, anumite formulare din sunt stocate cu un cod javascript (sau php), și odată afișate pe pagină, browserul interpretează codul dat ca atare, rulându-l pe calculatorul victimei. Împotriva acestui tip de atac, în Laravel se utilizează template engine-ul Blade, care folosește {{}} pentru ca browserul să facă escape automat la orice entități HTML.

Dintre cele mai periculoase tipuri de atacuri sunt considerate injectările de tip SQL Injection, sau Code Injection (PHP). Dacă sunt omise astfel de vulnerabilități, victima poate suferi deteriorări și pierderi la nivelul atât al datelor cât și al integrității platformei. Pentru a evita astfel de atacuri, sistemul utilizează un ORM specializat, Eloquent - care are niște parametri PDO speciali pentru a evita SQL Injection.

Mai jos avem un exemplu în care se presupune că utilizatorul malițios a injectat o expresie suplimentară la request-ul în care se cer date depre autentificarea sa (expresia suplimentară este *or 1=1).*

*SELECT * FROM users WHERE email = 'jason@example.com' or 1=1*

Datorită PDO parameters, sistemul își dă seama despre query greșit și îl transformă în ceva similar cu ce urmează:

*SELECT * FROM users WHERE email = 'jason@example.com or 1=1'*

Pe lângă securizarea împotriva atacurilor nedorite, mai există un aspect foarte important care trebuie tratat. Acesta este filtrarea request-urilor cu ajutorul **serviciilor de middleware**. Fiecare request, fie ajax sau nu, trebuie, în mod obligatoriu să treacă prin intermediul unui filtru, care să permită sau să dea reject acelui request, în funcție de tipul de utilizator care îl accesează (tipul fiind salvat în sesiune preventiv).

Sistemul realizat dispune de un set de filtre native:

- *web* - permite accesul tuturor clienților
- *auth* - permite accesul doar celor autentificați
- *role[nume_rol]* - permite accesul doar acelor utilizatori care au atașat rolul cu numele *nume_rol.*
- *permission[nume_permisiune]* - permite accesul doar acelor utilizatori care au permisiunea setată

Pentru a configura filtre personalizate, acestea se crează după modelul celor native, din directorul *middleware.* După ce se realizează o clasă, aceasta poate fi customizată să facă anumite excepții pentru anumite rute, clasa se indroduce ca dependență în array-ul *routeMiddleware* din cadrul clasei *App/Http/Kernel.*

5.4 Proiectarea Bazei de date

Sistemul propus utilizează o bază de date liberă - MySQL. În diagrama de deployment s-a reprezentat integritatea acesteia, care cum era menționat, rulează un driver pe același server cu compilatorul de PHP și interceptorul NGINX. Baza de date include 50 de tabele relaționate cu chei, și indexate pe coloanele pe care se realizează cele mai frecvente operații.

Fiecare tabel reprezintă de fapt o entitate din lumea reală. Datele sunt stocate pe un singur server chiar dacă în procesul de deployment se pot utiliza două servere balansate pentru procesarea request-urilor.

În procesul proiectării bazei de date s-a ținut cont de normalizarea acesteia, evitându-se câmpurile redundante, dar și alte procese după cum urmează:

- S-a definit o structură clară a tabelelor, cu nume de coloane și tabele sugestive
- Fiecare tabel conține informații legate doar de un singur obiect care a fost definit (ex. plan_plans - conține informații legate doar de planurile de afaceri, care la rândul lor face parte din modulul de planuri)
- Fiecare tabel are o cheie primară
- Câmpurile tabelelor au fost definite în funcție de cel mai optim tip (de exemplu: câmpul ID are mereu tipul INT și nu VARCHAR, deși funcțional este ok și așa și așa)
- Au fost definiți indecși pe coloanele care s-au folosit pentru stabilirea relațiilor între tabele, dar și pe coloanele care necesită căutări frecvente
- Se evită în codul de interogare situații în care se ajunge la full-scan pe tabele (chiar dacă, la început, tabelele sunt de dimensiuni mici)

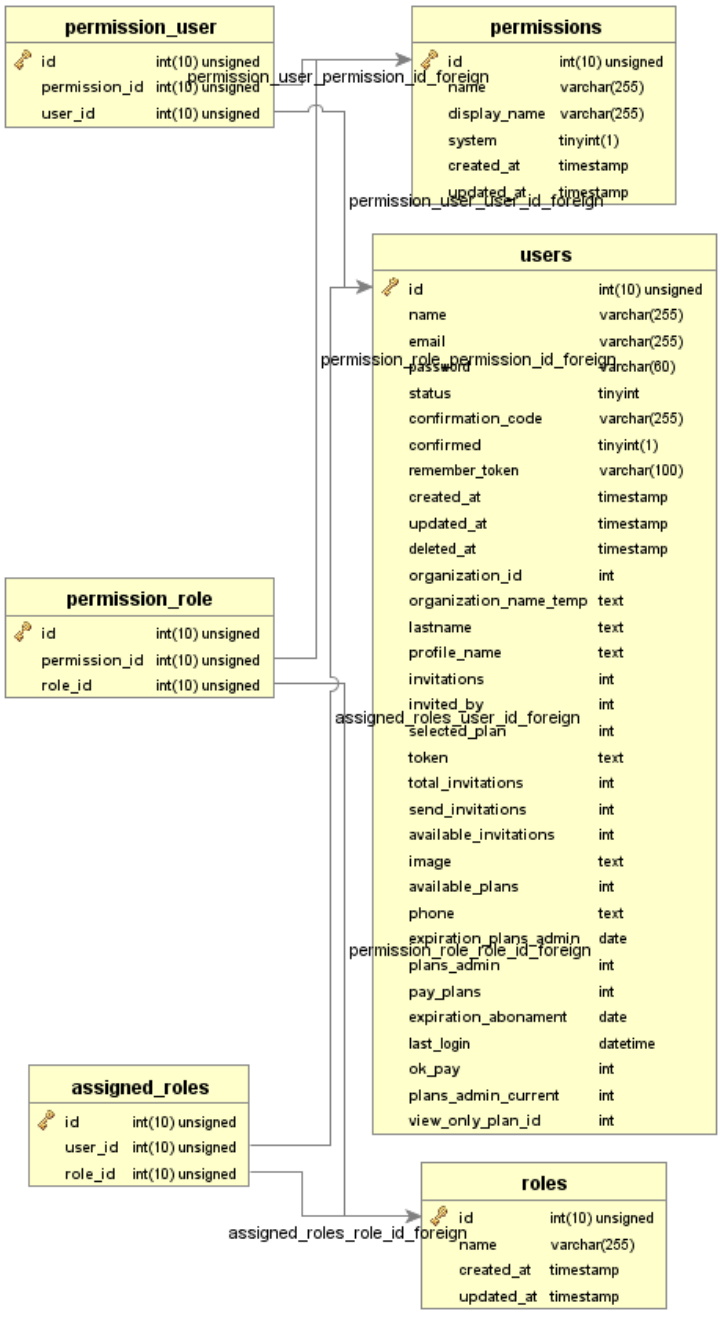

Fig. 5.9 Diagrama bazei de date - utilizatori, roluri și costuri fixe

În digrama din figura 5.9 este reprezentată baza de date pentru modulul de configurare al utilizatorilor. Se poate observa tabelul de *users* și *roles* care, sunt legate prin intermediul relației

assigned_roles , avem o relație *many-to-many.* Tot aici avem tabelul de permisiuni - *permissions,* acest tabel indică operațiile ce pe poate realiza un animit rol, spre exemplu: rolul Administrator, poate realiza operația de Ștergere pe entiatea Utilizatori.

În figura de mai jos se găsește reprezentarea în linii generale a relației planului de afaceri cu celelalte entități de care aparține - prezentare, activități, modul financiar (cu toate subcapitolele), opțiunile de configurare etc.

Fig. 5.10 Diagrama bazei de date - reprezentarea planurilor

În figura 5.10 se poate observa tabelul *plan_plans,* în acesta sunt stocate planurile de afaceri ale clientului. Desigur planurile sunt realizate pe o organizație (firmă), de aceea avem o cheie străină *(organization_id),* folosită pentru a relaționa cu tabelul de organizații.

Pentru modulul financiar se folosesc tabele similare cu cel pentru costuri variabile, acest tabel are numele *costs,* se observă cheia străină *plan_id,* care permite relația cu tabelul de planuri (*plan_plans).*

În tabelul *plan_items* se găsește meniul dinamic din modulul de capitole, pentru a prelucra tabelul dat se utilizeaza o clasă specială - *Baum [24]*. Ideea acestui modul este că trebuie să existe posibilitatea de a crea capitole și subcapitole dinamic ca adâncime, pentru aceasta se folosește o coloană pivot (*parent_id)* care pointează către acest tabel în adâncime.

Pe lângă tabelele funcționale, mai sunt câteva tabele de configurări, acestea sunt necesare pentru a menține sistemul adaptabil tuturor tipurilor de utilizatori. Configurările pot fi la nivelul proiectului (planului) sau la nivelul platformei în general - cum sunt instrucțiunile de lucru din cadrul capitolelor sau ale modulului financiar. Entitățile pentru configurări se pot vedea în urmatoarea imagine:

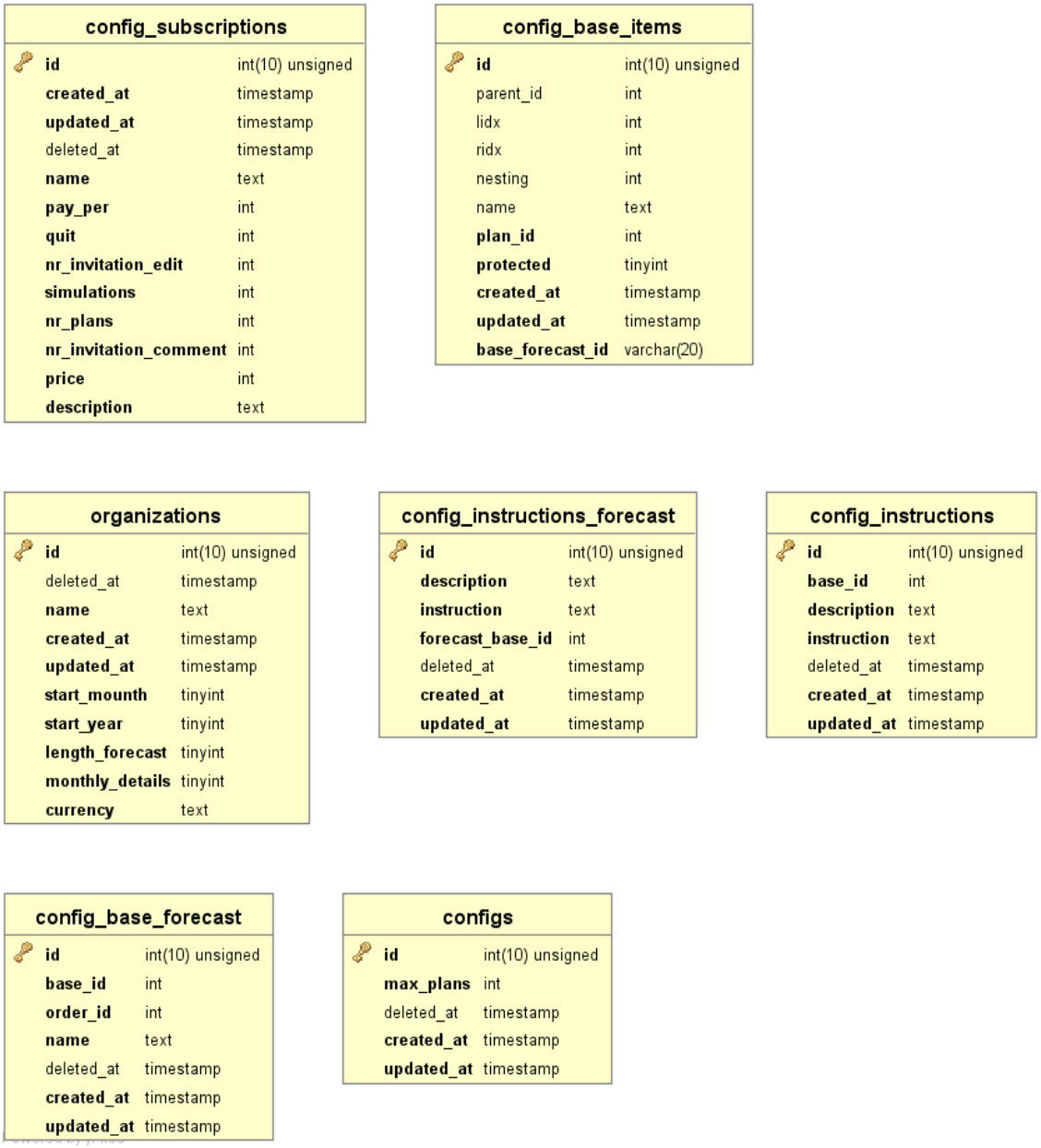

Fig. 5.11 Diagrama bazei de date - tabelele de configurare ale sistemului

Capitolul 6. Testare și validare

Pentru a testa sistemul acest sistem s-au utilizat cateva metode: metoda clasică (manual testing) - pentru verificarea exploatării codului HTML/CSS în relație cu modificările DOM aduse de JavaScript, prin acest tip de testare se acoperă cazurile de captare a evenimentelor pe partea de client, al doilea tip de testare este cea unitară, pusă la dispoziție de mediul PhpStorm și componentele Laravel.

Pentru testarea manuală s-a analizat așezarea controalelor HTML în pagini, elementele de design, operațiile de CRUD pe toate entitățile, corectitudinea formulelor utilizate în dezvoltarea modulului financiar, dar și captarea evenimentelor JavaScript.

Validarea este un proces foarte important, în acest sistem au fost implementate validări atât pe partea de client cât și pe partea de server, acest lucru este necesar pentru a evita manipulările nedorite cu JavaScript-ul din client și respectiv captarea înregistrărilor neconsistente în baza de date.

6.1 Testare automată

Pentru testarea automată se pot utiliza câteva unele. Mediul de dezvoltare în care s-a dezvoltat sistemul, JetBrains PHPStorm 2016.1 (versiunea actuală), pune la dispoziție un modul numit PHP-Unit, pentru a putea testa automat, modular proiectul. Problema utilizării unui astfel de tool constă în faptul că este destul de dificil să se testeze rutele, validitatea acestora, numărul parametrilor utilizați și desigur tipul de request (GET sau POST).

Unealta cea mai robustă pentru a testa sistemul web vine încorporată cu framework-ul Laravel 5.1, aceasta se prezintă printr-o librările - *phpunit* - care permite testarea aplicației prin simpla rulare a comenzii: *calea_spre_phpunit/phpunit.*

În continuare se vor prezenta pașii pentru a testa o anumită funcționalitate.

1. În directorul *tests* se generează o clasă pentru teste, a fost numită *ExampleTest,* codul de inițializare:

```
<?php

class ExampleTest extends TestCase {
```

2. În această clasă se generează o funcție, numele căreia trebuie să înceapă cu *test*, ca și în cazul testării cu utilitarul de la PHPStorm:

```
/**
 * A basic functional test example.
 *
 * @return void
 */
public function testBasicExample()
{
```

3. În această funcție se pot realiza toate verificările nesare pentru a valida anumite funcționalități din sistem. Un lucru de bază care se poate aduce ca exemplu ar fi interogarea unei

rute (de exemplu ruta care ne aduce pe pagina cu lista de planuri de afaceri /*plans-index*), codul pentru o astfel de verificare este:

```
public function testBasicExample()
{
  $response = $this->call('GET', '/plans-index');
  $this->assertEquals(200, $response->getStatusCode());
}
```

4. Vizualizarea rezultatelor, în imaginea de mai jos se poate vizualiza rezultatul rulării unui test:

```
[root@192-168-0-101:/Applications/XAMPP/xamppfiles/htdocs/bsplan$ vendor/bin/phpunit
PHPUnit 4.8.20 by Sebastian Bergmann and contributors.

.

Time: 167 ms, Memory: 14.75Mb

OK (1 test, 1 assertion)
root@192-168-0-101:/Applications/XAMPP/xamppfiles/htdocs/bsplan$
```

Fig. 6.1 Reprezentarea rulării unui test

6.2 Validarea

Urmând stilul OOP toate funcționalitățile încearcă să se focuseze spre o singură clasă părinte, care este responsabilă de interpretarea acțiunii care se dorește a se realiza. Din acest punct de vedere se poate spune că toate funcționalitățile sunt validate de către sistem. Un exemplu îl constituie structura de modele, în care fiecare model extinde o clasă pe post de Observer (*Model)*, iar fiecare acțiune, din orice zonă a proiectului nu s-ar realiza, este validată și logată în fișiere speciale, de către metodele acesteia.

Pe lângă validarea proceselor operaționale, sistemul are responsabilitatea de a proteja baza de date de intrări neconsistente, sau, de a comunica utilizatorului la anumite etape din procesul de lucru, anumite mesaje de validare a controlelor utilizate. Aici se poate vorbi despre validarea datelor de intrare. Tipurile de validări, tipurile de date care necesită validare se extind pe un spectru foarte larg. Framework-ul utilizat pentru dezvoltarea acestui sistem vine cu un set de reguli de validare, care acoperă aproape toate cazurile posibile, dar permite și o extindere, pentru a personaliza algoritmul de validare al anumitor date.

Pentru a valida un anumite set de date care vin spre server, este nevoie în mod obligatoriu de:

- numele campului ce a trimis data
- tipurile de constrângeri pentru el
- mesajele de notificare pentru fiecare constrângere

Mai jos se găsește codul pentru a valida un formular de la formularul pentru costuri variabile:

```
$validator = Validator::make(
   $data,
   Cost::generalValidatorRules(['tip_variatie' => $data['tip_variatie'],]),
   Cost::generalValidatorMessages()
);


if( $validator->passes() )
```

Funcția de validare este stocată static în clasa *Validator,* pentru a valida un array de date (*$data)* se utilizează această funcție cu cei trei parametri de care s-a menționat mai sus. Parametrul *$data* este un array ce ține numele controlului și valoarea sa, funcția *Cost::generalValidatorRules* - returnează o listă de reguli de validare asociate câmpurilor primite:

```
return [
    'cost_name'              => 'required',
    'tip_variatie'           => 'required|combobox',
    'stoc_tampon'             => 'required|floatrange',
    'period_credit'          => 'required|floatrange',
]
```

În parea dreaptă a tabloului asociativ se găsesc regulile de validare, *required* este o regulă implicită din framework, dar *combobox* și *floatrange,* sunt două reguli definite personalizat în clasa *AppServiceProvider*, acestea arată în felul următor:

```
Validator::extend('floatrange', function($attribute, $value, $parameters, $validator)
{
   $value = (float) $value;
   switch(count($parameters)){
      case 0:
         return $value >= 0;
         break;
      case 1:
         return $value >= $parameters[0];
         break;
      case 2:
         return $value >= $parameters[0] && $value <= $parameters[1];
         break;
      default:
         return false;
   }
});
Validator::extend('combobox', function($attribute, $value, $parameters, $validator)
{
   if(strlen($value) > 3 )
      return false;
   return true;
});
```

Al treilea parametru al funcției *make* este un tablou asociativ între numele regulii de validare și mesajul de notificare asociat:

```
return [
    'tip_variatie.required'          => 'Variația trebuie completată',
    'tip_variatie.combobox'          => 'Variația trebuie completată',
    'cost_name.required'             => 'Descrierea trebuie completată',
    'stoc_tampon.required'           => 'Stocul tampon trebuie completat',
```

După cum se poate observa, funcția *passes* de pe obiectul *Validator* indică dacă testul a fost trecut. Și doar după ce sunt validate toate datele pasate spre validare, se poate realiza o modificare în baza de date. Notificarea utilizatorului despre anumite erori de validare se realizează prin JavaScript (AngularJs) - care primește un JSON ce conține un tablou de obiecte cu asocierea dintre numele controlului de interfață și mesajul de validare asociat.

6.3 Concluzii

Scopul primordial pentru procesul de testare este (1) identificarea erorilor de software, (2) izolarea și fixarea defectelor ce au cauzat aceste erori [27]. În general implementarea modulelor de testare automată este un proces destul de complex. În cazul sistemelor web este și mai greu de gestionat modulele de testare, acest lucru este cauzat de limbajul utilizat în mare parte, de exemplu, testarea în JavaScript poate fi implementată în mare parte doar pe funcții pure, care nu depind de starea sistemului și returnează mereu același rezultat (dacă primește aceeași parametri de intrare).

După cum s-a analizat mai sus, sistemul a suportat o serie de teste manuale și automate, a fost securizat din punct de vedere al consistenței datelor prin setarea unui spectru larg de validatoare - acest lucru vorbește despre buna funcționare a sistemului. Testarea a fost implementată atât la nivel de client, tot ceea ce ține de modificările de DOM, fie prin intermediul JavaScript, fie prin generarea paginilor statice de către sistem, au fost testate și analizate toate stările posibile prin care pot să treacă paginile în decursul unui flux de lucru, cât și la nivel de de server cu ajutorul modulelor de UnitTestig.

Capitolul 7. Manual de instalare și utilizare

7.1 Specificații de instalare

Pentru a rula proiectul este nevoie de un pachet de utilitare (software) care ar permite suportul rulării sistemului dar să ofere și posibilitatea dezvoltării uleterioare a acestuia.

Pentru rularea proiectului sunt necesare:

- apache compilator
- php (version >= 5.6)
- mysql server
- spațiu stocare
- composer (version >= 1.0.2)

Pentru dezvoltarea ulterioară a proiectului sunt necesare:

- un mediu de dezvoltare (JetBrains PhPStorm)
- acces la un CLI (command line interface)

Pașii pentru rularea instalarea proiectului în scopul rulării acestuia sunt:

1. Dacă se dorește instalarea sistemului pe sistemul de operare Windows sau OS X, se poate utiliza soft-ul XAMPP, care beneficiază de o interfață grafică intuitivă, cu acces rapid la fișierele de configurare (my.cnf, php.ini, httpd.conf). XAMPP este un utilitar care vine integrat cu serviciul de baze de date MySQL și cu versiune actualizată de PHP, dar și cu un compilator apache pentru rularea codului PHP. Dacă se dorește instalarea pe un sistem linux (ex. Ubuntu, Centos), este necesară instalarea pachetului LAMP.
2. Se pornesc serviciile de Apache si MySQL (acestea pot fi rulate direct din interfața grafică XAMPP sau din linia de comandă CLI)
3. Se copiază fișierele proiectului pe HDD. Acest proces este recomandabil de al realiza prin intermediul git-ului, dar nu reprezintă o regulă.
4. Prin intermediul liniei de comanda, se ajunge în directorul cu proiectul, având precondiția că soft-ul *composer* este deja instalat, se rulează comanda: *composer install*
5. Tot în linia de comandă se rulează comanda: *copy .env.example .env,* aceasta va copia un fișier de configurare general în fișierul .env, care va reprezenta cel mai general punct de configurare al sistemului. În acest fișier este foarte intuitivă ordinea parametrilor:

```
APP_ENV=local
APP_DEBUG=true
APP_URL=localhost/
APP_LOCALE=ro
APP_FALLBACK_LOCALE=ro
APP_KEY=xxmKIa4qJYeXxiL033lNL0kmYNJfVKTq


DB_HOST=127.0.0.1
DB_DATABASE=bsplan
DB_USERNAME=root
```

```
DB_PASSWORD=

DB_DRIVER=mysql
CACHE_DRIVER=file
SESSION_DRIVER=file
QUEUE_DRIVER=sync
```

6. Se rulează comanda *php artisan key:generate* , această comandă va genera o cheie unică pentru acest proiect local.

7. După ce s-au configurat datele de conexiune la baza de date, se încearcă o rulare a versionărilor bazei de date prin comanda *php artisan migrate,* aceasta ar trebui sa inițializeze baza de date cu toate legăturile stabilite.

8. Introducerea datelor inițiale în baza de date, precum și a configurărilor de bază, se realizează prin rularea comenzii: *php artisan db:seed*

9. Ultima comandă necesară rulării proiectului este : *php artisan serve*, aceasta va rula proiectul pe portul 8080.

10. Se acesează într-un browser url-ul: localhost:8080.

7.2 Manual de utilizare

7.2.1 Înregistrarea și logarea în sistem

Această platformă oferă 2 posibilități de inregistrare a utilizatorilor, prima este de către utilizator, introducând datele: nume, prenume, email, parolă, repetare parolă, telefon, aceste date vor fi aprobate de către administratorul platformei (spre organizația selectată, dacă este autorizat să posede acces), și a doua metodă: utilizatorul nou, trimite aceste date administratorului, acesta oferindui acces pe organizația solicitată și autorizația respectivă.

Ca utilizator autorizat al aplicației, după ce veți primi pe adresa de email furnizată de dumneavoastră un nume de utilizator si parola, veți putea să vă conectați la sistem. Dacă este cazul, la prima logare pe platformă, vi se va cere să vă schimbați parola cu una aleasă de dumneavoastră. Vă rugăm să țineți minte această parolă.

7.2.2 Meniul sistemului

Pe pagina principală a aplicației există opțiunea de a înregistra un nou utilizator. După finalizarea completării datelor personale, utilizatorul va primi un e-mail de activare al contului și va trebui să facă clic pe un link. Odată activat contul, utilizatorul va accesa aplicația. E-mail de activare are o perioadă de valabilitate limitată. În cazul în care contul este activat în această perioadă de valabilitate, trebuie să repetați procesul de înregistrare.

După efectuarea logării, meniul principal al aplicației va fi afișat. În fereastra principală, în funcție de meniul selectionat, se va afișa conținutul principal al paginii specifice acestuia.

Meniul oferă acces la funcționalitățile pe care le oferă aplicația iar în submeniuri sunt puse la dispoziție instrumente cu cele mai importante funcții pentru fiecare. Unele dintre submeniuri au o opțiune drop-down pe partea dreaptă a butonului, care permite acces rapid elemente suplimentare. Opțiunile care sunt disponibile în fiecare submodul sunt descrise în secțiunile specifice. Aplicația va actualiza în mod automat modificările efectuate pentru toți utilizatorii logați, atunci când interacționează cu acesta (de exemplu, de fiecare dată când utilizatorul salvează niște date în baza de date sau comută între aplicații. Atunci când nu există nici o interacțiune cu aplicația, aceasta va actualiza automat , la fiecare 15 secunde, pagina actuală, pe care se află logat utilizatorul.

Meniuri sistemului sunt următoarele:

- **Meniul "Acasă"** – Aceasta fereastră este pagina în care utilizatorul are acces la lista de planuri pe care le poate gestiona. Tot aici poate selecta pentru lucru curent, adăuga, șterge, dublica, edita sau șterge un plan existent.

- **Meniul "Previziuni"** – Acest meniu definește toate modulele financiare de pe platformă, separate prin submeniuri din pagina respectivă.

- **Meniul "Prezentare"** – Aceasta este pagina care permite utilizatorului să-și definească detaliile legate de firma pe care se face planul financiar selectat.

- **Meniul "Plan"** – Este o altă zonă structurală, care include o structură de capitole predefinite, pentru a le pune la dispoziție spre completare utilizatorului.

- **Meniul "Activități"** – Această pagina oferă utilizatorului capabilitatea de a gestiona lista de activități necesare în scopul realizării obiectivelor puse în planul de afaceri.

- **Meniul "Opțiuni"** – Platforma a fost gândită să ofere o pagină de configurări separată pentru fiecare plan de lucru în parte.

- **Meniul "Cont"** – Aici utilizatorul își poate personaliza profilul (nu este atașabil unui plan), poate avea acces la modulul comercial etc.

- **Meniul "Share"** – Pagina respectivă pune la dispoziție o modalitate simplă de generare a planului sub formă de document (pdf), și trimiterea acestuia ca atașament direct în mail unui consultant financiar.

- **Meniul "Listă utilizatori"** – În această pagină se vor vizualiza utilizatorii înregistrați pe platformă care fac parte din aceeași organizație cu utilizatorul logat deja. Acest meniu este util în cazul în care departamentele organizației sunt situate geografic în puncte diferite și apare necesitatea de a lua legătura cu un coleg. Aici sunt datele de contact primordiale care au fost înregistrate în momentul înregistrării pe platformă.

- **Meniul "Administrare"** – Aici avem acces la nomenclatoarele sistemului.

7.2.3 Detalierea funcționalităților

Administrare cont si modul de plată - cont pe platforma isi poate face orice utilizator, care pe langa faptul ca este notificat pe mail de eventuale modificari administrative, dispune de un modul de plata a serviciilor, realizat prin intermediul API-ului de la Euplatesc.ro. Securitatea este asigurata de serviciul de SSL. În imaginea de mai jos se observă afișarea informațiilor referitoare la un abonament și butonul care duce pe pagina de plată.

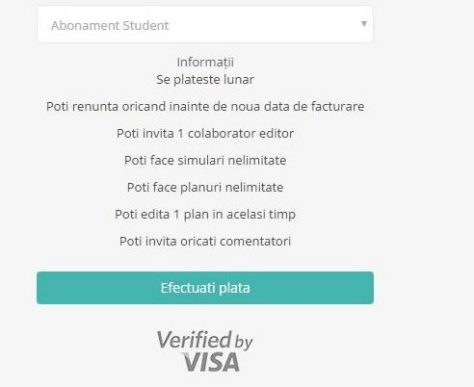

Fig. 7.1 Reprezentarea informațiilor despre un tip de abonament

Configurarea profilului - deoarece platforma este personalizată dupa taxele și impozitele stabilite in Romania, punem la dispozitie utilizatorului un modul foarte dinamic de configurare, pentru a-i oferi posibilitatea de a seta numele firmei, sigla, inceputul activitatii (utila pentru previziuni), tipul de impozit acceptat (IMM, II etc.), moneda si perioada de monitorizare a proiectului.

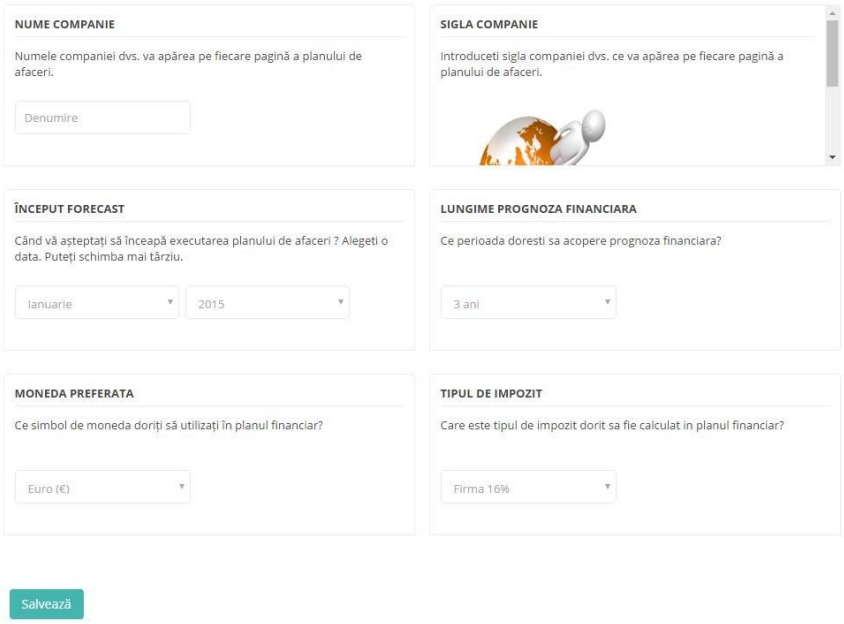

Fig. 7.2 Pagina de configurare a companiei

Planurile de afacere – este o zona speciala in care fiecare utilizator, chiar si cei care nu au un cont platit, pot sa-si vizualizeze planurile de lucru create pe parcursul activitatii. Modulul este dotat cu o functionalitate de acomodare a utilizatorilor noi pe platforma, prin care sistemul ii prezinta utilizatorului fiecare functionalitate, intr-un mod prietenos si foarte intuitiv. Pe aceasta sectiune dispunem de functionalitati generale care opereaza pe entitatea planului (CRUD) printre care si functionalitatea de dublicare a planului curent si selectare a planului pe care se doreste a se lucra.

Fig. 7.3 Lista planurilor de afaceri

Modul de prezentare – este sectiunea softului care se afla intr-o relatie 1-1 cu planul de afaceri. Aceasta sectiune permite configurarea firmei pe care se face acel plan de afaceri. Aceste functionalitati dispun de caracteristica one page application, si sunt foarte intuitive. In linii mari, aici descriem – Compania, Piata tinta, Competitivitatea, Finantarea, Prognoza, Echipa si roluri cheie, Parteneri etc. Imaginea din figura 7.4 reprezintă primul câmp de la personalizarea modulului de prezentare:

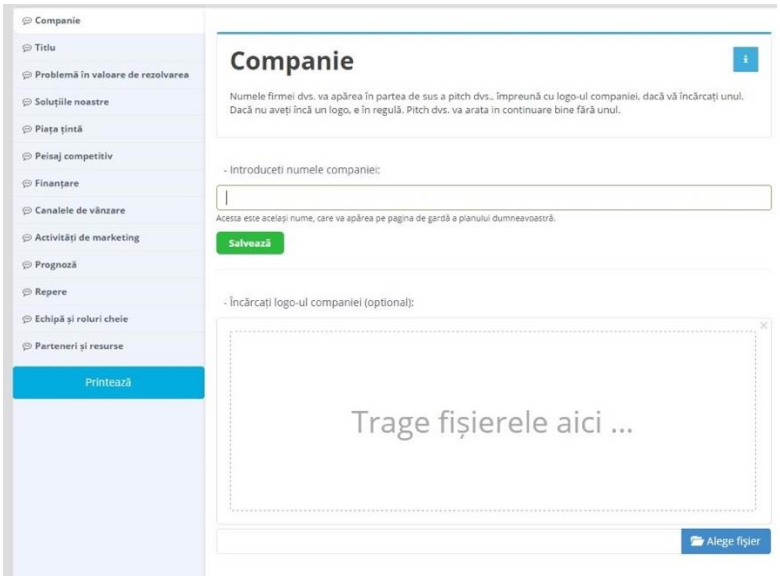

Fig. 7.4 Reprezentarea paginii de editare a companiei

Capitolele planului de afacere – orice plan de afaceri dispune de un un numar de capitole sablon care trebuie completate. In acest sens am facut un pas in fata, punand la dispozitie posibilitatea gestionarii dinamice a capitolelor si subcapitolelor planului de afacere, astfel incat sa fie cat se poate de customizabil procesul de creare a planului de afaceri pe un anumit proiect. Aici dispunem de un sumar pe fiecare capitol, dar si de o sectiune in care se preiau date din modulul de previziuni financiare, pentru a genera un raport de activitate cat mai clar.

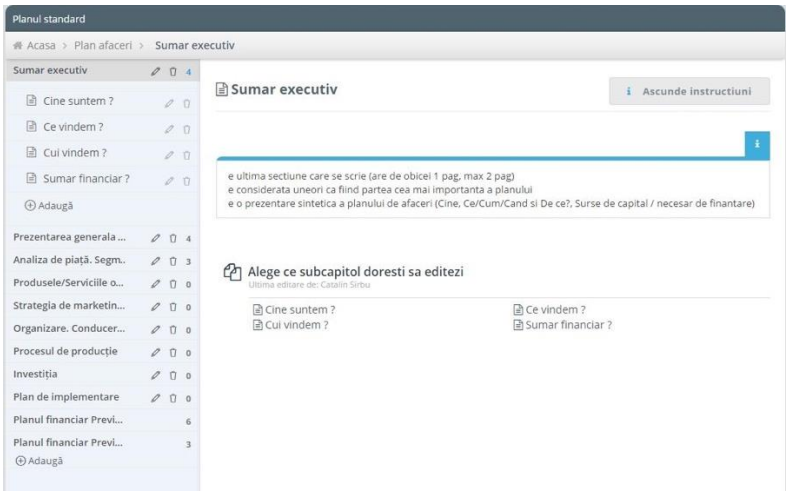

Fig. 7.5 Reprezentarea modulului de capitole

Activitati – lista de activitati pentru fiecare personaj din echipa. Aceasta lista dispune de o afisare sub forma de grafic Gantt, ceea ce implica adaugarea de activitati relationate intre ele.

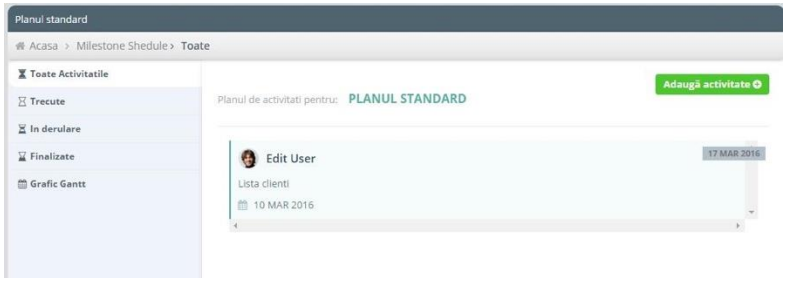

Fig. 7.6 Reprezentarea modulului de activități

Previziuni – Previziunile financiare sunt un modul foarte complex. Acesta trebuie sa se conformeze cu data de inceput a perioadei de activitate (care poate fi setata dinamic in configurarile generale). Pe fiecare linie de activitate regasim posibilitatea de a introduce datele fie pe luni de activitate sau o valoare predefinita pentru toata perioada. De asemenea avem exemple si instructiuni de introducere a datelor. Pentru a contura grafic activitatea, au fost create grafice (pe luni) si subgrafice (pe ani) care ne vor indica dinamicitatea serviciilor prestate. Modulul este compus din submodulele Venituri, Costuri (Varbiabile/Fixe/Personal), Resurse, Finantare (Capitaluri/Credite/Linie de credit), Rapoarte (CPP, Bilant, CF). Vizualizarea rapoartelor de asemenea poate fi pe 3 sau 5 ani.

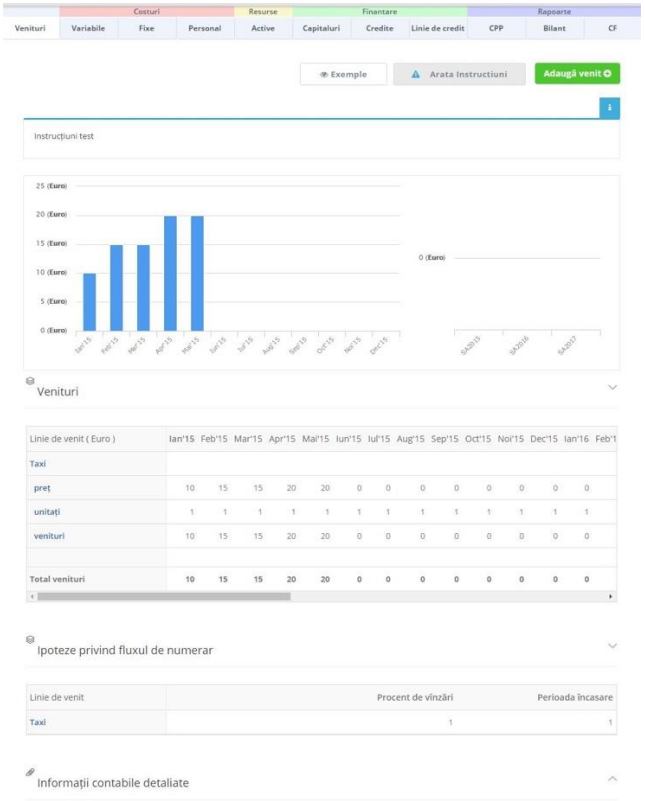

Fig. 7.7 Reprezentarea modulului de previziuni financiare

Conturi conectate – aici avem un modul interesant, in care utilizatorii pot invita spre colaborare alti utilizatori ai platformei. Acesta le poate permite sa comenteze capitole, sa editeze sau doar sa citeasca capitole si date din plan. Desigur toate operatiile sunt introduse in loguri de activitate.

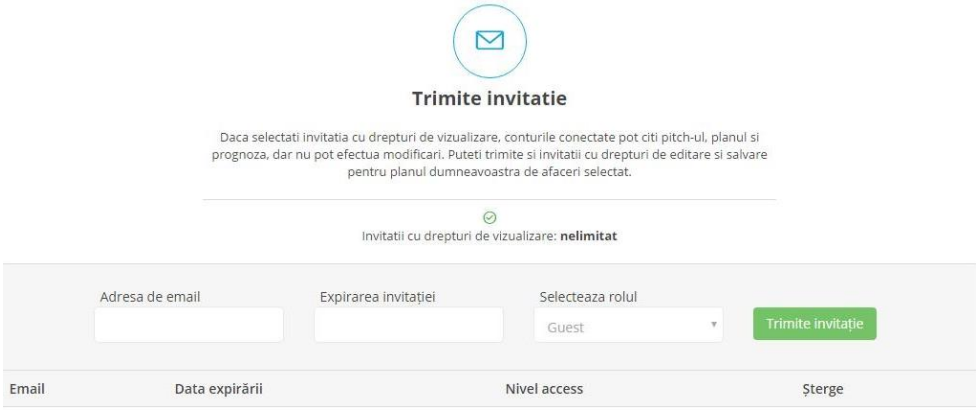

Fig. 7.8 Reprezentarea modulului de invitații

Raportare – Raportul include tot planul de afaceri cu previziunile aferente. Acesta poate fi create in pdf sau trimis prin email unui constultant.

Capitolul 8. Concluzii

În acest capitol se va prezenta tot ce s-a realizat prin intermediul acestui sistem inteligent, prezentarea realizându-se pe baza obiectivelor propuse și se va vizualiza prin prisma unor îmbunătățiri viitoare.

8.1 Realizarea obiectivelor propuse

În urma analizei cazurilor de utilizare, a performanțelor tehnice și a capabilităților de configurare ale sistemului se poate trage concluzia că sistemul este unul competitiv cu altele similare.

Atât obiectivele principale cât și cele secundare propuse pentru implementarea sistemului, au fost atinse integru - scopul primordial, integrarea și vizionarea unui număr nelimitat de planuri de afaceri din cadrul unei firme, pentru a putea lua anumite decizii de implementare - acest obiectiv a fost atins în totalitate, fiecare utilizator își poate crea și urmări evoluția unui număr nelimitat de planuri, pe fiecare dintre ele se pot adăuga cifre financiare diferite pentru a se vizualiza grafice comparative.

Dintre obiectivele secundare, posibilitatea generării unui document PDF, sub formă de raport, care să includă toate specificațiile planului de afaceri - a fost atins cu succes, pe lângă generarea acestui raport, el poate fi trimis direct printr-un e-mail unui consultant financiar spre exemplu.

Obiectivul separării și securizării rolurilor din cadrul platformei a fost asigurat prin introducerea filtrelor și middleware. Stocările la nivelul bazei de date, validate fiind, sunt atașate doar utilizatorilor care gestionează în acel moment planul, această breșă de securitate este înlăturată prin utilizarea sesiunilor.

Modulul financiar dispune de o precizie exactă, modelată după sistemul românesc în ceea ce privește aspectele specifice (de exemplu închiderea conturilor se realizează în luna Decembrie, pe când în alte state aceasta se închide la exact un an de la deschiderea firmei).

Posibilitatea acordării de suport în decizii economice de către un consultant, este asigurată de funcționalitatea trimiterii invitațiilor de acces cu diferit nivel de rol (citire, comentare). Această funcționalitate asigură integritatea planului proprietarului, dar totodată oferă acces partajat (unui utilizator aflat la distanță din punct de vedere geografic) la planul de lucru, pentru a lua anumite decizii.

8.2 Dezvoltări ulterioare

Fiind un sistem informatic care rezolvă o problemă cu aspect financiar, este inevitabilă necesitatea actualizărilor în timp. Fie din cauza schimbărilor de regulamente politice, fie din cauza faptului că sistemul pur și simplu nu acoperă toate cazurile de utilizare necesare unei firme în practică, necesitatea implementării noilor funcționalități este un proces inevitabil în timp.

Din perspectiva altor sisteme similare, dezvoltările ulterioare ar putea fi necesare se vor enumera mai jos.

- Sistemul are posibilitatea de a simula, la capitolul financiar, venituri din anumite surse. Una dintre aceste surse este "linie de credit", sau "credit". Pornind de la această premiză, în cadrul sistemului s-ar mai putea implementa un modul destul de mare, care să includă posibilitatea simulării unui credit, începând de la condițiile de eligibilitate ale băncilor și terminând cu documentația necesară pentru a îndeplini criteriile acestora.
- În urma elaborării planuli de afaceri, care se bazează pe capitolele documentate, acesta poate fi elaborat într-un document PDF, sub forma unui raport. La această funcționalitate de poate veni cu o actualizare care ar ajuta mult experiența utilizatorului, și anumite - implementarea unor șabloane, pe baza cărora să se genereze documentul. Aceste șabloane ar putea modifica atât ordinea apariției anumitor zone, cât și formatul, culorile, imaginile de copertă etc.
- Necesitatea suportului din partea unui consultant în activitatea elaborării planului de afaceri este foarte importantă, de aceea interacțiunea acestor două entități ar trebui tratată mai mult. În acest sens, sistemul ar putea beneficia de un chat intern, prin care să se poată comunica între diferite nivele de entități. Cele mai importante relații sunt văzute ca fiind: membrii aceleași firme (organizații), un membru al unei firme și un invitat pentru suport.
- Deoarece sistemul va stoca date foarte importante din punct de vedere economic, ar fi un mare avantaj realizarea unei copii de rezervă a bazei de date. Adică la nivelul serverului unde este stocată baza de date s-ar putea implementa un sistem de management al copiilor de rezerva.
- Dat fiind faptul că sistemul permite configurarea datelor personalizate ale firmei, formulele din spatele calculelor matematice (de la modulul financiar) sunt realizate în funcție de acei parametri și alți parametri considerați constanți la nivelul țării. Dar odată cu schimbarea legilor, se pot modifica și anumite formule, sau date limită pentru predarea anumitor documente. În acest caz sistemul nu va mai ține pasul, drept motiv pentru care, în acest scop se poate implementa un modul inteligent, care să comunice cu un API întreținut de o organizație de stat (ANAF spre exemplu).
- Sistemul oferă posibilitatea selectării monezii dorite, dar acest lucru nu modifică calculele, practic se consideră aceleași cifre, doar că în altă monedă. La acest capitol însă, s-ar putea realiza o comunicare cu site-ul BNR, pentru a prelua cursul actualizat al monezii utilizate, pentru a reface toate calculele în funcție de modificarea realizată de utilizator.

Bibliografie

[1] Alex Chaffee (2000-08-17). "What is a web application (or "webapp")?".
[2] *Anderson, Jerry (1997). Activex Programming with Visual C++. Que. ISBN 978-0-7897-1030-7.*
[3] Documentația oficială Laravel **:** https://laravel.com/
[4] The Best PHP Framework for 2015: SitePoint Survey Results https://www.sitepoint.com/best-php-framework-2015-sitepoint-survey-results/
[5] Web Framework Benchmarks http://www.techempower.com/benchmarks/#section=data-r9&hw=ec2&test=json&l=3y8&c=1
[6] *"ECMAScript 2015 Language Specification" (PDF).Ecma International. June 2015.*
[7] Guzzle PHP HTTP client library https://packagist.org/packages/guzzle/http
[8] Blade Templates https://laravel.com/docs/5.1/blade
[9] Logic-less templates http://mustache.github.io/
[10] *"A Relational Database Overview". https://oracle.com.*
[11] "Gray to be Honored With A. M. Turing Award This Spring". Microsoft PressPass. 1998-11-23. Archived from the original on 6 February 2009. Retrieved 2009-01-16.
[12] Gray, Jim (September 1981). "The Transaction Concept: Virtues and Limitations" (PDF). Proceedings of the 7th International Conference on Very Large Databases. 19333 Vallco Parkway, Cupertino CA 95014: Tandem Computers. pp. 144–154. Retrieved2006-11-09.
[13] Gray, Jim, and Reuter, Andreas, Distributed Transaction Processing: Concepts and Techniques. Morgan Kaufmann, 1993.ISBN 1-55860-190-2.
[14] Codd, E.F. (1970). "A Relational Model of Data for Large Shared Data Banks". Communications of the ACM **13** (6): 377–387. doi:10.1145/362384.362685.
[15] "What is MySQL?". MySQL 5.1 Reference Manual. Oracle. Retrieved 17 September 2012. The official way to pronounce “MySQL” is “My Ess Que Ell” (not “my sequel”)
[16] Urlocker, M. Zack (13 December 2005). "Google Runs MySQL". The Open Force. M. Zack Urlocker. Retrieved3 August 2010. AdWords was built using the MySQL database
[17] Claburn, Thomas (24 April 2007). "Google Releases Improved MySQL Code". InformationWeek (InformationWeek). Retrieved30 November 2008.
[18] Callaghan, Mark (13 April 2010). MySQL at Facebook.YouTube (Google). Retrieved 3 August 2010. x,000 servers, ... Master-slave replication, InnoDB
[19] Sobel, Jason (21 December 2007). "Keeping Up". The Facebook Blog. Facebook. Retrieved 30 October 2008.
[20] Malik, Om (25 April 2008). "Facebook’s Insatiable Hunger for Hardware". GigaOM. GigaOmniMedia. Retrieved 30 October2008.

[21] Cole, Jeremy (14 April 2011). Big and Small Data at @Twitter.YouTube (Google). Retrieved 20 October 2011.
[22] AngularJs https://en.wikipedia.org/wiki/AngularJS
[23] Bower - A package manager for the web https://bower.io/
[24] Baum - Implementation of the Nested http://etrepat.com/baum/
[25] *Stefanov, Stoyan (2010). JavaScript Patterns. O'Reilly Media, Inc. p. 5. ISBN 9781449396947. Retrieved 2016-01-12. The core JavaScript programming language [...] is based on theECMAScript standard, or ES for short.*
[26] Data Validation, Data Integrity, Designing Distributed Applications with Visual Studio .NET
[27] Kaner, Cem; Falk, Jack and Nguyen, Hung Quoc (1999). Testing Computer Software, 2nd Ed.. New York, et al: John Wiley and Sons, Inc.. pp. 480 pages. ISBN 0-471-35846-0

Anexa 1 Lista figurilor și a tabelelor din lucrare

Lista tabelelor din lucrare

Anexa 2 Glosar de termeni

API	Application Programming interface
URI	Uniform Resource Identifier, Accesul la aceste resurse se face cu ajutorul unei adrese
HTTP	Hypertext Transfer Protocol
MySQL	My Structured Query Language; database management system
AJAX	Asynchronous JavaScript and XML
XML	Extensible Markup Language
JSON	JavaScript Object Notation
BPM	Business process management
REST	Representational State Transfer
BLADE	Numele motorului de interpretare a șabloanelor
NGINX	Web server
DOM	Document object model
OOP	Object oriented programming