**UNIVERSITETI I PRISHTINËS**

**Fakulteti i Inxhinierisë Elektrike dhe Kompjuterike në Prishtinë**

**Inxhinieri Kompjuterike**



**PRAKTIKA PROFESIONALE**

**Punim Seminarik**

**Autentikimi dhe Autorizimi me RoR**

**Studenti: Mentorë:**

Synim Selimi Prof. Asoc. Dr. Kadri Sylejmani

ID: 180714100399

Prishtinë, Qershor 2018

**PUNIM SEMINARIK**

**AUTENTIKIMI DHE AUTORIZIMI ME ROR**

**Synim Selimi**

**Universiteti i Prishtinës, FIEK**

05.06.2018

Prishtinë, Kosovë

**ABSTRAKT**

Aktivitetet kryesore, përvoja dhe njohuritë e fituara gjatë punës praktike janë temë qendrore e këij punimi. Ndër zërat kryesorë të punimit bëjnë pjesë edhe përshkrimi i metodologjisë agjile të menaxhimit të projekteve, përshkrimi i platformave më të reja për organizimin dhe matjen e progresit në projekte dhe analiza e teknologjive dhe modeleve softuerike të implementuara gjatë angazhimit në projektet e punës praktike. Në kuadër të shpërfaqjes së teknologjive të shfrytëzuara për zhvillim të aplikacioneve në ueb është e pashmangshme përmendja dhe përshkrimi i shkurtër i RESTful modelit, MVC modelit dhe i karakteristikave themelore të kornizës punuese RoR, simotrave të saj dhe gjuhës programuese Ruby. Veprimtaria e përshkruar për zhvillimin e aplikacioneve në ueb kryesisht përfshinë ambientimin me kornizën punuese, krijimin e skeletit të aplikacionit, integrimin e moduleve të autentikimit dhe autorizimit dhe vendosjen e panelit administrues për administrimin dhe monitorimin e të gjitha të dhënave ekzistuese në aplikacion me mundësi zgjerimi për çfarëdo shtese që i vendoset aplikacionit, varësisht prej kërkesave për rritje apo shkallëzim softuerik. Si pjesë e dytë po aq e rëndësishme është edhe përshkrimi dhe ilustrimi i koncepteve të autentikimit dhe autorizimit si dhe aplikimi i tyre specifik në kornizën e shfrytëzuar të ueb aplikacioni. Përmbledhurazi punimi shpërfaq projektet në të cilat është punuar gjatë praktikës profesionale, aplikimin praktik të njohurive të fituara për zhvillim softuerik në ueb, format themelore të modelimit arkitektural të ueb aplikacioneve moderne, shfrytëzimin e metodave agjile për menaxhim dhe platformave të zgjedhura për koordinimin e vazhdimësisë dhe progresit të projektit.

**Fjalët kyçe**: Platforma Agjile, Ruby on Rails, Autentikim, Autorizim, Zhvillim në Ueb.

Përmbajtja

[1. Konceptet Themelore 4](#_Toc520632120)

[2. Metodat 6](#_Toc520632121)

[3. Përshkrimi i Aktiviteteve Kryesore 6](#_Toc520632122)

[3.1 Hyrja dhe Motivimi 6](#_Toc520632123)

[3.2 Përshkrimi i Problemit 7](#_Toc520632124)

[3.3 Platformat e Monitorimit dhe Menaxhimit të Projektit 7](#_Toc520632125)

[3.4 Autentikimi dhe autorizimi përmes RoR 7](#_Toc520632126)

[3.5 Administrimi i të dhënave përmes RoR 10](#_Toc520632127)

[4. Përfundimet 11](#_Toc520632128)

[Akronimet dhe Terminologjia 11](#_Toc520632129)

[Shtojcat 11](#_Toc520632130)

[Dallimet dhe konfliktet mes versioneve në Rails 11](#_Toc520632131)

[Referencat 12](#_Toc520632132)

[Bibliografia 12](#_Toc520632133)

Figurat dhe Tabelat

[Fig. 1 Përshkrim gjenerik i MVC modelit (Armel, 2014) 4](file:////Users/selimi/Desktop/CE/FIEK/Semestri%20i%20gjashtë/Lëndët/Praktika%20Profesionale/Punimi/Punimi_si_Moster/Punimi_Seminarik_Moster.docx#_Toc520632134)

[Fig. 2 Arkitektura e kornizës Ruby on Rails (Tk, 2017) 5](file:////Users/selimi/Desktop/CE/FIEK/Semestri%20i%20gjashtë/Lëndët/Praktika%20Profesionale/Punimi/Punimi_si_Moster/Punimi_Seminarik_Moster.docx#_Toc520632135)

[Fig. 3 Paraqitja skematike e ndërmjetësuesve mes Rails aplikacionit dhe ueb serverit 5](file:////Users/selimi/Desktop/CE/FIEK/Semestri%20i%20gjashtë/Lëndët/Praktika%20Profesionale/Punimi/Punimi_si_Moster/Punimi_Seminarik_Moster.docx#_Toc520632136)

[Tab. 1 CRUD veprimet e zakonshme mbi një resurs në Rails RESTful aplikacion (Bigg, et al., 2018) 6](#_Toc520632137)

[Fig. 4 Principi i autentikimit të principalit para autorizimit për qasje (Coulouris, et al., 2011) 8](file:////Users/selimi/Desktop/CE/FIEK/Semestri%20i%20gjashtë/Lëndët/Praktika%20Profesionale/Punimi/Punimi_si_Moster/Punimi_Seminarik_Moster.docx#_Toc520632138)

[Fig. 5 Përmbajtja e Gemfile me të gjitha gem paketat dhe versionet që kërkohen për Rails aplikacion 8](#_Toc520632139)

[Fig. 6 Formulari për kyçjen e klientit dhe formulari për përditesimin e të dhënave të klientit 9](file:////Users/selimi/Desktop/CE/FIEK/Semestri%20i%20gjashtë/Lëndët/Praktika%20Profesionale/Punimi/Punimi_si_Moster/Punimi_Seminarik_Moster.docx#_Toc520632140)

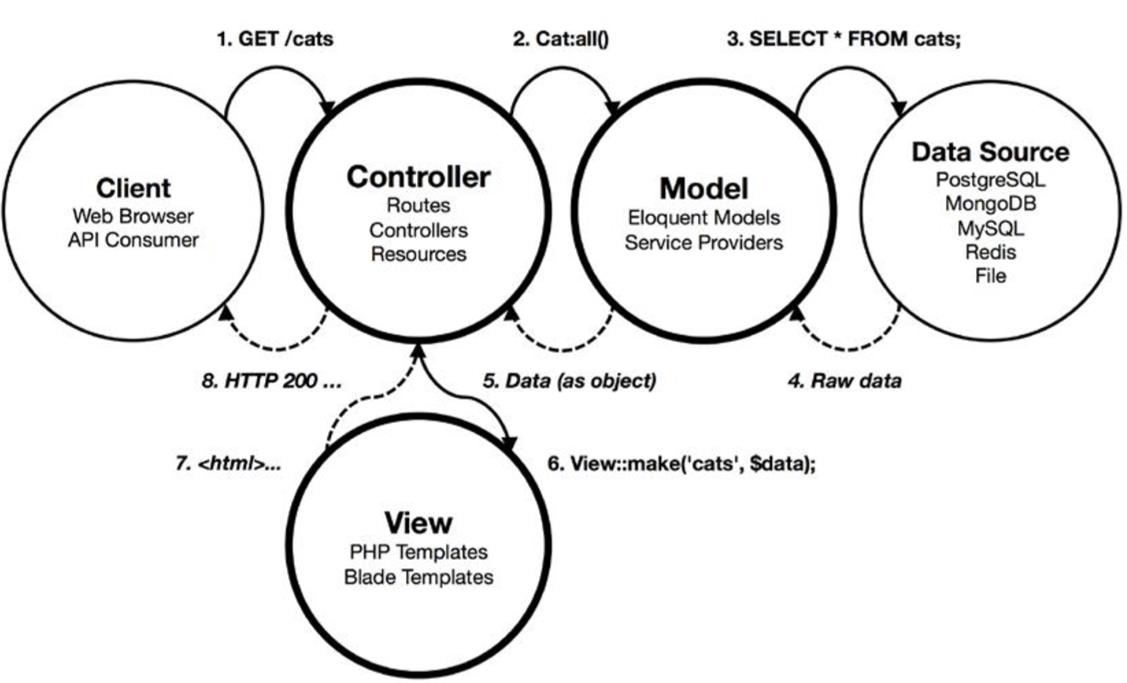
[Fig. 7 Pamja në server pas autentikimit të klientit nëpërmjet kredencialeve 10](#_Toc520632141)

# 1. Konceptet Themelore

Në mënyrë që të mund t’i përshkruajmë detalet më të avancuara dhe më specifike nga platforma si autentikimi, autorizimi dhe administrimi, fillimisht do ta paraqesim një pasqyrë të përgjithshme të teknologjisë RoR dhe të modeleve themelore mbi të cilat bazohet funksionimi i saj.

Ruby on Rails është një kornizë zhvillimore për ueb aplikacione e shkruar në gjuhën programuese Ruby[[1]](#footnote-1). Kjo platformë është specifikisht e dizajnuar për ta lehtësuar, thjeshtësuar dhe përshpejtuar programimin e ueb aplikacioneve, duke supozuar mbi praktikat më të mira zhvillimore dhe duke krijuar konventa programimi prej tyre (Bigg, et al., 2018). Rails është softuer i opinionuar; kjo nënkupton se mënyrën e funksionimit dhe vendimet për format më të mira dhe qasjet që duhet të ndërmirren për ta zgjidhur një problem gjatë zhvillimit softuerik i ka të ndikuara nga opinionet e krijuesve të kornizës zhvillimore. Andaj shpesh gjatë shfletimit të literaturës mbi këtë platformë zhvillimore ose edhe nga shfrytëzues të saj përmendet togfjalëshi “*Rruga e Rails*”[[2]](#footnote-2), duke dashur të përshkruhet pikërisht kjo formë e paradefinuar e zhvillimit me pretendimin për të arritur efiçiencë dhe produktivitet më të lartë, tipare për të cilat mund të dëshmohet lehtë nëse punohet me Rails për së afërmi.

Kjo kornizë zhvillimore ndërtohet mbi dy parime udhëzuese dhe nëpërmjet dy modeleve arkitekturale themelore, influenca e të cilave shprehet në secilën fazë të zhvillimit. Parimet themelore theksojnë së zhvilluesi nëpërmjet kësaj kornize arrin që minimalisht ta përsëritë kodin, maksimalisht ta rishfrytëzojë atë dhe nëpërmjet konventave dhe rregullave të paracaktuara ta reduktojë kohën e nevojshme për konfigurim[[3]](#footnote-3) (Bigg, et al., 2018). Ndërkaq modelet arkitekturale që e përbëjnë bërthamën e Rails janë MVC dhe REST modeli.



MVC është një prej mostrave arkitekturale nëpërmjet të cilës arrihet principi i dizajnit për ndarjen e përgjegjësive (*separation of concerns*). Kësisoji ofrimi i çdo funksionaliteti apo shërbimi në një aplikacion ndahet në komponentë, secila me përgjegjësi të caktuara për domenin specifik, të cilit i përket. MVC modeli në rastin e Rails e shpërndan punën në këto tri komponente kryesore:

* Modeli (*Model*) – i mirëmban marrëdhëniet mes objekteve dhe bazës së të dhënave dhe merret me validimin, asocimin mes objekteve, transaksionet, etj.;

Fig. 1 Përshkrim gjenerik i MVC modelit (Armel, 2014)

* Pamja (*View*) – ofron një reprezentim të të dhënave në format specifik, varësisht prej input-it të shfrytëzuesit apo kontrollave të kontrollerit;
* Kontrolleri (*Controller*) – komponenta brenda aplikacionit e cila e orienton trafikun, shfrytëzon dhe kërkon të dhëna nga baza e të dhënave në njërën anë dhe i organizon ato në forma specifike për t’iu përshtatur pamjeve në anën tjetër.

Rails, Laravel, ASP.NET, CakePHP, Codelgniter, etj. përveç këtyre tri komponenteve e shfrytëzojnë edhe një komponentë të katërt – *Routing*, e cila e mundëson hartimin dhe shpërndarjen e HTTP kërkesave drejt kontrollerëve dhe metodave specifike (Armel, 2014). Një përshkrim gjenerik i MVC modelit është dhënë në Fig. 1 ndërsa arkitektura specifike e kornizës Rails është dhënë më poshtë në Fig. 2.

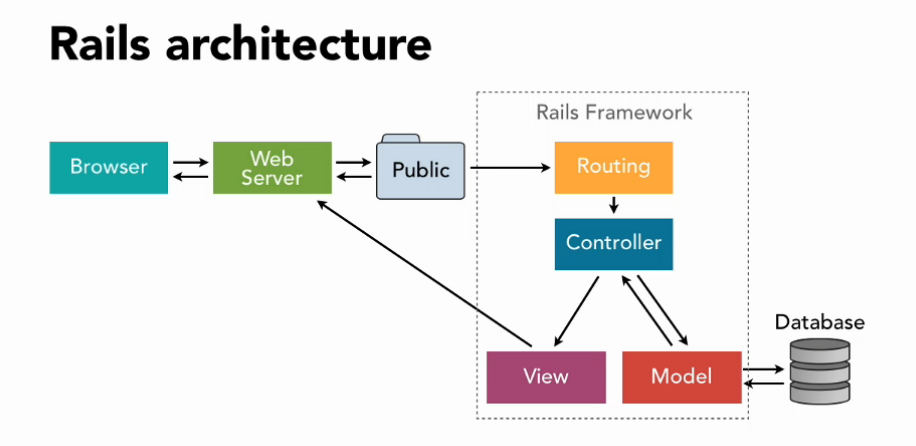


Fig. 2 Arkitektura e kornizës Ruby on Rails (Tk, 2017)

Vlen të theksohet se hostimi i ueb aplikacioneve në Rails apo platforma të ngjashme zhvillimore përveç ueb server-it (*web server*) për njohjen e HTTP kërkesave kërkon edhe server të aplikacionit (*application server*) të ndërtuar në gjuhën programuese specifike, e cila i kryen të gjitha detyrat, përkthimet dhe funksionalitetet që ofrohen prej kornizës zhvillimore. Në rastin tonë ndërmjetësimin (*middleware*) mes ueb serverit dhe funksionaliteteve të aplikacionit tonë në gjuhën Ruby e kryen i ashtuquajturi *Rack Webserver[[4]](#footnote-4)*. Një ilustrim i kësaj arkitekture është dhënë në Fig. 3.



Fig. 3 Paraqitja skematike e ndërmjetësuesve mes Rails aplikacionit dhe ueb serverit

Krahas MVC modelit, përgjatë tërë strukturës së kornizës Ruby on Rails vërehet edhe shfrytëzimi i theksuar i REST modelit. REST është një stil arkitektural për dizajnimin e ueb aplikacioneve, i cili në kontekstin e Rails nxit orientimin drejt përfaqësimit të të gjitha entiteteve dhe komponenteve të ueb aplikacionit si resurse të cila mund të krijohen, lexohen, përditësohen dhe fshihen[[5]](#footnote-5) (Hartl, 2016). Ky model arkitektural është mjaft i volitshëm, meqenëse këto operacione mbi resurset e ueb aplikacionit korrespondojnë plotësisht si me CRUD operacionet e bazave të të dhënave, ashtu edhe me HTTP metodat e kërkesave: POST, GET, PATCH, DELETE. Shembuj të entiteteve të një ueb aplikacioni që mund t’ju nënshtrohen këtyre operacioneve dhe REST modelit janë shfrytëzuesi, klienti, postimi, komenti, etj. E përbashkëta e të gjithave është se janë të krijueshme, të lexueshme, etj. Një ilustrim i implementimit të këtij modeli në Rails është dhënë në Tab. 1.

| **HTTP Verb** | **Path** | **Controller#Action** | **Arsyeja** |
| --- | --- | --- | --- |
| GET | /users | users#index | Shfaq listën e të gjithë shfrytëzuesve |
| GET | /users/new | users#new | Ktheje një HTML formularë për krijimin e një shfrytëzuesi të ri |
| POST | /users | users#create | Krijo një shfrytëzues të ri |
| GET | /users/:id | users#show | Shfaq pamjen e shfrytëzuesit me **:id** |
| GET | /users/:id/edit | users#edit | Kthe një HTML formularë për ndryshimin e të dhënave të shfrytëzuesit |
| PATCH/PUT | /users/:id | users#update | Përditëso të dhënat e shfrytëzuesit me **:id** |
| DELETE | /users/:id | users#destroy | Fshije shfrytëzuesin me **:id** |

Tab. 1 CRUD veprimet e zakonshme mbi një resurs në Rails RESTful aplikacion (Bigg, et al., 2018)

# 2. Metodat

Përpilimi i punimit dhe shtjellimi i koncepteve të përvetësuara për Rails është bazuar në pluralizmin metodologjik hulumtues e shpjegues duke përfshirë referimin, përmbledhjen dhe elaborimin e literaturës paraprake, metodat e mbledhjes së informacionit, metodat përshkruese, metodat ilustruese dhe metodat krahasuse.

# 3. Përshkrimi i Aktiviteteve Kryesore

## 3.1 Hyrja dhe Motivimi

Evoluimi i botës së WWW me shpejtësi marramendëse, nëpërmjet ndryshimit dhe përmirësimit të formave të interaksionit me shfrytëzues dhe shfaqjes së teknologjive të reja ka mundësuar transformimin e ueb faqeve në ueb aplikacione. Ueb aplikacionet sot kanë një sërë funksionalitetesh, të cilat kohë më parë ishin të mundshme vetëm nëpërmjet teknologjive më konvencionale të zhvillimit softuerik, siç janë *stand-alone* aplikacionet. Aktivitetet kryesore gjatë praktikës profesionale mundësuan pikërisht nisjen dhe zhvillimin e disa moduleve të një ueb aplikacioni nëpërmjet shfrytëzimit të RoR dhe teknologjive të zakonshme për prezentim në ueb. Modulet e zhvilluara janë mjaft gjenerike dhe mund të ripërdoren në çdo aplikacion tjetër, ku nevojitet autentikimi i shfrytëzuesve, autorizimi për qasje në resurse specifike dhe administrimi i të dhënave dhe përmbajtjes së krijuar nga shfrytëzuesit.

Ueb aplikacioni në të cilin është punuar gjatë praktikës ekziston si version më i vjetër, i cili për arsye objektive është konsideruar se ka nevojë për tipare të reja dhe përmirësime në performance, shfrytëzueshmëri. Kësisoji modulet e krijuara mundohen t’i plotësojnë pikërisht këto kërkesa, duke u mbështetur në karakteristikat që i ofron Rails.

## 3.2 Përshkrimi i Problemit

Në qendër të angazhimit qëndron krijimi i moduleve të cilat ofrojnë autentikim, autorizim dhe administrim në ueb aplikacion së bashku me të gjitha CRUD operacionet për shfrytëzuesin. Kjo nënkupton se në përfundim është kërkuar zhvillimi i të gjitha faqeve, formularëve, metodave dhe kontrollerëve të cilët ofrojnë mundësi regjistrimi, verifikimi/riverifikimi me email, modifikimi, fshirje, etj. të llogarive të krijuara. Duke ndjekur REST modelin, llogaritë e krijuara paraqesin resurset, të cilat u nënshtrohen CRUD operacioneve nëpërmjet HTTP kërkesave adekuate, rrugëtimit deri te metodat e duhura të kontrollerëve dhe kryerjes së veprimeve të caktuara në bazë relacionale për ta përfunduar kërkesën me sukses.

Sfidë e veçantë shpërfaqet edhe krijimi i panelit për administrim, pasi aty duhet të pasqyrohen të gjitha modelet ekzistuese të të dhënave me mundësi të plotë për lexim, modifikim, fshirje dhe në përgjithësi administrim. Fatmirësisht shfrytëzimi i Rails teknologjisë e lehtëson edhe këtë pengesë nëpërmjet shfrytëzimit te kodeve të gatshme me burim të hapur (open-source), të cilat me pak konfigurim e kryjnë tërë punën e lodhshme shumë shpejt dhe me shumë korrektësi[[6]](#footnote-6).

Sa të rëndësishme janë detalet e kërkesave dhe projektit në fjalë, po aq janë edhe qasjet dhe metodologjia e shfrytëzuar për menaxhimin e progresit, mbikëqyrjen e projektit dhe procesin e inxhinierisë softuerike. Në kuadër të punës praktike është kërkuar edhe ambientimi me metodologjinë agjile të punës, duke shfrytëzuar një kombinim të dy aplikimeve më të përhapura agjile: Scrum dhe Kanban.

## 3.3 Platformat e Monitorimit dhe Menaxhimit të Projektit

Përgjatë praktikës profesionale, komunikimi dhe monitorimi i progresit dhe punës është kryer duke shfrytëzuar një bashkësi platformash të dizajnuara me qëllim lehtësimin e këtyre proceseve. Procesi i ndjekur për zhvillimin softuerik është bazuar krejtësisht në dy aplikime të principeve agjile[[7]](#footnote-7). Nga Scrum janë shfrytëzuar kryesisht menaxhimi i kohës së zhvillimit dhe liferimit softuerik në sprinte, programimi në çift (*pair programming*) dhe ndarja e roleve dhe detyrave, ndërsa nga Kanban është huazuar vizualizimi i punës, përmirësimi i vazhdueshëm dhe zbërthimi i punës në Kanban tabelë.

Në anën tjetër, në projekte ku punohet vazhdimisht dhe paralelisht nga pjesëtar të ndryshëm të ekipit, është mjaft e rëndësishme forma që zgjidhet për kontrollimin e versioneve (*Version Control*) dhe krijimin e një vendndodhje të përbashkët ku ruhet puna në largësi e jo lokalisht në secilën pajisje të zhvilluesve. Për këtë qëllim gjatë zhvillimit të projektit është shfrytëzuar *Git* dhe *GitLab* si depo kryesore ku ruhet projekti dhe ku vendosen përditësimet sipas sistemit të kontrollit që ofron Git.

## 3.4 Autentikimi dhe autorizimi përmes RoR

Zakonisht zhvilluesit e aplikacioneve i referohen autentikimit dhe autorizimit bashkërisht me njërën nga këto dy terme. Në fakt që të dy konceptet janë plotësisht ortogonale dhe të pavarura, edhe pse që të dyja janë qëndrore në dizajnin e sigurisë së sistemeve dhe dështimi i cilësdo prej tyre i hap rrugë komprometimit të sistemit.

Autentikimi paraqet procesin e vërtetimit të identitetit të palës komunikuese. Parimisht kjo bëhet nëpërmjet verifikimit të kredencialeve apo ofrimit të çfarëdo sfide të dizajnuar në aplikacion me qëllimin e autentikimit të shfrytëzuesit. Përveç verifikimit të identitetit të palës, asaj i nevojitet edhe lejimi i qasjes në resurse të caktuara, varësisht prej politikave të qasjes që i implementon aplikacioni. Pra autorizimi, si komplement i drejpërdrejtë i autentikimit, paraqet procesin e kontrollit të qasjes në shërbime, resurse apo komponente varësisht prej identitetit të autentikuar dhe politikave të qasjes. Në Fig. 4 është paraqitur në formë gjenerike qasja e autoritetit të autentikuar (*principalit*) në një objekt në largësi (programi i server-it) me të drejta të qasjes të përcaktuara prej matricës së qasjes në server (*Access rights*). Për informacione të zgjeruara mbi modelin e sigurisë dhe principin e autentikimit referojuni (Coulouris, et al., 2011).

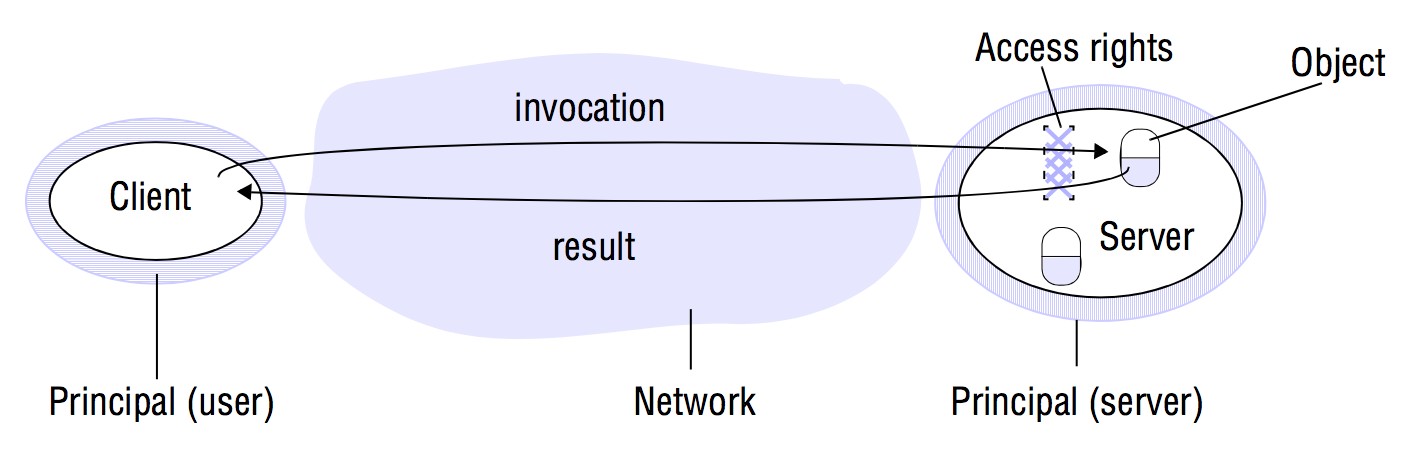


Fig. 4 Principi i autentikimit të principalit para autorizimit për qasje (Coulouris, et al., 2011)

Implementimin e këtyre dy koncepteve në Rails aplikacion gjatë punës praktike e kemi kryer duke e shfrytëzuar Devise gem paketën, e cila i ofron të gjitha tiparet e kërkuara dhe është mjaft e përshtatshme për shfrytëzim. Në Fig. 5 është paraqitur Gemfile me gem paketat e shfrytëzuara për implementimin e Rails aplikacionit. Ndër to vërehen Devise dhe ActiveAdmin, e para për mundësimin e autentikimit dhe autorizimit, ndërsa e dyta për vendosjen e panelit të administratorit. Sigurisht ky është vetëm hapi i parë i implementimit; për funksionalizim të plotë është dashur të instalohet paketa, të gjenerohen kontrollerët, pamjet dhe të modelet e resurseve që do të kenë llogari dhe kredenciale për qasje dhe të shtohen HTTP paths ne fajllin konfigurues të Routing komponentës së Rails (*routes.rb*).

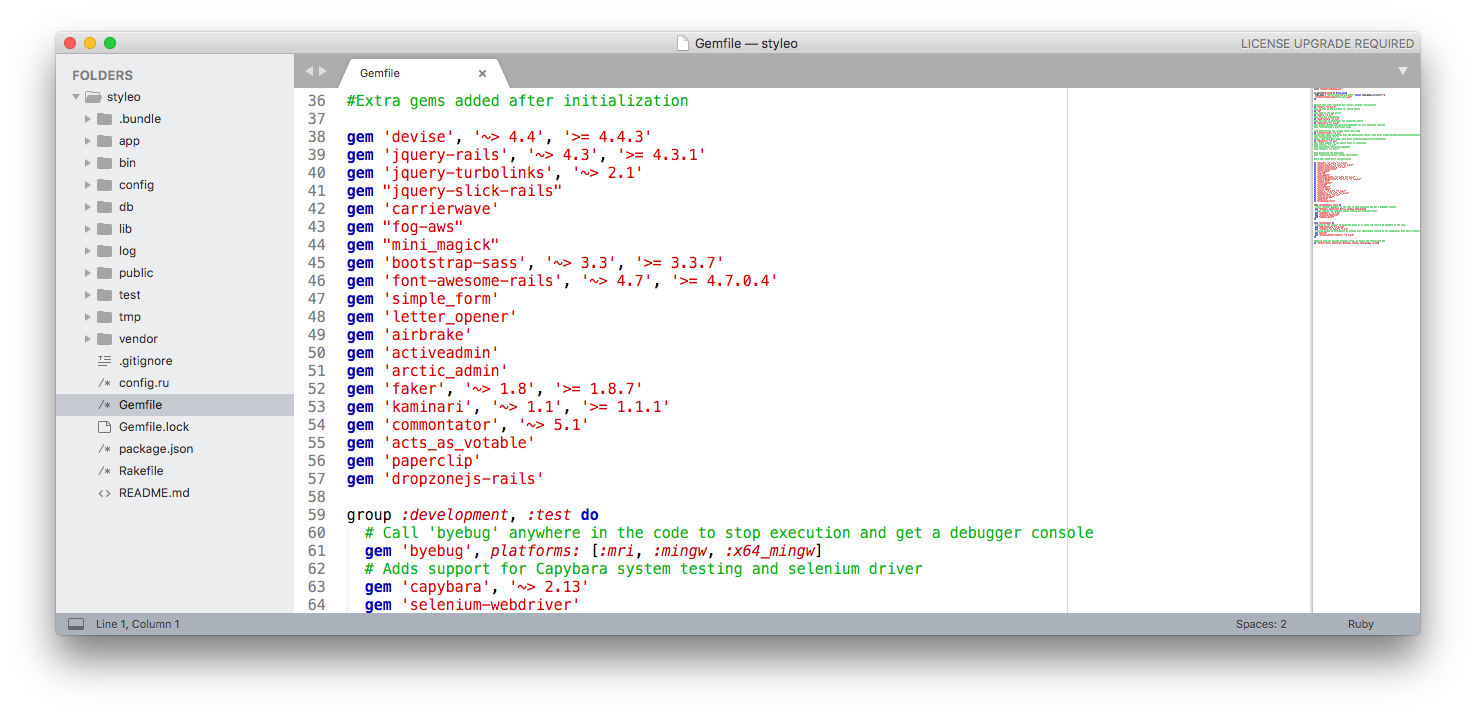


Fig. 5 Përmbajtja e Gemfile me të gjitha gem paketat dhe versionet që kërkohen për Rails aplikacion

Tipar dallues i kornizës Rails është krijimi i skeletit te aplikacionit gjithmonë me new në terminal edhe shfrytëzimi i gjeneratorëve për krijimin e pjesës së punës që është rutinore dhe gati gjithmonë përsëritet. Kësisoji ne duke i shfrytëzuar gjeneratorët e Devise i krijojmë fajllat që na mundësojnë kostumizimin e kodit për modelin, formularët, kontrollerët dhe pamjet e resursit në fjalë (Plataformatec, 2016).

**$** rails new myApplication

**$** rails generate devise Client

**$** rails generate devise:views clients

**$** rails generate devise:controllers clients

Parimisht nevojitet vetëm krijimi i skeletit të aplikacionit me new, shtimi i resursit “clients” në *routes.rb* dhe gjenerimi i modelit për entitetin “Client” ku vendosen fjalët kyçe të Devise paketës dhe e gjithë pjesa tjetër e kodit është konvencionale, mirëpo meqë neve na është dashur ta modifikojmë dhe përmirësojmë sipas kërkesave dhe tipareve specifike, ishte e domosdoshme gjenerimi i të gjitha elementeve.

Përparësi e kësaj forme të implementimit, përveç shpejtësisë dhe nevojës së vogël për testim, janë edhe metodat e gatshme të cilat i ofron Devise për implementimin e autorizimit me vetëm një rresht kod në kontrollerët specifik. Meqenëse çdo HTTP kërkesë kalon nëpër kontrollerin specifik, në krye të kontrollerit që kërkon autorizim shtohet ky rresht kodi:

before\_action :authenticate\_client!

Me kaq pak punë ne e zgjerojmë nevojën për autorizim në secilën faqe që na nevojitet. Veprimi që kryhet në momentin kur mungon autorizimi apo autentikimi është ridrejtimi i komunikimit në faqen e kyçjes së klientit. Në Fig. 6 është paraqitur formulari për kyçje të klientit dhe formulari i krijuar për editimin e të dhënave të klientit pas autentikimit/autorizimit. Këta formularë janë rezultati i modifikimit të kodit të gjeneruar me Devise sipas shijes së kërkuar. Në Fig. 7 është dhënë pamja në server pas kyçjes së klientit në fjalë. Duhet po ashtu të theksohet fakti se vendosja e sesionit, cookies dhe të gjitha tiparet e tjera që e ruajnë gjendjen kryhen automatikisht nga Devise paketa.

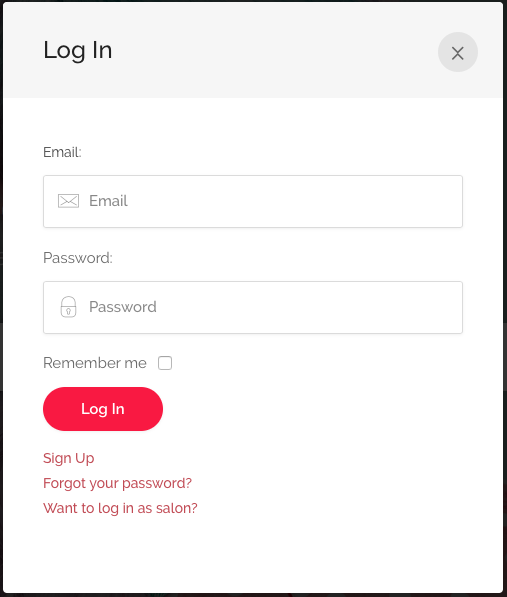
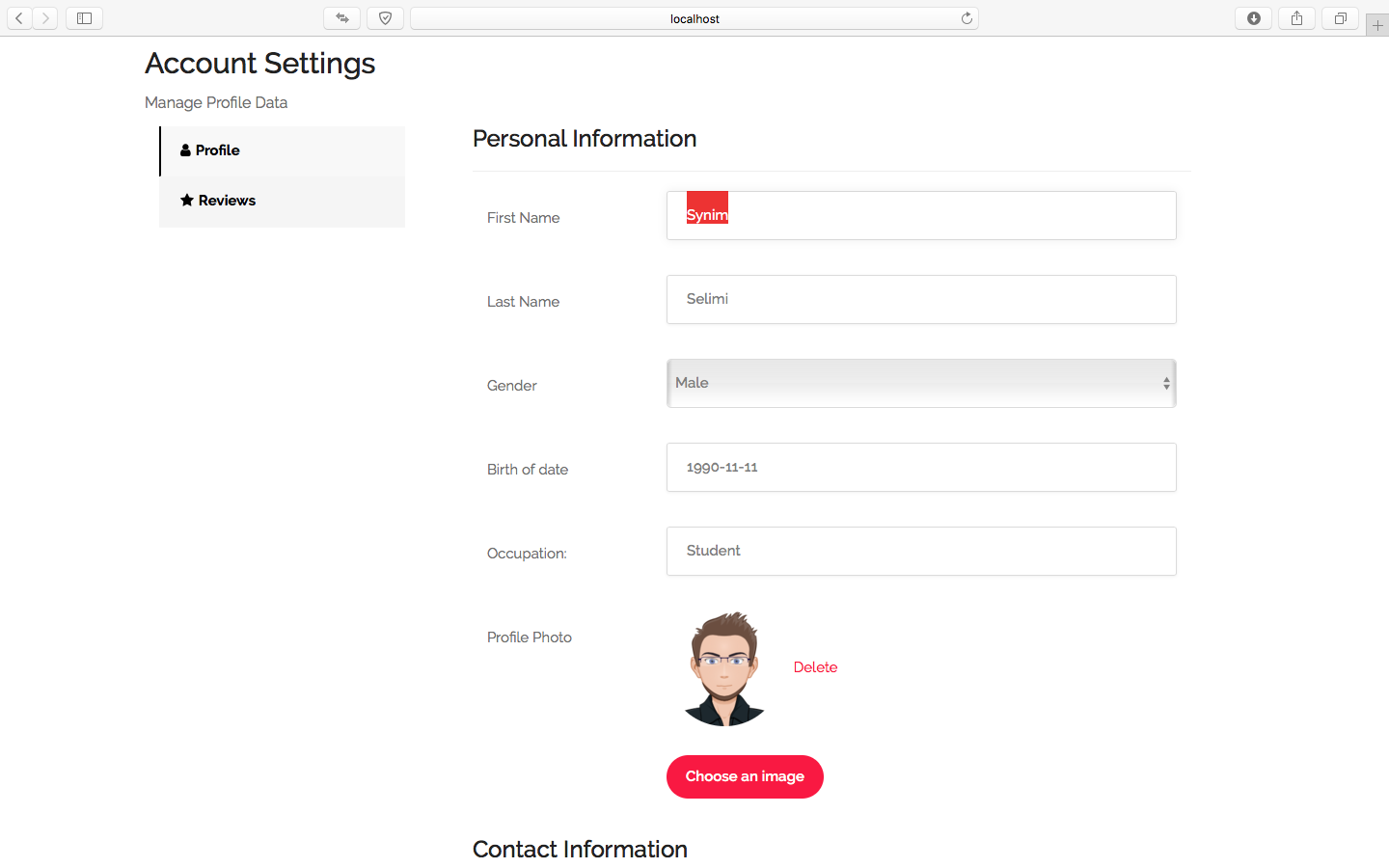


Fig. 6 Formulari për kyçjen e klientit dhe formulari për përditesimin e të dhënave të klientit

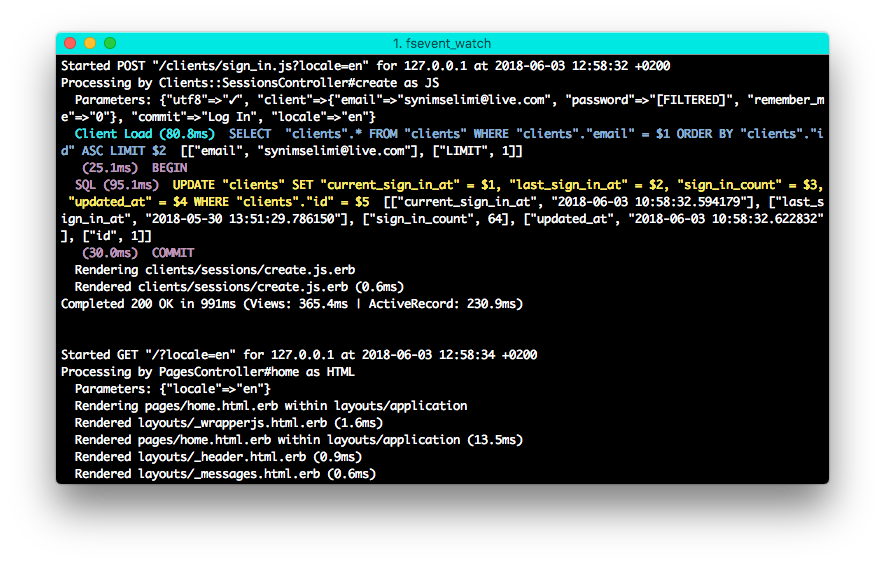


Fig. 7 Pamja në server pas autentikimit të klientit nëpërmjet kredencialeve

## 3.5 Administrimi i të dhënave përmes RoR

Në mënyrë krejtësisht analoge është bërë edhe përfshirja e ActiveAdmin gem paketës, e cila e krijon automatikisht panelin e administratorit për modelet e të dhënave të cilat dëshirojmë t’i përshtatim (Bell & VersaPay, 2011). Pas vendosjes së emrit të gem paketës në Gemfile dhe instalimit të ActiveAdmin, e kryjmë gjenerimin e fajllit që do të shërbejë si ndërfaqe për modifikimin dhe përshtatjen e tipareve të ActiveAdmin për modelin e resursit aktual. Nëpërmjet rreshtit vijues në terminal krijohet fajlli app/admin/clients.rb.

**$** rails generate active\_admin:resource Client

Parimisht ky fajll nuk ka nevojë të modifikohet fare, pasi që rëndom i ofron shumicën e funksionaliteteve që i duhen administrimit të modelit “Client” duke përfshirë të gjitha CRUD operacionet. Mirëpo për shkaqe të kërkesave mbi shfrytëzueshmërinë dhe përshtatjes së plotë për domenin ku do të shfrytëzohet aplikacione, është dashur të kryhen disa modifikime p.sh.: filtrimi sipas parametrave të caktuar, kushtëzimi i mbushjes së disa fushave, internacionalizimi i panelit të administratorit në dy gjuhë, etj.

Në panelin e krijuar të administrimit shfaqet indeksi i modelit “Client” me një mizëri veprimesh që mund të kryhen mbi instancat e modelit “Client”. Një bashkësi e veprimeve janë të gjeneruara automatikisht duke ndjekur stilin REST arkitektural, ndërsa të tjerat janë si rezultat e përshtatjes që e kemi bërë ndaj kërkesave. Vlen të theksohet se ekzisojnë edhe disa hapa konfiguruese, të cilat për qëllim të thjeshtësisë janë kaluar gjatë këtij shpjegimi. Për detale më specifike të implementimit të këtyre paketave referojuni (Bell & VersaPay, 2011) dhe (Plataformatec, 2016).

# 4. Përfundimet

Ky punim i përmbledh aktivitetet dhe angazhimin kryesor gjatë kryerjes së praktikës profesionale. Mes të tjerash u përmendën gjuha programuese Ruby, korniza zhvillimore Rails, aplikimet praktike SCRUM dhe Kanban të principeve agjile dhe u diskutua forma e implementimit të autentikimit, autorizimit dhe administrimit në ueb aplikacione. Përtej të gjithë përvojës teknike dhe teknologjike të fituar, do t’i theksoja si me shumë rëndësi njohjen për së afërmi me format agjile të menaxhimit të progresit dhe ambientimin me VCS (Git, GitLab, Github, etj.). Në projektet të përmasave moderne, zhvillimi i softuerit kryhet nga ekipe me madhësi marramendëse dhe në këto raste sistemi që shfrytëzohet për kontrollim të versioneve dhe aftësia e zhvilluesve për ta përdorur atë luajnë rol po aq të rëndësishëm sa vetë shkathtësitë e anëtarëve të ekipeve për t’i zgjidhur problemet. Në anën tjetër njoftimi me teknologjinë Rails dhe konceptet mbi të cilat ndërtohen ka mjaft voli dhe i jep secilit njohuri praktike të shëndoshë që të punojë me çfarëdo kornize tjetër zhvillimore dhe të ambientohet me çfarëdo stili arkitektural apo mostre programimi që nevojitet. Përfundimisht puna praktike me Ruby, MVC dhe stilin arkitektural REST e pasuron dukshëm mozaikun teorik me raste praktike të funksionimit apo mosfunksionimit dhe me qasje të reja për zgjidhje të problemeve, të cilat ndodhë në shikim të parë të duken triviale, por që domsdo kërkojnë vëmendje dhe zgjidhje specifike për të krijuar ueb aplikacione efiçiente, të shfrytëzueshme dhe të shkallëzueshme.

# Akronimet dhe Terminologjia

CRUD – Create, Read, Update and Destroy

CRUD operacionet – Operacionet e krijimit, fshirjes, përditësimit dhe shkatërrimit të definuara

sipas stilit arkitektural REST

REST – Representational State Transfer

RESTful App – Aplikacion i bazuar në stilin arkitektural REST

RoR – Ruby on Rails

VCS – Version Control System

# Shtojcat

### Dallimet dhe konfliktet mes versioneve në Rails

Në Fig. 5 mund të vërehet se në disa raste përveç emrit të gem paketave specifikohen me simbole të veçanta edhe versionet apo subversionet e lejuara. Një çështje e vazhdueshme e diskutimit për të gjithë softuerët me burim të hapur (open source) janë edhe varësitë mes paketave dhe kompatibiliteti mes versioneve, madje ka edhe sisteme të tëra operative të krijuara në përpjekje për ta zgjidhur problemin e përshtatjes mes versioneve aktuale dhe atyre më të reja të programeve të ndërvarura (CentOS).

E njëjta çështje e preokupon edhe kornizën Ruby on Rails. Përderisa ne punojmë me versionin 5.x të Rails dhe me versionet e paketave të paraqitura në Fig. 5, përditësimet e këtyre versioneve me gjasë të madhe do të shkaktonin konflikte mes kodit të ndërvarur. Meqë shpeshherë zhvilluesit krijojnë programe dhe shërbime mbi kodin ekzistues, ndryshimi i kodit themelor sigurisht që kërkon ndryshimin dhe përshtatjen e të gjitha programeve të derivuara prej tij. Ky efekt ortek bën shpesh që një aplikacion i tërë të ketë nevojë të rishkruhet si pasojë e këtyre konflikteve. Qasja kryesore që shfrytëzohet për ta përballuar këtë problem është specifikimi eksplicit i versioneve me të cilat punohet, si në figurën më lart dhe përditësimi gradual i shërbimeve duke e përmirësuar dhe përshtatur kodin vazhdimisht.

# Referencat

(Armel, 2014) Armel, J. (2014). *Web application development with Laravel PHP Framework version 4* (Unpublished bachelor's thesis). Helsinki Metropolia University of Applied Sciences. Marrë nga <https://www.theseus.fi/bitstream/handle/10024/74052/Author.pdf>

(Beck et al., 2001) Beck, K., Beedle, M., Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R., Mellor, S., Schwaber, K., Sutherland, J. and Thomas, D. (2001). *Manifesto for Agile Software Development*. Marrë nga: <http://www.agilemanifesto.org/>

(Bell & VersaPay, 2011) Bell, G., & VersaPay. (2011). Active Admin. Marrë nga <https://activeadmin.info/documentation.html>

(Bigg, et al., 2018) Bigg, R., Ben, O. D., Cheung, F., Darell, T., Dean, J., Gunderloy, M., . . . Dev, V. (2018). Ruby on Rails Guides (v5.2.0). Marrë nga <http://guides.rubyonrails.org/>

(Plataformatec, 2016) Plataformatec. (2016). Devise Documentation. Marrë nga <http://devise.plataformatec.com.br/>

(Sinclair, et al., 2018) Sinclair, G., Webber, W., Johnson, L., Gray, J. E., II, Britt, J., & Thomas, D. (2018). Ruby 2.5.1 Standard Library Documentation. Marrë nga <https://ruby-doc.org/>

(Tk, 2017) Tk. (2017). Ruby on Rails: HTTP, MVC and Routes – The Renaissance Developer – Medium. Marrë nga <https://medium.com/the-renaissance-developer/ruby-on-rails-http-mvc-and-routes-f02215a46a84>

# Bibliografia

(Coulouris, et al., 2011) Coulouris, G. F., Dollimore, J., Kindberg, T., & Blair, G. (2011). *Distributed systems: Concepts and design*. Harlow: Pearson Education Ltd.

(Fernandez, 2018) Fernandez, O. (2018). *The Rails 5 way*.

(Hartl, 2016) Hartl, M. (2016). *Ruby on Rails Tutorial: Learn Web Development with Rails*.

(Ruby, et al., 2017) Ruby, S., Copeland, D., Thomas, D., Hansson, D. H., & In Pfalzer, S. D. (2017). *Agile Web Development with Rails 5.1*.

1. Ruby është gjuhë programuese dinamike, e interpretuar, reflektive dhe orientuar në objekte. (Sinclair, et al., 2018) [↑](#footnote-ref-1)
2. Në origjinal *The Rails Way* (Fernandez, 2018). [↑](#footnote-ref-2)
3. Parimet kryesore mbi të cilat ndërtohet Rails në origjinal janë: *Don’t Repeat Yourself – DRY* dhe *Convention Over Configuration*. [↑](#footnote-ref-3)
4. Referojuni lidhjes vijuese për informata shtesë rreth Rack Ruby Webserver: <https://rack.github.io>. [↑](#footnote-ref-4)
5. Këto veprime në RESTful terminologjinë definohen si *CRUD* (*Create, Read, Update and Destroy*) operacione. [↑](#footnote-ref-5)
6. Rishfrytëzimi i kodit dhe librarive në Rails mundësohet nëpërmjet paketave *gem*, emrat e të cilave vendosen në *Gemfile* dhe janë të disponueshme për t’u shfrytëzuar pas instalimit me *RubyGems* në Rails aplikacion. *RubyGems* është menaxhuesi i paketave për gjuhën programuese Ruby, i cili ofron një format standard për distribuimin e programeve dhe librarive në Ruby dhe shërben si vegël e dizajnuar për menaxhimin e lehtë të instalimit të gems apo si server për distribuimin e tyre. [↑](#footnote-ref-6)
7. Manifestoja e principeve agjile është dokument i cili e shpërfaq një metodologji të re të zhvillimit softuerik, më qëllimin për ta përmirësuar cilësinë, produktivitetin dhe efiçiencën. Mbi këto principe bazohen të gjitha aplikimet e përmendura gjatë punimit. Manifeston agjile mund ta gjeni në (Beck et al., 2001). [↑](#footnote-ref-7)