Sistem de lansare automată a aplicațiilor web în cloud - Gitdone

LUCRARE DE LICENȚĂ

Absolvent: **Constantin-Andrei ARBA**

Coordonator ştiinţific: **Șef lucrări ing. Cosmina IVAN**

2014

DECAN,
Prof. dr. ing. Liviu MICLEA

DIRECTOR DEPARTAMENT,
Prof. dr. ing. Rodica POTOLEA

Absolvent: **Constantin-Andrei Arba**

Sistem de lansare automată a aplicațiilor web în cloud - Gitdone

1. **Enunţul temei:** *Lucrarea își propune realizarea unei aplicații web, care să efectueze în mod automat etapa de lansare a aplicațiilor web*

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*

3. **Locul documentării**: Universitatea Tehnică din Cluj-Napoca, Departamentul Calculatoare

4. **Consultanţi**: Șef lucrări ing. Cosmina Ivan

5. **Data emiterii temei:** 1 noiembrie 2013

6. **Data predării:** 11 Septembrie 2014

Absolvent: _____

Coordonator ştiinţific: _____

Declaraţie pe proprie răspundere privind autenticitatea lucrării de licenţă

Subsemnatul(a)_____ _____, legitimat(ă) cu _____ seria _____ nr. _____ CNP _____, autorul lucrării _____ _____ _____elaborată în vederea susţinerii examenului de finalizare a studiilor de licență la Facultatea de Automatică şi Calculatoare, Specializarea _____ din cadrul Universităţii Tehnice din Cluj-Napoca, sesiunea _____ a anului universitar _____, declar pe proprie răspundere, că această lucrare este rezultatul propriei activităţi intelectuale, pe baza cercetărilor mele şi pe baza informaţiilor obţinute din surse care au fost citate, în textul lucrării, şi în bibliografie.

Declar, că această lucrare nu conţine porţiuni plagiate, iar sursele bibliografice au fost folosite cu respectarea legislaţiei române şi a convenţiilor internaţionale privind drepturile de autor.

Declar, de asemenea, că această lucrare nu a mai fost prezentată în faţa unei alte comisii de examen de licenţă.

In cazul constatării ulterioare a unor declaraţii false, voi suporta sancţiunile administrative, respectiv, *anularea examenului de licenţă*.

Data

_____

Nume, Prenume

_____

Semnătura

Cuprins

Abstract

Obiectivul principal al acestei lucrări este surprinderea importanţei fazei de lansare a unei aplicaţii web şi depăşirea dificultăţilor pe care aceasta le ridică programatorilor de aplicaţii, în mod special partea de configurări care aparţine tot acestei faze a dezvoltării web, prin utilizarea unui sistem care să realizeze aceşti paşi, în mod automat, transformând această fază anevoioasă într-o secvenţă de selecţii şi click-uri. Gradul de actualitate şi utilitate al lucrării este amplificat de introducerea şi utilizarea serviciului de cloud. Astfel lansarea aplicaţiilor se va face pe servere obţinute printr-un API, de la un furnizor de servicii cloud.

1. Introducere

1.1. Contextul general

Odată cu apariţia calculatoarelor şi dezvoltarea lor, a aparut nevoia de a realiza interactiunea între acestea. Astfel, pe baza reţelelor de comunicaţii deja existente au apărut reţelele de calculatoare. O reţea de calculatoare reprezintă un mod de conectare a unor calculatoare individuale, astfel încât să poată folosi în comun anumite resurse.Dezvoltarea acestor reţele de comunicaţii a condus la apariţia reţelei Internetapărută în 1965. Faptul că această imensă reţea de reţele se întindea pe tot globul, interconectând o mulţime de dispozitive, a necesitat stabilirea anumitor protocoale de comunicaţii. Astfel, au apărut protocoalele fundamentale ale Internetului, care asigură interoperabilitatea între orice două calculatoare sau dispozitive inteligente care le implementează şi doresc sa comunice. Acestea sunt: Internet Protocol (IP),Transmission Control Protocol (TCP) şi User Datagram Protocol (UDP). Pe langăaceste protocoale care asigură transportul datelor în reţea au apărut şi protocoalele de la nivel de aplicaţie dintre care cel mai important pentru lucrarea de faţă este protocolul HTTP (HyperText Transfer Protocol)[1]. Pe baza acestui protocol a apărut site-urile sau aplicaţiile web.

O aplicaţie web este o aplicaţie care rulează pe un server web putând fi accesată printr-un browser de un număr nelimitat de clienţi. Aceste aplicaţii sunt foarte populare şi au cunoscut o crestere exponenţială, datorită faptului că sunt foarte uşor de accesat, utilizatorul având nevoie doar de un browser(thin client) iar restul responsabilităţilor care să asigure schimbul de date între cele două, revin serverului. Comunicarea între client şi server se realizează prin intermediul protocolului HTTP(HyperText Transfer Protocol). O aplicaţie web este compusă din două mari părţi: partea de back-end şi partea de front-end.

Partea de back-end este partea care se află efectiv pe server şi ruleaza acolo, iar în momentul în care apare o cerere a clientului, face o serie de operaţii şi trimite rezultatul catre acesta. De obicei pe partea de back-end se foloseşte un limbaj de programare orientat pe obiect.

Partea de front-end este partea aplicaţiei web care este trimisă ca răspuns al cererilor de către server şi rulează pe maşina clientului, într-un browser. Această parte este compusă dintr-o combinaţie între limbajele HTML(HyperText Markup Language) folosit pentru a reprezenta elementele în pagină, CSS(*Cascading Style Sheets*), folosit pentru stilizarea şi poziţionarea elementelor de HTML şi Javascript un limbaj de

programare care permite interacţiunea între elemente şi este capabil să realizeze în mod dinamic anumite modificări asupra elementelor din pagină, după ce aceasta este încarcată în browser, sau să realizeze o serie de operaţii atunci când apare un anumit eveniment.

Fiecare din aceste limbaje s-au dezvoltat în timp, obţinându-se framework-uri şi biblioteci care eficientizează şi facilitează dezvoltarea de aplicaţii web contribuind la o creştere semnificativă a numărului de aplicaţii web.

1.2. Contextul aplicaţiei

După partea de dezvoltare, în ciclul de viaţă al unei aplicaţii web urmează etapa de testare, apoi etapa de lansare, în care aplicaţia web ajunge pe un server de unde rulează pentru a servi clienţii. Această etapă constituie o problemă majoră pentru programatorii de aplicaţii web, întrucât în acest pas, pe lângă mutarea efectivă a codului sursa pe server, apar o serie de configurări ale mediului în care aplicaţia urmează să funcţioneze, acestea constituind de multe ori o problemă destul de serioasă pentru dezvoltatori. Sistemul descris de către lucrarea de faţă, are menirea de a face ca cea de-a treia etapă din ciclul de viaţă a unei aplicaţii web să se realizeze foarte uşor, prin intermediul unei interfeţe grafice de pe o pagină web, cu o secvenţă de selecţii şi click-uri, configurările realizându-se automat în funcţie de selecţiile utilizatorului. Aceasta se va putea realiza pentru aplicaţiile construite pe platforma wordpress, cât şi pentru aplicaţiile realizate pe framework-uri în php, ceea ce reprezintă un procent destul de mare din totalul aplicaţiilor web. În continuarea fazei de deploy, sistemul va permite realizarea unor operaţii specifice şi celei de-a patra faze din ciclul de viaţă al aplicaţiilor web, faza de mentenanţă. O opţiune importantă, tipică fazei de mentenanţă este adăugarea de noi versiuni ale aplicaţiei, de fiecare dată când utilizatorul va adauga o nouă versiune a aplicaţiei, cu un simplu click şi aplicaţia de pe server va fi la zi cu modificările. O altă caracteristică tipică fazei de mentenanţă este aceea că aplicaţia va permite utilizatorilor resetarea fizică a serverelor atunci când apare o problemă în cadrul acestora.

1.3. Conţinutul lucrării

Lucrarea de faţă este structurată astfel:

Capitolul 1 prezintă contextul general şi specific proiectului. Aici se face încadrarea lucrării în context, prin prezentarea şi descrierea conceptul de aplicaţie web, urmând o scurtă prezentare a structurii unei astfel de aplicaţii, ajungându-se treptat la necesitatea fazei de lansare a sistemului care constituie de fapt, obiectivul principal al lucrării.

Capitolul 2 descrie obiectivele urmărite în cadrul lucrării, împărţindu-le în obiective principale, respectiv obiective secundare.

Capitolul 3 prezintă starea actuală a domeniului, adică gradul de dezvoltare în prezent în ceea ce priveşte automatizarea etapei de lansare a aplicaţiilor web, făcând o paralelă între cele câteva sisteme similare existente, rezultând necesitatea unui sistem simplu, rapid şi ieftin pentru utilizatori.

Capitolul 4 conţine descrierea generală a sistemului şi a principalelor sale caracteristici, un desen cu arhitectură urmat de cerinţele funcţionale şi non-funcţionale, iar apoi principalele capabilităţi ale sistemului ilustrate sub forma de scenarii de utilizare,

capitolul încheindu-se cu prezentarea principalelor tehnologii şi protocoale utilizate în dezvoltarea aplicaţiei.

Capitolul 5 conține detalii de implementare și diagrame de clase și de secvențe pentru cele mai importante module.

Capitolul 6 prezintă cateva informaţii legate de testarea aplicaţiei.

Capitolul 7 este un manual de utilizare.

Capitolul 8 descrie obiectivele realizate, dezvoltările ulterioare și contribuțiile personale în realizarea sistemului.

2. Obiective

Acest capitol prezintă obiectivele aplicaţiei, catalogându-le în obiective principale respectiv obiective secundare.

2.1 Obiective principale

Obiectivul principal al lucrării de faţă este realizarea unui sistem care să fie capabil să realizeze în mod automat etapa de lansare a unei aplicaţii web, dezvoltată pe platforma wordpress sau pe frameworkul laravel 4.0 pe limbajul orientat obiect, php. Această etapă de lansare va consta în obţinerea codului sursă de la utilizator, crearea unui server în cloud, realizarea configurărilor necesare pentru ca aplicaţia să poată rula pe serverul astfel obţinut şi în final mutarea codului sursă a utilizatorului pe noul server, în aşa fel încât să ruleze fără probleme şi să deservească aplicaţiile client. Aşadar obiectivul principal al sistemului este realizarea fazei de lansare a aplicaţiilor utilizatorilor, pentru ca în final, acestea să-şi poată la rândul lor deservi aplicaţiile client.

2.2 Obiective secundare

Pentru realizarea obiectivului principal al lucrării de faţă, este necesară realizarea unor serii de paşi care pot fi considerate obiectivele secundare. La acestea se adaugă şi câteva funcţionalităţi suplimentare pe care le va oferi aplicaţia. In continuare vor fi detaliate aceste obiective secundare, pentru a împărţi aplicaţia în componente uşor de înteles şi implementat.

Înregistrarea utilizatorilor

Unul din obiectivele secundare ale sistemului este înregistrarea. Este o componentă importantă a acestuia, deoarece cu ajutorul său se pot identifica şi menţine utilizatorii aplicaţiei.În acest scop, aplicaţia va avea un formular de înregistrare prin intermediul căruia viitorii utilizatori se pot înscrie în sistem. În această fază se va folosi tehnica de confirmare a contului prin intermediul adresei de email, pentru a reduce încercările de înregistrări false. Utilizatorii înscrisi vor fi stocaţi într-o tabelă pe bază căreia li se vor putea asocia aplicaţii şi vor putea fi verificate datele furnizate în faza de autentificare.

Autentificarea utilizatorilor

O altă componentă importantă a aplicaţiei este partea de autentificare, necesară pentru a asigura doar accesul utilizatorilor autorizaţi în sistem. Accesul se va face prin furnizarea numelui de utilizator sau a adresei de email împreună cu parola, care va fi salvată în baza de date criptată, pentru a contribui la creşterea nivelului de securitate al sistemului. Odată autentificat, utilizatorul va putea să folosească funcţionalităţile pe care le oferă sistemul, aceea de a lansa aplicaţii web şi a de le întreţine. Orice încercare de a accesa un link al aplicaţiei fără a avea un utilizator autentificat, chiar şi prin tastarea manuala a adresei, va face o redirectare la pagina de autentificare.

Recuperare/resetare parolă

Tot de această parte de autorizare a accesului ţine şi capabilitatea sistemului de a oferi utilizatorilor săi posibilitatea de a reseta sau recupera parola în cazul în care acesta nu mai dispune de cuvântul cheie folosit la autenficare. Pentru a realiza aceasta, utilizatorul va introduce adresa de email, pe baza căreia va avea posibilitatea să-şi recupereze sau reseteze parola de acces, după care se va putea folosi din nou de funcţionalităţile pe care le oferă aplicaţia.

Sincronizarea contului de utilizator cu cel de github/bitbucket

Pentru a putea obţine codul sursă al aplicaţiilor pe care utilizatorii doresc să le lanseze în cloud, există câteva variante. Cea mai evidentă şi mai primitivă dintre ele este adaugarea unui element care face posibilă încărcarea de fişiere de către utilizator, acesta urmând a încarca o arhivă conţinând codul sursă al aplicaţiei sale. În lucrarea de faţă, vom încerca să evităm această metodă, pentru a facilita cât mai mult accesul utilizatorului la funcţionalităţile principale ale sistemului. Astfel ca cea mai bună metodă pentru a obţine codul sursă al utilizatorului este sincronizarea contului din aplicaţie cu contul de pe o platformă de găzduire a codului, bazată pe git. Cele mai cunoscute dintre acestea sunt platformele github.com şi bitbucket.org, care vor fi prezentate pe larg în secţiunea „Fundamentare teoretică”. Realizarea acestei etape de sincronizare a conturilor se va realiza prin intermediul protocolului Oauth 1.0, respectiv 2.0, care la rândul lor vor fi prezentate pe larg în secţiunile următoare. Acest protocol facilitează accesul la datele şi resursele utilizatorului de pe un anumit sistem, prin accesarea unui server intermediar care face o cerere de acces la acele resurse în numele utilizatorului, astfel că după ce acesta se va autentifica şi îşi va da acceptul, aplicaţia va putea folosi anumite resurse în numele utilizatorului care a făcut cererea.

Sincronizare cont utilizator cu cel de pe o platforma furnizor de cloud

La fel ca şi la platformele de găzduire a codului sursă, este nevoie de sincronizarea contului de utilizator şi cu contul acestuia de pe o platformă care furnizează servicii în cloud. Această fază este necesară pentru a putea genera serverele pe care se vor lansa aplicaţiile utilizatorului, folosind un API, pus la dispoziţie de către platforma care ofera serviciile în cloud. În cazul în care utilizatorul nu are încă un astfel de cont serverul la care se va face cererea de autentificare îl va redirecţiona spre o pagină de creeare a unui astfel de cont.Pentru a nu ridica complexitatea dezvoltării acestei componente am hotărât să folosesc un singur furnizor de cloud, pentru a putea ilustra uşor capabilităţile sistemului. Tratarea unei game mai largi de furnizori ar putea fi utilă pentru a acoperi preferinţele utilizatorilor şi pentru a nu-i complica pe aceştia cu creerea de conturi pe alte platforme furnizoare de cloud, dacă deja deţin un cont pe o astfel de platformă. Pentru eleganţa si flexibilitatea prin care se impune am ales ca şi furnizor de cloud, digitalocean.com, pe care îl voi prezenta în detaliu în următoarele secţiuni şi care prin API-ul pe care îl pune la dispoziţie, constituie o unealtă foarte utilă pentru realizarea obiectivelor propuse.

Realizarea etapei de update pentru o aplicaţie deja existentă

Un alt obiectiv important al aplicaţiei este asigurarea etapei de mentenanţă a proiectului, astfel încât utilizatorul să aibă capabilitatea de a actualiza versiunea care rulează pe server cu o versiune mai nouă, necesitate izvorâtă din tendinţa aplicaţiilor de a ţine pasul cu noile tehnologii care le obligă să fie în continuă dezvoltare. Astfel că o aplicaţie lansată de către utilizator va avea disponibilă opţiunea de actualizare la o noua

versiune. Această funcţionalitate va fi asigurată în mare parte de către sistemul de control al versiunilor, numit git, pe care îl vom discuta în detaliu în capitolele următoare.

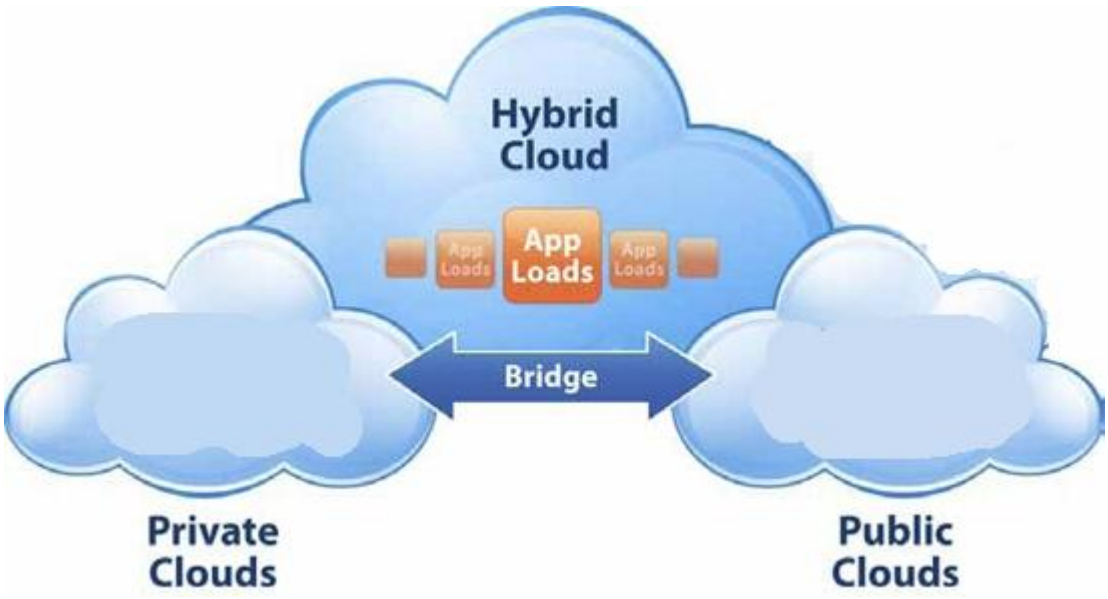
Resetarea fizică a serverelor la cererea utilizatorului

O altă componentă importantă a etapei de mentenanţă este asigurarea utilizatorului ca serverul pe care rulează aplicaţia sa este în permanenţă în stare de funcţionare şi în cazul apariţiei unor probleme(pene de curent temporare sau alte evenimente neasteptate) sau supraîncarcare din cauza traficului prea mare, serverul să poată fi repornit la cererea utilizatorului, astfel acesta îşi poate supraveghea aplicaţia după ce a lansat-o şi să se asigure de buna ei funcţionare. Aceastăfuncţionalitate se va asigura prin utilizarea API-ului pe care-l ofera furnizorul de cloud pe care l-am ales, Digitalocean. Vom analiza în amănunt şi acest aspect în capitolele următoare.

3. Studiu bibliografic

3.1 Conceptul de cloud

Conceptul de cloud este un serviciu care oferă spre închiriere utilizatorilor, resurse hardware şi software. Cum majoritatea utilizatorilor internetului au o conexiune permanentă şi rapidă, a apărut posibilitatea partajării resurselor între utilizatori şi accesarea lor prin internet doar atunci când este necesar. Această partajare între utilizatori se realizează fără ca aceştia să realizeze că se întâmplă acest lucru fizic, deoarece ei nu au acces direct la resursele fizice şi asta se poate face cu ajutorul tehnicii de virtualizare a resurselor. Clasificarea serviciilor de cloud se face după două criterii: după implementare şi după livrare. După implementare serviciile de cloud se împart în 3 categorii: private, publice şi hibride[2].



Figură 3.1 – Tipuri de cloud după implementare

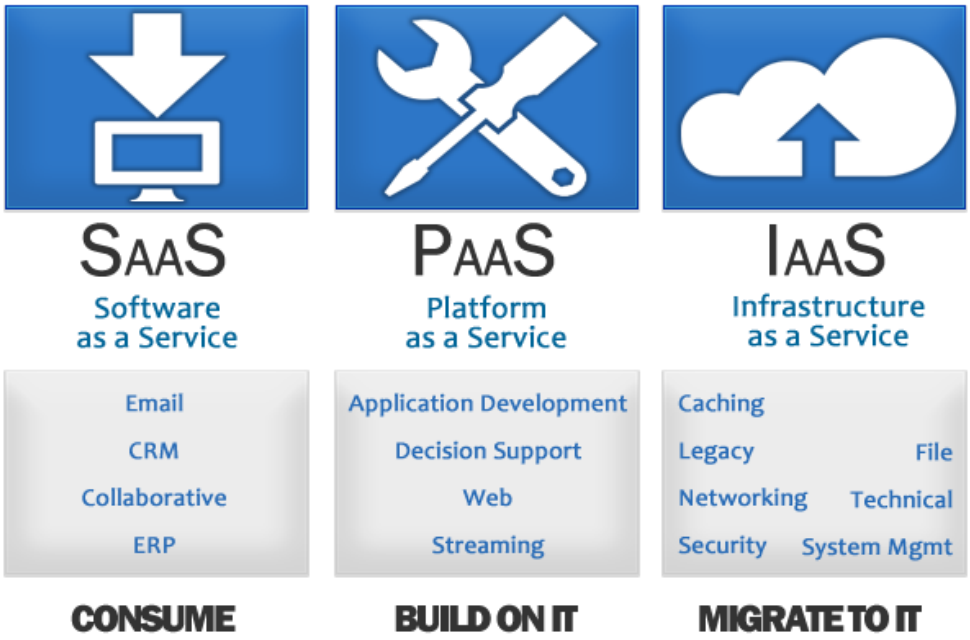
Serviciile de cloud private reprezintă o infrastructură a unei dedicată unei organizaţii individuale, fiind localizată la sediul acesteia sau este gestionat de o altă organizaţie în strictă legatură cu organizaţia beneficiar.

Serviciile de cloud publice reprezintă infrastructuri deţinute de un furnizor specializat în furnizarea de cloud, care pune la dispoziție anumite resurse hardware sau software pe care utilizatorii le folosesc în mod partajat. Serviciile pot fi accesate prin intermediul internetului, fapt care implică transferarea operațiunilor de prelucrare a datelor sau a datelor către sistemele furnizorului.Împreună cu datele, un utilizator este obligat să transfere o mare parte din controlul sau asupra datelor respective. Astfel furnizorului de cloud îi revine rolul de a asigura securitatea datelor pe care le incredinţeaza utilizatorii.

Pe lânga mediile de cloud publice şi private, există mediile de cloud intermediare sau hibride în care serviciile furnizate de infrastructurile private coexistă cu serviciile

achiziţionate din mediile publice. Aceasta este infrastructura de cloud cea mai des întâlnită.

În ceea ce priveşte clasificarea serviciilor de cloud dupa livrare, în funcţie de necesităţile utilizatorilor furnizorii de cloud oferă mai multe tipuri de servicii, acestea putându-se clasifica în trei mai categorii de modele de servicii: Infrastructura ca serviciu(IaaS – Infrastructure as a Service), Platforma ca serviciu(PaaS – Platform as a Service) şi Aplicaţia ca serviciu (SaaS – Software as a Service)[3].



Figură 3.2 – Tipuri de cloud dupa livrare

Infrastructura ca serviciu (IaaS) este tipul de serviciu cloud în care un furnizor închiriază o infrastructura tehnologică şi anume servere virtuale la distanţa pe care utilizatorul final se poate baza în conformitate cu mecanisme şi măsuri astfel încât să facă simplă, eficientăprecum şi avantajoasă opţiunea de a înlocui sistemele corporatiste de tehnologia informaţiei de la sediul întreprinderii si sa utilizeze infrastructura inchiriata impreuna cu sistemele corporatiste. Furnizorii sunt, de obicei, actori specializati care opereaza pe piata si se pot baza, de altfel, pe o infrastructura fizica complexa care cuprinde deseori mai multe zone geografice.

Platforma ca serviciu (PaaS) reprezinta clasa de servicii cloud în care un furnizor ofera soluţii pentru dezvoltarea şi găzduirea avansată de aplicaţii. Serviciile se adresează, utilizatorilor care dezvoltă şi găzduiesc soluţii bazate pe aplicaţii proprietar pentru a îndeplini cerinţe stabilite la nivel intern şi pentru a furniza servicii către terţi. Din nou, serviciile furnizate de către un furnizor PaaS scutesc utilizatorul de adăugarea de componente hardware adiţionale. Un bun exemplu de astfel de serviciu cloud este Google Application Engine (GAE), care oferă posibilitatea dezvoltării de aplicaţii cu infrastructură necesară stocată pe serverele Google.

Aplicaţia ca serviciu (SaaS) este modelul de cloud în care un furnizor oferă, prin intermediul internetului, diferite servicii de aplicaţii pe care le pune la dispoziţia

utilizatorilor finali. Aceste serviciiau menirea de a înlocui aplicaţiile convenţionale instalate de utilizatori în sistemele lor locale, astfel aceştia fiind obligaţi să-şi externalizeze datele către furnizorul individual. Acest lucru este valabil, de exemplu, în cazul aplicațiilor de birou tipice bazate pe internet precum fisiere, instrumente de editare a textelor, registre si agende computerizate, calendare partajate sau aplicații e-mail bazate pe medii de cloud computing. Cel mai bun exemplu de astfel model de servicii cloud îl reprezintă gama de servicii oferite de google: email, disc pentru stocare (drive), calendar, hărţi.

Principalele caracteristici ale conceptului de cloud sunt următoarele: partajarea resurselor, grad înalt de scalabilitate, elasticitate, faptul că utilizatorul plăteşte pe oră şi doar pe resursele pe care le consumă şi faptul că utilizatorul îşi gestionează singur resursele oferite de către furnizorul de cloud. [4]

Partajarea resurselor este un avantaj demn de luat în seamă pe care îl oferă serviciile de cloud, întrucât se lasă în urma acel model rigid de a aloca resurse dedicate fiecărui utilizator, model care folosea complet ineficient aceste resurse.Îngrijorarea cu privire la securitatea unui astfel de sistem este rezolvată de către platformele hibrid de servicii cloud.

Figură 3.3 – Partajarea resurselor

Resursele cruciale pe care le folosesc utilizatorii sunt: numarul de procesoare, capacitate de stocare, lăţimea de bandă. În sistemele bazate pe cloud acestea au un grad înalt de scalabilitate, astfel încât pot fi ajustate în orice moment de timp de către utilizator, în funcţie de nevoile acestuia.

Datorită faptului căutilizatorii pot să crească sau să descrească rapid cantitatea de resurse alocată activităţilor lor sistemul trebuie să realoce resursele eliberate sau să aloce resursele cerute din cele pe care le-au eliberat în prealabil alţi utilizatori. Această proprietate a serviciului de cloud se numeşte elasticitate şi este foarte importantă pentru platformele furnizoare de cloud, deoarece astfel se alocă eficient resursele disponibile contribuind la rentabilitatea acestor furnizori de servicii cloud.

Componenta economică, cu un impact important şi asupra utilizatorilor este plata pe care o fac aceştia pentru serviciile care le sunt furnizate. Astfel că utilizatorii plătesc pe oră şi plătesc doar pentru serviciile pe care le consumă, astfel putând profita de celelalte caracteristici ale serviciilor de cloud, extinzându-şi şi diminuându-şi resursele după bunul plac în orice moment de timp. Astfel se lasă în urma serviciile vechi de

hosting, în care utilizatorul îşi alegea încă de la început resursele şi să plătească lunar pentru acestea, fiind obligat să-şi menţină resursele, chiar dacă ar fi constatat că are nevoie de o modificare în ceea ce priveşte cantitatea acestora. Această constrângere era foarte apăsătoare pentru dezvoltatorii de aplicaţii care nu putea estima de la început de ce resurse au nevoie sau nu aveau nevoie de un server intreg dedicat.

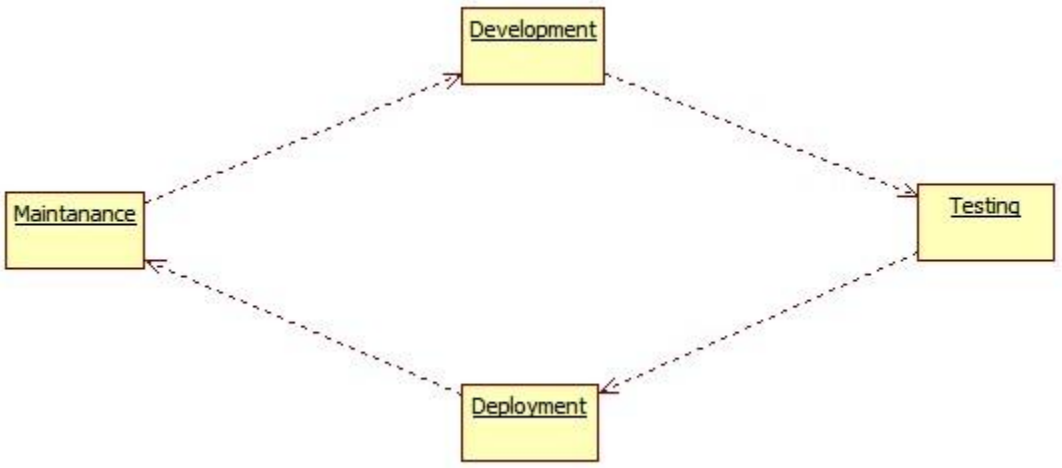
Ultima caracteristică a serviciilor cloud este aceea că beneficiarii îşi gestionează singuri resursele, adaugând şi reducând din acestea atunci când consideră că este necesar. Acest aspect este la rândul său unul foarte important deoarece serviciile de consultanţă pe care trebuie sa le ofere furnizorii sunt simplificate sau chiar reduse total, rezolvând această problemă prin punerea la dispoziţie a unei interfeţe sau a unui API bine construit şi intuitiv.

Cloud computing-ul este rezultatul a decenii de cercetare în virtualizare, într-o serie de domenii concurente: sisteme distribuite, retelistică, web şi aplicaţii software. Implică o arhitectura orientată pe servicii, un numar mic de cunoştinţe tehnologice pe care trebuie să le aibă utilizatorul final al aplicaţiei. De asemenea oferă scalabilitate şi flexibilitate mare şi un cost redus de distribuţie a serviciilor.

Modelul de cloud care va fi folosit în lucrarea de faţă este infrastructura ca serviciu (IaaS – Infrastructure as a Service), pentru a oferi utilizatorilor posibilitatea de a lansa propriile aplicaţii pe o infrastructura flexibilă şi scalabila oferită de către un furnizor de cloud. Furnizorii de cloud care oferă astfel de servicii sunt numeroşi, dar cei mai cunoscuţi sunt următorii: Amazon Web Service (liderul mondial), Microsoft Azure, DigitalOcean, Vmware, Joyent. Pentru simplitatea, multitudinea de posibilităţi pe care o oferă API-ul pe care îl pun la dispoziţie şi costurile reduse pe care le oferă, dar şi din motive de cunoaştere şi familiarizare cu platforma am ales să folosesc serviciile de cloud oferite de către DigitalOcean, unul din furnizorii de cloud aparuţi destul de recent dar care a avut o crestere exponenţială în rândul utilizatorilor. Se va discuta în detaliu această platformă în capitolul următor, în secţiunea de prezentare a tehnologiilor. [5]

3.2 Conceptul de lansare a unei aplicații web

În ciclul de viaţă a unei aplicaţii web, una din etapele importante este etapa de lansare, deoarece prin intermediul acestei etape, dezvoltatorii îşi publică aplicaţia, în aşteptarea rezultatelor pe care urmează să le aducă în faza de producţie.Deseori această fază de lansare poate părea o povară, din pricina instalărilor şi configurărilor de programe pe care le presupune.



Figură 3.4 – Ciclul de viață a unei aplicații web

Lucrarea de faţă îşi propune să realizeze un sistem care să simplifice această etapă, prin construirea unei aplicaţii web care să realizeze această etapă în mod automat, în funcţie de preferinţele utilizatorului. Pe lângă etapa de lansare, sistemul va avea un rol şi în etapa de mentenanţă a proiectelor, punând la dispoziţia utilizatorilor o opţiune de update la cea mai noua versiune a aplicaţiei.

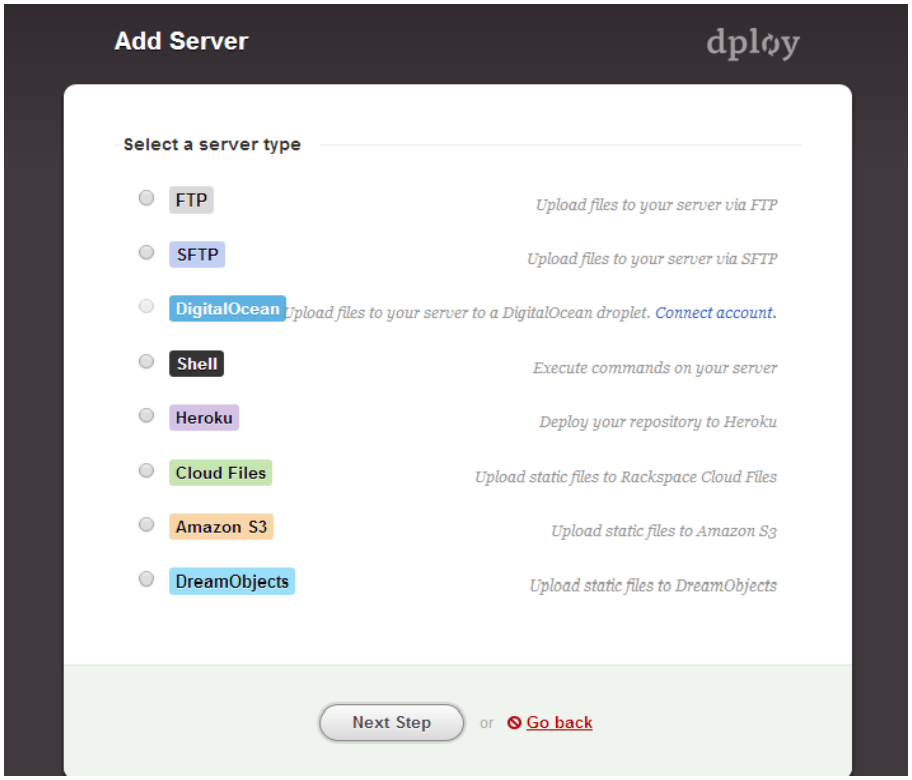
3.3 Starea actuală a domeniului

Deoarece acest aspect este foarte important pentru dezvoltatorii de aplicaţii web, în ultima perioada mai mulţi dezvoltatori au fost preocupaţi de această problemă şi în ultimii ani au apărut o serie de astfel de sisteme care să rezolve etapa de lansare a unei aplicaţii în mod automat, contribuind serios la simplificarea procesului de dezvoltare software. Deşi numărul aplicaţiilor de acest tip a început să crească în ultimii ani, totuşi aceste sisteme par încă insuficient de mature ca să rezolve problemele utilizatorilor deşi constituie un real avantaj pentru aceştia. Principala problemă e că nici una din aceste aplicaţii nu întruneşte preferinţele tututor utilizatorilor (cazul ideal), iar cele care se apropie de acest tel, sunt destul de scumpe pentru dezvoltatorii de aplicaţii mici. Astfel, îmi propun să realizez un sistem care să se apropie cât mai mult de satistfacerea unui număr cât mai mare de utilizatori, dar care să fie accesibil şi dezvoltatorilor de aplicaţii mici. Pentru început sistemul va suporta lansarea aplicaţiilor web dezvoltate pe platforma wordpress şi cele dezvoltate pe frameworkul Laravel din limbajul php. Acestea vor fi discutate în detaliu în sectiunea de protocoale şi tehnologii.

3.4 Sisteme similare

1. dploy.io - http://dploy.io/

Reprezinta un sistem foarte bun de deploy pentru aplicaţii web, deoarece oferă utilizatorilor posibilitatea de a-şi sincroniza contul cu sistemele de control a versiunii bazate pe git. Cele mai importante sisteme de acest tip sunt bitbucket şi github. Dploy.io realizează faza de deploy folosind containerele de cod pe care le deţine utilizatorul pe contul ales de pe unul din cele două sisteme şi comenzile specifice sistemului de control al versiunii numit git.



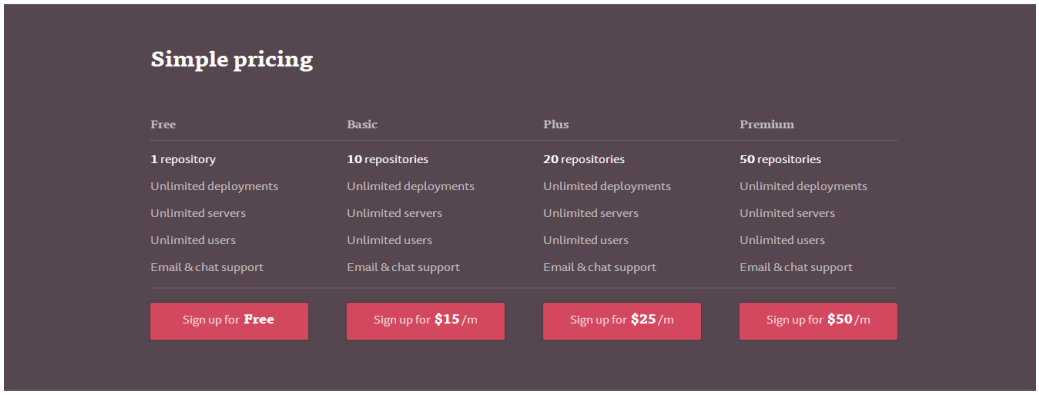
Figură 3.5 – Adăugare de server la dploy.io

Partea de configurare a serverelor se realizează într-o serie de formulare, zona în care dploy.io permite integrarea cu furnizori importanţi de cloud Amazon S3, Digitalocean, dar şi realizarea fazei de deploy prin upload de fişiere prin protocoalele ftp, sftp cât şi prin intermediul liniei de comandă, reuşind astfel să satisfacă nevoile unei game foarte largi de utilizatori oferind posibilitatea de a configura automat serverele pentru programatorii care nu stăpânesc partea de comenzi, dar si control utilizatorilor care cunosc deja metodele traditionale de transfer într-un mediu de producţie a unei aplicaţii.

Deoarece este un sistem foarte complex, deploy.io oferă o parte de interfaţă cu utilizatorul destul de încarcată, greu de înteles, unde acesta trebuie să introducă foarte multe date şi în unele secţiuni nu este foarte clar ceea ce se doreşte de la utilizator, astfel ca dploy.io poate fi considerat puţin greoi ca şi experienţa utilizator, dar aceasta nu

constituie neaparat o problemă deoarce majoritatea clienţilor acestei platforme sunt specialişti în IT. Pe lângă oportunitatea de a realiza faza de lansare a aplicaţiei web, dploy.io oferă şi posibilitatea de a realiza faza de mentenanţă a sistemului, de a actualiza aplicaţia la versiuni noi cu ajutorul sistemului de control al versiunii numit git.

Pe lângă acestea, dploy.io oferă posibilitatea de a sincroniza conturile şi cu sisteme de comunicare între echipe de dezvoltatori, cum ar fi Hipchat, Campfire prin intermediul cărora orice operaţie realizată în cadrul sistemului este adusă la cunoştinţa întregii echipe prin notificări.

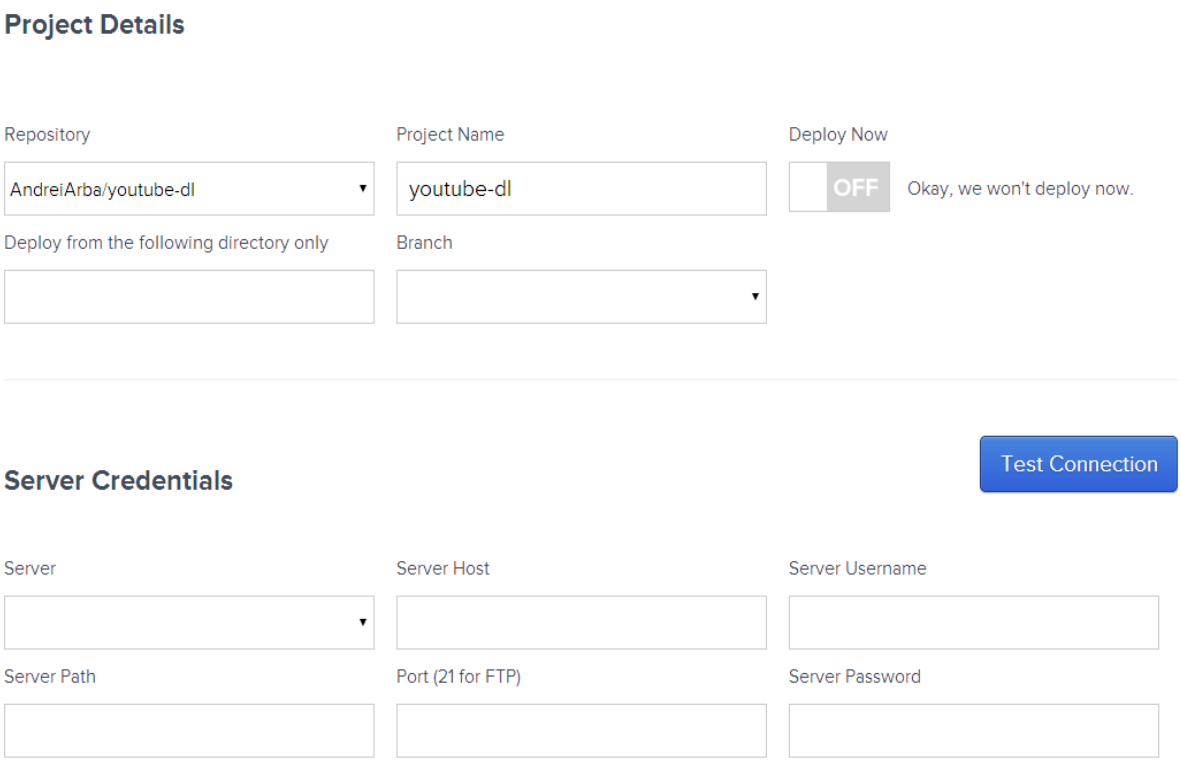


Figură 3.6 – Preturi la dploy.io

O altă latură importantă a unui astfel de sistem de deploy, o reprezintă preţul, care în cazul sistemului dploy.io este destul de costisitor, astfel că un pachet pentru mai mult de un proiect costă minim 15$ pe lună, un preţ accesibil totuşi, dacă ne gândim la multitudinea de posibilităţi pe care le oferă sistemul, dar dezavantajos pentru utilizatorii care au proiecte mici şi nu ar avea nevoie de partea de mentenanţă ci doar de lansarea aplicaţiei pe server şi realizarea operaţiei de reboot a serverelor atunci când apar defecţiuni. Insumând toate acestea, putem afirma cu certitudine că dploy.io este un sistem foarte bun de deploy automat a aplicaţiilor web, oferind o gamă foarte largă de posibilităţi în alegerea serverelor şi integrare atât cu sistemele bazate pe git cât şi cu sisteme de comunicare.

2. ftploy.com - http://ftploy.com/

Reprezintă un sistem de deploy pentru aplicaţii web care permite utilizatorilor atât integrarea cu un cont pe unul sistemele de control a versiunii bazate pe git, github sau bitbucket cât şi utilizarea unui cod structurat ca o ierarhie de directoare. Ftploy utilizează comenzi de git pentru a realiza operaţia de deploy pe un server. Pentru a sincroniza contul de utilizator de pe site cu cele de pe sistemele bazate pe git sistemul foloseşte api-ul disponibil cu un request de autorizare, în cazul platformei github.com şi prin adăugarea de cheie publică ssh în contul care urmează a fi folosit pentru deploy, în cazul platformei bitbucket.org. Această operaţie poate fi destul de dificilă pentru un utilizator neexperimentat, fapt pentru care site-ul îl îndrumă în a face acest pas, prin redirectarea pe pagina de chei ssh a plaformei bitbucket.

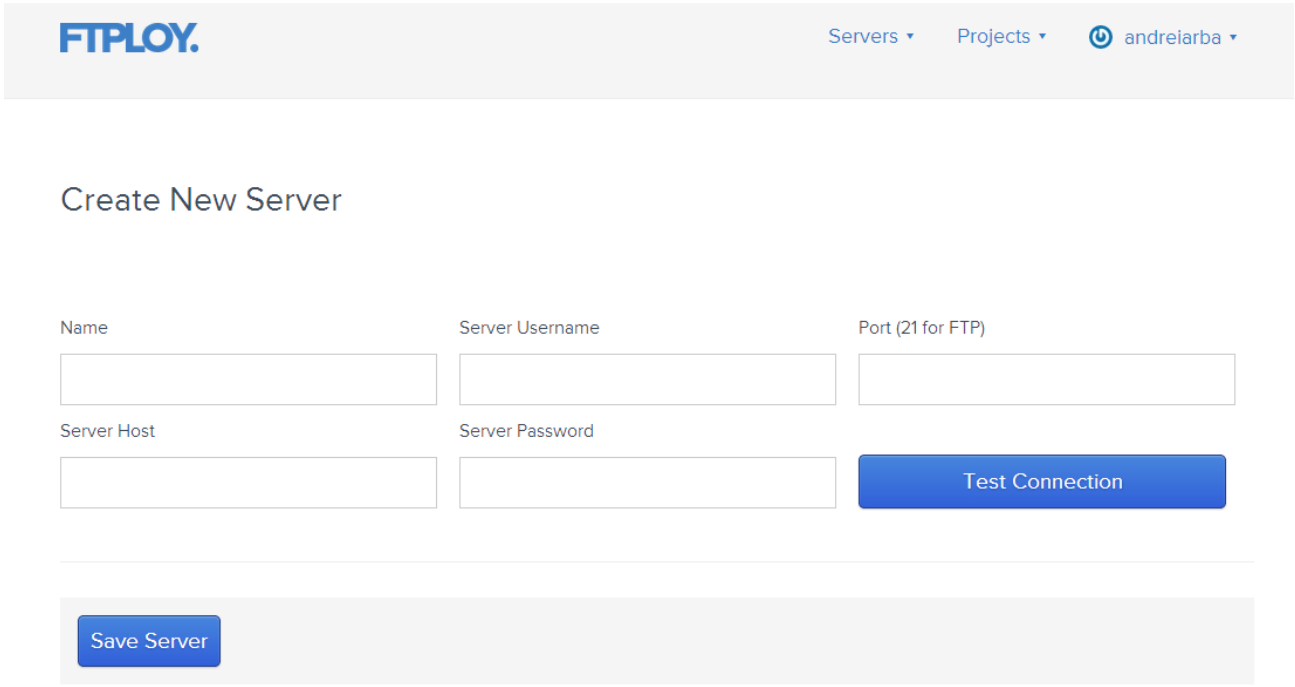


Figură 3.7 – Adăugarea de proiect la ftploy

Sistemul oferă şi partea de mentenanţă a proiectelor lansate prin intermediul său, prin posibilitatea de a actualiza aplicaţia prin utilizarea tehnicii de versionare oferita de sistemele bazate pe git.

Preţul cerut de sistem pare destul de mare faţă de ceea ce oferă, pentru mai mult de un proiect, costa aproximativ 15$ pe lună.

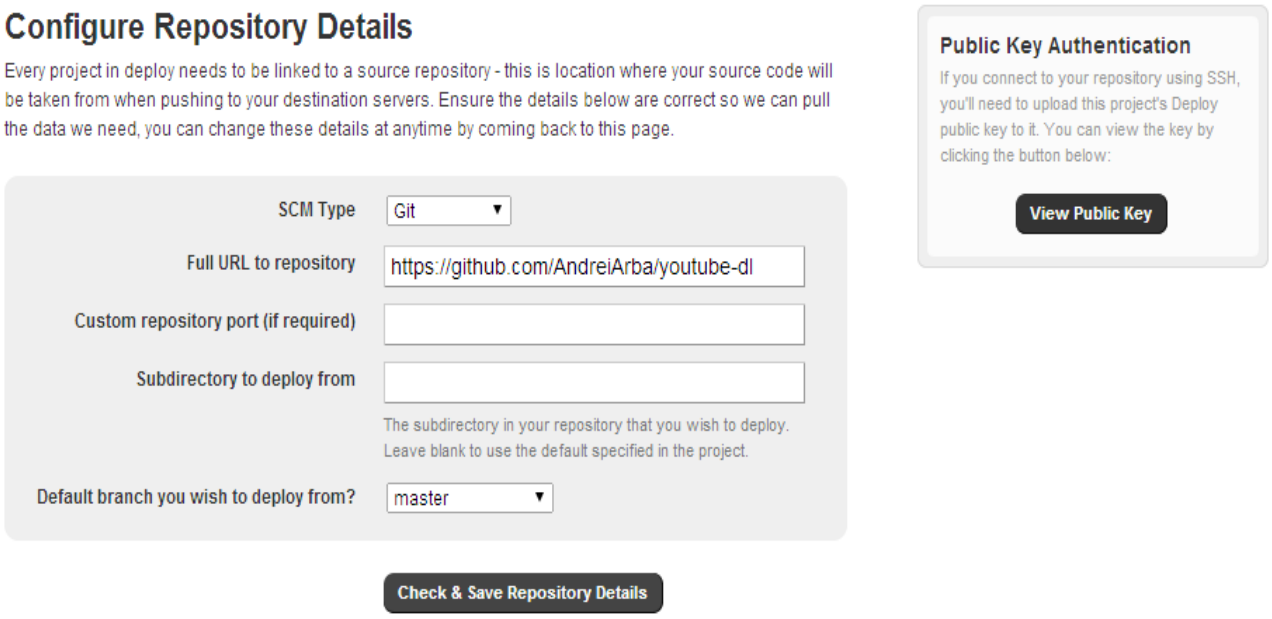
Principala problema a acestui sistem de deploy este aceea că utilizatorul trebuie să deţină deja serverul pe care urmează să facă deploy, ftploy.com nu oferă integrarea serviciilor din cloud, ceea ce constituie o problemă destul de mare, deoarece utilizatorului îi revine o misiune destul de dificilă, aceea de a crea un server şi de a face configurările necesare pentru a putea rula o aplicaţie web. Astfel, deşi operaţia de deploy în sine se realizează foarte uşor cu ajutorul acestui sistem, problema configurărilor şi creerii de servere nu este rezolvată, ceea ce, în opinia mea face ca sistemul să nu poată fi considerat un sistem de deploy complet, întrucât constituie principala problemă a acestei faze a dezvoltării unei aplicaţii web.



Figură 3.8 – Adăugarea de server la ftploy

3. deployhq - https://www.deployhq.com/

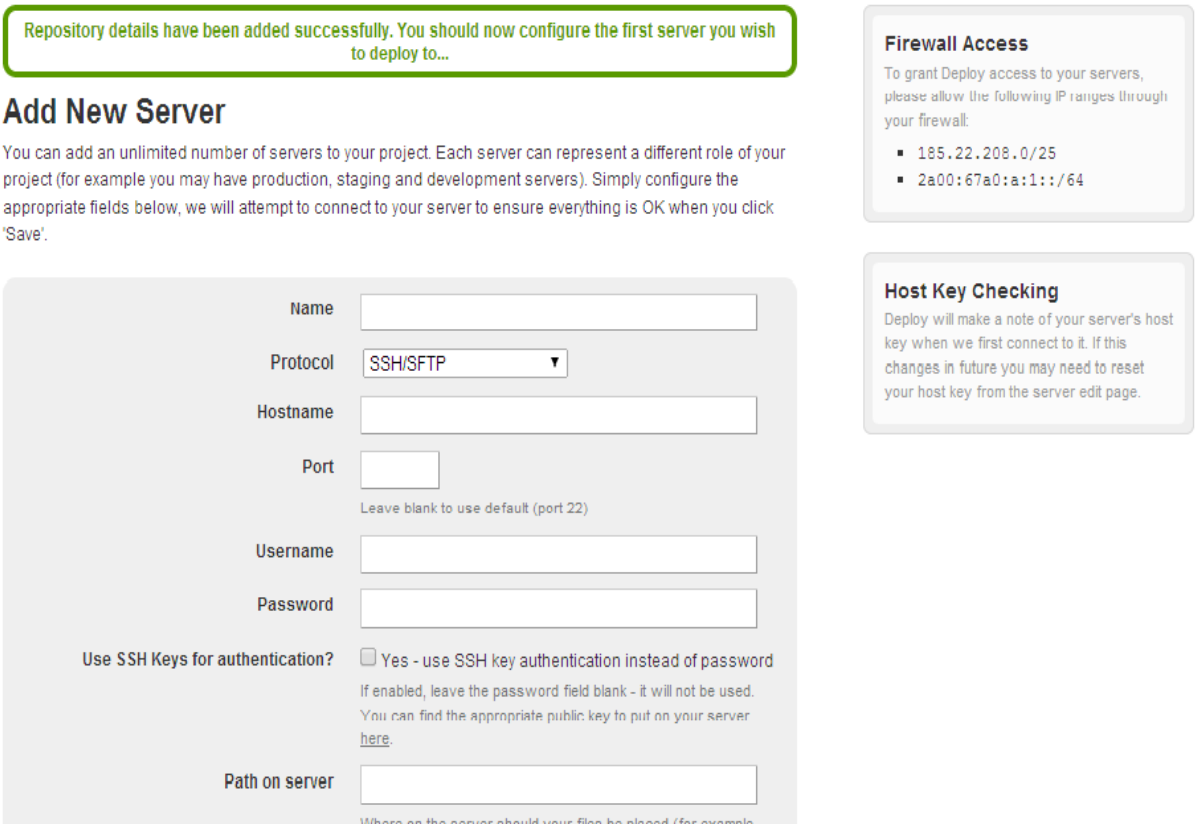
Reprezinta un alt sistem de deploy pentru aplicaţii web destul de bine pus la punct, care permite utilizatorilor să lanseze atât aplicaţii stocate local, ca structură de directoare cât şi aplicaţii stocate în containere pe sisteme bazate pe git. Deployhq utilizează setul de comenzi puse la dispoziţia de către git pentru a realiza operaţia de deploy pe un server, cât şi mentenanţa sistemului lansat, prin actualizarea versiunilor, funcţionalitate oferita tot de către git. Pentru a sincroniza contul de utilizator de pe site cu cele de pe sistemele bazate pe git sistemul foloseşte api-ul disponibil cu un request de autorizare, în cazul platformei github.com şi prin adaugarea de cheie publică ssh în contul care urmează a fi folosit pentru deploy, în cazul platformei bitbucket.org.



Figură 3.9 – Configurare deployhq

La fel ca şi la ftploy, preţul cerut de sistem pare destul de mare faţă de ceea ce oferă, pentru mai mult de un proiect, costa aproximativ 15$ pe lună.

În mare parte, deployhq oferă cam aceleaşi funcţionalităţi ca şi sistemul analizat în secţiunea precedentă, rămânând cu marea problemă că utilizatorul trebuie să deţină deja serverul pe care urmează să facă deploy, nici deployhq nu oferă integrarea serviciilor din cloud, ceea ce conduce la principala problemă a fazei de lansare a unei aplicaţii web, aceea de a crea un server şi de a face configurările necesare pentru a putea rula o aplicaţie web. Pe lângăaceasta, utilizatorul trebuie să şi dispună de un server dedicat pe care să-şi lanseze aplicaţia, iar separarea fazei de lansare de cea de găzduire pare a conduce la sisteme care să fie depăşite clar de către cele care oferă şi găzduire în cloud, prin utilizarea unor api-uri pe care principalii furnizori de cloud îl oferă dezvoltatorilor, iar aici ne referim în primul rând la Amazon S3 şi la DigitalOcean, ale căror api-uri sunt disponibile pentru a facilita activitatea dezvoltatorilor.



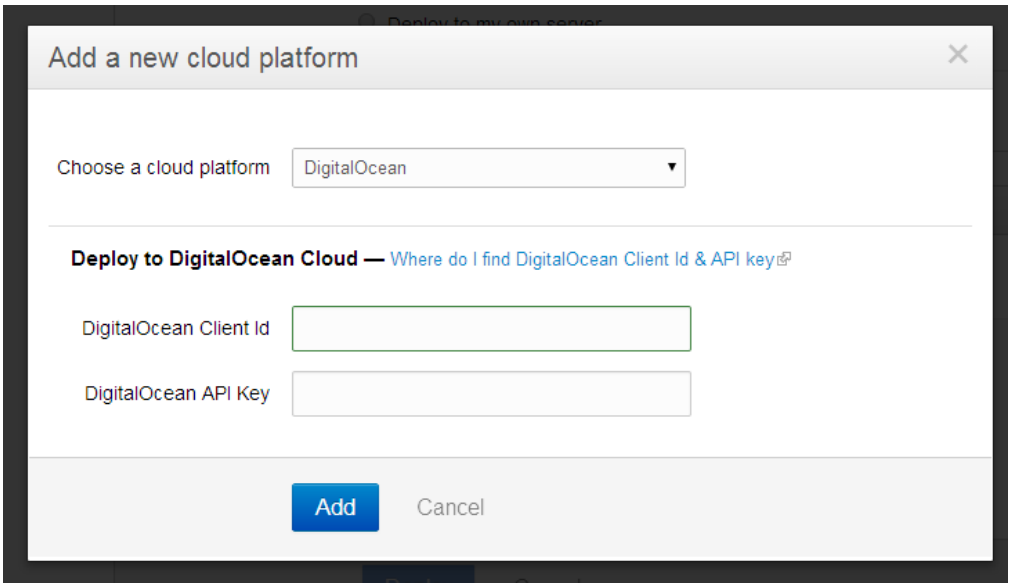
Figură 3.10 – Creere server la deployhq

Astfel, deşi în ceea ce ţine de operaţia de deploy efectiv deployqh o realizează foarte uşor, în doar câţiva paşi, problema configurărilor şi creerii de servere nu este rezolvată, ceea ce în opinia mea face ca sistemul să nu poată fi considerat un sistem de deploy complet, întrucât constituie principala problemă a acestei faze a dezvoltării unei aplicaţii web.

4. cloud66 - https://app.cloud66.com/

Reprezintă un sistem foarte bun de deploy pentru aplicaţii web, deoarece pe lângă posibilitatea de a-şi sincroniza contul cu sistemele de control a versiunii bazate pe git oferă utilizatorilor şi serviciile principalilor furnizori de cloud: Amazon AWS, DigitalOcean, Joyent, Rackspace. Pe lângă toate acestea sistemul oferă şi posibilitatea de a sincroniza contul cu alte sisteme folosite pentru comunicare cum ar fi Hipchat şi ţine la curent toată echipa cu modificările asupra unui proiect lansat, prin notificări.

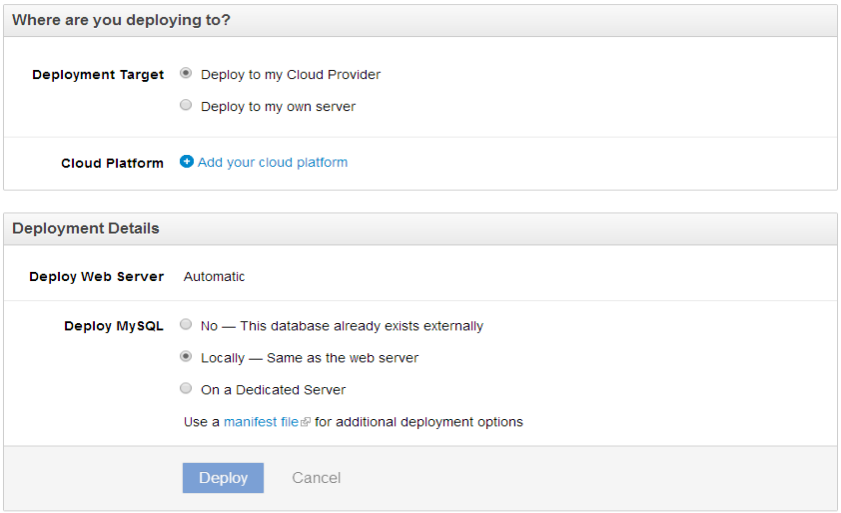
Utilizatorul îşi poate alege şi dacă serverul de baze de date, va fi pe acelaşi server ca şi aplicaţia sau pe unul dedicat, aceasta fiind o funcţionalitate importantă pentru aplicaţiile care rulează pe servere multiple. De asemenea faza de mentenanta a sistemului este oferită prin intermediul controlului versiunii de către git, actualizarea aplicaţiei se face prin intermediul contului pe unul din sistemele bazate pe git cu care este sincronizat contul curent al utilizatorului.



Figură 3.11 – Adăugare de platformă cloud cloud66

Preţul sistemului este unul foarte bun, 19$ pentru primul server pe lună, iar apoi se platesc 9$ pe lună pentru fiecare din următoarele servere. De subliniat ar fi şi faptul că se plăteşte pe oră, astfel că utilizatorii plătesc doar cât consumă, putând astfel profita de beneficiile pe care le oferă furnizorii de cloud, utile în special pentru adaptarea planului tarifar în funcţie de nivelul de trafic de pe site, prin reducerea şi cresterea numărului sau capacităţii serverelor atunci când este necesar.

Toate caracteristicile descrise mai sus, relevă faptul că sistemul numit cloud66 este un sistem complet de deploy al aplicaţiilor web, putem afirma cu certitudine că preţul cerut este unul meritat, oferind o gamă foarte largă de posibilităţi în alegerea serverelor şi integrare atât cu sistemele bazate pe git cât şi cu sisteme de comunicare.



Figură 3.12 – Lansare pentru cloud66

În continuare se va face o analiză comparativă între sistemele descrise mai sus, pe baza principalelor caracteristici pe care ar trebui sa le îndeplinească un sistem de deploy automat al aplicaţiilor web, analiza în care voi include şi sistemul numit Gitdone, care este ţinta finală a acestei lucrări.

Analiză comparativă:

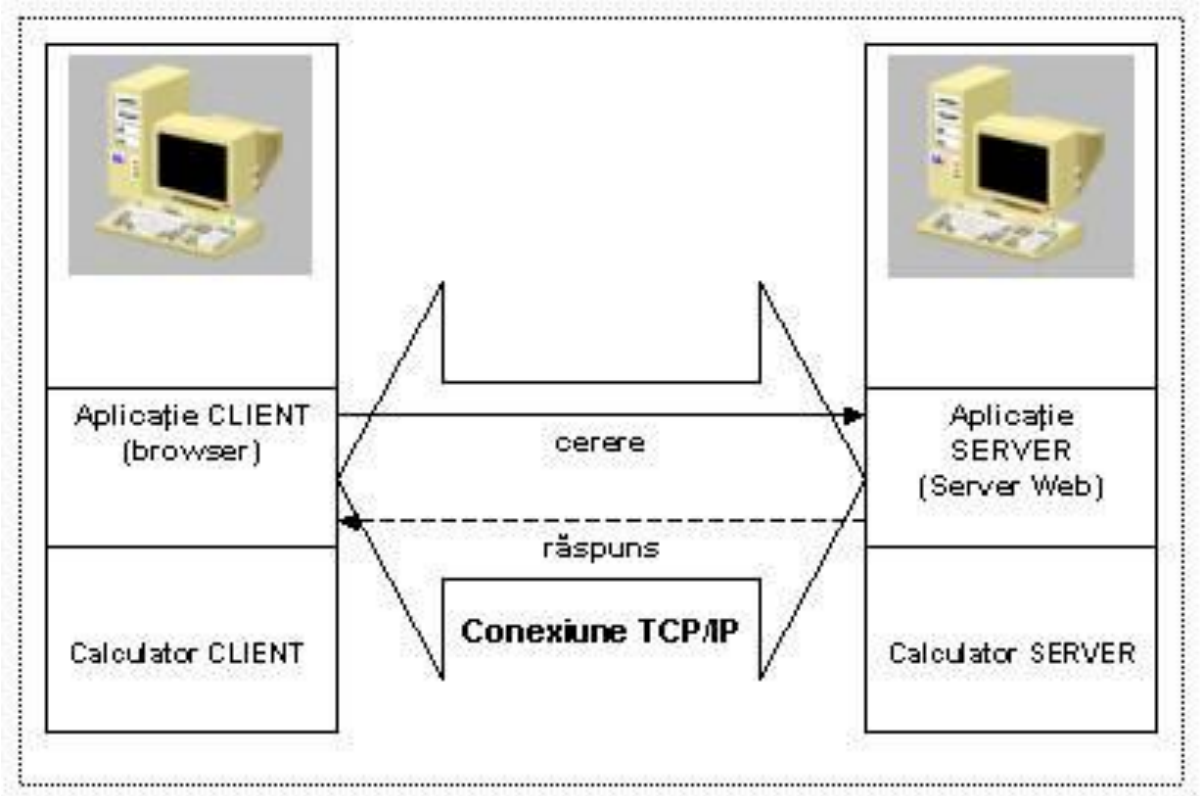
Caracteristica	Dploy	Ftploy	Deployhq	Cloud66	***Gitdone***
Integrare cu sisteme bazate pe git	X	X	X	X	X
Integrare cu furnizori de cloud	X	-	X	-	X
Asigurarea mentenanței	X	X	X	-	X
Integrare cu sisteme de comunicare	X	-	-	X	-
Alegerea locației serverului de baze de date	-	-	-	X	-
Reboot servere cloud	-	-	-	-	X
Preț	15$/proiect	15$/proiect	15$/proiect	19$/proiect	-

Tabel 3.1 – Analiza comparativă între sistemele similare

4.Analiză şi fundamentare teoretică

4.1 Arhitectura conceptuala

Sistemul este o aplicaţie web unde partea de back-end este realizată în limbajul php, folosind framework-ul Laravel 4.0, framework care impune aplicaţiei o arhitectură de model-view-controller, aducând câteva plusuri acestei arhitecturi, iar pe partea de front-end se vor folosi tehnologiile specifice HTML, CSS şi Javascript, împreună cu librăria Twitter Bootstrap pentru partea de pagini. Această aplicaţie, va fi accesată de către clienţi printr-un browser web, comunicarea dintre client şi aplicaţia server realizându-se prin intermediul protocolului HTTP. Toate aceste tehnologii şi protocoale vor fi prezentate în detaliu în acest capitol, în secţiunea de protocoale şi tehnologii utilizate. Pentru partea de persistenţă a datelor se va folosi o bază de date care va fi gestionată prin intermediul sistemului de gestiune al bazei de date relaționale mysql, a cărei structură va fi descrisă pe larg în capitolul următor.



Figură 4.1 – interacţiunea Client - Server

În figura de mai sus este descrisă interactiunea clientului care este reprezentat de un browser web cu partea de server care este aplicaţia web care este scopul lucrării de faţă. Astfel în zona de adrese a browserului se introduce numele domeniului, iar cu ajutorul serviciului de DNS, se localizează adresa ip a serverului, cu care browserul iniţiază o conexiune TCP, prin intermediul căreia se va desfăşura comunicarea. Şi aceste concepte vor fi detaliate în acest capitol, în secţiune de protocoale şi tehnologii.

4.2 Cerinţe

Cerinţele aplicaţiei reprezintă aşteptările pe care le avem de la aceastaîin faza de proiectareşi funcţionalităţile pe care trebuie să le îndeplinească pentru a deservi în final utilizatorii. Stabilirea acestor cerinţe reprezintă o etapă importantădin dezvoltarea acestui sistem şi a produselor software în general. În următoarea secţiune vor fi descrise cerinţele funcţionale şi non-funcţionale ale sistemului.

4.2.1 Cerinţe functionale

Cerinţele funcţionale reprezintă funcţionalităţile pe care sistemul trebuie să le îndeplinească, cât şi modul în care acesta trebuie să reacţioneze în diferite situaţii, la anumite intrări sau cereri ale utilizatorilor. Sistemul de lansare a aplicaţiilor web are ca ţintă utilizatori din domeniul tehnologiei informaţiei şi anume dezvoltatori de aplicaţii web, astfel se presupune că ei au anumite noţiuni specifice acestui domeniu şi pe această premisă sunt realizate principalele funcţionalităţi ale aplicaţiei.

	Descrierea cerinţei
CF1	Înregistrarea utilizatorilor
CF 2	Autentificarea utilizatorilor
CF 3	Recuperarea/resetarea de parolă
CF 4	Actualizarea datelor de profil a utilizatorului
CF 5.1	Sincronizarea contului de utilizator cu cel dintr-un sistem bazat pe git (github.com sau bitbucket.org)
CF 5.2	Sincronizarea contului de utilizator cu un cont pe platforma furnizoare de cloud - DigitalOcean
CF 6	Utilizatorul logat să poată să creeze un nou server, folosind un formular al aplicaţiei.
CF 7	Utilizatorul logat să poată să vizualizeze lista aplicaţiilor sale, lansate.
CF 8	Utilizatorul logat sa poată face update la una din aplicaţiile sale deja lansate din listă.

Tabel 4.1 – cerințe funcționale

4.2.2 Cerinţe non-funcţionale

Cerinţele non-functionale sunt cerinţele ataşate celor funcţionale şi nu se referă direct la funcţionalităţi pe care trebuie să le îndeplinească sistemul ci la proprietăţi pe care să le aibă sau constrângeri pe care trebuie să le respecte. Deşi par a ocupa planul secund, faţă de cerinţele funcţionale şi cerintele non-funcţionale sunt demne de luat în seamă, deoarece acestea asigură utilizatorii că aplicaţia se încadrează în anumite cerinţe, unele dintre caracteristicile non-funcţionale reprezentând adevărate unităţi de masură pentru valoarea şi utilitatea aplicaţiei.

Aplicaţia din lucrarea de faţă trebuie să respecte următoarele cerinţe non-funcţionale: disponibilitate, accesibilitate, eficacitate, securitate.

Disponibilitatea este dată de faptul că sistemul va fi lansat pe un server în cloud şi va fi disponibilă utilizatorilor 24 de ore din 24, iar în cazul apariţiei unui eveniment neaşteptat care să blocheze sau săîntrerupă funcţionarea serverului acesta poate fi repornit sau sistemul se poate muta pe un alt server, având deja configurările făcute în mai puţin de 15 minute. Astfel putem spune că sistemul va respecta cerinţa non-funcţională de disponibilitate.

Accesibilitatea este conferită de faptul că este o aplicaţie web şi poate fi accesată prin intermediul oricărui browser, astfel că utilizatorul are nevoie doar de un dispozitiv conectat la internet şi de un browser web instalat pentru a putea accesa aplicaţia.

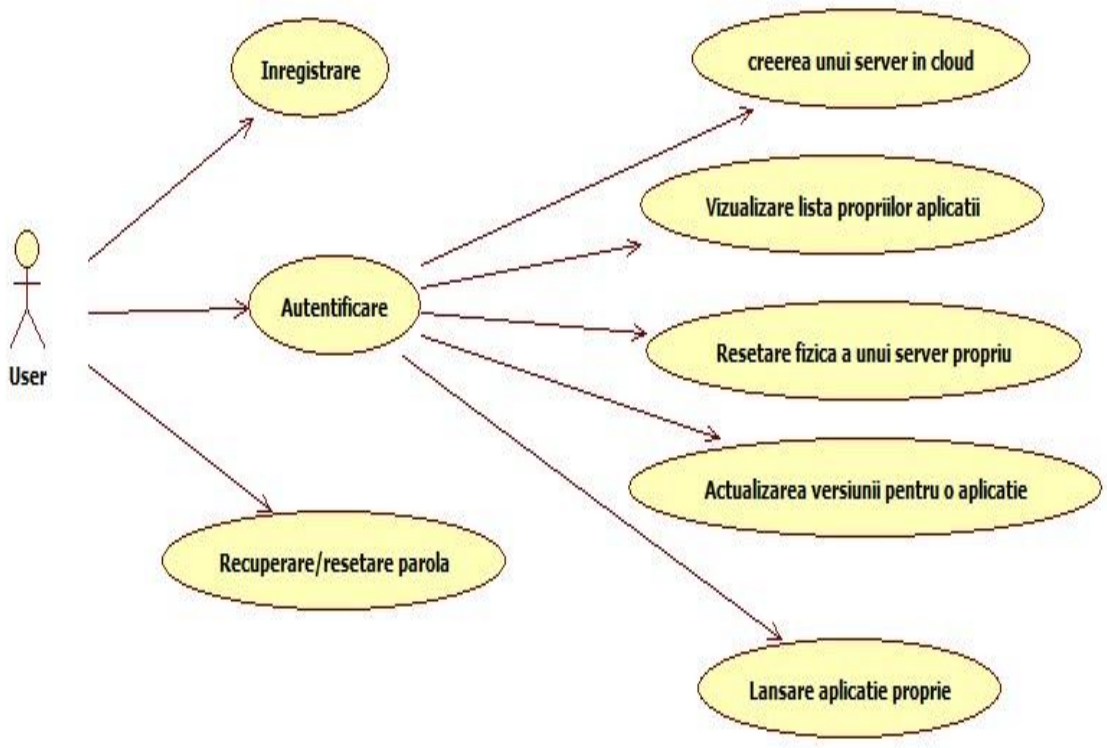
Eficacitatea este o cerinţă non-funcţională pe care o respectă sistemul din lucrarea de faţă, deoarece cu un efort minim din partea utilizatorului şi cu resurse minime consumate se realizează o etapă importantă din ciclul vieţii aplicaţiei sale web, pe care doreşte să o lanseze.

Securitatea sistemului este asigurată de introducerea accesului autorizat în aplicaţie, prin inregistrare şi autentificare şi prin pasarea responsabilităţii de autentificare în cadrul sistemelor bazate pe git şi platformei furnizoare de cloud serverului de Oauth. Un alt aspect care contribuie la asigurarea securităţii este stocarea parolelor criptate cu metoda de criptare Md5, dar şi de accesul la servere prin protocolul de criptare Rsa, bazat pe o pereche de chei publică – privată (vom prezenta pe scurt aceste aspecte în secţiunile următoare). Această metodă de criptare limitează foarte mult încercările de acces forţat la server, întrucât au chei pe 2048 de biti, astfel că timpul şi resursele necesare atacului, depăşesc beneficiile pe care le-ar putea avea atacatorul. Cu ajutorul anumitor funcţionalităţi pe care le oferă framework-ul pe care va fi dezvoltată aplicaţia, se evită şi injecţiile sql sau introducerea de cod HTML în zonele de text din formulare. Astfel putem spune că sistemul din lucrarea de faţă respectăîn mare masură cerinţa non-funcţională de securitate, asigurând un grad de securitate destul de ridicat utilizatorilor aplicaţiei, mai ales luând în considerare faptul că responsabilitatea autentificării spre resursele importante ale utilizatorilor, codul sursă al aplicaţiilor, găzduit pe platforme bazate pe git sau resursele de pe platforma furnizoare de cloud, este pasată serverului de Oauth.

	Descrierea cerinţei
CNF 1	Disponibilitate - disponibil 24/24, timp scurt de recuperare în caz de cădere
CNF 2	Accesibilitate - accesibil doar cu conexiune la internet şi un browser web disponibil
CNF 3	Eficacitate - realizarea lansării unei aplicaţii folosind efort şi resurse minime
CNF 4	Securitate - oferă un grad de securitate suficient de bun

Tabel 4.2 – cerințe non-funcționale

4.3 Cazuri de utilizare



Figură 4.2 – Cazuri de utilizare

Diagrama de mai sus surprinde principalele cazuri de utilizare ale sistemului, reprezentat dupa regulile modelului cazurilor de utilizare definit de procesul unificat pentru dezvoltare a aplicaţiilor software, acesta se focalizează pe determinarea, modelarea şi implementarea cât mai reală a cerinţelor utilizatorilor pe baza cazurilor de utilizare reprezentate în UML. După cum se poate observa singurul actor din diagramă este utilizatorul normal, deoarece utilizatorii nu interacţioneazăîn aplicaţie, ci fiecare se

foloseşte de resursele proprii pentru a lansa şi actualiza aplicaţii nefiind nevoie de un alt tip de utilizator. În continuare se vor prezenta principalele cazuri de utilizare pe larg.

4.3.1 Descrierea detaliată a cazurilor de utilizare

1. Înregistrarea utilizatorului

Descriere: Pentru a putea folosi aplicaţia şi funcţionalităţile pe care le oferă utilizatorii trebuie săîşi înregistreze un cont cu câteva date personale şi săîşi confirme adresa de email. Pentru a realiza acestea aplicaţia îi pune la dispoziţie un formular de înregistrare, pe care utilizatorul va trebui să-l completeze pentru a realiza această etapă.

Actor principal: Utilizatorul simplu

Precondiţii: Utilizatorul să aibă site-ul încărcat în browser.

Postcondiţii: Utilizatorul devine un utilizator înregistrat al aplicaţiei, putând profita de funcţionalităţile pe care le oferă aceasta.

Scenariul principal de succes:

- Utilizatorul accesează pagina de înregistrare.
- Utilizatorul introduce datele sale personale, împreună cu numele de utilizator, emailul şi parola aleasă.
- Utilizatorul apasă butonul de trimitere de pe formular.
- Datele completate trec de partea de validare
- Noul cont este salvat cu succes de către sistem.
- Utilizatorul primește un email pentru confirmarea adresei de email.

Extensii posibile:

- Utilizatorul nu completează toate datele obligatorii de pe formular
- Utilizatorul adaugă un nume de utilizator deja existent în cadrul aplicaţiei
- Utilizatorul adaugă o parolă care nu are o lungime suficient de mare pentru a fi acceptată.
- Utilizatorul introduce un email invalid

In oricare din situaţiile de mai sus, sistemul va anula cererea de înregistrare şi va afişa un mesaj de eroare utilizatorului.

2. Autentificarea utilizatorului

Descriere: Odată ce utilizatorul are un cont înregistrat el trebuie să se autentifice cu acesta pentru a putea folosi funcţionalităţile pe care le oferă aplicaţia. Această operaţie se va face folosind formulare de autentificare oferit de aplicaţie. Pe acest formular se vor face şi toate redirectările utilizatorilor care încearcă să acceseze linkuri introduse manual, fără să fie autentificaţi sau accesează o resursă la care nu au acces.

Actor principal: Utilizatorul simplu

Precondiţii: Utilizatorul să aibă site-ul încărcat în browser şi pagina de autentificare.

Postcondiţii: Utilizatorul este autentificat şi poate să se folosească de funcţionalităţile pe care le oferă aplicaţia.

Scenariul principal de succes:

- Utilizatorul accesează pagina de autentificare.
- Utilizatorul introduce numele de acces(email sau nume de utilizator) împreună cu parola.
- Utilizatorul apasă butonul de trimitere de pe formular.
- Datele completate trec de partea de validare
- Utilizatorul este autentificat în aplicaţie.

Extensii posibile:

- Utilizatorul nu completează numele de acces sau parola,
- Utilizatorul adaugă un nume de acces care nu există
- Utilizatorul adaugăo parola care nu corespunde cu cea furnizată la pasul de înregistrare

În oricare din situaţiile de mai sus, sistemul va anula cererea de autentificare şi va afişa un mesaj de eroare utilizatorului, lăsând câmpurile corecte completate, pentru a uşura activitatea acestuia.

3. Creearea unui server nou în cloud

Descriere: Odată ce utilizatorul este autentificat în aplicaţie el se bucură de o serie de funcţionalităţi pe care aceasta le oferă. Una din ele este posibilitatea creerii unui nou server obţinut de la platformă furnizoare de cloud prin API-ul pus la dispoziţie de către aceasta.

Actor principal: Utilizatorul autentificat

Precondiţii: Utilizatorul trebuie să fie autentificat, iar contul său de utilizator trebuie să fie sincronizat cu contul utilizatorului de pe platforma furnizoare de cloud.

Postcondiţii: Utilizatorul are disponibil un server în cloud, pe care poate lansa aplicaţii.

Scenariul principal de succes:

- Utilizatorul accesează pagina de creere server.
- Utilizatorul are deja sincronizat contul cu cel de pe platforma furnizoare de cloud.
- Utilizatorul completează formularul cu dimensiunile şi proprietăţile serverului pe care urmează să-l creeze.
- Datele completate trec de partea de validare
- Utilizatorul are serverul disponibil.

Extensii posibile:

- Utilizatorul nu este autentificat în aplicaţie

In acest caz aceasta opţiune poate fi accesată doar prin introducerea manuala a linkului şi astfel utilizatorul va fi redirecţionat spre pagina de autentificare.

- Utilizatorul nu are sincronizat contul cu cel de pe platforma furnizoare de cloud

Se face o cerere către serverul de Oauth pentru a sincroniza conturile, utilizatorului i se va cere să se autentifice în contul de pe acea platformă furnizoare de cloud.

- Utilizatorul nu are încă un cont pe platformă furnizoare de cloud

În încercarea serverului de Oauth de a sincroniza conturile,utilizatorului i se va cere să se autentifice iar dacă nu are un cont, se va putea înregistra în acel moment.

4. Lansarea unei aplicaţii pe un server în cloud

Descriere: După ce utilizatorul este autentificat în aplicaţie şi după etapa creerii unui nou server obţinut de la platforma furnizoare de cloud utilizatorul are posibilitatea de a se folosi de funcţionalitatea cheie a acestui sistem şi anume lansarea propriu-zisă de aplicaţii web.

Actor principal: Utilizatorul autentificat

Precondiţii: Utilizatorul trebuie să fie autentificat şi să aibă la dispoziţie cel puţin un server creat în prealabil.

Postcondiţii: Utilizatorul realizează lansarea aplicaţiei sale web, reprezentând funcţionalitatea cea mai importantă a sistemului din lucrarea de faţă.

Scenariul principal de succes:

- Utilizatorul accesează pagina de lansare a unei aplicaţii.
- Utilizatorul are deja un server creat pe platforma furnizoare de servicii cloud.
- Utilizatorul completează formularul cu opţiunile de configurare pentru aplicaţia sa web.
- Datele completate trec de partea de validare
- Utilizatorul are aplicaţia lansată şi poate să o actualizeze atunci când e nevoie.

Extensii posibile:

- Utilizatorul nu este autentificat în aplicaţie

In acest caz această opţiune poate fi accesată doar prin introducerea manuala a linkului şi astfel utilizatorul va fi redirecţionat spre pagina de autentificare.

- Utilizatorul nu are sincronizat contul cu cel de pe platforma furnizoare de cloud

Se face o cerere către serverul de Oauth pentru a sincroniza conturile, utilizatorului i se va cere să se autentifice în contul de pe acea platformă furnizoare de cloud.

- Utilizatorul nu are încă un cont pe platformă furnizoare de cloud

În încercarea serverului de Oauth de a sincroniza conturile, utilizatorului i se va cere să se autentifice iar dacă nu are un cont, se va putea înregistra în acel moment.

- Utilizatorul nu are sincronizat contul cu cel de pe o platforma de găzduire a codului sursă bazată pe git

Se face o cerere către serverul de Oauth pentru a sincroniza conturile, utilizatorului i se va cere să se autentifice în contul de pe acel sistem de găzduire a codului bazat pe git.

- Utilizatorul nu are încă un server obţinut în prealabil de la platforma furnizoare de cloud

In acest caz utilizatorul nu va putea realiza etapa de lansare a aplicaţiei sale şi va trebui să realizeze întâi etapa de creere a serverului.

5. Actualizarea unei aplicaţii lansate la cea mai nouă versiune

Descriere: Pentru aplicaţiile deja lansate de către utilizator folosind sistemul din lucrarea de faţă, acesta are posibilitatea de a actualiza aceste aplicaţii la versiuni mai noi, în cazul în care acestea sunt în curs de dezvoltare. Cum majoritatea aplicaţiilor web din această perioadă, în special cele de mici dimensiuni se dezvoltă granulat, pentru a lansa cât mai repede şi a pivota cu diferite îmbunătăţiri, pentru a studia piaţa şi a lua în considerare părerea şi reacţia utilizatorilor, această funcţionalitate este foarte importantă. Această funcţionalitate ţine de etapa de mentenanţă a aplicaţiei, următoarea după cea de lansare.

Actor principal: Utilizatorul autentificat

Precondiţii: Utilizatorul trebuie să fie autentificat şi să aibă la dispoziţie cel putin o aplicaţie deja lansată.

Postcondiţii: Utilizatorul realizează actualizarea la cea mai nouă versiune a aplicaţiei sale web deja lansate prin intermediul sistemului.

Scenariul principal de succes:

- Utilizatorul accesează pagina de vizualizare a aplicaţiilor sale deja lansate.
- Utilizatorul are cel putin o aplicaţie lansată prin intermediul sistemului.
- Utilizatorul apasă butonul de update pe una din aplicaţiile sale
- Se realizează actualizarea aplicaţiei aleasă de utilizator.

Extensii posibile:

- Utilizatorul nu este autentificat în aplicaţie

În acest caz această opţiune poate fi accesată doar prin introducerea manuala a linkului şi astfel utilizatorul va fi redirectionat spre pagina de autentificare.

- Utilizatorul nu are încă o aplicaţie lansată prin intermediul sistemului

În acest caz utilizatorul nu va putea realiza etapa de actualizare a aplicaţiei sale şi va trebui să realizeze întâi etapa de lansare a acesteia.

6. Resetarea fizică a serverului pe care rulează aplicaţia

Descriere: Pentru aplicaţiile deja lansate de către utilizator folosind sistemul din lucrarea de faţă, acesta are posibilitatea de a reseta serverul pe care rulează aplicaţia în cazul în care e nevoie de asta. Exista posibilitatea să apară un eveniment neprevazut şi din această cauzaă serverul să nu mai funcţioneze la parametrii normali, de obicei acest lucru se întamplă atunci când traficul este prea mare şi serverul nereuşind să facă faţă

tuturor cererilor se supraîncarcă la nivel de procesare. Această funcţionalitate ţine de etapa de mentenanţă a aplicaţiei, urmatoarea după cea de lansare.

Actor principal: Utilizatorul autentificat

Precondiţii: Utilizatorul trebuie sa fie autentificat şi să aibă la dispoziţie un server şi cel puţin o aplicaţie deja lansată.

Postcondiţii: Utilizatorul realizează resetarea serverului pe care rulează o aplicaţie deja lansată de către acesta.

Scenariul principal de succes:

- Utilizatorul accesează pagina de vizualizare a aplicaţiilor sale deja lansate.
- Utilizatorul are cel puţin o aplicaţie lansată prin intermediul sistemului.
- Utilizatorul apasă butonul de reset server în dreptul unei din aplicaţii
- Se realizează resetarea fizică a serverului ales de utilizator.

Extensii posibile:

- Utilizatorul nu este autentificat în aplicaţie

În acest caz această opţiune poate fi accesată doar prin introducerea manuală a linkului şi astfel utilizatorul va fi redirecţionat spre pagina de autentificare.

- Utilizatorul nu are încă un server obţinut în prealabil de la platforma furnizoare de cloud

În acest caz utilizatorul nu va putea realiza acest pas.

7. Vizualizarea listei de aplicaţii lansate

Descriere: Pentru aplicaţiile deja lansate de caăre utilizator folosind sistemul din lucrarea de faţă, acesta are posibilitatea de a vizualiza aceste aplicaţii pentru a putea apoi să realizeze operaţii care aparţin etapei de mentenanţă cum ar fi actualizarea sau restarea fizică a serverului.

Actor principal: Utilizatorul autentificat

Precondiţii: Utilizatorul trebuie sa fie autentificat.

Postcondiţii: Utilizatorul analizează lista sa de aplicaţii deja lansate.

Scenariul principal de succes:

- Utilizatorul accesează pagina de vizualizare a aplicaţiilor sale deja lansate.
-

Extensii posibile:

- Utilizatorul nu este autentificat in aplicatie

În acest caz această opţiune poate fi accesată doar prin introducerea manuala a linkului şi astfel utilizatorul va fi redirecţionat spre pagina de autentificare.

- Utilizatorul nu are încă o aplicaţie lansată prin intermediul sistemului

În acest caz lista aplicaţiilor sale va fi goală.

4.3 Tehnologii si protocoale utilizate

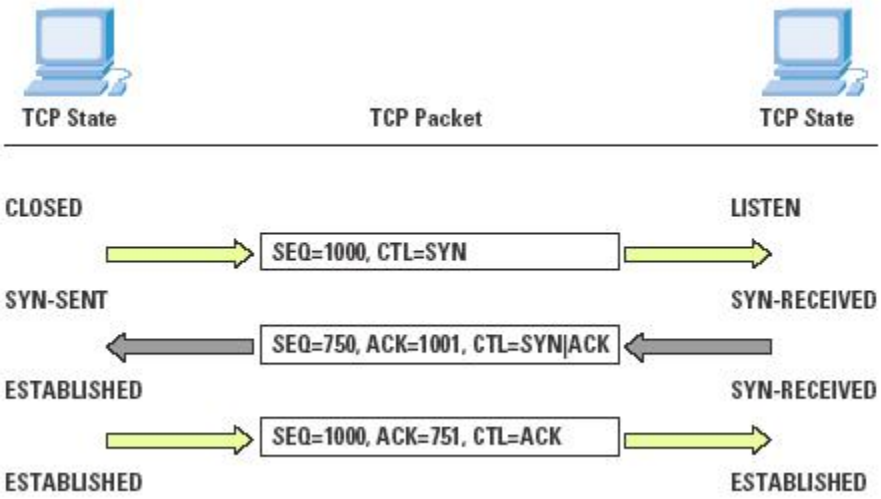
În continuare se va prezenta lista protocoalelor şi tehnologiilor care s-au utilizat în această lucrare, împreună cu explicaţii pentru fiecare dintre ele, pentru a pune la curent cu acestea şi utilizatorii care nu au experienţă pe partea de dezvoltare web.

4.3.1 HTTP/TCP

TCP (Transmission Control Protocol) – este unul dintre protocoalele de bază ale Internetului fiind una din cele două componente ale suitei TCP/IP. După cum îi spune și numele TCP este un protocol aflat la nivelul de transport de date, astfel acesta asigură transmiterea pachetelor intre diferitele dispozitive care comunică în rețea. „Efectuează o conectare virtuală full duplex între două puncte terminale, fiecare punct fiind definit de către o adresă IP și de către un port TCP. ”[6]

Notiunea de **port** se referă la un număr întreg fără semn, pe 16 biți (așadar, în intervalul 1, 65535) care definește canalele pe care un calculator poate comunica cu alte dispozitive folosind conexiuni prin **socket.**

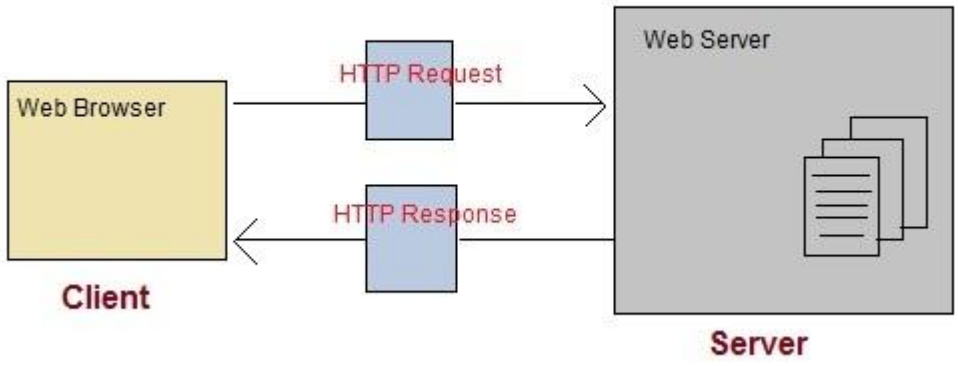
Un **socket** reprezintă un canal de comunicatie realizat între două dispozitive care fac schimb de informații de fiecare parte a acestuia, cele două dispozitive sunt identificate printr-un IP și un port.



Figură 4.3 – Protocolul TCP

Principalul avantaj al acestui protocol de transport este acela că se asigură de faptul că datele vor ajunge la destinație, astfel împarte datele în pachete de lungimi finite și le atașează anumite headere, iar pe masură ce primește confirmare de la dispozitivul destinație continuă sa trimită, sau retrimite părțile precedente în cazul în care acestea nu au ajuns la destinație. În majoritatea aplicațiilor este foarte important acest aspect întrucât, e absolut necesar să ne asigurăm că informațiile au ajuns în siguranța la destinație. Pentru a se asigura că datele trimise sunt primite de către dispozitivul

destinație și integritatea acestora nu a avut de suferit, protocolul TCP trebuie sa sacrifice viteza de transmisie, astfel așteptand după semnalele de confirmare de la dispozitivul receptor și existând o conexiune strânsă între cei doi participanți la schimbul de date viteza de transmisie este mai redusă. Astfel că pentru aplicațiile în care este importantă viteza de transmisie și mai puțin integritatea datelor transmise se va folosi celălat protocol de transport UDP (User Datagram Protocol), care trimite datele fără să se asigure de faptul că acestea vor ajunge în siguranță la destinatie și nefiind nevoit sa aștepte confirmări transmisia se va face mult mai repede. **Totuși majoritatea aplicațiilor preferă să piardă puțin din viteză în schimbul integrității datelor, astfel că cel mai folosit protocol de transport este protocolul TCP.**



Figură 4.4 – Rolul serverului web

HTTP (HyperText Transfer Protocol) - este un protocol de tip text de nivel aplicație, folosit pentru a accesa resursele din Internet păstrate în www (World Wide Web). Acesta este și protocolul implicit folosit pentru comunicare în Internet, astfel că dacă în zona de adrese a browserului nu punem nimic ca și protocol se va considera că este o cerere bazată pe protocolul HTTP. Astfel HTTP ofera o tehnica de comunicare prin care se pot transmite pagini web între două dispozitive.

Versiunile protocolului HTTP sunt HTTP/0.9, **HTTP/1.0**, **HTTP/1.1.** În prezent se folosește versiunea **HTTP/1.1** întrucât avantajul pe care-l aduce față de versiunea precedentă este că se realizează o singura conexiune TCP pentru toată comunicarea, spre deosebire de realizarea unei conexiuni pentru fiecare acces la resurse în versiunea **HTTP/1.0**. După cum am mentionat, HTTP functionează peste protocolul de transport TCP pe portul 80.

Astfel, atunci când se accesează un link, în zona de adrese a browserului, dacă protocolul setat este HTTP sau se folosește acesta ca și protocol implicit, întai se împarte linkul astfel: zona de după „www.” și până la denumirea domeniului de nivel superior (ex: .com) este considerată adresa dispozitivului gazdă (host). Ceea ce urmează în continuare este o cale locală pe acel dispozitiv, iar dacă e prezent marcatorul „?”, ceea ce urmează după el, uniti prin marcatori „&” reprezintă parametrii cererii HTTP. În funcție de metoda HTTP care se folosește acești parametrii pot să apară sau nu în link. Metodele care se pot folosi pentru transmiterea unei cereri HTTP sunt:

GET - este metoda cea mai des întalnită și este folosită atunci când serverului ii se cere o anumită resursă sau un set de resurse. În cadrul acestei metode parametrii sunt atașați în link, încep de la marcatorul „?” și sunt legate perechile de parametrii de forma „nume_parametru = valoare” prin marcatorul „&”.

HEAD - se comportă exact ca și metoda GET descrisă mai sus, diferența fiind că în cazul acestei metode, serverul returnează doar antetul resursei, ceea ce permite clientului să îl verifice fără a fi nevoit să obțină și resursa în sine.

PUT - este metoda folosită pentru a salva anumite documente pe server

POST – este o alta metodă cunoscută a protocolului HTTP, în care parametrii sunt trimiși în corpul mesajului nu în link(asemeni metodei GET) și a fost proiectată pentru a trimite date de intrare către server, fiind des întalnită la formulare cu datele utilizatorului, deoarece informații precum parolele și alte câmpuri confidențiale nu se pot trimite în link.

DELETE: este opusul metodei PUT, prin această metodă se șterg resursele de pe server.

OPTIONS: este folosită pentru identificarea capacităților serverului Web, înainte de a inția o cerere.

La fiecare cerere HTTP există și un raspuns pe care calculatorul care o initiază îl va primi de la server. Aceste raspunsuri se clasifică și ele în câteva categorii importante. Cele cinci clase de coduri răspunsuri HTTP sunt următoarele:

100 - Orice cod care începe cu 1 la cifra sutelor reprezintă erori informationale, adică un răspuns provizoriu pe care îl dă serverul, acesta anuntând utilizatorul că poate să continue cererea sau că e în curs de schimbare a protocolului

200 - Orice cod de raspuns care are 2 ca cifră a sutelor informează utilizatorul că cererea s-a efectuat cu succes.

300 - Codurile de răspuns care incep cu 3, sugerează faptul că va fi nevoie de o redirectare pentru a putea duce cererea la bun sfârșit, deși de multe ori operațiile de redirectare sunt responsabilitatea browserului.

400 - Codurile de răspuns care încep cu 4 reprezintă mesaje de eroare ale utilizatorului, de cele mai multe ori aceste coduri apar atunci când utilizatorul a formulat greșit cererea către server. Aceste coduri de răspuns îi indică utilizatorului faptul că va trebui să reformuleze cererea pentru a putea fi satisfacută. Câteva dintre cele mai importante răspunsuri de acest fel sunt **401** – neautorizat, **403** – interzis, **404** – negăsit.

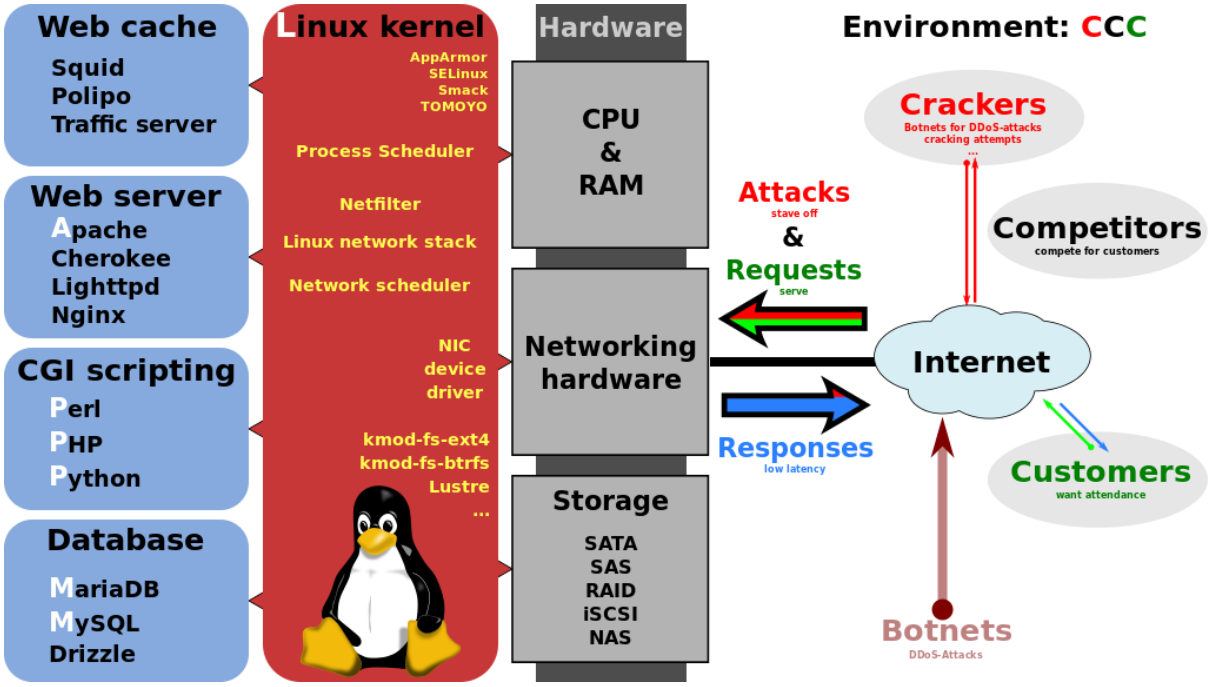
500 - Răspunsurile serverului care au 5 ca cifra sutelor indică faptul că serverul e incapabil să rezolve cererea datorită unei erori interne. În cazul acestor erori clientul nu

poate decât să încerce cererea într-un alt moment de timp, când partea de server va rezolva problemele interne. [7]

Pe partea de server, atunci când apare o cerere HTTP aceasta este preluată și rezolvată de către un server web. În continuare voi prezenta conceptul de **server web**.

4.3.2 Server Web

Serverul web reprezintă serverul care gazduiește și ține evidența paginilor web. Aceasta este entitatea care tratează cererile care vin prin intermediul protocolului HTTP, pe portul 80. Aceasta este de fapt o aplicație instalată pe un server fizic, care rulează la infinit și asteaptă cereri pe portul 80 al serverului. Astfel clientul care este reprezentat de un browser web pe mașina utilizatorului initiază o cerere HTTP pe portul 80 spre server, aceasta fiind tratată de către serverul web instalat pe mașina server, returnând răspunsul așteptat.



Figură 1.5 – Servere web

După cum putem observa și în schema de mai sus principalele tipuri de servere web sunt Apache, Cherokee, Lighttpd, Nginx. Pentru simplitate dar și datorită experientei cu aceste tipuri de servere web, în sistemul descris voi folosi serverul web Apache sau Nginx. [8]

4.3.3 Limbajul de programare Php

Php – este un limbaj de programare orientat pe obiect, care se folosește pentru partea de server a unei aplicații web. „Numele PHP provine din limba engleză și este un acronim recursiv : Php Hypertext Preprocessor. Folosit inițial pentru a produce pagini

web dinamice, este folosit pe scară largă în dezvoltarea paginilor și aplicațiilor web. Se folosește în principal înglobat în codul **HTML**, dar începând de la versiunea 4.3.0 se poate folosi și în mod „linie de comandă" (**CLI**), permițând crearea de aplicații independente". **Php este unul dintre cele mai importante limbaje de programare web, fiind disponibil pentru toate sistemele de operare. [9]**

Figură 4.6 - Php

Acest limbaj de programare are urmatoarele avantaje:

- **Intuitivitatea sintaxei** : sintaxa limbajului este foarte usor de înțeles fiind o combinație a sintaxelor limbajelor Perl și C;
- **Simplitatea** : sintaxa limbajului este destul de liberă. Nu este nevoie de includere de biblioteci sau de directive de compilare, codul PHP inclus într-un document executându-se între marcajele speciale
- **Eficiența** : PHP-ul se folosește de mecanisme de alocare a resurselor, foarte necesare unui mediu cu utilizatori multiplii, asa cum este domeniul web
- **Securitate** : PHP-ul pune la dispoziția programatorului un set flexibil și eficient de măsuri de siguranță
- **Flexibilitate** : fiind apărut din necesitatea dezvoltării web-ului, PHP a fost modularizat pentru a ține pasul cu dezvoltarea diferitelor tehnologii. Nefiind legat de un anumit server web, PHP-ul a fost integrat pentru numeroasele servere web existente: Apache, Nginx, etc.

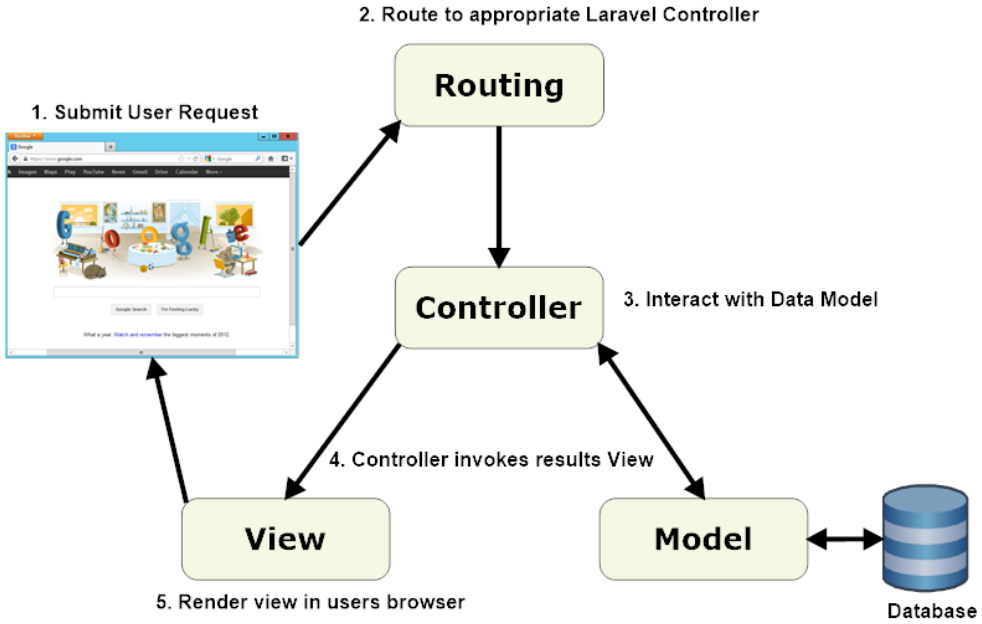
- **Gratuitate** : este probabil cea mai importantă caracteristică a PHP-ului. Dezvoltarea PHP-ului sub licența open-source a determinat adaptarea rapidă a PHP-ului la nevoile web-ului, eficientizarea și securizarea codului.

In sistemul analizat de lucrarea de față, limbajul de programare care se va folosi pe partea de server este limbajul Php, însă se va folosi un framework peste acest limbaj care simplifică foarte mult sarcinile dezvoltatorilor web. Acest framework se numește **Laravel 4.0** și va fi analizat în detaliu în secțiunea următoare.

4.3.3 Framework-ul Laravel 4.0

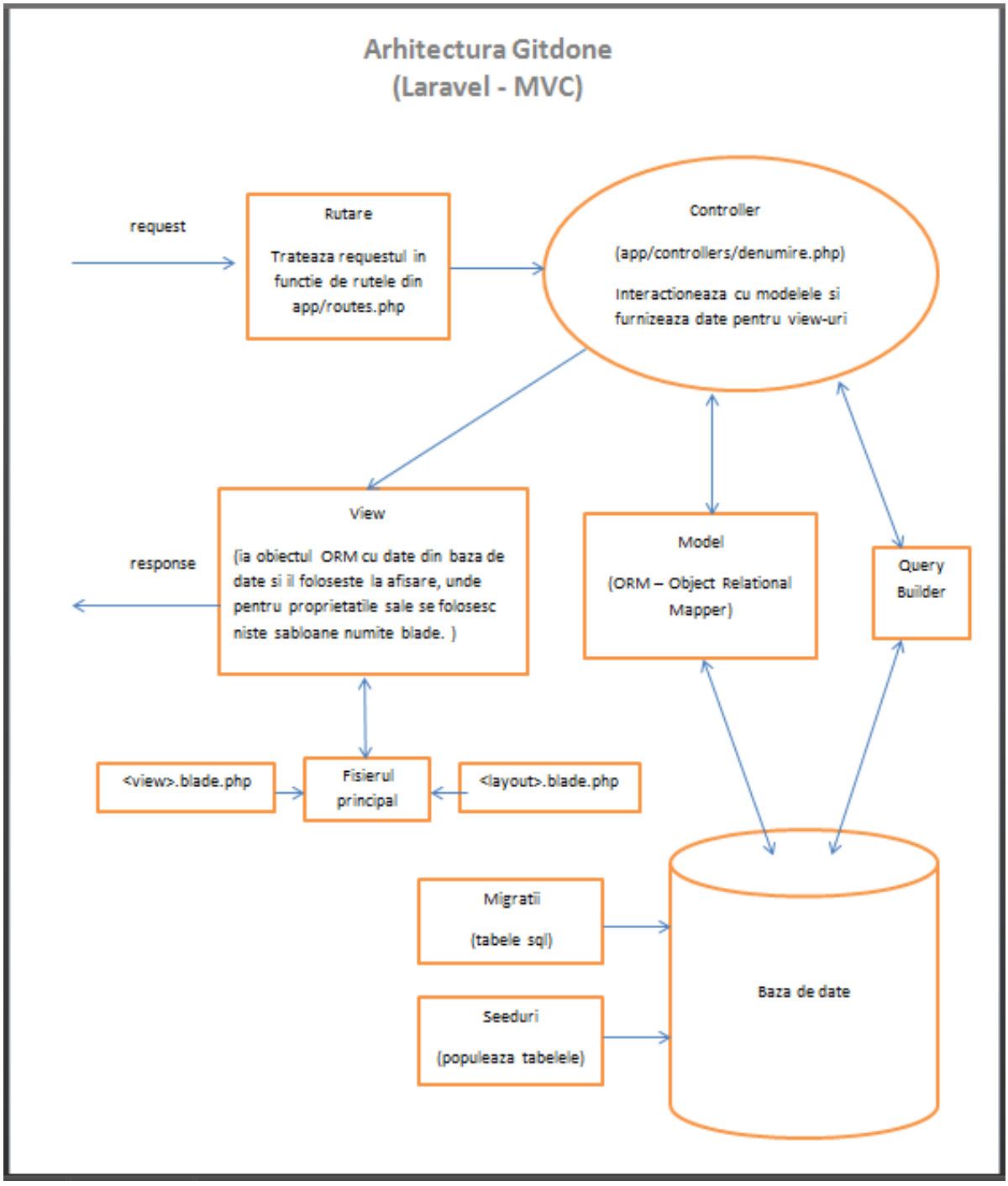
Pentru dezvoltarea aplicației de lansare automată a altor aplicații web voi folosi fraeworkul Laravel 4.0, care este un framework peste limbajul de programare php. Acesta este un framework gratis oferit utilizatorilor, fiind desemnat cel mai popular framework pe php în anul 2013. Totodata, fiind un framework în curs de dezvoltare codul sursă al acestuia se afla pe Github (https://github.com/laravel/laravel), fiind cel mai popular și mai vizualizat proiect php de pe faimoasa platformă.

Laravel este dezvoltat de Taylor Otwell și a apărut într-o primă versiune în 22 februarie 2012, iar ultima versiune stabilă a fost lansată în 6 august 2014, fiind versiunea pe care o voi folosi în dezvoltarea aplicației. După cum se poate observa framework-ul este într-o continuă evoluție apărând în permanență actualizări și îmbunătățiri. [10]



Figură 4.7 - Laravel

Arhitectura sistemului este una de tip Model-View-Controller cu cateva ajustări care-l adaptează pentru combinarea codului php cu html și pentru interactiunea cu baza de date. În figura de mai jos se poate observa în detaliu arhitectura sistemului.



Figură 4.8 – Arhitectura Laravel

Aspectul care m-a impresionat cel mai mult la acest framework este introducerea conceptului de ORM (Object Relational Mapping), care mapează obiectele la structura tabelelor din baza de date, astfel că fiecare coloană din tabelul pe care îl reprezintă modelul se va regăsi ca și atribut al acestuia.

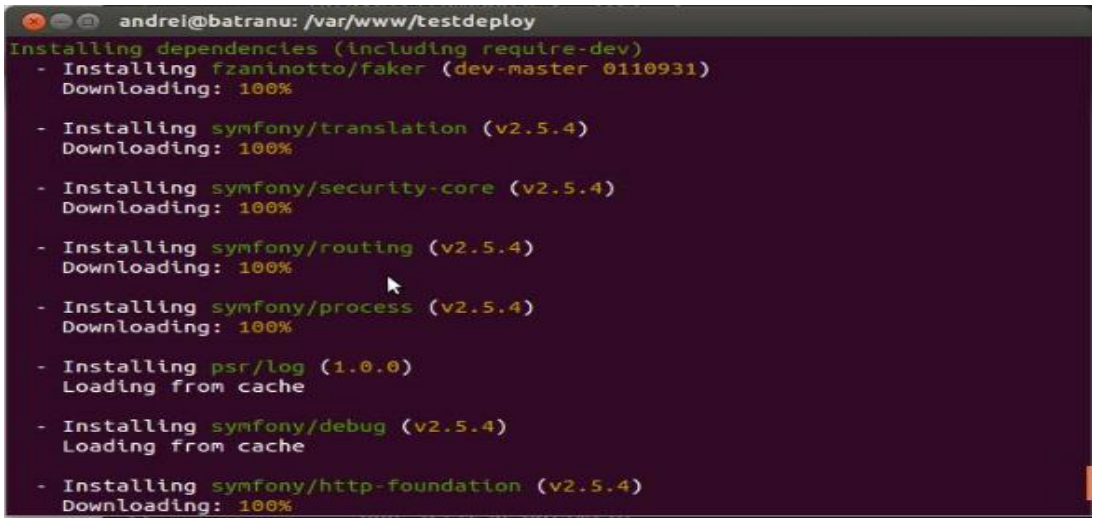
Un alt aspect de admirat al acestui framework este acela că poate include pachete și biblioteci construite pe php, prin asa-numita componentă composer, astfel că și framework-ul este dezvoltat granulat și inclus prin acest composer. Această componentă face ca orice biblioteca să fie foarte usor de folosit, doar se include denumirea pachetului într-un fisier cu extensia .json, iar composerul va include automat acel pachet. În continuare voi prezenta mai pe larg conceptul de **composer**.

4.3.3 Php Composer, bower

Termenul de composer descrie o unealta care organizează dependințele bibliotecilor din php. Astfel este necesară doar specificarea dependințelor într-un fișier cu extensia .json și composer-ul va include automat acele biblioteci. Instalarea acestei unelte foarte puternice se face simplu, prin comenzile:

curl -sS https://getcomposer.org/installer | php
php –r "readfile('https://getcomposer.org/installer');" | php

După aceasta, composer-ul este instalat și în directorul principal al aplicației va exista un fișier composer.json, unde vor fi incluse numele bibliotecilor, iar apoi prin rularea comenzii: **composer install**, vor fi instalate bibliotecile și dependințele acestora. În cazul în care se adauga biblioteci pe parcurs, se va apela comanda: **composer update** și acestea vor fi gata sa fie utilizate.



Figură 4.9 – Instalare pachete, composer

Așa cum pe partea de server se folosește **composer** pentru a include biblioteci și dependințele acestora, pe partea de front-end se folosește o unealta asemanatoare numită

bower care în aceeasi manieră ca și composer instalează bibliotecile și dependințelor lor, prin adaugarea lor intr-un fișier cu extensia .json și apelarea comenzii de instalare.

Pentru a instala bower se foloseste comanda:

npm install –g bower

Pentru a instala bibliotecile și dependințele lor, se includ într-un fișier bower.json și se apelează comanda **bower install**.

Astfel, bibliotecile de pe partea de front-end vor fi gata de utilizare.

Una dintre cele mai importante biblioteci de acest tip, care este mai degrabă un framework sau o colecție de unelte pentru partea de front-end, este **Twitter bootstrap,** care va fi prezentată în continuare.

4.4.4 Twitter bootstrap

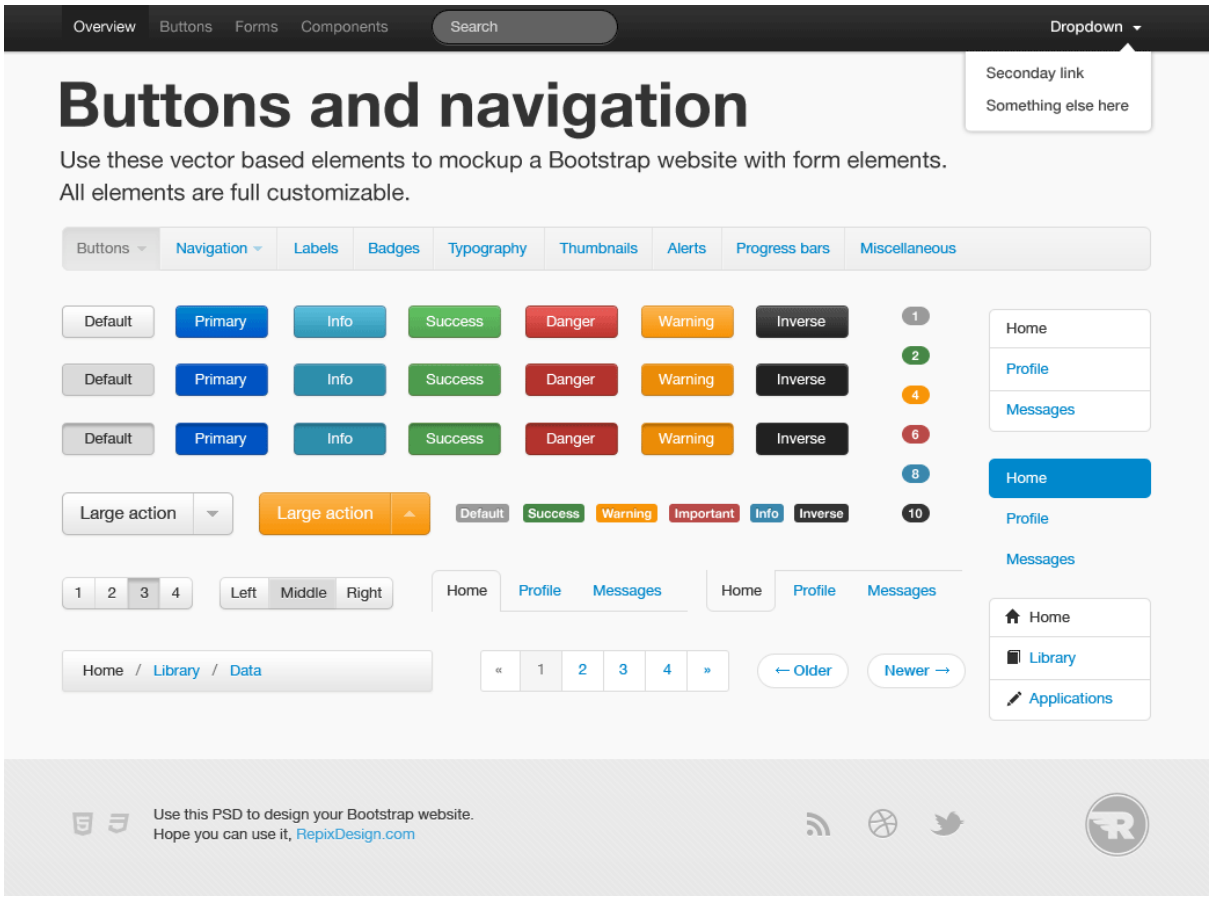
Twitter bootstrap este o colecție de unelte pentru partea de front-end a aplicațiilor web, aceasta conținand elemente de Html, Css și Javascript gata făcute, pentru ca dezvoltatorii sa poata trece cât mai rapid peste partea de design și aranjare a elementelor, folosind o serie de componente realizate de alți dezvoltatori.

Twitter bootstrap este open-source și a fost inițial creată ca o unealtă internă pentru cei de la Twitter, de aici provine și denumirea de Twitter bootstrap. Datorită faptului că a avut un succes neașteptat cei de la Twitter au hotărât să o publice și să o facă disponibilă gratuit pentru a contribui la dezvoltarea domeniului web.

Astfel, dezvoltatorii care vor să construiască aplicații web, pe care vor să le lanseze și testeze cât mai repede pentru a observa reacția utilizatorilor și a putea hotărî dacă este o aplicație în care merită investit și merită continuată, sunt scutiți de o mare parte din timpul dezvoltării prin includerea în aplicațiile lor a componentelor și elementelor grafice gata facute de la Twitter bootstrap. De fapt, în esență, acesta pare să fi fost și motivul principal pentru care cei de la Twitter au lansat acest set de unelte, după cum sugereaza și numele, bootstrap duce cu gândul la constrângeri financiare sau temporale.

În sistemul pe care-l analizează lucrarea de față, se vor folosi aceste componente și elemente de html, css și javascript de la Twitter bootstrap tocmai pentru că desi simplist, design-ul este unul suficient de bun și timpul alocat realizării paginilor este mult mai redus.

Mai jos sunt exemplificate câteva din aceste elemente și componente vizuale pe care le oferă acest set de unelte.



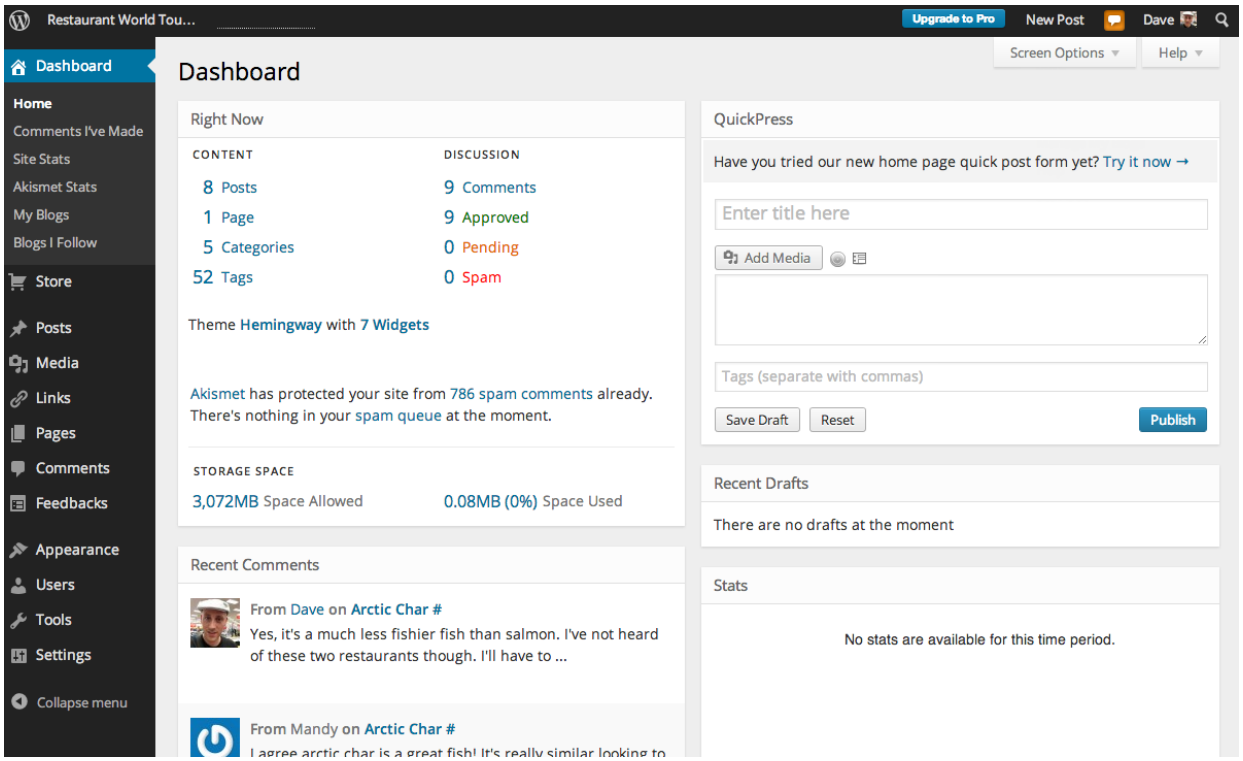
Figură 4.10 – Elemente de Boostrap

După această scurtă paralelă cu partea de front-end a aplicației, se va reveni la partea de server și se vor discuta cateva aspecte importante în economia aplicației. Urmatoarea tehnologie discutată va fi **Wordpress**, care deși nu va fi folosită la dezvoltarea aplicației în sine, va fi tehnologia prin intermediul căreia se vor dezvolta majoritatea aplicațiilor care vor fi lansate prin intermediul sistemului de lansare Gitdone. Întrucât este ușor de folosit pentru utilizatorii finali prin zona de administrare a aplicației care este intuitivă și completă, platforma wordpress este din ce în ce mai folosită de către dezvoltatorii de aplicații web, tocmai pentru că utilizatorii finali se descurcă foarte bine sa-și administreze conținutul pe astfel de aplicații.

4.4.4 Wordpress

Wordpress este o platformă open-source pentru dezvoltarea de aplicații web. Aceasta a fost inițial dezvoltată pentru publicarea de blog-uri, dar s-a modificat în timp în așa fel încât se pot realiza aplicații web destul de complexe folosind această platformă. Principalul său avantaj este acela că este foarte ușor de folosit de către clientul final al aplicației, intrucât acesta își poate adauga conținutul foarte ușor folosind zona de

administrare a platformei care este declarat unul dintre cele mai puternice CMS (Content Management System) - sistem de gestionare a conținutului.



Figură 4.11 – Wordpress zona de administrare

În imaginea de mai sus este prezentată zona de administrare a conținutului site-urilor realizate pe această platformă, fiind foarte flexibila și usor de înteles chiar și pentru utilizatorii fără cunoștințe tehnice în domeniul programării. În ultima vreme direcțiile wordpress-ului tind spre realizarea de teme configurabile care să satisfacă cererile a unui număr cât mai mare de utilizatori.

Aceste teme sunt apoi cumparate de către cei care își doresc să-și dezvolte o aplicație web și găsesc o astfel de temă care se mulează pe cerințele și funcționalitățile pe care le caută.

Cum adăugarea de conținut pe o astfel de aplicație este foarte simplă cu ajutorul sistemului de management a conținutului, cea mai grea etapă din ciclul de viață a dezvoltării unei astfel de aplicații devine lansarea. Aici intervine sistemul care este discutat în lucrarea de față, acești clienți reprezentând una din principalele categorii de utilizatori cărora se adreseaza.

Platforma wordpress este scrisă în limbajul php, iar sursa de date cu care interactionează acesta este o bază de date relațională gestionată cu ajutorul sistemului **MySQL**.

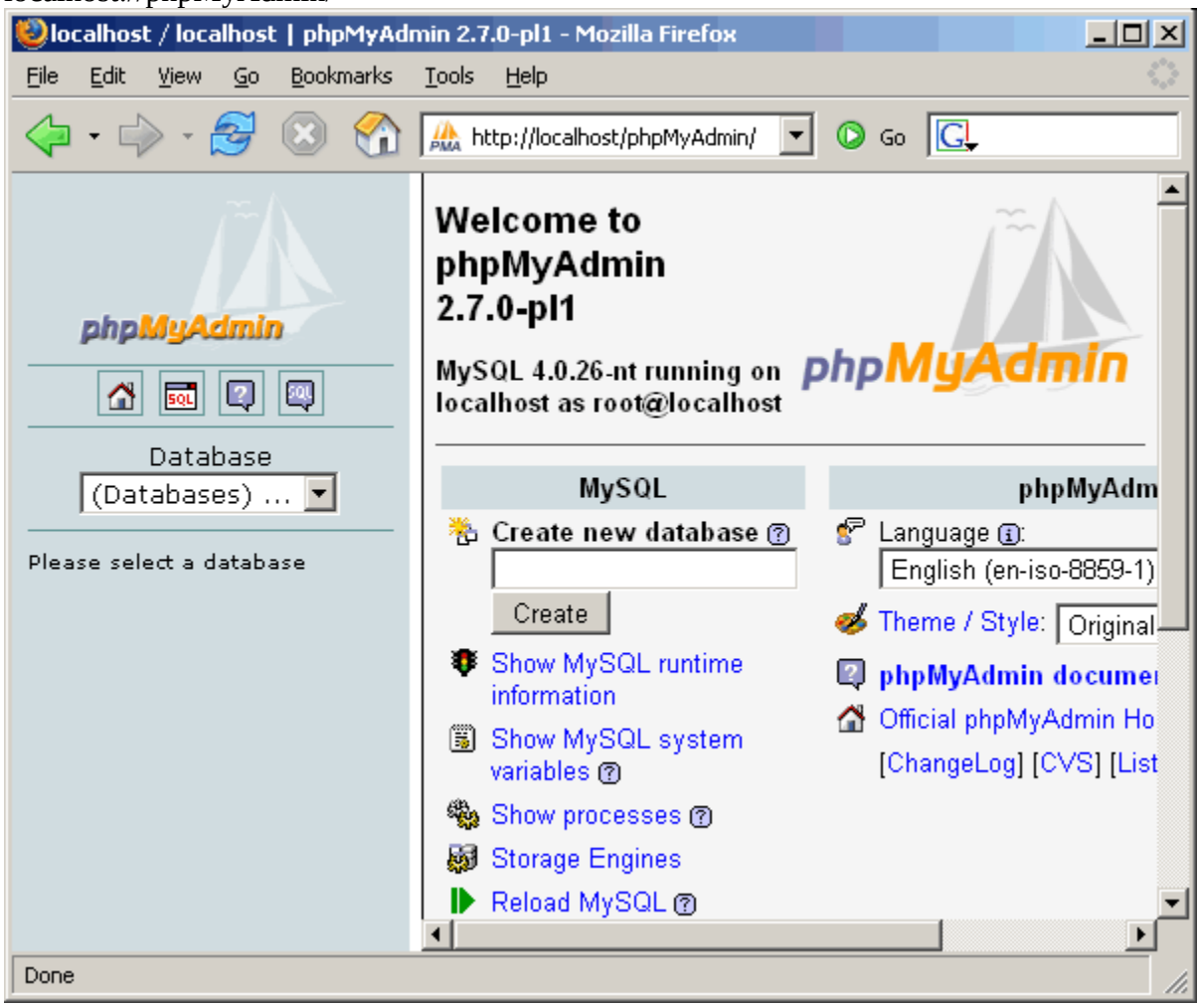
Sistemul de gestiune a bazelor de date, **MySQL** va fi folosit și de către sistemul de lansare al aplicațiilor, Gitdone. În etapa de dezvoltare se va folosi o unealtă pentru vizualizarea tabelelor din MySQL, numită PhpMyAdmin. În continuare, se va prezenta MySQL și foarte puțin din interfața pentru gestiunea datelor pe care o oferă unealtă PhpMyAdmin.

4.4.5 MySQL

MySQL reprezintă un sistem de gestiune a bazelor de date relationale open-source, devenit cel mai popular sistem de acest fel, folosit adeseori împreună cu php pentru dezvoltarea de aplicații web. Deși utilizarea frecventă acestuia împreună cu php face ca cele două sa fie considerate un tot unitar, MySQL se poate integra cu orice limbaj de programare orientat pe obiect, oferind drivere pentru fiecare dintre acestea. Pentru administrarea bazelor de date se poate folosi linia de comandă sau există diverse interfețe grafice care să rezolve această problemă. Una dintre cele mai populare interfețe de acest fel este phpMyAdmin, pe care o voi folosi și eu în etapa de dezvoltare a sistemului Gitdone.

Pe partea de server se poate folosi o unealta numită Adminer, care este alcatuită dintr-un singur fisier php, care se pune în directorul parinte al aplicației și se poate accesa din browser la adresa **ip_server/adminer.php.**

Mai jos se poate observa interfața uneltei phpMyAdmin, aceasta se accesează cu localhost://phpMyAdmin/

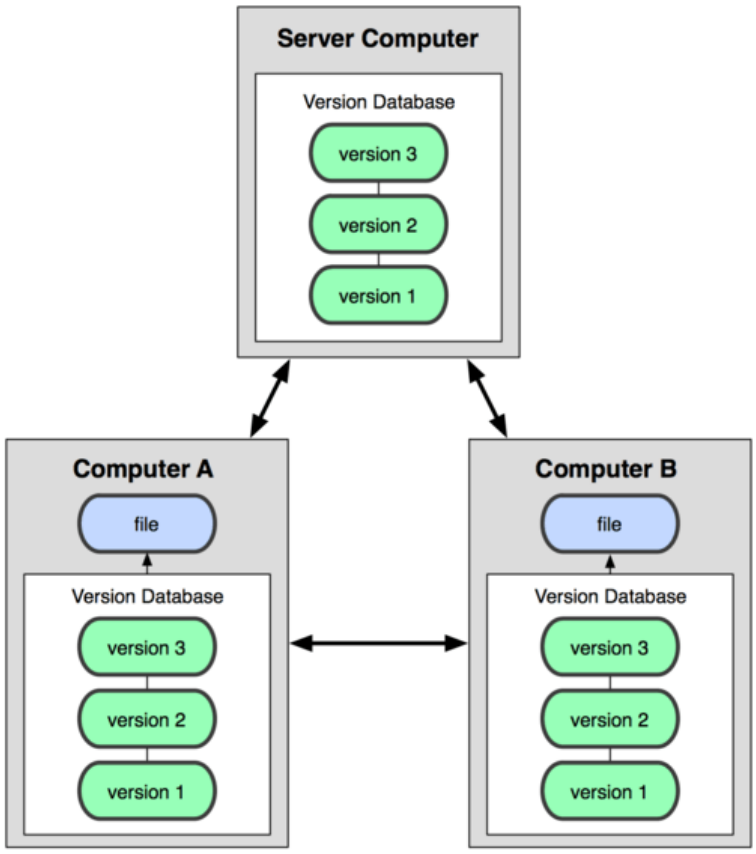


Figură 4.12 – PhpMyAdmin zonă de administrare

1.3.1. 4.4.6 Git, Bitbucket, Github

O componentă foarte importantă a aplicației o reprezintă sistemul distribuit de control al versiunilor, numit git. Git reprezintă un sistem de management al codului sursă (Source Code Management) și de control distribuit al versiunii care a fost lansat în 2005 initial pentru dezvoltarea nucleului de Linux.

Fiind un software open-source, util pentru dezvoltarea oricărui soft, de la proiecte mici până la cele gigant, git a devenit foarte popular în ultimii ani în rândul dezvoltătorilor de software. Comenzile pe care le folosește git sunt simple și ușor de învățat, făcându-l un sistem foarte accesibil.

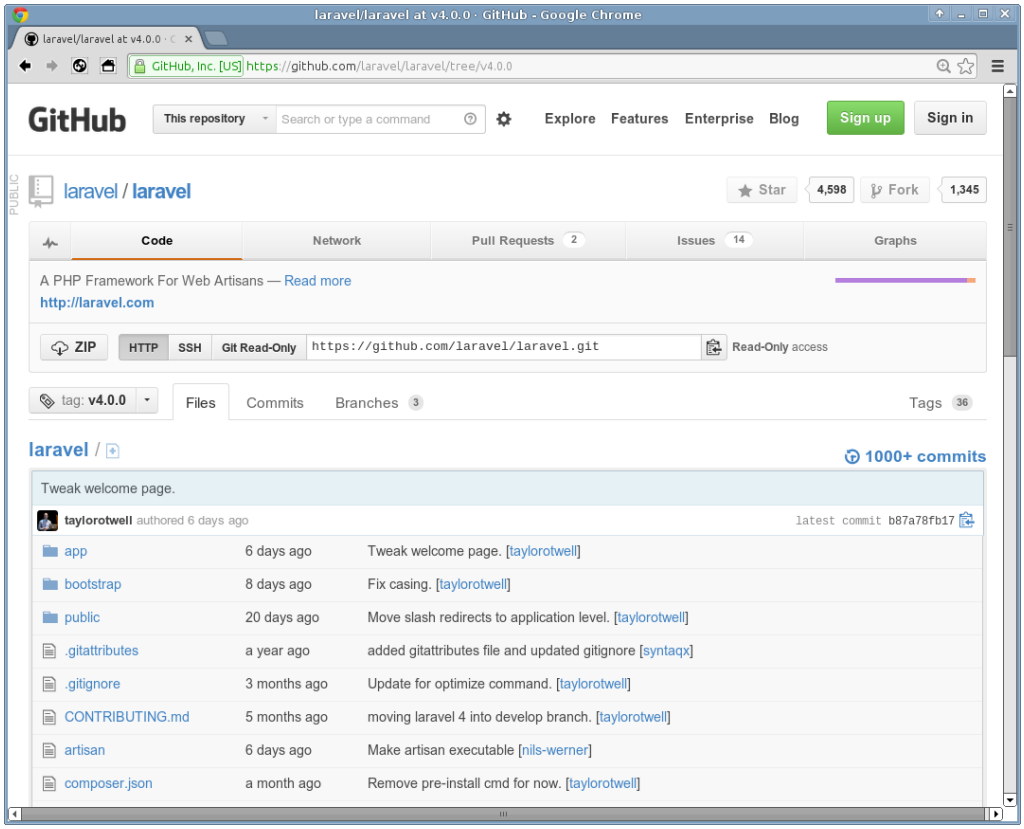


Figură 4.13 – Git, organizarea versiunilor

Ideea pe care se fundamentează git este aceea că un sistem are o versiune curenta stabilă, stocată pe un server, care este replicată pe mașinile locale ale dezvoltatorilor, aceștia din urmă folosind tehnicile de branch și merge realizează functionalități noi față de cele existente cât și rezolvări de defecțiuni ale sistemului curent. Într-un final codul aplicațiilor ajunge să fie pe acel server care ține versiunea curenta și de pe care dezvoltatorii o copiază de fiecare dată când lucrează la o funcționalitate. Pentru partea de server a sistemului git și cea de depozitare de cod există numeroase platforme, dintre care cea mai cunoscută este github, care este corespondentul facebook-ului de azi, în lumea dezvoltatorilor de soft. O altă platformă cunoscută de depozitare a codului este bitbucket dezvoltată de Atlassian.

Se vor discuta în detaliu cele două platforme, deoarece vor reprezenta sursele de unde sistemul va lua codul aplicațiilor pe care le vor lansa utilizatorii.

Github – este un serviciu de găzduire a codului pentru proiecte de dezvoltare soft care utilizează sistemul de control al versiunilor Git. Aceasta este cea mai mare platformă de acest fel, cu o popularitate foarte mare în rândul dezvoltatorilor de soft, oferind servicii gratuite pentru un număr limitat de proiecte, fiecărui utilizator.

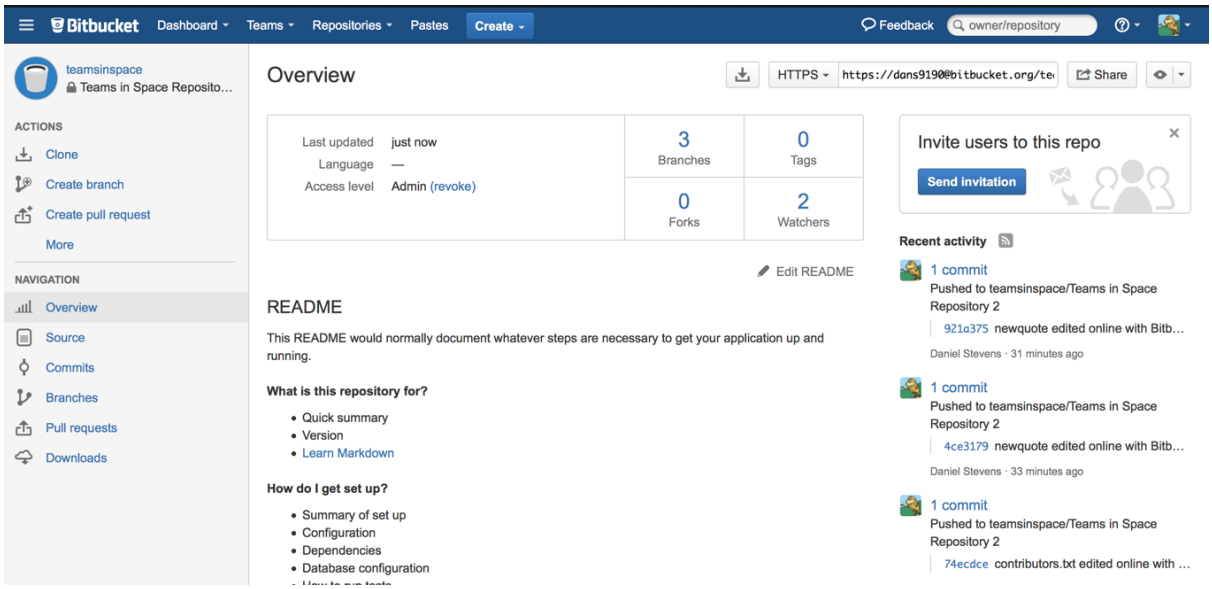


Figură 4.14 - Github

În imagine se observa interfata pe care o ofera github pentru gestionarea și versionarea structurii de directoare a unei aplicatii. Utilizatorii sistemului Gitdone vor avea posibilitatea de a lansa aplicatii ale caror cod sursa este depozitat pe aceasta platforma.

Cealalta platforma importanta bazata pe git este Bitbucket, dezvoltata de o renumita companie de software, Atlassian.

Bitbucket – este un alt serviciu de găzduire a codului pentru proiecte de dezvoltare soft care utilizează sistemul de control al versiunilor Git. Spre deosebire de Github sistemul este mai recent și mai putin cunoscut dar ofera aproximativ aceleasi functionalitati, oferind atât servicii gratuite cât și servicii enterprise. Diferența între cele două este că Github limitează numărul de proiecte pentru un anumit utilizator, în timp ce Bitbucket limitează numărul membrilor echipei și nu numărul proiectelor. Astfel că pentru echipe mici cu proiecte multe, aspect caracteristic firmelor mici de software, este mai potrivit Bitbucket, iar pentru echipele cu un număr de proiecte limitat este mai potrivit Github.



Figură 4.15 - Bitbucket

În imaginea de mai sus se observa panoul principal al unui proiect pe platforma Bitbucket cu principalele optiuni disponibile.

Utilizatorii sistemului Gitdone vor avea de asemenea posibilitatea de a lansa aplicatii ale caror cod sursa este depozitat și pe Bitbucket.

Utilizarea celor doua platforme se va face prin sincronizarea conturilor de utilizator din aplicație cu cele de pe una din aceste platforme de gazduire a codului. Aceasta functionalitate se va realiza prin intermediul protocolului de Oauth. Pentru Github se va folosi Oauth 2.0, iar pentru Bitbucket, Oauth 1.0. Utilizatorului care se inregistreaza în aplicație i se va cere sa își sincronizeze conturile, astfel va fi trimisa o cerere serverului de Oauth, care va cere utilizatorului sa se logheze în aplicatia de Github sau Bitbucket, de unde se va genera un token(cheie) de acces, care va fi stocata impreuna cu datele utilizatorului și va putea fi folosita pentru urmatoarele accesari ale resurselor de catre sistem, astfel vor putea fi efectuate lansarile aplicatiilor pe care utilizatorul doreste sa le selecteze.

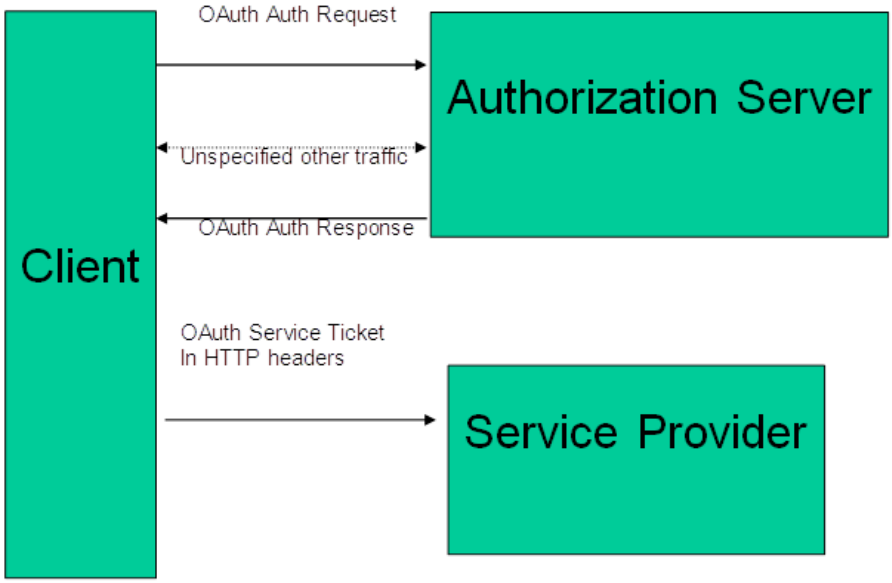
În continuare se va prezenta protocolul oauth și libraria care va fi folosita ca și client pentru serverul de Oauth.

4.4.7 Oauth, php-oauth client

Oauth este un protocol standard pentru autorizare. Prin intermediul acestuia utilizatorul poate oferi unei aplicații accesul la anumite resurse ale sale de pe o altă aplicație în numele său. Oauth funcționează pe protocolul HTTP.

Aplicația care cere accesul, face o cerere în numele utilizatorului către serverul de Oauth, care eventual va cere acestuia să se autentifice în aplicație și dacă acestea se efectuează cu succes răspunde cu un token(cheie) de acces pe care aplicația care a făcut cererea o va putea folosi în urmatoarele cereri pentru a accesa resursele acelui utilizator.

Pentru a putea realiza aceasta aplicatia va trebui să facă cereri HTTP, în a căror parametri să includă acel token primit de la serverul de Oauth.



Figură 4.16 – Oauth, interacțiune

Pentru a realiza această autorizare cu Oauth este nevoie și de o componenta care sa reprezinte clientul pentru serverul de Oauth. Întrucat pentru cele două platforme bazate pe git Github și Bitbucket exista deja o implementare partiala pentru sincronizare, sistemul Gitdone va folosi o biblioteca pentru această componenta pe care o va completa acolo unde este necesar.

Biblioteca se numeste php oauth client și va fi folosită prin includerea în aplicația cu ajutorul composer-ului, pe care l-am prezentat în secțiunile anterioare.

Tot cu Oauth va trebuie realizata sincronizarea contului de utilizator cu acela de pe platforma furnizoare de cloud, **DigitalOcean**. Clientul pentru aceasta componenta va trebui implementat integral.

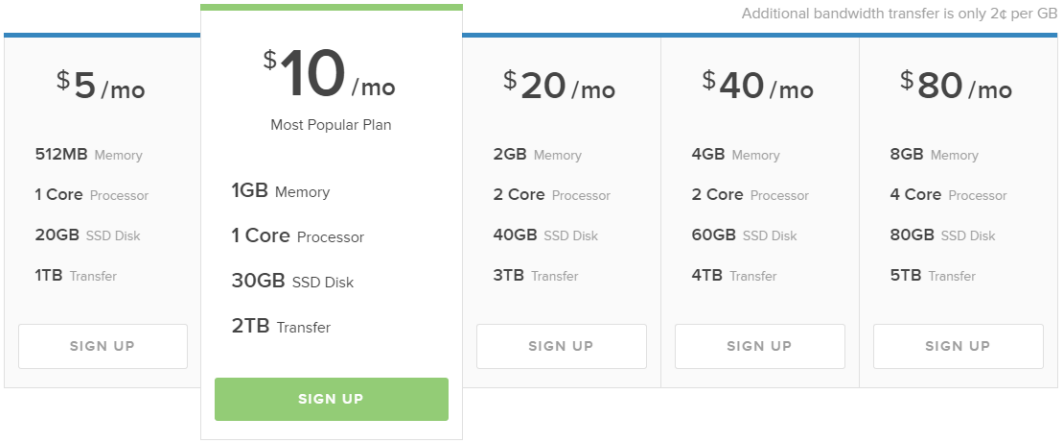
4.4.8 DigitalOcean

DigitalOcean reprezintă o platformă americană furnizoare de servicii cloud, relativ recent aparută pe piată și care s-a impus foarte repede prin simplitate. Aceasta platformă a aparut în 24 iunie 2011 și în prezent, la 3 ani de la lansare are aproximativ 2 milioane de vizitatori pe lună. În decembrie 2013 a fost declarat de catre site-ul Netcraft ca fiind „platforma furnizoare de servicii cloud cu cea mai rapida creștere din lume”, devansand liderul mondial (Amazon Web Services) ca și numărul de servere noi destinate aplicațiilor web. În iulie 2014 a devenit al cincilea furnizor de cloud din lume, surclasând Rackspace. [11]

În contextul sistemului de lansare Gitdone, DigitalOcean este foarte important, întrucât reprezinta furnizorul de servicii cloud. Acesta ofera o gama variata de servere, cu

o gama de preturi variată, în funcție de capacitatile serverului, constituind un furnizor de servicii cloud accesibil și dezvoltatorilor de aplicații web reduse, dar și un sistem pe care se pot baza companiile mari, care dezvolta aplicatii la scală mai mare.

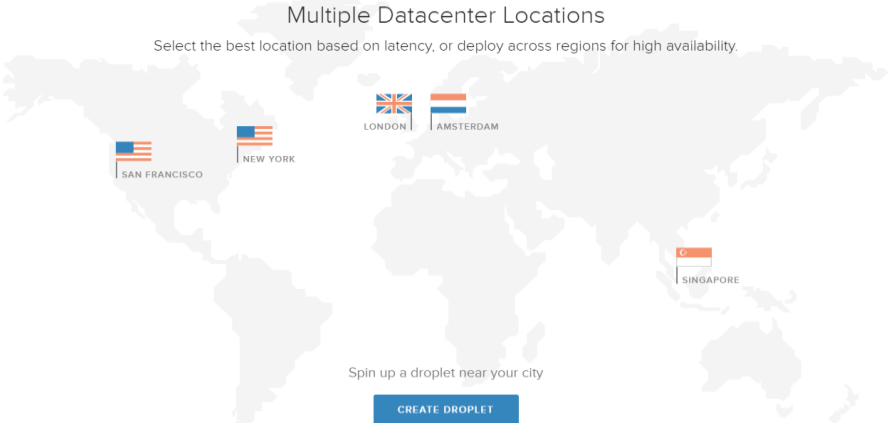
Din punct de vedere economic, principalul avantaj este că utilizatorul plătește serviciile consumate pentru fiecare oră în care le folosește, astfel că acesta este liber să-și modifice și ajusteze infrastructura oricând. [12]



Figură 4.17 – prețuri DigitalOcean

Desi prețurile sunt afisate pe luna, taxarea utilizatorului se face pe ora, astfel respectând principiile serviciilor cloud descrise mai sus. În funcție de resursele de care are nevoie și de constrangerile care i se aplică, utilizatorul se poate hotarî asupra tipului de server.

Un alt avantaj este faptul că locația serverelor este distribuită în mai multe zone geografice, existând servere în SUA, Europa și Asia, acest aspect fiind util clienților, pentru că-și pot alege serverele din zona cea mai apropiată pentru o conexiune rapidă sau își pot diversifica serverele, în funcție de nevoi și de restricțiile impuse de o anumită locație.



Figură 4.18 – Locații servere DigitalOcean

Pentru utilizarea de către dezvoltatori, platforma pune la dispozitie un API, prin intermediul caruia, pe baza demonstrarii identitatii, se pot accesa și crea resurse. Pentru a putea folosi conturile utilizatorilor se va folosi tehnica de Oauth pentru a obtine accesul acestora pe conturile de pe furnizorul de cloud.

Cererile catre API-ul oferit de DigitalOcean se vor face prin cereri HTTPS cu metoda GET, cererea avand parametri în link.

Un exemplu de cerere care obține toate serverele pe care le are un anumit utilizator:

GET https://api.digitalocean.com/v1/droplets?client_id=[id_client]&api_key=[api_key].

Cei doi parametri vor fi furnizați de utilizator în etapa de acordare de permisiuni.

Raspunsul sistemului va fi în format JSON și va arata cam asa:


```
{
"status":"OK",
"droplets":[
   {
    "id": 100823,
    "name": "test222",
    "image_id": 420,
    "size_id":33,
    "region_id": 1,
    "backups_active": false,
    "ip_address": "127.0.0.1",
    "private_ip_address": null,
    "locked": false,
    "status": "active",
    "created_at": "2013-01-01T09:30:00Z"
   }
 ]
}
```


Pe baza acestor cereri și raspunsuri, sistemul de lansare va interactiona cu platforma furnizoare de cloud, DigitalOcean.

Serverele care se vor crea pe aceasta platforma vor avea instalat sistemul de operare Linux, Ubuntu 12.10. Restul instalarilor și configurarilor se vor face prin intermediul partii de server a aplicației.

4.4.9 SSH, RSA – chei publice/private

Un protocol important care va fi folosit intr-o parte cheie a sistemului este protocolul **SSH (Secure Shell).** Acesta este un protocol care permite conectarea intre doua masini fizice aflate la distanta, în vederea executarii de comenzi pe masina tinta, de pe cea care initiază conexiunea. O caracteristica importanta a acestui protocol este aceea că realizează o conexiune securizată între cele doua dispozitive care comunică.

O astfel de cerere de comunicare SSH, initiază o conexiune pe protocolul de transport TCP, pe portul 22. Principala conditie ca aceste conexiuni sa functioneze este ca masina care initiază să permită trimiterea de pachete pe portul 22 spre exterior și să aibă instalat un client de ssh, iar mașina țintă sa permită primirea de pachete pe portul 22 din exterior și să aibă un client de ssh instalat.

Astfel de clienți SSH, sunt incluși default doar în sistemele de operare Unix și Mac OS, pe Windows lipsesc, Însă există anumite alternative. Nu se va intra în detalii cu privire la acest aspect deoarece sistemele de operare folosite de aplicații sunt bazate pe Unix.

Toate comenzile de configurare și instalare a uneltelor care să pregătească un server pentru funcționare se vor face prin intermediul acestui protocol.

Pentru a inița o astfel de conexiune se folosește comanda:

ssh nume_utilizator@host

host – poate fi un nume de domeniu, care se va converti prin DNS sau un ip.

După acest pas, se va trece la pasul de autentificare, existând mai multe variante posibile. Una din ele este introducerea parolei pentru user-ul respectiv, iar o alta variantă este adaugarea perechilor de chei publică – privată, generate prin algoritm de criptare RSA.

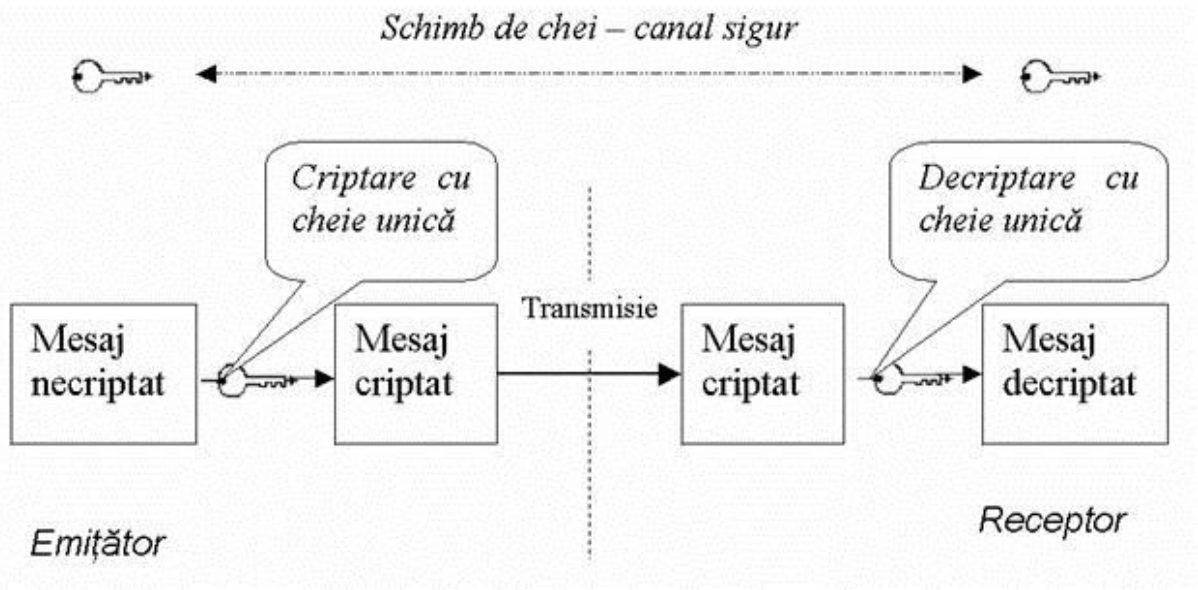
În continuare se va prezenta pe scurt acest algoritm de criptare.

RSA – reprezinta un algoritm criptografic publicat în 1978, dezvoltat de către autorii de la a caror initiale provine denumirea: **Ron Rivest, Adi Shamir și Leonard Adleman**. Este un algoritm folosit atât pentru criptare cât și pentru semnături electronice. [13]

Algoritmul se bazează pe folosirea a doua chei, una **publica** și una **privata.** Cheia privata este folosita pentru criptarea informatiei, iar cheia publica se potriveste peste aceasta cheie privata și se foloseste pentru decriptarea informatiei criptate cu aceasta. Cele doua chei sunt generate pe baza unor numere prime foarte mari și a unor operatii matematice asupra acestora. Aceste chei au o lungime de 1024, 2048 sau 4096 de biti, pentru a evita posibilitatea decriptarii fortate a mesajelor de catre utilizatorii rau intentionati.

Dificultatea decriptării forțate a acestor chei constă în dificultatea găsirii divizorilor primi ale numerelor foarte mari, algoritmii de rezolvare a acestei probleme a factorizarilor au o complexitate exponentiala. Astfel că pentru o cheie cu lungime suficient de mare timpul necesar pentru gasirea fortata, prin incercari a unei chei publice care sa se potrivească peste cea privată, este mult prea mare ca să merite acest efort.

În figură se ilustrează modelul criptarii RSA cu perechea de chei privată, publică.[14]



Figură 4.19 – Algoritmul de criptare RSA

5. Proiectare de detaliu și implementare

În acest capitol se va descrie partea de proiectare a aplicației, începând cu schema generală a aplicației, diagrama de deployment și continuând cu descrierea fiecărui modul în parte împreună cu diagramele de clase și secvențe, acolo unde e cazul. Tot în acest capitol va fi cuprinsă si structura bazei de date și a tabelelor.

5.1 Schema generală a aplicației

Sistemul de lansare Gitdone, este o aplicație web, astfel ca va reprezenta componenta de server, din comunicarea client – server tipică aplicațiilor web, iar partea de client va fi reprezentata de către un browser web instalat pe dispozitivul clientului. Acesta din urmă are nevoie doar de un astfel de browser si de o conexiune la internet pentru a putea accesa aplicația.

Ca orice aplicație web, are doua părți mari în care se împarte din punct de vedere al limbajelor de programare folosite și a scopului pe care îl realizează: partea de front-end și partea de back-end.

Pe partea de front-end se vor folosi tehnologiile specifice și anume **HTML, CSS** și **Javascript**, la acestea adăugându-se biblioteca **Twitter Bootstrap** pentru utilizarea de componente prefabricate, pe care am prezentat-o în capitolul anterior.

Pentru partea de pagini web, care vor reprezenta interfața grafică a aplicației, framework-ul pune la dispoziție un șablon pentru realizarea acestora, numit **blade.** Acesta permite organizarea mult mai ușoară a codului **HTML** cu cel de **php**, procedeu care se folosește fără excepție în aplicațiile web construite pe php ca limbaj de server. Deseori această intercalare a codului din cele două limbaje conduce la o dezorganizare a codului, reducându-i lizibilitatea si făcând din el un cod greu de întreținut. Cu ajutorul acestui șablon paginile arată mult mai organizate și se face ușor delimitarea între codul celor două limbaje.

Pe lânga acest rol, șablonul **blade** are rolul de a facilita crearea de ierarhii între fișierele **HTML**, permițând construirea de șabloane principale care să încadreze și să aranjeze paginile ca aspect (layout-uri), iar acestea să poată include și suprascrie pagini parțiale sau secțiuni.

Pe partea de back-end se va folosi limbajul de programare **php** și framework-ul **Laravel 4.0** peste acest limbaj. Acesta va impune aplicației anumite design pattern-uri si va permite utilizarea altor design pattern-uri adiționale în cazul în care este nevoie. Pentru a asigura încărcarea anumitor biblioteci și a dependențelor lor, dar și a unor pachete realizate pe module de către framework, se va folosi unealta numită **composer,** care la rândul ei a fost prezentată în secțiunea de protocoale și tehnologii.

Mai jos este prezentată schema generală a aplicației.

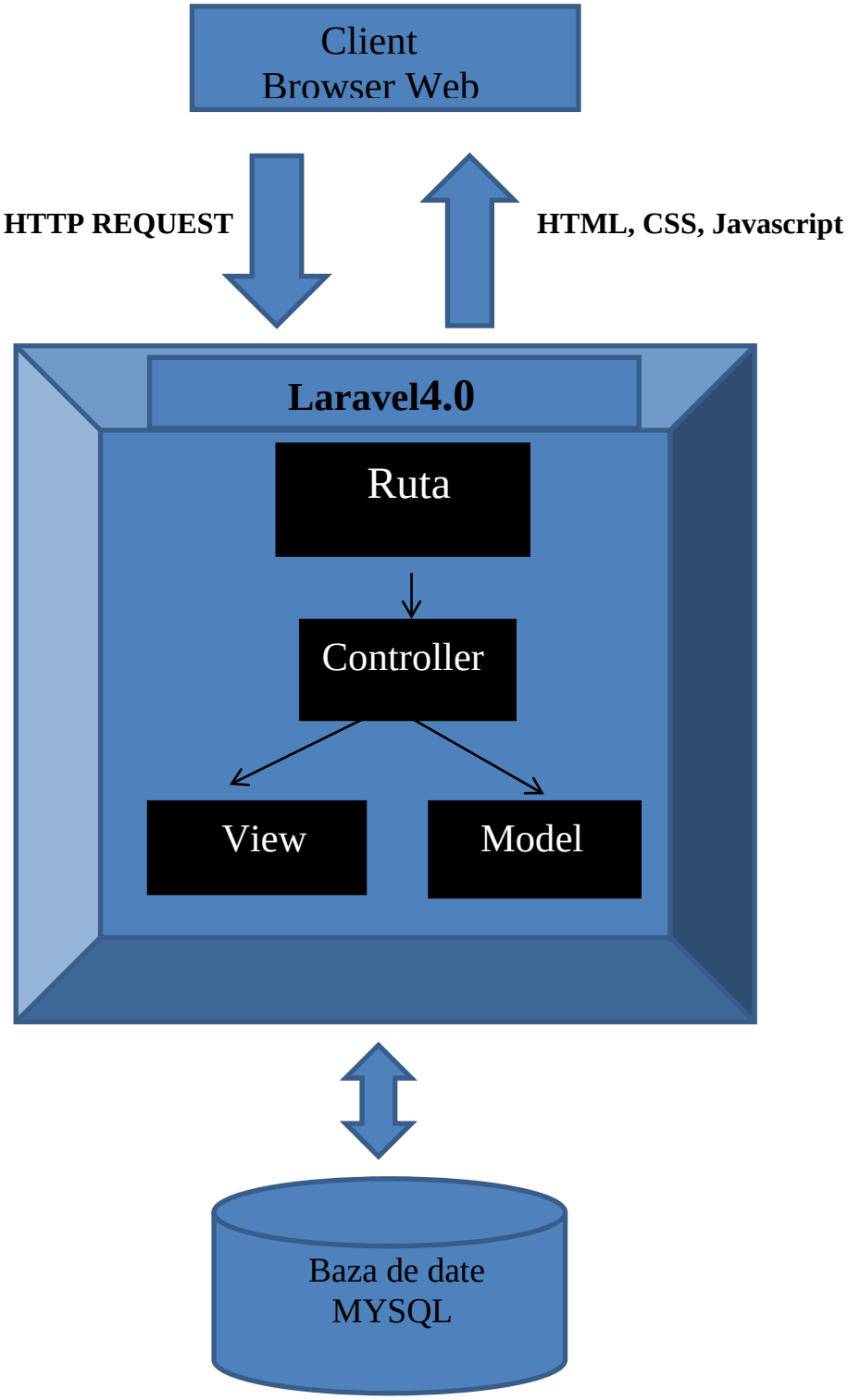


figura 5.1 – Arhitectura conceptuală

Pentru legătura între partea de modele și sursa de date, Laravel 4.0, oferă o tehnică care mapează coloanele din tabelele bazei de date relaționale în atribute pentru modelul corespondent. Accesul la sursa de date este învelit în entități de acest fel, care fac interacțiunea cu baza de date mult mai ușoară decât modele clasice.

Dupa cum se poate observa în diagrama de mai sus, framework-ul Laravel 4.0 este construit pe pattern-ul arhitectural Model-View-Controller, impunând aplicației această arhitectură.

Model-View-Controller(MVC) – reprezintă unul dintre pattern-urile arhitecturale ale limbajelor orientate pe obiect, pe acesta fiind fundamentate majoritatea aplicațiilor web, întrucât în contextul aplicațiilor web, separarea este foarte clară între cele trei părți ale aplicației.

Arhitectura MVC cere ca o aplicație vizuală să fie divizată în trei părți separate:

- Model - reprezintă partea care modeleaza problema pe care dorim sa o rezolvăm, acesta reprezintă intern datele problemei. Modelul trebuie sa fie independent atât de Controller cât si de View, astfel că modelul nu stie despre existența Controller-ului sau a View-ului si nu depinde în mod direct de nici unul dintre acestea, putând în schimb sa furnizeze acestora metode și să se modifice în funcție de cererile primite de la partea de control. Realizarea unui model bun al problemei pe care dorim să o rezolvăm este foarte importantă, fiind un prim pas spre rezolvarea corectă a problemei, indiferent din ce domeniu. Un model bine realizat conduce la o rezolvare mai rapidă si mai eficientă a problemei.

- View - reprezinta vizual datele si este interfața grafică cu ultilizatorul, un mijloc prin care utilizatorul poate comunica cu aplicația, poate introduce date de intrare si pe baza acestora să primească răspunsurile de care are nevoie. Partea de view(interfața grafică) trebuie sa fie prietenoasă și intuitivă pentru utilizator, să il indrume pe acesta spre o introducere corectă a datelor, pe baza cărora aplicația să furnizeze rezultatele dorite. Cu ajutorul view-ului utilizatorul poate să vadă ce face aplicația și să interacționeze cu aceasta prin intermediul formularelor și a altor elemente grafice.

- Controller – reprezintă partea aplicației care preia datele introduse de utilizator prin intermediul formularelor sau cererilor HTTP și le transpune în schimbări asupra modelului. Practic, controller-ul, dupa cum sugereaza si numele, detine controlul asupra aplicației, el preia informațiile de la utilizator și pe baza acestora decide ce urmează să facă restul componentelor. Este importantă separarea controller-ului de model pentru a face codul mai ușor de utilizat, depanat și refolosit, această separare conferind flexibilitate si robustețe.

-

5.2 Diagrama de deployment

Diagrama de deployment descrie nodurile fizice pe care se bazează aplicația denumite și artefacte ale sistemului și ilustrează interconectarea acestora. Scopul acestei diagrame este de a pune în evidență componentele fizice ale sistemului.

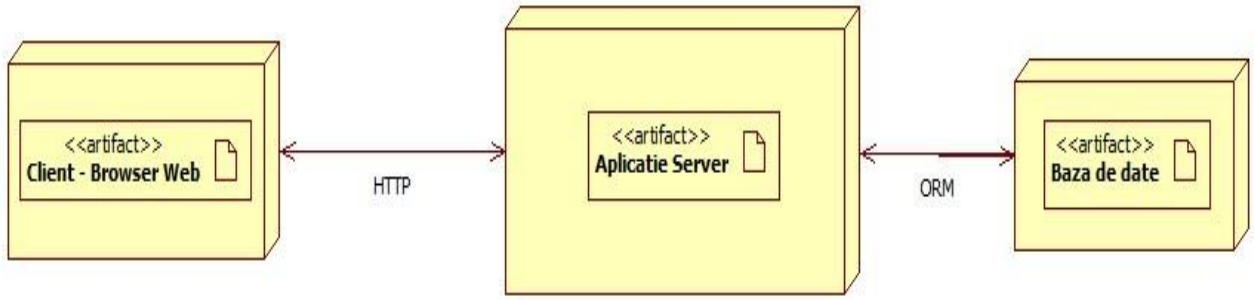


figura 5.2 – Diagrama de deployment

Ca și componente fizice sistemul are doua componente principale, aplicația în sine care va rula pe un server și va răspunde la cererile utilizatorilor, prin intermediul unui server web instalat, care va redirecta cererile HTTP pe portul 80 la aceasta aplicație, de unde conform figurii 5.1, cu arhitectura generală, se vor trata cererile, pentru fiecare rută sau link în parte sistemul face o anumită acțiune pe o metodă a unui controller, rezolvând astfel cererea și răspunzând utilizatorului.

Cea de-a doua componentă fizică a sistemului este reprezentată de client, care are nevoie de un browser web și o conexiune la internet pentru a accesa aplicația server. Datorită proprietăților serverului web, aplicația poate deservi oricâți clienți simultan, limitările depinzând doar de capacitatea de procesare a serverului pe care rulează.

Cea de-a treia componentă din diagrama de deployment este sursa de date, care în cazul sistemului de lansare a aplicatiilor web, Gitdone, va fi o bază de date SQL, gestionată prin intermediul sistemului de gestiune al bazelor de date, MySQL.

Aplicația este proiectată în așa fel, încât să suporte prezența bazei de date atât pe acelasi server cu ea, cât si pe un server la distanță, în funcție de cât succes și cât trafic ajunge să aibă aplicația, astfel lăsând o portiță pentru a scala aplicația din acest punct de vedere. Pentru început baza de date va putea sta pe același server ca și aplicația și va fi mutată pe un alt server dacă evoluția o va cere.

Tipic pentru o aplicație web clasică, sistemul va funcționa având la dispoziție cele 2 sau 3, dupa caz, componente fizice.

5.3 Structura de directoare a aplicației server

Un aspect al aplicației care merită luat în seamă este structura de directoare a acestuia, dirijată de structura framework-ului si ajustată pentru a corespunde propriilor particularități. [15]

Asadar, cele mai importante directoare și fișiere din directorul rădăcină al aplicației sunt următoarele:

- Fișierul **composer.json** – conține dependințele bibliotecilor pe care le va folosi aplicația, cât și directoarele ale căror clase din fișiere să fie încărcate automat la pornirea aplicației. Cea mai importantă parte din acest fișier este următoarea:

```
"require": {
        "laravel/framework": "4.2.*",
        "fzaninotto/faker": "dev-master",
        "way/generators": "dev-master",
        "robclancy/presenter": "1.1.*",
        "guzzlehttp/guzzle": "~4.0"
},
"autoload": {
        "classmap": [
                "server/app/commands",
                "server/app/controllers",
                "server/app/composers",
                "server/app/database/migrations",
                "server/app/database/seeds",
                "server/tests/TestCase.php"
        ],
        "psr-4": {
                "App\\": "server/app/domain"
        }
},
```

Astfel, atunci când se va rula comanda **composer install**, toate aceste biblioteci vor fi încărcate împreună cu dependințele lor, iar directoarele din autoload se vor încărca automat.

- Fișierul **bower.json** – are aceeași structură ca și composer.json, dar se referă la bibliotecile de pe partea de client.
- Directorul **public** – este unul din directoarele importante ale aplicației, întrucât aici se află fișierul care se va încărca prima dată din aplicației, **index.php**, iar de aici se controleaza mai departe cum se comportă aplicația, încărcând modulele framework-ului. Pentru o aplicație scrisă în php fără nici un framework, acest fișier se găsește în directorul rădăcină și de aici se va face manual partea de rutare (ce acțiune se va face pe fiecare link introdus de utilizator).

  Tot in directorul public se vor găsi imaginile și fișierele publice care vor putea fi accesate ca și resurse de către browser.

  Aici se găsesc și fișierele CSS si Javascript finale, pe care le incarcă browserul.
- Directorul **client** – aici se găsesc bibliotecile folosite pe partea de front-end și aici se creează inițial și fișierele CSS (pentru CSS se folosește un

preprocesor numit LESS) si Javascript (se folosește biblioteca jQuery), care după compilare se vor găsi in directorul **public,** prezentat anterior, de unde sunt pregătite să fie încărcate de către browser.

- Directorul **server** – este directorul din aplicație în interiorul căruia se află toată partea de programare orientată pe obiect si configurările necesare pentru ca framework-ul să funcționeze. Cel mai important director din interiorul acestuia este directorul **app**.

  Acest director are și el în interior o structură de directoare foarte stufoasă, dintre care se vor aminti pe scurt cele mai importante:

  - **routes.php** – fișierul care are înregistrate acțiunile pentru fiecare link accesat de utilizator
  - **controllers** – conține controllerele aplicației
  - **database –** conține resursele pentru creerea de tabele și popularea acestora
  - **start –** conține fișiere care ajută la configurarea sistemului
  - **config –** în acest director sunt principalele fișiere de configurări ale framework-ului
  - **composers –** conține fișiere în care se setează anumite date care vor fi trimise la pagini HTML, date care nu țin de modelul în sine, care este trimis oricum de către controller spre aceste pagini HTML.
  - **domain** – este directorul a cărui conținut va fi prezentat pe larg în continuare, întrucât aici se găsește logica aplicației și toate detaliile legate de modelare. Se va folosi tehnica de DDD -Domain Driven Development (Dezvoltare Dirijată de Domeniu), astfel că modelele nu vor sta toate la un loc, ci se vor împărti pe module și fiecare modul are clasele proprii, care îi definesc logica și funcționalitatea.

Pe lângă aceste directoare, framework-ul mai are foarte multe fișiere și directoare, însă nu se va intra în aceste detalii, întrucât conceptele importante au fost introduse pentru a înțelege fluxul aplicației. Ar mai fi de menționat faptul că pentru realizarea structurii bazei de date, creere de tabele și modificări asupra lor, se vor folosi așa numitele fișiere de **migrations**, iar pentru popularea acestora se vor folosi fișiere de **seeds,** acestea fiind denumirile specifice pe care le dă framework-ul.

Legat de directorul de configurări, înăuntrul acestuia se pot face o serie de configurări pentru mediul de producție (atunci când aplicația va rula pe server) și un alt set de configurări pentru mediul local, perioada în care se află în dezvoltare.

5.4 Modulele principale ale aplicației

În continuare vor fi prezentate în detaliu, cele mai importante module ale aplicației, diagramele UML pentru cele mai importante clase ale lor, cât și diagramele de secvențe pentru funcționalitatea cheie a fiecăruia din module.

Principalele module sunt:

- Modulul de autentificare și gestionarea de chei ssh
- Modulul de utilizare API DigitalOcean și gestionare servere
- Modului de configurări și instalări

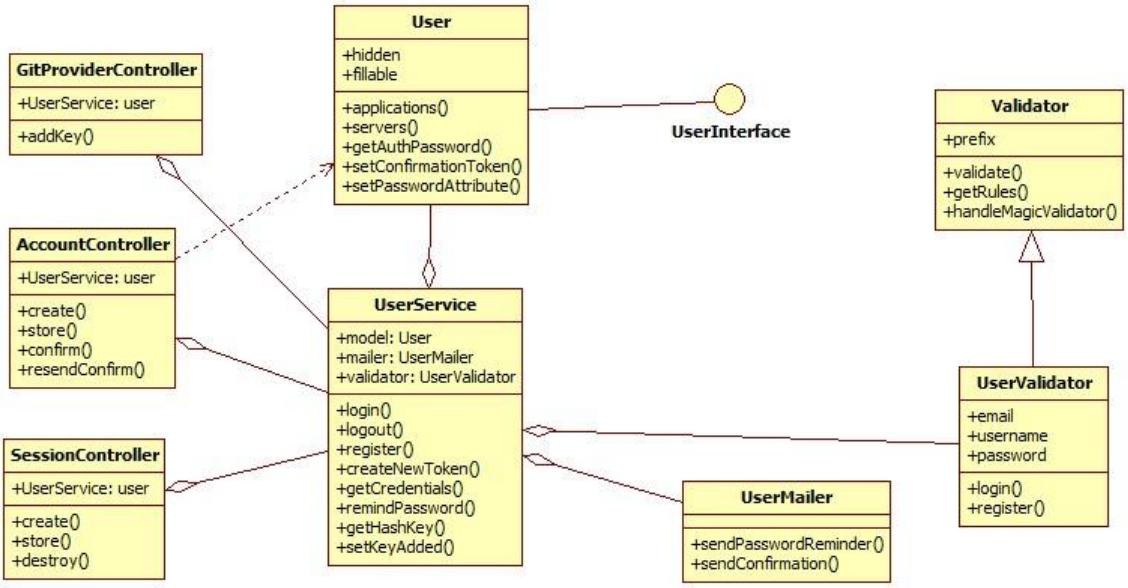
Se va prezenta fiecare modul în parte.

5.4.1 Modulul de autentificare și gestionarea de chei ssh

Această componentă realizează partea de înregistrare și autentificare a utilizatorilor și se ocupă de adăugarea de chei ssh pe contul utilizatorului pe una din platformele de găzduire a codului bazate pe git Github sau Bitbucket, pentru a autoriza astfel aplicația să aibă acces la resursele de pe acel cont al utilizatorului. Această parte s-ar putea realiza si prin intermediul protocolului Oauth, dar pentru această versiune a aplicației se va face prin adăugare de cheie publică, până când Bitbucket va face trecerea la Oauth 2.0, deoarece momentan nu suportă decât versiunea Oauth 1.0.

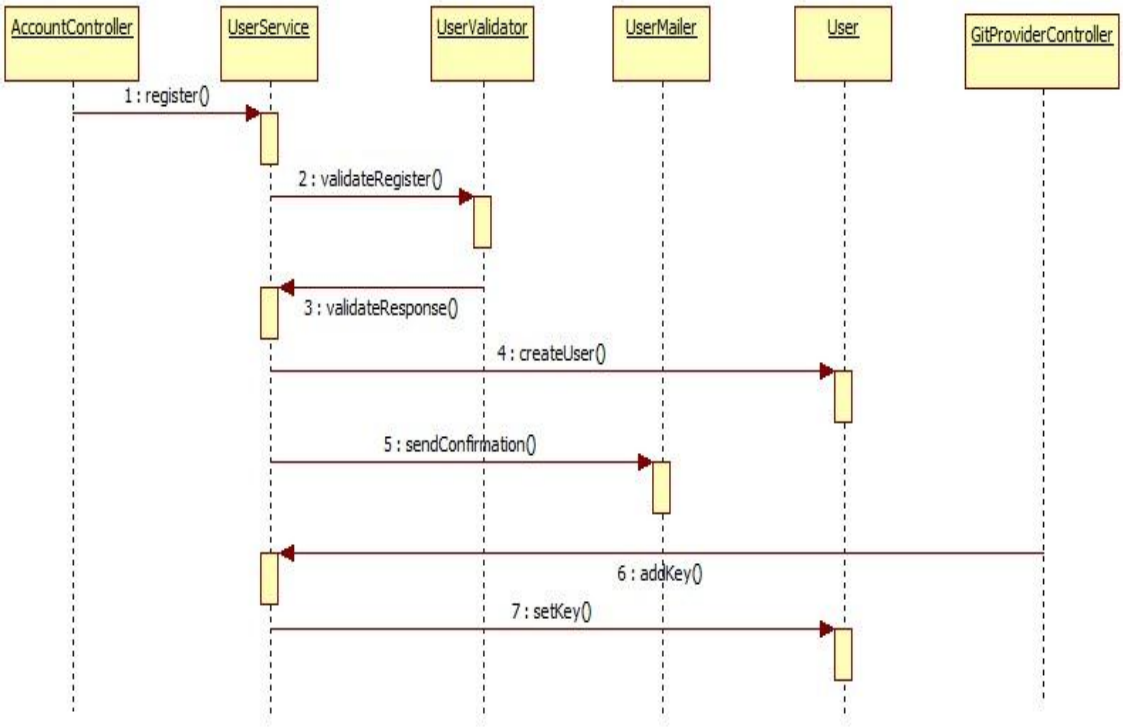
Este o componetă foarte importantă pentru aplicație, deoarece prin intermediului setului de clase care compun acest modul, se realizează acordarea de acces aplicației către resursele pe care le are utilizatorul în contul său de pe una din plaformele bazate peg it suportate. Pentru început vor fi suportate doar cele două mari astfel de platform, Bitbucket și Github, urmând să se adauge ulterior și alte platforme de acest fel, dacă utilizatori o vor cere.

Diagrama de clase pentru acest modul:



Figură 5.3 – Diagrama de clase pentru modulul de autentificare

Diagrama de secvente pentru autentificare si trimiterea email-ului de confirmare:



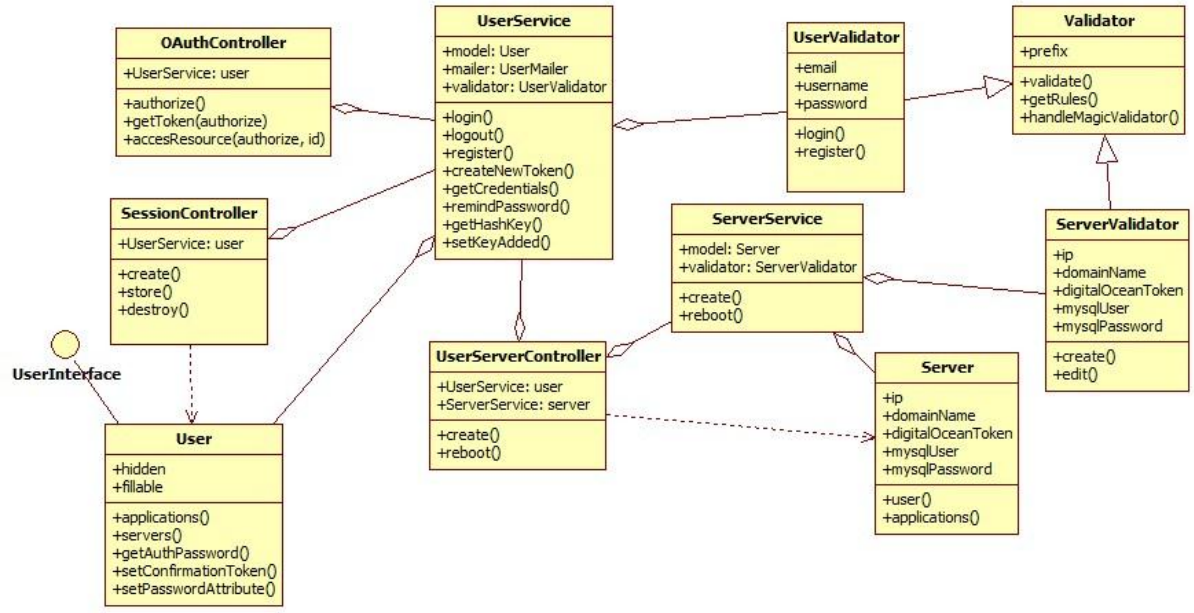
Figură 5.4 – Diagrama de secvențe pentru modulul de autentificare

Dupa cum se poate observa, după înregistrarea cu succes a utilizatorului și confirmarea adresei de email, acesta va adauga cheia SSH. În diagrama de secvențe acțiunile pornesc întotdeauna de la controllere, deoarece evenimentul care declanseaza aceste acțiuni este o cerere a utilizatorului pe un anumit link și partea de rutare va duce direct la o acțiune pe controller.

5.4.2 Modulul de utilizare API DigitalOcean și gestionare servere

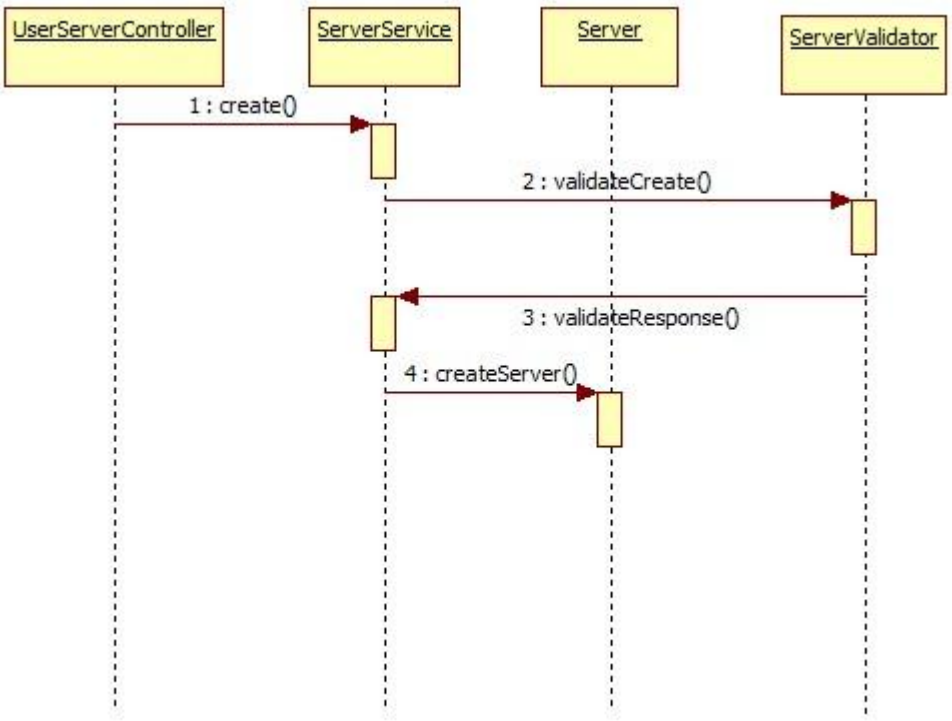
O altă componentă foarte importantă pentru aplicație este componenta care se ocupă de creerea și organizarea serverelor care se obțin prin intermediul furnizorului de servicii cloud, DigitalOcean și mai precis, prin intermediul API-ului pe care acesta îl pune la dispoziție. Utilizatorul va trebui să rezolve partea de acordare de acces la resursele de pe contul său, într-o primă fază adăugând cheile de acces în aplicație, iar când DigitalOcean va face trecerea la versiunea a 2-a a API-ului, aceasta operatie se va realiza prin intermediul protocolului Oauth 2.0. Aplicația va permite ambele metode.

Diagrama de clase pentru acest modul:



Figură 5.5 - Diagrama de clase pentru modulul de gestionare servere

Diagrama de secvențe:



Figură 5.6 – Diagrama de secvențe pentru modulul de gestionare servere

5.4.3 Modulul de configurări și instalări

Cea de-a treia componentă de bază a aplicației este realizarea etapelor de instalare și configurare a uneltelor și programelor necesare pentru ca aplicația utilizatorului să funcționeze. Aceste configurări se vor face pe serverul obținut de la DigitalOcean, care va avea doar sistemul de operare instalat, Ubuntu 12.10. Aceast modul al sistemului, are pe lângă setul de clase care asigură această etapă, un director cu rețete de configurări pentru diferitele unelte și programe de care are nevoie o aplicație web. De exemplu exista un fișier de configurări pentru serverul web, fie că e Apache, fie că e Nginx, sau există un fișier de configurări pentru php. Realizarea efectivă a operațiilor de configurare și instalare pe servere se va face printr-un alt modul intermediar, modulul de SSH, care pe scurt, folosește un client de ssh pentru a executa comenzi pe un server remote. Pe lângă acele rețete de configurare, care se vor copia direct în directoarele uneltelor și programelor care se vor folosi, sistemul are o serie de rețete de instalări de astfel de unelte și programe, care trebuie să se facă într-o anumită ordine și cu anumite constrângeri. Aceasta este partea de contribuții personale la aceast sistem, restul aplicației constând în integrarea de biblioteci și unelte open-source, care sunt puse împreună într-un mod eficace.

6. Testare și validare

Pentru partea de testare a aplicației, există mai multe variante disponibile. În cazul aplicațiilor web, cea mai sigură este testarea manuală a fiecărui link și a fiecărui scenariu în parte. Astfel, s-a testat ca fiecare funcționalitate să se realizeze și s-au căutat erori în fiecare scenariu. Pentru testarea atingerii tuturor obiectivelor aplicației s-a făcut testare la nivel de interfață, prin selectarea de scenarii de utilizare.

Testarea aplicației s-a realizat în mod manual pe baza scenariilor de test realizate în funcție de cerințele funcționale ale aplicației.

În continuare se va descrie un scenariu de test, de exemplu înregistrarea utilizatorului și adăugarea cheii ssh pe contul platformei bazate pe git.

Scenariu de test: Creare de cont și adăugarea unei chei ssh pe platforma bazata pe git.

Descriere: Utilizatorul trebuie sa își poată crea un cont cu ajutorul unei adrese de email valide și a unei parole de cel puțin șase caractere, împreună și cu un nume de utilizator care trebuie să fie unic în aplicație în momentul înregistrării.

Scenariu 1:
1: Utilizatorul nu are conexiune la internet
2: Incearcă să acceseze aplicația in browser

Rezultat așteptat: Browserul va afișa un mesaj de eroare.

Scenariu 2:
1: Utilizatorul are conexiune la internet
2: Există un firewall care blochează pachetele care ies pe portul 80 sau toate pachetele de ieșire
3: Incearcă să acceseze aplicația in browser
Rezultat așteptat: Browserul va afișa un mesaj de eroare.

Scenariu 3:
1: Utilizatorul are conexiune la internet
2: Nu există nici o regulă de firewall sau altceva care se blocheze cererile HTTP
3: Se încarcă aplicația in browser pe formularul de înregistrare
4: Utilizatorul nu completează unul din câmpurile de pe formular

Rezultat așteptat: Un mesaj de eroare de validare care să afișeze ce câmpuri trebuie corectate.

Scenariu 4:
1: Utilizatorul are conexiune la internet
2: Nu există nici o regulă de firewall sau altceva care se blocheze cererile HTTP

3: Se încarcă aplicația in browser pe formularul de înregistrare
4: Utilizatorul completează toate datele obligatorii
5: Utilizatorul adaugă un email invalid

Rezultat așteptat: Un mesaj de eroare de validare care să afișeze eroarea de validare a email-ului.

Scenariu 5:
1: Utilizatorul are conexiune la internet
2: Nu există nici o regulă de firewall sau altceva care se blocheze cererile HTTP
3: Se încarcă aplicația in browser pe formularul de înregistrare
4: Utilizatorul completează toate datele obligatorii
5: Utilizatorul adaugă o parolă prea scurtă

Rezultat așteptat: Un mesaj de eroare de validare care să afișeze eroarea de validare a parolei.

Scenariu 6:
1: Utilizatorul are conexiune la internet
2: Nu există nici o regulă de firewall sau altceva care se blocheze cererile HTTP
3: Se încarcă aplicația in browser pe formularul de înregistrare
4: Utilizatorul completează toate datele obligatorii
5: Utilizatorul introduce toate datele valide
6: Utilizatorul nu își confirmă adresa de email

Rezultat așteptat: Aplicația nu va oferi nici o funcționalitate, afisând doar un mesaj că trebuie făcută confirmarea adresei.

Scenariu 7:
1: Utilizatorul are conexiune la internet
2: Nu există nici o regulă de firewall sau altceva care se blocheze cererile HTTP
3: Se încarcă aplicația in browser pe formularul de înregistrare
4: Utilizatorul completează toate datele obligatorii
5: Utilizatorul introduce toate datele valide
6: Utilizatorul își confirmă adresa de email
7: Utilizatorul nu introduce cheia ssh pe contul său de pe platforma de git selectată sau o introduce greșit

Rezultat așteptat: Aplicația va încerca să facă o primă lansare și va primi o eroare de acces, astfel marcând contul utilizatoruluicăruia i se va cere să adauge din nou cheia ssh.

Scenariu 8:
1: Utilizatorul are conexiune la internet
2: Nu există nici o regulă de firewall sau altceva care se blocheze cererile HTTP
3: Se încarcă aplicația in browser pe formularul de înregistrare

4: Utilizatorul completează toate datele obligatorii
5: Utilizatorul introduce toate datele valide
6: Utilizatorul își confirmă adresa de email
7: Utilizatorul introduce cheia ssh pe contul său de pe platforma de git selectată

Rezultat așteptat: Utilizatorul va trece la partea de creere servere în cloud.

7. Manual de instalare și utilizare

Acest capitol cuprinde cerințele pe care trebuie să le îndeplinească sistemul utilizatorului pentru a putea utiliza aplicația. Fiind o aplicație web, care va fi lansată, sistemul permite utilizatorilor accesarea acesteia doar folosind un browser web. Astfel această etapă este simplificată foarte mult, resursele necesare fiind minime.

7.1 Resurse hardware

Singurele resurse hardware de care are nevoie clientul pentru a accesa aplicația sunt deținerea unui calculator și a unei conexiuni la internet.

7.2 Resurse software

Ca și resurse software, utilizatorul trebuie sa aibă un browser web instalat, de obicei la instalarea sistemului de operare există cel puțin un astfel de browser pe sistem.

Pentru simplitate, indiferent de sistemul de operare folosit, utilizatorul poate să downloadeze browserul Google Chrome, folosind browserul default instalat pe sistem și mergând la adresa https://www.google.com/chrome/browser/#eula unde există o versiune disponibilă pe fiecare sistem de operare.

Pentru o utilizare cât mai ușoară a aplicației utilizatorul ar trebui să aibă un cont valid pe platforma furnizoare de servicii cloud, DigitalOcean și un cont pe una din platformele de găzduire a codului bazată pe git: Github - https://github.com/ sau Bitbucket - https://bitbucket.org/ . Odată ce a realizat acestea, utilizatorul poate folosi aplicația.

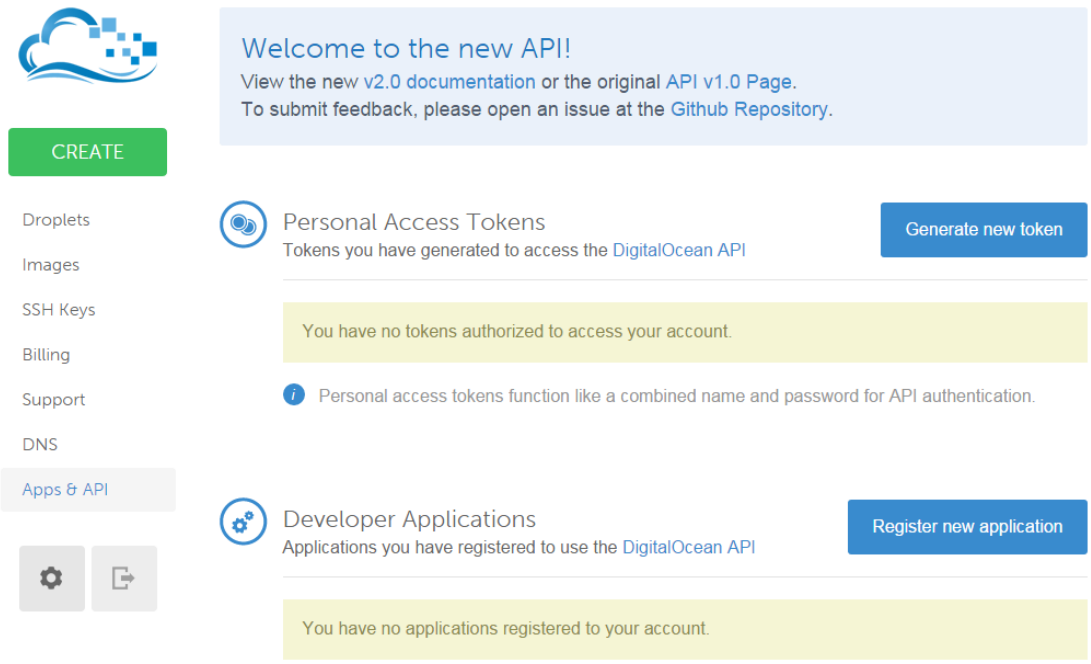
7.3 Manual de utilizare

Din punctul de vedere al utilizării aplicației, utilizatorul trebuie întâi să se înregistreze în sistem pe http://gitdone.deiu.ro/register, unde va trebui să completeze formularul de înregistrare.

După acest pas, utilizatorul trebuie să confirme adresa de email, unde va merge pe linkul din corpul mesajului, care îl va aduce apoi din nou în aplicație.

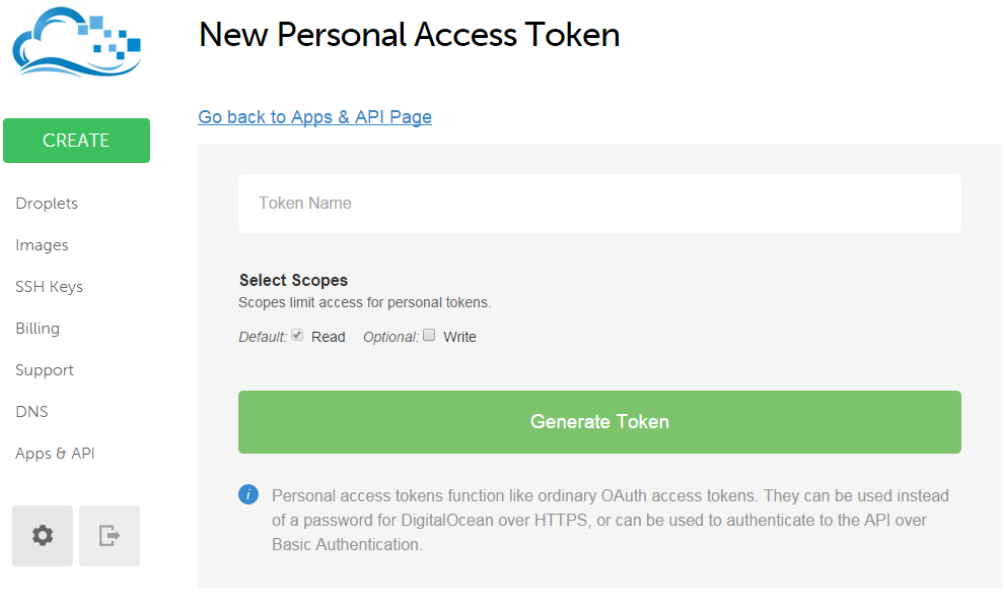
Următorul pas este adăugarea cheii ssh, care apare pe pagina principală a aplicației pe una din platformele precizate in secțiunea anterioară, redirectarea spre acestea se va face de către aplicație după ce utilizatorul selectează unul din aceste două opțiuni și apasă butonul Go. Ajuns pe site-ul uneia din cele două platforme utilizatorul eventual se autentifică și apasă apoi ADD SSH KEY, unde va da o denumire cheii și va pune cheia copiată din pagina principală a site-ului Gitdone, după care apasă save.

Pentru a putea folosi serviciile aplicației, utilizatorul are nevoie și de un cont pe platforma furnizoare de servicii cloud, DigitalOcean. Astfel, dacă nu are deja un astfel de cont acesta își creează unul și trece de pașii de confirmare a indentității pe care îl îndrumă platforma. După aceasta trebuie să meargă în meniu pe **Apps&Api**, unde va apărea pagina următoare:



Figură 7.1 – pagina de setări pe DigitalOcean

După ce apasă pe **Generate new token** va apărea următoarea pagină:



Figură 7.2 - Generarea unei chei de acces pe DigitalOcean

Aici utilizatorul va da o denumire cheii, va selecta opțiunea write și va apăsa Generate Token. Utilizatorul trebuie să stocheze undeva acel token pe care-l generează DigitalOcean, întrucât nu va mai fi afișat. Apoi va trebui să-l adauge în formularul de pe pagina principală a aplicației și să apese butonul Save. După această etapă utilizatorul poate folosi serviciile aplicației Gitdone.

8. Concluzii

Acest ultim capitol prezintă rezultatele obținute și măsura în care obiectivele propuse au fost atinse. Tot aici, sunt enumerate dezvoltările ulterioare ale aplicației.

8.1 Realizări

Aplicația Gitdone, a fost dezvoltată pentru a permite dezvoltatorilor de aplicații web să realizeze etapa de lansare a aplicațiilor într-un mod automat, selectând o serie de configurări care consideră că se potrivesc aplicației.

Lansarea acestor aplicații se va face pe servere din cloud, obținute de la furnizorul de cloud DigitalOcean. Momentan aplicația permite doar utilizarea acestui furnizor.

Pentru moment s-a realizat un sistem care permite aceasta etapă de lansare pentru aplicațiile dezvoltate pe platforma wordpress și pentru cele dezvoltate pe framework-ul Laravel pe php. În ceea ce privește procurarea codului sursă pentru aplicațiile pe care utilizatorii doresc să le lanseze, sistemul permite folosirea celor mai mari 2 platforme de găzduire a codului bazate pe git, Github si Bitbucket.

Astfel sistemul facilitează munca dezvoltatorilor de aplicații web și deși datorită limitărilor impuse asupra tehnologiilor cu care sunt scrise aplicațiile pe care sistemul le poate lansa, aplicațiile realizate pe aceste tehnologii reprezintă un procent semnificativ, astfel că sistemul are un public destul de larg căruia i se adresează.

8.2 Dezvoltări ulterioare

Ca și dezolvtări ulterioare, aplicația are ca scop în primul rând să diversifice furnizorii de servicii cloud care pot fi folosiți pentru obtinerea de servere, pentru a nu obliga utilizatorul să aibă conturi pe sisteme multiple de acest fel.

O altă componentă care poate fi dezvoltată este integrarea platformelor care găzduiesc cod sursă bazate pe git, incluzând si alte platforme pentru a cuprinde un număr cât mai mare de utilizatori.

Pentru partea de lansare efectivă idealul ar fi adăugarea unui număr cât mai mare de tehnologii pe care pot fi realizate aplicațiile care se lanseaza prin sistemul Gitdone. Pentru aceasta dezvoltatorul trebuie să înțeleagă în prealabil cum se realizează aplicațiile pe tehnologiile țintă. În viitorul apropiat s-ar putea ajunge cu ușurință la a suporta toate framework-urile cunoscute pe limbajul php, deoarece sunt similare ca și mod de lucru cu Laravel, cel suportat momentan și platforma Drupal care este similară cu Wordpress, suportat momentan.

Integrarea sistemelor bazate pe git, dar și a platformelor bazate pe cloud ar fi ideal să se facă la fiecare prin intermediul protocolului Oauth 2.0. Astfel că dupăce acestea vor face trecerea la o nouă versiune de API care să suporte acest protocol, sistemul va putea fi dezvoltat în acest sens.

8.3 Contribuții personale

Contribuțiile personale în acest proiect îl reprezintă rețetele de configurare și de instalare a resurselor și uneltelor pe servere. Configurările sunt cele care se copiază direct în fișierul de configurări a soft-ului țintă, iar rețetele de instalare se referă la găsirea acestor soft-uri si unelte, cât și a ordinii în care trebuie instalate pentru a oferi mediul în care să ruleze aplicațiile utilizator, furnizorul de cloud oferind servere care au instalat doar sistemul de operare.

Bibliografie

[1] Istoric Internet, http://ro.wikipedia.org/wiki/Internet

[2] Articol tipuri de cloud, https://www.linkedin.com/today/post/article/20140325024745-246665791-hybrid-cloud-is-the-best-option

[3] Articol protectia datelor, http://ec.europa.eu/justice/data-protection/article-29/documentation/opinion-recommendation/files/2012/wp196_ro.pdf

[4] Laborator ARC, UTCN – prezentare cloud.

[5] http://www.gartner.com/technology/reprints.do?id=1-1UKQQA6&ct=140528&st=sb

[6] TCP, http://ro.wikipedia.org/wiki/Protocol_de_control_al_transmisiei

[7] HTTP, http://www.ustudy.in/def_HTTP

[8] WEB Servers, http://ro.wikipedia.org/wiki/Server_web#mediaviewer/File:LAMP_software_bundle.svg

[9] PHP, http://ro.wikipedia.org/wiki/PHP

[10] Laravel 4.0, http://laravel.com/docs

[11] DigitalOcean, http://en.wikipedia.org/wiki/DigitalOcean

[12] DigitalOcean, https://www.digitalocean.com/

[13] RSA, http://ro.wikipedia.org/wiki/RSA

[14] Securitate, http://www.scritub.com/stiinta/informatica/Aspecte-de-securitate-n-retele221113154.php

[15] Andrei Canta – BoilerPlate, https://github.com/deiucanta/boilerplate

Glosar de termeni:

API – Application Programming Interface
CGI - Common Gateway Interface
CMS - Content Management System
HTTP - Hypertext Transfer Protocol
HTTPS – HyperText Transfer Protocol Secure
JSON - JavaScript Object Notation
MD5 - Message Digest Algorithm 5
ORM - Object-relational mapping
PHP - Php: Hypertext Preprocessor
SGBD - Sistem de Gestiune a Bazelor de Date
WWW - World Wide Web
TCP - Transmission Control Protocol
UML - Unified Modeling Language