Projektiranje sustava temeljenog na znanju primjenom CommonKADS metodologije

Čvrk, Matija

SVEUČILIŠTE U SPLITU

PRIRODOSLOVNO – MATEMATIČKI FAKULTET

DIPLOMSKI RAD

PROJEKTIRANJE SUSTAVA TEMELJENOG NA ZNANJU PRIMJENOM COMMONKADS METODOLOGIJE

Matija Čvrk

Informatika i tehnika

Mentor: **izv. prof. dr. sc. Saša Mladenović**

Voditelj: **dr. sc. Goran Zaharija**

Split, rujan 2017.

Temeljna dokumentacijska kartica

Diplomski rad

Sveučilište u Splitu
Prirodoslovno-matematički fakultet
Odjel za informatiku
Ruđera Boškovića 33, 21000 Split, Hrvatska

PROJEKTIRANJE SUSTAVA TEMELJENOG NA ZNANJU PRIMJENOM COMMONKADS METODOLOGIJE

Matija Čvrk

SAŽETAK

Metodologija CommonKADS je kolekcija strukturiranih metoda za izradu sustava temeljenih na znanju. Fokus metodologije CommonKADS kod razvoja sustava temeljenih na znanju je na modeliranje aktivnosti, stoga je osnova ovih metoda konstruiranje velikog broja modela koji predstavljaju različite poglede na ponašanje usmjereno rješavanju problema. Projekt u sklopu diplomskog rada ima za cilj osmisliti i implementirati koncept sustava za vođenje evidencije o posudbi, odnosno preuzimanju ključeva od prostorija obrazovnih institucija, primjenjivo konkretno na Prirodoslovno-matematičkom fakultetu u Splitu.

Ključne riječi: informatika, menadžment, programiranje, inženjering znanja, sustav temeljen na znanju, agenti, CommonKADS, RFID tehnologija, Internet stvari, responzivni dizajn .

Rad je pohranjen u knjižnici Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu

Rad sadrži: 58 stranica, 24 grafičkih prikaza, 1 tablica i 10 literaturnih navoda.

Izvornik je na hrvatskom jeziku.

Mentor: **Dr.sc. Saša Mladenović,** izvanredni profesor Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu

Neposredni voditelj: Dr.sc. Goran Zaharija, poslijedoktorand Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu

Ocjenjivači: **Dr.sc. Saša Mladenović,** izvanredni profesor Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu

Dr.sc. Goran Zaharija, poslijedoktorand Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu

Divna Krpan, predavač Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu

Rad prihvaćen: Rujan, 2017.

Basic documentation card

Graduate thesis

University of Split
Faculty of Science
Department of computer science
Ruđera Boškovića 33, 21000 Split, Hrvatska

KNOWLEDGE BASED SYSTEM DESIGN

BASED ON COMMONKADS METHODOLOGY

Matija Čvrk

ABSTRACT

The CommonKADS methodology is a collection of generic methods for designing knowledge based systems. The focus of CommonKADS methodology during the design process of knowledge based systems is on activities, therefore the basics of these methods is the construction of a large number of models which provide different views on the behaviour directed at problem solving. The aim oft he thesis project is to design and implement a concept of a system for managing records of borrowing or returning keys (sessions) form the premises of educational institutions, specifically applicable at the Faculty of Science in Split.

Key Words: informatics, management, programming, knowledge engineering, knowledge based system, agents, CommonKADS, RFID technology, Internet of things, responsive design.

Thesis deposited in library of Faculty of science, University of Split.

Thesis consist of: 58 pages, 24 figures, 1 tables and 10 references

Original language: Croatian

Supervisor: **Saša Mladenović, Ph.D.** Associate Professor of Faculty of Science, University of Split
Co-Supervisor: **Goran Zaharija, Ph.D.**, Postdoctorant Researcher of Faculty of Science, University of Split
Reviewers: **Saša Mladenović, Ph.D.** Associate Professor of Faculty of Science, University of Split
Goran Zaharija, Ph.D. Postdoctorant Researcher of Faculty of Science, University of Split
Divna Krpan, Lecturer Professor of Faculty of Science, University of Split

Thesis accepted: September, 2017.

Sadržaj

1. Uvod

Temeljnu motivaciju za izvedbu ovog diplomskog rada predstavlja projekt, jer pruža okruženje za istraživanje i rad na tematskom području Informacijsko-komunikacijske znanosti od interesa, preciznije raznim metodama prikupljanja i razmjene informacija, sustava za upravljanje informacijama i znanjem, konačno, metodama upravljanja ljudima i resursima na tom području.

Također, projekt diplomskog rada se razvijao u duljem vremenskom periodu te je dio toga rada obuhvaćen i u sklopu kolegija Informatički menadžment. Tako je jednim dijelom koncipiran i strukturiran u obliku projektne nastave navedenog kolegija, dajući mu i određeni pedagoški okvir za primjenu u nekoj budućoj projektnoj nastavi ili problematici sličnih kolegija.

Nadalje, okosnicu projekta čini metodologija za vođenje i organizaciju ljudi i resursa iz područja menadžmenta koji pokriva projekte visoke sistematizacije podataka i koja se prvenstveno provodi za projektiranje sustava temeljenih na znanju. Ta metodologija se naziva CommonKADS, a u nastavku slijedi više o tome i o sustavima temeljenima na znanju, kao i o samom projektu.

2. Osnovna ideja i domena projekta

Projekt u sklopu ovog diplomskog rada ima za cilj osmisliti i implementirati koncept sustava za vođenje evidencije o posudbi, odnosno preuzimanju ključeva od prostorija obrazovnih institucija, primjenjivo konkretno na Prirodoslovno-matematičkom fakultetu u Splitu.

Projektni zadatak u sklopu diplomskog rada je osmišljen da bi se zaista implementirao u praksi, odnosno konstruirano projektno rješenje je ono koje bi trebalo biti autonomno i potpuno integrabilno u području primjene. Zbog toga, je projektno rješenje, odnosno sustav trebao proći kroz određene faze razvoja tijekom kojih je bilo potrebno osigurati da se provedu na adekvatan način sa željenim rezultatima. Upravo zato je primijenjena metodologija CommonKADS za organizaciju i kvalitetno vođenje projekta kroz potrebne faze razvoja.

Kako se projektno rješenje, odnosno sustav primjenjuje u dinamičkom okruženju, prostori Prirodoslovno-matematičkog fakulteta u Splitu, tako je malo vjerojatno da bi se na početku izvođenja projekta moglo popisati sve moguće slučajeve primjene, potrebne zahtjeve i značajke sustava i predvidjeti sve korake integracije i formiranja konačnog rješenja. Stoga, se organizaciji i vođenju trebalo pristupiti na način da se zadaci i problemi mogu rješavati u hodu, ali tako da ih se cijepa u manje cjeline koje je lakše analizirati i adekvatno riješiti. Takvi preduvjeti rodili su potrebu za primjenu iterativnog modela razvoja projektnog rješenja, odnosno sustava, a CommonKADS spada u tu skupinu modela organizacije i vođenja projekata.

Znanje je cijeli skup podataka i informacija koje ljudi nastoje upregnuti u praktičnu upotrebu, kako bi izveli različite zadatke i formirali nove informacije. Znanje pruža dva različita aspekta: prvi, osjećaj svrhe, s obzirom na to da je znanje „intelektualni mehanizam“ koji se koristi za postizanje ciljeva; kao drugo, generativna sposobnost, jer jedna od glavnih funkcija znanja predstavlja formiranje novih informacija. Nije slučajno, stoga, da se znanje smatra novim „faktorom proizvodnje“. (*G. Schreiben, 2000.*)

Informacije koje dizajnirani sustav prikuplja su određene domenom projekta, a to je rješavanje evidencije i upravljanja ključevima od prostorija obrazovnih institucija, što zahtjeva sistematičan pristup strukturiranju i prikazu podataka, koji ujedno moraju biti uvijek dostupni i lako pristupačni u realnome vremenu. Sve navedeno razlog je za konstruiranje sustava temeljenog na znanju, koje spada u domenu inženjeringa znanja, no više o tome u sljedećem poglavlju.

Također, kako bi informacije koje sustav omogućuje korisniku bile lako dostupne u realnome vremenu, javlja se potreba za jako mobilnim rješenjem i to rješenjem neovisnim o platformi s koje korisnik pristupa sustavu. Univerzalni i danas neizostavni način razmjene informacija i usluga je putem mreže svih mreža, Internet, koji predstavlja hipermedijski prostor koji je prisutan na danas svim platformama. Stoga, je i sam projektni sustav, ujedno i hipermedijski sustav, o čemu će također biti više riječi u nastavku.

3. Problematika i sustavi upravljanja znanjem

Znanje i upravljanje znanjem je važno jer ono često nadživi konkretnu implementaciju. (*D. Fensel et alt, 2010.*) Sustav temeljen na znanju je sustav koji rješava problem iz realne domene ljudskog života koristeći se znanjem o domeni aplikacije i zadatku unutar same aplikacije. Postoje još i ekspertni sustavi, no oni rješavaju problem koji bi u čovjeka zahtijevao viši nivo znanja i poznavanja da bi se riješio. Kako bi se u nastavku bolje opisala cijela problematika i metode rješavanja, potrebno je prvo objasniti neke ključne pojmove vezane za sustave upravljanja znanjem.

3.1. Ključni pojmovi

Domena je općenito, područje od interesa, unutar koje se nalazi problem kojeg sustav treba riješiti. Domena može biti prehrambena industrija, drvna industrija, proizvodnja automobila, bilo koje područje ljudskog djelovanja gdje je potrebno povezati znanje sa aktivnostima koje donose čovjeku vrijednost.

Nadalje, zadatak je općenito, nešto što treba obaviti agent, primjerice, nadgledanje i praćenje procesa, formiranje plana, analiza nekarakterističnog ponašanja sustava i slično.

Prema tome, agent je izvršitelj zadatka unutar domene djelovanja, tipično je to čovjek ili nekakav sustav (programske podrške). Aplikacija je kontekst zadatka unutar domene kojeg treba izvršiti nekakav agent ili agenti.

Domena aplikacije je posebno područje od interesa uključeno unutar aplikacije, dakle podskup opće domene djelovanja. Zadatak unutar aplikacije je zadatak visoke razine kojeg treba izvesti određena aplikacija.

Dakle, na primjeru diplomskog rada, vidljivo je da je domena djelovanja organizacija preuzimanja ključeva od prostorija fakulteta, no domena aplikacije unutar te domene je projektiranje takvog rješenja koje putem interneta pruža korisnicima fakulteta da dobiju uvid u realnome vremenu o posjedovanju određenih ključeva u nekom trenutku.

Zadataka unutar domene djelovanja ima raznih, no oni zadaci koji spadaju u odabrani podskup za rješavanje ovim sustavom nazivaju se zadaci unutar domene aplikacije. Ti zadaci ubrajaju, dostupnost informacija putem interneta, pohranu i evidentiranje podataka o ključevima, korisnicima i vremenu evidentiranja, zatim prikaz željenih podataka na zahtjev, te način prikazivanja unutar sučelja putem koje korisnik pristupa sustavu.

Također, postoje i zadaci koji nisu direktno u doticaju sa samim korisnikom, već su posljedica potrebe radi izrade ili testiranja sustava, te osiguravanja autonomnog i nesmetanog rada.

Konačno, sami agenti su pored ljudskih resursa, hijerarhijski poredanih korisnika (asistenata, profesora i administratora) i sami programski agenti, odnosno klijentski preglednici koji služe kao agenti za komunikaciju sa sustavom za pohranu i razmjenu podataka (server) koji na zahtjev stvara novu instancu agenta za posluživanje zahtjev od pojedinog klijenta. Generiranje se odvija po redu, dakle prema FIFO principu (eng. *First In, First Out*).

3.2. CommonKADS metodologija

CommonKADS metodologija (metodologija prikupljanja znanja i formiranja dokumentacije eng. *Knowledge Acquisition and Documentation Structuring*) je strukturirani način razvoja sustava temeljnih na znanju ili KBS (eng. *Knowledge Based System*), odnosno ekspertnih sustava (eng. *Expert System*). (J.B. Waldner, 1992.) Razvijena je na Sveučilištu u Amsterdamu kao alternative evolucijskom pristupu i sada je općeprihvaćena kao Europski standard za sustave temeljene na znanju (u kontekstu europskog “Esprit IT” programa).

CommonKADS je najkorištenija metodologija na području inženjeringa znanja, uz metodologije MOKA i 47-Step Procedure. Inženjering znanja je, prema definiciji E. Feigenbaum, P. McCorduck, 1983. inženjerska disciplina koja podrazumijeva integraciju znanja u računalni sustav s ciljem rješavanja

kompleksnih problemskih zadataka koji inače iziskuju visoku razinu ljudske stručnosti. Osnovna problematika metodologija inženjeringa znanja je uvođenje ontologija.

Metodologija CommonKADS je kolekcija strukturiranih metoda za izradu sustava temeljenih na znanju. Fokus metodologije CommonKADS kod razvoja sustava temeljenih na znanju je na modeliranje aktivnosti, stoga je osnova ovih metoda konstruiranje velikog broja modela koji predstavljaju različite poglede na ponašanje usmjereno rješavanju problema. Metode zbirno imenovane kao CommonKADS metode, pokazale su se vrlo korisne I ponovo upotrebljive diljem velikog broja različitih zadataka u razvoju sustava temeljenih na znanju. (J. Kingston, 1996.)

Ključan faktor u uspjehu metodologije CommonKADS je njena biblioteka generičkih modela zaključivanja koji se mogu primjeniti na zadatke određenog tipa. Ovi modeli sugeriraju korake formiranja zaključka prilikom izvedbe zadatka određenog tipa, al ii uloge koje izvodi domena znanja u cjelokupnom procesu rješavanja problema.

Primjerice, generički model za zadatak sistematske dijagnoze uključuje korake formiranja zaključka kao što su raščlamba skupa mogućih pogrešaka i testiranje promatranih vrijednosti naspram očekivanih vrijednosti.

Također, navedeni model pokazuje skup mogućih pogreški što služi u dvije svrhe kod postupka davanja dijagnoze, na prvom mjestu kao dio modela koji opisuje ponašanje sustava sa pogreškom, zatim kao hipotetski uzroci simptoma (jednog ili više) koje se promatra. Navedeni generički modeli se mogu koristiti bilo prema strategiji razvoja temeljenog na odnosu opće ka pojedinačno (eng. *top-down*), gdje se modeli ponašaju kao platforma (eng. *framework*) za prikupljanje znanja, bilo kao način provjere dovršenosti modela, gdje se razvija strategijom temeljenom na odnosu pojedinačno ka općem (eng. *bottom-up*), dakle analizom domene. (J. Kingston, 1996.)

4. Dizajn, razvoj i implementacija

Za dizajn i razvoj sustava temeljenih na znanju postoji više razvijenih i dokazanih metodologija, a jedna od njih je CommonKADS. CommonKADS je, zbog svoje prirode i sadržaja, logičan izbor pri osmišljavanju i izradi ITS-a. Metodologija prati spiralni model razvoja što osigurava neprekidan proces poboljšavanja i nadzora nad samim sustavom. Spiralni model razvoja se može prikazati shemom na Slici 1. Kako postoji

dobra dokumentacija za CommonKADS metodologiju, moguće je definirati parametre važne za unaprjeđenje i iskoristiti ih u daljem radu.

Slika 1. – Spiralni model razvoja programske podrške

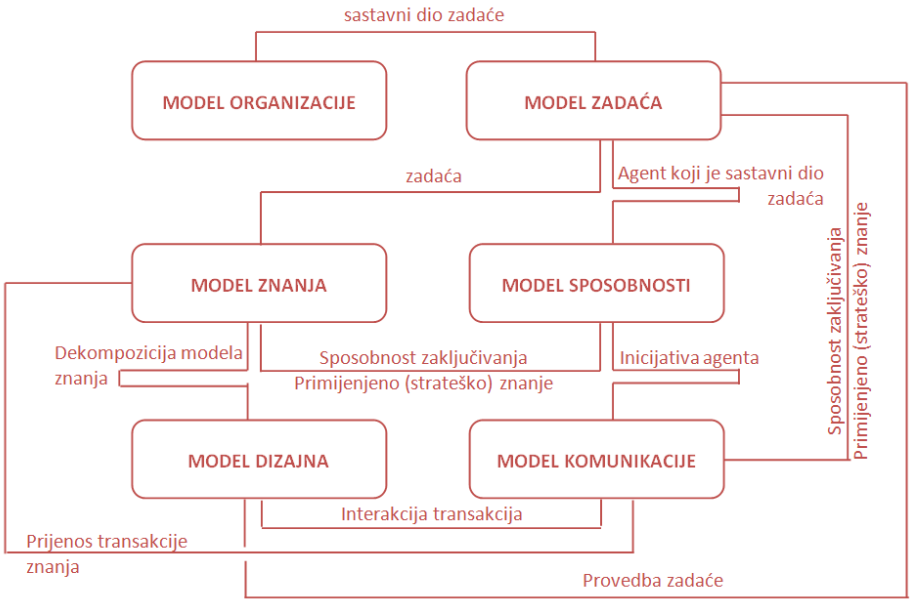
Spiralni model je kombinacija iterativnog i linearnog modela (aktivnosti: oblikovanje, implementacija i dostavljanj) razvoja, gdje svaki puni krug spirale predstavlja izrađeni prototip, a aktivnosti modela uključuju: planiranje, analizu rizika, oblikovanje, implementaciju, dostavljanje i evaluacija kupca.

Pri realizaciji projekta, CommonKADS metodologija prepoznaje i preporučuje izradu više modela:

- Model strukture i funkcije organizacije. Ključni elementi ovog modela su poslovni procesi, strukturni elementi, poslovni resursi i različite vrste veze među njima.
- Model zadataka potreban je za izvođenje pojedine operacije. Ključni elementi u ovom modelu su zadatci potrebni za opisivanje jednog poslovnog procesa.
- Model komunikacije među agentima za vrijeme obavljanja zadataka. Ključni elementi ovog modela su transakcije.
- Model znanja koje je potrebno za obavljanje zadatka.
- Model dizajna sustava na temelju znanja (za upravljanje znanjem) kako bi bio u mogućnosti obaviti svaku od zadataka u cijelosti ili dijelom, prema zahtjevima. Ključni korak u dizajnu

sustava CommonKADS metodologijom je (uobičajeno), funkcionalna dekompozicija zadataka temeljenih na znanju na komponente koje se nazivaju funkcionalne jedinice.

CommonKADS metodologija prati razvoj sustava temeljenog na znanju kroz izradu brojnih odvojenih modela (vidjeti Sliku 2) koji opisuju glavne značajke sustava i njegovog okruženja. Proces razvoja sustava temeljenih na znanju, sastoji se od popunjavanja niza predložaka modela. U predlošku modela definirana su stanja koja označavaju točku razvoja modela. Oznake omogućuju vođenje projekta koji tada prati, klasično definirani, spiralni model razvoja.



Slika 2. – Skup CommonKADS modela

Ovako opisani skup modela još ne definira rezultate koji se očekuju nakon svake od faza KADS modela. Za razvoj sustava temeljenih na znanju prema KADS metodologiji definirani su svi koraci, faze, aktivnosti i rezultati.

Općenito, modeli se definiraju na različitim razinama unutar nekog projekta pa tako postoje: modeli predloška, modeli instanci, modeli verzija i višestruki modeli instanci.

Modeli predloška su predefinirane, fiksne strukture, koje se lako mogu konfigurirati prema potrebi. Modeli verzija su verzije različitih instanci modela, gdje se instance razlikuju prema broju, tj. identifikatoru verzije instance, a taj se identifikator obično sprema u bazu podataka kao zaseban stupac tablice modela. Nadalje, modeli se mogu razvijati paralelno pa to zovemo višestrukim modelima

instanci. (Napomena, ovdje se instanca odnosi na pojedini kreirani model, ne u smislu objektne paradigme programiranja kao instanca klase.)

5. Prilagodba korisniku

Prilagodba sustava za upravljanje ključeva korisniku poseban je izazov jer je potrebno sustav omogućiti korisnicima u bilo kojem trenutku, tako da pruže trenutne podatke, dakle u realnom vremenu. Platforma koja konceptualno zamišljena da riješi takav problem je Internet, kao mreža svih mreža koja povezuje međusobno ljude, robu i usluge putem takozvanih hipermedijskih sustava. Hipermedijski sustavi odavno su u upotrebi za upravljanje znanjem i resursima, općenito ne ograničavaju korisnika, uzmimo za primjer hipermedijski sustav Wikipedije.

Hipermedijski sustav je u osnovi sustav koji se zasniva na pružanju medijskih sadržaja korisniku u formi internetskih stranica, što znači da mu se pristupa putem internetskog preglednika. Zbog široke primjene interneta u raznim uređajima, od pametnih telefona do stolnih računala, ali i raznoraznih pametnih naprava, hipermedijski sustavi predstavljaju glavni način pružanja informacija, sadržaja i usluga za navedene uređaje korisnicima diljem svijeta. Zato su na gotovo svim uređajima povezanima na Internet, prisutni internetski preglednici raznih proizvođača (među vodećima su preglednici Firefox od organizacije Mozilla i Chrome od globalne tvrtke Google).

Upravo zato je projekt u prilogu ovog diplomskog rada od početka osmišljen i izrađen kako bi se pružao na platformi interneta, odnosno kao hipermedijski sustav.

Nadalje, kako bi se projekt mogao izvesti u praksi, programsko rješenje i popratna tehnička podrška moraju biti ekonomični i efikasni, dakle to predstavlja jedan od preduvjeta za izvedbu projekta. No pružanje informacija korisniku nije jedini aspekt koji je potrebno ekonomizirati, već i samo prikupljanje informacija.

Kao što je već navedeno u opisu osnovne ideje projekta, informacije koje dizajnirani sustav prikuplja su određene domenom projekta, a to je rješavanje evidencije i upravljanja ključevima od prostorija obrazovnih institucija, dakle prikupljaju se informacije o sesiji preuzimanja ključeva. Cilj je popratiti i ekonomizirati korištenje resursa (prostorija i pripadnih ključeva), kako bi se olakšalo djelatnicima obrazovnih institucija da sa određenom sigurnosti mogu provjeriti stanje resursa u realnome vremenu. Informacije popraćene unutar sesije određene sesije uključuju vrijeme početka i završetka sesije, oznaku

korisnika, oznaku prostorije i oznaku ključa (jer postoje više ključeva od istih prostorija, odnosno duplikati ključeva), no više o tome u dijelu gdje se govori o modelu znanja diskutiranog sustava.

Upravo zbog različitih informacije koje je potrebno pratiti, konstruirati i koje sustav treba moći rekonstruirati i pružiti nekom korisniku u realnome vremenu, jasno je da se projektom mora upravljati iz više različitih smjerova, od osmišljavanja ideje sustava, planiranja izvođenja i implementacije projekta, do razvijanja koncepta sustava i njegove integracije u realni svijet, ali i raznih smjerova sa tehničke strane, kao što su nesmetani nastavak rada sustava sa smetnjama iii bez napajanja, odabir i povezivanje tehničke podrške sustava, kao i modeliranje znanja sustava (skupa informacija kojima sustav upravlja) te konačno i modeliranje hijerarhije provođenja aktivnosti i različitih smjerova komunikacije unutar sustava. Zato je metodologija CommonKADS našla svoju primjenu na ovakvom projektu, jer pruža izvrsnu podršku za vođenje jednog ovakvog projekta iz raznih smjerova pomoću generičkih modela koje uvodi.

Organizacijski model podržava analizu organizacije, cilj je detektirati probleme, mogućnosti i moguće posljedice sustava prilikom razvoja. Model zadataka opisuje zadatke koji se izvode ili će se izvesti u okruženju organizacije projekta, ali i sustava u užem smislu. Model agenata opisuje mogućnosti, norme, preference i dozvole agenata, gdje je agent izvršitelj zadatka. Model znanja daje opis skupa znanja neovisan o implementaciji rješenja, koji su uključeni u zadatke. Komunikacijski model opisuje transakcije poruka između agenata. Model dizajna opisuje strukturu sustava koji je potrebno konstruirati.

5.1. Model organizacije

Struktura zadataka projektiranja sustava ili WBS (eng. *Work Breakdown Structure*) je tehnika podjele zadataka u projektnom menadžmentu i sistemskom inženjeringu na njihove manje komponente. Prema tome, ukupan posao treba razdijeliti u cjeline veličina prikladnih za provođenje kontrole projekta, a najniža razina podjele mora biti dovoljno konkretna da omogući kontrolu i razvidnost.

S obzirom da je postavljanje organizacijskog modela jedna od glavnih komponenti CommonKADS metodologije, WBS se pokazao kao idealna tehnika za strukturiranje i organizaciju zadataka. Hijerarhijska raščlamba zadataka (WBS) projekta koji je proveden u svrhu rada se nalazi na Shemi 1. Na navedenoj shemi, u zaglavlju nalazi se legenda sa prikazom faza razvoja projekta označenih bojom. Tim su bojama označeni određeni poslovni zadaci prema fazi u kojoj se izvode, unutar hijerarhije zadataka cijelog

projekta. Tako se na primjer, „Studija tehničke izvedbe“ provodi u fazi pokretanja projekta i odlučuje početni pravac razvoja projekta, ili na primjer, „Izrada aplikativnog softvera“ je aktivnost koja se provodi već u fazi izrade, ili na primjer, „Testiranje tehničkih komponenti“ je aktivnost koja se provodi nakon implementacije određene iteracije programskog rješenja sustava, dakle u fazi kontrole sustava.

Prema prikazanome stablu WBS, zadaci su grupirani u 7 kategorija (prema glavnim aktivnostima unutar projekta): planiranje, prototipiranje, izrada, testiranje, dokumentiranje, educiranje, zaključivanje.

Tako kategorija planiranja obuhvaća zadatke koje je potrebno obaviti da bi se realizirao projekt na konceptualnoj, ekonomskoj i organizacijskoj razini. Odabrano je ostvarivanje projekta kroz 2 koncepta sustava – primarni (digitalni, elektronički) i sekundarni (kada primarni sustav otkaže, zablokira ili se ukine napajanje).

Kategorija izrade prototipa se bavi inicijalnim osmišljavanjem sustava i iterativnim poboljšavanjem minimalno isporučivog proizvoda ili MVP (eng. *Minimum Viable Product*) u pogledu tehničke i programske podrške, što uključuje i nabavu potrebnih tehničkih komponenti i instalaciju potrebnih programskih alata i paketa.

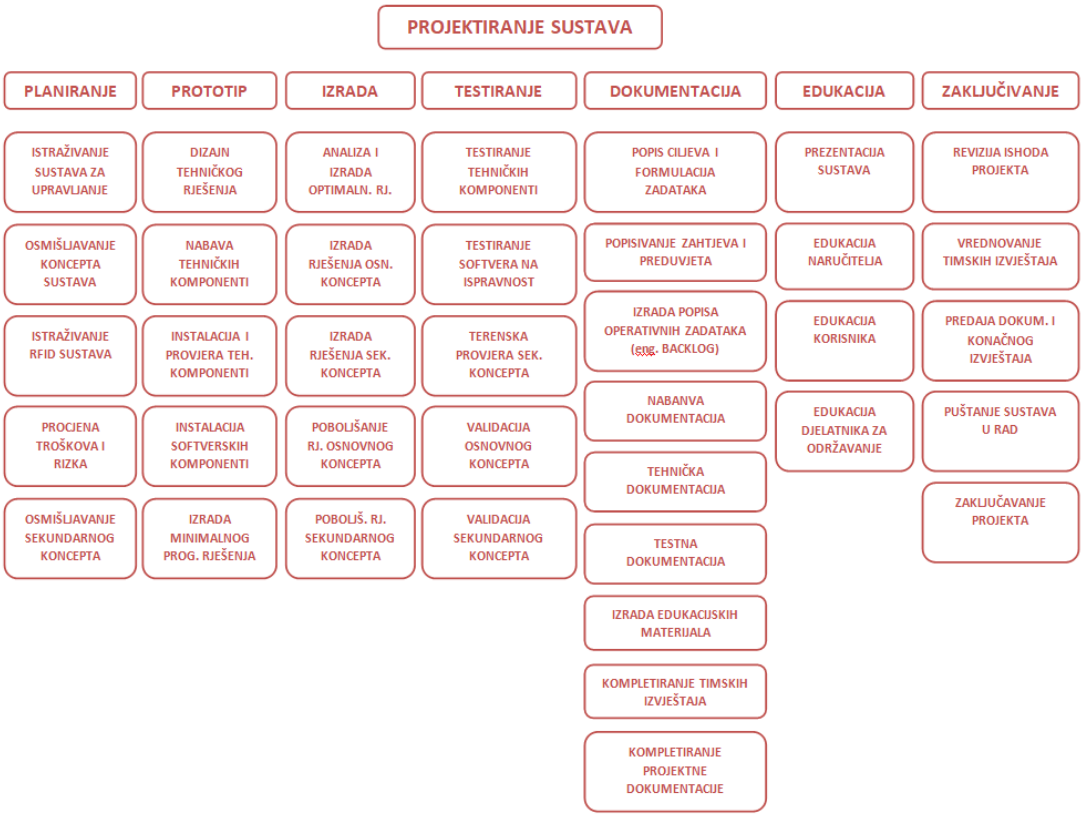
Kategorija testiranja obuhvaća zadatke provjere ispravnosti tehničke i programske strane sustava i njegovih komponenti prije, tijekom i nakon integracije. Provjere se vrše i nad prototipom i nad kompletiranim proizvodom.

Zadaci izrade provode se nakon određenih koraka ostvarenih u fazi pokretanja, planiranja i izvođenja prototipa sustava, dakle kada već postoji određeni testirani MVP.

Zadaci dokumentacije obuhvaćaju različite zadatke koji se protežu tijekom trajanja cijelog projekta, jer je potrebno dokumentirati sve zahtjeve projekta i sustava, inicijalne procjene i analize problema, procijenjeni rizici, pridijeljeni i nepridijeljeni zadaci, testni prilozi, programski kod, priručnici, prekretnice, prezentacijski i edukativni materijali itd.

Zadaci iz kategorije edukacije obuhvaćaju aktivnosti vezane za približavanje sustava krajnjem korisniku, dakle prezentacije, priručnici, video lekcije i slično.

Konačno, zadaci zaključivanja se odnose na kompletiranje timskih izvještaja, upuštanje u rad.



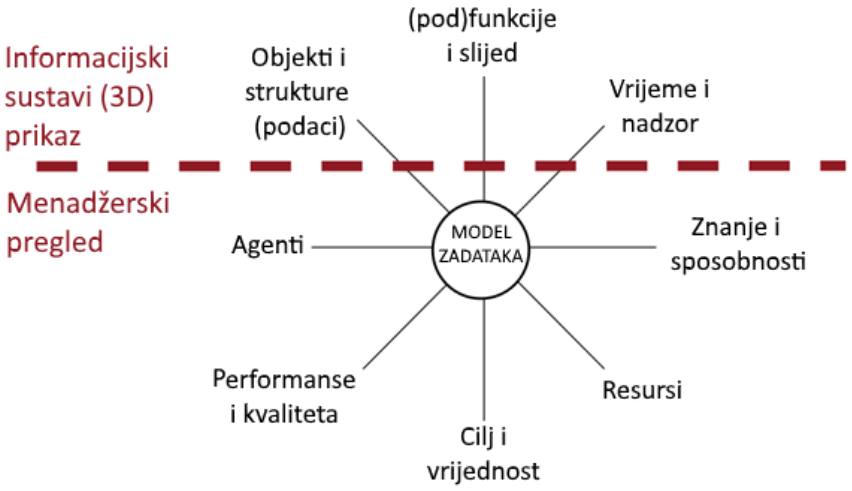
Shema 1 – hijerarhijska struktura zadataka ili WBS

5.2. Model zadataka

Zadatak je ono što treba uraditi agent. Zadatak unutar aplikacije je zadatak najviše razine koji se treba uraditi unutar dosega aplikacije. Model zadataka opisuje koje zadatke treba izvršiti i kako se oni trebaju izvršiti unutar organizacijskog okruženja.

Znanje zadataka treba biti orijentirano prema određenom cilju i treba biti funkcionalno dekomponirano. Primjerice, procjena hipoteke realizirana putem aplikacije kako bi se minimizirao rizik gubitka novca, ili pronalaženje defekta i neispravnosti fotokopirnog uređaja kako bi se obnovio i održala usluga, ili dizajniranje dizala za novu zgradu. Općenito, potrebno je opisati strategije koje se mogu primijeniti za postizanje ciljeva, što se tipično realizira putem hijerarhijske strukture.

Zadatak iz poslovne perspektive je dio cjelokupnog poslovnog procesa. Također, to je prema cilju orijentirana aktivnost, koja treba davati vrijednost poslovnom procesu. Zato zadatak troši resurse, znanje i sposobnosti, ali ih i pruža. Kao ilustraciju pogledati Sliku 4.



Slika 4. – Prikaz zadatka u kontekstu cjelokupnog poslovnog procesa

Prema tome, kao na Slici 4, model zadataka projekta diplomskog rada moguće je definirati definiranjem *zadataka za određivanje cilja i vrijednosti*, *resursa*, *znanja i sposobnosti*, *agenata*, *zadatke određivanja i provođenja performansi i kvalitete*, *zadacima praćenja vremena i obavljanja nadzora* nad razvojnim procesom, *objektima i strukturama* te *slijedom funkcija*.

Zadaci koji omogućuju formiranje cilja i određivanje vrijednosti su zadaci koje je potrebno obaviti u ranoj fazi planiranja, a to su zadaci vezani za *istraživanje sustava za upravljanje* ključeva i sustavima koji nude mogućnost evidencije i upravljanja sličnim resursima.

Resursi raspoloživi za projekt su osobe uključene u proces razvoja projektnog rješenja, od voditelja projekta do razvojnog tima. To su tzv. ljudski resursi. Također, raspoloživi su i tehnološki resursi, među kojima se ubrajaju računala za izradu projektnog rješenja, za vođenje dokumentacije, kompletna programska podrška za razvoj i vođenje dokumentacije, te također i tehnološki resursi potrebni za implementaciju sustava u materijalnom svijetu, dakle za implementaciju tehničkog modela projektnog rješenja (uređaj za praćenje resursa, napajanje sustava, elektroničke komponente za identifikaciju resursa). Zadaci koji omogućuju definiranje resursa za projekt opisan u ovom radu su *istraživanje RFID sustava*, *procjena troškova i rizika, izrada nabavne dokumentacije*, *nabava tehničkih komponenti*, *instalacija i provjera tehničkih komponenti*, *instalacija softverskih komponenti (programa)*.

Znanja koja iziskuje domena projekta opisanog u ovom radu uključuju znanje o procesu rukovanja resursima, iskustva korisnika sa prethodnim sustavima, znanja o načinu rada sličnih sustava i najvažnije neposredna znanja koja sustav prikuplja, to su informacije iz realnog svijeta koje se sistematski povezuju i prevode u reprezentaciju znanja i shvaćanja inteligentnog sustava.

Sposobnosti potrebne za izvođenje projekta i rukovanje znanjem ubrajaju sposobnosti integracije tehničkih i programskih komponenti sustava, sposobnosti vođenja ljudskih i tehničkih resursa, edukacija, sposobnost analize i testiranja koncepta i rješenja sustava, te same sposobnosti unutar aplikativne domene sustava, dakle sposobnost prikupljanja znanja i informacija, obrade postojećih i formiranja novih reprezentacija znanja, te sposobnosti pružanja usluge korisnicima u realnome vremenu. Više o sposobnostima i znanju u nastavku, no sami zadaci definiranja istih za opisani projekt su sljedeći: *popis ciljeva i formulacija zadataka, popis zahtjeva i preduvjeta, osmišljavanje koncepta sustava, osmišljavanje sekundarnog koncepta sustava, edukacija naručitelja, edukacija korisnika, validacija primarnog i sekundarnog koncepta rješenja* itd.

Zadaci koji definiraju uloge agenata u procesu razvoja su *izrada popisa operativnih zadataka* (jer ti zadaci zahtijevaju konkretne sposobnosti, prema kojima se dodjeljuju agentima), *kompletiranje i vrednovanje timskih izvještaja* (jer se ovisno o prikupljenom znanju i iskustvima agenata nakon obavljanja određenog zadatka ili dijela zadatka može promijeniti raspored i uloga agenata ili čak upariti više agenata po zadatku).

Kako bi se osigurala kvaliteta i tražene performanse tijekom provedbe projekta, no i kod samog projektnog rješenja, potrebno je nakon postizanja krupnijih zadataka provesti reviziju, provjeru ostvarenosti zadataka ili validaciju samog koncepta. Prema tome, zadaci ove kategorije u projektu opisanom u ovom radu su *analiza i izrada optimalnog rješenja, testiranje tehničkih i programskih komponenti, validacija primarnog i sekundarnog koncepta rješenja, izrada edukacijskih materijala, kompletiranje projektne dokumentacije, vrednovanje timskih izvještaja.*

Zadaci nadzora i praćenja vremena provedbe zadataka razlikuju se ovisno o fazi projekta, no zasnivaju se na praćenju rada agenata i ispunjavanju zadataka prema dogovorenim rokovima. Zatim se na temelju efikasnosti agenata i vremenu potrošenom za izvedbu zadataka vrše projekcije i daljnje planiranje zadataka projekta. Slijed provedbe zadataka također je uvjetovan ovakvim projekcijama.

Zadaci formiranja strukture rješenja, dakle objekata i funkcija, te njihova slijeda opisani su više u dizajnu samog rješenja.

5.3. Model znanja

Model znanja daje opis skupa znanja neovisan o implementaciji rješenja koji su uključeni u zadatke. Kategorije znanja mogu biti znanje vezano za domenu, znanje vezano za zaključivanje i znanje vezano za zadatke. (*D. Fensel et alt., 2010.*)

Znanje vezano za domenu ili znanje domene daje informacije relevantne za domenu, obično statične informacije. Znanje iz domene zahtjeva shemu domene, odnosno shematski opis znanja i tipova informacija, što se definira kroz konstrukcije domene. Također, znanje iz domene zahtjeva bazu znanja, odnosno skup instanci znanja. Konstrukcije domene su koncept (eng. *object class*), relacija (eng. *association*), atribut kao nekakva osnovna vrijednost (eng. *primitive value*) i tip pravila koja predstavljaju izraze (eng. *expressions*). Dijagram 1 prikazuje primjer konstrukcije domenskog znanja o spremniku goriva u autu.

Dijagram 1 – Primjer konstrukcije domenskog znanja o spremniku goriva u autu

Iz primjera sa Dijagrama 1, prema D. Fensel et alt., 2010. može se konstruirati domensko znanje na sljedeći način:

KONCEPT *kazaljka goriva*

ATRIBUTI *vrijednost: kazaljka goriva*

KONAČNI KONCEPT *kazaljka goriva*

KONCEPT *spremnik goriva*

ATRIBUTI *status: {pun, gotovo-pun, prazan}*

KONAČNI KONCEPT *spremnik goriva*

TIP VRIJEDNOSTI *vrijednost kazaljke*

LISTA VRIJENDOSTI: *{nula, niska, normalna}*

GRUPA: *ordinalni podatak*

KONAČNI TIP VRIJEDNOSTI *vrijednost kazaljke*

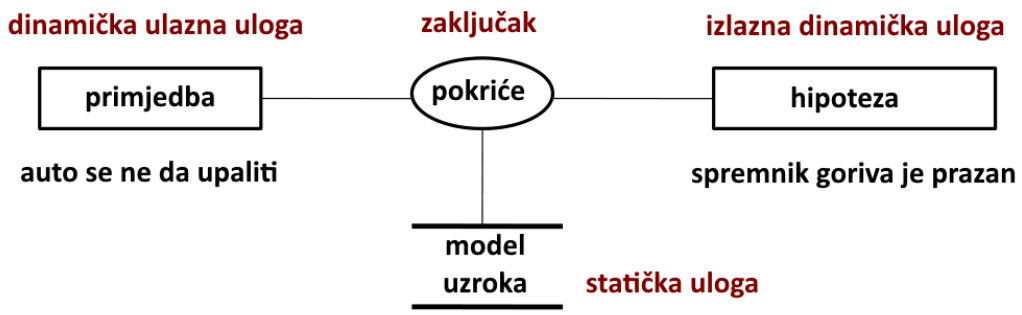
Modeliranje pravila izvodi se analizom znanja čiji fokus je pronalaženje pravila sa zajedničkom strukturom. Pravilo je instanca tipa pravila. Pravilo modelira relaciju između premisa o atributima. (D. Fensel et alt., 2010.)

Npr. „ ako kazaljka goriva pokazuje na nulu, onda je stanje spremnika goriva takvo da je prazan“.

Modeliranje pravila obuhvaća skup realnih pravila sa sličnom strukturom.

Znanje vezano za zaključivanje uključuje korake zaključivanja koji se zasnivaju na znanju iz domene i direktno se primjenjuju u zadacima. To znanje predstavlja najniži nivo funkcijske dekompozicije, a zasniva se na osnovnim jedinicama obrade informacija (zaključivanje vodi ka rasuđivanju, transfer znanja vodi komunikaciji s ostalim agentima).

Na primjeru sa Dijagrama 1, model zaključivanja uključuje korake prikazane na Dijagramu kompozicije 2.



Dijagram 2 – dijagram kompozicije postupka rasuđivanja o stanju spremnika na temelju kazaljke goriva

Znanje vezano za zadatke je orijentirano prema cilju, te je rezultat funkcionalne dekompozicije znanja iz domene. Ono opisuje ciljeve, te strategije koje se mogu primijeniti kako bi se ostvarili ciljevi, uobičajeno se prikazuju hijerahijski.

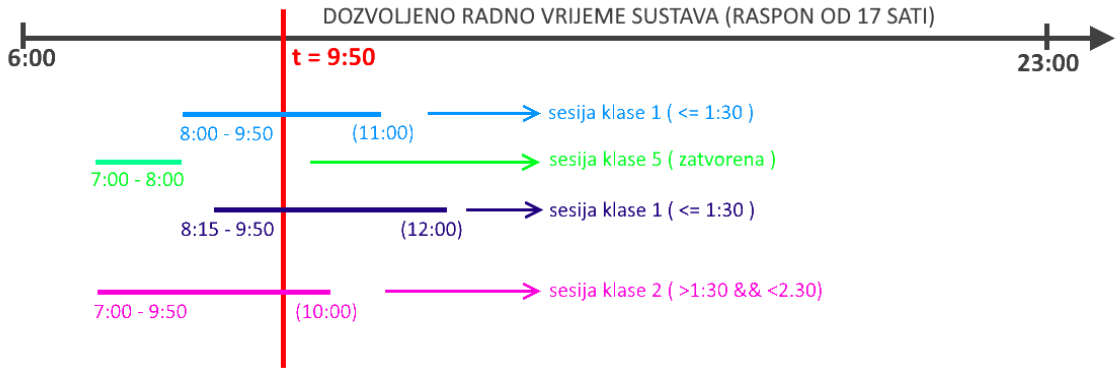
Model znanja projekta diplomskog rada zasniva se na praćenju sesija, odnosno preuzimanja ključeva korisnika sustava. Da bi se modelirao problem praćenja sustava potrebno je definirati prostor opcija sustava, odnosno ispisati stablo slučajeva.

Ako je u sustavu, u nekom trenutku, registrirano P – prirodan broj prostorija, tj. $1 \leq p \leq m$. Također, za svaku od tih prostorija može postojati Q – prirodan broj duplikata ključa koji otvaraju istu bravu pojedine prostorije, dakle $1 \leq q \leq n$.

Sada kad je definiran prostor opcija sustava, javljaju se 2 problema:

1. Ako osobne predigne 1 duplikat ključa za neku od prostorija, odradi svoju sesiju i ne vrati odmah ključ, kako onda odrediti je li prostorija slobodna?
2. Ako se dogodila situacija kao kod problema 1, ali je sada druga osoba posudila drugi duplikat ključa za istu prostoriju kao i prva osoba, a nijedna nije vratila ključ u nekom trenutku, kako odrediti koja osoba koristi prostoriju?

Ilustracija ova dva problema i prostor slučajeva nalazi se na Slici 5.



Slika 5 – primjer istovremenih sesije

Postavlja se nekoliko pitanja za dani problem, kao prvo, koja je sesija realizirana u trenutku *t*? Zatim, drugo pitanje je, postoji li uopće sesija koja bi mogla biti realizirana u tom trenutku?

Algoritam za rješavanje problema, odnosno za filtriranje sesija je sljedeći: prvo se uzmu sesije za određenu prostoriju broja K za trenutni dan. Zatim se klasificiraju odabrane sesije prema trajanju, kako bi se mogle filtrirati. Za dani problem identificirane su sljedeće klase sesija vidljive na Tablici 1.

Dakle, vidljivo je sa tablice da postoji 5 klasa sesija, a sesije koje će se prve zanemariti filtriranjem su one klase 4. I 5. Razlog tome je što je sustav zamišljen tako da sesije klase 4 svodi na sesije klase 3, zatvaranjem dijela sesija od prethodnog dana i obnavljanjem sesije za trenutni dan, bilo prilikom ponovnog paljenja sustava ako sustav radi u načinu štednje, tako da se gasi van dozvoljenog radnog vremena, bilo nakon ponoći ako sustav ne radi u načinu štednje, dakle radi i van dozvoljenog radnog vremena. Nakon filtriranja sesije, izračunajmo trajanja sesije do trenutka promatranja *t*.

KLASA SESIJE (PREMA VREMENU TRAJANJA)	VRIJEME TRAJANJA SESIJE
Klasa 1	≤1h 30min (Sesija traje malo više od 1 predavanja)
Klasa 2	≤2h 30min (Sesija traje malo više od 2 predavanja)
Klasa 3	≤24h (Sesija traje do kraja dana)
Klasa 4	>24h (Sesija traje više od jednog dana.)
Klasa 5	Zatvorena/Završena sesija

Tablica 1 – prikaz klasa sesija

Dakle, usporedimo li oba problema s početka, da se lako zaključiti da je drugi problem na jednostavniji za riješiti, pa je zato dobro prvo njemu se posvetiti. Dakle, kada se traži ključ od neke prostorije, tj. status neke prostorije, prikažu se sve sesije za neku prostoriju sortirano prema datumu i vremenu preuzimanja od najnovije, a sakriju one sesije koje su zatvorene (gdje su ključevi vraćeni, klasa sesije 5). Na taj način eliminiramo razmatrati osobe koje su vratile ključ, a gledamo od zadnje koja je uzela.

Odgovor na prvo pitanje, ako samo jedna osoba predigne ključ i ne vrati ga do određenog trenutka t, sustav nema dovoljno informacija, odnosno znanja da bi zaključio automatski stanje zauzeća prostorije, jer ne postoje druge sesije na temelju kojih bi se dalo formirati hipotezu pa je jedina preostala opcija korisniku da kontaktira u realnom svijetu tu osobu koja je započela i nije završila sesiju.

Naime, cilj sustava je omogućiti praćenje, evidenciju i upravljanje ključevima, sami problem upravljanja korisnicima, uvjetovanje određenog ponašanja ili pravila rukovanja ključevima je van dosega zahtjeva pa i samog koncepta sustava.

5.4. Model sposobnosti

Općenito, sposobnosti su definirane iz zahtjeva projekta, odnosno iz definiranih pripadnih zadataka. Na višoj razini projekta, model sposobnosti predstavlja specifikaciju i grupiranje sposobnosti koje se traže od agenata za obavljanje pojedinih zadaća, stoga zadaci će se pridjeljivati pojedinim agentima sukladno njihovim sposobnostima.

Potreba za agentom, tako je definirana traženim sposobnostima, odnosno zadacima koje agenti trebaju riješiti. Zapravo, agent predstavlja osobu ili automatski proces. Na nižoj razini vođenja projekta, gdje je

fokus na implementaciju rješenja, tj. sustava, analiziraju se potrebne sposobnosti sustava za zadovoljavanje zadataka koje treba sustav moći izvoditi.

Dakle, to su one temeljne, atomizirane sposobnosti, odnosno zadaci sustava. Atomizirani u smislu nedjeljivi na manje pod zadatke, za razliku od pogleda na višoj razini gdje su zadaci i pripadne sposobnosti kompozitni, stoga djeljivi.

Prema tome, ako agenta određuju njegove sposobnosti, prema kojima mu se pridjeljuju zadaci, onda srodne sposobnosti određuju uloge ili grupe agenata.

Uloge definirane na višoj razini vođenja projekta su voditelj projekta, stručnjak za područje, stručnjak za sustav, programer, krajnji korisnik. Na nižoj razini, potrebno je definirati pojedine automatske procese, dakle agente kao dijelove sustava, potrebne da bi se postigli zadaci na toj razini.

Voditelj projekta je osoba koja je dobila odobrenje za realizaciju projekta, proračun za projekt temeljem predstavljenog plana i rokova predstavljenih sponzorima projekta. Voditelj projekta mora imati znanja iz područja vođenja projekata u tehničkom, financijskom i poslovnom smislu. Voditelj projekta mora biti u mogućnosti pratiti plan i realizaciju te upravljati rizicima.

Stručnjak za područje je osoba čije je znanje i iskustvo potrebno kako bi programer ili tim programera ispravno implementirao rješenje, ali i kako bi se sustav modelirao ispravno. Stručnjak za područje, može biti i osoba iskusna u pojedinoj tehnologiji, kao na primjer pojedinih telekomunikacijskih standarda ili elektronike za raspoznavanje identifikacijskih kartica ili ključeva korisnika. Tijekom provođenja ovog projekta, stručnjak za područje standarda identifikacijskih kartica i pripadnih tehnologija je bio upravo mentor na projektu zbog iskustva u spomenutim standardima.

No, je još jedan stručnjak za područje, i to područje elektronike, bio uključen u projekt. Naime, zadužen je za tehničku podršku sustava.

Općenito, stručnjak za sustav (ne u smislu konkretnog sustava ovog projekta) je osoba određena od strane voditelja projekta, a može biti i sam voditelj, a ključna je za definiranje zadaća programeru (ili timu programera). U projektu ovog rada, glavni stručnjak za sustav je ujedno i voditelj, no u fazi planiranja projekta i osmišljavanja koncepta sustava, ovu ulogu su popunjavali i ostali članovi tima, jer tim bio većinom menadžerski. Razlog tome bio je nastajanje projekta u okruženju kolegija Informatički menadžment pa je sukladno tome, uloga članova u timu bila takvog tipa, jer se tražilo postizanje što

boljeg koncepta sustava. Na taj način, više stručnjaka određivalo je dogovorom zahtjeve, zadatke i potrebne sposobnosti agenata za izvedbu, također i delegiranje zadataka agentima.

Programer je osoba koja ima znanja programiranja u pojedinim programskim jezicima, poput Python ili Javascript zbog izrade i integracije programskih rješenja u sustav koji se razvija. Tijekom provođenja ovog projekta, bio je samo jedan programer, ujedno voditelj projekta, no to u praksi nije preporučljivo jer ista osoba određuje kretanje projekta sa više i niže razine, stoga 1 shvaćanje definira ishod projekta. Razlog ovakvom ishodu je bio manjak raspoloživih programera za ovaj projekta, no kao kompenzacija tome, uvedeno je više menadžera, odnosno stručnjaka za sustav pa se prvobitno jednodimenzionalno shvaćanje voditelja mijenja i oblikuje zajedničkim radom i suradnjom ostalih članova tima. Sličan princip provode velike tvrtke, uvođenjem odbora direktora kao najvišeg tijela za upravljanje tvrtkom. Na taj način dijeli odgovornost, znanje i iskustvo za vođenje tvrtkom među pojedincima odbora, čime se osigurava nabolje moguće vođenje.

5.5. Model komunikacije

Model komunikacije među agentima za vrijeme obavljanja zadataka. Ključni elementi ovog modela su transakcije. Važno je definirati tko i na kakav način u timu razmjenjuje podatke. Bez jasno definiranog komunikacijskog protokola u timu se mogu javiti dvije negativne pojave. Prva je mogućnost izražena u početnoj fazi razvoja projekta, a radi se o poplavi informacija koja tada dovodi do zagađenja razvojnog tima idejama koje nisu filtrirane od strane stručnjaka za sustav. Druga mogućnost izraženija je pri kraju razvojnog ciklusa projekta ili u bilo kojoj kriznoj situaciji koja se detektira za vrijeme samog razvoja, a to je potpuni nedostatak komunikacije. Tim koji je zadužen za razvoj često na sebe preuzima previše odgovornosti, a ako nije jasno definiran model komunikacije slijedi se logika „ubij glasnika koji nosi loše vijesti“ pa se problemi jednostavno skrivaju. Zbog ovakvog načina razmišljanja i ponašanja moguće su situacije u kojima propadaju projekti koji se do prije nekoliko tjedana izgledali vrlo uspješni. Model komunikacije može se prikazati shemom kako slijedi na Shemi 2.

Shema 2. – Model komunikacije sustava za upravljanje ključevima (intranet)

5.5.1. Internet stvari ili IOT

Dizajnirani tip programskog rješenja spada u kategoriju IOT (eng. *Internet Of Things*) aplikacija, jer se izvodi na minijaturnom uređaju sa specifičnom ulogom očitavanja RFID oznaka na privjescima ključeva, a ujedno pruža komunikaciju prema vanjskom svijetu putem intraneta (lokalno postavljene mreže). Naime, samom uređaju je dodijeljena fiksna IP (eng. *Internet Protocol*) adresa 192.168.0.19 na lokalnoj mreži što omogućuje korisnicima da se povežu sa uređajem putem računala, tableta ili mobilnih uređaja i izravno dobiju podatke iz aplikacije, u realnome vremenu. Aplikacija koja pruža korisnicima uvid zapravo je web aplikacija, dakle pokreće se u internetskom pregledniku klijentskog uređaja koji se povezuje sa izvornim uređajem koji se ponaša kao server. Jedini preduvjet je da su klijentski uređaji povezani na lokalnu mrežu i naravno, za uređaje koji se spajaju bežično da je jačina signala rutera dovoljno jaka, odnosno da se uređaj nalazi u pojasu signala.

5.5.2. Komunikacija putem HTTP protokola

Prema tome, kako bi se osigurao pristup podacima neovisno o platformi uređaja koji treba prikazati i koristiti te podatke, programsko rješenje je izvedeno kao web aplikacija, a sam model komunikacije tog

programskog rješenja u radnom okruženju se zasniva na HTTP (eng. *Hypertext Transport Protocol*) komunikacijskom protokolu. Taj protokol omogućava pregledniku klijentskog uređaja da pošalje HTTP zahtjev serveru, u ovom slučaju izvornom uređaju sa RFID čitačem, koji šalje odgovor s traženim podacima natrag klijentskom uređaju. Prema tom protokolu, zadani izlazni broj porta servera je 80.

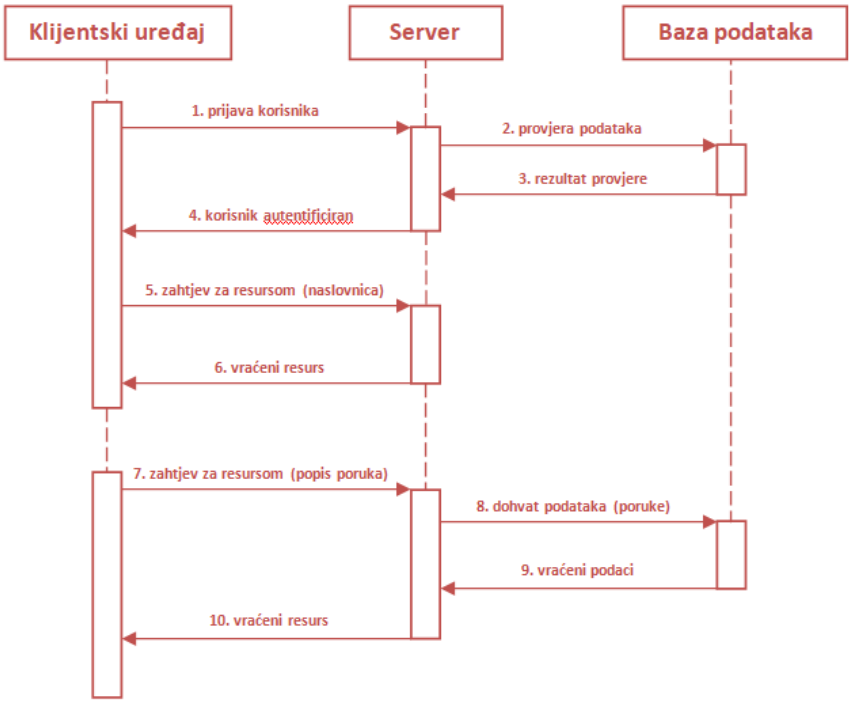
Prema troslojnoj arhitekturi web aplikacija, moderna web aplikacija sastoji se od klijentskog ili prezentacijskog sloja (klijentski uređaji, odnosno agenti preko interaktivnog sučelja šalju zahtjeve za hipermedijskim sadržajem putem HTTP protokola), zatim aplikacijski server ili poslovna logika kao srednji sloj i konačno, sloj podataka ili baza podataka. Tako je slučaj i za web aplikaciju, odnosno hipermedijski sustav na uređaju Raspberry Pi.

Kako bi se postigla što efikasnija razmjena informacija između klijenta i servera, moguće je minimizirati količinu informacija koje se šalju ako su one redundantne, tj. mogu se slati samo one promjene stanja sučelja aplikacije na klijentskoj strani onda kada to korisnik zatraži i kada je server u mogućnosti te promjene izvršiti.

Za takvu komunikaciju postoje određene konvencije koje omogućuju jednostavniju implementaciju, a to su kreiranje REST API krajnjih točaka na poslovnoj strani i primjenu AJAX poziva za dohvat resursa sa tih API lokacija. O tome slijedi u nastavku.

5.5.3. AJAX pozivi i REST API krajnje točke

AJAX (eng. Asynchronous Javascript and XML) je skup metoda u razvoju web aplikacija, čija je uloga implementacija asinkronih web zahtjeva od klijenta ka poslužitelju i zaprimanje asinkronih odgovora na iste. U osnovi to znači da AJAX omogućuje unutar web aplikacije slanje zahtjeva prema serveru tako da se čekanje na odgovor smješta u pozadinski proces, dok sama aktivna stranica nastavlja dalje sa radom kao da je odgovor na poslani zahtjev već stigao, vidjeti Dijagram 3. To omogućuje korisniku da se interaktivno služi trenutnim sadržajem stranice, dok mu ne dođe odgovor sa servera s novim podacima koji će se tada prikazati dinamički na stranici, bez osvježavanja stranice što je inače potrebno u klasičnoj web aplikaciji.



Dijagram 3 – Dijagram slijeda AJAX poziva prilikom akcije prijave korisnika u sustav i preuzimanja resursa

Opisano asinkrono ponašanje je moguće izvesti na nekoliko načina u praksi, zbog čega AJAX obuhvaća nekoliko metoda, a to su: primjenu funkcije povratnog poziva (eng. *callback*) i primjenu funkcije obećanja (eng. *promise*). Metode se razlikuju prema porijeklu, jer funkcija povratnog poziva spada u funkcije višega reda i one su specifične za određene programske jezike.

Funkcija obećanja je zapravo arhitekturalni obrazac kod objektnog dizajna programiranja, gdje se oslanja na primjenu posebnih objekata na temelju kojih se poziva, objekata pod nazivom proxy model (eng. *Proxy*).

5.5.3.1. Funkcije povratnog poziva (eng. Callbacks)

Funkcija povratnog poziva, u osnovi, je funkcija koja se proslijedi nekoj drugoj funkciji kao parametar, tada ju ta druga funkcija može pozvati nakon završetka izvođenja, dakle kada se druga funkcija kompletira sa izvođenjem ona pozove prvu funkciju koja joj je proslijeđena kao parametar. To može biti izvedeno na 2 načina: sinkrono i asinkrono. Kod sinkrone funkcije, prvi proces mora se završiti da bi se zatim pozvala funkcija povratnog poziva, te onda nastavio rad programa, dakle osnovni program se

zaustavlja. Ako je asinkrona, kad se pozove, osnovni program nastavlja sa radom i pri završetku te asinkrone funkcije dojavi se osnovnom programu rezultat funkcijom povratnog poziva.

5.5.3.2. Funkcije obećanja (eng. Promises)

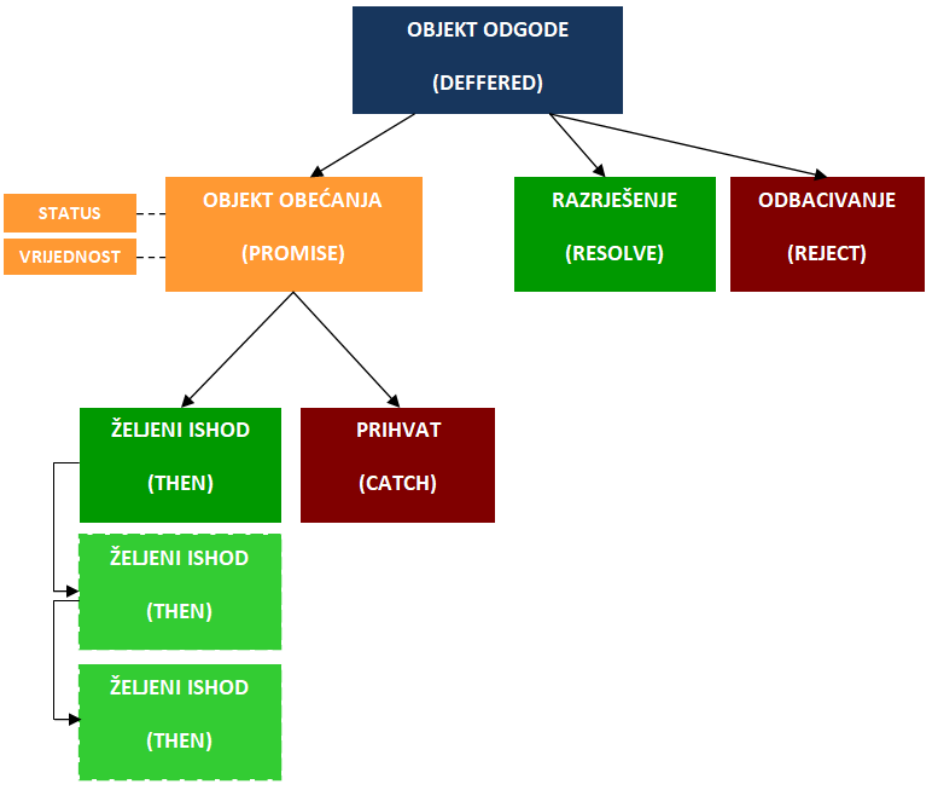
AJAX može biti izveden i funkcijama obećanjima. Da bi se to izvelo, upotrebljava se poseban objekt odgode (eng. *deffered*) koji sadrži polja razrješenja (eng. *resolve*) i odbacivanja (eng. *reject*), čija je uloga omogućiti praćenje AJAX poziva, odnosno određivanje stanja poziva, te u slučaju da je stanje uspješno razriješeno poziva se funkcija obećanja sa željenim ishodom (eng *then*), suprotno, ako je odbačen poziv zbog pogreške poziva se funkcija prihvata (eng. catch). Pogledati Dijagram primjeraka 4 za ilustraciju.

Prednost ove metode je što se može nanizati bilo koji željeni broj funkcija sa željenim ishodom sekvencijalno, tako kad se jedna razriješi njen rezultat se proslijedi sljedećoj u sekvenci, no ako se bilo koja odbaci sve pogreške će biti „uhvaćene“ i proslijeđene funkciji prihvata. Ovo je također moguće izvesti sa funkcijama povratnog poziva, no onda je potrebno nizati funkcije kao parametre tim istim funkcijama međusobno, što postane ne pregledno i zbog toga se teže s time programeru snaći i postaje teže pratiti.

Funkcije obećanja jednostavno omogućuju nizanje funkcija na istoj razini unutar koda pa su lakše za praćenje i intuitivne za shvaćanje, štoviše podsjećaju na višestruko grananje (eng. *switch*) poznato u mnogim programskim jezicima.

Nadalje, same funkcije željenog ishoda i prihvata poziva objekt samog obećanja (eng. *promise*) koji je također polje početnog objekta dogode, no ovaj objekt prati stanje i trenutnu vrijednost operacije, tj. AJAX poziva. Na taj način kada dođe do promjene stanja poziva, objekt obećanja proslijedi rezultat roditeljskom objektu odgode koje ovisno o promjeni aktivira polje razrješenja ili odbacivanja i eliminira drugo polje (izgubi ga potpuno).

Tada objekt odgode to prosljeđuje podređenom objektu obećanja koje eliminira granu funkcija obećanja sukladno polju ishoda koje je nadređeni objekt uklonio. Tada objekt obećanja može izvršiti preostalu granu funkcija obećanja.



Dijagram 4 – Dijagram primjeraka za objekt odgode, objekt obećanja i funkcije obećanja

5.5.3.3. Arhitektura transfera stanja reprezentacije ili REST

Arhitektura transfera stanja reprezentacije ili REST (eng. *REpresentational State Transfer*) je takva arhitektura web aplikacije ili servisa koja predstavlja konvenciju u načinu razmjene informacija između 2 sustava putem interneta na način da se razmjenjuju informacije u obliku tekstualnih reprezentacija web resursa upotrebom jedinstvenih i predefiniranih operacija bez pamćenja stanja sa serverske strane (eng. *Stateless protocol*).

Neke od tih predefiniranih operacija, odnosno metoda su: GET, POST, PUT, DELETE. HEAD itd. Web servisi koji podržavaju REST (eng. *RESTful services*) pružaju resurse definirane navedenim metodama dohvata, ali razlikovanih prema jedinstvenoj web adresi resursa ili URL (eng. *Unified Resource Location*). Ove resurse, odnosno krajnje točke REST podržanih servisa mogu biti kodirane tako da vraćaju resurse u odgovorima XML (eng. *eXtended Markup Language*); HTML (eng. *Hypertext Markup Language*) ili JSON

(eng. *JavaScript Object Notation*) tipa. Tip odgovora, odnosno reprezentacija podataka se mora specificirati u konfiguraciji web servera.

HTML je osnovni kodni jezik za izradu internetskih stranica, svaka stranica je sačinjena od koda pisanog tim jezikom i tako je isporučena klijentskoj strani. HTML se sastoji od ključnih i predefiniranih ključnih kodnih riječi koje se nazivaju HTML elementi. Oni definiraju strukturu i značenje dijelova internetske stranice i upravo oni zadaju internetskom pregledniku pravila prema kojima će ih preglednik prikazati. Naime, internetski preglednici formiraju posebnu strukturu za pamćenje hijerarhije HTML elemenata stranice koja se naziva DOM (eng. Document Object Model). Bez HTML elemenata internetski preglednik ne bi mogao prikazati sadržaj stranice kodiran unutar samih elemenata (dakle, oni se ponašaju kao čvorovi DOM stabla).

Kao i HTML, XML ima elemente, no s tom razlikom što se ti elementi sami definiraju, odnosno programer ih sam smišlja, pa se često upotrebljavaju unutar konfiguracijskih datoteka određenih web tehnologija za realizaciju servera, poput ASP.NET tehnologija.

JSON, s druge strane, predstavlja pravu tekstualnu reprezentaciju nekog objekta sa servera, zbog čega se i koristi pa se često takvi resursi, odnosno krajnje točke koje vraćaju odgovore u obliku JSON objekata nazivaju i JSON (REST) API (eng. *Application Programming Interface*). Programsko rješenje iz projekta ovog diplomskog rada je konstruirano kao REST API temeljen na JSON reprezentacijama u odgovorima.

5.6. Model dizajna

Programsko rješenje je realizirano kao kombinacija web servera posluženog izravno sa mini računala (IOT uređaja) unutar lokalne mreže (intranet) i programa za upravljanje hardverskim signalima modula sa RFID čitačem. Mini računalo u upotrebi se naziva Raspberry Pi, vrlo je ekonomično i da se lako nabaviti, ujedno je i modularno rješenje jer se pored programske podrška i hardverska podrška da nadograditi upotrebom vanjskih modula koji se povezuju putem žica na posebnu ulazno-izlaznu sabirnicu uređaja. U nastavku slijedi opis uređaja i njegova primjena za izvedbu programskog rješenja projekta i samog sustava.

5.6.1. Računalo Raspberry Pi

Raspberry Pi je računalo veličine kreditne kartice (pogledati Sliku 6 na strani iza) koje je u potpunosti smješteno na jednoj matičnoj ploči, dakle spada u kategoriju SBC računala (eng. *Single Board Computer*). Razvio ga je fond pod nazivom „Raspberry Pi Foundation“ u Ujedinjenom Kraljevstvu s ciljem promoviranja učenja osnova računalnih znanosti u školama. Štoviše, osnovna ideju o jeftinom malom računalu su skovali studenti Sveučilišta u Cambridgeu, Eben Upton, Rob Mullins, Jack Lang i Alan Mycroft, zato jer su se studenti koji žele studirati računarstvo tada, od godine 2000., sve više okretali isključivo programiranju, za razliku od 1990.-tih kada su većina studenata bili iskusni programeri hobisti sa znanjem hardvera i elektronike. (V. Pleština, 2017.) Shema i dizajn tiskane pločice ili PCB (eng. *Printed Circuit Board*) su javno dostupni na službenim stranicama organizacije, a od originalnog dizajna do danas razvilo se nekoliko različitih modela.

Računalo podržava i grafičko ili GUI (eng. Graphical User Interface) i tekstualno ili CLI (eng. Command Line Interface) sučelje pa se mogu jednostavno zadati različite naredbe u radu sa CLI za različite sistemske zadatke, kao na primjer provjera modela i verzije računala, kao što je to vidljivo na Slici 7. Oznaka broja revizije (eng. *Revision number*) određuje generaciju, tj. model računala, a ti se podaci mogu pronaći i usporediti sa podacima dostupnim na internetu, važno je pronaći tablicu koja sadrži sve brojeve revizija i pridružene nazive modela računala. Na primjeru sa Slike 7 dobiven je revizijski broj 000e, što kada se pogleda na spomenutoj tablici dostupnoj na internetu, određuje da se radi o Modelu B Rev 2 s 512 MB RAM, no više o modelima u sljedećem dijelu rada.

Slika 6 – mini računalo Raspberry Pi 2 model B rev. 2

```
processor       : 0
model name      : ARMv6-compatible processor rev 7 (v6l)
BogoMIPS        : 2.00
Features        : half thumb fastmult vfp edsp java tls
CPU implementer : 0x41
CPI architecture : 7
CPU variant     : 0x0
CPU part        : 0xb76
CPU revision    : 7

Hardware        : BCM2708
Revision        : 0002
Serial          : 00000000fc0c2f7e
```

Slika 7 – ispis naredbe računala Raspberry Pi za provjeru modela i verzije

Koncept računala Raspberry Pi, kao što je već navedeno, je razvio fond „Raspberry Pi Foundation“, koji i dalje razvija, usavršava i nadgleda cjelokupni projekt pa je tako razvijeno dosada 13 njihovih modela od kojih su 6 trenutno dostupni u prodaji putem službene stranice fonda. Karakteristike tih 6 modela dostupnih su zapisanih na Tablici 2, no pored samih modela računala dostupni su i raznorazni dodaci i vanjski moduli za povezivanje sa tim modelima. Dakle, kao što je vidljivo na Tablici 2, službeni modeli računala Raspberry Pi su slijedom: Raspberry Pi 1 model A+, Raspberry Pi 1 model B+, Raspberry Pi 2 model B, Raspberry Pi Zero, Raspberry Pi Zero W i Raspberry Pi 3.

Provjeriti model računala Raspberry Pi je moguće pomoću provjerom sadržaja datoteke /proc/cpuinfo pohranjene na sustavu Raspbian, kao što je to prikazano na Slici 7 sa prethodne strane. Dakle, kao što je vidljivo sa Slike 7, uređaj koji je korišten za potrebe projekta je Raspberry Pi 2 model B sa 512 MB RAM.

Računalo Raspberry Pi pokreće njegov vlastiti operacijski sustav naziva Raspbian, a zasniva se na drugom opće poznatom operacijskom sustavu otvorenog tipa koda, a to je Debian Linux.

5.6.2. Programski jezik Python

Programski jezik koji se koristi za oba dijela programskog rješenja je Python, to je skriptni jezik, dakle oslanja se na gotove skripte i biblioteke te je otvorenog tipa (eng. *open-source*). Odabran je zbog neovisnosti o operativnom sustavu na kojem se izvodi, interpretirani je jezik pa nije potrebno prevoditi (za razliku od kompiliranih jezika) za određenu platformu i operativni sustav, jer njegov interpreter CPython pruža podršku za sve operativne sustave.

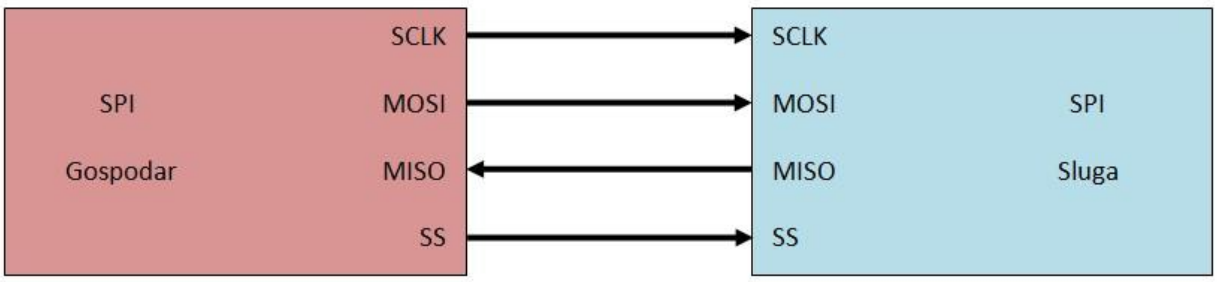
Kao i drugi interpretirani jezici, svaku instrukciju koda zapisanog u programskom jeziku Python njegov interpreter izravno izvodi na stroju (premda ju prethodno interpreter prevodi u strojni jezik), no samo izvođenje je sporije nego u programskim jezicima bližim hardveru poput C ili C++, ipak brzina nije prioritet za izvedbu programskog rješenja ovog projekta.

Također, jedan od razloga odabira jezika Python je modularnost, prema CommonKADS metodologiji, tj principu otvorene arhitekture koju metodologija predlaže, strukturiranje koda prema pojedinim aktivnostima, odnosno zadacima upotrebom modula, prema dokumentaciji programskog jezika to su tzv. Python moduli.

5.6.3. Model gospodara-roba i serijska komunikacija (SPI)

Python ima širok izbor biblioteka koje sadrže gotova sučelja za rad izravno sa hardverom, odnosno za primjenu kod IOT rješenja, štoviše Python podržava biblioteke pisane u programskim jezicima C ili C++, koje se skupno zovu ekstenzije. Jedna takva biblioteka korištena je i u ovom projektu, naziva SPI-py (eng. *Serial Peripheral Interface library*) koja predstavlja programsko sučelje za upravljanje i sinkronu serijsku komunikaciju uz pomoć ugrađenog SPI čipa i SPI sabirnice (eng. *Serial Peripheral Interface bus*), a koristi se primarno u tzv. ugrađenim sustavima (eng. *Embedded systems*) koji predstavljaju sustave ugrađene u 1 stroj ili računalo, čija se logika rada može zasnivati na 1 programu (Ž. Panian, 2005.). SPI sučelje je razvila tvrtka Motorola 1980-ih godina i postala je standard za serijsku komunikaciju između uređaja na međusobno kratkoj udaljenosti.

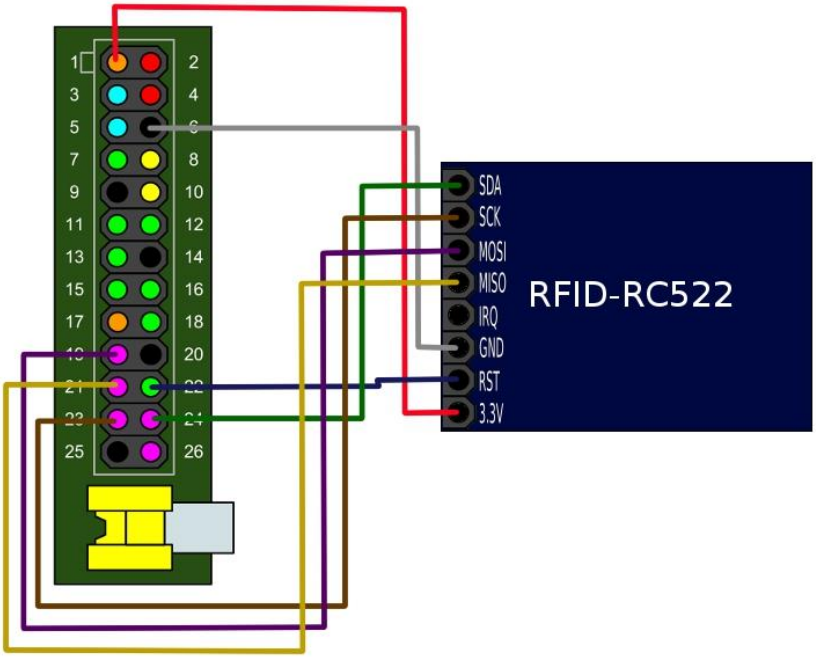
Ta serijska komunikacija je važna jer se koristi za uspostavu odnosa gospodara-roba (eng. *Master-Slave*) između računala Raspberry Pi (gospodar) i vanjskog modula sa RFID čitačem (rob). Ovaj odnos je ključan za integraciju i komunikaciju RFID čitača sa mini računalom Raspberry Pi jer određuje standard komunikacije, odnosno omogućuje dvosmjerni prijenos podataka ili FDX (eng. *Full Duplex*), pogledati Shemu 3.



Shema 3 – model gospodara-roba u izvedbi preko SPI sabirnice

Model gospodara-roba je model komunikacijskog protokola u kojem jedan uređaj ili proces ima jednosmjernu kontrolu nad jednim ili više drugi uređaja ili procesa. Kad je uspostavljen, smjer komunikacije je uvijek od gospodara prema robu. (Ž. Panian, 2005.)

Zato odnos gospodara-roba zahtjeva i korištenje točno određenih pinova GPIO sabirnice računala Raspberry Pi, kao i prikladno povezivanje sa pinovima vanjskog modula sa RFID čitačem, pogledati Sliku 8 koja prikazuje spomenuti način spajanja. Model čitača koji se primjenjuje je RFID-RC522, tj. MFRC-522.

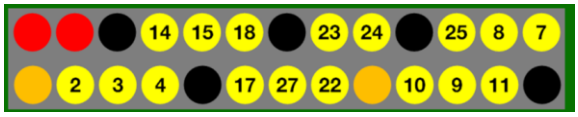


Slika 8 – Način spajanja računala Raspberry Pi sa modulom RFID-RC522 prema modelu gospodara-roba

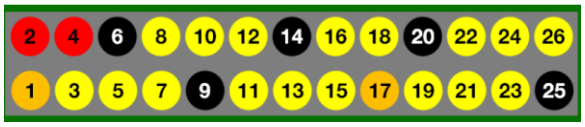
Također, serijska komunikacija je izvedena preko GPIO sabirnice (eng. *General Purpose Input Output bus*) s priključnim kontaktima, često se nazivaju pinovi, na koje se priključuju vodiči koji vode signale iz i

prema vanjskom modulu i njegovim priključnim kontaktima za serijsku komunikaciju. Kako bi se omogućila upotreba GPIO pinova na Raspberry Pi računalu potrebno je omogućiti serijsku komunikaciju odnosno u datoteci /etc/modprobe.d/raspi-blacklist.conf, koja predstavlja konfiguraciju osnovnih postavki Raspberry Pi računala, omogućiti spi_bcm2708, odnosno ugrađeni čip za serijsku komunikaciju preko pinova GPIO sabirnice.

Općenito, postoje 2 standarda rasporeda GPIO pinova na mini računalu Raspberry Pi, to su onaj zadani kojeg računalo raspoznaje, kao na Slici 9 a) i drugi Broadcomov raspored, dan na Slici 9 b), koji je osmišljen od istoimene tvrtke koja se bavi proizvodnjom elektronike, posebice uređaja za bežičnu komunikaciju.

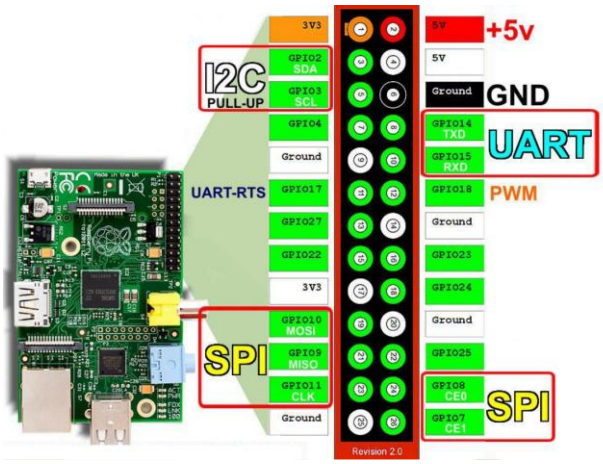


Slika 9 a) – Raspored GPIO pinova definiran na način kako ih računalo raspoznaje



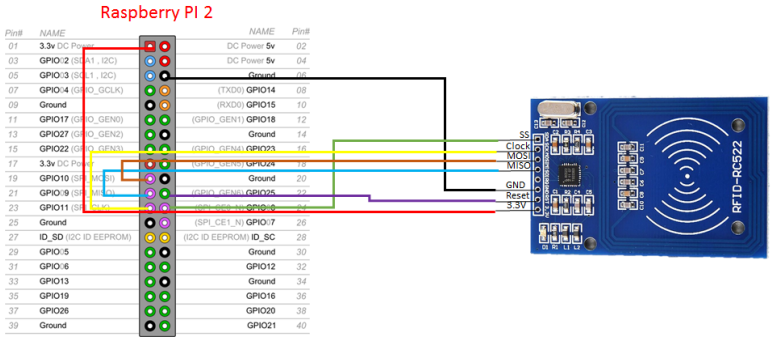
Slika 9 b) – Raspored GPIO pinova definiran prema BCM (eng. *Broadcom*) modelu

Detaljan prikaz GPIO sabirnice sa rasporedom pinova prema BCM modelu može se vidjeti na Slici 10, sljedeća strana. Iz tog pregleda vidljivo je kako se za serijsku komunikaciju (prema SPI standardu) upotrebljavaju pinovi 19 MOSI, 21 MISO, 23 CLK, te opcionalno 24 CE0 i 26 CE1. Pin 23, označen kao CLK ili u nekoj literaturi SCLK je za određivanje takta (eng. *Clock*) i služi za sinkronizaciju signala između RFID modula i računala Raspberry Pi. Pinovi 24 i 26 mogu služiti za prijenos podataka serijski u oba smjera, tj. to su SDA pinovi (eng. *Serial DAta line*). No najvažniji su pinovi MOSI i MISO. Pin 19 MOSI (eng. *Master Output Slave Input*) služi za prijenos izlaznih podataka od gospodara prema sluzi. MISO (eng. *Master Input Slave Output*) služi za prijenos izlaznih podataka od sluge ka gospodaru.



Slika 10 – Detaljan prikaz pinova GPIO sabirnice rasporeda prema BCM modelu

Sada kada su definirani potrebni pinovi za SPI komunikaciju na strani računala Raspberry Pi, postaje jasniji način povezivanja sa vanjskim modulom sa RFID čitačem, što je detaljno prikazano na Slici 11.



Slika 11 – Detaljan prikaz povezivanja računala Raspberry Pi i modula RFID-RC522

5.6.4. Sesije ili preuzimanja ključeva

Serijska komunikacija, odnosno odnos gospodara-roba omogućuje slanje podataka očitanih sa modula RFID-RC522, ali potrebno je definirati koji su to podaci koje čitač osluškuje. Zato je potrebno podrobnije proučiti i objasniti RFID tehnologiju.

RFID (eng. *Radio-Frequency IDentification*) je tehnologija koja koristi radio frekvenciju kako bi se razmjenjivale informacije između prijenosnih uređaja ili memorija i izvornih centralnih računala. RFID sustav se obično sastoji od tokena ili taga (eng. *Tag*) koji sadrži elektronički pohranjene podatke, antenu

koja komunicira s tagovima i kontroler, koji upravlja i nadzire komunikacijom između antene i izvornog računala.

Modul RFID čitača koji se primjenjuje u projektu sadrži antenu i kontroler, te pomoću njih šalje radio signale koji „pobuđuju“ tagove koji se približe anteni čitača. Jačina signala raste s kvadratom udaljenosti, stoga je domet ovisan o kvaliteti antene, odnosno cijeni. Antena čitača MFRC-522 je dometa do 10cm, jer je sam modul čitača namijenjen za tzv. „uradi sam majstore“ ili ljude kojima je hobi elektronika.

Upravo zato i ovaj segment spada pod model sposobnosti prema podjeli CommonKADS metodologije, jer je potrebno da agent koji rukuje čitačem i ostalom tehničkom podrškom ima sposobnost, iskustvo i znanje elektronike, ali i lemljenja i povezivanja elektroničkih komponenti za rješavanje zadataka postavljanja tehničke podrške sustava.

Također, na nižoj razini, agent kao automatski proces sustava za prikupljanje informacija iz vanjskog svijeta, mora imati sposobnost programskog upravljanja vanjskim modulom čitača, odnosno mora imati pristup signalim koji idu putem GPIO sabirnice računala Raspberry Pi i to sve iz programskog koda. Ta sposobnost ostvarena je primjenom ranije spomenute biblioteke koda za serijsku komunikaciju uz pomoć programskog jezika Python, a zove se SPI-py.

No, pored te biblioteke potrebna još jedna druga biblioteka, također s podrškom za programski jezik Python, koja je direktno korištena u izvedbi agenta za upravljanje čitačem, a to je biblioteka MFRC522-python. Naime, ova biblioteka omogućuje upravljanje MFRC-522 modulom RFID čitača za čitanje i pisanje informacija sa i na RFID tagove, no i razne druge mogućnosti, poput enkripcije. Time dakle, agent za upravljanje čitačem dobiva cijelu lepezu sposobnosti za rješavanje raznih zadataka vezanih uz RFID tehnologiju i prikupljanje informacija iz realnog svijeta. Sama biblioteka raspoloživa je na internetskom portalu slobodnog i otvorenog koda ili FOSS (eng. *Free Open-Source Software*) zvanog Github.

Nakon što se preuzme biblioteka MFRC522-python sa portala Github, moguće je uvesti (eng. *import*) njezine Python module u programski kod agenta čitača ili bilo kojeg drugog dijela sustava, npr. serverskog dijela web aplikacije koja poslužuje informacije. Tako agent ima pristup bilo kojoj funkcionalnosti biblioteke, čime ostvaruje tražene sposobnosti upravljanja RFID modulom.

Kako GPIO sabirnica preko koje se ostvaruje komunikacija sa RFID modulom može imati i druge uređaje povezane, koji možda dijele funkciju s već ugrađenim ostalim komponentama na tiskanoj pločici računala Raspberry Pi pa je potrebno ograničiti bilo kakve elektroničke komponente i cjelokupnu

tehničku podršku na računalu Raspberry Pi tako da su sve dodatne komponente automatski isključene, te da se dopusti primanje signala za komponentu samo kad se zatraži.

Taj način rada se zove stablasta konfiguracija uređaja ili DT (eng. *Device Tree*), a omogućuje se u postavkama za paljenje računala Raspberry Pi. Potrebno je otvoriti datoteku /boot/config.txt ili ju napraviti ako već ne postoji, te u njoj unijeti liniju „device_tree=on“ što će omogućiti DT konfiguraciju. Zatim treba spremiti te izmijene i za svaki slučaj ponovno pokrenuti uređaj Raspberry Pi. Sada je bilo kakav dio tehničke podrške nešto sigurniji od nenamjerne štete.

Konačno, sustav ima mogućnost prikupljanja informacija iz realnog svijeta, no potrebno je definirati koje informacije treba prikupljati. Kako je iz zahtjeva projekta potrebno pratiti preuzimanje ključeva, očito je da će se trebati pamtiti nekakav identifikator ključa, što predstavlja serijski broj RFID taga na privjesku ključa. Također, potrebno je pratiti koji korisnik preuzima ključ od prostorije, što je moguće praćenjem serijskog broja RFID kartice s ugrađenim tagom, koja se daje profesorima i ostalom osoblju koje treba imati pristup ključevima. Osim te dvije informacije, sesija mora imati definirano vrijeme preuzimanja, no i vrijeme vraćanja, tj. zatvaranja sesije. Dodatno, za potrebe sortiranja i filtriranja rezultata, sesije mogu imati posebnu zastavicu, u osnovi istinitosnu vrijednost za to je li ključ vraćen ili ne.

Prema tome, sesija se sastoji od serijskog broja RFID taga ključa, serijskog broja RFID taga kartice korisnika, vremena početka, vremena zatvaranja sesija i statusa, odnosno zastavice provjere je li ključ vraćen ili ne. Ilustracija prikupljanja informacija o sesiji korisnika dana je na Shemi 4.

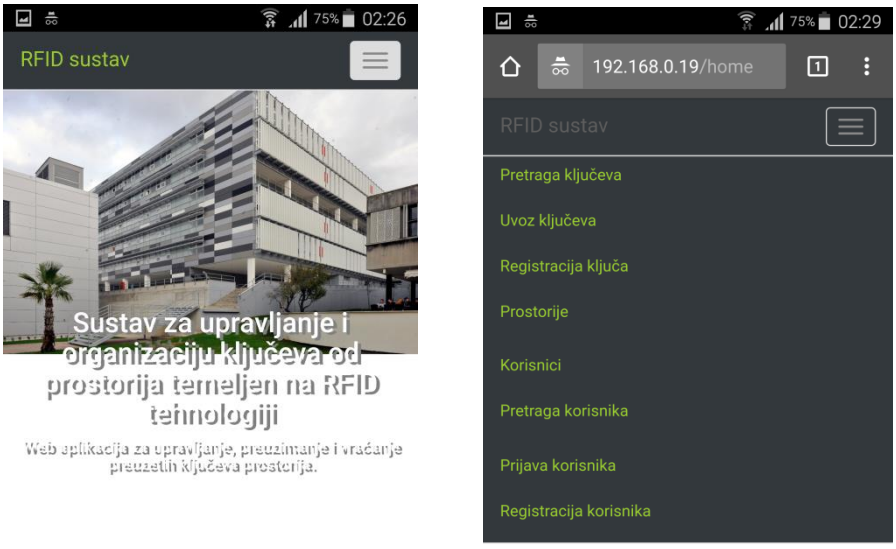
Shema 4 – Shema tehničkog sustava (prikaz izvedbe akcije preuzimanja ključeva)

5.6.4. Prezentacija podataka

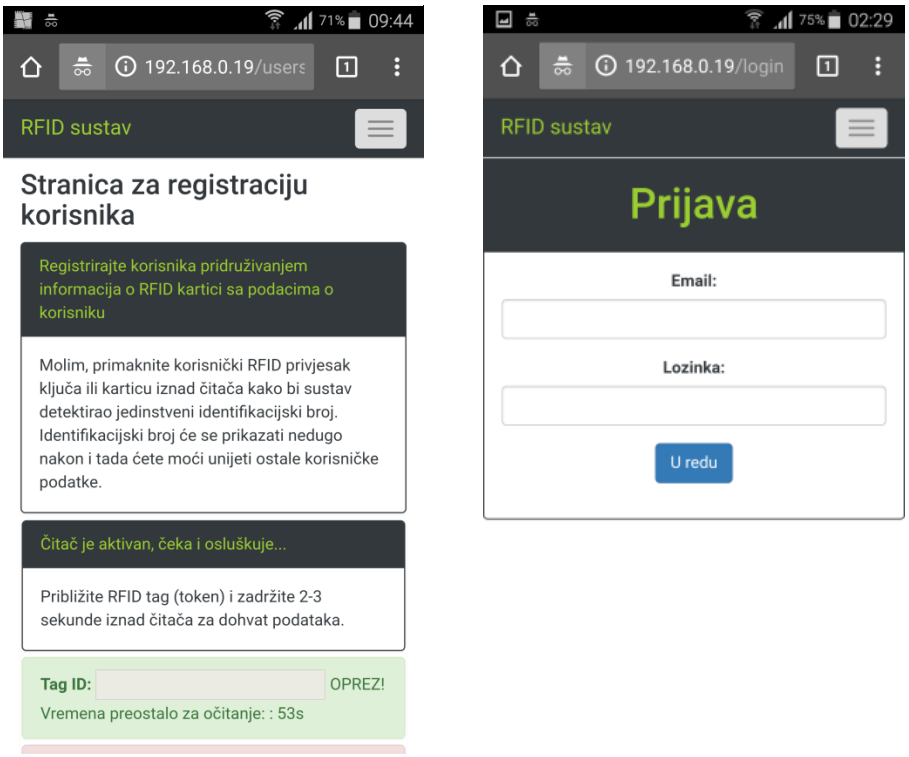
Podatci koje sustav pruža korisniku posluženi su kao web aplikacija. Preciznije, web aplikacija dizajnirana tako da je prilagodljiva bilo kojem uređaju, a od početka je ciljana biti prikazana prvo na mobilnim uređajima, dakle dizajn web aplikacije ima funkcionalnu ulogu. Dizajn internetskih stranica bavi se rasporedom elemenata stranice na zaslonu uređaja na kojem se prikazuje.

Kako je cilj dizajna aplikacije prilagodljiv bilo kojem uređaju, primijenjen je tzv. responzivni dizajn. Responzivni dizajn podrazumijeva takvo uređivanje izgleda internetske stranice da se u pregledniku bilo kojeg uređaja ne naruši koncepcija elemenata i funkcionalnost. Dakle, mora biti svejedno posjeti li se internetska stranica sa mobitela, tableta, osobnog računala ili nekog drugog uređaja različitih dimenzija zaslona.

Konkretno, responzivni dizajn je pristup koji zahtjeva da se sučelje internetske stranice prilagodi korisničkim navikama i potrebama s obzirom na veličinu zaslona, platformu i orijentaciju. Konvencija je kod takvog dizajna krenuti s definiranjem dizajna za mobilne uređaje prvo, potom za ostale platforme. Prema tome, dizajn aplikacije sustava iz projekta prvo je definiran za prikaz na pametnom telefonu, kao na Slikama 12 a) i 12 b).



Slika 12 a) – Prikaz dizajna na moblinim uređajima (lijevo naslovnica, desno izbornik s navigacijama)



Slika 12 b) – Stranica za registraciju lijevo i prijavu desno, dizajn na mobilnom uređaju

Dakle, sa Slika 12 a) i 12 b) vidljiv je izgled sučelja web aplikacije pokrenute u pregledniku pametnog telefona. Očigledno da je zbog malog prostora za prikaz, komponente i podatke potrebno mudro grupirati i prikazati samo ono što je najnužnije. Bilo kakve dodatne akcije treba omogućiti korisniku odabirom na dinamičkom izborniku. Upravo taj dinamički izbornik, vidljiv na Slici 12 a), dostupan je odabirom botuna koji izgledom podsjeća na hamburger (eng. *Hamburger menu*). Kada nije odabran, izbornik je uvučen, odnosno sakriven, a kada se odabere dodirom spusti se, tj. prikaže korisniku cijela navigacija s ponuđenim akcijama.

Pored navigacije i izbornika, ključan problem za reducirani responzivni dizajn je problem prikaza formi za ispunu podataka (primjerice kod prijave korisnika u sustav) i također prikaz velikih ulomaka teksta. Kada se forme prikazuju bez vođenja računa o širini zaslona uređaja na kojem se prikazuju, polja za unos često mogu biti šira od zaslona pa sa za bilo kakav dulji unos korisnik mora mučiti povlačenjem zaslona u stranu, što nije intuitivno i samo po sebi za korisnika predstavlja poteškoću u interakciji sa aplikacijom. Zato je generalno pravilo kod responzivnog dizajna smanjiti broj interakcija koje korisnik treba napraviti koje zahtijevaju horizontalni pomak po stranici, odnosno povlačenje zaslona u horizontalnom smjeru.

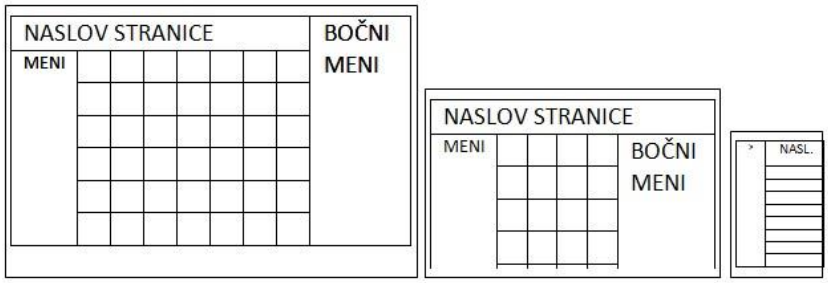
Prema tome, forme se zadaju relativno određene širine polja za unos i ostali komponenti u odnosu na širinu zaslona.

Problem prikaza velikog ulomka teksta kada se ne vodi računa o širini zaslona, uzroku opet iste poteškoće za korisnika, gdje korisnik mora klizati po zaslonu horizontalno kako bi pregledao ostatak teksta. No, rješenje tog problema je malo drugačije od problema formi, jer pored relativnog zadavanja širine ulomka u odnosu na širinu zaslona, potrebno je zadati da se redci ulomka prekinu na onim mjestima uz rub zaslona i da se prenesu u sljedeći redak, dakle treba izvesti dinamičko preslagivanje teksta (eng. *word-wrap*).

5.6.4.1. Bootstrap (Javascript, CSS)

Prethodno su opisani samo osnovni problemi koje responzivni dizajn rješava, postoje i ostali, stoga je jasno kako web aplikacije s većim brojem podataka, odnosno s više sadržaja zahtijevaju puno posla za održavanje pa se mora pribjeći gotovim ili polu-gotovim rješenjima.

Jedno takvo rješenje je upotreba skupa biblioteka koda za stilizaciju i responzivni dizajn pod nazivom Bootstrap, koju su izradili nekolicina programera iz Twittera (društvena mreža i mikro-blog). Ta biblioteka, odnosno programski koncept rješenja (eng. *framework*) obuhvaća gotova rješenja u programskim jezicima HTML, CSS i Javascript koji su neizostavni za izradu bilo kakvih web aplikacija (koncept rješenja na Slici 13).



Slika 13 – programski koncept rješenja Bootstrap (eng. *front-end framework*)

Kao što je vidljivo na Slici 14, koncept rješenja Bootstrap omogućuje naizgled konzistentan i intuitivan raspored elemenata sučelja web aplikacije ili stranice na uređajima različitih dimenzija zaslona, dakle pogađa razne platforme uređaja. Upravo zato je korišten, jer omogućuje zadovoljiti jedan od glavnih

zahtjeva projekta, a to je zadovoljiti široku platformu uređaja i proširiti doseg sustava na što više korisnika od značaja.

Ilustracija poviše prikazuje Bootstrap prilagodljivu tablicu prema kojoj se poravnavaju elementi na uređajima različitih dimenzija zaslona. Raspored pri tome određuje dizajner, kao i pragove, tj. prijelazne dimenzije zaslona prilikom kojih se mijenja raspored elemenata na sučelju.

Od navedena tri jezika koja obuhvaća Bootstrap, HTML je prethodno već objašnjen, Javascript je ponešto spomenut, no CSS nije dosada naveden u radu. Ukratko, CSS (eng. *Cascading Style Sheets*) je skriptni jezik za preglednik koji određuje način, često spomenut kao stil, prikazivanja elemenata sučelja unutar preglednika uređaja na kojem se web stranica prikazuje. Kasnije se Bootstrap proširio pa sada uključuje i naprednije izvedenice CSS jezika.

Javascript, također skriptni jezik, je programski jezik, u početku pisan za preglednik, no u moderno vrijeme postoje i druge primjene, a služi za dinamičko posluživanje sadržaja na stranicu, korisničke interakcije, animacije i ostalo što statične HTML stranice ne mogu. Javascript postiže navedeno, jer ima direktan pristup DOM (eng. *Document Object Model*) strukturi web stranice.

Bootstrap je uveden kao odgovor na problem prikaza sadržaja web aplikacije na različitim uređajima, stoga na nižoj razini sustava predstavlja sposobnost prilagodbe sustava, dakle upada u domenu modela sposobnosti kao generičkog modela CommonKADS metodologije. Također, programer ili dizajner kao agent mora imati sposobnost, tj. iskustvo upotrebe tog programskog okvira pa je i to segment modela sposobnosti.

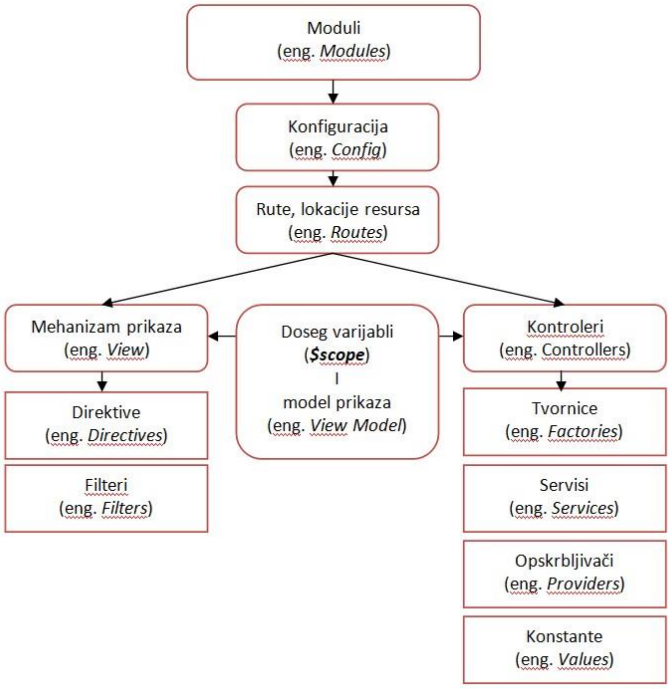
5.6.4.2. AngularJS (Javascript)

Pored olakšanog razvoja responzivnog dizajna uz Boostrap, potrebno je pojednostavniti rad s dinamičkim prikazom sadržaja na klijentskoj strani, implementaciji AJAX zahtjeva i minimizaciji funkcionalnih grupa elemenata ili područja kontrola na web sučelju aplikacije što se postiže modularnim dizajnom.

Upravo to sve navedeno rješava posebna arhitektura web sučelja s klijentske strane, a naziva se arhitektura web aplikacija s prikazom svih komponenti na jednoj stranici, skraćeno SPA (eng. *Single Page Application*).

Arhitektura web aplikacija s jedinstvenom stranicom nalaže da se osnovna stranica posluži jednom, a zatim, ovisno o interakciji korisnika i zahtjevima upućenim prema serveru, poslužuju ostali dijelovi stranice dinamički i asinkrono pomoću AJAX zahtjeva i programskog jezika Javascript.

Upravo zato, Google je razvio vlastitu biblioteku koda, tj. programski koncept rješenja SPA aplikacije temeljen na jeziku Javascript kojeg su nazvali AngularJS (koncept rješenja nalazi se na Slici 14). Dakle, AngularJS je nastao kao Googleov odgovor na pitanja SPA aplikacija i omogućuje izradu takvih web aplikacija uz pomoć znanja programskog jezika Javascript. Podržava poznate koncepte jezika Javascript poput funkcija obećanja (eng. *promises*) i upravljanja DOM strukturom web stranice.



Slika 14 – programski koncept rješenja AngularJS (eng. *framework*)

No, ono čime doprinosi taj programski koncept rješenja je modularni dizajn dinamičke stranice na konzistentan način upotrebom takozvanih direktiva (eng. *directives*), ali i proširivanjem HTML jezika vlastitim oznakama i elementima, što prije AngularJS nije bilo ostvareno na takav način. Mogućnost da se izrade vlastite oznake ili elementi i da se potom prikažu na stranici olakšava logičku podjelu dijelova web sučelja aplikacije.

Primjerice, izrađena je komponentu sučelja koja komunicira pomoću AJAX poziva sa serverskom stranom šaljući zahtjev za očitanje RFID taga pomoću modula MFRC-522, dok je u isto vrijeme korisniku dostupan

ostatak stranice s kojim može slobodno vršiti interakciju dok čitač ne pošalje rezultat očitanja, čime bi komponenta sučelja promijenila sadržaj i prikazala to očitanje.

Štoviše, AngularJS je programski koncept rješenja otvorenog tipa pa je zato postao široko prihvaćen u zajednici tzv. „front-end“ programera, odnosno programera koji se bave razvojem sučelja web aplikacija na klijentskoj strani, a podržavaju ga individualci i korporacije diljem svijeta.

Zbog uvođenja tehnologije AngularJS, logika web aplikacije podijeljena je na 2 pod aplikacije, serversku ili REST API logiku koja se bavi posluživanjem resursa na AJAX zahtjeve i klijentsku ili AngularJS logiku, koja dinamički poslužuje i upravlja sadržajem na sučelju web aplikacije u pregledniku korisnika.

Klijentska ili AngularJS logika predstavlja zasebnu aplikacijsku logiku cjelokupnog programskog rješenja, jer je cilj sustava dati korisniku podatke u realnome vremenu pa je veći naglasak na prikazu podataka sa klijentske strane i u interakciji na tom dijelu, nego li je u samoj obradi na serverskoj strani (obrađuje se samo nekoliko vrsta podataka što nije suviše zahtjevno). Kako bi se ostvarila ta interakcija na klijentskoj strani i bez upita za prikaz cijele stranice sa servera, upotrebljena je dodatna logika na klijentskoj strani putem koncepta u jeziku Javascript, tj. rješenja AngularJS.

AngularJS logika ponaša se kao aplikacija za sebe, prema konvenciji u formiranju arhitekture aplikacije naziva Model-Prikaz-Kontroler ili MVC (eng. Model View Controller). Prema navedenoj konvenciji, miješanje logike obrade i transporta podataka sa mehanizmom prikaza podataka ili samom pohranom podataka u perzistentnoj pohrani krši pravila kvalitetno organiziranog dizajna i otežava razvoj web aplikacija, stoga je takva arhitektura aplikacije nepoželjna. Konvencija Model-Prikaz-Kontroler ili MVC nudi podjelu arhitekture aplikacije, tj. njenih funkcionalnih jedinica prema navedenim područjima kategorički.

Tako sve funkcionalnosti koje se bave dohvatom ili pohranom u perzistentnu pohranu podataka ili predstavljaju reprezentaciju podataka koji se pohranjuju u tu pohranu se nazivaju modelima. Postoje i posebni modeli prikaza ili MV (eng. *Model-View*) koji se služe samo za prikaz podataka direktno iz pohrane na konkretnu stranicu.

Prikaz podataka na stranici, dakle dizajn i razmještaj određuju funkcionalnosti prikaza, to su kod AngularJS direktive i filteri za filtriranje podataka već prikazanih na stranici.

Konačno, logiku za kolanje i transport podataka među određenim dijelovima na stranici ili prema i od servera ka klijentu obuhvaćaju funkcionalni dijelovi koje nazivamo kontrolerima, koji mogu dio posla

dodjeliti podređenim cjelinama naziva tvornice (eng. *factory*), servisi (eng. *service*), opskrbljivači (eng. *providers*) i konstante (eng. *values*).

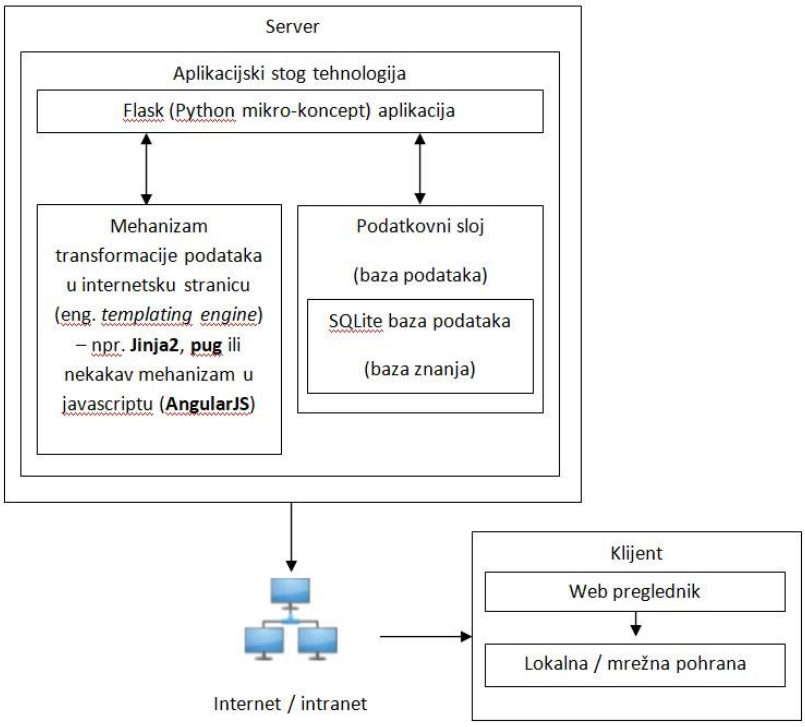
Cjelokupni arhitekturni obrazac programiranja spada u objektno orijentirane konvencije dizajna aplikacije prema objektno orijentiranoj paradigmi. Upravo na taj način doprinosi AngularJS, jer omogućuje takvu podjelu aplikacije prema funkcionalnim dijelovima.

U konačnici, takva logika aplikacije na klijentskoj strani pruža vrlo interaktivno sučelje i minimizira potrebu za dohvat velikog broja podataka sa servera, već samo prema interakciji sa korisnikom, dakle u realnome vremenu.

5.6.4.3. Flask (Python)

Kako bi se izveo REST API server u programskom jeziku Python, a opet imati konzistentan način formiranja koda i kontrolu nad njime, potrebno je upotrijebiti neki već postojeći programski koncept rješenja (eng. *framework*), koji je moguće jednostavno prilagoditi potrebama projekta.

Jedan takav programski mikro-koncept rješenja (eng. micro-framework) je Flask, napravljen u programskom jeziku Python (koncept dizajna aplikacije temeljen na rješenju Flask prikazan je na Slici 15). Flask je prilagodljivi (eng. *customizable*) koncept, tj. mikro-koncept rješenja, stoga je idealan za ovaj projekt. Štoviše nudi samo ono osnovno da bi se izradio web server, tj. web aplikacija, a opet nije posve ostavljeno sve na leđa programera kao što bi bilo da se server radi u samom programskom jeziku Python.



Slika 15 – programski mikro-koncept rješenja Flask (eng. *micro-framework*)

Koncept dizajna uobičajene aplikacije izrađene na temelju koncepta rješenja koje je Flask (Slika 15), sastoji se od osnovnog seta funkcionalnosti koje se zasnivaju na WSGI (eng. *Web Server Gateway Interface*) tehnologiji tipičnoj kod web aplikacija nastalih u programskom jeziku Python, zatim dio aplikacije koji služi prikazu web stranica korisniku (mehanizam transformacije podataka u obliku web stranica, obično je to Jinja2, ali može se primijeniti i nekakav mehanizam nastao na skriptnom jeziku Javascript), te konačno od samog podatkovnog sloja ili baze podataka, npr. baze SQLite za ugrađene sustave ili kod uređaja za Internet stvari, s ograničenom radnom memorijom.

Navedena tri sloja primjenjena su i u projektu diplomskog rada, u sljedećem poretku: osnovne funkcionalnosti aplikacijskog servera realizirane pomoću rješenja Flask, mehanizam transformacije podataka i prikaza pomoću internetskih stranica sa rješenjem AngularJS i konačno perzistentnost podataka realizirana sa bazom podataka SQLite. Ovakav troslojni aplikacijski stog tehnologija omogućuje izvedbu jednostavnog servera, bez velikih zahtjeva za uređaj na kojem se server emulira, tj. izvodi.

Naime, programski jezik Python sam po sebi može se upotrijebiti za emuliranje servera, jer sadrži gotovi ugrađeni server koji se može pokrenuti naredbom ***„python -m SimpleHTTPServer 8000“*** u verziji 2 programskog jezika Python, a taj modul servera premješten je u „http.server“ modul pa je naredba

„python –m http.server 8000“ u verziji 3. No, kako bi se napravio pravi REST API server, potrebno je taj osnovni server izmijeniti što nije jednostavno, te omogućiti preostala 2 sloja kako bi bio potpun.

Zato i zbog razloga što se aplikacijski server treba koristiti na jednostavnom uređaju u primjeni kod Interneta stvari, odluka je pala na primjenu nečeg bližeg namjeni, a to je programski mikro-koncept rješenja web aplikacije, naziva Flask. Flask kao i Python, podržava module, ima razne dodatke i općenito može primijeniti bilo koju ekstenziju namjenjenu za Python, zbog čega je vrlo fleksibilan.

6. Ontologije

Ontologija je oblik predstavljanja znanja o stvarnom svijetu. Prema definiciji T. Gubera, ontologija je eksplicitna specifikacija konceptualizacije (Gómez-Pérez et al., 2004). Iz perspektive računarskih znanosti, ontologija se odnosi na skup osnovnih reprezentacijskih funkcija pomoću kojih se može oblikovati model određene domene znanja.

Kako u računalnim znanostima tako i u informacijskim znanostima ontologija je obrazac podatka koji predstavlja koncepte unutar neke domene i odnose između tih koncepata. Koristi se za razumijevanje objekata unutar te domene. Ontologije općenito opisuju individue, klase ili koncepte, atribute i odnose.

Individue su osnovna, početna razina sastavnica ontologije. Individue u ontologiji mogu uključiti konkretne objekte poput ljudi, životinja, molekula i planeta. Jednako kao i apstraktne individue poput brojeva i riječi. Strogo rečeno, ontologija ne mora uključiti individuu, no jedna od općih svrha ontologije jest omogućavanje sredstava za klasifikaciju individua pa i ako te individue nisu dio same ontologije.

Klase su apstraktne grupe, skupine ili zbirke objekata. Mogu sadržavati individue, druge klase ili kombinaciju jednih i drugih (Osoba – klasa svih ljudi, Molekula – klasa svih molekula) Također, konceptualizacije iz izvorne Gruberove definicije ontologije su apstraktni modeli koji uključuju domenu ontologije, identificirane temeljne koncepte te domene i relacije koncepata.

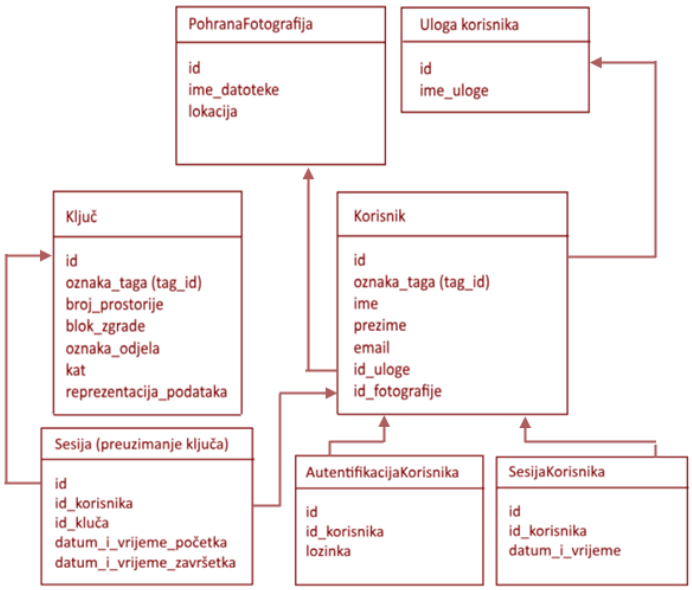
Objekti u ontologiji mogu biti opisani tako da im se pridruže atributi. Svaki atribut ima bar ime i vrijednost, a koristi se za pohranu informacija specifičnih za objekt na koji se odnosi. Primjerice, Fiat Stilo ima atribute kao: Ime: Fiat Stilo; Broj vrata: 5; Motor: 1,2 l; Prijenos: 6 brzina.

Vrijednost atributa naravno može biti kompleksni oblik podatka, u ovom primjeru vrijednost atributa nazvanog motor može biti lista, a ne samo jedna vrijednost. Potrebno je definirati atribute koncepata, ontologija definira domenu.

Važan način uporabe atributa je opis odnosa, (relacija) između objekata u ontologiji. Uobičajeno, relacija je atribut čija je vrijednost drugi atribut u ontologiji. Primjerice, imamo noviji i stariji model automobila, stariji model može biti prikazan kao sljedeći atribut: nasljednik, ime prijašnjeg modela. Skup relacija opisuje semantiku domene.

Domena ontologije oblikuje specifičnu domenu ili dio svijeta. Ona predstavlja pojedinačno značenje izraza kako se primjenjuju na tu domenu. Primjerice engleska riječ „card“ ima više različitih značenja. Ontologija o domeni pokera bi oblikovala značenje te riječi kao igraću kartu, dok bi ontologija o računalnoj opremi oblikovala značenje te riječi kao grafičku karticu.

Ontologije formirane u domeni projekta diplomskog rada prikazane su na Dijagramu klasa 5.



Dijagram 5 – dijagram klasa

Važno je da je domena ontologije eksplicitno zadana, što znači da su definirana značenja za sve koncepte. Također, ontologije moraju biti formalno zapisane, tako da ih računalo može interpretirati (osim toga da su binarne pa se računalo može njima koristiti, puno je važnije da ih može sistematično svrstavati, prikazivati i na temelju njih formirati nove koncepte).

Te ontologije moraju biti zajedničke među agentima, dakle moraju biti dogovorene. (K. Jurčić et alt, 2013.) Zbog navedenih razloga u nastavku slijedi opis i značenje određenih ontologije koje su upotrijebljene u projektu diplomskog rada, a odredile su model znanja sustava unutar projekta, kao što je to prikazano na Dijagramu 5.

Dakle, kao što je vidljivo na Dijagramu 5, model znanja je usmjeren na korisnika (eng *User*), jer najviše veza iz drugih tablica upućuju na tablicu korisnika. Podaci korisnika su njegov jedinstveni identifikacijski broj ili **id**, serijski broj RFID taga ili **tag_id**, zatim njegovo ime, prezime i email, te titula ili uloge i eventualno fotografija. Ime, prezime i email (**first_name, last_name, email**) su polja uključena u tablicu korisnika, no titula i fotografija zahtijevaju odvojenu tablicu.

Razlog za titulu je, jer je potrebno sistematizirati titule, definirati nekakve globalne titule koje korisnik kasnije odabire pri registraciji. Titule (**role**) su: profesor i student. Titula također određuje mogučnosti, tj. prava upravljanja resursima unutar aplikacije, dakle profesor ima sva prava, student samo prava pregledavanja – ne smije uređivati tuđe podatke ni brisati.

Odvajanje fotografija u odvojenu tablicu omogućuje pohranjivanje lokacije na datotečnom sistemu servera, tj. uređaja Raspberry Pi. Dakle, fotografije se izravno pohranjuju na datotečni sitem i pamte im se lokacije, za razliku od klasičnih metoda pohrane slika unutar baze podataka, transformirane kao tzv. blob.

S obzirom da se procjenjuje manji broj korisnika aplikacije, dakle lokalizirana je primjena baze na intranet, ne postoji izrazita potreba za upotrebom vanjskog servisa za čuvanje i hashiranje (prikrivanje) lozinki korisnika pa se lozinke povezuju sa korisnicima i pohranjuju direktno u bazu podataka u zasebnu tablicu prijava (eng. *AuthUser*).

Nadalje, tehnologija baze podataka koja se primjenjuje naziva se SQLite. SQLite je vrsta baze podataka koja se često primjenjuje u mobilnim uređajima, tj. pametnim telefonima, ali i ostalim tzv. ugrađenim sustavima (eng. *Embedded Systems*) gdje su radna i trajna memorija oskudni resursi pa je potrebno memorijski lakše, brže rješenje (eng. *light*, odatle i naziv).

Podaci o ključevima pohranjuju se u zasebnu tablicu (eng. *Key*), a ključni zbog načina na koji se spremaju. Naime, podaci o ključu predstavljaju model znanja za sebe, jer su u osnovi dekompozicija podataka koji čine oznaku prostorije. Naime, svaka institucija ima svoj način vođenja i označavanja prostorija, nekima je ključ na broj katova, drugima je ključan predio zgrade, trećima namjena ili odjel.

Na Prirodoslovno-matematičkom fakultetu u Splitu, za koje je prvenstveno osmišljen i izrađen sustav za evidenciju i upravljanje ključeva, način označavanje prostorija je sljedeći, oznaku čine podaci slijedom: **[predio zgrade (jedno slovo)] [prvo slovo odjela] [broj kata] – [broj sobe]**, npr. BI2-73 što znači predio zgrade (**block_name**) B, odjel (**sector_name**) za Informatiku, prostorija se nalazi na 2. katu (**floor**), a broj prostorije (**room_id**) je 73.

Nadalje, sesija (eng. *Session*) je trag preuzimanja i vraćanja ključa, odnosno polaganje prava na resurs ključa i prostorije. Podaci koji se prikupljaju, već su spomenuti ranije u tekstu, kod modela znanja, dakle to su identifikacijski broj ili broj retka ključa u tablici (**key_id**), te slično broj retka korisnika u tablici (**user_id**), datum i vrijeme preuzimanja (**started_on**) ključa, datum i vrijeme vraćanja (**closed_on**) ključa i zatvaranja sesije.

Također, sustav ima predviđeno praćenje zasebnih sesija korisnika, za potrebe naprednije integracije sustava sa tzv. pametnom bravom. Dakle, u slučaju da nivo rizika od povrede pravila pristupa prostorijama zahtjeva viši nivo sigurnosti, sustav ima mogućnost nadogradnje povezivanjem elektronske brave sa računalom Raspberry Pi i modulom RFID čitača, za automatsko otvaranje i zatvaranje prostorije isključivo putem RFID kartice korisnika.

Razlog zašto to nije primaran sustav, odnosno mehanizam rješenja je što u slučaju prekida struje ili nekakve mehaničke blokade, korisnik s trenutnim izvedbama elektronske brave nije u mogućnosti koristiti bravu, time ni otključati ili zaključati ju.

Pored toga, za tako limitirajuće rješenje, cijena za integraciju, tj. ugradnju je preskupa u omjeru sa vrijednosti koju doprinosi, tj. prednostima takvog sustava. Za kvalitetniju i efikasniju izvedbu takvog sustava, koja je moguća, potrebno je malo više istraživanja i razvoja ili R. & D. (eng. Research and Development).

Također, prema G. Schreiber, 2000. postoje zasebna istraživanja o ontologiji koja imaju dvostruki cilj:

1. Iskoristiti postojeće ontologije u drugim aplikacijskim domenama i zadacima. Ako je to moguće, dostupnost takvih postojećih ontologija može ubrzati postupak razvoja sustava temeljenog na znanju u velikoj mjeri.
2. Baze znanja obično predstavljaju veliki trošak resursa, a u praksi se vrlo teško ponovno iskorištavaju. Jedna svrha ontologija bi mogla biti iskoristiti postojeće semantičke opise kao

temelj za izradu novih baza znanja i time omogućiti bolje razumijevanje pojedinih dijelova u bazi znanja.

Dakle, moglo bi se osmisliti tehnološki naprednije rješenje koje je komplementarno tradicionalnoj bravi, no omogućuje logistiku i autentifikaciju korisnika prisustvom napajanja, no omogućuje i analogni način otključavanja i zaključavanja.

Štoviše, postoje primjeri postojećih rješenja koja pružaju komunikaciju putem bluetooth (bežična tehnologija dometa do 100m) tehnologije sa pametnim telefonom putem kojeg se korisnik autentificira. Dakle, ovakav jedan sustav predstavlja skoru budućnost upravljanja prostorijama i resursima.

7. Pregled standarda

Standarde određuju organizacije za kreiranje standarda, poput Internacionalne organizacije za standardizaciju ili ISO (eng. *International Organization for Standardization*) i Internacionalne komisije za elektrotehniku ili IEC (eng. *International Electrotechnical Commission*).

S obzirom da se RFID tehnologija, primijenjena u projektu diplomskog rada, općenito upotrebljava i prilikom plaćanja, nalazi se prikačena na odjeću u transportu i trgovinama za logistiku i praćenje, kao i u ostalim granama trgovine, te za praćenje životinja i ljudi, to podiže razna sigurnosna pitanja.

Stoga je takva tehnologija regulirana standardom ISO/IEC 18000 i ISO/IEC 20248 što obuhvaća upotrebu čipova za kriptografiju, autentifikaciju tagova i čitača, privatnost podataka koji putuju zrakom kao medijem. Nadalje, standard ISO/IEC 20248 specificira strukturu digitalnog potpisa podataka za tehnologiju RFID i barkodova. Ovakav uspjeh posljedica je primjene ISO/IEC JTC 1/SC 31 metode za automatsku identifikaciju i dohvaćanje podataka raznim tehnikama. Prema ISO/IEC 18000 dopuštene frekvencije su: ispod 135 kHz, 13,56 MHZ, 2,45Ghz, 5,8 GHz, 860-960 MHz i 433 MHz (aktivno u upotrebi danas). Ovoliki odabir frekvencija postoji zbog podrške za starije tehnologije (eng. *legacy*).

Tagovi frekvencijskog područja 135 KHz dijele se na 2 tipa: tip A s dvosmjernim prijenosom podataka ili FDX (eng. *Full Duplex*) i tip B s jednosmjernim prijenosom podataka ili HDX (eng. *Half Duplex*).

Tagovi frekvencijskog područja 13,56 MHz pružaju 2 načina rada: prvi za praćenje robe s brzinama čitanja od 1,65 kbps do 26,48 kbps, te drugi način za veće brzine i memorije PJM (eng. *Phase-Jitter Modulation*), gdje su brzine do 423,75 kbps.

Tagovi frekvencijskog područja 2,45 GHz imaju također 2 načina rada, no razlikuju se u dometu signala, prvi način pruža domet do 3 stope ili približno 1 metar, te drugi način rada koji pruža domet i do 300 stopa ili približno 100 metara, s brzinom čitanja i pisanja do 384 kbps. Pored navedenog, pružaju mnoge druge sofisticiranije postavke i mogućnosti, poput promjene širine pojasa komunikacijskog kanala, ograničavanje modulacije radio valova, nove metode kodiranje podataka i sl.

Tagovi frekvencijskog pojasa od 860 do 960 MHz imaju 2 tipa, tip A, tip B i tip C. Razlikuju se međusobno u metodama hashiranja vrijednosti ključeva, tj. serijskih brojeva, dakle prema metodama anti-kolizije.

Hashiranje je općenito postupak upotrebe takozvane hash funkcije za učinkovito preslikavanje određenih ključeva (na primjer imena ljudi) u njima pridružene vrijednosti (na primjer telefonske brojeve). (D. Knuth, 2011.)

Čitači pojasa visokih frekvencija ili UHF čitači (eng. *Ultra High Frequency*) rade u pojasu frekvencija od 300 MHZ do 3 GHz, stoga se primjenjuju u svim kategorijama osim prve dvije. Ograničenja frekvencija postavila je Federalna komisija za komunikaciju ili FCC (eng. *Federal Communications Commision*) iz Sjedinjenih Američkih Država.

Usprkos svim standardima i visokim tijelima koja nadgledaju razvoj i distribuciju RFID tehnologije, postoje ozbiljni sigurnosni problemi i propusti zbog čega je tehnologija, posebice od izuma UHF čitača, vrlo izložena riziku krađe informacija.

Postoji nekoliko metoda proboja sigurnosti RFID tehnologije, od direktnog napada na čip uređaja, preko kloniranja kartica s tagovima do ručnog unosa u registar čitača dajući neispravno stanje da je korisnik identificiran i autentificiran, a i prosječne metode izvedive su unutar 100 milisekunda.

Preduvjet za izvedbu metoda je naravno visok stupanj znanja, iskustva i informiranosti o području RFID tehnologija i sigurnosti općenito, te posjedovanje alata, programa i algoritama za izvođenje proboja. Stoga je prag za izvedbu proboja u praksi visoko postavljen, no nije nedostižan.

Problem sigurnosti upotrebe RFID tehnologije unutar projekta diplomskog rada i za primjenu sustava u praksi nije od velikog značaja, jer se ne primjenjuje za čuvanje resursa koji su stroga tajna već za

praćenje upotrebe i preuzimanja ključeva, a subjekt primjene su obrazovne institucije, preciznije upotrebljavaju se unutar prostora zgrada od institucija kojima je pristup dozvoljen samo studentima, učenicima, nastavnicima i ostalom radnom osoblju.

Od većeg je značaja sigurnost web aplikacije sustava, jer su u pitanju izravno podaci pojedinih korisnika koji mogu biti kompromitirani, ako nisu adekvatno zaštićeni, za razliku od RFID tagova koji čuvaju podatke o serijskom broju ključeva, koji bez znanja sustava ne predstavljaju gotovo nikakve ključne podatke.

8. Zaključak

Okosnicu projekta čini metodologija za vođenje i organizaciju ljudi i resursa iz područja menadžmenta koji pokriva projekte visoke sistematizacije podataka i koja se prvenstveno provodi za projektiranje sustava temeljenih na znanju. Ta metodologija, naziva CommonKADS, primjenjuje se na sustavima temeljenima na znanju, među koje se ubraja i sam projekt diplomskog rada.

Kao što je kroz rad opisano, pojašnjeno i ilustrirano, CommonKADS metodologija pruža skup generičkih modela vrlo zgodnih za funkcionalnu dekompoziciju organizacije i vođenja projekta, znanja, sposobnosti, zadataka, komunikacije i dizajna sustava temeljenog na znanju, odnosno projekta čiji je on produkt.

Sam projekt diplomskog rada predstavlja projekt u domeni menadžmenta, inženjeringa znanja, programiranja, elektronike i web dizajna i interneta stvari. Ova svestranost različitih područja i potrebnih sposobnosti za ta konkretna područja zahtjeva nekakvu vođenu strukturu i mudriji način organizacije i interakcije tih različitih područja i agenata koji djeluju unutar tih područja.

Stoga, je potrebna nekakva metodologija, konvencija ili dogovor koji osigurava kvalitetan rad i nadgledanje razvoja sustava temeljenih na znanju. Kao takva, metodologija CommonKADS, pokazala se isplativom i praktičnom u domeni razvoja sustava temeljenih na znanju.

Tehničke i tehnološke komponente sustava, standardi i paradigme su podložne promjeni vremena, no problemi organizacije poslovnih procesa, rad i usklađivanje ljudi i resursa, agenata, ostaje ključno pitanje menadžmenta, ali i ostalih područja ljudske aktivnosti, na koje možda upravo CommonKADS ima odgovor.

9. Literatura

[1] Schreiber, G. (2000). *Knowledge engineering and management: the CommonKADS methodology*. MIT press.

[2] Kingston, J., Shadbolt, N., & Tate, A. (1996, August). CommonKADS models for knowledge based planning. In *AAAI/IAAI, Vol. 1* (pp. 477-482), raspoloživo na poveznici:

http://www.aiai.ed.ac.uk/publications/documents/1996/96-aaai-commonkads-planning.pdf

[3] Fensel, D., & Krummenacher, R. (2010). Intelligent Systems., raspoloživo na poveznici:

http://liris.cnrs.fr/amille/enseignements/MasterCode/IC_IA/2012-2013/06_Intelligent_Systems-CommonKADS.pdf

[4] Waldner, J. B. (1992). *CIM: principles of computer-integrated manufacturing*. John Wiley & Sons.

[5] Feigenbaum, E., & McCorduck, P. (1983). The Fifth Generation. New York, NY: New American Library with Addison. *Wesley Publishing Company, Inc*, *238*, 40-81.

[6] Ž. Panian, 2005., Informatički enciklopedijski rječnik - Englesko-Hrvatski, Jutarnji List

[7] Gruber, T. R. (1993). A translation approach to portable ontology specifications. *Knowledge acquisition*, *5*(2), 199-220, raspoloživo na poveznici:

http://tomgruber.org/writing/ontolingua-kaj-1993.pdf

[8] Juričić, K., & Meštrović, A. (2013). Overview of ontology alignment techniques and methods. *Zbornik Veleučilišta u Rijeci*, *1*(1), 75-93.

[9] Corcho, O., Fernández-López, M., & Gómez-Pérez, A. (2006). Ontological engineering: principles, methods, tools and languages. In *Ontologies for software engineering and software technology* (pp. 1-48). Springer Berlin Heidelberg.

[10] Knuth, D. (2011). The Art of Programming. *ITNow*, *53*(4).

10. Prilozi

10.1. Kazalo slika

10.2. Kazalo tablica

10.3. Kazalo shematskih prikaza

10.4. Kazalo dijagrama